

Zbigniew Michalewicz
David B. Fogel

jak to rozwiązać czyli nowoczesna heurystyka



Wydawnictwa Naukowo-Techniczne

Z angielskiego przełożyli:

Aleksy Schubert

Joanna Schubert

O Autorach:

Zbigniew Michalewicz jest absolwentem Politechniki Warszawskiej. Doktoryzował się w 1981 roku (PAN), habilitował w 1997 roku (PAN), a tytuł profesora otrzymał w 2002 roku. Bez wątpienia jest obywatelem świata. W latach 1982-1989 był wykładowcą w Victoria University w Nowej Zelandii, a potem, w latach 1988-2004, w University of North Carolina at Charlotte. Obecnie jest związany ze School of Computer Science przy University of Adelaide i ze State Key Laboratory of Software Engineering przy Wuhan University w Chinach. Współpracuje też z Instytutem Informatyki PAN i Polsko-Japońską Wyższą Szkołą Technik Komputerowych.

W swojej pracy badawczej zajmuje się obliczeniami ewolucyjnymi i innymi metodami heurystycznymi. Jest autorem znanej już w Polsce książki „Algorytmy genetyczne + struktury danych = programy ewolucyjne”, której trzy wydania ukazały się nakładem Wydawnictw Naukowo-Technicznych. W dorobku naukowym ma też ponad 200 artykułów w prasie fachowej i materiałach konferencyjnych.

Był jednym z redaktorów naukowych pracy zbiorowej pt. „Handbook of Evolutionary Computation” i przewodniczącym pierwszej zorganizowanej przez IEEE międzynarodowej konferencji poświęconej obliczeniom ewolucyjnym, która odbyła się w Orlando w 1994 roku. Obecnie jest członkiem komitetów redakcyjnych lub zastępcą redaktorów naczelnych 11 czasopism o międzynarodowym zasięgu. Jest też prezesem zarządu firmy Solvelt Software, którą założył wspólnie z kilkoma znanimi na całym świecie informatykami, specjalistami w dziedzinie sztucznej inteligencji.

David B. Fogel ma stopień doktora nauk inżynierjnych, który uzyskał w 1992 roku (University of California at San Diego). Ma ponad 19-letnie doświadczenie w stosowaniu metod sztucznej inteligencji do rozwiązywania problemów związanych z przemysłem, medycyną i obronnością. Jest założycielem i dyrektorem naczelnym firmy Natural Selection, Inc., działającej od 1993 roku. Wcześniej, w latach 1984-1988 był analitykiem systemowym w Titan Corporation, a w latach 1988-1993 głównym inżynierem w OROCON Corporation. Zajmował się też pracą dydaktyczną. Prowadził zajęcia na uczelni z obliczeń ewolucyjnych, procesów stochastycznych i statystycznego sterowania procesami.

W swoim dorobku ma kilka książek, m.in. „Blondie 24: Playing at the Edge of AI” i „Evolutionary Computation: Toward a New Philosophy of Machine Intelligence”, a także ponad 200 artykułów w fachowych czasopismach. W latach 1996-2002 był redaktorem naczelnym pisma *IEEE Transactions on Evolutionary Computation*. Obecnie jest redaktorem serii wydawanej przez IEEE i Wileya, poświęconej inteligencji komputerowej.

Jest członkiem IEEE i należy do komitetu redakcyjnego sprawozdań IEEE. Nadzoruje też prace nad publikacjami IEEE Neural Networks Society.

Dane o oryginale

PROF. ZBIGNIEW MICHALEWICZ

School of Computer Science, University of Adelaide

Adelaide, South Australia 5005

zbyszek@cs.adelaide.edu.au

DR DAVID B. FOGEL

Natural Selection, Inc., Suite 200

3333 N. Torrey Pines Ct.

La Jolla, CA 92037, USA

dfogel@natural-selection.com

How to Solve It: Modern Heuristics

Second, Revised and Extended Edition

Translation from the English language edition:

How to Solve It: Modern Heuristics by Zbigniew Michalewicz and David B. Fogel

Copyright © Springer-Verlag Berlin Heidelberg 2000, 2004

All Rights Reserved

Redaktor *Zofia Dackiewicz*

Okładkę i strony tytułów projektował *Paweł G. Rubaszewski*

Na okładce reprodukcja obrazu namalowanego przez *Ewę J. Michalewicz*

Redaktor techniczny *Ewa Kosińska*

Korekta *Zespół*

Skład i łamanie *Ewa Koistar*

Wymienione w niniejszej książce produkty są znakami towarowymi odpowiednich firm.
Nazwy te są pisane w tekście wielkimi literami lub zaczynają się wielką literą.

Podręcznik akademicki dotowany przez Ministra Edukacji i Nauki

© Copyright for the Polish edition by Wydawnictwa Naukowo-Techniczne
Warszawa 2006

All Rights Reserved

Printed in Poland

Utwór w całości ani we fragmentach nie może być powielany ani rozpowszechniany za pomocą urządzeń elektronicznych, mechanicznych, kopujących, nagrywających i innych, w tym również nie może być umieszczany ani rozpowszechniany w postaci cyfrowej zarówno w Internecie, jak i w sieciach lokalnych bez pisemnej zgody posiadacza praw autorskich.

Wydawnictwa Naukowo-Techniczne

00-048 Warszawa, ul. Mazowiecka 2/4

tel. (0-22) 826 72 71, e-mail: wnt@wnt.pl

www.wnt.pl

ISBN 83-204-3108-5

*Dla kochanej Ewy – najmilszej „zagadki”,
jaką rozwiążuję od 30 lat!*

Z.M.

*Dla moich nauczycieli matematyki
ze szkoły średniej: pana Flattuma,
pana McPhee, pana Jacksona
i pana Copperthwaite'a*

D.B.F.

Spis treści

Przedmowa do drugiego wydania	17
Przedmowa do pierwszego wydania	19
Wstęp	23
I. Ile lat mają moi trzej synowie?	31
1. Dlaczego niektóre problemy są trudne do rozwiązania?	35
1.1. Wielkość przestrzeni przeszukiwania	36
1.2. Modelowanie problemu	40
1.3. Zmiana związana z czasem	44
1.4. Ograniczenia	46
1.5. Problemy z udowadnianiem	48
1.6. Twoja szansa na sławę	51
1.7. Podsumowanie	54
II. Jak ważny jest model?	57
2. Podstawowe pojęcia	61
2.1. Reprezentacja	61
2.2. Cel	62
2.3. Funkcja oceny	63
2.4. Określanie problemu poszukiwania	65
2.5. Otoczenia i lokalne optima	66

2.6.	Metody wspinania się	69
2.7.	Czy umiesz tak sprytnie uderzyć bilę?	72
2.8.	Podsumowanie	74
III.	Jakie są ceny w sklepach „7–11”?	75
3.	Metody tradycyjne – część I	81
3.1.	Przeszukiwanie wyczerpujące	85
3.1.1.	Metody wyliczeniowe dla SAT	85
3.1.2.	Metody wyliczeniowe dla TSP	87
3.1.3.	Metody wyliczeniowe dla NLP	89
3.2.	Przeszukiwanie lokalne	91
3.2.1.	Przeszukiwanie lokalne i SAT	92
3.2.2.	Przeszukiwanie lokalne i TSP	93
3.2.3.	Przeszukiwanie lokalne i NLP	96
3.3.	Programowanie liniowe: metoda sympleks	105
3.4.	Podsumowanie	109
IV.	Jakie to liczby?	111
4.	Metody tradycyjne – część II	115
4.1.	Algorytmy zachłanne	115
4.1.1.	Algorytmy zachłanne i SAT	115
4.1.2.	Algorytmy zachłanne i TSP	117
4.1.3.	Algorytmy zachłanne i NLP	118
4.2.	Dziel i rządź	119
4.3.	Programowanie dynamiczne	121
4.4.	Metoda podziału i ograniczeń	131
4.5.	Algorytm A^*	135
4.6.	Podsumowanie	138
V.	Jakiego koloru jest niedźwiedź?	141
5.	Unikanie lokalnych optimów	145
5.1.	Symulowane wyżarzanie	147
5.2.	Poszukiwanie z tabu	155
5.3.	Podsumowanie	164

VI.	Jak dobrą masz intuicję?	167
6.	Podejście ewolucyjne	171
6.1.	Podejście ewolucyjne do SAT	175
6.2.	Podejście ewolucyjne do TSP	178
6.3.	Podejście ewolucyjne do NLP	181
6.4.	Podsumowanie	184
VII.	Jedna z tych rzeczy jest niepodobna do pozostałych	191
7.	Projektowanie algorytmów ewolucyjnych	195
7.1.	Reprezentacja	200
7.1.1.	Wektory symboli o ustalonej długości	201
7.1.2.	Permutacje	202
7.1.3.	Automaty ze skończoną liczbą stanów	203
7.1.4.	Wyrażenia symboliczne	203
7.2.	Funkcja oceny	204
7.3.	Operatory różnicowania	208
7.3.1.	Wektory symboli o ustalonej długości	208
7.3.2.	Permutacje	209
7.3.3.	Automaty ze skończoną liczbą stanów	211
7.3.4.	Wyrażenia symboliczne	212
7.4.	Selekcja	214
7.5.	Inicjowanie	217
7.6.	Podsumowanie	218
VIII.	Jaka jest najkrótsza droga?	221
8.	Problem komiwojażera	225
8.1.	W poszukiwaniu dobrych operatorów różnicowania	228
8.2.	Uzupełnianie o metody poszukiwania lokalnego	251
8.3.	Inne możliwości	254
8.3.1.	Krzyżowanie ze składaniem krawędzi	254
8.3.2.	Operator inver-over	257
8.4.	Podsumowanie	260
IX.	Kto ma zebrę?	263
9.	Metody radzenia sobie z ograniczeniami	269
9.1.	Ogólne rozważania	270
9.1.1.	Określanie funkcji $eval_f$	273
9.1.2.	Określanie funkcji $eval_u$	275

9.1.3.	Zależności między $eval_f$ a $eval_u$	276
9.1.4.	Odrzucanie rozwiązań niedopuszczalnych	277
9.1.5.	Poprawianie osobników niedopuszczalnych	278
9.1.6.	Zastępowanie osobników ich poprawionymi wersjami .	278
9.1.7.	Nakładanie kar na osobniki niedopuszczalne	279
9.1.8.	Utrzymywanie populacji dopuszczalnej za pomocą specjalnych sposobów reprezentacji i operatorów różnicowania	280
9.1.9.	Stosowanie dekoderów	281
9.1.10.	Oddzielanie osobników i ograniczeń	282
9.1.11.	Badanie granicy między dopuszczalną a niedopuszczalną częścią przestrzeni przeszukiwania .	282
9.1.12.	Znajdowanie rozwiązań dopuszczalnych	284
9.2.	Optymalizacja numeryczna	285
9.2.1.	Metody oparte na zachowywaniu dopuszczalności rozwiązań	285
9.2.2.	Metody oparte na funkcjach kary	289
9.2.3.	Metody oparte na poszukiwaniu rozwiązań dopuszczalnych	297
9.2.4.	Metody oparte na dekoderach	305
9.2.5.	Metody hybrydowe	308
9.3.	Podsumowanie	310
X.	Czy potrafisz dostosować się do problemu?	313
10.	Dostosowywanie algorytmu do problemu	319
10.1.	Parametry sterujące w algorytmach ewolucyjnych	320
10.2.	Objaśnienie zagadnienia za pomocą NLP	324
10.3.	Taksonomia metod sterowania	327
10.4.	Möżliwości sterowania parametrami	331
10.4.1.	Reprezentacja	331
10.4.2.	Funkcja oceny	333
10.4.3.	Operatory mutacji i ich prawdopodobieństwa	334
10.4.4.	Operatory krzyżowania i ich prawdopodobieństwa	336
10.4.5.	Selekcja rodziców	338
10.4.6.	Populacja	339
10.5.	Łączenie sposobów sterowania parametrami	340
10.6.	Podsumowanie	342

XI. Czy potrafisz dać mata w dwóch ruchach?	347
11. Środowiska zmienne w czasie oraz szum	351
11.1. Życie to dynamiczny krajobraz	351
11.2. Świat rzeczywisty jest zaszumiony	362
11.2.1. Zapewnianie różnorodności	366
11.3. Modelowanie trasy statku	370
11.4. Podsumowanie	375
XII. Dzień tygodnia, w którym wypada pierwszy stycznia	381
12. Sieci neuronowe	385
12.1. Neurony progowe i liniowe funkcje dyskryminacyjne	386
12.2. Propagacja wsteczna dla wielowarstwowych perceptronów ze sprzężeniem do przodu	392
12.3. Szkolenie i testowanie	397
12.4. Sieci rekurencyjne i architektury rozszerzone	399
12.4.1. Standardowe sieci rekurencyjne	399
12.4.2. Sieć Hopfielda	400
12.4.3. Maszyna Boltzmanna	401
12.4.4. Sieć wielu współpracujących programów	402
12.5. Grupowanie za pomocą uczenia konkurencyjnego	403
12.6. Wykorzystywanie sieci neuronowych do rozwiązywania TSP .	406
12.7. Ewolujące sieci neuronowe	407
12.8. Podsumowanie	409
XIII. Jakiej długości była lina?	411
13. Systemy rozmyte	415
13.1. Zbiory rozmyte	416
13.2. Zbiory rozmyte i miary probabilistyczne	417
13.3. Operacje na zbiorach rozmytych	418
13.4. Relacje rozmyte	420
13.5. Projektowanie regulatora rozmytego	423
13.6. Grupowanie rozmyte	429
13.7. Rozmyte sieci neuronowe	431
13.8. Podejście rozmyte do TSP	435
13.9. Ewolujące systemy rozmyte	436
13.10. Podsumowanie	437

XIV. Wszystko od czegoś zależy	439
14. Systemy koewolucyjne	449
14.1. Gry	451
14.2. Uczymy się grać w warcaby	453
14.3. Optymalizacja z użyciem konkurujących populacji	456
14.4. Jeszcze jeden przykład gry	461
14.5. Sztuczne życie	464
14.6. Gry ze współpracą i konkurowaniem	466
14.7. Modelowanie inteligentnych agentów	472
14.8. Zagadnienia związane z koewolucją	474
14.9. Przykład koewolucji ze współpracą w zastosowaniu do TSP . .	478
14.10. Podsumowanie	484
XV. Kto jest wyższy?	487
15. Wielokryterialne podejmowanie decyzji	491
15.1. Redukowanie do problemów z jedną liczbą	497
15.1.1. Valuated State Space	500
15.1.2. Metody rozmyte wielokryterialnego podejmowania decyzji	506
15.1.3. Podsumowanie	507
15.2. Podejście ewolucyjne do wielokryterialnego podejmowania decyzji	507
15.2.1. Krótka historia	509
15.2.2. Algorytm ewolucyjny dla wielokryterialnego NLP . .	517
15.3. Podsumowanie	520
XVI. Czy lubisz proste rozwiązania?	523
16. Systemy hybrydowe	529
16.1. Podsumowanie	539
17. Podsumowanie	541
Dodatek A. Rachunek prawdopodobieństwa i statystyka	555
A.1. Podstawowe pojęcia rachunku prawdopodobieństwa	555
A.2. Zmienne losowe	557
A.2.1. Dyskretne zmienne losowe	557
A.2.2. Ciągłe zmienne losowe	561
A.3. Statystyki opisowe zmiennych losowych	562

A.4. Twierdzenia graniczne i nierówności rachunku prawdopodobieństwa	565
A.5. Dodawanie zmiennych losowych	567
A.6. Generowanie liczb losowych przez komputer	568
A.7. Szacowanie	569
A.8. Statystyczne testowanie hipotez	572
A.9. Regresja liniowa	574
A.10. Podsumowanie	576
Dodatek B. Zadania i projekty	577
B.1. Kilka praktycznych problemów	579
B.2. Ogłaszanie wyników eksperymentów obliczeniowych z zastosowaniem metod heurystycznych	585
Bibliografia	587
Skorowidz	617

Przedmowa do drugiego wydania

Nic niewarta jest przyjemność,
w której nie ma różnorodności.

Publiusz Syriusz: *Maksymy moralne*¹

Przez ostatnie cztery lata, które upłynęły od pierwszego wydania tej książki w 1999 roku, otrzymaliśmy od naszych czytelników fantastyczne opinie na jej temat. Otuchy dodaje fakt, że tak wiele osób ją doceniło i – co ważniejsze – używało jej przy rozwiązywaniu swoich problemów. Pewien profesor, który opublikował recenzję książki, napisał, że gdy jego studenci zaczęli korzystać z naszego tekstu, otrzymał najlepsze od 15 lat oceny wykładu. Czy może człowieka spotkać większa pochwała? Warto jeszcze tylko dodać, że także jedna z recenzji książki opublikowana w czasopiśmie SIAM została uznana za najlepszą recenzję. Bardzo jesteśmy wdzięczni za wszystkie miłe słowa i komentarze, jakie nadeszły od czytelników, jak również za wskazanie nam drukarskich, i nie tylko takich, błędów. Dziękujemy wszystkim za to cudowne wsparcie.

Jedną z widomych oznak znaczenia książki jest fakt przetłumaczenia jej na inny język. Szczególne podziękowania należą się Hongqing Cao, która przygotowała chińskie wydanie (a także znalazła kilka błędów). Jeszcze wyraźniejszym znakiem, że książka ma powodzenie, jest przetłumaczenie jej na dwa języki, a ostatnio dotarła do nas informacja, że w przygotowaniu jest polski przekład. Innym dobrym znakiem jest dążenie wydawcy do opublikowania nowego wydania książki. Od czasu pierwszego wydania ponad 80 stron tekstu zostało przere-dagowanych i poprawionych. Mamy więc nadzieję, że to drugie wydanie będzie wolne od błędów! Na prośbę czytelników zawiera ono dwa nowe rozdziały: jeden o systemach koewolucyjnych i drugi o wielokryterialnym podejmowaniu decyzji. Aby zachować charakter poprzedniego wydania, dołączylismy do nich także kilka nowych zagadek. Mamy nadzieję, że czytelnicy uznają je za równie ambitne. Kilka innych rozdziałów też zawiera zmiany, a rozdział 11 został poszerzony o nowy podrozdział.

¹Oryginał łaciński: *Iucundum nihil est, nisi quod reficit varietas* (przyp. tłum.).

Chcielibyśmy skorzystać z okazji, żeby podziękować wszystkim, którzy poświęcili czas na podzielenie się z nami swoimi spostrzeżeniami na temat zawartego w książce materiału. Były one naprawdę bardzo pomocne. Podziękowania należą się setkom czytelników, którzy przysłali nam e-maile z komentarzami dotyczącymi poszczególnych partii tekstu, przykładów, odnośników i zagadek. Dziękujemy także Rodneyowi Johnsonowi, Michaelowi Melichowi i Dave'owi Schafferowi za nader przydatne opinie na temat fragmentów nowych rozdziałów, Min Sun za pomoc w wykonaniu kilku nowych ilustracji, a Brianowi Mayo-howi i Antoniemu Mazurkiewiczowi za zwrócenie naszej uwagi na „zagadkę piratów” i „zagadkę rowerową” (umieszczone w rozdziale XIV). Szczególne podziękowania składamy Carlosowi A. Coello Coello, który przy opracowywaniu opisów metody optymalizacji wielokryterialnej służył nam nieocenioną pomocą i zapewnił eksperymentalne wyniki do kilku zestawów danych. Są one zawarte w punkcie 15.2.2. Dziękujemy również trzem osobom ze Springer-Verlag: Neville'owi Hankinsowi i Ronanowi Nugentowi za czuwanie nad stylem i Ingeborg Mayer za pomoc w czasie całego tego przedsięwzięcia.

Podobnie jak w wypadku pierwszego wydania, będziemy usatysfakcjonowani, jeśli czytelnicy uznają tę książkę za ambitną, zajmującą i skłaniającą do myślenia.

Charlotte, NC
La Jolla, CA
kwiecień 2004

Zbigniew Michalewicz
David B. Fogel

Przedmowa do pierwszego wydania

– Wyjaśnię ci – rzekł pustelnik do Lancelota – sedno sprawy.
Anonim, *Queste del Saint Graal*

Książka *Jak to rozwiązać?* Györgyego Pólyi [357] jest jedną z najważniejszych w XX wieku pozycji poświęconych rozwiązywaniu problemów. Nawet teraz, u progu nowego tysiąclecia, jest ulubioną lekturą nauczycieli i studentów ze względu na pouczające elementy heurystyczne. Pierwsze wydanie tej książki ukazało się w 1945 roku, na krótko przed końcem II wojny światowej i kilka lat przed wynalezieniem tranzystora. Szybko odniosło sukces i drugie wydanie zostało opublikowane w 1957 roku.

Książka *Jak to rozwiązać?* to kompendium sposobów podejścia do problemów, jakie spotykamy w matematyce. Dostarcza nie tylko przykładów metod i procedur, ale także instrukcji, jak przeprowadzać analogie, używać narzędzi pomocniczych, dokonywać analizy wstępnej od celu do danych wejściowych itd. Jest zasadniczo encyklopedią metod rozwiązywania problemów do wykonania odręcznie; co więcej, jest także rozprawą na temat myślenia o formułowaniu i atakowaniu problemów.

Dostępne obecnie książki o heurystycznym rozwiązywaniu problemów dostarczają szczegółowych opisów procedur algorytmicznych dla każdego z kilku klasycznych algorytmów. Niestety, często brakuje w nich właściwych wskazówek, kiedy używać tych algorytmów i – co ważniejsze – kiedy ich nie używać. Opierają się one natomiast na gotowym zestawie przepisów i pozostawiają czytelnikowi ustalenie, czy określona metoda jest odpowiednia do danego zadania, czy nie. Najczęściej jednak czytelnik jest zupełnie nieprzygotowany do podjęcia takiej decyzji, ponieważ nigdy nie nauczono go, na czym ma taki proces polegać, ani nawet nie uświadomiono, że w ogóle jest coś do rozważenia!

Ta sytuacja jest niewątpliwie wynikiem rewolucji komputerowej. Gdyby był jakiś jeden czynnik, który mógłby wymusić unowocześnione podejście do rozwiązywania problemów prawie 50 lat po ukazaniu się książki Pólyi, to byłaby nim dostępność komputerów osobistych po przystępnych cenach. Dziś

do rozstrzygnięcia jakiegoś problemu nie używa się ołówka i papieru. Zamiast tego korzysta się z algorytmów komputerowych, żeby przybliżać i rozszerzać zakres pytań, do których można wygenerować przydatne odpowiedzi. Ponieważ komputery są tak efektywne, człowiek rozwiązujący problem często podejmuje próbę „odfajkowania” rozwiązania lub przynajmniej czegoś, co udaje rozwiązanie, bez odpowiedniego rozważenia założeń, które leżą u podstaw zastosowanej metody.

W rezultacie, mimo ogromnego postępu, jaki dokonał się w kierunku optymalizacji działań w medycynie, obronności, przemyśle, finansach itd., tylko w niewielkiej części wykazaliśmy nasze możliwości. Ilość pieniędzy na przykład, oszczędzona dzięki użyciu typowych metod programowania liniowego, a nie ręcznych obliczeń i spekulacji, musi urastać do miliardów dolarów rocznie, a jednak w rzeczywistym świecie z programowania liniowego prawie zawsze korzysta się niewłaściwie. Ludzie i firmy szukają bez ustanku i niemal na oślep jakiegoś doraźnego środka wśród nieistniejących rozwiązań z półki (ang. *commercial-off-the-shelf*, COTS). Wyobraźmy sobie, ile pieniędzy można by zaoszczędzić (lub zarobić!), gdyby naprawdę odpowiednie metody były stosowane do problemów, które wykraczają poza prostą heurystykę programowania liniowego.

Ponieważ świat zmierza w kierunku bardziej otwartych i wolnych rynków, konkurencja jest siłą napędową wprowadzania skuteczniejszych metod rozwiązywania problemów. Analogia między mechanizmem doboru naturalnego z teorii Darwina a dynamiką wolnego rynku jest trafna. Ludzie, którzy nie pozyskają niezbędnych środków, zbankrutują, co jest ekonomicznym odpowiednikiem śmierci i przetrwania najsilniejszych. Wystarczy tylko mała przewaga nad rywalem, żeby wyeliminować go z rynku. Przetrwają tylko światłe jednostki, firmy i agencje, które starają się przyjąć nowoczesną heurystykę do rozwiązywania swoich problemów.

Obecnie potrzebne jest dwuczęściowe unowocześnione podejście do doboru Pólyi. Po pierwsze, czytelnik musi poznać konkretne metody, dostępne głównie dzięki zastosowaniu algorytmów komputerowych. Po drugie, musi zrozumieć, kiedy te metody powinny być używane, a kiedy nie, oraz dowiedzieć się, jak formułować własne problemy tak, by jak najlepiej dało się w nich wykorzystać heurystykę, której Pólya nie mógł w swojej książce *Jak to rozwiązać?* przewidzieć.

Nasza książka jest pierwszą próbą pełnego spojrzenia na rozwiązywanie problemów w XXI wieku. Czytelnik poznaje główne zagadnienia przez opisy działań, analogie i przykłady oraz przez zbiór problemów i zagadek umieszczonych w tekście i między rozdziałami. Pisząc tę książkę, założyliśmy, że będzie ona lekturą obowiązkową do wykładu z nowoczesnej heurystyki. Jesteśmy przekonani, że taki wykład jest *niezbędny* dla studentów nauk ścisłych, zarządzania czy inżynierii. Dobrze więc, by czytelnik miał podstawowe wiadomości z matematyki dyskretnej i programowania komputerowego. Ktoś, kto ma problemy do rozwiązywania, a brakuje mu tej wiedzy, powinien poświęcić trochę czasu na zdobycie jej, bo naprawdę warto. Dla kogoś, kto chce bardziej dogłębnej analizy matematycznej algorytmów, przeczytanie tej książki będzie pozytywnym punktem wyjścia do bardziej zaawansowanych wykładów.

Książka składa się z 15 rozdziałów, które poprzedza ogólne omówienie we wstępie tematu rozwiązywania problemów. W rozdziale 1 wskazujemy główne źródła trudności w rozwiązywaniu problemów, a w krótkim rozdziale 2 podajemy podstawowe pojęcia i definicje. W rozdziałach 3 i 4 oceniamy klasyczne algorytmy optymalizacyjne. Wraz z rozdziałem 5, w którym opisujemy dwa nowoczesne algorytmy przeszukiwania, rozdziały te stanowią pierwszą część książki. Dalej zajmujemy się podejściem ewolucyjnym do rozwiązywania problemów. W rozdziałach 6 i 7 przyjmujemy pewne intuicyjne założenia i podstawy związane z nowymi algorytmami ewolucyjnymi. W rozdziałach 8-10 zajmujemy się kilkoma ambitnymi zagadnieniami dotyczącymi problemów, które wymagają poszukiwania konkretnych permutacji elementów, radzenia sobie z ograniczeniami i dostosowywania algorytmów do problemu. Rozdziały te stanowią szczegółowy przegląd osiągnięć uzyskanych w tych dziedzinach. Rozdział 11 dotyczy zmiennego w czasie środowiska i zakłóceń. Dwa następne rozdziały (12 i 13) zawierają materiał dotyczący sieci neuronowych i systemów rozmytych. W rozdziale 14 omawiamy ogólnie systemy hybrydowe i rozwinięcia algorytmów ewolucyjnych. Książkę kończy rozdział 15, będący podsumowaniem materiału i zawierający wskazówki do praktycznego rozwiązywania problemów. Każdy zbiór zagadek (I, II itd.) ilustruje zagadnienie omówione w następnym rozdziale (1, 2 itd.) – rozwiązywanie problemów bowiem powinno być zabawą. Mamy nadzieję, że takie właśnie podejście sprawi czytelnikowi przyjemność i będzie ciekawe.

W książce są jeszcze dwa dodatki, które dostarczają informacji uzupełniających. Dodatek A zawiera ogólny przegląd podstawowych pojęć związanych z prawdopodobieństwem i statystyką. Na początku przedstawiamy aksjomaty rachunku prawdopodobieństwa, a dalej zajmujemy się testowaniem hipotez statystycznych i regresją liniową. Dodatek ten nie może zastąpić podręcznika z pełnym wykładem procesów losowych, ale można go wykorzystać do odświeżenia pamięci i wyjaśnienia ważnych zagadnień związanych z zastosowaniem prawdopodobieństwa i statystyki do rozwiązywania problemów. W dodatku B proponujemy problemy i zadania, z których można skorzystać, gdy książka jest używana jako podręcznik do wykładu. Nawet jeśli czytasz tę książkę bez specjalnej zachęty ze strony Twojego nauczyciela, zachęcamy Cię gorąco do rozwiązywania problemów oraz wdrażania, testowania i stosowania pomysłów, które w niej poznałeś. Materiał zawarty w dodatku B może służyć jako przewodnik do takich zastosowań.

Dziękujemy wszystkim, którzy poświęcili swój czas i podzielili się z nami uwagami na temat naszego tekstu; były one bardzo pomocne. Na szczególną wdzięczność zasługują: Dan Ashlock, Ole Caprani, Tom English, Larry Fogel, Larry Hall, Jim Keller, Kenneth Kreutz-Delgado, Martin Schmidt i Thomas Stidsen. Specjalne wyrazy uznania należą się Kumarowi Chellapilli, który nie tylko zrecenzował niektóre partie książki, ale także służył nieocenioną pomocą przy opracowaniu niektórych rysunków. Miło nam również wyrazić wdzięczność za pomoc kilku współautorom, z którymi współpracowaliśmy przez ostatnie dwa lata; wiele wyników tej współpracy uwzględniliśmy w tekście. Dziękujemy również takim osobom, jak: Thomas Bäck, Hongqing Cao, Ole Caprani, Dipankar Dasgupta, Kalyan Deb, Gusz Eiben, Susana Esquivel, Raul Gallard, Özdemir

Göl, Robert Hinterding, Sławomir Kozięł, Lishan Kang, Moutaz Khouja, Witold Kosiński, Thiemo Krink, Guillermo Leguizamón, Brian Mayoh, Maciej Michalewicz, Guo Tao, Krzysztof Trojanowski, Marc Schoenauer, Martin Schmidt, Thomas Stidsen, Roman Śmierzchalski, Martyna Weigl, Janek Wieczorek, Jing Xiao oraz Lixin Zhang.

Dziękujemy redaktorowi naczelnemu Springer-Verlag, Hansowi Wössnerowi, za pomoc w trakcie trwania tego przedsięwzięcia, prof. Leonardowi Bolcowi za zachętę i prof. Antoniemu Mazurkiewiczowi za interesujące dyskusje na temat wielu zagadek zamieszczonych w książce. Specjalne podziękowania należą się Joylene Vette, redaktorowi wydania angielskiego w Springer-Verlag, za cenne komentarze do pierwotnej wersji tekstu. Pierwszy z autorów chciałby także wyrazić wdzięczność za wspariałe środowisko pracy w Aarhus University, gdzie przebywał na urlopie naukowym od sierpnia 1998 roku do lipca 1999 roku, a także za granty National Science Foundation (IRI-9322400 i IRI-9725424) oraz ESPRIT Project 20288 Corporation Research in Information Technology (CRIT-2): „Evolutionary Real-time Optimization System for Ecological Power Control”, dzięki którym możliwe było przygotowanie kilku rozdziałów tej książki. Chciałby również podziękować wszystkim studentom studiów magisterskich z UNC Charlotte (USA), Universidad Nacional de San Luis (Argentyna) oraz Aarhus University (Dania), którzy brali udział w wykładach prowadzonych w latach 1997-1999 i przeszli przez bolesny proces rozwiązywania problemów. Drugi autor pragnie podziękować studentom uczęszczającym na wykład z uczenia się maszyn i rozpoznawania wzorców, który prowadził w UC San Diego zimą 1999 roku, oraz swoim kolegom z Natural Selection: Billowi Porto, Pete Angeline'owi i Gary'emu Fogelowi za ich zainteresowanie i wsparcie. Specjalne podziękowania należą się Jacquelyn Moore, która poświęciła wiele weekendów i wieczorów, żeby ta książka mogła się ukazać.

Nasz cel zostanie osiągnięty, gdy czytelnicy uznają tę książkę za ambitną, zajmującą i skłaniającą do refleksji. Mamy nadzieję, że przeczytanie jej sprawi im taką przyjemność, jak nam pisanie jej, i że ponadto odniosą z niej jakieś korzyści dla siebie.

Charlotte, NC
La Jolla, CA
wrzesień 1999

Zbigniew Michalewicz
David B. Fogel

Wstęp

Rozsądny człowiek przystosowuje się do świata, człowiek nierożsądny ciągle stara się przystosować świat do siebie. Zatem wszelki postęp zależy od człowieka nierożsądnego.

George Bernard Shaw: *Maksymy dla rewolucjonistów*¹

To nie jest książka o algorytmach. Oczywiście jest w niej mnóstwo algorytmów, ale to nie o nich jest ta książka. To jest książka o możliwościach. Jej celem jest wyposażenie czytelnika nie tylko we wstępную obowiązkową wiedzę na temat dostępnych metod rozwiązywania problemów, ale także – co ważniejsze – rozwinięcie jego umiejętności formułowania nowych problemów i twórczego myślenia, która jest zapomnianą sztuką. Krótko mówiąc, chodzi tu o rozwiązanie problemu, jak rozwiązywać problemy. Zamiast poświęcić odpowiedni czas na przemyślenie sformułowania problemu i sensowne zestawienie elementów układanki, zbyt pewnie sięgamy po najwygodniejszą procedurę – magiczny lek na wszystkie nasze bolączki. Kłopot z magią polega na tym, że w rzeczywistości rzadko odnosi ona skutek i często odwołuje się do takich środków, jak lustro czy dym. Jak w wypadku magii, większość metod rozwiązywania problemów w realnym świecie jest iluzoryczna. Rozwiązania bowiem mogłyby być dużo lepsze.

Potrzeba efektywnego rozwiązywania problemów nigdy nie była większa. Technologia wyposażała nas w możliwość wpływania na środowisko w takim stopniu, że decyzje, które podejmujemy dzisiaj, mogą mieć nieodwracalne skutki w przyszłości. Ta sama technologia sprawia, że nieustannie powiększa się liczba osób, z którymi mamy kontakt i które mają wpływ na nasze życie. W rezultacie rozwiązywanie problemów staje się coraz trudniejsze, ponieważ trzeba brać pod uwagę coraz więcej czynników, a potencjalnie najgorszym sposobem działania jest ignorowanie ich w nadziei, że rozwiązania opracowane dla „prostszego problemu” zadziałają mimo tych dodatkowych czynników. Bez względu na to, czy ustalasz odpowiednie funkcjonowanie swojej firmy, właściwe

¹Na podstawie: <http://belfer.univ.szczecin.pl/~edipp/aktywnosc1.htm> (przyp. tłum.).

wykorzystanie zasobów naturalnych swojego kraju, czy jedynie najkrótszą drogę do pracy, nie możesz sobie pozwolić na zignorowanie swoich związków z resztą świata. Ponieważ związki te stają się coraz częstsze i bardziej skomplikowane, pilnie potrzebne są procedury radzenia sobie z rzeczywistymi problemami. Wiele można zyskać na rozwiązywaniu problemów, ale też wiele stracić na rozwiązywaniu ich źle.

Wszyscy mamy problemy. Problem pojawia się, kiedy istnieje stwierdzona rozbieżność między stanem rzeczywistym a pożadanym. Rozwiązania z kolei są sposobami na takie przydzielenie dostępnych zasobów, żeby tę rozbieżność zmniejszyć. Żeby zrozumieć sytuację, musimy najpierw zdać sobie sprawę, że problemy mają ludzie podejmujący decyzje prowadzące do określonych celów. Jeśli nie mamy celu, nie mamy problemu. Sąk w tym, że nie tylko każdy z nas ma cel, ale też często nasze cele są rozbieżne, a przynajmniej się nie uzupełniają. Żeby ustalić, jak najlepiej rozwiązać problem, nie możesz go rozważyć w oderwaniu od wszystkiego, ale raczej brać pod uwagę, jak działania innych osób mogą wpływać na możliwe rozwiązanie. Jeśli masz kontakt z innymi ludźmi, jest bardzo prawdopodobne, że będą zmieniali ulokowanie swoich zasobów – czasami tak, żeby Ci pomóc, czasem, żeby stanąć Ci na przeszkodzie. Wdrażanie rozwiązania bez uprzedniego rozważenia: „Co dalej?”, jest jak traktowanie pokera jako gry jednoosobowej; przy takim podejściu Ty i Twoje pieniądze szybko się rozstanecie.

Warunkiem niezbędnym, żeby rozpatrywać świat jako pole gry wielu graczy, jest umiejętność manipulowania zestawem metod rozwiązywania problemów, tj. algorytmów opracowanych dla różnych sytuacji. Niestety, rzeczywistość prawie zawsze obdarza nas warunkami, które mało lub zasadniczo różnią się od wymaganych przez te metody. Na przykład pewna konkretna metoda umożliwia obliczenie minimalnych kosztów przedziału zasobów dla problemu, w którym zarówno funkcja kosztu, jak i ograniczenia (równości i nierówności) są liniowe. Metoda jest szybka i niezawodna, a jednak w rzeczywistych warunkach, w których funkcja kosztów i ograniczenia są zwykle nieliniowe, jest prawie zawsze używana *niewłaściwie*. Zasadniczo każdy, kto używa tej metody, ryzykuje wygenerowanie dobrego rozwiązania problemu, który nie istnieje. Strata poniesiona przy użyciu tego rozwiązania jest wielokrotna: nie tylko przez odpowiednie rozpatrzenie dostępnych ograniczeń w świetle rzeczywistej funkcji kosztów można by uzyskać lepsze rozwiązanie, ale w dodatku przeciwnik, odpowiednio potraktowawszy sytuację, może naprawdę znaleźć lepsze rozwiązanie! Wystarczy mała przewaga po naszej stronie, żeby doprowadzić konkurenta do bankructwa. (Las Vegas działa na tej zasadzie od wielu lat). Z drugiej strony używanie szybkich, ale przybliżonych rozwiązań może okazać się właściwe. Jeśli przeciwnik używa wolnej procedury do wypracowania dokładnego rozwiązania, my możemy wcześniej znaleźć niedokładne, ale użyteczne rozwiązanie; a niekiedy każde rozwiązanie jest lepsze od czekania na jeszcze lepsze. Musimy rozważyć, który sposób postępowania będzie właściwy.

Niestety, rozważania takie są w zaniku, a przynajmniej nie w modzie. Być może, jest tak dlatego, że wymagają one wyżejonej pracy. Żeby dobrze rozważyć rozwiązanie problemu, trzeba wiedzieć, co każda z metod zakłada, jakie

ma wymagania obliczeniowe, jaki jest jej stopień niezawodności itd. Uzyskanie tych informacji wymaga czasu i wysiłku. Znacznie łatwiej jest po prostu przejrzeć Internet lub dostępne na rynku oprogramowanie i wyszukać coś, co prawdopodobnie rozwiąże problem od ręki. Niektóre firmy i agencje jako zasadę stosują kupowanie takiego oprogramowania z półki sklepowej (COTS). Oczywiście kosztuje to początkowo mniej, ale często jest drobną oszczędnością przy jednoczesnym bezsensownym wydawaniu dużych sum. Po latach, gdy okazuje się, że zakupione oprogramowanie nie było odpowiednie dla potrzeb danego biznesu, a konkurencja wypracowała sensowne rozwiązania szybciej, leży ono bezużytecznie, podczas gdy pracownicy znów starają się znaleźć rozwiązanie pierwotnego problemu, czasami ręcznie!

Zamiast rozpaczать nad tą sytuacją, należy w niej dostrzec szansę. Wielka nagroda czeka na tych, którzy potrafią efektywnie rozwiązywać problemy, a umiejętność tę można rozwijać. Należy zacząć od dobrego zrozumienia celu, który mamy osiągnąć. Stanowczo zbyt często, próbując wykorzystać znany algorytm, zmieniamy cel tylko po to, żeby pasował do algorytmu. To tak, jakby za pomocą młotka próbować wkręcić śrubę w deskę: wbijamy śrubę w deskę, chociaż wiemy, że już nigdy nie zdołamy jej wykręcić. Gdy mamy tylko młotek, wszystko wygląda jak gwóźdź. Podobnie, gdy posiadamy jedynie mały zestaw klasycznych metod rozwiązywania problemów, wszystkie funkcje kosztów magicznie zmieniają się w gładkie, liniowe, wypukłe odwzorowania, a każdy model jest optymalizowany przy braniu pod uwagę średniego kwadratu błędu. Skutkuje to „idealnymi” odpowiedziami o wątpliwym związku z pierwotnym problemem. Zrozumienie, kiedy cel podejmującego decyzję powinien lub nie powinien być przybliżany przez metody liniowe lub inne metody upraszczające, ma podstawowe znaczenie. Ujęcie subiektywnego celu w obiektywnych, mierzalnych kategoriach tak, żeby była możliwa ocena istotnych różnic między przybliżeniem a rzeczywistością, to trudna i wymagająca naprawy umiejętność. Możemy ją nabyć jedynie przez praktykę.

Kolejnym krokiem po ocenie wymaganego przy modelowaniu celu stopnia wierności jest wybór taktyki znalezienia optymalnego przydziału dostępnych środków. Istnieje mnóstwo sposobów, żeby wykonać to zadanie. Jest wiele książek poświęconych algorytmom rozwiązywania problemów, z których każda prezentuje bogactwo konkretnych metod. Aż kusi, żeby po prostu wypróbować algorytm, tak jak wyprobowuje się przepis na ciasto. Ten naturalny odruch jest pozbawiony aspektu twórczego – nie daje nam szansy wykazania się pomysłowością. Zamiast tego zmusza, żeby dopasować problem do ograniczeń konkretnego algorytmu. Prezentując pośpiesznie metody rozwiązywania problemów, każdą z nich najczęściej opisujemy oddzielnie, przeoczając w ten sposób potencjalnie istotne zjawiska synergiczne zachodzące między poszczególnymi metodami. To tak, jakby można było skomplikowaną układankę ułożyć bez rozważenia, w jaki sposób dopasowywać poszczególne kawałki. Łączenie różnych podejść z reguły umożliwia osiągnięcie lepszych rozwiązań rzeczywistych problemów. Efektywne rozwiązywanie problemów wymaga czegoś więcej niż tylko znajomości algorytmów; wymaga starannego ustalenia najlepszej kombinacji metod, potrzebnych do osiągnięcia wybranego celu w danym czasie i przy spodziewanych reakcjach

innych osób, mających ten sam cel. Uświadomienie sobie złożoności realnych problemów jest niezbędne do ich efektywnego rozwiązania.

Wcześniej jednak napotykamy różnego rodzaju trudności, nawet już w szkole podstawowej. Jesteśmy uczeń rozkładania problemów i rozwiązywania oddziennie mniejszych, prostszych zagadnień. Podejście to jest cudowne, jeśli działa. Skomplikowane rzeczywiste problemy nieczęsto można jednak łatwo i sensownie podzielić. Co gorsza, karmi się nas rozwiązaniami, rozdział po rozdziale, nigdy nie zmuszając do zastanowienia się, czy problemy, przed którymi stajemy, powinny być rozwiązywane za pomocą metod właśnie opisywanych w podręczniku. Oczywiście, że muszą być tak rozwiązywane! *Inaczej po co ten problem byłby w tym rozdziale?!* Sytuacja ta dotyczy nie tylko podręczników matematyki dla szkół podstawowych i średnich, ale także większości podręczników akademickich, a nawet wielu monografii wyjaśniających poszczególne metody i ilustrujących ich użycie różnymi przykładami. W książkach tych problem i jego rozwiązanie nigdy nie są daleko od siebie.

Przyjrzyjmy się podręcznikowi do matematyki dla amerykańskiej szkoły średniej. Jest w niej rozdział o rozwiązywaniu równań kwadratowych. Na końcu rozdziału umieszczone kilka zadań do rozwiązania. Jedno z nich brzmi: farmer ma prostokątne ranczo o obwodzie 110 m i powierzchni 700 m^2 . Jakie są wymiary tego rancza? Uczeń wie, że aby rozwiązać ten problem, powinien zastosować coś, co było omówione w tym samym rozdziale. Próbuje więc ułożyć równanie kwadratowe. Przedzej czy później dochodzi do dwóch równości:

$$\begin{cases} 2x + 2y = 110 \\ xy = 700 \end{cases}$$

które prowadzą do równania kwadratowego:

$$x(55 - x) = 700$$

Następnie, stosując znane procedury, z łatwością oblicza wartość x i rozwiązuje zadanie.

W innym rozdziale dotyczącym geometrii omówiono własności trójkątów. Wyjaśniono twierdzenie Pitagorasa i zakończono kolejną porcją zadań do rozwiązania. Uczeń nie ma wątpliwości, że powinien zastosować to twierdzenie przy rozwiązywaniu podanych problemów.

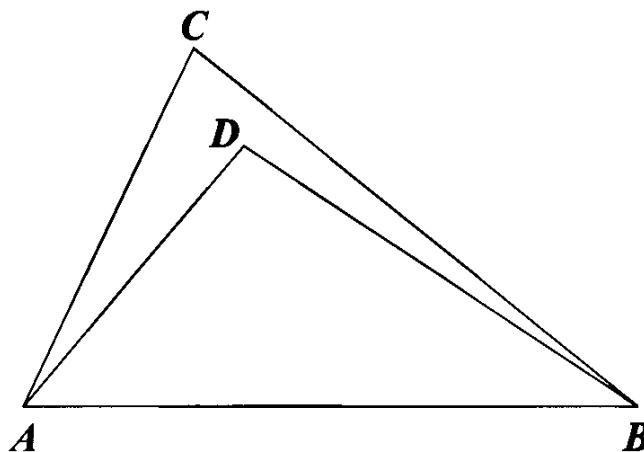
Wydaje się jednak, że nie jest to właściwy sposób nauczania. Związek między problemem a metodą powinien być omawiany z perspektywy problemu, a nie metody. Działanie odwrotne na dłuższą metę przynosi więcej szkody niż pozytku, ponieważ jak wszystkie „gotowce” czyni ucznia niezdolnym do radzenia sobie (tj. myślenia) samodzielnie.

Zaprezentujmy ten problem na przykładzie dwóch zadań. Pierwsze z nich jest następujące: mamy trójkąt ABC oraz punkt D , będący dowolnym punktem wewnętrznym trójkąta (rys. W.1). Należy udowodnić, że

$$AD + DB < AC + CB$$

przy czym: AD oznacza odległość między punktami A i D , a pozostałe oznaczenia wskazują analogicznie odległości między odpowiadającymi im wierzchołkami.

Problem ten jest *tak prosty*, że wydaje się, iż nie ma czego udowadniać. Jest oczywiste, że suma długości dwóch odcinków wewnątrz trójkąta *musi* być mniejsza od sumy długości dwóch jego boków! Ale tym razem problem został wyjęty z kontekstu swojego „rozdziału” i w związku z tym uczeń nie ma pojęcia, czy powinien zastosować twierdzenie Pitagorasa, ułożyć równanie kwadratowe, czy zrobić jeszcze coś innego!



Rysunek W.1. Trójkąt ABC z wewnętrznym punktem D

Zadanie to jest trudniejsze, niż się na pierwszy rzut oka wydaje. Daliśmy je do rozwiązania wielu osobom, w tym studentom studiów licencjackich i magisterskich, a nawet profesorom matematyki, informatyki i nauk technicznych. Mniej niż 5% z nich rozwiązało zadanie w przeciągu godziny, wielu z nich potrzebowało na to kilku godzin, a niektórzy w ogóle nie znaleźli rozwiązania. To interesujące, zważywszy fakt, że zadanie pochodzi z podręcznika matematyki dla piątklasistów w USA. Łatwo je rozwiązać, gdy znajduje się na końcu odpowiedniego rozdziału, ale pozbawione jego kontekstu okazuje się niezwykle trudne.

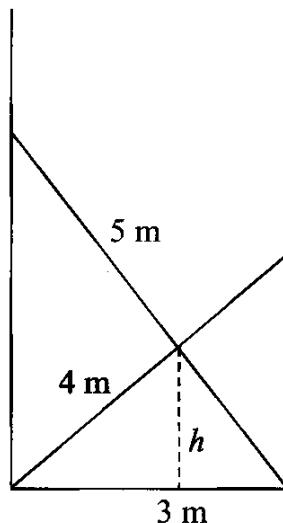
W tym miejscu zachęcamy do odłożenia książki na bok i próby rozwiązania tego problemu. Żeby zmniejszyć pokusę dalszej lektury bez *pomyślenia*, umieściliśmy rozwiązanie tej zagadki w ostatnim rozdziale książki. Jeśli nie uda Ci się znaleźć rozwiązania w przeciągu godziny, książka ta prawdopodobnie jest dla Ciebie!¹

Przejdzmy do innej zagadki. Mamy studnię otwartą u góry, o średnicy 3 metrów. Wrzucamy do niej dwa patyki o długości odpowiednio 4 i 5 metrów. Załóżmy, że patyki ułożyły się tak, jak przedstawiono na rys. W.2, tzn. leżą w tej samej płaszczyźnie i „przecinają się”. Problem jest w zasadzie dwuwymiarowy, gdyż studnię można by zastąpić otwartym u góry prostokątem.

Zadanie polega na określeniu odległości h od dna studni do miejsca, w którym patyki się przecinają. Ułożenie i długości patyków zapewniają jednoznaczne rozwiązanie i wydaje się, że zadanie jest łatwe. I rzeczywiście jest łatwe;

¹Zauważ jednak, że daliśmy Ci już cenną podpowiedź: problem jest podstawowy, nie ma więc potrzeby używania jakichkolwiek wyrafinowanych narzędzi. Bez tej wskazówki byłoby znacznie trudniej. Zapewne skorzystasz też, czytając naszą książkę, nawet jeśli uda Ci się rozwiązać ten problem w godzinę!

jego rozwiązanie zajmuje mniej niż minutę i nie powinno nas dziwić, że zagadka i tym razem pochodzi z materiału dla szkoły podstawowej. Teraz jednak brakuje kontekstu odpowiedniego rozdziału, a my ponownie zachęcamy czytelnika do znalezienia rozwiązania. Jeśli uda Ci się to w przeciągu godziny (!), będziesz należał do elitarnego grona 1% testowanych przez nas osób, które zmieściły się w tym czasie. Co więcej, wszyscy badani mieli przynajmniej stopień licencjata z matematyki, informatyki bądź nauk technicznych. Tak jak w wypadku poprzedniej zagadki, rozwiązanie znajdziesz w ostatnim rozdziale książki.



Rysunek W.2. Dwa patyki w studni

Chcielibyśmy jasno uświadomić wszystkim, że uczymy się stosowania konkretnych metod do konkretnych problemów, ale tylko wiedząc, że te właśnie metody powinny idealnie pasować do tych problemów. Prawie nigdy nie uczymy się ogólnego myślenia o rozwiązywaniu problemów; ograniczamy się do koncentrowania się na poszczególnych zadaniach na końcu rozdziału, korzystając z informacji zawartych w tym rozdziale. Mamy nadzieję, że to się zmieni. Z utęsknieniem czekamy na dzień, w którym zadania w podręczniku matematyki dla pierwszoklasistów ulegną zmianie. Na przykład, gdy zadanie:

Lot rakiety z Ziemi na Księżyc trwa 48 godzin. Ile czasu będzie trwał lot, gdy rakieta będzie leciała dwa razy szybciej?

zamieni się na:

Lot rakiety z Ziemi na Księżyc trwa 48 godzin. Ile czasu będzie trwał lot dwóch rakiet?

co zmusi dziecko (a nawet dorosłego) do *pomyślenia* (czy należy pomnożyć, czy podzielić 48 przez 2, czy też może jest to nadal 48 godzin). Nawet dla starszych klas widzimy inne możliwości:

Rzucamy monetą. Wypada orzeł. Jakie są Twoje przewidywania co do następnego rzutu?

To, czego brakuje większości programów na wszystkich poziomach nauczania – od szkoły podstawowej po uniwersytet – to rozwijanie umiejętności

rozwiązywania problemów. Młodzi ludzie często mają trudności z rozwiązyaniem realnego problemu, ponieważ rzeczywiste problemy nie występują na końcu rozdziałów! Celem tej książki jest nie tylko zaprezentowanie czytelnikowi szeregu algorytmów i metod do rozwiązywania książkowych problemów, ale także zmuszenie go do *myślzenia*, jak formułować i rozwiązywać problemy, zwłaszcza te, których nie spotyka się na końcu książkowych rozdziałów.

My, ludzie, lubimy chwalić się naszą zdolnością do rozwiązywania problemów. Posunęliśmy się nawet do zdefiniowania się w kategoriach myślenia: wiedzy, oświecenia, geniuszu. Wynika z tego jasno, że tak mądra, oświecona i genialna istota powinna być w stanie rozwiązywać swoje problemy. A jednak świat jest pełen ludzi mających problemy. Mamy nadzieję, że książka ta pomoże Ci rozwiązać nie tylko problemy własne, ale i innych ludzi.

I. Ile lat mają moi trzej synowie?

Jednym z powodów, dla których rozwiązywanie problemów jest często trudne, jest fakt, że nie wiemy, jak zacząć. Gdy mamy już wytyczony jakiś sposób postępowania, możemy podążać za nim i szczerze dotrzeć do rozwiązania. Ale wyczarowanie tej drogi postępowania to nie lada wyzwanie. Często jest to najtrudniejszy etap w poszukiwaniu rozwiązania, ponieważ jest całkowicie twórczy: trzeba dosłownie *stworzyć* plan osiągnięcia rozwiązania. Bez planu jesteśmy straceni. To jest pora, kiedy powinniśmy w pełni wykorzystać możliwość zabłyśnięcia naszą inteligencją! Jednym ze sposobów zawsze pomocnych przy zbliżeniu się od problemu do rozwiązania jest rozważenie wszystkich dostępnych danych. Ta rada sprawdza się zawsze, ale jest szczególnie ważna przy pierwszym rozważaniu, jak znaleźć rozwiązanie. Niewzięcie na wstępie wszystkiego pod uwagę oznacza, że możemy przegapić tę jedyną szansę na rozpoczęcie rozwiązania. Oceń dokładnie wszystkie dane, ustal, co z nich wynika, a następnie zobacz, czy możesz z nimi powiązać cel.

Wypróbujmy tę procedurę na przykładzie następującego problemu. Dwaj mężczyźni spotykają się na ulicy. Nie widzieli się od wielu lat. Rozmawiają o różnych rzeczach i po chwili jeden z nich mówi do drugiego:

– Skoro jesteś profesorem matematyki, dam ci zagadkę do rozwiązania. Dziś jest dla mnie bardzo ważny dzień: wszyscy moi synowie mają dziś urodziny! Czy możesz mi powiedzieć, ile ka  dy z nich ma lat?

– Oczywiście – odpowiada matematyk – ale musisz mi trochę o nich opowiedzie .

– W porządku, dam ci parę wskazówek – odpowiada ojciec trzech synów. Iloczyn lat życia moich synów wynosi 36.

– Dobrze – mówi matematyk – ale potrzebuję czego  więcej.

– Suma ich lat życia równa si  liczbie okien w tym budynku – odpowiada ojciec, wskazując na budowl  za nimi.

Matematyk myśli przez chwilę, po czym mówi:

- No cóż, potrzebuję dodatkowej wskazówki, żeby rozwiązać twoją zagadkę.
- Mój najstarszy syn ma niebieskie oczy – dodaje ojciec.
- To wystarczy! – wykrzykuje matematyk i podaje koledze prawidłowy wiek jego trzech synów.

Twoim zadaniem jest zrobić to samo: rozumować jak matematyk i rozwiązać zagadkę. Zagadka i tym razem jest całkiem prosta, a jednak wiele osób ma z nią trudności.

· · · ·

· · · ·

No i jak Ci poszło? Jeśli nie oparłeś się pokusie, żeby po prostu czytać dalej i poznać odpowiedź, dla własnego dobra zrób to teraz, zanim będzie za późno!

Na początek przeanalizujmy dokładnie wszystkie informacje dostarczone w rozmowie. Czego dowiadujemy się na podstawie pierwszej wskazówki? Jeśli iloczyn lat życia trzech synów wynosi 36, jest tylko osiem możliwości do rozważenia; to ogranicza obszar naszych poszukiwań do tych ośmiu przypadków:

Wiek 1. syna	Wiek 2. syna	Wiek 3. syna
36	1	1
18	2	1
12	3	1
9	4	1
9	2	2
6	6	1
6	3	2
4	3	3

Druga wskazówka mówi nam, że suma lat życia synów jest taka sama jak liczba okien w pobliskim budynku. Zakładamy, że matematyk znał liczbę okien, znał więc też tę sumę. Jakie mamy tu możliwości? W jaki sposób ta informacja jest użyteczna? Dodanie liczb we wszystkich ośmiu przypadkach daje następujące sumy:

36	+	1	+	1	=	38
18	+	2	+	1	=	21
12	+	3	+	1	=	16
9	+	4	+	1	=	14
9	+	2	+	2	=	13
6	+	6	+	1	=	13
6	+	3	+	2	=	11
4	+	3	+	3	=	10

Nagle wszystko staje się jasne. Gdyby liczba okien w budynku wynosiła 21 (lub 38, 16, 14, 11 czy 10), matematyk podałby odpowiedź natychmiast. Zamiast tego, poprosił o dodatkową wskazówkę. Oznacza to, że okien musiało być 13, pozostawiając nam dwie (i tylko dwie) możliwości:

(9, 2, 2) lub (6, 6, 1)

Ponieważ druga z możliwości nie wykazuje istnienia *najstarszego* syna („*najstarszy* syn ma niebieskie oczy”), synowie muszą mieć odpowiednio 9, 2 i 2 lata.

Co czyni ten problem łatwiejszym, niż mógłby być, to fakt, że należy rozważyć podane w nim informacje w kolejności ich zaprezentowania. Zupełnie, jak gdyby autorzy zagadki podpowiadali nam właściwą odpowiedź! Prawdziwe problemy nie są tak ładnie uporządkowane. Mimo to, jeśli rozważysz wszystkie dostępne dane, dasz sobie najlepszą szansę na znalezienie dobrego punktu wyjścia.

1. Dlaczego niektóre problemy są trudne do rozwiązania?

Wszystkie nasze problemy zostały stworzone przez ludzi
i mogą zostać przez ludzi rozwiązane.

John F. Kennedy, przemówienie z 10 czerwca 1963 roku¹

Chyba zaryzykujemy rozpoczęcie tautologią, stwierdzając, że rzeczywiste problemy są trudne do rozwiązania. Przyczyn trudności jest wiele.

- Liczba możliwych rozwiązań w *przestrzeni przeszukiwania* (przestrzeni poszukiwań) jest tak duża, że uniemożliwia przeszukiwanie wyczerpujące w celu znalezienia najlepszego z nich.
- Problem jest tak skomplikowany, że aby uzyskać jakąkolwiek odpowiedź, musimy używać modeli problemu uproszczonych na tyle, że każdy rezultat jest praktycznie bezużyteczny.
- *Funkcja oceny*, która opisuje jakość dowolnego proponowanego rozwiązania, jest zakłócana lub zmienia się w czasie, dlatego wymaga nie jednego rozwiązania, ale szeregu rozwiązań.
- Możliwe rozwiązania są tak mocno ograniczone, że trudno jest wygenerować choćby jedno dopuszczalne rozwiązanie, a co dopiero znaleźć rozwiązanie optymalne.
- Osoba rozwiązująca problem nie jest odpowiednio przygotowana albo jest obarczona jakąś psychologiczną barierą, uniemożliwiającą jej znalezienie rozwiązania.

Listę tę można by oczywiście wydłużyć o inne możliwe przeszkody. Moglibyśmy tutaj dodać zakłócenia lub szумy związane z naszymi obserwacjami i pomiarami, niepewność związaną z posiadanymi informacjami oraz trudności pojawiające się w problemach, w których dąży się do osiągnięcia wielu, być może, pozostających w konflikcie, celów (być może w takiej sytuacji konieczne jest uzyskanie zestawu rozwiązań, a nie jednego rozwiązania). Na razie powyższa lista nam wystarczy. Każda jej pozycja stanowi problem sam w sobie.

¹Za <http://zlotemysli.w.interia.pl/sentencje/k-/kennedy.html> (przyp. tłum.).

1.1. Wielkość przestrzeni przeszukiwania

Jednym z podstawowych problemów w logice jest problem spełnialności formuł boolowskich (SAT). Zadanie polega na sprawieniu, że zdanie złożone ze zmiennych boolowskich przyjmuje wartość TRUE. Weźmy na przykład następujący problem ze 100 zmiennymi dany w koniunkcyjnej postaci normalnej:

$$F(\mathbf{x}) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97})$$

Należy znaleźć wartościowanie logiczne dla każdej zmiennej x_i , dla wszystkich $i = 1, \dots, 100$ takie, że $F(\mathbf{x}) = \text{TRUE}$. Możemy użyć 1 i 0 jako odpowiedników TRUE i FALSE, zauważmy jednak, że \bar{x}_i jest tutaj zaprzeczeniem x_i (tzn. gdyby x_i było TRUE lub 1, to \bar{x}_i byłoby FALSE lub 0).

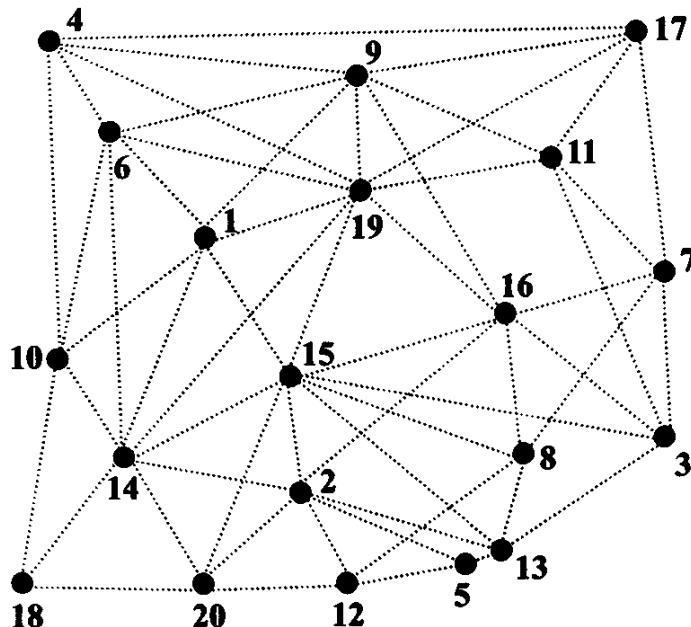
Niezależnie od przedstawionego problemu, zawsze warto rozważyć przestrzeń możliwych rozwiązań. W tym wypadku dowolny ciąg dwójkowy o długości 100 jest potencjalnym rozwiązaniem problemu. Mamy dwie możliwości dla każdej zmiennej, a przy ponad 100 zmiennych daje to 2^{100} możliwości. W ten sposób wielkość przestrzeni przeszukiwania \mathcal{S} wynosi $|\mathcal{S}| = 2^{100} \approx 10^{30}$. To ogromna liczba! Wypróbowywanie wszystkich tych możliwości nie wchodzi w rachubę. Gdybyśmy dysponowali komputerem mogącym przetestować 1000 ciągów znaków na sekundę i zdołali zacząć go używać na początku czasu, podczas Wielkiego Wybuchu 15 miliardów lat temu, przetestowalibyśmy do tej pory mniej niż 1% wszystkich możliwości!

Co więcej, nie jest jasne, jakiej funkcji oceny użyć. Chcielibyśmy, żeby funkcja ta dała nam jakieś wskazówki co do jakości proponowanego rozwiązania. Rozwiązania bliższe właściwej odpowiedzi powinny uzyskiwać lepsze oceny niż rozwiązania dalsze. Ale w tym wypadku jedyne co mamy to $F(\mathbf{x})$, która daje tylko TRUE lub FALSE. Jeśli wybierzymy ciąg \mathbf{x} i $F(\mathbf{x})$ da TRUE, to w porządku – mamy odpowiedź. Ale co jeśli $F(\mathbf{x})$ da FALSE? Co wtedy? Co więcej, prawie każdy możliwy ciąg zer i jedynek, który możemy wypróbować, prawdopodobnie da FALSE, jak więc rozpoznamy, które z potencjalnych rozwiązań są „lepsze”, a które „gorsze”? Jeśli skorzystalibyśmy z przeszukiwania wyczerpującego, byłoby nam wszystko jedno, gdyż po prostu przechodzilibyśmy przez możliwe rozwiązania do momentu znalezienia czegoś interesującego. Jeśli jednak chcemy, żeby funkcja oceny pomogła nam znaleźć najlepsze rozwiązania szybciej niż przeszukiwanie wyczerpujące, potrzebujemy czegoś więcej niż tylko odpowiedzi „dobrze” czy „źle”. Na pierwszy rzut oka nie ma sposobu na uzyskanie takiej ulepszonej funkcji oceny dla problemu SAT.

Niektóre problemy są łatwiejsze od problemu SAT, gdyż w sposób naturalny sugerują możliwą funkcję oceny. Nawet wtedy jednak przestrzeń przeszukiwania może wciąż być ogromna. Weźmy na przykład problem komiwojażera (ang. *travelling salesman problem* – TSP). Pomyśl jest bardzo prosty: komiwojażer musi odwiedzić każde miasto na swoim terenie tylko raz, a następnie powrócić do domu, przemierzając jak najkrótszą drogę. W niektórych podobnych problemach założono nieco odmienne kryteria, takie jak opracowanie trasy między miastami, która zajmie jak najmniej czasu lub zminimalizuje koszty paliwa, ale pomysł jest ten sam. Znając koszt podróży między każdą parą

miast, jak komiwojażer powinien zaplanować swoją trasę, żeby ponieść najmniejsze koszty?

Na rysunku 1.1 przedstawiono prosty, symetryczny TSP złożony z 20 miast, w którym odległość między każdą parą miast i i j jest taka sama w obu kierunkach. Oznacza to, że $dist(i, j) = dist(j, i)$. Rzeczywiste odległości nie są zaznaczone na rysunku, ale możemy założyć, że tak właśnie jest. Alternatywą byłoby rozważenie asymetrycznego TSP, w którym $dist(i, j) \neq dist(j, i)$ dla niektórych i oraz j . Te dwa typy TSP stwarzają odmienne problemy przy poszukiwaniu tras o minimalnym koszcie.



Rysunek 1.1. Przykładowy TSP. TSP podane w książkach z reguły dopuszczają ścieżki z każdego miasta do każdego innego miasta, ale rzeczywiste problemy nie zawsze oferują takie możliwości

A więc jaka jest przestrzeń przeszukiwania dla naszego TSP? Jedną z możliwości może być postrzeganie jej jako zbioru permutacji n miast. Dowolna permutacja n miast dostarcza uporządkowaną listę, która definiuje kolejność miast do odwiedzenia, rozpoczynając od domu komiwojażera i przechodząc przez wszystkie miasta, żeby powrócić do punktu wyjścia. Optymalnym rozwiązaniem jest permutacja, która daje trasę o minimalnym koszcie. Zauważmy, że takie trasy, jak:

$$\begin{aligned} & 2 - \dots - 6 - 15 - 3 - 11 - 19 - 17 \\ & 15 - 3 - 11 - 19 - 17 - 2 - \dots - 6 \\ & 3 - 11 - 19 - 17 - 2 - \dots - 6 - 15 \text{ itd.} \end{aligned}$$

są identyczne, ponieważ droga, jaką trzeba przebyć podczas każdej z nich, jest taka sama, niezależnie od początkowego miasta. Takich podróży jest $n!$ dla dowolnego TSP o liczbie miast n . Łatwo też można zauważyć, że każda trasa może być przedstawiona na $2n$ różnych sposobów (dla symetrycznego TSP). Skoro zaś istnieje $n!$ sposobów, żeby permutować n liczb, wielkość przestrzeni przeszukiwania wynosi $|\mathcal{S}| = n!/(2n) = (n-1)!/2$.

To znów ogromna liczba! Dla dowolnego $n > 6$ liczba możliwych rozwiązań dla TSP z n miastami jest większa niż liczba możliwych rozwiązań problemu SAT z n zmiennymi. Co więcej, różnica między wielkością tych dwóch przestrzeni przeszukiwania wzrasta bardzo szybko wraz ze zwiększającym się n . Dla $n = 6$ istnieje $5!/2 = 60$ różnych rozwiązań TSP i $2^6 = 64$ rozwiązań SAT. Ale już dla $n = 7$ liczby te wynoszą odpowiednio 360 i 128.

Aby zobaczyć szalone tempo wzrostu $(n - 1)!/2$, prześledź następujące liczby:

Na Ziemi jest tylko 1 000 000 000 000 000 000 litrów wody, TSP z 50 miastami ma więc niewyobrażalnie wielką przestrzeń przeszukiwania. Jest ona dosłownie tak wielka, że jako ludzie po prostu nie potrafimy wyobrazić sobie zbioru z tyloma elementami.

Chociaż TSP ma niewiarygodnie wielką przestrzeń przeszukiwania, funkcja oceny, której możemy użyć do oceny jakości dowolnej konkretnej trasy przez miasta, jest dużo bardziej rzetelna od tej, którą widzieliśmy dla SAT. Możemy w tym wypadku skorzystać z tabeli zawierającej wszystkie odległości pomiędzy każdą parą miast i, po wykonaniu n dodawań, obliczyć długość dowolnej testowanej trasy, a następnie użyć uzyskane wyniki do oceny zalet tych tras. Na przykład, koszt trasy

$$15 - 3 - 11 - 19 - 17 - 2 - \dots - 6$$

wynosi

$$cost = dist(15, 3) + dist(3, 11) + dist(11, 19) + \dots + dist(6, 15)$$

Możemy mieć nadzieję, że ta bardziej naturalna funkcja oceny byłaby lepsza w poszukiwaniu użytecznych rozwiązań TSP, niezależnie od wielkości przestrzeni przeszukiwania.

Rozważmy trzeci przykład – konkretny problem programowania nielinowego (NLP). Jest to trudne zadanie omawiane w fachowej literaturze, a żadna tradycyjna metoda optymalizacyjna nie przyniosła dotychczas satysfakcjonującego rozwiązania. Zaprezentowano go w [254]. Polega on na maksymalizowaniu funkcji¹

$$G2(\mathbf{x}) = \left| \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n i x_i^2}} \right|$$

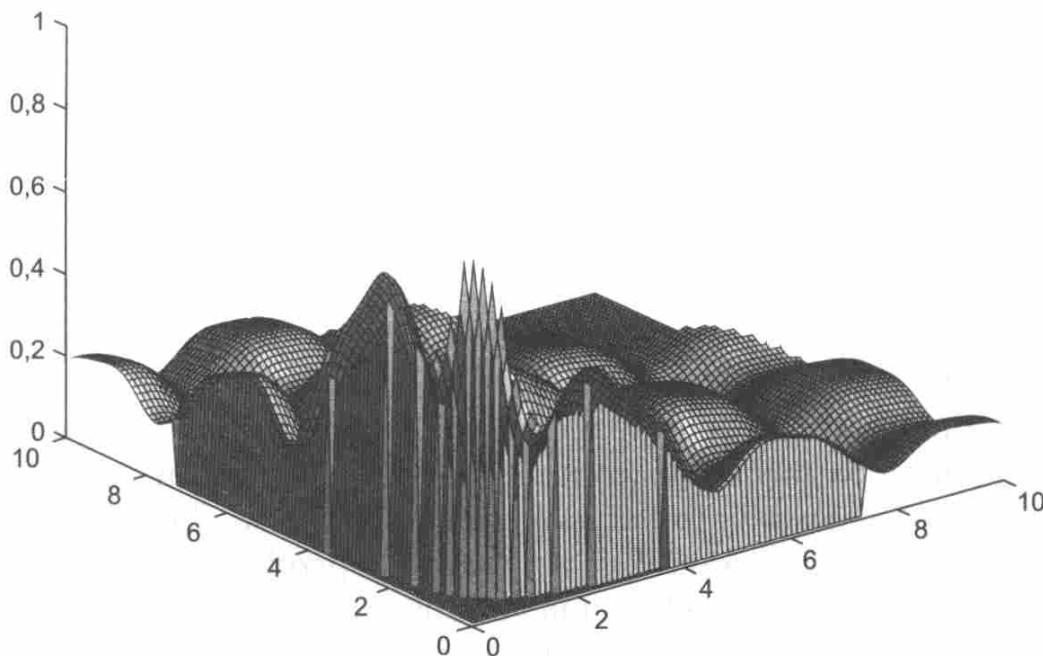
¹Utrzymujemy notację G_2 użytą dla tej funkcji w [319]. Jest to druga funkcja w złożonym z 11 funkcji zestawie testowym dla problemów programowania nieliniowego.

przy ograniczeniach:

$$\prod_{i=1}^n x_i \geq 0,75, \quad \sum_{i=1}^n x_i \leq 7,5n \text{ i zakresach } 0 \leq x_i \leq 10 \text{ dla } 1 \leq i \leq n$$

Funkcja $G2$ jest nieliniowa, a jej globalne maksimum nie jest znane, ale leży w pobliżu środka układu współrzędnych. Problem optymalizacji stawia jedno nieliniowe ograniczenie i jedno liniowe ograniczenie (to ostatnie nie ma znaczenia w pobliżu środka układu współrzędnych).

Jaka jest w tym wypadku wielkość przestrzeni przeszukiwania? W pewnym sensie zależy to od rozmiaru problemu, czyli liczby zmiennych. Jeśli postraktujemy problem jak czysto matematyczny, z n zmiennymi, to każdy wymiar może zawierać nieskończoność wiele wartości, mamy więc nieskończoność wielką przestrzeni przeszukiwania. Ale na komputerze wszystko jest cyfrowe i skończone. Gdybyśmy więc zamierzali zaimplementować jakiś rodzaj algorytmu, żeby znaleźć optymalne $G2$, musielibyśmy wziąć pod uwagę dostępną dokładność obliczeniową. Gdyby nasza dokładność sięgała sześciu miejsc po przecinku, każda zmienna mogłaby przyjąć $10\ 000\ 000^n$ różnych wartości. W ten sposób wielkość przestrzeni przeszukiwania wyniosłaby $|\mathcal{S}| = 10\ 000\ 000^n = 10^{7n}$. Liczba ta jest dużo większa niż liczba rozwiązań TSP. Nawet dla $n = 50$ istnieje 10^{350} rozwiązań NLP, przy dokładności sięgającej jedynie sześciu miejsc po przecinku. Większość komputerów oferuje dwa razy większą dokładność.



Rysunek 1.2. Wykres funkcji $G2$ dla $n = 2$. Nieosiągalnym w praktyce rozwiązań przypisano wartość 0

A co z funkcją oceny? Jak możemy zmierzyć jakość alternatywnych rozwiązań? Jednym ze sposobów byłoby użycie funkcji $G2$ jako funkcji oceny. Te rozwiązania, które przyniosłyby większe wartości $G2$, byłyby postrzegane jako lepsze od dających wartości mniejsze. Są jednak pewne trudności, gdyż – jak pokazano na rys. 1.2 – w praktyce istnieją obszary nierozwiązywalne. Wszystkim tym trudno rozwiązywalnym punktom została przypisana wartość 0.

Interesująca granica między rozwiązywalnymi a praktycznie nieroziązywalnymi obszarami jest zdefiniowana równaniem $\prod_{i=1}^n x_i = 0,75$, a optymalne rozwiązanie leży na (lub blisko) tej granicy. Poszukiwanie granic rozwiązywalnej części przestrzeni przeszukiwania nie jest łatwe. Wymaga wyspecjalizowanych operatorów dostosowanych specjalnie do tego celu, dla tego konkretnego problemu. Stanowi to kolejny poziom trudności, którego nie obserwowaliśmy przy SAT czy TSP (chociaż w TSP, który nie dopuszczał połączeń między wszystkimi możliwymi parami miast, niektóre permutacje też byłyby w praktyce nieroziązywalne). Nawet bez tego dodatkowego utrudnienia widać wyraźnie, że problemy, które z początku wydają się proste, mogą stanowić nie lada wyzwanie z powodu liczby alternatywnych rozwiązań. Sposób opracowania metod oceny tych rozwiązań nie zawsze jest jasny.

1.2. Modelowanie problemu

Za każdym razem, gdy rozwiązujeśmy problem, musimy sobie uświadomić, że w rzeczywistości znajdujemy jedynie rozwiązanie *modelu* problemu. Wszystkie modele są uproszczeniem realnych sytuacji, inaczej byłoby bowiem tak skomplikowane i trudne w obsłudze jak świat rzeczywisty. Rozwiązywanie problemu składa się z dwóch oddzielnych, ogólnych etapów: 1) tworzenia modelu problemu i 2) użycia tego modelu do utworzenia rozwiązania:

Problem \Rightarrow Model \Rightarrow Rozwiązanie

Rozwiązanie jest jedynie rozwiązaniem w kategoriach modelu. Jeśli nasz model ma duży stopień dokładności, możemy mieć większą pewność, że nasze rozwiązanie będzie sensowne. I odwrotnie: jeśli model ma zbyt wiele niespełnionych założeń i odległych przybliżeń, rozwiązanie może nie być sensowne lub być wręcz bzdurne.

SAT, TSP i NLP to trzy kanoniczne postacie modeli, które mogą być stosowane w wielu różnych kontekstach. Wyobraźmy sobie na przykład fabrykę, która produkuje samochody w różnych kolorach, przy liczbie n wszystkich kolorów. Zadanie polega na opracowaniu optymalnego planu produkcji, który zminimalizowałby całkowity koszt malowania pojazdów. Zauważmy, że każda maszyna używana na linii produkcyjnej musi być przełączana z jednego koloru na drugi, a koszt takiego przełączenia (zwanego przejściem) zależy od dwóch zaangażowanych w proces kolorów. Koszt przełączenia z żółtego na czarny może wynosić 30 jednostek. Możemy je liczyć w dolarach, minutach lub w inny sensowny sposób. Koszt przełączenia z powrotem z czarnego na żółty może wynosić 80 jednostek¹. Koszt przejścia z żółtego na zielony może wynosić 35 jednostek itd. Zeby zminimalizować koszty, musimy znaleźć taką sekwencję przejść, która pokryje zapotrzebowanie produkcji co do liczby samochodów w każdym kolorze,

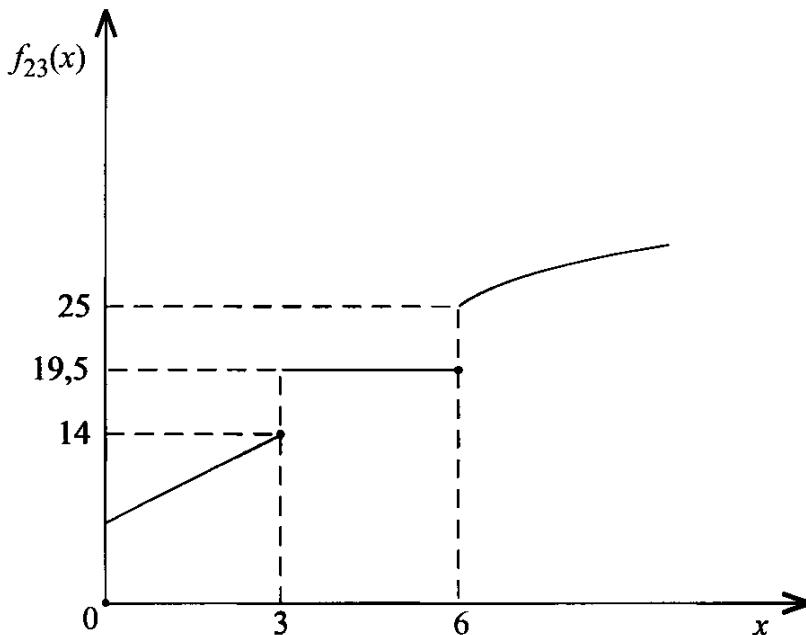
¹Zauważmy zaistniała asymetrię – koszty przejścia między dwoma kolorami nie muszą być takie same. Przejście z żółtego na czarny z reguły nie jest tak drogie jak odwrotne przejście.

w zgodzie z obecną modą, przy utrzymaniu jak najmniejszego kosztu przejść. Możemy na to spojrzeć z punktu widzenia TSP, gdzie każde miasto odpowiada teraz pomalowaniu konkretnego samochodu na dany kolor, a odległość między miastami odpowiada kosztowi przejścia. W tym wypadku TSP byłby asymetryczny.

Spójrzmy na następujący scenariusz ukazujący, jak uproszczenia są nieodłącznie związane z modelowaniem. Założymy, że firma ma n magazynów, w których przechowuje papier w ryzach. Zapasy te mają być dostarczone do k central dystrybucji. Magazyny i centrale dystrybucji mogą być postrzegane jako źródła i cele. Każda możliwa trasa między magazynem i a centralą dystrybucji j ma mierzalny koszt transportu, określony funkcją f_{ij} . Kształt tej funkcji zależy od wielu czynników, na przykład: odległość między magazynem a centralą dystrybucji, jakość drogi, natężenie ruchu, liczba wymaganych postojów, średnie ograniczenie prędkości itd. Na przykład, funkcja kosztu transportu między magazynem 2 a centralą dystrybucji 3 mogłaby zostać zdefiniowana następująco:

$$f_{23}(x) = \begin{cases} 0 & \text{jeśli } x = 0 \\ 4 + 3,33x & \text{jeśli } 0 < x \leq 3 \\ 19,5 & \text{jeśli } 3 < x \leq 6 \\ 0,5 + 10\sqrt{x} & \text{jeśli } x > 6 \end{cases}$$

przy czym x odnosi się do ilości towaru przewożonego z i do j (na rys. 1.3 przedstawiono wykres funkcji f_{23})¹.



Rysunek 1.3. Przykład funkcji kosztu transportu dla danego źródła i danego celu

Nawet jeśli funkcja ta wygląda trochę niekonwencjonalnie, uzasadnienie jej użycia w naszym modelu problemu transportu jest proste. Jeśli nie ma dostawy, koszt oczywiście wynosi zero. Jeśli transport nie przekracza trzech

¹Zauważmy, że punkty nieciągłości są typowe dla większości rzeczywistych funkcji kosztu transportu.

ryz papieru, możemy użyć specjalnego kontenera wysyłkowego. Daje to ogólny koszt czterech jednostek za kontener i dodatkowy koszt 3,33 jednostki na każdą ryżę. Tak więc koszt w tym wypadku rośnie liniowo. Jeśli jednak transport składa się z więcej niż trzech, ale mniej niż sześciu ryz, to możemy użyć specjalnego pudełka z drucianej siatki. Tym razem koszt jest stały i wynosi 19,5 jednostki, niezależnie od liczby przewożonych ryz. Wreszcie, jeśli transport zawiera więcej niż sześć ryz, trzeba użyć dużej, wzmacnionej skrzynki, a całkowity koszt transportu zależy teraz od liczby przewożonych ryz i rośnie jak pierwiastek kwadratowy z tej liczby powiększony o niewielki narzut równy 0,5 jednostki.

Przy takich założeniach możemy stworzyć model problemu

$$\text{zminimalizuj } \sum_{i=1}^n \sum_{j=1}^k f_{ij}(x_{ij})$$

przy ograniczeniach:

$$\begin{aligned} \sum_{j=1}^k x_{ij} &\leq sour(i) && \text{dla } i = 1, 2, \dots, n \\ \sum_{j=1}^n x_{ij} &\geq dest(j) && \text{dla } j = 1, 2, \dots, k \\ x_{ij} &\geq 0 && \text{dla } i = 1, 2, \dots, n \text{ oraz } j = 1, 2, \dots, k \end{aligned}$$

przy czym: *sour* oznacza źródło (ang. *source*), a *dest* – cel (ang. *destination*). Ograniczenia problemu definiują łatwo osiągalne rozwiązania: żaden transport z żadnego magazynu nie przekracza liczby ryz dostępnych w tym magazynie, a całkowity transport do dowolnej centrali dystrybucji musi zaspokajać jej zapotrzebowanie (tzn. całkowity transport musi być co najmniej równy liczbie zamówionych ryz).

Może się tak zdarzyć, że ta funkcja kosztu opisze wiernie rzeczywistą sytuację (tzn. dokładnie), pomijając koszty innych aspektów wymienionych wcześniej, takich jak natężenie ruchu między źródłem a celem itp. Możemy także utworzyć podobnie dokładne funkcje opisujące koszty transportu ryz papieru z każdego magazynu do każdej centrali dystrybucji. Wciąż jednak tak precyzyjny model problemu może mieć ograniczoną użyteczność, ponieważ funkcje te są zbyt skomplikowane dla wielu tradycyjnych algorytmów optymalizacyjnych, chociażby dlatego, że są nieciągłe, a nieciągłości powodują poważne problemy. Poza tym wyniki, które otrzymalibyśmy po zastosowaniu niektórych metod gradientowych dla tych funkcji, byłyby raczej kiepskie [320]. Dlatego nie możemy znaleźć rozwiązania opartego na tym modelu, tak więc model ten – skądinął idealny – jest bezużyteczny!

Jakie więc mamy możliwości? Istnieją przynajmniej dwa sposoby postępowania.

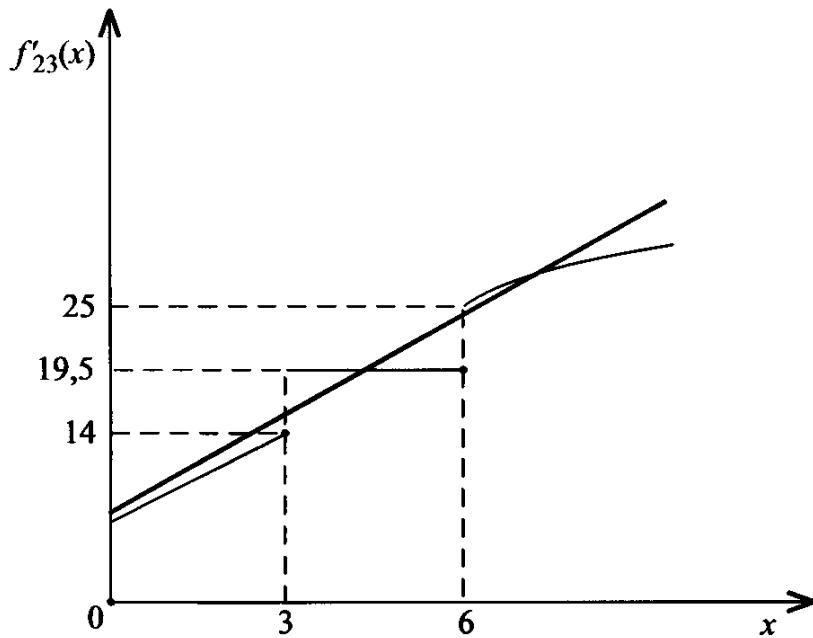
1. Możemy spróbować tak uprościć model, żeby tradycyjne metody optymalizacyjne mogły dawać lepsze odpowiedzi.

2. Możemy utrzymać posiadany model i zastosować nietradycyjne podejście do znalezienia rozwiązania bliskiego optymalnemu.

Pierwszy sposób jest kuszący. Możemy na przykład *przybliżyć* funkcję f_{23} w następujący sposób:

$$f'_{23}(x) = 2,66x + 8,25$$

przy czym x oznacza liczbę ryz przewożonych z 2 do 3 (na rys. 1.4 przedstawiono wykres przybliżonej funkcji f'_{23} razem z oryginalną funkcją f_{23}).



Rysunek 1.4. Przybliżenie funkcji kosztu transportu (pogrążona linia) dla danego źródła i danego celu

W tym wypadku uprościliśmy funkcję kosztu transportu f_{23} i możemy dokonać podobnych uproszczeń pozostałych funkcji. Zauważmy, że gdyby wszystkie funkcje f'_{ij} były liniowe, otrzymalibyśmy liniowy model problemu, który może być rozwiązywany dokładnie za pomocą metody programowania liniowego. Zauważmy jednak, że to dokładne rozwiązanie byłoby wtedy rozwiązaniem dla uproszczonego modelu, a nie dla rzeczywistego problemu!

Drugim sposobem postępowania jest utrzymanie posiadanego dokładnego modelu, ze wszystkimi jego nieciągłościami, i użycie niestandardowej metody (na przykład symulowanego wyżarzania lub algorytmu ewolucyjnego) do znalezienia rozwiązania bliskiego optymalnemu. Wiele eksperymentalnych dowodów pokazuje, że podejście to często może być z powodzeniem użyte.

Spójrzmy na to zagadnienie inaczej. Istnieją dwa możliwe podejścia. W pierwszym z nich używamy przybliżonego modelu problemu $Model_p$, a potem znajdujemy dokładne rozwiązanie dla tego przybliżonego modelu – $Rozwiazanie_d(Model_p)$:

$$\text{Problem} \Rightarrow Model_p \Rightarrow Rozwiazanie_d(Model_p)$$

W drugim podejściu używamy dokładnego modelu problemu – $Model_d$, a następnie znajdujemy przybliżone rozwiązanie dla tego dokładnego modelu –

Rozwiazanie_p(Model_d):

Problem \Rightarrow Model_d \Rightarrow Rozwiazanie_p(Model_d)

Z tych dwóch podejść drugie jest często lepsze, tzn. *Rozwiazanie_p(Model_d)* jest lepsze niż *Rozwiazanie_d(Model_p)* jako rozwiązanie oryginalnego problemu.

Jest też drugie źródło trudności, które napotykamy przy rozwiązywaniu problemów: trudno jest uzyskać dokładne rozwiązanie problemu, ponieważ musimy albo przybliżyć model, albo przybliżyć rozwiązanie.

1.3. Zmiana związana z czasem

Jakby powyższych trudności było mało, rzeczywiste problemy nastęczają jeszcze innego rodzaju trudności, a mianowicie ulegają zmianie. Zmieniają się, zanim stworzymy ich model, w trakcie poszukiwania rozwiązania i po tym, jak zrealizujemy ich rozwiązanie. Przyjrzyjmy się kilku źródłom tej trudności.

Wróćmy do problemu komiwojażera z rys. 1.1. Wyobraź sobie, że jesteś komiwojażerem i opuszczasz swój dom w mieście 1, wyruszając do miasta 6. Przemierzałeś tę trasę już wcześniej i wiesz, że odległość między tymi miastami wynosi, powiedzmy, 30 kilometrów. Ale czy wiesz, ile czasu zajmie Ci podróż między nimi? Niezupełnie. Możesz oszacować, że z reguły podrózujesz ze średnią prędkością 45 kilometrów na godzinę, a więc średnio podróż zajmuje Ci 40 minut. Jaka jest jednak szansa, że 40 minut będzie dokładnym czasem Twojej dzisiejszej podróży? Bardzo niewielka.

Czas podróży zależy od wielu czynników. Możesz na przykład być szczęściarzem i trafić po drodze na falę zielonych światła. Możesz z kolei nie mieć szczęścia i nie tylko trafić na czerwone światła, ale też utknąć za wolno jadącą ciężarówką. Co gorsza, możesz „złapać gumę”, co znacząco wydłużycie czas Twojej podróży. Wszystkie podane możliwości i niewyobrażalna liczba innych czynników, takich jak: pogoda, stan nawierzchni, wypadki drogowe, pojazdy uprzwilejowane, którym musisz ustąpić pierwszeństwa, pociąg, którego tory przecinają Twoją drogę, itp., mogą być opisane jako *szum* lub *losowe zakłócenia*. Zanim nie wyjedziesz, nie możesz przewidzieć, czy którykolwiek z tych czynników będzie miał wpływ na Twoją podróż. Jedyne, co możesz wiedzieć bądź oszacować, to prawdopodobieństwo ich wystąpienia i związane z nimi konsekwencje dla czasu Twojej podróży. Musisz jednak pogodzić się z faktem, że nigdy nie będziesz mógł przewidzieć każdej możliwości.

Przypuśćmy, że zdecydowałeś się obliczyć przewidywany czas podróży na podstawie prawdopodobieństwa wystąpienia znanych możliwych czynników i ich wpływu na czas podrózowania. Uprośćmy trochę sprawę. Powiedzmy, że są tylko dwie możliwości dla Twojej podróży: 1) wszystko idzie dobrze i udaje Ci się dotrzeć do miasta 6 w 40 minut lub 2) jedziesz za wolną ciężarówką i podróż zabiera Ci 60 minut. Ponadto założmy, że obie te możliwości są jednakowo prawdopodobne. W takim razie przewidywany czas podróży wynosi 50 minut.

Zauważ jednak, że żadna z opisanych możliwości nie trwa 50 minut. Jeśli użyjesz 50 minut jako przybliżonego czasu, jest pewne, że jest to wartość, która nigdy nie wystąpi w rzeczywistości. Możesz być pewny, że Twoja wartość będzie błędna. Można sobie łatwo wyobrazić, że gdybyś użył szeregu takich błędnych przybliżeń do określenia czasu podróży do każdego z miast na trasie, obliczając swoje błędy w miarę posuwania się naprzód, otrzymany przez Ciebie ostatecznie czas mógłby mocno różnić się od jakiegokolwiek możliwego czasu trwania podróży w rzeczywistości. W każdej decyzji, którą podejmujesz na podstawie tego uśrednionego czasu, nie bierzesz pod uwagę zmienności czasu, a to może mieć poważny wpływ na podejmowanie dobrych decyzji.

Przypadkowość wystąpienia danego czynnika nie jest jedynym źródłem zmiany w razie rzeczywistych problemów. Czasami napotykamy na czysto deterministyczne trudności. Wiesz na przykład, że w każdym mieście podróż w godzinach szczytu jest bardziej czasochłonna niż o północy. Możesz nie wiedzieć dokładnie, ile czasu Ci zajmie – to zależy od przypadku – ale istnieje przekonanie, że godziny szczytu zmniejszą Twoje tempo przemieszczania się. To przekonanie jest stałe i przewidywalne, musisz je więc uwzględnić, gdyż w przeciwnym razie Twój model nie będzie odpowiadał rzeczywistości, a oparte na nim rozwiązańokażą się bezużyteczne. W najgorszym wypadku konkretna trasa między dwoma miastami może być przejezdna tylko o określonych porach dnia (zdarza się to często w miastach, w których planiści ograniczyli możliwości skrętu w boczne ulice w godzinach szczytu, żeby zwiększyć płynność ruchu ulicznego). Postępowanie, jak gdyby niedostępna trasa wciąż była możliwa opcją, prowadziłoby do rozwiązań niedopuszczalnych, czyli w praktyce żadnych rozwiązań.

Ważne jest także upewnienie się, że model odzwierciedla aktualną wiedzę o problemie. Może się zdarzyć, że modernizacja drogi albo nowa autostrada między dwoma miastami z Twojej listy umożliwi Ci szybszy przejazd. Jeśli nie uaktualnisz swojego modelu tak, aby brał pod uwagę tę zmianę, będziesz generował rozwiązań problemu, który już nie istnieje.

Sytuacja jest jednak jeszcze bardziej skomplikowana. Powyższe dziwactwa mogą pojawić się jako funkcja środowiskowych zmian lub niekontrolowanych wydarzeń, ale żadne z nich nie było skierowane świadomie przeciwko Tobie. W rzeczywistym świecie często zdarza się, że inni ludzie próbują stropować Twoje rozwiązańe, a to wymaga od Ciebie ciągłego aktualizowania modelu z uwzględnieniem przewidywanych działań.

Wyobraźmy sobie na przykład, że jesteś właścicielem dużej sieci supermarketów. Musisz zdecydować, gdzie wybudować kolejny sklep, szacujesz więc koszt budowy w każdej możliwej lokalizacji przy określonej strukturze demograficznej okolicy, istniejącej konkurencji itp., sporządzasz funkcję oceny, która prawdopodobnie da problem programowania nieliniowego, i wreszcie decydujesz, które miejsce jest najlepsze. Jest jednak jeszcze jeden problem: w trakcie podejmowania przez Ciebie decyzji konkurencja przewiduje Twój wybór i planuje własny sklep. Próbuje aktywnie znaleźć najlepszą lokalizację, która zminimalizowałaby Twój zysk. Jeśli rozwiążesz problem wyboru najlepszego miejsca pod budowę tylko na podstawie aktualnych warunków, zachowasz się,

jak gdybyś był jedynym graczem w tej grze. Realny problem często ulega zmianie w trakcie opracowywania jego rozwiązania, a czasami zmienia się w sposób, który znacznie utrudnia Ci życie.

1.4. Ograniczenia

Problemy są często jeszcze bardziej skomplikowane, niż udało nam się do tej pory rozpoznać, gdyż nie obrazują nam wszystkich możliwości, które chcielibyśmy mieć. Prawie wszystkie praktyczne problemy mają ograniczenia, a jeśli je zignorujemy, nie będziemy mogli zrealizować rozwiązania. Wróćmy do NLP z podrozdziału 1.1. Mieliśmy tam sytuację, w której nie wystarczyło znalezienie maksimum funkcji G_2 , ale musielibyśmy się upewnić, że proponowane rozwiązanie znajduje się w obszarze rozwiązań dopuszczalnych, wyznaczonym przez ograniczenia dotyczące sumy i iloczynu (zob. rys. 1.2). Z początku możesz pomyśleć, że takie ograniczenie problemów ułatwia życie – w końcu mamy wtedy mniejszą przestrzeń przeszukiwania, a co za tym idzie mniejszą liczbę możliwości do rozważenia. To prawda, ale pamiętaj, że aby poszukiwać ulepszonych rozwiązań, musimy przechodzić od jednego rozwiązania do następnego. Potrzebujemy operatorów, które będą działały na rozwiązańach dopuszczalnych i w wyniku generowały nowe rozwiązania dopuszczone, będące ulepszeniem tych, które do tej pory uzyskaliśmy. W tym miejscu geometria przestrzeni przeszukiwania robi się trudna.

Wyobraźmy sobie na przykład, że stajemy przed problemem opracowania grafiku zajęć dla uczelni na jeden semestr. Pomyślmy, co to oznacza. Po pierwsze, musimy zrobić listę wszystkich proponowanych przedmiotów. Po drugie, musimy sporządzić listę wszystkich studentów przydzielonych do danych zajęć, nie zapominając o prowadzących je profesorach. Po trzecie, potrzebujemy listy dostępnych sal, z uwzględnieniem ich wielkości i wyposażenia (np. biała tablica, odtwarzacz wideo, sprzęt laboratoryjny itd.). A teraz co chcemy osiągnąć? Mamy trzy poważne ograniczenia.

- Każde zajęcia muszą być przypisane do dostępnej sali, w której jest wystarczająco dużo miejsc dla zapisanych studentów i odpowiedni dla danych zajęć sprzęt (np. laboratorium chemiczne musi być wyposażone w zlewki, pałniki Bunsena, potrzebne chemikalia, odpowiednie zabezpieczenia itp.).
- Studenci zapisani na więcej niż jedno zajęcie nie mogą mieć ich tego samego dnia o tej samej porze.
- Profesorowie nie mogą prowadzić zajęć, które pokrywają się w czasie.

Jak już powiedzieliśmy, są to poważne ograniczenia. Rozumiemy przez to, że muszą koniecznie zostać wzięte pod uwagę, żeby uzyskać sensowne rozwiązanie. Co więcej, w świetle dotychczasowych rozważań każdy plan zajęć uwzględniający te ograniczenia stanowiłby rozwiązanie problemu. Oznacza to, że zadanie to jest bardzo podobne do problemu SAT: musimy znaleźć taki plan

zajęć (w SAT znajdowaliśmy plan rozłożenia wartości logicznych), że całościowa funkcja oceny daje wartość TRUE. Najmniejsze pogwałcenie ograniczeń oznacza, że nasza funkcja oceny daje wartość FALSE. To jednak nie dostarcza nam wystarczających danych, żeby pomóc nam w poszukiwaniu dopuszczalnego rozwiązania.

Być może, możemy zastosować jakąś strategię, która dałaby nam te dodatkowe informacje. Możemy na przykład oceniać jakość rozwiązania nie tylko na podstawie tego, czy spełnia ograniczenia czy nie, ale w wypadku rozwiązań łamiących ograniczenia możemy sprawdzić, ile razy dane ograniczenie jest złamane (np. z każdym kolejnym studentem zapisanym na zajęcia odbywające się w tym samym czasie, zwiększa się liczba naruszeń odpowiedniego ograniczenia). Umożliwiłyby nam to ilościowe zmierzenie, jak złe były nasze niedopuszczalne rozwiązania i mogłyby pomóc w znalezieniu lepszych rozwiązań, minimalizując liczbę naruszeń ograniczeń. Moglibyśmy użyć innych operatorów do przepisania zajęć do innych sal, profesorów do zajęć itd. i z czasem liczylibyśmy na uzyskanie rozwiązania spełniającego posiadane ograniczenia.

Są jednak jeszcze *łagodne ograniczenia*, czyli takie, które chcielibyśmy zachować, ale które nie są obligatoryjne. Oto one.

- Idealnie byłoby, żeby zajęcia odbywające się dwa razy w tygodniu odbywały się w poniedziałki i środy lub wtorki i czwartki. Ich rozplanowanie w następujące po sobie dni lub z dwudniową albo większą przerwą nie jest wskazane.
- Zajęcia odbywające się trzy razy w tygodniu powinny zostać wyznaczone na poniedziałki, wtorki lub środy oraz czwartki. Nie powinny występować w kolejne następujące po sobie dni, nie powinno też być między nimi dwudniowej ani dłuższej przerwy.
- Zajęcia powinny być tak rozplanowane, żeby studenci nie musieli zdawać końcowych egzaminów z kilku kursów bez żadnej przerwy między nimi (egzamin końcowy najczęściej odbywa się o tej samej godzinie, o której były zajęcia).
- Jeśli licencjackie zajęcia, które są warunkiem uczestnictwa w zajęciach magisterskich, odbywają się tego samego dnia co ich magisterskie odpowiedniki, powinny odbywać się przed nimi (umożliwiłyby to poznawanie podstawowego materiału przed materiałem bardziej zaawansowanym w trakcie tego samego dnia).
- Jeśli więcej niż jedna sala spełnia wymogi dla danego kursu i jest dostępna w danym czasie, zajęcia powinny zostać przypisane do sali, której wielkość jest najbliższa wielkości grupy (zapobiegłoby to umieszczaniu małych grup w dużych pomieszczeniach, wzmacniając w ten sposób aktywny udział studentów w zajęciach).

Możemy znaleźć znacznie więcej tego rodzaju ograniczeń. Każdy plan zajęć, który spełni poważne ograniczenia, jest rozwiązaniem sensownym, ale niekoniecznie optymalnym w świetle łagodnych ograniczeń. Tu właśnie problem się komplikuje. Po pierwsze, musimy określić ilościowo w kategoriach matematycznych każde z łagodnych ograniczeń, żebyśmy mogli ocenić dowolną parę planów

zajęć i zdecydować, który z nich jest lepszy. Po drugie, musimy mieć możliwość zmodyfikowania jednego dopuszczalnego rozwiązania i uzyskanie następnego, które lepiej spełni łagodne ograniczenia.

Zastanówmy się nad pierwszym z tych zagadnień: każde łagodne ograniczenie musi zostać określone ilościowo. Rozważając pierwsze z nich możemy stwierdzić, że dla każdego dopuszczalnego rozwiązania moglibyśmy policzyć, ile razy zajęcia odbywające się dwa razy w tygodniu odbywają się co dwa lub więcej dni albo w następujące po sobie kolejno dni. Im mniejszą uzyskalibyśm wartość, tym lepsze jest rozwiązanie. Rzeczywiście moglibyśmy zastosować podobne podejście do każdego łagodnego ograniczenia. Ale co potem? Wciąż potrzebowalibyśmy całościowej metody do określenia stopnia naruszenia każdego z tych ograniczeń. Oznacza to, że musielibyśmy znaleźć odpowiedzi na pytania typu: co jest gorsze – plan, według którego pięciu studentów pisze dwa końcowe egzaminy z rzędu, czy taki plan, który przypisze dwa następujące po sobie egzaminy końcowe do sal znajdujących się na przeciwnieległych krańcach kampusu? Każdy z takich możliwych kompromisów musiałby zostać rozważony i określony ilościowo w ramach jakiejś funkcji oceny, co jest nie lada wyzwaniem!

Sprawy mają się jeszcze gorzej, gdyż nawet ilościowe określenie wszystkich łagodnych ograniczeń nie kończy procesu poszukiwania najlepszego planu zajęć, tj. rozwiązania, które będzie zarówno dopuszczalne, jak i zminimalizuje naszą funkcję oceny łagodnych ograniczeń. Wyobraźmy sobie, że znaleźliśmy dopuszczalne rozwiązanie, które jednak nie radzi sobie dobrze z łagodnymi ograniczeniami. Powiedzmy, że zastosujemy do niego operatory różnicowania i wyraźnie poprawiamy sytuację, jeśli chodzi o łagodne ograniczenia, ale w rezultacie otrzymujemy rozwiązanie, które narusza jedno z poważnych ograniczeń. I co teraz? Możemy odrzucić to rozwiązanie, ponieważ jest niedopuszczalne, lub możemy spróbować, czy nie da się go naprawić tak, żeby uzyskać rozwiązanie dopuszczalne, które poradzi sobie z łagodnymi ograniczeniami. Jest to z reguły trudna praca. Jeszcze lepiej byłoby opracować operatory różnicowania, które nigdy nie zmieniają rozwiązania dopuszczalnego w niedopuszczalne, ale jednocześnie intensywnie przeszukują przestrzeń rozwiązań dopuszczalnych w poszukiwaniu tych, które najlepiej spełniają łagodne ograniczenia. To wspaniałe pragnienie, ale często pozostaje jedynie nabożnym życzeniem. Efektywne radzenie sobie z rzeczywistymi problemami mającymi ograniczenia jest jednym z największych wyzwań, przed jakimi stajemy.

1.5. Problemy z udowadnianiem

Choć wydaje się to dziwne, w naszym dążeniu do rozwiązywania problemów często sami stwarzamy sobie trudności. Pracując ze studentami na wielu uniwersytetach i w wielu krajach, zaobserwowałyśmy następujące zjawisko: student poproszony o *znanie* rozwiązania problemu uzna to za łatwiejsze od prośby o *udowodnienie* czegoś związanego z rozwiązaniem, nawet jeśli oba te zadania są dokładnie takie same matematycznie.

Rozważmy matematyczny problem o średnim stopniu trudności. Zadanie powinno polegać na znalezieniu jakiejś wartości – może to być wysokość budynku, prędkość pojazdu albo czas potrzebny do wykonania pracy domowej. Cokolwiek to będzie, naszym celem powinno być znalezienie wartości x . Na przykład: napełnienie basenu dużym wężem zajmuje cztery godziny, a małym wężem – sześć. Jak dużo czasu zajmie napełnienie basenu obydwojma wężami? Jeśli problem jest sformułowany w kategoriach „znalezienia wartości x ” (jak dużo czasu zajmie napełnienie basenu obydwojma wężami?), jest to zadanie bardzo łatwe. Jeśli jednak nieco zmienimy to zadanie i spytamy: udowodnij, że napełnienie basenu obydwojma wężami zajmuje mniej czasu niż a , to mniejsza liczba studentów poradzi sobie z problemem, chociaż nie jest on ani trochę bardziej trudny. Jeśli możesz znaleźć czas potrzebny do napełnienia basenu i jest on mniejszy od stałej a , to dowód jest zakończony. Żeby to sprawdzić, weź podane wyżej wartości oznaczające czas napełnienia basenu za pomocą każdego z węzłów i przetestuj swoich kolegów. Poproś część z nich o znalezienie czasu potrzebnego do napełnienia basenu obydwojma wężami, a pozostałych o udowodnienie, że czas potrzebny do wykonania tej czynności jest mniejszy niż 2 godziny i 25 minut¹.

Naszym zdaniem, powodem niechęci do udowadniania jest fakt, że ludzie nie mają doświadczenia w tej kwestii i nie wiedzą, od czego zacząć. Uogólniając tę obserwację, możemy powiedzieć, że wiele problemów wydaje się trudnych tylko dlatego, gdyż trudna jest odpowiedź na pytanie: „Jak się do tego zabrać?”.

Oto przykład, który pozwoli Ci poćwiczyć umiejętności formułowania problemu i rozpoczęwanie poszukiwania rozwiązania.

Udowodnij, że dowolny wielościan musi mieć przynajmniej dwie ściany o tej samej liczbie krawędzi.

Niewątpliwie pierwszą myślą, jaka przychodzi Ci do głowy po przeczytaniu tego zadania jest: „Czy naprawdę muszę?”. Tak, musisz, lepiej więc zabieraj się do pracy.

Na wstępie warto sobie przypomnieć pewne podstawowe fakty dotyczące dowodzenia. Jednym ze sposobów udowadniania jest pokazanie, że wniosek wynika bezpośrednio z wiedzy zawartej w zadaniu. Założymy na przykład, że $a > b$ oraz $b > c$, a następnie udowodnijmy, że $a > c$. Cytujemy prawo przechodniości i problem jest rozwiązany. Innym sposobem udowadniania jest rozważenie kontrapozycji. Przypomnij sobie, że „jeśli p , to q ” jest logicznie równoważne „jeśli nie q , to nie p ”. Czasami może być łatwiej dowieść tego związku. Jeszcze innym sposobem jest założenie, że to, co się chce udowodnić, jest fałszywe, a następnie wykazać, że taka fałszywość jest niemożliwa. Ten sposób udowadniania nazywamy *dowodem przez sprzeczność*. Wypróbujmy go tutaj.

Przeformułując nasz problem, potrzebujemy zatem dowodu, który wykaże, że niemożliwe jest utworzenie wielościanu, którego wszystkie ściany miałyby różną liczbę krawędzi. Problem wydaje się trudny, ponieważ nie ma zbyt

¹ W rzeczywistości czas potrzebny do napełnienia basenu obydwojma wężami wynosi 2 godziny 24 minuty.

wielu ogólnych twierdzeń dotyczących wielościanów. Istnieje słynne twierdzenie Eulera, zgodnie z którym następujący wzór jest prawdziwy dla każdego wielościanu:

$$v + f = e + 2$$

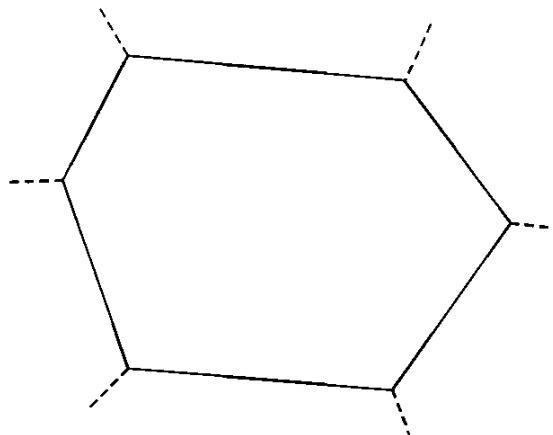
przy czym v , e i f oznaczają liczbę wierzchołków (ang. *vertices*), krawędzi (ang. *edges*) i ścian (ang. *faces*). Nie jest jednak jasne, w jaki sposób moglibyśmy użyć tego twierdzenia do udowodnienia, że wszystkie ściany muszą mieć różną liczbę krawędzi.

Pierwszy krok jest najtrudniejszą częścią zagadki. Nasz problem nie daje nam żadnego wygodnego punktu wyjścia. Nie podaje żadnej liczby, niczego, co można by rozłożyć na czynniki, podzielić przez dwa lub pomnożyć przez sześć, albo czegoś w tym rodzaju. W takiej sytuacji często pomaga wprowadzenie takiej liczby. Rozważmy wielościan z liczbą ścian f . W ten sposób zyskujemy coś – jakąś zmienną, z którą możemy pracować.

Musimy udowodnić coś w związku z krawędziami i ścianami tego wielościanu. Każda ściana ma jakąś liczbę krawędzi. Zeby powiedzieć coś o tej liczbie, dobrze byłoby przynajmniej znać zakres wartości, jakich możemy się spodziewać. Zapytajmy więc: jaka jest najmniejsza liczba krawędzi, którą może mieć ściana? Odpowiedź brzmi: trzy, i wówczas ściana jest trójkątem. Ta najmniejsza liczba jest niezależna od całkowitej liczby f ścian wielościanu.

A jaka jest największa liczba krawędzi, którą może mieć ściana? No cóż, każda krawędź należy dokładnie do dwóch ścian, jeśli więc mamy ścianę o sześciu krawędziach, to wiemy, że ściana ta jest częścią wielościanu o przynajmniej siedmiu ścianach (ta ściana i sześć nowych, po jednej dla każdej krawędzi; zob. rys. 1.5). Tak więc dowolna ściana wielościanu, który ma w sumie f ścian, nie może mieć więcej niż $f - 1$ krawędzi, ponieważ nowa ściana wychodzi z każdej krawędzi.

To zamyka dowód.



Rysunek 1.5. Ściana wielościanu o sześciu krawędziach

Jeśli Cię to dziwi, to zwróć uwagę, że straciłeś z oczu, co próbowałeś udowodnić. Cofnij się i przypomnij sobie problem. Oto powód, dla którego dowód jest zakończony: jeśli w sumie jest f ścian, a liczba krawędzi każdej ze ścian wynosi między 3 a $f - 1$, to jakieś powtórzenia liczby krawędzi muszą się

pojawić wśród f ścian. Ścian jest więcej niż możliwych liczb krawędzi, a więc niektóre z tych liczb muszą się powtarzać. Dlatego przynajmniej dwie ściany będą miały taką samą liczbę krawędzi.

Kluczem do rozwiązywania tego problemu było znalezienie punktu wyjścia i niestracenie z oczu celu.

1.6. Twoja szansa na sławę

Teraz, gdy już jesteś przygotowany do rozwiązywania problemów, masz szansę udowodnić, że umiesz to robić. Ten problem zaproponował nam Peter Ross z University of Edinburgh.

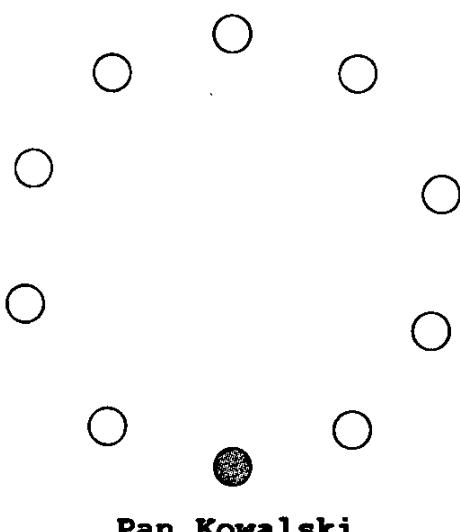
Pan Kowalski z żoną zaprosili na przyjęcie cztery pary. Gdy wszyscy przybyli, niektóre osoby wymieniły uściski dłoni. Oczywiście nikt nie witał się ze swoim współmałżonkiem ani nie witał się dwa razy z tą samą osobą.

Następnie pan Kowalski zapytał każdego z obecnych, ile razy uścisnął czyjaś dłoń, i od każdego uzyskał inną odpowiedź.

Ile razy pani Kowalska uścisnęła czyjaś dłoń?

Zachęcamy Cię do odłożenia książki na bok i samodzielnego sformułowania problemu. Spróbuj pomyśleć o punkcie wyjścia – może nim być graficzne przedstawienie problemu – i sprawdź, czy możesz dojść szczerze do rozwiązania.

Problem ten jest ciekawym wyzwaniem, ponieważ po raz kolejny nie mamy oczywistego punktu wyjścia, niemniej jednak naprawdę łatwo jest stworzyć graficzny model tego problemu (rys. 1.6).



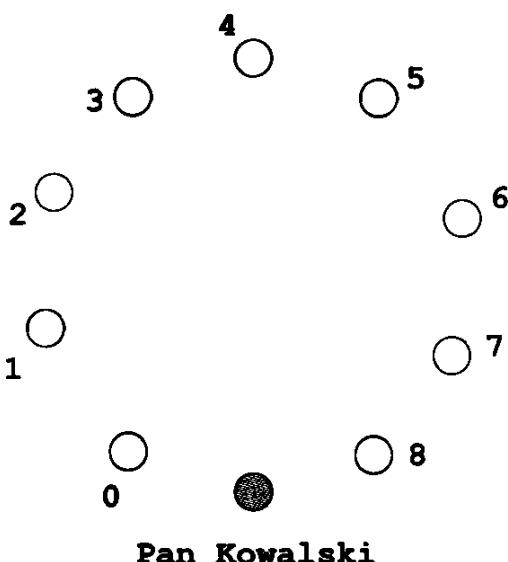
Rysunek 1.6. Pan Kowalski i pozostałe osoby

To jest bardzo dobry model, gdyż wyraźnie widzimy na nim pana Kowalskiego (ciemne kółko) i pozostałe dziewięć osób (w tym jego żonę). Jakie posiadamy informacje? (Pamiętaj, analizuj informacje, którymi dysponujesz).

Jedyną informacją podaną w zadaniu jest fakt, że pan Kowalski zapytał każdego z obecnych, ile razy dana osoba uścisnęła czyjaś dłoń, i że wszystkie otrzymane odpowiedzi były różne.

Jakie więc odpowiedzi otrzymał? Minimalną liczbą uścisków jest 0: możliwe, że jakaś nietwarzyska osoba nie wymieniła uścisku z nikim. Jaka jest jednak maksymalna liczba uścisków dla jednej osoby? Trudniej jest wpaść na zadanie tego pytania, niż znaleźć na nie odpowiedź. Maksymalną liczbą uścisków jest osiem, ponieważ mamy dziesięć osób w pokoju, a nie można wymienić uścisku ze sobą ani ze swoim współmałżonkiem.

Zbierzmy teraz wszystkie posiadane dane. Pan Kowalski zadał pytanie wszystkim obecnym w pokoju i wszystkie odpowiedzi były różne. Ponadto każda z nich była liczbą od 0 do 8. Wynika z tego, że uzyskane odpowiedzi brzmiały: zero, jeden, dwa, trzy, cztery, pięć, sześć, siedem i osiem (oczywiście niekoniecznie w tej kolejności!). Możemy więc odpowiednio zmodyfikować nasz model (rys. 1.7).

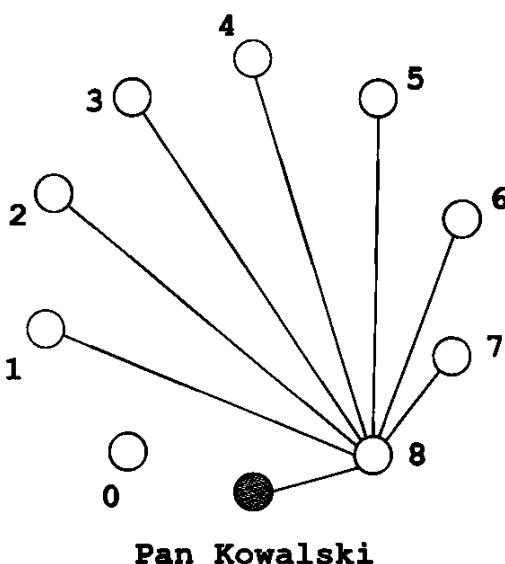


Rysunek 1.7. Pan Kowalski i pozostałe osoby. Podano liczbę uścisków dloni dla każdej z osób (poza panem Kowalskim)

A zatem co dalej? Co jeszcze możemy wywnioskować? Zdaje się, że nie wiele. Wykorzystaliśmy już fakt, że wszystkie odpowiedzi były różne i znamy już te odpowiedzi. Ale ile razy pani Kowalska uścisnęła czyjaś dłoń? I którą z osób na rys. 1.7 jest pani Kowalska? Gdybyśmy wiedzieli, kto wymieniał uściski z kim, wszystko byłoby prostsze.

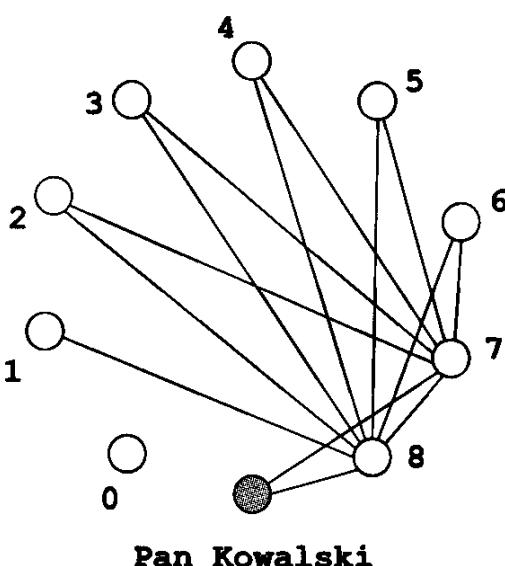
Ale czy nie możemy się dowiedzieć? Spróbujmy narysować wszystkie wymienione uściski dloni. Osoba 8 (nazwiemy każdą z osób, poza panią Kowalską, liczbą wymienionych przez daną osobę uścisków) przywitała się osiem razy, tzn. ze wszystkimi obecnymi poza sobą i swoim współmałżonkiem. Na podstawie tej obserwacji możemy zrobić dwie rzeczy: narysować wszystkie uściski wymienione przez osobę 8 (zob. rys. 1.8) i wywnioskować, że współmałżonkiem osoby 8 jest osoba 0.

Czy już świta Ci w głowie? Możemy teraz zająć się osobą 7. Wymieniała ona siedem uścisków dloni, tzn. ze wszystkimi obecnymi poza sobą, swoim



Rysunek 1.8. Pan Kowalski i pozostałe osoby. Zaznaczono wszystkie uściski dłoni wymienione przez osobę 8

współmałżonkiem i współmałżonkiem osoby 8. I znów możemy dodać powitania osoby 7 do naszego modelu (rys. 1.9). Możemy również wywnioskować, że współmałżonkiem osoby 7 jest osoba 1.



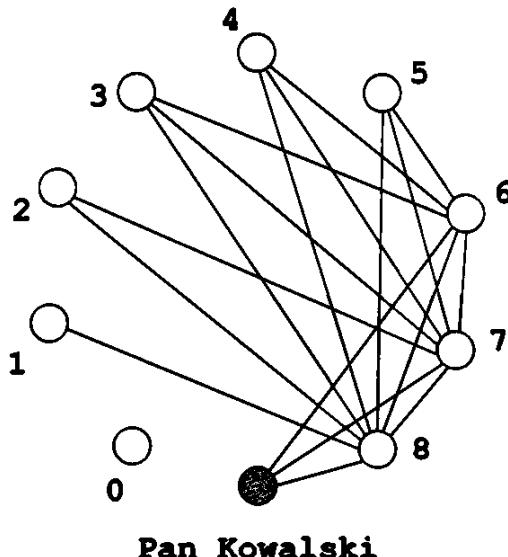
Rysunek 1.9. Pan Kowalski i pozostałe osoby. Zaznaczono wszystkie uściski dłoni wymienione przez osoby 8 i 7

Możemy powtórzyć to rozumowanie dla osób 6 i 5. Po narysowaniu wymienionych przez nie uścisków dłoni otrzymujemy interesujący obraz przedstawiony na rys. 1.10. Natychmiast nasuwają się wnioski:

- Współmałżonkiem osoby 6 jest osoba 2.
- Współmałżonkiem osoby 5 jest osoba 3.

Na podstawie tych wniosków wiemy, że współmałżonkiem pana Kowalskiego jest osoba 4. A zatem pani Kowalska uściśnięła czyjas dłoń cztery razy.

A Tobie jak poszło rozwiązywanie?



Rysunek 1.10. Pan Kowalski i pozostałe osoby. Zaznaczono wszystkie wymienione uściski dłoni

1.7. Podsumowanie

Rozwiązywanie problemów jest trudne z kilku powodów.

- Złożone problemy często mają olbrzymią liczbę możliwych rozwiązań.
- Żeby dojść do jakiegokolwiek rozwiązania, musimy często wprowadzić uproszczenia, które uczynią problem możliwym do rozwiązania. Użyte w rezultacie rozwiązania mogą być nieużyteczne.
- Warunki problemu zmieniają się w czasie, mogą na nie mieć też wpływ osoby, które chcą Twojego niepowodzenia.
- Rzeczywiste problemy często mają ograniczenia, które wymagają specjalnych operacji do generowania rozwiązań dopuszczalnych.

Ponadto rozwiązywanie problemów jest często trudniejsze, niż powinno być, ponieważ przeraża nas myśl o *udowodnieniu* czegoś.

- Nie pozwól, aby słowo *udowodnić* onieśmielało Cię.
- Pomyśl o różnych sposobach udowadniania poprawności rozwiązania. Czasami warto jest założyć, że to, co próbujesz udowodnić, jest fałszywe, a następnie wykazać, że taka fałszywość jest niemożliwa.
- Bądź gotów wykonać pierwszy krok, nawet jeśli problem zdaje się nie zawierać żadnych wskazówek, dokąd iść. Wymyśl coś – zmienną, która opisze jakiś aspekt problemu – i zobacz, czy to pomoże. Nie martw się, jeśli będziesz musiał zaczynać kilka razy od początku, zanim w końcu odkryjesz drogę, która poprowadzi Cię do rozwiązania.

Jednym ze sposobów ułatwiających wykonanie pierwszego kroku jest zrozumienie przestrzeni przeszukiwania: jakie mamy zmienne? Jakie są ich możliwe wartości? Jakie mamy ograniczenia? No i najważniejsze:

- Nie trać z oczu celu!

Jeśli zapomnisz, co próbujesz udowodnić, możesz równie dobrze pooglądać telewizję, gdyż szansa odniesienia sukcesu będzie wówczas taka sama: zero! Bądź ustawicznie skupiony na końcowym rezultacie. Zawsze pytaj siebie, czy działania, które podejmujesz, ułatwiają dotarcie tam, dokąd zmierzasz. Możesz nie być do końca pewny odpowiedzi, ale i tak nie przestawaj się pytać. W najgorszym razie pozwoli Ci to szybciej rozpoznać te drogi, które nie prowadzą do celu i zminimalizować ilość czasu spędzonego w ślepych zaułkach. W najlepszym przypadku zaś z łatwością rozpoznasz właściwą drogę i znajdziesz się na tropie rozwiązania.

II. Jak ważny jest model?

Zawsze gdy rozwiążujesz rzeczywisty problem, musisz najpierw stworzyć jego model. Bardzo istotne jest uświadomienie sobie, że utworzony model nie jest tym samym co nasz problem. Każdy model coś pomija. Taka jest konieczność – inaczej byłby tak skomplikowany i trudny w obsłudze jak rzeczywisty problem. Zawsze pracujemy na uproszczeniach realnych sytuacji i musimy to zaakceptować. Każde stworzone przez nas rozwiązanie jest, dokładnie rzecz ujmując, rozwiązaniem *jedynie dla modelu*, który uważamy za użyteczne odzwierciedlenie jakiejś prawdziwej sytuacji.

Kłopot z modelami polega na tym, że z każdym z nich jest związany jakiś zestaw założeń. Najczęstszym błędem podczas pracy z modelem jest zapominanie o tych założeniach. Wyobraźmy sobie na przykład, że rzucamy piłką i próbujemy sporządzić wykres trajektorii jej lotu podczas spadania na ziemię. Jaki jest kształt tej trajektorii? Najprostszą odpowiedzią jest: „paraboliczny”, ale to nieprawda. „O tak, zapomniałem o oporze powietrza” – możesz odpowiedzieć. To prawda, pomijamy go tutaj, ale to nie dlatego odpowiedź jest błędna. Nawet bez oporu powietrza trajektoria lotu piłki nie jest paraboliczna. Prawidłową odpowiedzią jest, że trajektoria jest eliptyczna. Dlaczego? Ponieważ jest paraboliczna jedynie przy założeniu, że ziemia jest płaska. Gdy założymy, że ziemia jest zakrzywiona, trajektoria przestaje być paraboliczna.

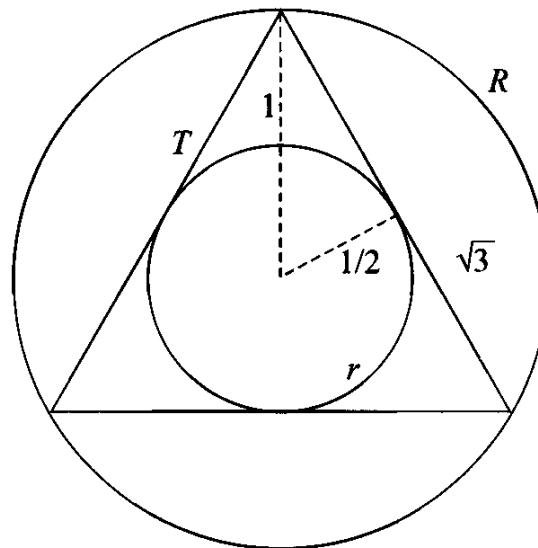
W skali lokalnej (tj. na odległość, na którą możesz rzucić piłką) ziemia wydaje się płaska. Dlatego nie ma większego znaczenia, czy rozwinieśmy model na bazie trajektorii parabolicznej czy eliptycznej. Żeby bardziej dosadnie zilustrować skutki przyjmowania niejawnych założeń, użyjmy prostego przykładu, który dostarczył nam Jan Plaza ze State University of New York w Plattsburghu.

Masz przed sobą na ziemi koło R o promieniu 1. Prosta L o nieskończonej długości (w obu kierunkach) jest poprowadzona na ziemi w taki sposób, że

przecina koło. Jakie jest prawdopodobieństwo p , że długość l cięciwy utworzonej przez przecięcie koła prostą jest równe lub większe niż $\sqrt{3}$?

Żeby obliczyć to prawdopodobieństwo, musimy wyobrazić sobie przestrzeń możliwych wyników, a następnie przypisać odpowiednie prawdopodobieństwo przypadkowi, który obejmuje wszystkie takie wyniki, że długość cięciwy jest większa lub równa $\sqrt{3}$.

Zanim zaczniemy rozważać kilka sposobów stworzenia przestrzeni wyników, przypomnijmy sobie pewne podstawowe fakty z geometrii: 1) długość boku trójkąta równobocznego T wpisanego w koło R wynosi $\sqrt{3}$ i 2) promień koła r wpisanego w trójkąt T wynosi $1/2$ (rys. II.1).



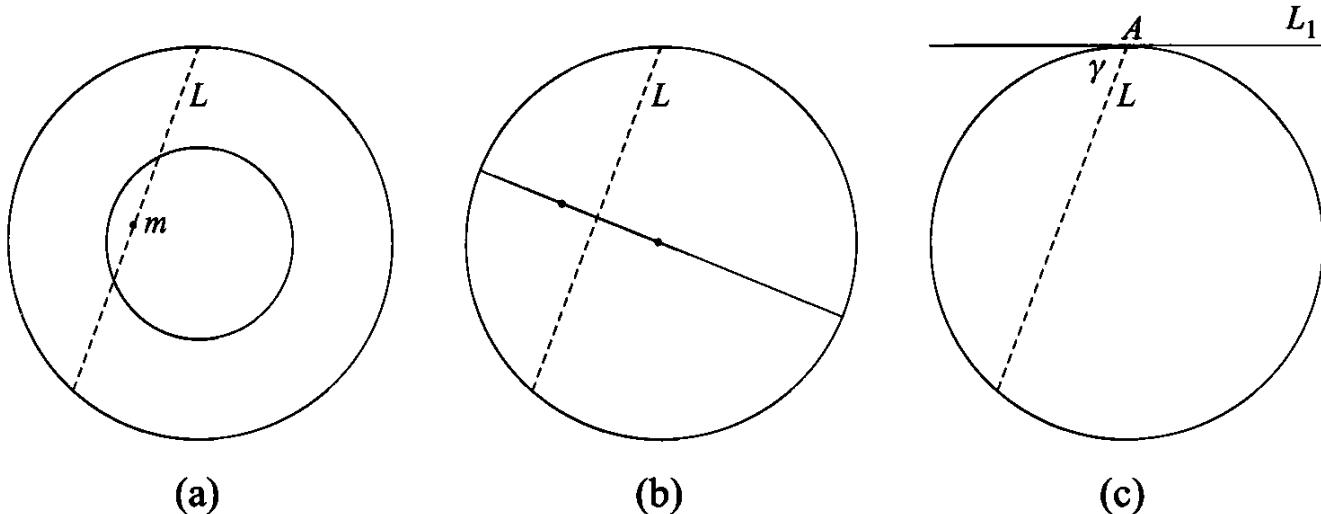
Rysunek II.1. Koło R , trójkąt równoboczny T i wpisane w niego koło r

Sposób 1. Wiemy, że długość l cięciwy może zmieniać się od 0 do 2, ponieważ średnica R wynosi 2. Rozważmy środek m cięciwy. Jeśli m znajduje się wewnątrz r , to $l \geq \sqrt{3}$, w przeciwnym razie m jest wewnątrz R , ale nie wewnątrz r , i $l < \sqrt{3}$ (rys. II.2a). Skoro m może znajdować się w dowolnym punkcie wewnątrz R , prawdopodobieństwo, że znajdzie się także wewnątrz r , z czego wynika, że $l \geq \sqrt{3}$, jest określone stosunkiem powierzchni r do R . Stąd

$$p = \frac{\pi(\frac{1}{2})^2}{\pi(1)^2} = \frac{1}{4}$$

Sposób 2. Wyznaczmy dowolną średnicę R . Po poprowadzeniu prostej L możemy obrócić R tak, żeby średnica była prostopadła do prostej, co pokazano na rys. II.2b. Teraz interesuje nas już tylko połowa koła, ponieważ długość cięciwy musi być mniejsza lub równa średnicy R . Środek cięciwy będzie odległy od środka R o nie więcej niż połowę albo o więcej niż połowę. W pierwszym przypadku długość cięciwy jest większa lub równa $\sqrt{3}$, a w drugim mniejsza niż $\sqrt{3}$. Tak więc prawdopodobieństwo p można określić na podstawie stosunku średnic r i R :

$$p = \frac{\frac{1}{2}}{1} = \frac{1}{2}$$



Rysunek II.2. Trzy interpretacje zadania z prostą

Sposób 3. Wybierzmy dowolny punkt A na okręgu R . Po poprowadzeniu prostej L możemy obrócić okrąg R tak, żeby A należało do L . Wykreślmy także prostą L_1 styczną do R i przechodzącą przez A (rys. II.2c). Z rysunku tego widzimy, że kąt γ między prostymi L i L_1 określa wynik zadania. Jeśli wartość γ wynosi między $\pi/3$ a $2\pi/3$, to środek L znajdzie się wewnątrz r , a więc l wyniesie nie mniej niż $\sqrt{3}$. Jeśli jednak $0 \leq \gamma < \pi/3$ lub $2\pi/3 < \gamma \leq \pi$, to l będzie mniejsze niż $\sqrt{3}$. Ponieważ istnieje π możliwych radianów dla γ , a wycinek $\pi/3$ generuje proste o długości $l \geq \sqrt{3}$, prawdopodobieństwo wynosi:

$$p = \frac{\frac{1}{3}\pi}{\pi} = \frac{1}{3}$$

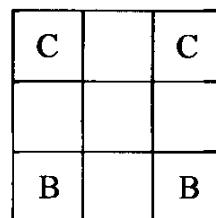
Który z tych sposobów jest prawidłowy? Wszystkie trzy wydają się sensowne i wszystkie wartości znalezione dla p mają sens! A jednak są różne! Jak to możliwe?

Wszystkie te możliwe rozwiązania są prawidłowe. Wszystko zależy od sposobu, w jaki zamodelujemy prowadzenie losowej prostej. Różnice w rezultacie powoduje fakt, że każdy model zawiera ukryte założenia co do sposobu prowadzenia prostej przez okrąg. Czy możemy je wydobyć na światło dzienne? Podamy Ci odpowiedź dla pierwszego sposobu: w założeniu podano, że środek prostej L może wypaść w dowolnym punkcie wewnątrz R , a żaden punkt ani obszar nie jest lepszy od innych. Jeśli uda Ci się odkryć założenia w przypadku drugiego i trzeciego sposobu, zobaczysz, dlaczego odpowiedzi na postawione pytanie są różne w każdym przypadku. Jeśli Ci się to nie uda, powinieneś nabrać szacunku dla modeli, ponieważ nawet w tak prostej sytuacji ukryte założenie może prowadzić do drastycznie różnych wyników. Pomyśl tylko, w jakie tarypaty możesz wpaść przy bardziej skomplikowanych problemach!

Zagadka z XVI wieku (ten problem wciąż jest nietrywialny!) również ilustruje, jak ważne jest posiadanie właściwego modelu.

Masz przed sobą małą szachownicę o wymiarach 3×3 pola (rys. II.3). Dwa czarne i dwa białe skoczki ustawiono w rogach szachownicy. Twoim zadaniem

jest zamienienie miejscami czarnych i białych skoczków przy jak najmniejszej liczbie ruchów. Skoczki muszą poruszać się w sposób, w jaki robią to w szachach.



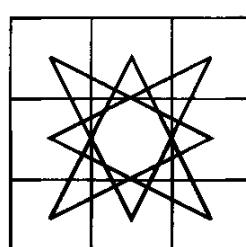
Rysunek II.3. Dwa czarne (C) i dwa białe (B) skoczki na dziewięciopolowej szachownicy

Ta zagadka świetnie się nadaje do rozwiązywania metodą „prób i błędów” – rzecz w tym, że natrafisz głównie na „błędy”. Jeśli jednak wyobrażysz sobie dobry model, wszystko okaże się proste.

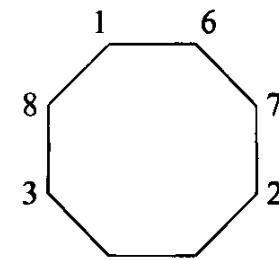
Jeśli przerysujemy szachownicę, numerując wszystkie pola liczbami od 1 do 9 (rys. II.4a), możemy wskazać wszystkie możliwe ruchy skoczka między tymi polami. Żeby skoczek obiegł szachownicę i wrócił na swoją wyjściową pozycję, trzeba wykonać konkretny ciąg ośmiu ruchów, rozpoczynając od górnego lewego pola, przez środkowe prawe, następnie (unikając cofania się) na dolne lewe, po czym górne środkowe itd. (rys. II.4b). Jeśli „rozplaczemy” tę drogę, żeby otrzymać zwyczajną pętlę (rys. II.4c), z łatwością dostrzeżemy, że każdy skoczek musi podążać w określonym kierunku i żaden z nich nie może minąć innego po drodze. Ponieważ dotarcie skoczka na daną pozycję wymaga 4 ruchów, całkowita liczba ruchów potrzebnych i wystarczających wynosi 16. Kolejność wykonywania ruchów jest również oczywista: wykonujemy po jednym ruchu każdym skoczkiem, dla każdego powtarzając sekwencję czterech ruchów.

1	2	3
4	5	6
7	8	9

(a)



(b)



(c)

Rysunek II.4. Szachownica z ponumerowanymi polami (a), przebieg (pętla) możliwych ruchów (b) i rozplotana pętla (c)

Istnieje jeszcze inna wersja tego problemu, w której dowolna liczba następujących po sobie ruchów tego samego pionka jest postrzegana jako jeden ruch. W tym wypadku potrzebnych jest siedem ruchów, żeby rozwiązać to zadanie. Czy umiesz je wykonać?

2. Podstawowe pojęcia

Rozwiązywanie problemu leży w jego zrozumieniu – odpowiedź nie znajduje się na zewnątrz, poza problemem, jest w nim samym.

Jiddu Krishnamurti: *The Penguin Krishnamurti Reader*¹

Trzy podstawowe pojęcia są wspólne dla wszystkich algorytmicznych metod rozwiązywania problemów. Niezależnie od zastosowanej przez Ciebie metody musisz określić: 1) reprezentację, 2) cel i 3) funkcję oceny. Reprezentacja koduje potencjalne rozwiązania, na których są wykonywane operacje, cel opisuje, co ma zostać osiągnięte, a funkcja oceny daje określoną wartość, która wyznacza jakość dowolnego rozwiązania przy danej reprezentacji (albo przynajmniej umożliwia porównanie jakości dwóch alternatywnych rozwiązań). Rozważmy kolejno każdy z tych aspektów rozwiązywania problemów.

2.1. Reprezentacja

Trzy problemy omówione w poprzednim rozdziale dają dobrą podstawę do przyjrzenia się różnym sposobom reprezentacji. Zaczynając od problemu spełnialności formuł boolowskich (SAT), gdzie mamy liczbę n zmiennych będących bitami logicznymi, oczywistym sposobem przedstawienia potencjalnego rozwiązania jest ciąg binarny o długości n . Każdy element w ciągu odpowiada zmiennej w problemie. Wielkość otrzymanej przy tej reprezentacji przestrzeni przeszukiwania wynosi 2^n , gdyż mamy 2^n różnych ciągów binarnych o długości n . Każdy punkt w tej przestrzeni przeszukiwania jest dopuszczalnym rozwiązaniem.

Dla problemu komiwojażera (TSP) z n miastami rozważyliśmy już jedną możliwą reprezentację: permutację liczb naturalnych $1, \dots, n$, przy czym każda liczba odpowiada miastu, które ma zostać odwiedzone. Przy tej reprezentacji przestrzeń przeszukiwania \mathcal{S} składa się ze wszystkich możliwych permutacji,

¹Za <http://www.unus.psychologia.uni.wroc.pl/teksty/ak/jiddu.html>, przekład Tadzimir (przyp. tłum.).

a jest ich $n!$. Jeśli jednak skupimy się na symetrycznym TSP, gdzie koszt podróży z miasta i do j jest taki sam w obu kierunkach, to nie ma znaczenia, czy będziemy się przemieszczać zgodnie z listą miast z lewa do prawa, czy z prawa do lewa, gdyż w obie strony trasa jest taka sama. Oznacza to, że możemy zmniejszyć przestrzeń przeszukiwania o połowę. Co więcej, droga będzie taka sama niezależnie od wyjściowego miasta, a to zmniejsza przestrzeń przeszukiwania o czynnik n . W rezultacie prawdziwy rozmiar przestrzeni przeszukiwania zmniejsza się do $(n - 1)!/2$. Widzimy jednak, że jest to ogromna liczba nawet dla umiarkowanie dużych problemów TSP.

W problemie programowania nieliniowego (NLP) przestrzeń przeszukiwania składa się ze wszystkich liczb rzeczywistych w n wymiarach. Korzystamy tu zwykle z reprezentacji zmiennopozycyjnej, umożliwiającej przybliżenie liczb rzeczywistych na komputerze z pojedynczą lub podwójną precyzją. W poprzednim rozdziale stwierdziliśmy, że przy sześciocyfrowej dokładności dla zmiennych w przedziale od 0 do 10 istnieje 10^{7n} różnych możliwych wartości.

W przypadku każdego problemu reprezentacja możliwego rozwiązania i odpowiadająca mu interpretacja implikuje przestrzeń przeszukiwania i jej wielkość. Ważne jest, żeby to sobie uświadomić, że wielkość przestrzeni przeszukiwania nie jest zdeterminowana przez problem, ale przez Twoją reprezentację i sposób, w jaki ją kodujesz.

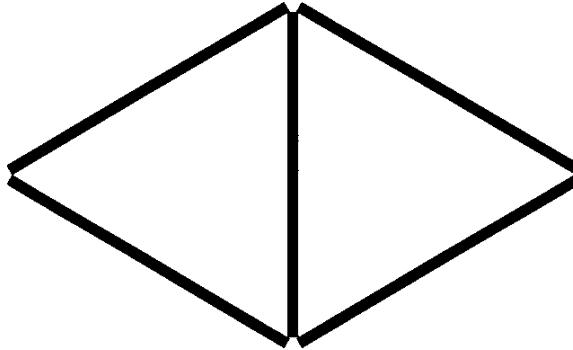
Wybranie właściwej przestrzeni przeszukiwania ma zasadnicze znaczenie. Jeśli nie zacznesz od wybrania właściwego obszaru do poszukiwań, to albo do dasz do listy możliwości liczne, niewykonalne lub powtarzające się rozwiązania, albo wręcz uniemożliwisz sobie znalezienie właściwego rozwiązania.

Oto dobry przykład: na stole leży sześć zapałek, a Twoje zadanie polega na ułożeniu z nich czterech trójkątów równobocznych o bokach równych długości zapałki. Łatwo jest ułożyć dwa takie trójkąty przy użyciu pięciu zapałek (rys. 2.1), ale trudno jest dodać kolejne dwa, zwłaszcza mając już tylko jedną zapałkę. W tym wypadku sformułowanie problemu jest zwodnicze, ponieważ sugeruje dwuwymiarową przestrzeń przeszukiwania (tzn. zapałki leżą na stole). Znalezienie rozwiązania wymaga przeniesienia się w przestrzeń trójwymiarową (rys. 2.2). Jeśli zacznesz od złej przestrzeni przeszukiwania, nigdy nie znajdziesz właściwego rozwiązania¹.

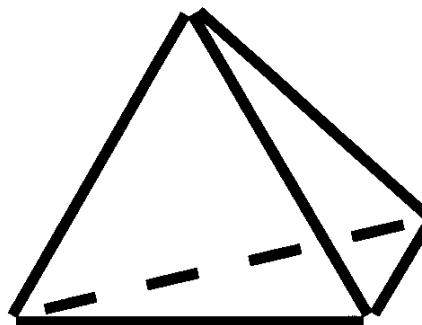
2.2. Cel

Po określeniu przestrzeni przeszukiwania musisz zdecydować, czego szukasz. Jaki jest cel Twojego problemu? To jest matematyczny opis zadania, które masz wykonać. Nie jest to funkcja, ale raczej wyrażenie. Weźmy na przykład TSP.

¹Kiedy dajesz komuś ten problem do rozwiązania, możesz poeksperymentować, dając mu dodatkowe „wskazówki” typu: „Wolno łamać zapałki na mniejsze kawałki”. Takie „wskazówki” są szkodliwe, ponieważ: 1) nie pomagają w dojściu do rozwiązania i 2) w znaczący sposób powiększają przestrzeń przeszukiwania. Nie eksperymentuj więc na przyjaciołach!



Rysunek 2.1. Dwa trójkąty; użyto pięciu zapałek, jedna wolna



Rysunek 2.2. Cztery trójkąty; użyto sześciu zapałek

Celem jest z reguły zminimalizowanie całkowitej drogi pokonanej przez komiwojażera, przy czym musi on odwiedzić każde z miast dokładnie raz oraz powrócić do miasta wyjścia. W kategoriach matematycznych cel ten przedstawia się następująco:

$$\min \sum dist(x, y)$$

W problemie SAT celem jest znalezienie takiego wektora bitów, że dane złożone wyrażenie boolowskie jest spełnione (tzn. daje wartość TRUE). W problemie NLP celem jest z reguły zminimalizowanie lub zmaksymalizowanie pewnej nieliniowej funkcji opisującej potencjalne rozwiązania.

2.3. Funkcja oceny

Cel nie jest jednak tym samym co funkcja oceny. Funkcja ta jest najczęściej odwzorowaniem z przestrzeni możliwych kandydatów na rozwiązania w obranej przez nas reprezentacji w zbiór liczb (np. liczb rzeczywistych). Przy odwzorowaniu tym każdemu elementowi przestrzeni możliwych rozwiązań (tj. przestrzeni przeszukiwania) zostaje przypisana wartość liczbową, która wskazuje jego jakość. Funkcja oceny umożliwia porównanie wartości alternatywnych rozwiązań Twojego problemu zgodnie z tym, jak go zamodelowałeś. Niektóre funkcje oceny umożliwiają rozpoznanie uszeregowania wszystkich możliwych rozwiązań. Noszą one nazwę *porządkujących* funkcji oceny. Alternatywą są inne funkcje

oceny zwane *numerycznymi*, które wskazują nie tylko kolejność rozwiązań ze względu na ich jakość, ale także stopień tej jakości.

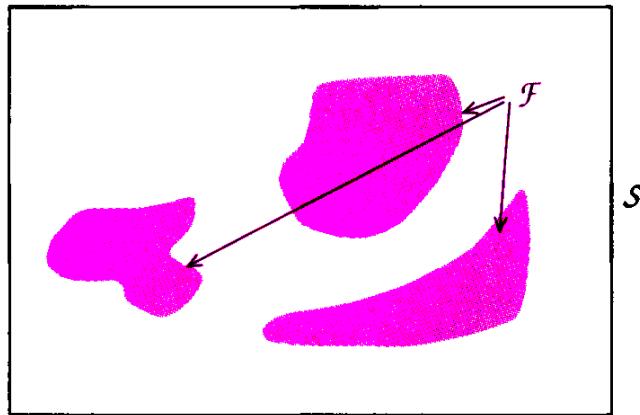
Przypuśćmy, na przykład, że chcielibyśmy znaleźć dobre rozwiązanie dla TSP. Celem jest zminimalizowanie sumy odległości między poszczególnymi miastami na trasie, przy jednoczesnym spełnieniu ograniczeń. Jedna z możliwych funkcji oceny mogłaby odwzorowywać każdą trasę na jej całkowitą długość. Następnie moglibyśmy porównać alternatywne trasy i nie tylko stwierdzić, która z nich jest lepsza, ale także dokładnie określić, o ile jest lepsza. Jednakże obliczenie dokładnej jakości konkretnego rozwiązania jest czasami złożone obliczeniowo i może wystarczyłoby wiedzieć w przybliżeniu, na ile dobre lub złe jest rozwiązanie albo czy wypada dobrze bądź nie w porównaniu z jakąś inną opcją. W takim wypadku funkcja oceny mogłaby sprawdzać tylko dwa potencjalne rozwiązania i wskazywać, które z nich jest preferowane.

W wypadku rzeczywistego problemu wybierasz funkcję oceny – nie jest Ci ona dana wraz z problemem. W jaki sposób powinieneś dokonać tego wyboru? Jednym z oczywistych kryteriów, które należy spełnić, jest fakt, że jeśli rozwiązanie w całości sprosta wymaganiom, powinno też mieć najlepszą ocenę. Funkcja oceny nie powinna wskazywać, które z rozwiązań niespełniających wymagań jest lepsze od tego, które je spełnia. Są to jednak dopiero podstawy do projektowania funkcji oceny.

Często cel sugeruje jakąś konkretną funkcję oceny. Powyżej, na przykład, rozważaliśmy użycie odległości w możliwym rozwiązań dla TSP jako funkcji oceny. Jest to zgodne z celem, który polega na zminimalizowaniu całkowitej przebytej odległości: funkcja oceny wynika bezpośrednio z celu. Podobnie jest często w przypadku NLP. W problemie polegającym na zmaksymalizowaniu funkcji przedstawionej na rys. 1.2 sama funkcja może posłużyć jako funkcja oceny, w której lepsze rozwiązania dają większe wartości. Nie zawsze jednak sensowną funkcję oceny daje się wyprowadzić z celu. W problemie SAT, gdzie celem jest spełnienie danego złożonego wyrażenia boolowskiego (tzn. sprawienie, żeby dało wartość TRUE), każde przybliżone rozwiązanie daje FALSE i nie przynosi Ci żadnej użytecznej informacji, w jaki sposób ulepszyć możliwe rozwiązanie, czy też jak szukać lepszej alternatywy. W takich wypadkach musimy być sprytniejsi i zastosować jakąś zastępczą funkcję oceny, która byłaby odpowiednia dla danego zadania, wybranej przez nas reprezentacji oraz operatorów, używanych przez nas, żeby przejść od jednego rozwiązania do następnego.

Projektując funkcję oceny, musisz także mieć na uwadze fakt, że w wypadku wielu problemów jedyne interesujące rozwiązania będą się znajdować w podzbiorze przestrzeni przeszukiwania \mathcal{S} . Interesują Cię jedynie rozwiązania *dopuszczalne*, to znaczy takie, które spełniają właściwe dla danego problemu ograniczenia. NLP z rysunku 1.2 jest doskonałym przykładem: interesują nas tylko takie zmiennopozycyjne wektory, że iloczyn ich składowych nie jest mniejszy niż 0,75, a ich suma nie przekracza iloczynu 7,5 i liczby zmiennych. Innymi słowy, przestrzeń przeszukiwania \mathcal{S} dla NLP możemy zdefiniować jako zbiór takich wszystkich zmiennopozycyjnych wektorów, że ich składowe znajdują się między 0 i 10. Tylko podzbiór tych wektorów spełnia dodatkowe ograniczenia

(dotyczące ich iloczynu i sumy składowych). Podzbiór ten jest dopuszczalną częścią przestrzeni przeszukiwania \mathcal{F} . W dalszej części książki zawsze będziemy robić rozróżnienie między przestrzenią przeszukiwania \mathcal{S} a dopuszczalną częścią przestrzeni przeszukiwania $\mathcal{F} \subseteq \mathcal{S}$ (rys. 2.3). Zauważ jednak, że dla SAT i TSP mamy $\mathcal{S} = \mathcal{F}$; wszystkie punkty z przestrzeni przeszukiwania są dopuszczalne (przy założeniu w TSP, że istnieje droga z każdego miasta do każdego miasta).



Rysunek 2.3. Przestrzeń przeszukiwania \mathcal{S} i jej część dopuszczalna \mathcal{F} . Zauważ, że obszary dopuszczalne mogą być niespójne

2.4. Określanie problemu poszukiwania

Możemy teraz określić problem poszukiwania¹. Mając przestrzeń przeszukiwania \mathcal{S} wraz z jej dopuszczalną częścią $\mathcal{F} \subseteq \mathcal{S}$, znajdź $x \in \mathcal{F}$ takie, że

$$\text{eval}(x) \leq \text{eval}(y)$$

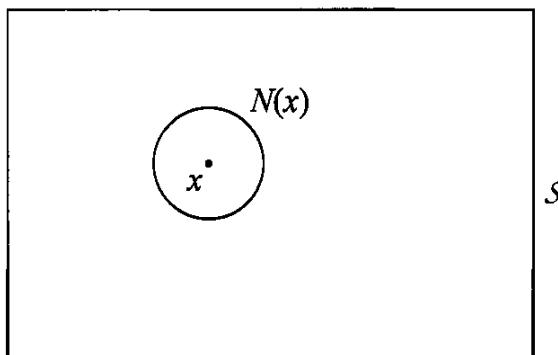
dla wszystkich $y \in \mathcal{F}$. Zauważ, że używamy tutaj funkcji oceny, dla której rozwiązania dające mniejsze wartości są uważane za lepsze (tzn. jest to problem minimalizacji), ale dla *każdego* problemu moglibyśmy równie dobrze użyć funkcji oceny, dla której są preferowane większe wartości, zmieniając problem poszukiwania w problem maksymalizacji. Ani problem, ani przybliżony model nie zawierają niczego, co wymagałoby konkretnie ustawienia funkcji oceny na minimalizację albo maksymalizację. Cel jako taki nie pojawia się nigdzie w określonym powyżej problemie poszukiwania. Takie postawienie problemu mogłoby z równym powodzeniem zostać użyte do opisania TSP, SAT czy NLP. Poszukiwanie nie wie, jaki problem rozwiążujesz. Wszystko, co „wie”, to informacje dostarczone mu przez funkcję oceny, używaną przez Ciebie reprezentację oraz sposób, w jaki wypróbowujesz możliwe rozwiązania. Jeśli Twoja funkcja oceny nie pasuje do celu, będziesz szukał dobrej odpowiedzi dla niewłaściwego problemu!

¹Pojęcia „problem poszukiwania” i „problem optymalizacyjny” są uważane za synonimiczne. Poszukiwanie najlepszego dopuszczalnego rozwiązania jest problemem optymalizacyjnym.

Punkt x , który spełnia powyższy warunek, nazywamy rozwiązaniem *globalnym*. Znalezienie takiego globalnego rozwiązania problemu może być bardzo trudne. Odnosi się to do wszystkich trzech omówionych dotychczas przykładów problemów (tzn. SAT, TSP i NLP). Czasami jednak znalezienie najlepszego rozwiązania jest łatwiejsze, jeśli możemy skupić się na relatywnie małym podzbiorze całej (dopuszczalnej lub także niedopuszczalnej) przestrzeni przeszukiwania. Mniejsza liczba możliwości może czasem ułatwić życie. Ta istotna obserwacja leży u podstaw wielu metod przeszukiwania.

2.5. Otoczenia i lokalne optima

Jeśli skupimy się na obszarze przestrzeni przeszukiwania, który znajduje się „blisko” jakiegoś konkretnego punktu w tej przestrzeni, możemy opisać to jako przyglądarkanie się *otoczeniu* (ang. *neighbourhood*) tego punktu. Rozważmy graficznie jakąś abstrakcyjną przestrzeń przeszukiwania \mathcal{S} (rys. 2.4) wraz z punktem $x \in \mathcal{S}$. Domyślamy się, że otoczeniem $N(x)$ jest zbiór wszystkich punktów przestrzeni przeszukiwania \mathcal{S} , które znajdują się w jakimś mierzalnym sensie blisko danego punktu x .



Rysunek 2.4. Przestrzeń przeszukiwania \mathcal{S} , możliwe rozwiązanie x i jego otoczenie $N(x)$ określone przez wnętrze koła wokół x

Możemy określić bliskość punktów w przestrzeni przeszukiwania na wiele różnych sposobów. Oto dwa z nich.

- Na przestrzeni przeszukiwania \mathcal{S} możemy określić funkcję odległości (ang. *distance function*) $dist$:

$$dist : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}$$

a następnie określić otoczenie $N(x)$ jako

$$N(x) = \{y \in \mathcal{S} : dist(x, y) \leq \epsilon\}$$

dla pewnego $\epsilon \geq 0$. Inaczej mówiąc, jeśli y spełnia warunek dla powyższego $N(x)$, to jest w epsilonowym otoczeniu x . Zauważ, że jeśli mamy

do czynienia z przestrzeniami przeszukiwania określonymi dla zmiennych ciągłych (np. NLP), naturalnym sposobem określenia otoczenia jest użycie odległości euklidesowej:

$$dist(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Podobnie jak w wypadku SAT, odległość między dwoma ciągami binarnymi możemy określić jako odległość Hamminga, tzn. liczbę pozycji, na których bity mają różne wartości logiczne. Uważaj jednak, gdyż wybór „naturalny” nie zawsze oznacza najlepszy w świetle innych opcji. Mimo to naturalne wybory często są dobrym punktem wyjścia.

- Na przestrzeni przeszukiwania \mathcal{S} możemy określić odwzorowanie m jako

$$m : \mathcal{S} \rightarrow 2^{\mathcal{S}}$$

przy czym odwzorowanie to określa otoczenie dla dowolnego punktu $x \in \mathcal{S}$. Możemy na przykład określić odwzorowanie *zamiany w parze* dla TSP, które tworzy nowy zbiór potencjalnych rozwiązań z danego potencjalnego rozwiązania x . Możemy powiedzieć, że każde rozwiązanie, które może być stworzone przez zamianę miejscami dwóch miast w wybranej trasie, leży w otoczeniu tej trasy, wyznaczonym przez operator zamiany w parze. Konkretnie, rozwiązanie x (permutacja $n = 20$ miast):

$$15 - 3 - 11 - 19 - 17 - 2 - \dots - 6$$

ma $n(n - 1)/2$ sąsiadów. Są wśród nich:

15 - 17 - 11 - 19 - 3 - 2 - ... - 6	(zamiana miejscami miast z drugiej i piątej pozycji)
2 - 3 - 11 - 19 - 17 - 15 - ... - 6	(zamiana miejscami miast z pierwszej i szóstej pozycji)
15 - 3 - 6 - 19 - 17 - 2 - ... - 11	(zamiana miejscami miast z trzeciej i ostatniej pozycji)

itd.

W przeciwieństwie do TSP, w SAT możemy określić odwzorowanie *jednego przełączenia*, które tworzy zbiór możliwych rozwiązań z dowolnego możliwego rozwiązania x . Są to wszystkie rozwiązania, które możemy otrzymać przez przełączenie 1 bitu w konkretnym wektorze binarnym i które znalazłyby się w otoczeniu x , wyznaczonym przez operator jednego przełączenia. Na przykład rozwiązanie x (ciąg binarny o – powiedzmy – $n = 20$ zmiennych):

01101010001000011111

ma n sąsiadów. Są wśród nich:

11101010001000011111 (przełączenie pierwszego bitu)
 00101010001000011111 (przełączenie drugiego bitu)
 01001010001000011111 (przełączenie trzeciego bitu)
 itd.

W tym wypadku otoczenie moglibyśmy także określić za pomocą funkcji odległości: otoczenie zawiera wszystkie ciągi w odległości Hamminga mniejszej lub równej jeden od x .

Mając zdefiniowane pojęcie „otoczenia”, możemy przejść do omówienia pojęcia *lokalnego optimum*. Możliwym rozwiązaniem $x \in \mathcal{F}$ jest lokalne optimum związane z otoczeniem N wtedy i tylko wtedy, gdy

$$\text{eval}(x) \leq \text{eval}(y)$$

dla wszystkich $y \in N(x)$ (ponownie przyjmując kryterium minimalizacji). Często, kiedy wielkość otoczenia jest bardzo mała lub gdy wiemy coś o pochodnej funkcji oceny (jeśli istnieje), dość łatwo jest sprawdzić, czy dane rozwiązanie jest lokalnym optimum, czy też nie.

Wiele metod poszukiwania jest opartych na statystykach zebranych z otoczenia danego punktu; to znaczy ciąg punktów generowany przez te metody podczas poszukiwania najlepszego możliwego rozwiązania zależy na każdym kroku po drodze od *lokalnej* informacji. Te metody są opracowane do lokalizowania takich rozwiązań w otoczeniu danego punktu, które dają lepsze wyniki. Odpowiednio do ich roli są one znane jako strategie przeszukiwania *otoczenia* albo strategie przeszukiwania *lokalnego* [476]. Przypuśćmy na przykład, że stajesz przed problemem maksymalizacji funkcji $f(x) = -x^2$ i zdecydowałeś się użyć samej $f(x)$ jako funkcji oceny. Lepsze rozwiązania dają większe wartości $f(x)$. Powiedzmy, że Twoje aktualnie najlepsze rozwiązanie x wynosi 2,0 i daje wartość -4,0. Możesz określić otoczenie epsilonowe w odległości 0,1 po obu stronach 2,0, a następnie szukać w tym przedziale. Gdybyś wziął nowy punkt w przedziale $[1,9; 2,1]$ i stwierdził, że ma on lepszą ocenę niż 2,0, wymieniłbyś 2,0 na to lepsze rozwiązanie i dalej szukał od tego miejsca. W przeciwnym razie odrzuciłbyś nowe rozwiązanie i wziął kolejny punkt z przedziału $[1,9; 2,1]$.

Większość funkcji oceny w rzeczywistych problemach nie wygląda jak funkcja kwadratowa, taka jak $f(x) = -x^2$. Sytuacja prawie zawsze jest bardziej skomplikowana. Funkcja oceny określa powierzchnię odpowiedzi, przypominającą swoją topografią góry i doliny (co pokazano na rys. 1.2), a problem znalezienia najlepszego rozwiązania przypomina poszukiwanie szczytu w paśmie górkim przy gęstej mgle. Możesz jedynie brać nowe punkty w swoim bezpośrednim sąsiedztwie i podejmować tylko lokalne decyzje, gdzie szukać dalej. Jeśli zawsze idziesz w górę, w końcu osiągniesz szczyt, ale może nie być to

najwyższy szczyt górskiego pasma. Może to być jedynie „lokalne optimum”. Może będziesz musiał schodzić w dół przez jakiś czas, żeby znaleźć pozycję, z której szereg lokalnych decyzji w obrębie kolejnych otoczeń doprowadzi Cię do najwyższego szczytu.

Metody przeszukiwania lokalnego (rozdział 3) przedstawiają interesujący kompromis między wielkością otoczenia $N(x)$ a efektywnością poszukiwania. Jeśli wielkość otoczenia jest dość mała, algorytm może szybko przeszukać całe otoczenie. Być może, tylko kilka możliwych rozwiązań trzeba będzie ocenić przed zdecydowaniem, które nowe rozwiązanie powinno być rozważone w następnej kolejności. Taki mały zakres widoczności zwiększa jednak szansę na utknięcie w lokalnym optimum! Sugeruje to używanie dużego otoczenia: większy zakres widoczności może pomóc w podejmowaniu lepszych decyzji. W szczególności, gdyby widoczność była nieograniczona (tzn. wielkość otoczenia była taka sama jak cała przestrzeń przeszukiwania), wówczas znaleźlibyśmy najlepszy sposób postępowania. Liczba ocen może jednak stać się ogromna, możemy więc nie być w stanie skończyć wymaganych obliczeń w czasie istnienia wszechświata (jak zauważono w rozdziale 1). Podczas stosowania metod przeszukiwania lokalnego odpowiednia wielkość otoczenia nie może być określana dowolnie – musi pasować do danego zadania.

2.6. Metody wspinania się

Przyjrzyjmy się teraz podstawowej metodzie *wspinania się* i jej związkom z pojęciem otoczenia. W metodach wspinania się, podobnie jak we wszystkich metodach przeszukiwania lokalnego (rozdział 3), stosujemy technikę iteracyjnego ulepszania rozwiązania¹. Ta technika jest stosowana dla jednego punktu – punktu bieżącego – w przestrzeni przeszukiwania. Podczas każdej iteracji z otoczenia punktu bieżącego jest wybierany nowy punkt. Jeśli ten nowy punkt daje lepszą wartość w świetle funkcji oceny, staje się punktem bieżącym. W przeciwnym razie wybieramy jakiś inny nowy punkt i porównujemy go z punktem bieżącym. Metoda kończy działanie, gdy niemożliwe jest dalsze ulepszenie albo gdy skończy się nam czas lub cierpliwość.

Widać wyraźnie, że takie metody wspinania się mogą dać jedynie wartości optymalne lokalnie, które zależą od wybranego punktu wyjściowego. Co więcej, nie ma żadnej ogólnej procedury umożliwiającej ograniczanie błędu względnego związanego z globalnym optimum, ponieważ nie jest ono znane. Mając problem, w którym poszukiwania prowadzą jedynie do lokalnie optymalnych rozwiązań, często musimy uruchamiać metody wspinania się w wielu różnych punktach. Mamy nadzieję, że przynajmniej niektóre z tych początkowych lokalizacji zaprowadzą nas do globalnego optimum. Możemy wybrać te początkowe punkty

¹Pojęcie *wspinania się* sugeruje problem maksymalizacji, ale możemy sobie wyobrazić odpowiednią metodę *zstępowania* dla problemu minimalizacji. Dla uproszczenia pojęcie wspinania się będzie stosowane do opisu obu metod, ale w taki sposób, że nie będzie to prowadziło do utraty ogólności.

losowo, na podstawie jakiejś siatki albo regularnego wzoru, a nawet na podstawie innych informacji dostępnych w wyniku wcześniejszych poszukiwań (np. opartych na próbach innych osób, które starały się rozwiązać dany problem).

Istnieje kilka wersji algorytmów wspinania się. Różnią się one głównie sposobem wybierania nowego rozwiązania w celu porównania z rozwiązaniem bieżącym. Jedną z wersji prostego iteracyjnego algorytmu wspinania się przedstawiono na rys. 2.5 (wspinanie się po najbardziej stromym zboczu). Na początku rozważani są wszyscy możliwi sąsiedzi bieżącego rozwiązania i ten \mathbf{v}_n , który da najlepszą wartość $eval(\mathbf{v}_n)$, zostaje wybrany do porównania z bieżącym ciągiem \mathbf{v}_c . Jeśli $eval(\mathbf{v}_c)$ jest gorsze niż $eval(\mathbf{v}_n)$, to nowy ciąg \mathbf{v}_n staje się ciągiem bieżącym. W przeciwnym razie nie jest możliwe żadne lokalne ulepszenie: algorytm osiągnął lokalne lub globalne optimum ($local = \text{TRUE}$). W takim wypadku następna iteracja ($t \leftarrow t + 1$) algorytmu jest dokonywana na nowym ciągu bieżącym wybranym losowo.

Sukces lub porażka jednej iteracji (tzn. jednej pełnej wspinaczki) powyższego algorytmu wspinania się zależy całkowicie od punktu wyjściowego. W problemach, które mają wiele lokalnych optimów, a zwłaszcza takich, w których optima mają duże *obszary przyciągania*, często bardzo trudno jest zlokalizować globalnie optymalne rozwiązanie.

```

procedure iteracyjne wspinanie się
begin
     $t \leftarrow 0$ 
    inicjuj best
    repeat
        local  $\leftarrow \text{FALSE}$ 
        wybierz losowo bieżący punkt  $\mathbf{v}_c$ 
        oceń  $\mathbf{v}_c$ 
        repeat
            wybierz wszystkie nowe punkty w otoczeniu  $\mathbf{v}_c$ 
            wybierz punkt  $\mathbf{v}_n$  ze zbioru nowych punktów
                o najlepszej wartości funkcji oceny eval
            if  $eval(\mathbf{v}_n)$  jest lepsze niż  $eval(\mathbf{v}_c)$ 
                then  $\mathbf{v}_c \leftarrow \mathbf{v}_n$ 
                else local  $\leftarrow \text{TRUE}$ 
            until local
             $t \leftarrow t + 1$ 
            if  $\mathbf{v}_c$  jest lepsze niż best
                then best  $\leftarrow \mathbf{v}_c$ 
            until  $t = MAX$ 
    end
```

Rysunek 2.5. Prosta procedura iteracyjnego wspinania się

Algorytmy wspinania się mają kilka wad.

- Z reguły kończą się na rozwiązańach, które są tylko lokalnie optymalne.
- Nie dostarczają żadnej informacji, na ile znalezione lokalne optimum odbiega od globalnego, a nawet innych lokalnych optimów.
- Osiągnięte optimum zależy od początkowej konfiguracji.
- Na ogólnie nie jest możliwe ograniczenie od góry czasu obliczeń.

Metody wspinania się mają jednak jedną kuszącą zaletę: są bardzo łatwe w użyciu! Wszystko, czego potrzebujemy, to reprezentacja, funkcja oceny i miara określająca otoczenie danego rozwiązania.

Efektywne metody poszukiwania mają mechanizm umożliwiający pogodzenie dwóch pozornie przeciwnych celów: *wykorzystania* najlepszego ze znalezionych dotychczas rozwiązań przy jednoczesnym *badaniu* przestrzeni przeszukiwania¹. Metody wspinania się wykorzystują najlepsze dostępne rozwiązanie do możliwego ulepszenia, ale zaniedbują badanie dużej części przestrzeni przeszukiwania \mathcal{S} . Działają przeciwnie do poszukiwania losowego, gdzie punkty są wybierane z \mathcal{S} z równym prawdopodobieństwem i które gruntownie bada przestrzeń przeszukiwania, ale nie wykorzystuje obiecujących obszarów przestrzeni. Każda przestrzeń przeszukiwania jest inna i nawet identyczne przestrzenie mogą wydawać się bardzo różne przy innych reprezentacjach i innych funkcjach oceny. Nie ma więc możliwości wybrania jednej metody poszukiwania, która sprawdzałaby się dobrze w każdym przypadku. Można to wręcz udowodnić matematycznie [499, 156].

Utknięcie w lokalnych optimach jest poważnym problemem. Jest to jedna z głównych wad przemysłowych aplikacji optymalizacji numerycznej. Prawie każde rozwiązanie rzeczywistego problemu dotyczącego planowania w zakładzie produkcyjnym, przewidywania popytu, zarządzania gruntami itp. w najlepszym wypadku jest tylko lokalnie optymalne.

Co możemy zrobić? Jak opracować algorytm, który potrafiłby wybrnąć z lokalnych optimów, godzić badanie i wykorzystywanie oraz uniezależniałby poszukiwanie od początkowej konfiguracji? Jest na to kilka sposobów, które omówimy w rozdziale 5, ale pamiętaj, że właściwe wybory zawsze zależą od problemu. Jedną z możliwości, którą omówiliśmy wcześniej, jest wykonanie wybranego algorytmu poszukiwania dla dużej liczby początkowych konfiguracji. Co więcej, często udaje się wykorzystać wyniki wcześniejszych prób do poprawienia wyboru początkowej konfiguracji dla następnej próby. Opłacalne może okazać się również użycie bardziej skomplikowanych środków do wygenerowania nowych rozwiązań lub powiększenie rozmiaru otoczenia. Możemy także dopuścić takie przejścia do nowych punktów, które odpowiadają *ujemnej* zmianie w funkcji oceny. Oznacza to, że możemy chcieć zaakceptować gorsze rozwiązanie z lokalnego otoczenia w nadziei, że doprowadzi nas ostatecznie do rozwiązania lepszego.

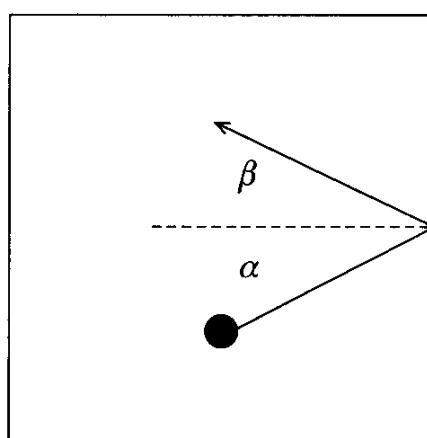
¹Tę równowagę między badaniem a wykorzystywaniem dostrzegł już w latach pięćdziesiątych XX wieku słynny statystyk G. E. P. Box [54].

Zanim jednak przejdziemy do omówienia bardziej wyrafinowanych metod poszukiwania, musimy przejrzeć podstawowe, klasyczne metody rozwiązywania problemów, takie jak programowanie dynamiczne, algorytm A^* i kilka innych. Wcześniej jeszcze otrzymasz kolejną szansę wykazania się sprytem, rozwiązuając interesujący problem geometryczny.

2.7. Czy umiesz tak sprytnie uderzyć bilę?

Teraz, gdy poznałeś już kilka różnych sposobów poszukiwania i podstawowe zagadnienia związane z wyborem reprezentacji, celu i funkcji oceny, zobaczymy, czy potrafisz rozwiązać tę trudną zagadkę. Pamiętaj, że możesz użyć informacji zawartych we wcześniejszych rozdziałach lub nie. Nic nie podpowiadamy!

Jedna bila leży nieruchomo na kwadratowym stole bilardowym. Zostaże uderzona i porusza się po stole w nieskończoność, zgodnie z podstawowym prawem fizyki: kąty α i β (rys. 2.6) są zawsze równe. Jakie warunki muszą być spełnione, żeby ruch bili był cykliczny?



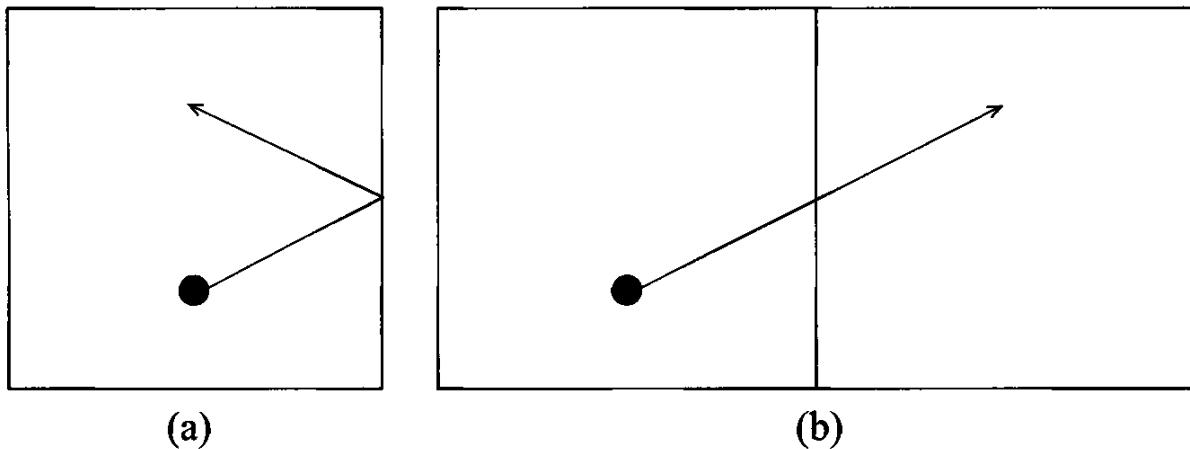
Rysunek 2.6. Kwadratowy stół bilardowy i ruch bili

Odłóż książkę na bok i pomyśl.

Jednym ze sposobów podejścia do tego problemu jest skupienie uwagi na powierzchni stołu bilardowego i próba rozumowania „wewnątrz” tego kwadratu. Trudno jednak zobaczyć tu jakikolwiek wzór ruchu bili, ponieważ liczba jej odbić może być bardzo duża (nieskończenie duża w granicy), a jej droga może składać się z wielu odcinków.

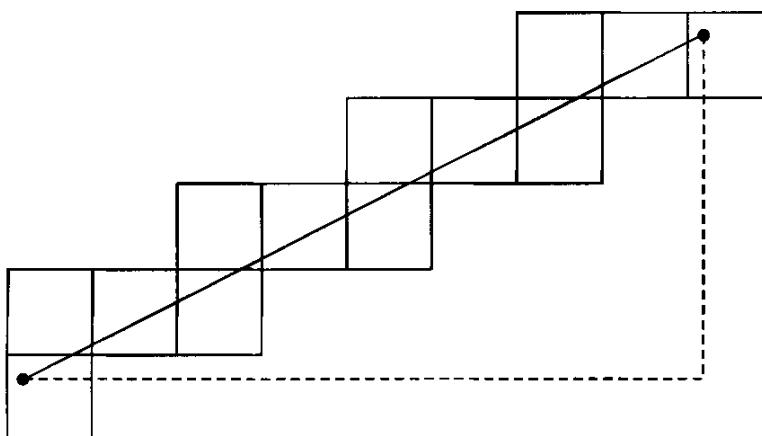
„Powiekszmy” obszar, na którym rozważamy nasz problem. Zamiast wyobrażać sobie bilę odbijającą się od każdej ze ścian stołu, wyobraźmy sobie bilę, jak gdyby poruszała się w linii prostej i rozważmy *odbicie stołu*! Na rysunku 2.7 przedstawiono ten pomysł.

Jest to znaczne odejście od sposobu, w jaki rozwiązujeśmy większość problemów. Z reguły rozważamy manipulowanie obiektami w danych ramach. Możliwa jest jednak zmiana podejścia i manipulacja ramami otaczającymi obiekty. Jeśli przeorientowałeś swoje myślenie bez oszukiwania i przeczytania o tym w naszej książce, gratulujemy – czeka Cię świetlana przyszłość!



Rysunek 2.7. Odbicie bili na stole bilardowym (a); odpowiadające mu odbicie stołu bilardowego (b)

Za każdym razem, gdy bila uderza w ścianę stołu, możemy narysować odbicie stołu względem tej ściany. Zauważ, że teraz bila może uderzać jedynie w górną lub prawą ścianę i będzie się poruszała ruchem cyklicznym, jeśli wróci na swoją początkową pozycję (zob. rys. 2.8).



Rysunek 2.8. Uzyskanie cyklu. Bila powraca do swojej początkowej pozycji na kolejnym odbiciu stołu bilardowego

Przypuśćmy, że bila porusza się ruchem cyklicznym. Oznacza to, że „przebyła” jakąś liczbę stołów „w górę” (powiedzmy p stołów) i jakąś liczbę stołów „w prawo” (powiedzmy q stołów). Dla kąta $\alpha = 0$, $p = 0$. Dla kąta $\alpha = \pi/2$, $q = 0$. Dla $0 < \alpha < \pi/2$, $p > 0$ i $q > 0$. Mając to na uwadze, możemy odpowiedzieć na zadane pytanie. Bila będzie poruszała się po stole ruchem cyklicznym wtedy i tylko wtedy, gdy albo $\alpha = \pi/2$, albo tangens kąta α będzie liczbą wymierną (tzn. $\operatorname{tg}(\alpha) = p/q$ dla $0 \leq \alpha < \pi/2$).

Pierwszy przypadek jest jasny: bila będzie poruszała się w górę i w dół w nieskończoność. Jeśli drugi przypadek jest niejasny, przypomnij sobie, że w trójkącie prostokątnym tangens kąta ostrego jest równy stosunkowi długości przeciwległej przyprostokątnej do długości przyległej przyprostokątnej. Popatrz na rys. 2.8 i wyobraź sobie drogę bili odpowiadającą przeciwprostokątnej trójkąta prostokątnego. Jego podstawa ma długość ośmiu stołów, a wysokość wynosi cztery stoły. Żeby osiągnąć identyczną pozycję na którymś z kolejnych odbić

od stołów, bila musi przebyć całkowitą, parzystą liczbę stołów w góre i w prawo (liczba stołów musi być parzysta, żeby zachować pierwotny kąt ruchu bili). Tak więc tangens kąta α musi być liczbą wymierną. Gdy α jest równe 0, bila porusza się poziomo w nieskończoność.

Do rozwiązania zadania wystarczy wiedza z trygonometrii z zakresu szkoły średniej. Trudną częścią jest uniknięcie wpadnięcia w pułapkę, jaką jest opis problemu, kuszący nas użyciem ram, które nie pomagają w znalezieniu rozwiązania. Ominięcie tej pułapki jest nie lada sztuką, ponieważ w realnym świecie biele się poruszają, a stoły bilardowe nie! W rozwiązywaniu problemów musisz jednak nauczyć się wychodzić poza schematy – albo poza granice bilardowego stołu.

2.8. Podsumowanie

Aby zaimplementować algorytm poszukiwania, musisz rozważyć

- reprezentację,
- cel,
- funkcję oceny.

Reprezentacja jest sposobem, w jaki komputer przechowuje potencjalne rozwiązania oraz obiekty, na których wykonuje operacje w poszukiwaniu nowych rozwiązań. Cel jest tym, co próbujesz osiągnąć. Funkcja oceny jest narzędziem do pomiaru jakości dowolnego możliwego rozwiązania problemu. Te trzy elementy razem wzięte stanowią strategiczną trójkę. Każdy z osobna musisz zawsze dokładnie rozważyć.

Czasami dobrze jest spojrzeć na przestrzeń przeszukiwania z punktu widzenia geometrii, oceniąc otoczenie danego rozwiązania i rozważając, czy możemy w tym otoczeniu znaleźć lepsze rozwiązanie. Jeśli nie możemy, to znaczy, że znaleźliśmy lokalne optimum. Może się zdarzyć, że rozwiązanie to jest w rzeczywistości globalnym optimum, tzn. niezależnie od tego, jak duże otoczenie wzięlibyśmy, rozwiązanie to wciąż byłoby najlepsze. Często prowadzenie poszukiwania poza małym otoczeniem jest zbyt kosztowne obliczeniowo. W rezultacie grozi nam niebezpieczeństwo utknięcia w lokalnych optimach, zwłaszcza jeśli stosujemy metodę wspinania się.

I ostatnia uwaga: gdy formułujesz problem, nie sugeruj się fizycznymi ograniczeniami realnego świata. Pozwól sobie stawiać pytania typu: co by było, gdyby stół bilardowy przemieszczał się względem bili? Co by się stało, gdybym opuścił powierzchnię stołu i zbudował z tych zapałek piramidę?

III. Jakie są ceny w sklepach „7–11”?

Istnieje wiele różnych sposobów poszukiwania rozwiązania problemu. Żeby się z tym uporać, zwłaszcza gdy przestrzeń możliwych rozwiązań jest duża, trzeba podejść do szukania systematycznie. Co więcej, struktura przeszukiwania musi „pasować” do struktury problemu. Jeśli tak nie jest, Twoje przeszukiwanie może dać gorszy wynik niż przypadkowe zgadywanie.

Niektóre heurystyki do przeszukiwania przestrzeni rozwiązań potrafią wycinać te rozwiązania, które nie mogą być prawidłowe. W ten sposób wielkość całkowitej przestrzeni zmniejsza się i przy odrobinie szczęścia możemy szybciej skupić się na dopuszczalnych rozwiązaniach. Pomyśl teraz, jak mógłbyś wyeliminować niemożliwe rozwiązania w tej kłopotliwej scence związanej z cenami, znanej pod nazwą problemu „7–11”. Opis jej brzmi następująco:

W Stanach Zjednoczonych jest sieć sklepów o nazwie 7–11. Prawdopodobnie nazywają się tak, ponieważ kiedyś były otwarte od siódmej rano do jedenastej wieczorem, ale teraz są czynne przez całą dobę. Pewnego dnia klient odwiedził jeden z tych sklepów i wybrał cztery artykuły. Następnie podszedł do kas, żeby zapłacić. Sprzedawca wziął kalkulator, nacisnął kilka przycisków i powiedział:

– Płaci pan 7,11 dolara.

Klient chciał zażartować:

– Dlaczego? Czy muszę zapłacić 7,11 dolara, bo sklep nazywa się 7–11?

Sprzedawca nie zrozumiał dowcipu i odpowiedział:

– Oczywiście, że nie! Pomnożyłem ceny tych czterech artykułów i podałem panu wynik!

Klient był bardzo zdziwiony.

– Dlaczego pan je pomnożył? Powinien pan dodać je, żeby otrzymać końcową należność!

Sprzedawca odparł:

– No tak, przepraszam! Strasznie boli mnie głowa i nacisnąłem niewłaściwy przycisk.

Następnie sprzedawca powtórzył obliczenie, tzn. dodał ceny wszystkich czterech artykułów, ale ku wielkiemu zdziwieniu obu mężczyzn suma wciąż wynosiła 7,11 dolara.

Zadanie polega na znalezieniu cen tych czterech artykułów!

Jest to typowy problem poszukiwania. Mamy cztery zmienne: x, y, z i t , oznaczające ceny czterech artykułów. Każda zmienna przyjmuje wartość ze swojej dziedziny, a w tym wypadku wszystkie dziedziny są takie same: od 0,01 do 7,08. Najniższa cena może wynosić 0,01, a skoro suma cen wszystkich czterech artykułów wynosi 7,11, najwyższą możliwą ceną jednego artykułu jest 7,08. Tak więc dziedziną każdej zmiennej jest zbiór

$$\{0,01, 0,02, 0,03, \dots, 7,06, 7,07, 7,08\}$$

Żeby uniknąć działania na ułamkach, zamieńmy je na liczby całkowite. Będziemy opisywać ceny w centach, a nie w dolarach i centach. W ten sposób domena każdej zmiennej staje się zbiorem liczb całkowitych

$$P = \{i \in N : 1 \leq i \leq 708\}$$

Problem polega teraz na znalezieniu takich x, y, z i t należących do zbioru P , że

$$x + y + z + t = 711$$

$$xyzt = 711\,000\,000$$

Zauważ, że podczas przeliczania dolarów i centów na same centy każda zmienna została pomnożona przez 100. Z tego powodu ich iloczyn zwiększył się 10^8 razy, dając $7,11 \cdot 10^8 = 711\,000\,000$.

Ponieważ $711\,000\,000 = 2^6 \cdot 3^2 \cdot 5^6 \cdot 79$, mamy wiele przypadków do rozważenia. Przeszukanie wyczerpujące wszystkich możliwych kombinacji dałoby nam rozwiązanie, trwałyby jednak pewien czas – ogromnie dużo czasu. Przyjrzymy się pewnej metodzie dojścia do wyniku na skróty, zblżonej do tak zwanej metody *podziału i ograniczeń* (ang. *branch and bound*).

Jednym z pomysłów, który możemy tu zastosować, zwłaszcza że mamy dwa równania, z których ostatnie jest iloczynem wyrażeń, jest rozkład na czynniki pierwsze. Możemy rozłożyć 711 000 000 na czynniki pierwsze. Może to być pomocne, ponieważ badane zmienne muszą być wielokrotnościami tych czynników pierwszych. 79 jest w takim razie największym pierwszym dzielnikiem 711 000 000. Okazuje się więc, że cena jednego z artykułów, powiedzmy artykułu x (oczywiście w centach), jest podzielna przez 79.

Rozważmy kilka możliwości, które dotyczą 79. Wybieramy 79, ponieważ z tego, że jest to największy pierwszy dzielnik 711 000 000 wynika, że będziemy mieli najmniejszą liczbę możliwości do rozważenia.

- Jeśli $x = 79$, problem dla pozostałych trzech zmiennych sprowadza się do:

$$\begin{aligned}y + z + t &= 632 \\yzt &= 2^6 \cdot 3^2 \cdot 5^6\end{aligned}$$

Niemôżliwe jest, żeby wszystkie zmienne y , z i t były podzielne przez 5 (ponieważ ich suma nie jest podzielna przez 5), tak więc iloczyn dwóch z tych zmiennych (powiedzmy y i z) jest podzielny przez $5^6 = 15\,625$. Mamy teraz kilka możliwości:

- Możliwość (a): $y = 15\,625y'$
- Możliwość (b): $y = 3125y'$ i $z = 5z'$
- Możliwość (c): $y = 625y'$ i $z = 25z'$
- Możliwość (d): $y = 125y'$ i $z = 125z'$

dla pewnych $y' \geq 1$ i $z' \geq 1$. W możliwościach (a)–(c) $y + z > 632$ (co jest niemożliwe), tak więc jedyny punkt, który musimy rozważyć, to możliwość (d). Problem sprowadza się w niej do:

$$\begin{aligned}125(y' + z') + t &= 632 \\y' \cdot z' \cdot t &= 2^6 \cdot 3^2\end{aligned}$$

W pierwszym równaniu zakładamy, że $y' + z' \leq 5$, a więc to, co należy teraz zrobić, to rozważyć sześć dodatkowych możliwości, ponieważ możemy założyć, że $y' \leq z'$:

- Możliwość (d1): $y' = 1$ i $z' = 1$
- Możliwość (d2): $y' = 1$ i $z' = 2$
- Możliwość (d3): $y' = 1$ i $z' = 3$
- Możliwość (d4): $y' = 1$ i $z' = 4$
- Możliwość (d5): $y' = 2$ i $z' = 2$
- Możliwość (d6): $y' = 2$ i $z' = 3$

Żadna z tych sześciu możliwości nie daje rozwiązań. W możliwości (d1) na przykład mamy:

$$\begin{aligned}x &= 79 \\y &= 125 \cdot y' = 125 \\z &= 125 \cdot z' = 125\end{aligned}$$

co wymaga $t = 382$ (żeby wszystkie cztery zmienne sumowały się do 711), ale wówczas ich iloczyn nie wynosiłby 711\,000\,000.

- Jeśli $x = 2 \cdot 79 = 158$, to problem dla pozostałych trzech zmiennych sprowadza się do:

$$\begin{aligned}y + z + t &= 553 \\yzt &= 2^5 \cdot 3^2 \cdot 5^6\end{aligned}$$

Musimy powtórzyć przeprowadzone już rozumowanie, jak czyniliśmy uprzednio: y musi wynosić $125y'$, $z = 125z'$, a

$$125(y' + z') + t = 553$$

tak więc $y' + z' \leq 4$. Żadna z tych możliwości:

$$\begin{aligned} y' &= 1 \quad i \quad z' = 1 \\ y' &= 1 \quad i \quad z' = 2 \\ y' &= 1 \quad i \quad z' = 3 \\ y' &= 2 \quad i \quad z' = 2 \end{aligned}$$

także nie prowadzi do rozwiązania.

Łatwo zauważyc, że jeśli

- $x = 3 \cdot 79 = 237$
- $x = 5 \cdot 79 = 395$
- $x = 6 \cdot 79 = 474$
- $x = 7 \cdot 79 = 553$
- $x = 8 \cdot 79 = 632$

to problem nie ma rozwiązań. Przy każdej z tych możliwości maleje liczba dodatkowych możliwości do rozważenia. Na przykład: jeśli $x = 553$, to dla $y = 125y'$ i $z = 125z'$ mamy $125(y' + z') + t = 158$, co natychmiast daje sprzeczność, ponieważ $y' + z' \geq 2$.

Jeśli problem naprawdę ma rozwiązanie, to $x = 316$. W tej (ostatniej) możliwości problem dla pozostałych trzech zmiennych sprowadza się do:

$$y + z = t = 395$$

$$yzt = 2^4 \cdot 3^2 \cdot 5^6$$

Ponownie musimy rozważyć kilka możliwości. Ponieważ suma trzech pozostałych zmiennych jest podzielna przez 5, jedna albo wszystkie trzy zmienne są podzielne przez 5. Jeśli tylko jedna zmienna (powiedzmy x) jest podzielna przez 5, to $y = 5^6y'$ i mamy sprzeczność, $y > 395$. Stąd wiemy, że wszystkie zmienne y , z i t są podzielne przez 5:

$$y = 5y', \quad z = 5z', \quad t = 5t'$$

Teraz problem wygląda następująco:

$$y' + z' + t' = 79$$

$$y'z't' = 2^4 \cdot 3^2 \cdot 5^3$$

Ponieważ $y' + z' + t' = 79$, jedna ze zmiennych z primami nie jest podzielna przez 5, a więc któraś z pozostałych zmiennych musi być podzielna przez 25 (powiedzmy $y' = 25y''$). W takim wypadku mamy:

$$25y'' + z' + t' = 79$$

$$y''z't' = 2^4 \cdot 3^2 \cdot 5 = 720$$

Oznacza to, że y'' wynosi 1 lub 2, lub 3 (w przeciwnym razie $25y'' > 79$). Dwie ostatnie możliwości nie prowadzą do rozwiązania (np. jeśli $y'' = 2$, to $z' + t' = 29$ i $z't' = 360$, co nie ma rozwiązania w liczbach rzeczywistych, a z zadaniem jesteśmy już wystarczająco „zespoleni”, czyż nie?). Tak więc $y'' = 1$ (tzn. $y = 125$). W takim razie:

$$z' + t' = 54$$

$$z't' = 720$$

co daje nam $z' = 24$ i $t' = 30$ lub odwrotnie. Tak więc $z = 5 \cdot 24 = 120$ oraz $t = 5 \cdot 30 = 150$.

To rozwiązuje problem. Rzeczywiście ma on jedno rozwiązanie (przeliczliśmy ceny z powrotem na dolary i centy):

$$x = 3,16 \$, \quad y = 1,25 \$, \quad z = 1,20 \$ \quad \text{i} \quad t = 1,50 \$$$

Bardzo ważne jest systematyczne podejście do rozwiązywania tego problemu. Krok po kroku wyeliminowaliśmy niemożliwe rozwiązania i – co jeszcze ważniejsze – rozpoczęliśmy od znacznego okrojenia przestrzeni rozwiązań poprzez zbadanie pierwszych dzielników iloczynu cen.

Można było oczywiście użyć również innej metody. Można było także zgadywać. Z reguły odrobina zgadywania jest wskazana i może ułatwić wgląd w problem. Przekonamy się o tym bliżej w następnym rozdziale. Warto również zauważać, że proste zgadywanie może doprowadzić do zadziwiających „odkryć”, takich jak następujące zbiory cen:

$$(3,16 \$, 1,25 \$, 1,25 \$, 1,44 \$) \quad \text{lub} \quad (2,37 \$, 2,00 \$, 2,00 \$, 0,75 \$)$$

W przypadku obu zbiorów iloczyn cen wynosi 7,11 \$, suma cen w pierwszym z nich wynosi 7,10 \$, a w drugim 7,12 \$. Problem ze zgadywaniem polega na tym, że rzadko prowadzi Cię do kolejnego, lepszego rozwiązania. Pomysł poprawiania jakości rozwiązania jest wpisany w wiele klasycznych i współczesnych metod optymalizacji. Pokrzepiający jest więc fakt, że sprawy idą ku lepszemu!

3. Metody tradycyjne – część I

W słowach wszyscy mężczyźni podobni są do siebie;
czyny dopiero odsłaniają różnice.

Molière: *Skąpiec*¹

Istnieje wiele klasycznych algorytmów działających w ten sposób, że w danej przestrzeni wyszukują optymalne rozwiązanie. W rzeczywistości jest ich tak wiele, że pojawia się naturalne pytanie, dlaczego mamy do czynienia z taką mnogością wyborów. Smutna odpowiedź brzmi: żadna z tradycyjnych metod nie jest wystarczająco elastyczna. Za każdym razem, gdy zmienia się problem, musimy zmienić algorytm. Jest to jedna z podstawowych wad powszechnie przyjętych metod optymalizacyjnych. Dla każdego zadania mamy metodę – problem jednak polega na tym, że większość ludzi zna jedną metodę lub w najlepszym razie kilka. W rezultacie często używają oni złego narzędzia i uzyskują kiepskie wyniki.

Klasyczna metoda optymalizacyjna, jeśli odpowiada zadaniu, które rozwiążujemy, może dać bardzo dobre wyniki. Oznacza to, że opłaca się wiedzieć, kiedy ją stosować i kiedy jej nie stosować. Metody te możemy w szerokim sensie ująć w dwie klasy:

- algorytmy, które oceniają tylko pełne rozwiązania,
- algorytmy, które wymagają oceny rozwiązań częściowo skonstruowanych lub przybliżonych.

Powyzsze rozróżnienie jest bardzo ważne. Ilekroć algorytm zajmuje się pełnymi rozwiązaniami, można go zatrzymać w dowolnej chwili i będziemy mieli co najmniej jedno potencjalne rozwiązanie, z którego możemy korzystać. Z kolei po przerwaniu pracy algorytmu pracującego na częściowych rozwiązaaniach możemy nie móc skorzystać z wyników jego pracy. Pełne rozwiązania oznaczają tylko, że wszystkie zmienne decyzyjne są określone. Na przykład dwójkowe

¹Przekład Tadeusz Żeleński (Boy), Zakład Narodowy im. Ossolińskich, wydanie X, 1964 (przyp. tłum.).

ciągi o długości n są pełnym rozwiązaniem SAT z n zmiennymi. Permutacje n miast są pełnym rozwiązaniem TSP. Wektory n liczb zmiennopozycyjnych są pełnym rozwiązaniem NLP. Możemy łatwo porównać dwa pełne rozwiązania za pomocą funkcji oceny. Wiele algorytmów działa na zasadzie właśnie takich porównań i zajmuje się w każdej chwili jednym pełnym rozwiązaniem. Po znalezieniu rozwiązania, które jest lepiej oceniane niż dotychczas najlepsze, następujemy dotychczasowe rozwiązanie nowym. Przykłady tego typu podejścia obejmują przeszukiwanie wyczerpujące, wyszukiwanie lokalne, wspinanie się oraz gradientowe metody optymalizacyjne analizy numerycznej. Niektóre z nowoczesnych metod heurystycznych, jak symulowane wyżarzanie, przeszukiwanie z tabu oraz algorytmy ewolucyjne, także należą do tej kategorii (zob. rozdziały 5 i 6).

Z kolei rozwiązania częściowe pojawiają się w dwóch postaciach: 1) jako niepełne rozwiązania pierwotnego problemu lub 2) jako pełne rozwiązania zredukowanego (tzn. prostszego) problemu.

Niepełne rozwiązania znajdują się w podzbiorze pierwotnej przestrzeni przeszukiwania. Na przykład w problemie SAT możemy rozważać wszystkie ciągi binarne, w których pierwsze dwie zmienne mają wartość 1 (tzn. TRUE). Z kolei w TSP możemy interesować się wszystkimi permutacjami miast, które zawierają ciąg 7 – 11 – 2 – 16. Podobnie dla NLP możemy rozpatrywać wszystkie rozwiązania, które mają $x_3 = 12,0129972$. W każdym z tych przypadków skupiamy naszą uwagę na podzbiorze przestrzeni przeszukiwania, który ma określoną własność. Liczymy tutaj, że własność tę ma także prawdziwe rozwiązanie!

Stosując inne podejście, możemy niejednokrotnie ująć pierwotne zadanie w zbiór mniejszych i prostszych podzadań. Mamy wówczas nadzieję, że po rozwiązaniu wszystkich tych prostszych problemów będziemy mogli połączyć wynikające z nich rozwiązania częściowe i uzyskać rozwiązanie docelowe. W przypadku TSP możemy zajmować się zbiorem k spośród n miast i próbować ustalić najkrótszą trasę od miasta i do miasta j , która przechodzi przez wszystkie miasta z tego zbioru (zob. podrozdział 4.3). W sytuacji z NLP możemy ograniczyć dziedziny niektórych zmiennych x_i – zmniejszając w ten sposób radykalnie przestrzeń przeszukiwania – a następnie próbować znaleźć pełne rozwiązanie w tak ograniczonym obszarze. Takie częściowe rozwiązania mogą niekiedy służyć jako cegiełki przy tworzeniu rozwiązania pierwotnego problemu.

Upraszczanie złożonego problemu przez dzielenie go na małe kawałki, z którymi łatwo możemy sobie poradzić, brzmi bardzo kusząco. Algorytmy działające na częściowych rozwiązanach stwarzają jednak dodatkowe trudności. Trzeba tutaj: 1) opracować sposób organizacji podprzestrzeni tak, żeby można je było efektywnie przeszukiwać oraz 2) stworzyć nową funkcję oceny, która będzie mogła określić jakość częściowych rozwiązań. To ostatnie zadanie jest w wypadku rzeczywistych problemów szczególnie trudne – jaka jest bowiem wartość pilota służącego do zmiany kanałów? Odpowiedź zależy od tego, czy w pobliżu znajduje się odbiornik telewizyjny! Funkcjonalność pilota zależy w dużym stopniu od obecności innych przedmiotów (tj. telewizora, baterii, być może podłączenia telewizji kablowej). Oceniając częściowe rozwiązanie,

nie możemy wiedzieć, jak będzie wyglądało jego otoczenie. W najlepszym razie możemy próbować przybliżyć jego postać, przy czym zawsze możemy się pomylić.

Wymieniony jako pierwszy – problem z organizacją przestrzeni przeszukiwania w postaci podzbiorów – jest również groźny, zaproponowano jednak wiele użytecznych sposobów radzenia sobie z nim. Możemy zorganizować przestrzeń przeszukiwania w drzewo, którego liście zawierają pełne rozwiązania. Przy takim układzie przechodzimy po gałęziach drzewa w głąb, stopniowo tworząc oczekiwane rozwiązanie. Czasami możliwe jest opuszczenie wielu rozgałęzień, o których wiemy, że prowadzą do kiepskich rozwiązań. Istnieją też inne sposoby użycia metody „podziału i ograniczeń”. Każdy z nich zależy od sposobu rozłożenia rozwiązań cząstkowych w przestrzeni przeszukiwania, a to z kolei od konkretnej ich reprezentacji.

Warto pamiętać, że zawsze istnieje wiele sposobów reprezentacji rozwiązań. Widzieliśmy trzy powszechnie stosowane reprezentacje dla problemów SAT, TSP i NLP. Były to odpowiednio: 1) ciągi binarne, 2) permutacje i 3) wektory liczb zmiennopozycyjnych. Nie musimy się jednak do nich ograniczać. Moglibyśmy na przykład reprezentować rozwiązania NLP jako ciągi binarne. Każde rozwiązanie miałoby tutaj długość kn , przy czym n wciąż oznacza liczbę zmiennych w problemie, k zaś określa liczbę bitów potrzebną do zapisania wartości każdej zmiennej. Można łatwo, w dwóch krokach, odwzorować ciąg binarny $\langle b_{k-1} \dots b_0 \rangle$ na wartość zmiennopozycyjną x_i z przedziału $[l_i, u_i]$:

- przekształcić ciąg binarny $\langle b_{k-1} \dots b_0 \rangle$ z postaci dwójkowej na dziesiętną:

$$(\langle b_{k-1} \dots b_0 \rangle)_2 = \left(\sum_{i=0}^{k-1} b_i \cdot 2^i \right)_{10} = x'$$

- znaleźć odpowiednią wartość zmiennopozycyjną x w podanym przedziale:

$$x = l_i + x' \cdot \frac{u_i - l_i}{2^k - 1}$$

Gdyby na przykład zadanym przedziałem był przedział $[-2, 3]$ i zastosowaliśmy ciągi o długości 10, to ciąg $\langle 1001010001 \rangle$ zostałby odwzorowany na 0,89833822, gdyż:

$$0,89833822 = -2 + 593 \cdot \frac{5}{1023}$$

Zauważmy, że wraz ze zwiększaniem wartości k uzyskujemy większą dokładność w przybliżaniu interesujących nas wartości zmiennopozycyjnych. W naszym przypadku po $\langle 1001010001 \rangle$ następuje ciąg $\langle 1001010010 \rangle$, który jest odwzorowany na 0,90322581 – luka między liczbami zmiennopozycyjnymi wynosi tutaj prawie 0,005. Gdybyśmy potrzebowali większej dokładności, musielibyśmy zwiększyć wartość k . Zwiększyłoby to jednocześnie czas obliczeń, gdyż za

każdym razem, gdy chcielibyśmy uzyskać wartość konkretnej zmiennej w NLP, musielibyśmy odkodowywać dłuższe ciągi binarne. Nic nas jednak nie zmusza do rozwiązywania NLP bezpośrednio za pomocą wartości zmiennopozycyjnych, choć sam problem jest zdefiniowany przy ich użyciu.

Ta swoboda wyboru reprezentacji umożliwia w rzeczywistości rozwinięcie kreatywności. Przy rozwiązywaniu TSP możemy używać wspomnianej wcześniej reprezentacji permutacyjnej. Nawet wtedy jednak możemy przyjąć trochę inną interpretację permutacji. Przypuśćmy, że i -ty element permutacji jest liczbą z zakresu od 1 do $n - i + 1$, która wskazuje miasto w pozostałej części pewnej listy C miast dla danego zadania TSP. Przyjmijmy, że $n = 9$ oraz że nasza lista miast C to

$$C = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

Wtedy trasę

$$5 - 1 - 3 - 2 - 8 - 4 - 9 - 6 - 7$$

reprezentuje wektor

$$(5 \ 1 \ 2 \ 1 \ 4 \ 1 \ 3 \ 1 \ 1)$$

Interpretacja wektora wygląda tak: 1) bierzemy piąte miasto z bieżącej listy C i usuwamy je (jest to miasto numer 5); 2) bierzemy pierwsze miasto z bieżącej listy C i usuwamy je (jest to miasto numer 1); 3) bierzemy drugie miasto z bieżącej listy C i usuwamy je (jest to miasto numer 3); i tak dalej. Ta reprezentacja wydaje się dość luźno związana z samym problemem, ale w rzeczywistości możemy jej użyć do identyfikacji każdej możliwej trasy. Co więcej, prawdopodobnie taka reprezentacja umożliwia zorganizowanie przestrzeni przeszukiwania w nowy, interesujący sposób, który nie byłby możliwy do zauważenia przy typowej reprezentacji permutacyjnej (w rozdziale 8 umieszczone rozważania dotyczące innych możliwych reprezentacji danych w TSP).

Nawet dla problemu SAT możliwe jest wiele różnych reprezentacji. Moglibyśmy na przykład reprezentować rozwiązanie jako wektor wartości zmiennopozycyjnych z zakresu $[-1, 1]$ i przypisać wartość logiczną TRUE każdej zmiennej, dla której odpowiednia zmenna zmiennopozycyjna jest nieujemna, zaś FALSE w przeciwnym wypadku (zob. podrozdział 9.1).

Dla dowolnej obranej przez nas reprezentacji rozwiązań określonego problemu możemy rozważyć określoną przez nią przestrzeń przeszukiwania jako zbiór pełnych rozwiązań i wykonać w nim przeszukiwanie w określony sposób. Możemy też podzielić przestrzeń na podzbiory i pracować na częściowych lub niepełnych rozwiązańach. Ponieważ istnieje tak dużo algorytmów, które możemy przedstawić, podzieliliśmy je na grupy. W tym rozdziale przyjrzymy się tym, które pracują na rozwiązańach pełnych, następnie – w rozdziale 4 – skupimy się na tych, które działają na rozwiązańach częściowych lub niepełnych.

3.1. Przeszukiwanie wyczerpujące

Przeszukiwanie wyczerpujące (ang. *exhaustive search*), jak wynika z nazwy, polega na sprawdzeniu każdego rozwiązania z przestrzeni przeszukiwania i wybieraniu tego z nich, które jest globalnie najlepsze. Innymi słowy, jeśli nie znamy wartości, którą możemy wyliczyć z najlepszego rozwiązania, to przy przeszukiwaniu wyczerpującym nie ma innego sposobu na uzyskanie pewności, że odkryło się takie rozwiązanie, jak wszystko przejrzeć. Staje się jasne, dlaczego metoda ta jest określana jako *wyczerpująca*. Nawet dla zadań niewielkiego rozmiaru zdążymy się zmęczyć i zestarzeć, zanim otrzymamy odpowiedź! Pamiętajmy, że rozmiar przestrzeni przeszukiwania jest zwykle ogromny. Sprawdzenie każdego możliwego rozwiązania może wymagać wieków pracy obliczeniowej komputera. W podrozdziale 1.1 zauważymy, że TSP dla 50 miast daje 10^{62} możliwych tras. Oczekiwanie na zakończenie przeszukiwania wyczerpującego takiej przestrzeni zajęłoby bardzo dużo czasu!

Algorytmy przeszukiwania wyczerpującego (czasami nazywane *wyliczeniowymi*; ang. *enumerative*) są pod pewnymi względami interesujące. Przede wszystkim są proste. Jedyne wymaganie, jakie mamy, to konieczność systematycznego wygenerowania wszystkich potencjalnych rozwiązań. Dodatkowo istnieją sposoby na zredukowanie związanej z tym pracy. Jednym z nich jest metoda *wyszukiwania z nawrotami* (ang. *backtracking*) – wkrótce przyjrzymy się jej bliżej. Co więcej, niektóre klasyczne algorytmy optymalizacyjne tworzące pełne rozwiązanie na podstawie rozwiązań cząstkowych (np. algorytm podziału i ograniczeń lub algorytm A^*) są oparte na przeszukiwaniu wyczerpującym. W związku z tym warto poświęcić tej metodzie trochę więcej czasu.

Podstawowe pytanie pojawiające się przy stosowaniu przeszukiwania wyczerpującego brzmi: jak wygenerować ciąg wszystkich potencjalnych rozwiązań zadania? Kolejność, w jakiej rozwiązania są generowane i oceniane, nie jest ważna, gdyż w planie mamy ocenienie ich wszystkich. Odpowiedź na to pytanie zależy od przyjętej reprezentacji.

3.1.1. Metody wyliczeniowe dla SAT

Wyliczenie wszystkich rozwiązań w przestrzeni przeszukiwania dla SAT z n zmiennymi wymaga wygenerowania każdego ciągu binarnego o długości n . Takich ciągów, począwszy od $\langle 0 \dots 000 \rangle$, a skończywszy na $\langle 1 \dots 111 \rangle$, jest 2^n . Wszystkie te ciągi odpowiadają jednoznacznie liczbom całkowitym (zob. tab. 3.1).

Tabela 3.1. Odpowiedniość między ciągami binarnymi o długości $n = 4$ a ich dziesiętnymi równoważnikami

0000	0	0100	4	1000	8	1100	12
0001	1	0101	5	1001	9	1101	13
0010	2	0110	6	1010	10	1110	14
0011	3	0111	7	1011	11	1111	15

Wystarczy wygenerować wszystkie nieujemne liczby od 0 do $2^n - 1$. Następnie przekształcamy każdą liczbę w odpowiadający jej ciąg binarny o długości n i bity tego ciągu będą stanowiły przypisanie wartości logicznych zmiennym z problemu.

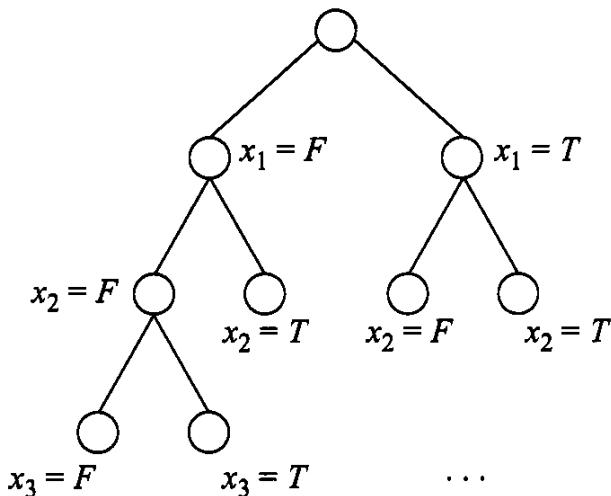
Inna możliwość polega na działaniu bezpośrednio na ciągach binarnych i na dodawaniu jedynki $\langle 0 \dots 001 \rangle$ w systemie dwójkowym. Po otrzymaniu w ten sposób następnego ciągu obliczamy jego wartość i jeśli okaże się ona lepsza od wcześniejszych, to zapamiętujemy dany ciąg jako najlepsze dotychczas rozwiązanie. W dalszym ciągu pozostaje kwestia, jak powinna zostać dobrana funkcja oceniająca, w tym wypadku może to być po prostu:

1 – jeśli ciąg prowadzi do spełnienia zadanej formuły boolowskiej oraz

0 – w przeciwnym wypadku

Dla naszego problemu możemy w rzeczywistości wstrzymać przeszukiwanie już w momencie, gdy natrafimy na pierwszy ciąg dający wynik 1.

Nawet w tak prostej sytuacji możemy zorganizować przeszukiwanie i wyjąć część ciągów, których w rzeczywistości nie musimy sprawdzać. Przypuśćmy, że dzielimy przestrzeń przeszukiwania dla SAT na dwie rozłączne części: pierwsza z nich zawiera wszystkie wektory, dla których $x_1 = F$ (FALSE), druga zaś zawiera wszystkie wektory z $x_1 = T$ (TRUE). W takiej sytuacji możemy zorganizować całą przestrzeń w drzewo (rys. 3.1).



Rysunek 3.1. Drzewo poszukiwań binarnych dla SAT

Na kolejnym poziomie w głąb drzewa ponownie dzielimy przestrzeń na dwie rozłączne części w zależności od wartości przypisanej x_2 . Kontynuujemy takie postępowanie dla wszystkich n zmiennych. Powstałe tak drzewo binarne ma n poziomów, przy czym i -temu poziomowi odpowiada i -ta zmienna.

Po takim uporządkowaniu przestrzeni rozwiązań możemy zastosować metodę przeszukiwania wyczerpującego, która szczególnie dobrze pasuje do drzew – przeszukiwanie w głąb (ang. *depth-first search*). Ta prosta rekurencyjna procedura (rys. 3.2) przechodzi graf skierowany (w szczególności drzewo), zaczynając od wybranego wierzchołka, a następnie przeszukując rekurencyjnie jego sąsiadów.

```

procedure depth-first( $v$ )
begin
    odwiedź  $v$ 
    for każdy potomek  $w$  wierzchołka  $v$  do
        depth-first( $w$ )
    end

```

Rysunek 3.2. Przeszukiwanie w głąb

Zaletą takiego sposobu organizacji przeszukiwania jest możliwość skorzystania z faktu, iż w niektórych sytuacjach możemy określić, że gałęzie poniżej danego wierzchołka mogą prowadzić tylko w ślepą uliczkę. W takiej sytuacji możemy wykonać „nawrót” do ojca danego wierzchołka na wyższym poziomie i kontynuować poszukiwanie w innym rozgałęzieniu drzewa. Możemy mieć do czynienia na przykład z sytuacją, gdy przypisanie pewnych wartości logicznych określonym zmiennym może powodować, że cała formuła przyjmuje wartość FALSE. W takiej sytuacji nie ma potrzeby kontynuowania pracy wzdłuż danej gałęzi. Jeśli zatem formuła SAT zawiera klauzulę

$$\dots \wedge (x_1 \vee \overline{x_3} \vee x_4) \wedge \dots$$

i pierwszym czterem zmiennym przypisano wartości:

$$x_1 = \text{FALSE}; \quad x_2 = \text{TRUE}; \quad x_3 = \text{TRUE}; \quad x_4 = \text{FALSE}$$

lub

$$x_1 = \text{FALSE}; \quad x_2 = \text{FALSE}; \quad x_3 = \text{TRUE}; \quad x_4 = \text{FALSE}$$

to niezależnie od tego, co zostanie przypisane x_5 oraz innym zmiennym, cała formuła SAT przyjmuje wartość FALSE; czas zabrać się za inną gałąź.

Przeszukiwanie w głąb przez sprawdzanie, czy dany wierzchołek (podprzestrzeń) może zawierać rozwiązanie, wyłącza pewne obszary przestrzeni przeszukiwania. Jeśli się da, to przeszukiwanie idzie w głąb, a jeśli się nie da, to przeszukiwanie wraca do ojca danego wierzchołka. Takie nawracanie jest powszechnie stosowane w wielu algorytmach drzewowych. Podobny pomysł zostało przedstawiony w rozdziale 4 przy omawianiu metod podziału i ograniczeń (podrozdział 4.4).

3.1.2. Metody wyliczeniowe dla TSP

Zastosowanie algorytmu wyczerpującego w wypadku TSP nie jest takie proste. Pojawia się tutaj przede wszystkim pytanie: Jak wygenerować wszystkie możliwe permutacje? Pierwsza trudność polega na tym, że jeśli graf nie jest „w pełni połączony” (spójny), to nie wszystkie permutacje muszą być sensowne. Miasto i może nie mieć (bezpośredniego) połączenia z miastem j , a wtedy żadna permutacja, w której te dwa miasta sąsiadują ze sobą, nie jest możliwa do zrealizowania. Możemy obejść tę przeszkodę, dopuszczając takie permutacje, ale nadając im bardzo duży koszt – o wiele większy niż długość dowolnej

dopuszczalnej trasy. W takiej sytuacji najlepsze rozwiązanie wśród wszystkich permutacji będzie też najlepszym dopuszczalnym rozwiązaniem (o ile oczywiście takie dopuszczalne rozwiązanie istnieje).

Drugim problemem są zwykłe drobne różnice związane z generowaniem wszystkich możliwych permutacji n liczb. Przyjrzyjmy się kilku podejściom. Możemy przyjąć podejście podobne do tego, którego używaliśmy dla SAT, gdzie mieliśmy odwzorowanie między liczbami całkowitymi a ciągami binarnymi. W związku z tym, że istnieje $n!$ różnych permutacji n liczb, możemy określić funkcję, która numeruje wszystkie permutacje od 0 do $n! - 1$ zgodnie z następującą zasadą. Zauważmy, że każda liczba całkowita k między 0 a $n! - 1$ może być jednoznacznie przedstawiona za pomocą poniżej opisanej tablicy liczb całkowitych:

$$c[n], \dots, c[2] \text{ przy czym } 0 \leq c[i] \leq i - 1 \text{ (przy } i = 2, \dots, n)$$

jako

$$k = \sum_{i=2}^n c[i] \cdot (i - 1)!$$

Generowanie permutacji odpowiadającej k (lub tablicy $c[i]$) możemy wykonać za pomocą następującej procedury [426]:

```

for  $i = 1$  do
     $P[i] \leftarrow i$ 
for  $i = n$  step  $-1$  do
    zamień  $(P[i], P[c[i] + 1])$ 
```

Powyższa metoda może być używana na wiele sposobów. W szczególności zamiast generować permutacje dla kolejnych liczb (od 0 do $n! - 1$) moglibyśmy generować te liczby całkowite losowo; wtedy jednak musielibyśmy śledzić, które permutacje zostały już wygenerowane (np. utrzymując wektor bitowy o długości $n!$).

Inna możliwość (zob. [425]) polega na użyciu procedury rekurencyjnej (rys. 3.3). Procedura `gen1_permutation` wywołana z wartością k ustaloną na -1 , parametrem i równym 0 i wyzerowanymi wszystkimi pozycjami w tablicy P wypisuje wszystkie permutacje ciągu $(1, \dots, n)$. Działa w ten sposób, że ustala 1 na pierwszej pozycji i generuje wszystkie $(n - 1)!$ permutacji liczb od 2 do n . Następne $(n - 1)!$ permutacji zostanie wypisane z 1 na drugiej pozycji i tak dalej. Dla $n = 3$ procedura ta daje następujący wynik: $(1\ 2\ 3)$, $(1\ 3\ 2)$, $(2\ 1\ 3)$, $(3\ 1\ 2)$, $(2\ 3\ 1)$, $(3\ 2\ 1)$. W rzeczywistych zastosowaniach zwykle nie wypisujemy bieżącej permutacji, ale raczej robimy z nią coś użytecznego (np. obliczamy sumę wszystkich odległości między miastami w permutacji).

Problem z dotychczas przedstawionymi metodami polega na tym, że nie są zbyt praktyczne. Łatwo możemy się przekonać, że tak jest, jeśli spróbujemy użyć którejkolwiek z nich dla $n > 20$. Bardziej praktyczne podejście polega na przekształcaniu jednej permutacji w drugą, nową (to łatwiejsza część zadania)

```

procedure gen1_permutation(i)
begin
    k  $\leftarrow k + 1$ 
    P[i]  $\leftarrow k$ 
    if k = n then
        for q = 1 to n do
            print P[q]
        for q = 1 to n do
            if P[q] = 0 then gen1_permutation(q)
        k  $\leftarrow k - 1$ 
        P[i]  $\leftarrow 0$ 
    end

```

Rysunek 3.3. Procedura generująca wszystkie permutacje liczb całkowitych od 1 do *n*

i zapewnieniu, że żadna permutacja się nie powtórzy, dopóki wszystkie nie zostaną wygenerowane (ta część zadania to prawdziwe wyzwanie). Zaletą tego podejścia jest to, że nie trzeba śledzić, które permutacje zostały już wygenerowane. Bieżąca permutacja zawsze wyznacza następną! Tak postawiony cel jest osiągany przez wiele algorytmów; jeden z nich, wypisujący kolejne permutacje, przedstawiono na rys. 3.4 [80].

Inna możliwość, która dobrze pasuje do TSP, polega na wyliczeniu wszystkich możliwych permutacji przez kolejne zamiany sąsiednich elementów. Przy takim podejściu oszczędzamy znaczącą ilość czasu, gdyż obliczenie kosztu nowej permutacji sprowadza się do określenia wpływu, w miejscu, gdzie nastąpiła zamiana – reszta permutacji pozostaje taka sama, nie trzeba więc przeliczać całego jej kosztu. Więcej na temat różnych algorytmów generowania permutacji można znaleźć w [426]. W pracy tej pojawiają się też wskazówki na temat efektywnych implementacji opisywanych algorytmów.

3.1.3. Metody wyliczeniowe dla NLP

Przeszukiwanie wyczerpujące w przestrzeniach ciągłych jest w ścisłym sensie niemożliwe – mamy przecież do czynienia z nieskończenie wieloma możliwościami. Możliwe jest tutaj podzielenie dziedziny każdej ciągłej zmiennej na skończoną liczbę przedziałów i rozważenie ich iloczynu kartezjańskiego. Przy szukaniu maksimum funkcji $f(x_1, x_2)$, przy czym $x_1 \in [-1, 3]$ oraz $x_2 \in [0, 12]$, możemy na przykład podzielić zakres x_1 na 400 odcinków o długości 0,01, a zakres x_2 na 600 odcinków o długości 0,02. Daje to $400 \times 600 = 240\,000$ komórek, które możemy kolejno zbadać. Wartość odpowiadającą każdej komórce możemy obliczyć, stosując funkcję kosztu do określonego wierzchołka komórki lub też do jej środka. Po określeniu komórki o najlepszej wartości możemy dalej zastosować naszą metodę i wykonać całą procedurę wewnątrz wyznaczonej komórki.

```

procedure gen2-permutation( $I, P, n$ )
  begin
    if not  $I$  then
      begin
        for  $i \leftarrow 0$  step 1 until  $n$  do
           $x[i] \leftarrow 0$ 
           $x[n] \leftarrow -1$ 
        goto  $E$ 
      end
      for  $i \leftarrow n$  step -1 until 0 do
      begin
        if  $x[i] \neq i$  then goto  $A$ 
         $x[i] \leftarrow 0$ 
      end
       $P[0] \leftarrow -1$ 
      goto  $E$ 
     $A:$     $x[i] \leftarrow x[i] + 1$ 
     $P[0] = 0$ 
    for  $i \leftarrow 1$  step 1 until  $n$  do
    begin
       $P[i] \leftarrow P[i - x[i]]$ 
       $P[i - x[i]] \leftarrow i$ 
    end
   $E:$    end

```

Rysunek 3.4. Procedura ta tworzy wszystkie permutacje liczb od 0 do n . Przy wejściu do procedury z $I = \text{FALSE}$ wykonuje ona operacje inicjujące i nie daje w wyniku żadnej permutacji. Każde następne wejście do procedury z $I = \text{TRUE}$ powoduje umieszczenie nowej permutacji w komórkach od $P[0]$ do $P[n]$. Po zakończeniu tego procesu jest ustawiany wartownik – $P[0] = -1$

Metoda ta jest poszukiwaniem wyczerpującym w tym sensie, że przy jej pomocy próbujemy „pokryć” wszystkie możliwe rozwiązania. Nie jest to jednak prawdziwe przeszukiwanie wyczerpujące, gdyż dla każdej komórki brane pod uwagę jest tylko jedno rozwiązanie i tylko ono jest obliczane. Duże znaczenie ma tutaj rozmiar komórki – im mniejsze są komórki, tym bardziej metoda ta zbliża się do rzeczywistego „wyczerpywania” przestrzeni przeszukiwania. Warto sobie też zdawać sprawę z wad tego podejścia.

- Stosowanie komórek o większej rozdzielczości znacząco zwiększa całkowitą ich liczbę. Gdybyśmy dziedzinę pierwszej zmiennej podzieliли na 4000 odcinków o długości 0,001, dziedzinę zaś drugiej na 6000 odcinków o długości 0,002, to całkowita liczba komórek wzrosłaby o dwa rzędy wielkości z 240 000 do 24 000 000.

- Użycie komórek o małej rozdzielniości zwiększa prawdopodobieństwo, że najlepsze rozwiązanie nie będzie znalezione, tzn. komórka z najlepszym rozwiązaniem może nie mieć najlepszej wartości, gdyż badaliśmy problem w zbyt prosty sposób.
- Metoda ta przy stosowaniu do problemów o bardzo wielu wymiarach (zmiennych) staje się niepraktyczna, gdyż pojawiające się liczby komórek są ogromne. Problem optymalizacyjny z 50 zmiennymi, dla których zastosowaliśmy podział na zaledwie 100 przedziałów, daje 10^{100} komórek.

Jeśli mamy do czynienia z małym problemem i mamy dosyć czasu, żeby wyliczyć wszystkie elementy przestrzeni przeszukiwania, to możemy uzyskać gwarancję znalezienia najlepszego rozwiązania. Nie należy jednak stosować tego podejścia do większych problemów, gdyż proces wyliczania może się nigdy nie skończyć.

3.2. Przeszukiwanie lokalne

Nie musimy koniecznie w sposób wyczerpujący przeszukiwać całej przestrzeni rozwiązań. Możemy skupić naszą uwagę na lokalnym sąsiedztwie jakiegoś konkretnego rozwiązania. Odpowiednia procedura składa się z czterech kroków.

1. Pobierz rozwiązanie z przestrzeni przeszukiwania i oblicz jego wartość. Stanowi ono *bieżące rozwiązanie*.
2. Zastosuj do bieżącego rozwiązania pewne przekształcenie. Dla tak otrzymanego wyniku oblicz wartość.
3. Jeśli nowe rozwiązanie jest lepsze niż bieżące rozwiązanie, to zamień je rolami; gdy nie jest – odrzuć nowe rozwiązanie.
4. Powtarzaj kroki 2 i 3 dopóty, dopóki żadne przekształcenie z dostępnego zestawu nie polepsza bieżącego rozwiązania.

Kluczem do zrozumienia, jak takie *przeszukiwanie lokalne* działa, jest rodzaj przekształceń stosowanych do bieżącego rozwiązania. Jednym skrajnym wyborem jest użycie przekształcenia, która daje potencjalne rozwiązanie, wybierając je jednostajnie w sposób losowy z dostępnej przestrzeni przeszukiwania. W tej sytuacji bieżące rozwiązanie nie ma wpływu na prawdopodobieństwo wyboru dowolnego innego rozwiązania, a co za tym idzie przeszukiwanie w istocie staje się wyliczeniowe. W rzeczywistości jest możliwe, że takie przeszukiwanie jest jeszcze gorsze od metody wyliczeniowej, gdyż jest tu niewykluczone ponowne sprawdzanie punktów, które już zostały zbadane. Na drugim końcu skrajności leży przekształcenie, które zawsze daje bieżące rozwiązanie, prowadząc oczywiście donikąd!

W praktyce sensowne rozwiązanie leży gdzieś pomiędzy tymi ekstremami. Szukanie w lokalnym otoczeniu bieżącego rozwiązania jest takim użytecznym kompromisem. W ten sposób bieżące rozwiązanie wywiera wpływ na to, gdzie

ma się odbywać dalsze poszukiwanie, a gdy znajdziemy coś lepszego, możemy uaktualnić wartość bieżącego punktu nowym rozwiązaniem i w ten sposób zachować to, czego się nauczyliśmy. Jeśli rozmiar otoczenia jest niewielki, to możemy je bardzo szybko przeszukać. Może to jednak prowadzić do wpadnięcia w pułapkę lokalnego optimum. Z kolei, jeśli rozmiar otoczenia jest duży, mamy mniejsze szanse na takie utknięcie, ale cierpi na tym efektywność naszego poszukiwania. Zastosowany rodzaj przekształcenia bieżącego rozwiązania określa rozmiar jego otoczenia, musimy zatem bardzo ostrożnie je dobrać, biorąc pod uwagę to, co wiemy o funkcji oceny oraz o reprezentacji rozwiązań.

3.2.1. Przeszukiwanie lokalne i SAT

Algorytmy przeszukiwania lokalnego zadziwiająco dobrze odnajdują podstawienia spełniające formuły z pewnych klas SAT [428]. Jednym z najlepiej znanych (losowych) algorytmów lokalnego przeszukiwania dla SAT (dla formuł zadań w koniunkcyjnej postaci normalnej) jest pokazana na rys. 3.5 procedura GSAT.

```
procedure GSAT
begin
    for  $i \leftarrow 1$  step 1 until MAX-TRIES do
        begin
             $T \leftarrow a$  wygenerowane losowo podstawienie zmiennych
            for  $j \leftarrow 1$  step 1 until MAX-FLIPS do
                if dla  $T$  formuła jest spełnialna then return( $T$ )
                else wykonaj zamianę
            end
            return(„nie znaleziono podstawienia spełniającego formułę”)
        end
```

Rysunek 3.5. Procedura GSAT. Polecenie „wykonaj zamianę” oznacza zmianę wartości na przeciwną tej zmiennej w T , dla której zmiana spowoduje największe zmniejszenie liczby niespełnionych klauzul

Procedura GSAT ma dwa parametry: MAX-TRIES, który określa liczbę cykli (ang. *trials*) w jednym poszukiwaniu, oraz MAX-FLIPS, który określa maksymalną liczbę zamian (ang. *flips*) w cyklu.

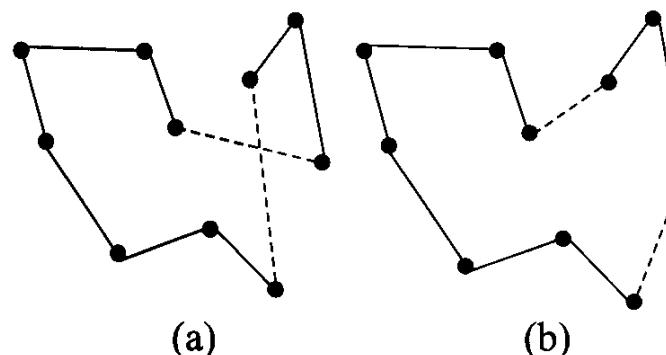
Algorytm GSAT rozpoczyna pracę od wygenerowanego losowo wartościowania logicznego. Jeśli nam się uda i to wartościowanie spełnia formuły, to algorytm się zatrzymuje. Taki uśmiech losu zdarza się jednak rzadko. W następnym kroku procedura zamienia wartości każdej zmiennej po kolej z TRUE na FALSE lub z FALSE na TRUE i za każdym razem zapisuje, o ile zmniejszyła się liczba niespełnionych klauzul. Po sprawdzeniu wszystkich możliwych zamian uaktualnia bieżące rozwiązanie, przypisując mu to nowe rozwiązanie, które dawało największe zmniejszenie liczby niespełnionych klauzul. I znowu,

jeśli tak obrane rozwiązanie daje spełnialność formuły, to kończymy pracę; jeśli nie, to algorytm zaczyna zamianie wartości zmiennych od początku.

Algorytm ten ma pewną interesującą własność. Najlepsza dostępna zamiana może tak naprawdę zwiększać liczbę niespełnionych klauzul. Wyboru dokonujemy jedynie spośród najbliższego otoczenia bieżącego rozwiązania. Jeśli każdy sąsiad (co rozumiemy jako podstawienie odległe o jedną zamianę) jest gorszy niż bieżące rozwiązanie w takim sensie, że spełnia mniejszą liczbę klauzul, to GSAT wybierze tego z nich, który jest zły, ale w najmniejszym stopniu. Procedura ta przypomina tu trochę wybory prezydenckie. Jak zobaczymy w rozdziale 5, taka sytuacja może okazać się bardzo korzystna – algorytm dostaje szansę na ucieczkę z lokalnego optimum. Jest to jednak jedynie szansa, a nie gwarancja, gdyż algorytm może zacząć oscylować między jakimiś punktami i nigdy nie wydostać się poza obręb jakiegoś plateau¹. Rzeczywiście eksperymenty wykazały, że w GSAT ruchy po plateau w poszukiwaniach dominują [428]. Zaproponowano wiele modyfikacji tej procedury, mających na celu uniknięcie tych problemów [182, 429]. Możemy na przykład każdej klauzuli przypisać wagę [429], a następnie na końcu danej próby możemy zwiększyć wagi wszystkich klauzul, które zostały niespełnione. Klauzula, dla której nie udało się zapewnić spełnialności w wielu próbach, będzie miała wagę większą od pozostałych, a co za tym idzie nowe rozwiązanie sprawiające, że stanie się ona spełnialna, będzie miało odpowiednio większy wynik.

3.2.2. Przeszukiwanie lokalne i TSP

Dla problemu TSP opracowano wiele algorytmów przeszukiwania lokalnego. Najprostszy z nich jest nazywany *2-opt*. Procedura ta zaczyna od losowej permutacji miast (nazwijmy tę trasę T) i próbuje ją ulepszyć. Otoczenie T jest zdefiniowane jako zbiór wszystkich tras, które możemy osiągnąć przez zmianę dwóch niesąsiadujących ze sobą krawędzi T . Taki ruch jest nazywany *zamianą dwukrawędziową* (ang. *2-interchange*); jest on przedstawiony na rys. 3.6.



Rysunek 3.6. Trasa przed (a) i po (b) wykonaniu zamiany w parach. Linie przerwane oznaczają zmieniane krawędzie

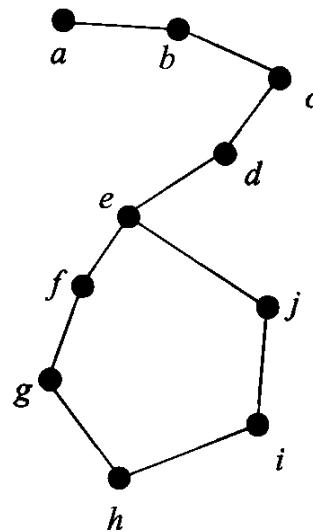
¹ Płaskowyżu (przyp. tłum.).

Nowa trasa T' , jeśli jest lepsza¹, zastępuje trasę T . Jeśli żadna trasa w otoczeniu T nie jest lepsza od tej trasy, to trasa T jest nazywana trasą *2-optimalną* (ang. *2-optimal*) i algorytm kończy pracę. Tak jak w wypadku GSAT algorytm powinien być uruchomiony dla kilku losowych permutacji początkowych, zanim naprawdę się poddamy.

Algorytm *2-opt* łatwo jest uogólnić do procedury *k-opt*, w której dokładnie k lub co najwyżej k krawędzi zostaje przeznaczonych do zastąpienia przez połączenia o mniejszym koszcie. W tym miejscu istotne znaczenie ma związek między rozmiarem otoczenia a efektywnością poszukiwania. Gdy k jest małe (tzn. rozmiar otoczenia jest niewielki), możemy szybko przeszukać całe otoczenie; taki mały zakres poszukiwań zwiększa jednak szanse na uzyskanie nie do końca optymalnej odpowiedzi. Z kolei dla większych wartości k liczba rozwiązań w otoczeniu staje się olbrzymia – liczba podzbiorów rośnie wykładniczo wraz z k . Z tego względu algorytm *k-opt* jest rzadko używany dla $k > 3$.

Najbardziej znaną procedurą lokalnych poszukiwań dla TSP jest algorytm Lina-Kernighana [284]. W pewnym sensie procedura ta jest ulepszoną wersją strategii *k-opt*, w której dopusczamy różne wartości k w różnych iteracjach. Co więcej, zamiast jak w *k-opt* korzystać z „pierwszego ulepszenia”, wybieramy raczej ulepszenie o największą wartość. Po znalezieniu lepszej trasy nie zastępujemy natychmiast dotychczasowej najlepszej trasy.

Istnieje wiele wersji podstawowej metody Lina-Kernighana. Różnią się one pewnymi szczegółami. Opiszemy tutaj w skrócie tę z nich (zob. też [78]), w której reprezentacja nie jest oparta na ścieżce Hamiltona, ale na tak zwanej δ -ścieżce. Zaraz przekonamy się, jak ona działa.



Rysunek 3.7. Obraz δ -ścieżki

Definicja δ -ścieżki mówi, że jest to ścieżka, dla której liczba wierzchołków jest większa niż liczba krawędzi oraz wszystkie wierzchołki pojawiają się na ścieżce tylko raz z wyjątkiem wierzchołka ostatniego, który musi pojawić się

¹Zwróćmy uwagę, że zastępujemy trasę za każdym razem, gdy znajdziemy ulepszenie. Innymi słowy, kończymy poszukiwanie w otoczeniu T , gdy znajdziemy „pierwsze ulepszenie”.

wcześniej. Za pomocą rysunku można o wiele zgrabniej wyjaśnić to pojęcie, spójrz więc na rys. 3.7, na którym widać δ -ścieżkę

$$a - b - c - d - e - f - g - h - i - j - e$$

Jest to δ -ścieżka, gdyż ma 10 krawędzi, 11 wierzchołków i każdy wierzchołek jest inny z wyjątkiem ostatniego wierzchołka e .

Każda δ -ścieżka może zostać przekształcona w legalną trasę przez zastąpienie jednej krawędzi – ostatniej. Zastępując na przykład krawędź $(j \ e)$ krawędzią $(j \ a)$, uzyskujemy poprawną trasę. Oprócz tego, zastępując ostatnią krawędź $(j \ e)$ dowolną inną krawędzią zaczynającą się w j , możemy używać nową δ -ścieżkę, na przykład zastępując krawędź $(j \ e)$ przez $(j \ f)$. Niech $\text{przełącz}(e, f)$ oznacza takie zastąpienie jednej δ -ścieżki przez inną, przy czym etykiety e i f wskazują przełączane wierzchołki. Kolejność wierzchołków jest tutaj istotna – pierwszy wierzchołek (e) jest usuwany i zastępowany drugim (f). Gdyby etykiety nie miały znaczenia, oznaczylibyśmy ten ruch po prostu jako *przełącz*.

Koszt nowej δ -ścieżki zwiększa się o

$$\text{cost}(j, f) - \text{cost}(j, e)$$

przy czym to „zwiększenie” może być ujemne. Ważne jest, żeby pamiętać, że δ -ścieżka ma dwie „ostatnie krawędzie”. W sytuacji z rys. 3.7 moglibyśmy, budując poprawną trasę lub generując nową δ -ścieżkę, zajmować się krawędzią $(j \ e)$ lub też krawędzią $(f \ e)$.

W tym momencie jesteśmy już gotowi do opisania podstawowych kroków algorytmu Lina-Kernighana. Zaczyna on pracę od wygenerowania losowej trasy T , a następnie generuje ciąg δ -ścieżek. Pierwsza jest tworzona na podstawie T , każda następna zaś za pomocą poprzedniej i operacji *przełącz*.

Na rysunku 3.8 pokazano podstawowy zarys algorytmu Lina-Kernighana. Kilka kolejnych wierszy kodu

```
if istnieje poprawiające koszt przełączanie p
then
    wykonaj przełącz, otrzymując nową  $\delta$ -ścieżkę  $p$ 
```

przedstawia ogólną koncepcję, ale jednocześnie ukrywa ważne szczegóły. Proces generowania nowej δ -ścieżki jest oparty na dodawaniu i usuwaniu krawędzi. Dla danego wierzchołka k i krawędzi $(i \ k)$ (tzn. dla dowolnej jednej iteracji pętli algorytmu) krawędź może być albo dodana, albo usunięta, ale nie mogą być wykonane obie te operacje. Wymaga to utrzymywania dwóch list odpowiednio z *dodawanymi* i *usuwanymi* krawędziami. Jeśli krawędź znajduje się już na jednej z tych list, to dopuszczalne przełączenia są ograniczone. Jeśli krawędź znajduje się na liście *dodanych* krawędzi, to przełączenie może ją tylko usunąć. Dzięki temu w tej części algorytmu mamy ograniczenie do najwyżej n przełączeń, przy czym n oznacza liczbę wierzchołków. Zauważmy jeszcze, że każda wygenerowana δ -ścieżka ma koszt ograniczony przez $\text{cost}(T)$. Kolejny ważny szczegół jest związany z wyborem wierzchołka w obu instrukcjach „*if istnieje*”, gdzie jest zmieniana trasa w δ -ścieżkę i gdzie jest tworzona nowa

```

procedure Lin-Kernighan
begin
    generuj trasę  $T$ 
    najlepszy_koszt =  $cost(T)$ 
    for wszystkie wierzchołki  $k$  w  $T$  do
        for wszystkie krawędzie  $(i \ k)$  w  $T$  do
            C:   begin
                if istnieje  $j \neq k$  takie, że  $cost(i, j) \leq cost(i, k)$ 
                then stwórz  $\delta$ -ścieżkę  $p$ , usuwając  $(i \ k)$  i dodając  $(i \ j)$ 
                else goto B
            A:   utwórz trasę  $T$  na podstawie  $p$ 
                if  $cost(T) <$ najlepszy_koszt then
                    najlepszy_koszt  $\leftarrow cost(T)$ 
                    zapamiętaj  $T$ 
                    if istnieje przełączenie  $p$  dające  $\delta$ -ścieżkę
                        o koszcie nie większym od  $cost(T)$ 
                    then
                        wykonaj przełącz, otrzymując nową  $\delta$ -ścieżkę  $p$ 
                        goto A
                B:   if  $cost(T) <$ najlepszy_koszt then
                    najlepszy_koszt  $\leftarrow cost(T)$ 
                    zapamiętaj  $T$ 
                    if pozostały niesprawdzone kombinacje wierzchołek/krawędź
                    then goto C
                end
            end
        end
    end

```

Rysunek 3.8. Struktura algorytmu Lina-Kernighana

δ -ścieżka z bieżącej ścieżki (rys. 3.8). Ponieważ sprawdzanie wszystkich możliwych wierzchołków może być zbyt drogie, w tym miejscu ma sens ograniczenie poszukiwania, co też zostało wykonane na wiele sposobów [242].

Procedura Lina-Kernighana może działać bardzo szybko i dawać rozwiązania bliskie optymalnych (np. o najwyżej dwa procent gorsze od najlepszej ścieżki [243]) dla zadań TSP z milionami miast. Uruchomiona na nowoczesnej stacji roboczej daje ona takie wyniki w czasie poniżej godziny [239].

3.2.3. Przeszukiwanie lokalne i NLP

Większość numerycznych algorytmów optymalizacyjnych dla NLP jest oparta na jakiejś wersji zasady przeszukiwania lokalnego. Są to bardzo różnorodne metody. Trudno je też poklasyfikować w jakieś zgrabne kategorie, gdyż występuje tutaj wiele różnych możliwości. Niektóre z nich używają heurystyk do

określania kolejnych punktów obliczania funkcji, inne korzystają z pochodnych funkcji oceny, a jeszcze inne mają charakter ściśle lokalny i są zamknięte w ograniczonym obszarze przestrzeni przeszukiwania. Wszystkie pracują na pełnych rozwiązaniach i wszystkie przeszukują przestrzeń pełnych rozwiązań.

Przypomnijmy, że w NLP chodzi o znalezienie wektora \mathbf{x} , który

optymalizuje $f(\mathbf{x})$ przy $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$

przy czym $\mathbf{x} \in \mathcal{F} \subseteq \mathcal{S}$. Funkcja oceny f jest określona na przestrzeni przeszukiwania $\mathcal{S} \subseteq \mathbb{R}^n$, a zbiór $\mathcal{F} \subseteq \mathcal{S}$ określa obszar rozwiązań dopuszczalnych. Zwykle przestrzeń przeszukiwania \mathcal{S} jest określona jako n -wymiarowa kostka w \mathbb{R}^n , przy czym dziedziny poszczególnych zmiennych są określone przez ich dolne i górne ograniczenia:

$$d(i) \leq x_i \leq g(i), \quad 1 \leq i \leq n$$

natomiast obszar rozwiązań dopuszczalnych \mathcal{F} jest zdefiniowany jako zbiór m dodatkowych ograniczeń ($m \geq 0$),

$$g_j(\mathbf{x}) \leq 0 \text{ dla } j = 1, \dots, q \text{ oraz } h_j(\mathbf{x}) = 0 \text{ dla } j = q + 1, \dots, m$$

Dla dowolnego punktu $\mathbf{x} \in \mathcal{F}$ ograniczenia g_k , dla których mamy $g_k(\mathbf{x}) = 0$, są nazywane *aktywnymi* ograniczeniami na \mathbf{x} . Ograniczenia równościowe h_j są analogicznie określane jako aktywne we wszystkich punktach \mathcal{S} .

Jednym z głównych powodów, dla których istnieje tak wiele podejść do NLP, jest fakt, że żadna metoda nie jest lepsza od pozostałych. Na ogół nie jest możliwe opracowanie deterministycznej metody znajdowania rozwiązania globalnie najlepszego ze względu na f , która byłaby lepsza od przeszukiwania wyczerpującego. W pracy [204] Gregory napisał:

Oczekiwanie, że uda się znaleźć jeden ogólny kod rozwiązujący NLP dla wszystkich rodzajów modeli nieliniowych, jest nierealne. Zamiast tego należy się skoncentrować na wyborze kodu, który pasuje do rozwiązywanego problemu. Jeśli nasz problem nie pasuje do żadnej kategorii z wyjątkiem kategorii „ogólne” lub jeśli upieramy się przy uzyskaniu rozwiązania, które jest globalnie optymalne (a nie widać szans, żeby uniknąć sytuacji z wieloma lokalnymi optimami), to powinniśmy się przygotować do stosowania metody, która sprowadza się do przeszukiwania wyczerpującego, gdyż mamy problem trudny do rozwiązania.

Proponowane tutaj metody możemy podzielić z dwóch punktów widzenia: 1) ze względu na rodzaj rozwiązywanego problemu i 2) ze względu na rodzaj użytej metody [219]. Pierwszy punkt widzenia prowadzi do kilku niezależnych kategorii problemów: problemów z ograniczeniami¹ w przeciwnieństwie

¹Zauważmy, że problemy z ograniczeniami możemy dalej podzielić na problemy z ograniczeniami równościowymi, z ograniczeniami w postaci nierówności oraz na problemy z oboma rodzajami ograniczeń; możemy także wprowadzać podział ze względu na typ ograniczeń (np. problemy z ograniczeniami liniowymi).

do problemów bez ograniczeń, problemów dyskretnych w przeciwnieństwie do problemów ciągłych, a także do problemów uwzględniających konkretne rodzaje funkcji oceny – wypukłe, kwadratowe, liniowo separowalne itp. Drugi punkt widzenia daje klasyfikację na metody oparte na pochodnych¹ w przeciwnieństwie do metod bez użycia pochodnych, na metody analityczne w przeciwnieństwie do numerycznych, na metody deterministyczne w przeciwnieństwie do losowych itd.

Jest też wiele istotnych problemów w ramach szerzej rozumianej kategorii problemów NLP. Jedna grupa problemów polega na opisanym wcześniej szukaniu ekstremum funkcji. Druga zajmuje się znajdowaniem jej miejsc zerowych. Chociaż wydaje się, że te problemy są zupełnie różne i często się tak je traktuje, możemy sobie wyobrazić, że podamy dodatkową funkcję oceny, która będzie oceniała, na ile dany punkt jest miejscem zerowym jakiejś funkcji. Jeśli wybór będzie odpowiedni, to taka funkcja da nam odpowiedź identyczną z odpowiedzią pochodzącą z funkcji, która nas interesuje. Rzućmy okiem na kilka lokalnych metod radzenia sobie z zadaniami z zakresu NLP.

Metody przedziałowe. Przyjrzyjmy się funkcji z rys. 3.9, dla której $f(x^*) = 0$. Nasze zadanie polega na znalezieniu x^* . Założymy, że wiemy, jak ograniczyć x^* dwiema innymi liczbami a i b . Jeden z prostych sposobów znajdowania x^* polega na wykonywaniu bisekcji zakresu między a a b (tzn. dzieleniu tego zakresu na połowy). Wybieramy przy tym środkowy punkt m , określamy wartość $f(m)$, a następnie przenosimy lewy lub prawy koniec zakresu na m w zależności od tego, czy $f(m)$ jest dodatnie. Kontynuując tę procedurę, możemy po pewnym czasie dowolnie zbliżyć się do x^* , dla którego $f(x^*) = 0$. (Jest to doskonały przykład podejścia typu dziel i rządź² omawianego w podrozdziale 4.2).

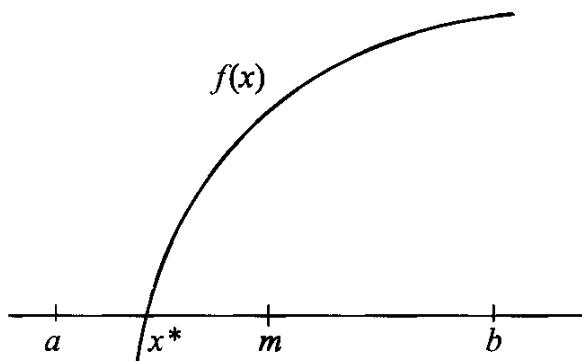
Zwróćmy uwagę, że ta metoda wykonywania *bisekcji* jest metodą przeszukiwania lokalnego. Działa ona dzięki dwóm punktom ograniczającym a i b , wciąż jednak tutaj aktualniamy tylko jeden punkt i w każdym kroku mamy tylko jedno aktualnie najlepsze rozwiązanie.

W omawianym rozwiązaniu korzystamy z założenia, że $f(a)f(b) < 0$; musimy być pewni, że $f(a)$ i $f(b)$ leżą po przeciwnych stronach zera. Algorytm bisekcji jest prosty.

1. Wybierz a i b tak, żeby $f(a)f(b) < 0$ i $x^* \in [a, b]$.
2. Wygeneruj $m \leftarrow (a + b)/2$.

¹Te z kolei możemy podzielić na oparte na pierwszych, drugich itd. pochodnych.

²W języku angielskim łacińskie „divide et impera” jest oddawane na dwa sposoby: „divide and rule” oraz „divide and conquer”. Te dwa tłumaczenia mają trochę inne odcienia znaczeniowe, co jest niekiedy wykorzystywane przez angielskojęzycznych autorów. W związku z tym, że po polsku jest tylko jedno utarte tłumaczenie dla tego łacińskiego zwrotu, a konieczność rozróżnienia znaczenia angielskich odpowiedników pojawi się rzadko, postanowiliśmy użyć utartego polskiego zwrotu, unikając dosłownego tłumaczenia „dziel i zdobywaj” (przyp. tłum.).



Rysunek 3.9. W metodzie bisekcji znajdujemy miejsce zerowe funkcji, najpierw ograniczając miejsce wystąpienia tego miejsca zerowego do pewnego przedziału, a następnie iteracyjnie dzieląc interesujący nas obszar na pół. Tutaj a i b oznaczają lewy i prawy koniec przedziału, punkt m stanowi środek, który będzie służył za następne sprawdzane miejsce, a x^* jest faktyczną odpowiedzią. W metodzie tej przyjmujemy, że kończymy pracę, gdy długość przedziału zmniejszy się poniżej zadanej z góry wielkości

3. Jeśli $f(m) \neq 0$, to albo $f(a)f((a+b)/2) < 0$, albo $f((a+b)/2)f(b) < 0$. Jeśli zachodzi ta pierwsza sytuacja, to przyjmij $b \leftarrow m$, w przeciwnym wypadku przyjmij $a \leftarrow m$.
4. Jeśli $(b - a) < \epsilon$, to się zatrzymaj, w przeciwnym wypadku przejdź do punktu 2.

Wartość ϵ oznacza małą liczbę, która wyznacza margines błędu, jaki jesteśmy gotowi dopuszczać.

Zajmiemy się teraz wyznaczeniem, jak szybko zmniejsza się w tym podejściu błąd oszacowania dla miejsca zerowego $f(x)$, gdyż to określi nam, ile dużych kroków będzie musiał wykonać ten algorytm. Jeśli zaczynamy od odcinka o długości $L_0 = b - a$, to po pierwszym kroku długość będzie wynosiła $L_1 = (b-a)/2$. Po kolejnym kroku będzie to $L_2 = (b-a)/2^2$ i tak dalej. Oznacza to, że długość przedziału jako funkcja liczby iteracji zmniejsza się geometrycznie: $L_n = (b-a)/2^n$. Jak się zaraz przekonamy, chociaż wynik ten wydaje się bardzo zachęcający, to istnieje wiele metod, które dają rozwiązanie o wiele szybciej.

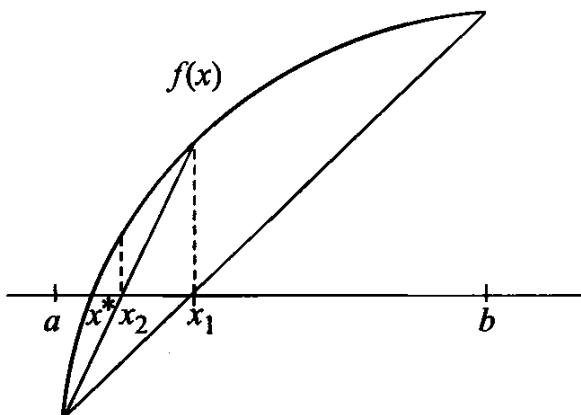
Jedną z takich metod jest metoda przedziałowa zwana *Regula Falsi*. W tej metodzie także zaczynamy od przedziału $[a, b]$ obejmującego miejsce zerowe $f(x)$ i spełniającego $f(a)f(b) < 0$. Zamiast jednak dzielić przedział na pół, prowadzimy sieczną między punktami $(a, f(a))$ a $(b, f(b))$ i znajdujemy punkt s , w którym linia ta przecina osią x . Parametry siecznej obliczamy ze wzoru

$$\frac{y - f(b)}{x - b} = \frac{f(a) - f(b)}{a - b}$$

który daje

$$s = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

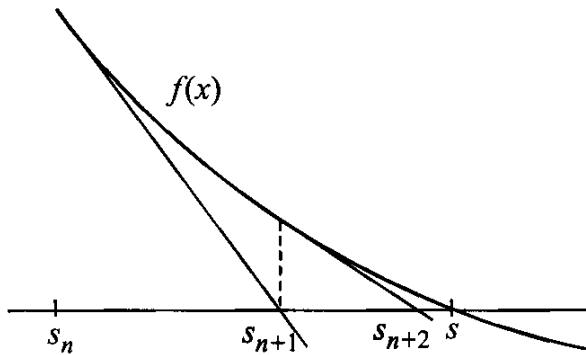
Wartość s jest pierwszym przybliżeniem pierwiastka $f(x)$ (miejscia zerowego). Jak poprzednio zastępujemy lewy lub prawy koniec przedziału $[a, b]$ w zależności od tego, czy $f(a)f(s)$ jest dodatnie, czy ujemne, i kończymy powtarzanie kroków procedury, gdy długość przedziału stanie się wystarczająco mała. Graficzne przedstawienie działania tej metody pokazano na rys. 3.10.



Rysunek 3.10. Metoda *Regula Falsi* korzysta z istnienia przedziału wokół miejsca zerowego funkcji. Zamiast odwoływać się do środka odcinka między a i b , łączymy prostą wartości funkcji w tych punktach i za następny punkt do przetwarzania bierzemy miejsce przecięcia tej prostej z osią x . W ten sposób przedział w kolejnych iteracjach zmniejsza się i kończymy pracę, gdy długość przedziału zmniejszy się poniżej zadanej z góry wielkości

Długość przedziału L_n z n -tego kroku *Regula Falsi* może nie zbiegać do zera przy $n \rightarrow \infty$. Zależy to od funkcji $f(x)$. Jednak metoda ta może dawać wynik szybciej niż metoda bisekcji. Dzieje się tak dlatego, że jeśli $|f(a)| > |f(b)|$, to będziemy się spodziewać, iż miejsce zerowe $f(x)$ znajduje się bliżej b niż a i vice versa. Przy korzystaniu z siecznej będziemy brać pod uwagę tę obserwację, o ile tylko odpowiada ona prawdzie. Można pokazać, że jeśli druga pochodna $f''(x)$ jest ciągła, to *Regula Falsi* daje wynik z szybkością $|s_n - x^*| \leq \lambda^{n-1} \epsilon$, gdzie $0 < \lambda < 1$ oraz ϵ jest z góry ustaloną stałą.

Metody stałopunktowe. Chociaż bisekcja i *Regula Falsi* dają wynik dla funkcji ciągłych, w porównaniu z innymi metodami, takimi jak *metoda Newtona*, są powolne. Metoda ta, trzeba jednak wiedzieć, nie zawsze musi być zbieżna z odpowiedzią nawet dla funkcji ciągłych. Stosowanie jej wymaga zatem większej ostrożności. Koncepcję tej metody przedstawiono na rys. 3.11. Dla danej funkcji $f(x)$ oraz początkowo zgadniętej wartości, dla której spodziewaliśmy się, że $f(x) = 0$, powiedzmy s , znajdźmy prostą styczną do funkcji w punkcie $F(s)$, a następnie sprawdźmy, gdzie ta prosta przecina się z osią x . Następnie użyjemy tego punktu przecięcia jako następnego zgadniętego miejsca zerowego. Kroki te powtarzamy, aż odległość między zgadniętymi wartościami w dwóch kolejnych rundach stanie się wystarczająco mała.



Rysunek 3.11. Ilustracja metody Newtona znajdowania miejsca zerowego funkcji. Zamiast umieszczenia miejsca zerowego w przedziale, jak jest w metodzie bisekcji czy *Regula Falsi*, bierzemy pod uwagę nachylenie funkcji w miejscu, które sprawdzamy, i obliczamy punkt przecięcia stycznej do wykresu funkcji w tym punkcie z osią x . Ten punkt przecięcia będzie następnym sprawdzanym przez nas punktem. Kończymy pracę, gdy odległość między kolejnymi znalezionymi punktami zmniejszy się poniżej wyznaczonego progu

Matematycznie ciąg zgadnień dla miejsca zerowego $f(x)$ wyraża wzór:

$$s_{n+1} = s_n - \frac{f(s_n)}{f'(s_n)}$$

dla $n = 0, 1, \dots$. Możemy spojrzeć na ten wzór jako na szczególny przypadek ciągu $s_{n+1} = g(s_n)$ występującego w problemie punktu stałego. Chcemy tutaj znaleźć funkcję $g(x)$ taką, że jeśli $f(s) = 0$, to $s = g(s)$. Zwykle istnieje wiele możliwych sposobów doboru tej pomocniczej funkcji $g(x)$. Jeśli na przykład $f(x) = x^2 - 4x + 4$, to jako $g(x)$ mogą być przydatne $g(x) = -(x - 4)/4$, $g(x) = x^2 - 3x + 4$ lub $g(x) = \sqrt{4x - 4}$. We wszystkich tych trzech przypadkach dla $x = 2$, dającego $f(x) = 0$, mamy $g(2) = 2$.

Przy znajdowaniu odpowiedniej funkcji pomocniczej $g(x)$ pomaga wprowadzenie dodatkowych ograniczeń. Pierwsze polega na tym, że dla dowolnego przedziału $L = [a, b]$ oraz dowolnego $x \in L$ ma zachodzić $g(x) \in L$. Innymi słowy, jeśli x znajduje się w przedziale $[a, b]$, to także znajduje się tam $g(x)$. Jeśli dodatkowo zażądamy, żeby $g(x)$ była ciągła, to będziemy mieć gwarancję, że w L znajduje się jej punkt stały.

Metoda Newtona polega zatem na znalezieniu funkcji pomocniczej $g(x)$ spełniającej powyższe ograniczenia oraz przedziału $L = [a, b]$, który zawiera pierwiastek $f(x)$, a następnie powtarzaniu obliczania funkcji uaktualniającej $s_{n+1} = g(s_n)$. Jeśli $|g'(x)| \leq B < 1$, to w L będzie tylko jeden punkt stały, przy czym dla x^* będącego pierwiastkiem $f(x)$ błąd $e_n = s_n - x^*$ spełnia $|e_n| \leq B^n / (1 - B) \cdot |s_1 - s_0|$. Działanie metody możemy zatrzymać, gdy $|s_{n+1} - s_n| < \epsilon$. Jeśli dobrze zgadniemy umiejscowienie x^* , to metoda Newtona może dać wynik bardzo szybko. Przy założeniu, że $f''(x)$ jest ciągła i $f'(x) \neq 0$ w przedziale L zawierającym x^* , istnieje ϵ takie, że metoda Newtona jest zbieżna w sposób kwadratowy, jeśli początkowa pozycja s_0 spełnia $|s_0 - x^*| < \epsilon$.

Metoda Newtona, chociaż daje wynik szybko przy spełnieniu wymienionych warunków, ma dwie wady: 1) może wymagać, żeby pierwsze przybliżenie

było bardzo bliskie x^* oraz 2) funkcja uaktualniająca s_{n+1} wymaga obliczenia $f'(x)$, co może być kosztowne obliczeniowo. Zamiast obliczania tej pochodnej możemy w jej miejsce wstawić przybliżenie:

$$\frac{f(s_n) - f(s_{n-1})}{s_n - s_{n-1}}$$

Wzór na uaktualnienie jest teraz następujący:

$$s_{n+1} = s_n - \frac{f(s_n)(s_n - s_{n-1})}{f(s_n) - f(s_{n-1})}$$

Metodę tę nazywamy *metodą siecznych*. Uzasadnienie jej jest bardzo podobne do uzasadnienia *Reguły Falsi*, metoda ta jednak jest zbieżna z szybkością ponadliniową (ale nie z szybkością kwadratową, jak metoda Newtona). Formalnie metoda siecznych nie jest metodą stałopunktową, gdyż odwołujemy się w niej do dwóch poprzednich wartości s , a nie tylko do wartości ostatniej. Zwróćmy uwagę, że aby metoda ta działała poprawnie, nigdy $f(s_n) - f(s_{n-1})$ nie może być równe zeru.

Gradientowe metody minimalizacji. Wracając do typowego problemu znajdowania ekstremum pewnej funkcji $f(x)$, możemy, jak widzieliśmy przy okazji metody Newtona, skorzystać z gradientu (nachylenia) funkcji w punkcie będącym kandydatem na rozwiązanie. Możemy skorzystać z tej informacji do ukierunkowywania naszych poszukiwań ku lepszym, przynajmniej lokalnie, rozwiązaniom. Istnieje tak wiele metod odwołujących się w jakiś sposób do gradientu, że pełny ich przegląd wymagałby oddzielnej książki. Skupimy się tutaj zatem na głównych zasadach, odsyłając czytelnika do [35] po bardziej szczegółowe informacje na temat odmian tych metod gradientowych.

Podstawowy pomysł polega na tym, żeby znaleźć *pochodną kierunkową*, żebyśmy mogli podążyć w kierunku najbardziej stromego wznoszenia się lub spadku funkcji, w zależności od tego, czy szukamy minimum, czy maksimum. Będziemy tutaj rozważać tylko znajdowanie minimum. Nie będzie to jednak miało wpływu na ogólność metody. Jeśli tylko w punkcie będącym kandydatem na rozwiązanie funkcja jest wystarczająco gładka, taka pochodna kierunkowa istnieje. Naszym zadaniem jest znalezienie kąta w otoczeniu obecnego przybliżenia rozwiązania, dla którego wielkość pochodnej funkcji oceny związana z pewnym rozmiarem kroku s jest maksymalna. Z rachunku różniczkowego wiemy, że takie maksimum występuje w kierunku opisywanym przez ujemny gradient $-\nabla f(\mathbf{x})$ [245]. Przypomnijmy, że

$$\nabla f(\mathbf{x}) = \left[\frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, \dots, \frac{\delta f}{\delta x_n} \right]_{\mathbf{x}}$$

Metoda zstępowania po najbardziej stromym spadku jest zatem następująca: Zaczynamy od kandydata na rozwiązanie \mathbf{x}_k , przy czym $k = 1$. Następnie tworzymy nowe rozwiązanie za pomocą następującej reguły uaktualniania:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla f(\mathbf{x}_k)$$

przy czym $k \geq 1$, wartość $\nabla f(\mathbf{x}_k)$ oznacza gradient w \mathbf{x}_k , α_k zaś rozmiar kroku. Im większe jest α_k , tym mniej *lokalne* jest poszukiwanie. W metodzie uwzględniamy konieczność wykonywania kroków odpowiedniej wielkości – chodzi o to, żeby znaleźć odpowiedni dla wielu prób rozmiar kroku, gwarantujący najlepszą szybkość zmniejszania się kandydatów na rozwiązania.

Jeśli do wzoru na uaktualnienie dodamy informacje o drugiej pochodnej, to otrzymamy równanie uaktualniające, dające w istocie metodę Newtona:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - (H(f(\mathbf{x}_k)))^{-1} \nabla f(\mathbf{x}_k)$$

przy czym $H(f(\mathbf{x}_k))$ jest *macierzą Hessego* (hesjanem) funkcji f ¹:

$$H(f(\mathbf{x}_k)) = \begin{bmatrix} \frac{\delta^2 f}{\delta x_1^2} & \frac{\delta^2 f}{\delta x_1 \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_n} \\ \frac{\delta^2 f}{\delta x_2 \delta x_1} & \frac{\delta^2 f}{\delta x_2^2} & \cdots & \frac{\delta^2 f}{\delta x_2 \delta x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\delta^2 f}{\delta x_n \delta x_1} & \frac{\delta^2 f}{\delta x_n \delta x_2} & \cdots & \frac{\delta^2 f}{\delta x_n^2} \end{bmatrix}_{\mathbf{x}_k}$$

Metoda ta została tak opracowana, żeby w jednym kroku znajdować minimum funkcji kwadratowej. Taka zbieżność jest bardzo szybka! Wypróbujmy to.

Przypuśćmy, że $f(x_1, x_2) = x_1^2 + x_2^2$. Niech naszym początkowym kandydatem będzie $x_1 = [5 \ 1]^T$. Zastosowanie tej metody wymaga obliczania wektora gradientu

$$\nabla f(x_1) = [2x_1 \ 2x_2]^T = [2(5) \ 2(1)]^T = [10 \ 2]^T$$

oraz macierzy Hessego

$$H(f(x_1)) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

Następnie musimy znaleźć odwrotność macierzy Hessego

$$H(f(x_1)) = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

Możemy teraz zapisać wzór uaktualniający i otrzymać następny punkt

$$\mathbf{x}_{k+1} = [5 \ 1]^T - \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} [10 \ 2]^T$$

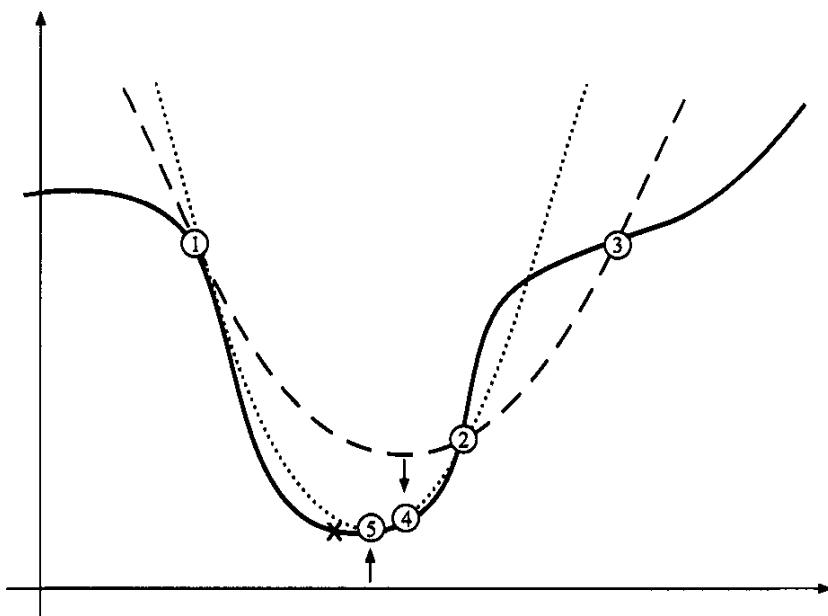
którym okazuje się $[0 \ 0]^T$. Rzeczywiście – jest to minimum. Łatwo jest się przekonać, że dla dowolnego \mathbf{x}_k wzór uaktualniający, o ile funkcja oceny jest kwadratowa, da następne rozwiązanie \mathbf{x}_{k+1} w początku układu współrzędnych. Dla funkcji, które nie są kwadratowe, sprawy nie są tak proste. Pierwsza obserwacja jest taka, że ponieważ w metodzie korzystamy z informacji o gradiencie, możemy utknąć w jakimś optimum lokalnym. Oprócz tego musimy przyjąć jakiś dopuszczalny poziom błędu ϵ i zatrzymywać algorytm, gdy różnica między \mathbf{x}_{k+1} a \mathbf{x}_k jest mniejsza niż ta wartość.

¹Macierz Hessego $H(f(\mathbf{x}_k))$ jest często oznaczana $\nabla^2 f(\mathbf{x}_k)$.

Trzeba przyznać, że obliczanie odwrotności macierzy Hessego rzadko jest tak łatwe jak w naszym przykładzie. W wielu tak zwanych *quasi-newtonowskich* metodach zastosowano metody przybliżania macierzy Hessego. I znowu liczba różnych metod korzystających z tego pomysłu jest ogromna; więcej na ich temat Czytelnik znajdzie w [35]. Jeśli odwrotność macierzy Hessego jest obliczana za pomocą eliminacji Gaussa, to metodę nazywamy metodą Newtona-Gaussa. W dalszej części książki nie będziemy, jeśli nie zostanie zaznaczone, że jest inaczej, rozróżniać metod Newtona-Gaussa i Newtona.

Metoda Newtona-Gaussa i wszystkie metody quasi-newtonowskie są w istocie procedurami przeszukiwania lokalnego. Polegają one na zastępowaniu jednego pełnego rozwiązania innym pełnym rozwiązaniem. W trakcie działania wykorzystują informacje lokalne związane z aktualnym najlepszym punktem przestrzeni przeszukiwania i generują nowy punkt. Dodatkowo działają one do granicy otoczenia punktu zdefiniowanego najczęściej przez odwrotność macierzy Hessego.

Jeszcze jedna nieco pokrewna metoda jest nazywana *metodą Brenta*. Jest oparta na metodzie przedziałowej opracowanej dla problemu znajdowania minimum funkcji kwadratowej. W trakcie działania korzysta z trzech punktów – zamiast z dwóch, jak to miało miejsce w przypadku bisekcji czy *Regula Falsi* – i próbuje dopasować do nich parabolę. Następnie obiera minimum tej paraboli jako następne przybliżenie minimum funkcji. W kolejnych krokach tej metody na podstawie nowego punktu jest zmniejszany przedział. Na rysunku 3.12 przedstawiono działanie tej procedury; kod algorytmu można znaleźć w [363].



Rysunek 3.12. Metoda Brenta to metoda przedziałowa, w której korzysta się z trzech punktów. Żeby znaleźć minimum funkcji, na podstawie tych trzech punktów obliczamy pomocniczą parabolę oraz wyznaczamy jej minimum. Minimum to jest następnie używane jako jeden z trzech punktów definiujących następną parabolę itd. W metodzie tej kończymy pracę, gdy różnica zgadniętych minimów funkcji będzie mniejsza od określonego progu

3.3. Programowanie liniowe: metoda sympleks

W końcu dotarliśmy do ostatniej tradycyjnej metody przetwarzania pełnych rozwiązań – jest to metoda sympleks dla problemów programowania liniowego (ang. *linear programming* – LP). W problemach LP jest wymagane znalezienie ekstremum (powiedzmy maksimum) liniowej kombinacji zmiennych:

$$eval = a_{01}x_1 + \dots + a_{0n}x_n$$

przy ograniczeniach

$$x_1 \geq 0, \dots, x_n \geq 0$$

oraz $M = m_1 + m_2 + m_3$ dodatkowych warunkach, z których m_1 ma postać

$$a_{i1}x_1 + \dots + a_{in}x_n \leq b_i, \quad b_i \geq 0, \quad i = 1, \dots, m_1$$

m_2 ma postać

$$a_{j1}x_1 + \dots + a_{jn}x_n \geq b_j, \quad b_j \geq 0, \quad j = m_1 + 1, \dots, m_1 + m_2$$

a m_3 ma postać

$$a_{k1}x_1 + \dots + a_{kn}x_n = b_k, \quad b_k \geq 0, \quad k = m_1 + m_2 + 1, \dots, M$$

Każdy wektor \mathbf{x} , który spełnia wszystkie ograniczenia, jest określany jako *dopuszczalny*, a naszym zadaniem jest znalezienie takiego wektora dopuszcjalnego, który daje najlepszy wynik wyrażony za pomocą funkcji oceny.

W metodzie sympleks najlepszy wektor \mathbf{x} jest tworzony w ciągu operacji wykorzystujących liniowość problemu. Opisaną tutaj sytuację przedstawiono na rys. 3.13. Potencjalnie na funkcję ewaluacji są nałożone różne ograniczenia w postaci równań i nierówności, w związku z tym jednak, że funkcja jest liniowa, rozwiązanie znajduje się na jednym z wierzchołków widocznego sympleksu (wielokąta obejmującego wszystkie dopuszcjalne rozwiązania) lub w wypadku zdegenerowanym – gdziekolwiek na krawędzi brzegowej. Mamy gwarancję, że procedura sympleks znajduje najlepszy wierzchołek sympleksu.

Metodę tę najlepiej możemy zilustrować, analizując proste zadanie. Przyjrzyjmy się następującemu scenariuszowi. Fabryka produkuje dwa rodzaje przedmiotów: małe fantazyjne krzesła i duże proste stoły. Zysk z jednego krzesła wynosi 20 dolarów, a zysk ze stołu – 30 dolarów. Krzesło wymaga jednej jednostki drewna oraz trzech godzin pracy. Z kolei każdy stół wymaga sześciu jednostek drewna i jednej godziny pracy. Istnieją pewne ograniczenia związane z samym procesem produkcji – w danym czasie dostępnych jest 288 jednostek drewna i 99 godzin pracy. Zadanie polega na maksymalizacji zysku fabryki.

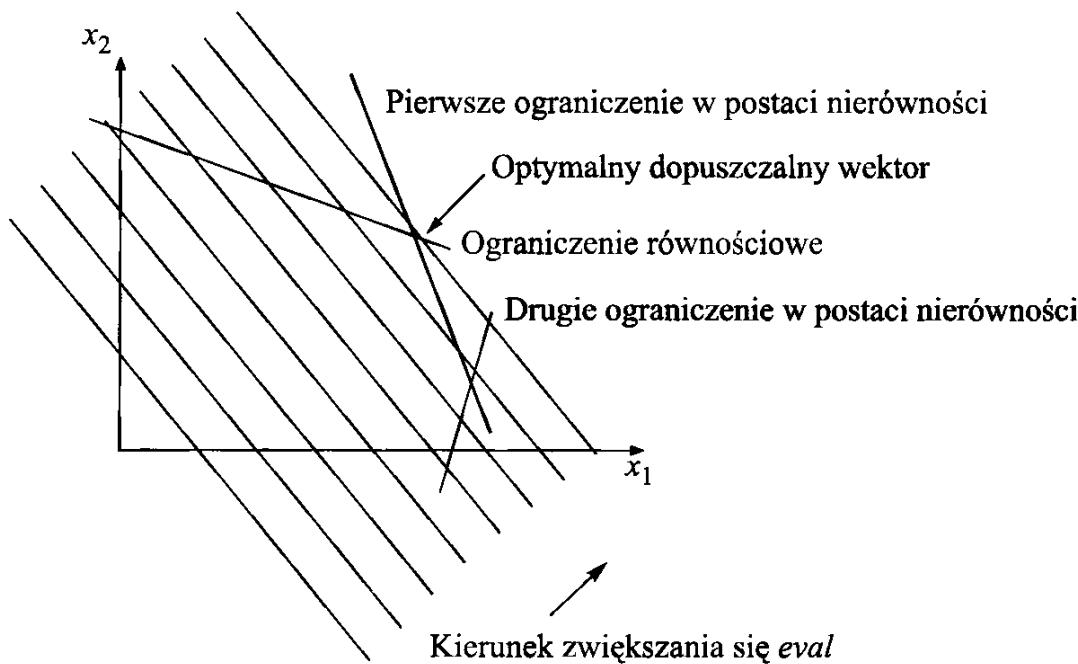
Zadanie to możemy przedstawić jako problem programowania liniowego:

$$\text{zmaksymalizuj } eval = 20x_1 + 30x_2$$

przy założeniu, że

$$x_1 + 6x_2 \leq 288 \text{ (ograniczenie na liczbę jednostek drewna)}$$

$$3x_1 + x_2 \leq 99 \text{ (ograniczenie na liczbę godzin pracy)}$$



Rysunek 3.13. Interpretacja graficzna problemu programowania liniowego. Ukośne poziomice reprezentują wartości funkcji *eval* rosnącej w kierunku pierwszej ćwiartki układu współrzędnych. Pogrubione linie ukazują ograniczenia (albo równościowe, albo w postaci nierówności). Wektor optymalny leży na linii wyznaczonej przez ograniczenie równościowe oraz w wierzchołku sympleksu

Dodatkowo mamy $x_1 \geq 0$ i $x_2 \geq 0$.

Zwróćmy jeszcze uwagę, że powyższe zadanie możemy przekształcić w następujący sposób:

$$\text{zmaksymalizuj } 20x_1 + 30x_2$$

przy założeniu, że:

$$\begin{aligned} x_3 &= -x_1 - 6x_2 + 288 \\ x_4 &= -3x_1 - x_2 + 99 \\ x_1 &\geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0 \end{aligned} \tag{3.1}$$

Następnie tworzymy *tablicę sympleksu*, w której komórki są wypełnione współczynnikami ograniczeń oraz współczynnikami funkcji *eval* w ostatnim wierszu.

	x_1	x_2	x_3	x_4	
x_3	-1	-6	0	0	288
x_4	-3	-1	0	0	99
<i>eval</i>	20	30	0	0	0

Początkowy dopuszczalny punkt uzyskujemy, ustawiając zmienne po prawej stronie warunków (3.1) na zero, a następnie rozwiązujeając układ ze względu na równania po lewej stronie. W tym wypadku otrzymujemy $(x_1, x_2, x_3, x_4) = (0, 0, 288, 99)$.

Główna koncepcja metody sympleks polega na zamienianiu zmiennych po prawej stronie na zmienne po lewej za każdym razem, gdy może to dać poprawienie wyniku funkcji oceny. W naszym przykładzie możemy sprawdzić zmienne x_1 oraz x_2 i zobaczyć, czy zwiększenie którejkolwiek z nich sprawi, że zwiększy się wynik *eval*. Zauważmy, że maksymalna wartość, o jaką możemy zwiększyć x_1 , wynosi 33 (ze względu na ograniczenie $x_4 \geq 0$, równania (3.1)), wskutek czego funkcja ewaluacji zwiększa się o 660. Maksymalna wartość, o jaką możemy zwiększyć x_2 , wynosi 48 (ze względu na ograniczenie $x_4 \geq 0$, równania (3.1)), wskutek czego wartość funkcji ewaluacji zwiększa się o 1440. Zakładamy zatem przypisanie

$$x_2 \leftarrow x_2 + 48$$

Zauważmy, że na tym etapie x_3 staje się zerem i musimy zapisać ponownie (3.1), żeby utrzymać wszystkie „wyzerowane” zmienne (np. x_1 i x_3) po prawej stronie równań. Musimy zatem przepisać pierwotny problem, w którym zmiana x_2 była wyrażona jako funkcja zmiennych x_1 i x_3 . Prowadzi to do nowego problemu:

$$\text{zmaksymalizuj } 15x_1 - 5x_3 + 1440$$

przy założeniu, że

$$\begin{aligned} x_2 &= -\frac{1}{6}x_1 - \frac{1}{6}x_3 + 48 \\ x_4 &= -\frac{17}{6}x_1 + \frac{1}{6}x_3 + 51 \\ x_1 &\geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0 \end{aligned} \tag{3.2}$$

Teraz tablica sympleksu ma postać

	x_1	x_2	x_3	x_4	
x_2	-1/6	0	-1/6	0	48
x_4	-17/6	0	1/6	0	51
<i>eval</i>	15	0	-5	0	1440

Tym razem musimy sprawdzić zmienne x_1 i x_3 , by zobaczyć, czy zwiększenie którejkolwiek z nich zwiększy wartość funkcji *eval*. Jest tak tylko dla x_1 . Zauważmy, że x_1 możemy zwiększyć co najwyżej o 18, ze względu na ograniczenie $x_4 \geq 0$, wzór (3.2). Zatem zapisujemy ponownie

$$x_1 \leftarrow x_1 + 18$$

co prowadzi do nowego problemu

$$\text{zmaksymalizuj } -\frac{100}{17}x_3 - \frac{90}{17}x_4 + 1710$$

przy założeniu, że

$$x_1 = \frac{1}{17}x_3 - \frac{6}{17}x_4 + 18$$

$$x_2 = -\frac{3}{17}x_3 + \frac{1}{17}x_4 + 45$$

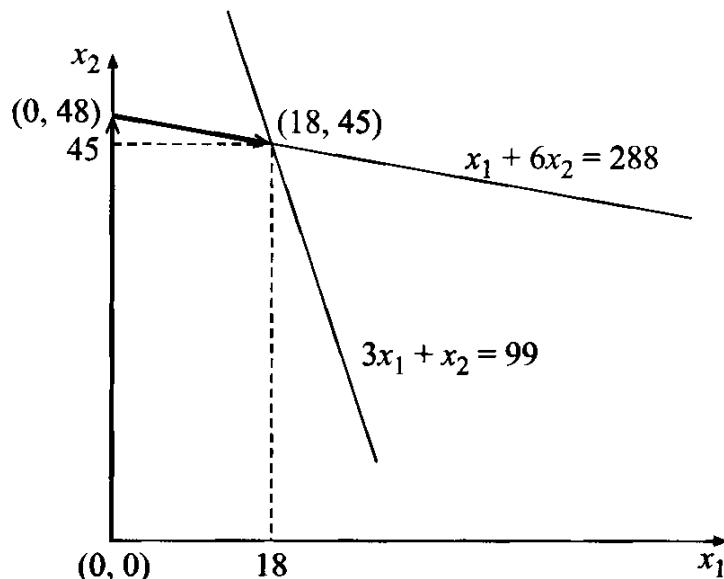
$$x_1 \geq 0, \quad x_2 \geq 0, \quad x_3 \geq 0, \quad x_4 \geq 0$$

Teraz tablica sympleksu ma postać

	x_1	x_2	x_3	x_4	
x_1	0	0	-1/17	-6/17	18
x_4	0	0	-3/17	1/17	45
<i>eval</i>	0	0	-100/17	-90/17	1710

Tym razem zatrzymaliśmy się. Każde zwiększenie x_3 lub x_4 daje zmniejszenie wartości *eval*, a to oznacza, że znaleźliśmy prawidłową odpowiedź: $x_1 = 18$, $x_2 = 45$, dla której *eval* = 1710 (co opisuje nam maksymalny zysk).

Zauważmy, że na ogół skupialiśmy się na wszystkich „zmiennych po prawej stronie”, dla których *eval* mogło ulec zwiększeniu. Proces ten pokazano na rys. 3.14.



Rysunek 3.14. Przechodzenie z jednego wierzchołka do drugiego w metodzie sympleks. *Pogrubione strzałki* odpowiadają krokom procedury. Obszar dopuszczalny leży poniżej dwóch prostych $x_1 + 6x_2 = 288$ i $3x_1 + x_2 = 99$

Istnieje wiele sztuczek umożliwiających przekształcenie problemów programowania liniowego na wymaganą w tej procedurze postać, a także wiele jej wariantów dla różnych przypadków. Podstawowy pomysł polega na tym, aby wykorzystać coś pośredniego między pracą z pełnymi a pracą z cząstkowymi rozwiązaniami. Za pomocą analitycznych sposobów określamy, którą zmienną zwiększyć i przez ciąg zamian z innymi zmiennymi po kolei przeglądamy inne wymiary problemu, sprawdzając, czy nie da się uzyskać jakiegoś ulepszenia. Liniowa natura problemu sprawia, że cała procedura jest wykonalna. Gdy tylko funkcja oceny staje się nieliniowa, musimy uzupełnić metodę o dodatkowe triki, których nie obejmuje nasz wykład. Wreszcie, gdy funkcja oceny może mieć potencjalnie wiele ekstremów lokalnych lub wręcz, gdy jest nieciągła, metoda ta zawodzi. Jedyne rozwiązanie, które możemy ewentualnie zastosować, to spróbować wykonać liniowe lub kwadratowe przybliżenie problemu albo też ograniczyć

się tylko do liczb całkowitych. Po wykonaniu takiego ruchu będziemy jednak szukać dobrego rozwiązania dla złego problemu, nie mając gwarancji, co do tego, na ile zły będzie uzyskany wynik.

3.4. Podsumowanie

Istnieje wiele klasycznych algorytmów działających na pełnych rozwiązaniach. Każdą z tych procedur możemy przerwać w dowolnym momencie i dostarczy ona jakieś potencjalne rozwiązanie. Jakość tego rozwiązania zależy jednak w dużej mierze od tego, do jakiego stopnia procedura pasuje do problemu. Jeśli, na przykład, nasze zadanie polega na znalezieniu maksimum funkcji oceny, która jest liniową kombinacją zmiennych, na które nałożono dodatkowe ograniczenia wyrażone za pomocą równań lub nierówności liniowych, to odpowiednie są metody programowania liniowego. Jeśli funkcja oceny jest gładka (tzn. różniczkowalna), ma jedno ekstremum i chcemy znaleźć jej minimum, to przydatne są metody gradientowe oraz oparte na funkcjach wyższego rzędu. Jeśli mamy do czynienia z problemem kombinatorycznym takim jak TSP, to istnieje wiele operatorów lokalnych, których możemy użyć, żeby wykorzystać własności grafów na płaszczyźnie euklidesowej. Operatory te krok po kroku ulepszają pełne trasy i mogą szybko doprowadzić do takiej, która jest bliska optymalnej.

Wada, której musimy być świadomi, polega jednak na tym, że w rzeczywistości nie wiemy za wiele o problemie lub też, jeśli problem nie pasuje dobrze do żadnej z klas obsługiwanych przez te algorytmy, pozostaje nam tylko przejrzenie wszystkich możliwych rozwiązań. Dla wszystkich występujących w praktyce problemów takie podejście jest prawie zawsze niepraktyczne, gdyż liczba możliwych rozwiązań jest bardzo duża.

Wspomnieliśmy, że istnieją dwa rodzaje klasycznych algorytmów: 1) działające na pełnych rozwiązaniach i 2) obliczające rozwiązania częściowe lub przybliżone. Gdy metody opisane w tym rozdziale zawiodą, warto czasem pomyśleć o drugiej ze wskazanych klas i spróbować podzielić problem na prostsze podproblemy, które potem możemy złożyć w całość lub też, być może, rozwiązać skomplikowany problem za pomocą ciągu prostszych etapów. Tego typu metody są szczegółowo opisane w następnym rozdziale.

IV. Jakie to liczby?

Rozważmy zagadkę, która pokazuje, że proces znajdowania wszystkich rozwiązań niektórych problemów nie jest wcale prosty. Problem ten dostarczył nam Peter Ross z University of Edinburgh. Jest to pozornie łatwe zadanie: znajdź takie liczby naturalne a , b , c i d , że

$$\begin{aligned} ab &= 2(c + d) \\ cd &= 2(a + b) \end{aligned}$$

Jednym z rozwiązań tego zadania jest na przykład $(a, b) = (1, 54)$ i $(c, d) = (5, 22)$. Niektóre z rozwiązań są jednak dużo mniej trywialne.

Bardzo ważne jest przeanalizowanie tego zadania przed próbą rozwiązania go. Dobrym pomysłem jest wzięcie czterech losowych liczb całkowitych i zobaczenie, czy sprawdzają się dla nich powyższe równania. Szansa na to jest znikoma! Możemy jednak następnie zmieniać za każdym razem jedną z liczb, żeby „wczuć” się w problem. Szybko odkryjemy, że wszystkie te liczby nie mogą być zbyt duże. Jeśli a , b , c i d są odpowiednio duże, to iloczyn a i b jest dużo większy niż ich podwojona suma. To samo dotyczy c i d . Na przykład: jeśli $a \geq 5$ i $b \geq 5$, to

$$\begin{aligned} ab &= (5 + a')(5 + b') = 25 + 5a' + 5b' + a'b' = (5 + 3a' + 3b' + a'b') + \\ &\quad + (2a' + 10) + (2b' + 10) > (2a' + 10) + (2b' + 10) = 2a + 2b \end{aligned}$$

dla dowolnego $a' \geq 0$ i $b' \geq 0$ (przyjęliśmy $a = 5 + a'$ i $b = 5 + b'$). Ta prosta obserwacja prowadzi do następującego wniosku: jeśli wszystkie liczby a , b , c i d wynoszą 5 lub więcej, to

$$ab > 2(a + b) = cd > 2(c + d) = ab$$

co daje sprzeczność!

Przy założeniu, że $a = \min\{a, b, c, d\}$, wystarczy rozważyć jedynie cztery możliwości, a mianowicie $1 \leq a \leq 4$. Jeśli $a = 1$, problem sprowadza się do:

$$\begin{aligned}b &= 2(c + d) \\cd &= 2(1 + b) = 2 + 2b\end{aligned}$$

Jak widać, b musi być parzyste, więc $b = 2b'$ i mamy:

$$\begin{aligned}b' &= c + d \\cd &= 2 + 4b'\end{aligned}$$

Zastępując b' w drugim równaniu, otrzymujemy

$$cd = 2 + 4c + 4d$$

co jest łatwe do rozwiązania:

$$c = \frac{4d + 2}{d - 4} = 4 + \frac{18}{d - 4}$$

Ponieważ c jest liczbą naturalną, d musi wynosić 5, 6, 7, 10, 13 lub 22. Otrzymujemy dla nich trzy pary (symetrycznych) rozwiązań:

$$(c, d) = (5, 22); \quad (c, d) = (6, 13) \text{ lub } (c, d) = (7, 10)$$

Te trzy rozwiązania dają odpowiednio $b = 54$, $b = 38$ i $b = 34$. W ten sposób znaleźliśmy pierwsze trzy rozwiązania problemu:

$$\begin{aligned}(1, 54), (5, 22) \\(1, 38), (6, 13) \\(1, 34), (7, 10)\end{aligned}$$

Druga możliwość jest dla $a = 2$. Problem sprowadza się wtedy do:

$$\begin{aligned}2b &= 2(c + d) \\cd &= 2(2 + b)\end{aligned}$$

co upraszcza się do:

$$\begin{aligned}b &= c + d \\cd &= 4 + 2b\end{aligned}$$

Podobnie jak poprzednio, zastępując b w drugim równaniu, otrzymujemy

$$cd = 4 + 2c + 2d$$

co daje nam

$$c = \frac{2d + 4}{d - 2} = 2 + \frac{8}{d - 2}$$

Tak jak poprzednio, d musi wynosić 3, 4, 6 lub 10. Otrzymujemy dla nich dwie pary symetrycznych rozwiązań:

$$(c, d) = (3, 10) \text{ i } (c, d) = (4, 6)$$

Te dwa rozwiązania dają odpowiednio $b = 13$ i $b = 10$. W ten sposób znaleźliśmy dwa dodatkowe rozwiązania problemu:

$$\begin{aligned} &(2, 13), (3, 10) \\ &(2, 10), (4, 6) \end{aligned}$$

Trzecia możliwość do rozważenia jest dla $a = 3$. Wartość b musi być wówczas parzysta ($b = 2b'$, ponieważ iloczyn a i b jest parzysty) i mamy:

$$\begin{aligned} 3b' &= c + d \\ cd &= 6 + 4b' \end{aligned}$$

W tym wypadku, po zamianie b' w drugim równaniu, otrzymujemy

$$c = \frac{4d + 18}{3d - 4} = 1 + \frac{d + 22}{3d - 4}$$

Widzimy, że $d > 6$ daje $c < 3$ (ponieważ $(d + 22)/(3d - 4) < 2$), co daje sprzeczność (gdyż $a = 3 < c$). Tak więc musimy już tylko rozważyć możliwości dla d równego 3, 4, 5 i 6. Pierwsza i ostatnia możliwość dają takie samo (tzn. symetryczne) rozwiązanie:

$$(c, d) = (3, 6)$$

co daje $b = 6$. I tak znaleźliśmy szóste rozwiązanie problemu:

$$(3, 6), (3, 6)$$

Ostatnia możliwość jest dla $a = 4$. Mamy wówczas:

$$\begin{aligned} 4b &= 2(c + d) \\ cd &= 2(4 + b) \end{aligned}$$

co prowadzi do

$$cd = 8 + c + d$$

Tak więc

$$c = \frac{d + 8}{d - 1} = 1 + \frac{9}{d - 1}$$

Ponieważ $d \geq a = 4$, więc d wynosi 4 lub 10. Druga z tych wartości daje $c = 2$, co jest sprzecznością, podczas gdy $d = 4$ daje $c = 4$ i $d = 4$. Ta możliwość przynosi ostatnie rozwiązanie problemu:

$$(4, 4), (4, 4)$$

co oznacza, że problem ma w sumie siedem rozwiązań.

Ważne jest, żeby cały czas mieć na uwadze sposób, w jaki została zmniejszona przestrzeń przeszukiwania w tym zadaniu. „Bawienie” się problemem może dać nam dobry wgląd w jego naturę i możliwość znalezienia sposobów na rozważenie tylko części możliwych rozwiązań. Najważniejsze to nie poddawać się! I nie zgadzać się na niedoskonałe rozwiązanie, jeśli nie ma takiej potrzeby.

Możesz te umiejętności natychmiast poćwiczyć. Kolejną podobną zagadkę otrzymaliśmy od Antoniego Mazurkiewicza z Polskiej Akademii Nauk.

Zadanie polega na znalezieniu takich liczb naturalnych a , b i c , że:

$a + b + 1$ jest podzielne przez c

$a + c + 1$ jest podzielne przez b

$b + c + 1$ jest podzielne przez a

Zadanie to ma 12 rozwiązań. Jednym z nich jest na przykład

(21, 14, 6)

gdyż:

$12 + 14 + 1 = 36$, które dzieli się przez 6

$21 + 6 + 1 = 28$, które dzieli się przez 14

$14 + 6 + 1 = 21$, które oczywiście dzieli się przez 21

Spróbuj znaleźć wszystkich 12 rozwiązań!

Jeśli wolisz coś łatwiejszego, możesz spróbować rozwiązać zagadkę, w której dwa „fałsze” dają „prawdę”! Zagadka ta:

$$\begin{array}{r} \text{F} \quad \text{A} \quad \text{Ł} \quad \text{S} \quad \text{Z} \\ \text{F} \quad \text{A} \quad \text{Ł} \quad \text{S} \quad \text{Z} \\ \hline \text{P} \quad \text{R} \quad \text{A} \quad \text{W} \quad \text{D} \quad \text{A} \end{array}$$

ma kilka rozwiązań (zakładamy dodawanie). Jednym z nich jest

$$\begin{array}{r} 9 \quad 0 \quad 2 \quad 3 \quad 5 \\ 9 \quad 0 \quad 2 \quad 3 \quad 5 \\ \hline 1 \quad 8 \quad 0 \quad 4 \quad 7 \quad 0 \end{array}$$

Zapamiętaj to, gdyż może Ci się przydać!

4. Metody tradycyjne – część II

– Dwa i dwa to cztery
trzy i trzy to sześć
powtarzajcie: sześć –
mówi nauczyciel

Jacques Prévert: *Ptak*¹

Zobaczyliśmy już mnóstwo metod, które działają na podstawie pełnych rozwiązań. Przyjrzyjmy się teraz algorytmom, które działają na bazie częściowych lub niepełnych rozwiązań i tworzą rozwiązania „po kawałku”. Zaczniemy od najlepiej znanej klasy tego typu algorytmów – od algorytmów zachłannych.

4.1. Algorytmy zachłanne

Algorytmy zachłanne atakują problem, tworząc pełne rozwiązanie za pomocą ciągu kroków. Ich popularność wynika z prostoty. Ogólna koncepcja kryjąca się za podejściem zachłannym jest zadziwiająco łatwa – należy przypisywać wartości wszystkim zmiennym problemu po kolej, za każdym razem podejmując możliwie najlepszą decyzję. W podejściu tym założono istnienie heurystyki podejmowania decyzji, która daje w każdym kroku najlepszy ruch, najlepszy „zysk”. Stąd pochodzi też nazwa *zachłanne*. Jest jednak jasne, że takie podejście jest także krótkowzroczne, gdyż podejmowanie najlepszej decyzji w każdym poszczególnym kroku nie musi zawsze prowadzić do globalnego optymalnego rozwiązania.

4.1.1. Algorytmy zachłanne i SAT

Przyjrzyjmy się kilku przykładom, zaczynając od problemu SAT. Będziemy przypisywali zmiennym wartości boolowskie (TRUE i FALSE), potrzebujemy jednak pewnej heurystyki, która będzie nas prowadziła przez ten proces podejmowania decyzji. Zastanówmy się nad następującym podejściem:

¹ *Spacer Picasso i inne wiersze*, przekład Stanisław Baliński, wydane nakładem Polskiej Fundacji Kulturalnej, Londyn, około 1968 (przyp. tłum.).

- Każdej zmiennej od 1 do n , w dowolnej ustalonej kolejności, przypiszmy wartość boolowską, która prowadzi do spełnienia największej liczby aktualnie niespełnionych klauzul. Jeśli wychodzi więcej niż jedna najlepsza możliwość, wybierzmy jedną z nich losowo.

Ta heurystyka jest właśnie zachłanna – w każdym kroku próbujemy zapewnić spełnienie największej liczby aktualnie niespełnionych klauzul! Możemy łatwo wykazać, że wyniki stosowania takiej niekontrolowanej postaci zachłanności są dosyć kiepskie nawet dla pewnych prostych zadań. Rozważmy, na przykład, następujący przypadek:

$$\bar{x}_1 \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4)$$

Jeśli zajmiemy się najpierw zmienną x_1 , to możemy przyjąć, że $x_1 = \text{TRUE}$, gdyż takie podstawienie sprawi, że zostaną spełnione trzy klauzule. Pierwsza klauzula nie może być jednak spełniona dla $x_1 = \text{TRUE}$, a zatem cały wysiłek algorytmu skierowany na analizę tej sytuacji zostanie zmarnowany. Przy żadnym przypisaniu wartości do x_2 , x_3 i x_4 formuła nie zostanie spełniona, jeśli x_1 pozostanie TRUE .

Możemy łatwo określić, w czym tkwi problem dla opisanego podejścia zachłannego. Jest ono zbyt zachłanne w tym sensie, że nie przyłożyliśmy wystarczającej wagi do kolejności, w jakiej powinny być rozważane zmienne. Lepsze wyniki uzyskalibyśmy, korzystając z dodatkowej heurystyki wyboru ich kolejności. W szczególności moglibyśmy zacząć od tych zmiennych, które występują rzadziej (tak jak x_2 , x_3 i x_4 w powyższej formule), zostawiając częściej występujące (jak x_1) na później. Tak więc następny, poprawiony algorytm zachłanny rozwiązywania SAT mógłby wyglądać tak:

- uporządkuj zmienne w zależności od częstości ich występowania w klauzulach – od najrzadziej do najczęściej występujących;
- kolejnym zmiennym, branym w tym porządku, przypisz wartość, która spowoduje spełnienie największej liczby aktualnie niespełnionych klauzul; gdy każda z wartości daje ten sam wynik, podejmij dowolną decyzję.

Po dodatkowym namyśle nad tym algorytmem możemy odkryć pewne jego słabości. Przyjrzyjmy się bowiem następującemu przypadkowi:

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge F$$

przy czym dodatkowo zakładamy, że formuła F nie zawiera zmiennych x_1 oraz x_2 , ale zawiera wiele wystąpień pozostałych zmiennych. I tak x_1 występuje tylko w trzech klauzulach, x_2 zaś w czterech, a wszystkie pozostałe zmienne mają większe częstości występowania. Nasz ostatni algorytm zachłanny przypisze x_1 wartość TRUE , gdyż takie podstawienie zapewnia spełnialność dwóch klauzul. Podobnie do x_2 przypisze wartość TRUE , ponieważ da to spełnienie dwóch

dodatkowych klauzul. W tej sytuacji nie będzie jednak możliwe spełnienie trzeciej i czwartej klauzuli:

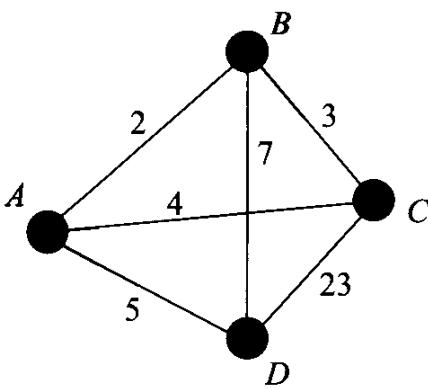
$$(\bar{x}_1 \vee x_4) \wedge (\bar{x}_2 \vee \bar{x}_4)$$

i algorytm nie znajdzie wymaganego podstawienia.

Jeśli nie chcemy być już tutaj pokonani, istnieje dużo miejsca na dalsze „ulepszenia”. Możemy wprowadzić dodatkową regułę, która będzie zabraniała szczególnego podstawienia, jeśli spowoduje ono, że jakaś klauzula będzie miała wartość FALSE. Możemy też rozważaćczęstość występowania zmiennych w *pozostałych* (tzn. do tej pory niespełnionych) klauzulach. Możemy też próbować być bardziej wyrafinowani. Zamiast porządkować zmienne wyłącznie na podstawie częstości ich występowania, możemy brać pod uwagę długość klauzuli. Intuicja tu jest taka, że wystąpienie zmiennej w krótkiej klauzuli ma większe znaczenie niż w długiej. Żadne z tych podejść nie prowadzi do algorytmu, który zadziała dla problemu SAT. *Nie ma algorytmu zachłanego dla SAT!*

4.1.2. Algorytmy zachłanne i TSP

Najbardziej intuicyjny algorytm zachłanny dla TSP jest oparty na heurystyce najbliższego sąsiada: zaczynamy od dowolnego miasta, przechodzimy do najbliższego nieodwiedzonego miasta i kontynuujemy dopóty, dopóki wszystkie miasta nie zostaną odwiedzone, po czym wracamy do pierwszego miasta. Taka trasa może jednak być daleka od doskonałości. Zwykle płacimy dużą cenę za zachłanne wybory na początku procesu szukania trasy. Prosty przykład ilustrujący tę sytuację pokazano na rys. 4.1. Zaczynamy od miasta A i za pomocą heurystyki zachłannej tworzymy trasę $A - B - C - D - A$ o całkowitym koszcie $2 + 3 + 23 + 5 = 33$, choć koszt trasy $A - C - B - D - A$ wynosi zaledwie $4 + 3 + 7 + 5 = 19$.



Rysunek 4.1. Przykład zadania TSP dla czterech miast. Zwróćmy uwagę, że koszty krawędzi nie odpowiadają widocznym na rysunku odległościom między miastami

Innym zachłannym sposobem jest tworzenie trasy przez wybieranie najkrótszych dostępnych połączeń (tj. krawędzi) między miastami, unikając przy tym sytuacji, w których jedno miasto jest dwa razy odwiedzane. Innymi słowy, zaczynamy od znalezienia najtańszego połączenia między dowolnymi parami

miast, a następnie dobieramy drugie co do taniości itd. Jeśli dobierzemy już połączenia między, powiedzmy, miastami $A - B$ i $B - E$, to nie możemy już wybrać żadnego więcej połączenia z miastem B . Zauważmy, że takie zachłanne podejście może dla przykładu z rys. 4.1 dać tę samą odpowiedź co poprzednia zachłanna taktyka.

Jeszcze jedno zachłanne podejście [73] jest oparte na następującym pomyśle. Wybieramy jedno miasto oraz tworzymy początkową „trasę”¹ w ten sposób, że zaczynamy od wybranego miasta, przechodzimy do pewnego nieodwiedzonego miasta, po czym wracamy do wybranego miasta, przechodzimy do innego nieodwiedzonego itd. Innymi słowy, początkowa trasa jest ścieżką składającą się z jednakowych segmentów prowadzących od wybranego miasta do jakiegoś innego i z powrotem. Następnie – tutaj mają miejsce zachłanne wybory – rozważamy kolejne pary miast (omijając wybrane miasto). Znajdujemy taką parę miast (powiedzmy i oraz j), dla której uzyskamy największą oszczędność, przechodząc od i do j bezpośrednio, a nie przez wybrane miasto. W algorytmie tym przeglądamy pary niewybranych miast w nierosnącym porządku dawanych przez nie oszczędności i, jeśli to możliwe, poprawiamy początkową trasę (zaszczepiamy się konieczność unikania przedwcześniejszych cykli). Metoda ta nie zadziała dla prostego przykładu z rys. 4.1. Dla każdej prostej heurystyki, którą moglibyśmy wymyślić, możemy znaleźć jakiś patologiczny układ, który sprawi, że będzie wyglądała niedorzecznie.

4.1.3. Algorytmy zachłanne i NLP

Zajmijmy się teraz ostatnim przypadkiem – NLP. Nie istnieją naprawdę efektywne algorytmy zachłanne dla NLP, możemy jednak opracować algorytm, który ma pewne cechy tego podejścia. Optymalizując funkcję o, na przykład, dwóch zmiennych x_1 i x_2 , moglibyśmy ustalić jedną ze zmiennych, powiedzmy, x_1 i zmieniać x_2 do momentu osiągnięcia optimum. Następnie moglibyśmy ustalić x_2 na tak uzyskanej wartości i zmieniać x_1 do momentu osiągnięcia optimum itd. Takie *szukanie po prostych* można uogólnić na n wymiarów, ale jak stwierdził Himmelblau [219]:

Ten proces działa kiepsko, jeśli wzajemne oddziaływanie między x_1 a x_2 jest duże, to znaczy, jeśli w badanej funkcji występują dominujące wyrażenia zawierające iloczyn x_1 i x_2 . A zatem metoda ta nie może być zalecana, chyba że użytkownik dysponuje funkcją [oceny], w której takie oddziaływanie są nieznaczące.

Ścisłe mówiąc, szukanie po prostych nie jest metodą zachłanną, gdyż oceniamy tutaj tylko pełne rozwiązania. Niemniej jednak koncepcja wybierania najlepszej dostępnej możliwości ze względu na jeden wymiar na raz przypomina ogólny sposób postępowania w algorytmach zachłannych.

¹Ta początkowa trasa nie jest trasą w sensie TSP, gdyż wybrane miasto jest w niej odwiedzane wiele razy.

Metody zachłanne, czy to zastosowane do problemów SAT, TSP, NLP, czy też do innych zagadnień, są w swej istocie proste, ale zwykle za tę prostotę musimy zapłacić tym, że nie dają zadowalających rozwiązań złożonych problemów, których parametry między sobą oddziałują. A właśnie z takimi problemami często mamy do czynienia w świecie rzeczywistym.

4.2. Dziel i rządź

Czasami dobrym pomysłem na rozwiązywanie wydawałoby się skomplikowanego problemu jest podzielenie go na mniejsze i prostsze zadania. Możemy umieć rozwiązać każdy z tych łatwiejszych problemów oddzielnie i znaleźć sposób na złożenie pełnej odpowiedzi z uzyskanych w ten sposób części. Takie podejście typu „dziel i rządź” jest efektywne, jeśli czas i wysiłek potrzebne na wykonanie podziału, rozwiązywanie małych zagadnień i złożenie odpowiedzi jest mniejszy niż koszt rozwiązywania problemu w pierwotnej postaci z całą jego wewnętrzną złożonością. Podczas składania rozwiązywania z mniejszych kawałków należy też zwrócić uwagę, czy rzeczywiście otrzymujemy odpowiedź, której szukaliśmy. Czasami szansa na złożenie pełnego rozwiązania znika po podzieleniu problemu.

Ogólny schemat podejścia dziel i rządź pokazano na rys. 4.2.

```

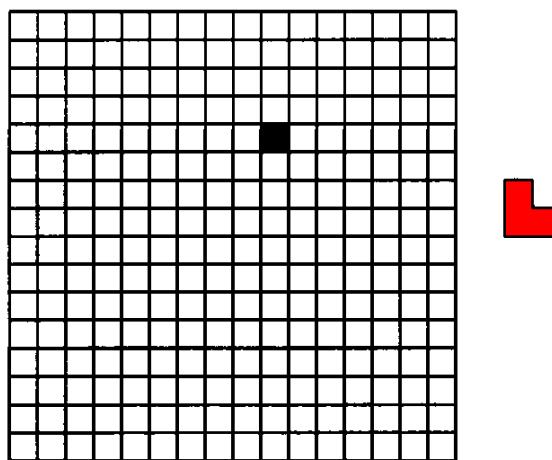
procedure DiR ( $P$ )
begin
    rozbij problem  $P$  na podproblemy  $P_1, \dots, P_k$ 
    for  $i = 1$  to  $k$  do
        if  $\text{rozmiar}(P_i) < \rho$  then rozwiąż  $P_i$  (otrzymując  $s_i$ )
            else  $s_i \leftarrow \text{DiR} (P_i)$ 
        połącz wszystkie  $s_i$  w końcowe rozwiązanie
    end
```

Rysunek 4.2. Procedura rekurencyjna typu dziel i rządź rozwiązuje problem P

Początkowy problem P jest zastępowany grupą podproblemów, z których każdy jest dalej rozkładany na pod-podproblemy itd. (często proces ten jest organizowany rekurencyjnie). Proces ten trwa do momentu, aż problemy staną się na tyle trywialne (tzn. mniejsze niż pewna stała ρ występująca w schemacie z rys. 4.2), że będzie je można rozwiązać „ręcznie”. Następnie algorytm wykonuje spiralę powrotną, składając rozwiązań w rozwiązanie większych podproblemów.

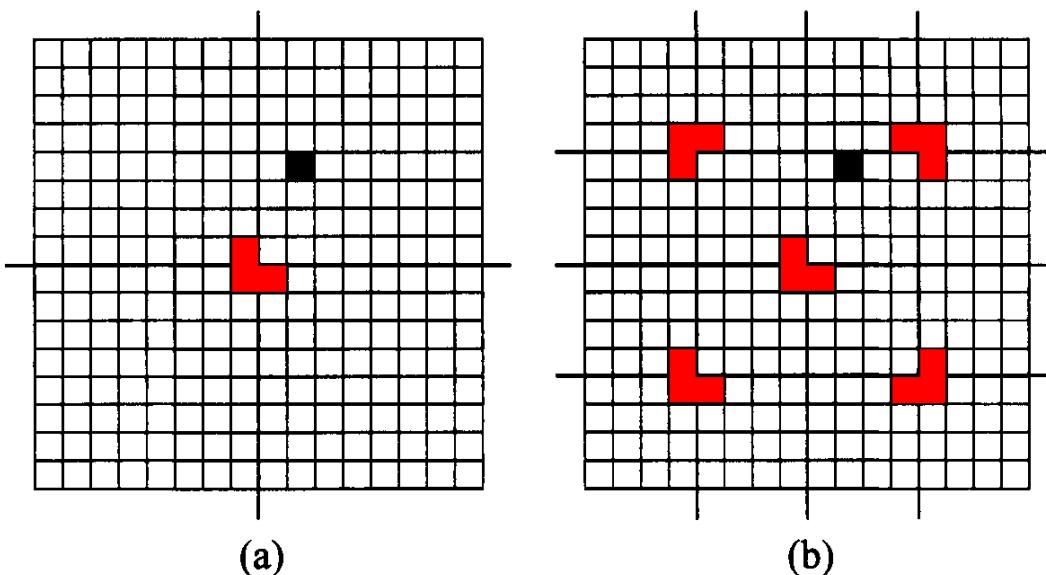
Zasada dziel i rządź jest nierozerwalnie związana z wieloma algorytmami sortowania, takimi jak sortowanie szybkie (ang. *quicksort*) czy sortowanie przez łączenie (ang. *mergesort*). Stosując tę metodę, można też wykonać mnożenie wielomianów i macierzy. Jako klarowny przykład użycia tego podejścia rozważmy następujący problem. Mamy szachownicę o boku 2^m (tj. składa się ona z 2^{2m} pól) z jedną dziurą w środku, tzn. dowolne pole zostało usunięte.

Dysponujemy dodatkowo pewną liczbą kostek w kształcie litery L (rys. 4.3); zadanie polega na pokryciu planszy tymi kostkami. (Kierunek ustawienia kostek nie jest istotny).



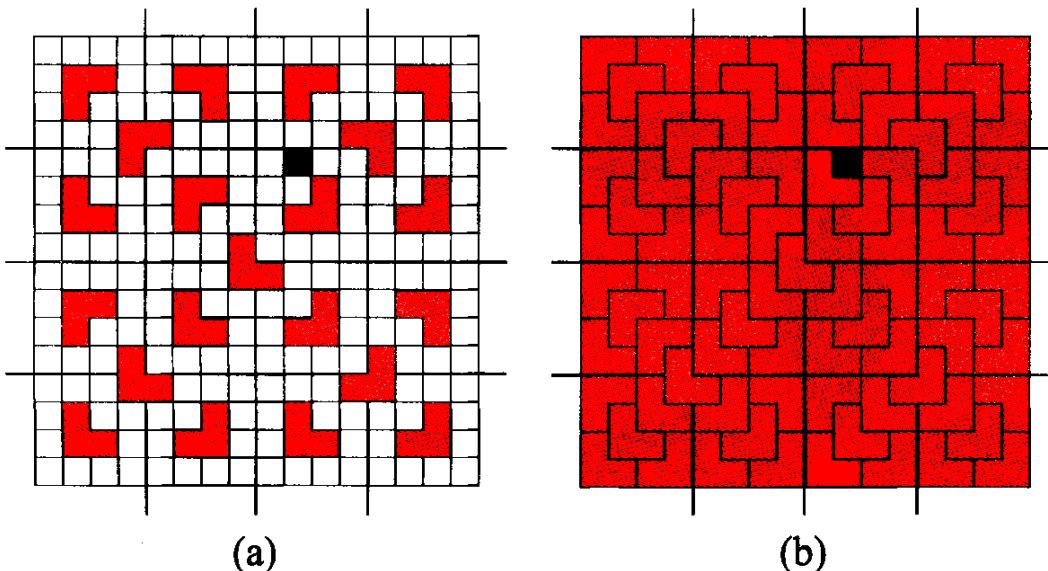
Rysunek 4.3. Szachownica $2^m \times 2^m$ ($m = 4$) i kostka

W tym wypadku przestrzeń przeszukiwania składa się z wielu możliwych ułożen kostek na szachownicy, a my musimy znaleźć to prawidłowe ułożenie. Możemy zastosować wiele podejść, ale metoda dziel i rządź jest tutaj idealna. Możemy po prostu podzielić planszę na cztery równe obszary – dziura znajdzie się w jednym z nich. Możemy teraz umieścić pierwszą kostkę w taki sposób, że pokryje trzy pola – po jednym z każdego obszaru, który nie zawiera początkowej dziury (rys. 4.4a). Dzięki temu zamiast jednego kwadratu (z dziurą) mamy teraz cztery mniejsze kwadraty. Każdy z nich ma dziurę albo tę pierwotną, albo utworzoną przez umieszczenie kostki. W ten sposób możemy kontynuować ten proces, dzieląc każdy z mniejszych kwadratów na cztery jeszcze mniejsze kwadraty i umieszczając nowe kostki tak, że te nowe mniejsze kwadraty mają dokładnie po jednej „dziurze” (rys. 4.4b).



Rysunek 4.4. Położenie pierwszej kostki (a) oraz następnych czterech kostek (b)

Zwróćmy uwagę, że każdy kwadrat (mają one teraz rozmiar 4×4) ma dokładnie jedno brakujące pole, tj. jedną dziurę. Kontynuując tę procedurę, możemy każdy z kwadratów podzielić na cztery kwadraty 2×2 , a po odpowiednim umieszczeniu kostek każdy z tych nowych małych kwadratów będzie miał dokładnie jedną dziurę (rys. 4.5a). W tym momencie mamy przed sobą najłatwiejsze na świecie zadanie: umieścić kostkę w kwadracie 2×2 mającym jedno brakujące pole! W ten sposób łatwo otrzymujemy ostateczne rozwiązanie problemu (rys. 4.5b).



Rysunek 4.5. Rozmieszczenie następnych 16 kostek (a) oraz ostateczne rozwiązanie (b)

4.3. Programowanie dynamiczne

Programowanie dynamiczne działa w ten sposób, że znajduje pełne rozwiązanie na podstawie pośredniego punktu, który znajduje się między naszym obecnym miejscem pobytu a celem, do którego dążymy¹. Procedura ta ma charakter rekurencyjny w tym sensie, że każdy następny punkt pośredni jest obliczany jako funkcja już obliczonych punktów. Problem nadający się do rozwiązywania metodą programowania dynamicznego ma następujące własności:

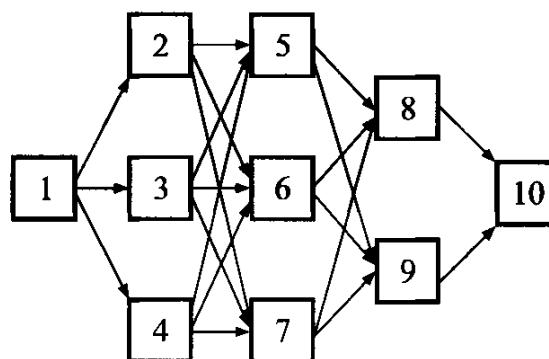
- można go rozłożyć na ciąg decyzji podejmowanych na różnych etapach;
- każdy etap ma pewną liczbę możliwych stanów;
- podjęcie decyzji oznacza przejście z jednego stanu na pewnym etapie do innego stanu na następnym etapie;

¹Określenie *dynamiczne* zostało użyte, żeby wskazać, że podejście to jest przydatne w problemach, „w których ważną rolę odgrywa czas i w których kolejność wykonywania operacji może być istotna” [40].

- najlepszy ciąg decyzji (zwany też *strategią*) na dowolnym etapie nie zależy od decyzji podjętych na wcześniejszych etapach;
- istnieje dobrze określony koszt przejścia z jednego stanu do drugiego; co więcej, istnieje rekurencyjna zależność umożliwiająca określenie najlepszej możliwej decyzji.

Metodę tę możemy zastosować w ten sposób, że zaczynamy od celu i cofamy się krok po kroku do stanu bieżącego. Innymi słowy, możemy najpierw określić, jak wygląda najlepsza decyzja na ostatnim etapie. Na tej postawie określamy najlepszą decyzję na przedostatnim etapie, mając na uwadze, jak wygląda najlepsza decyzja na ostatnim itd.

Najprostszym przykładem jest *problem dyliżansu* opracowany przez Harveya Wagnera [217]. W pierwotnym opisie tego problemu powiedziano bardzo dużo na temat zmieniania zasłonek w dyliżansie przez komiwojażera przy przeprowadzaniu dyliżansu przez terytoria wrogo nastawionych Indian amerykańskich; darujemy sobie te opisy i przejdziemy od razu do sedna.



Rysunek 4.6. Diagram przepływu wskazujący wybory komiwojażera przeprowadzającego dyliżans przez wrogie terytorium

Komiwojażer musi zacząć od swojej bieżącej pozycji i dostać się do ustalonego celu. W zasadzie ma on trzy etapy, na których może dokonać wyboru drogi (rys. 4.6). Na pierwszym z nich ma do dyspozycji trzy ścieżki. Podobnie na drugim etapie ma znowu trzy ścieżki. Wreszcie na trzecim ma dwie. Jest jeszcze czwarty etap, ale tam nie ma żadnej możliwości wyboru. Koszty przejścia z każdego ze stanów na każdym etapie do wszystkich możliwych następnych stanów pokazano na rys. 4.7. Zadanie polega na znalezieniu ścieżki o najmniejszym koszcie, biegającej z pierwszego stanu (1) do stanu ostatniego (10).

Przed przystąpieniem do omawiania sposobu dojścia do odpowiedzi musimy jeszcze pewne sprawy wyjaśnić.

- Rozważamy n etapów; decyzja, do którego stanu należy dalej przejść, jest oznaczona x_n . W naszym przykładzie $n = 4$.
- Wyrażenie $f_n(s, x_n)$ oznacza koszt najlepszego ciągu decyzji (strategii) na wszystkich pozostałych etapach, przy założeniu że komiwojażer jest w stanie s , na etapie n i wybrał x_n jako następny stan.

	2	3	4		5	6	7
1	2	4	3	2	7	4	6
				3	3	2	4
				4	4	1	5
	8	9			10		
5	1	4		8	3		
6	6	3		9	4		
7	3	3					

Rysunek 4.7. Koszt wykonania każdego z możliwych wyborów na każdym etapie decyzyjnym. Wiersz oznacza stan bieżący, kolumna zaś stan następny. Cztery tabelki odpowiadają czterem etapom. Zwróćmy uwagę, że tylko trzy etapy dają możliwość wyboru stanu

- Wyrażenie $x_n^*(s)$ oznacza wartość x_n , która minimalizuje $f_n(s, x_n)$, a $f_n^*(s)$ oznacza odpowiedni minimalny koszt.
- Celem jest znalezienie $f_1^*(1)$, ponieważ komiwojażer zaczyna w stanie 1. Uzyskujemy to, znajdując $f_4^*(s)$, $f_3^*(s)$, $f_2^*(s)$ i wreszcie $f_1^*(1)$.

Czym zatem jest $f_4^*(s)$? Dla jakiego x_4 wartość $f_4(s, x_4)$ jest minimalna? Jest to, prawdę mówiąc, podchwytliwe pytanie, gdyż na czwartym etapie jest tylko jeden możliwy stan do wybrania: $x_4 = 10$. Łatwo jest zatem obliczyć poniżej przedstawione wartości, czym kończymy pierwszą fazę procedury programowania dynamicznego.

s	$f_4^*(s)$	$x_4^*(s)$
8	3	10
9	4	10

Powyższa tabelka ukazuje koszt przejścia od stanu 8 lub 9 do stanu w końcowym etapie (w tym wypadku jest tylko jeden taki stan).

W następnej fazie pytamy się o wartość $f_3^*(s)$. Przypomnij sobie, że decyzje są podejmowane niezależnie, a zatem $f_3(s, x_3) = c_{sx_3} + f_4^*(x_3)$, przy czym c_{sx_3} oznacza koszt podróży z s do x_3 . W poniższej tabelce przedstawiono odpowiednie wartości $f_3(s, x_3)$ dla każdego możliwego wyboru celu, przy założeniu że s jest równe 5, 6 lub 7.

s	$f_3(s, 8)$	$f_3(s, 9)$	$f_3^*(s)$	$x_3^*(s)$
5	4	8	4	8
6	9	7	7	9
7	6	7	6	8

Widzimy, że jeśli jesteśmy w stanie 5, to stan 8 minimalizuje pozostały koszt. Stan 9 minimalizuje koszt, gdy $s = 6$, a stan 8 minimalizuje koszt, gdy $s = 7$.

Robiąc krok do tyłu, musimy znaleźć wartość $f_2^*(s)$. Poniższa tabelka w analogiczny sposób jak poprzednia pokazuje odpowiednie wartości $f_2(s, x_2)$. Zauważ, że $f_2(s, x_2) = c_{sx_2} + f_3^*(x_2)$.

s	$f_2(s, 5)$	$f_2(s, 6)$	$f_2(s, 7)$	$f_2^*(s)$	$x_2^*(s)$
2	11	11	12	11	5 lub 6
3	7	9	10	7	5
4	8	8	11	8	5 lub 6

Wreszcie, jeśli chodzi o $f_1^*(s)$, ostatnia tabelka daje koszty dotarcia do stanów 2, 3 i 4:

s	$f_1(s, 2)$	$f_1(s, 3)$	$f_1(s, 4)$	$f_1^*(s)$	$x_1^*(s)$
1	13	11	11	11	3 lub 4

To już jest ostateczna tabelka w naszym problemie dyliżansu. Tak oto użyliśmy metody programowania dynamicznego do cofnięcia się wzdłuż całej drogi do pierwszego etapu. Wartości w powyższej tabelce określają koszt podróży ze stanu 1 do stanów 2, 3 i 4.

W tym momencie możemy określić najlepsze rozwiązanie całego problemu. W rzeczywistości istnieją tu trzy najlepsze odpowiedzi (strategie), każda z nich ma ten sam koszt równy 11 jednostek:

- $1 \rightarrow 3 \rightarrow 5 \rightarrow 8 \rightarrow 10$
- $1 \rightarrow 4 \rightarrow 5 \rightarrow 8 \rightarrow 10$
- $1 \rightarrow 4 \rightarrow 6 \rightarrow 9 \rightarrow 10$

Zauważmy, że odwrotna procedura, polegająca na wybieraniu ścieżki o najmniejszym koszcie przy poruszaniu się do przodu przez kolejne etapy, nie daje optymalnego rozwiązania. Takie rozwiązanie otrzymane metodą zachłanną ma koszt 13 jednostek (możesz to sprawdzić). Czyli mamy tutaj do czynienia z sytuacją, gdy podejście zachłanne zawodzi, a programowanie dynamiczne daje poprawną odpowiedź. Wadą tego podejścia jest jednak objawiająca się czasami jego złożoność obliczeniowa. Jeśli N oznacza zarówno liczbę etapów, jak i stanów w każdym z etapów, to wymagana liczba operacji sięga N^3 . Metodę tę możemy jednak tak zaadaptować, żeby dawała wyniki dla wielu problemów optymalizacyjnych, w tym dla problemu znajdowania kolejności mnożenia macierzy oraz znajdowania takiej organizacji drzewa, która minimalizuje koszt przeszukiwania [425].

Algorytmy programowania dynamicznego są niekiedy trudne do zrozumienia. Wynika to z tego, że konstrukcja programu dynamicznego zależy od samego problemu. Opracowywanie jej jest „działaniem intelektualnym o charakterze artystycznym częściowo zależącym od konkretnej struktury rozwiązywanego sekwencyjnego problemu decyzyjnego” [432]. Dlatego zilustrujemy tę metodę, przedstawiając jeszcze dwa przykłady: mnożenie macierzy i TSP.

Założymy, że wymiary macierzy A_1, A_2, A_3 i A_4 wynoszą odpowiednio $20 \times 2, 2 \times 15, 15 \times 40$ i 40×4 i że chcemy znaleźć optymalny sposób obliczenia $A_1 \times A_2 \times A_3 \times A_4$, tzn. chcielibyśmy obliczyć ten iloczyn, stosując minimalną

liczbę mnożeń. Zakładamy tutaj, że aby pomnożyć dwie macierze, P o wymiarze $n \times k$ i Q o wymiarze $k \times m$, należy wykonać nkm mnożeń. Macierz wynikowa R ma wymiar $n \times m$ i

$$r_{ij} = \sum_{v=1}^k p_{iv} q_{vj}$$

dla wszystkich $1 \leq i \leq n$ oraz $1 \leq j \leq m$.

Zauważmy, że mnożenie macierzy w różnej kolejności daje różne koszty. Na przykład:

- $A(B(CD))$ wymaga $15 \cdot 40 \cdot 4 + 2 \cdot 15 \cdot 4 + 20 \cdot 2 \cdot 4 = 2680$ mnożeń
- $(AB)(CD)$ wymaga $20 \cdot 2 \cdot 15 + 15 \cdot 40 \cdot 4 + 20 \cdot 15 \cdot 4 = 4200$ mnożeń
- $((AB)C)D$ wymaga $20 \cdot 2 \cdot 15 + 20 \cdot 15 \cdot 40 + 20 \cdot 40 \cdot 4 = 15\,800$ mnożeń!

Stosując programowanie dynamiczne, tworzymy tutaj strukturę $M(i, j)$, w której zapisujemy minimalną liczbę mnożeń wymaganych do pomnożenia macierzy od A_i do A_j ($i \leq j$). Oczywiście $M(1, 1) = M(2, 2) = M(3, 3) = M(4, 4) = 0$, gdyż tutaj nie wykonujemy żadnych mnożeń. Zauważmy jednak, że problem polega na znalezieniu $M(1, 4)$.

Związek między rozwiązaniami dla mniejszych problemów a rozwiązaniem dla większego problemu jest następujący:

$$M(i, j) = \min_{j \leq k < j} \{M(i, k) + M(k + 1, j) + cost_{ij}^k\}$$

przy czym $cost_{ij}^k$ oznacza liczbę mnożeń wymaganych do pomnożenia iloczynu $A_i \dots A_k$ przez $A_{k+1} \dots A_j$. Chodzi o to, że aby optymalnie pomnożyć ciąg od A_i do A_j , musimy znaleźć najlepszy punkt podziału k , dla którego całkowita liczba mnożeń wymaganych do obliczenia iloczynu $A_i \dots A_k$, wynosząca $M(i, k)$, iloczynu $A_{k+1} \dots A_j$, wynosząca $M(k + 1, j)$ oraz obu tych iloczynów razem (równa $cost_{ij}^k$) była minimalna.

Pamiętając o tym, możemy wykonać następujące kroki. Uzyskanie:

$$M(1, 2) = 600$$

$$M(2, 3) = 1200$$

$$M(3, 4) = 2400$$

jest łatwym ćwiczeniem, gdyż przy mnożeniu dwóch macierzy nie ma punktu podziału. Dalej

$$M(1, 3) = 2800$$

$$M(2, 4) = 1520$$

Zauważmy, że $M(1, 3)$ jest tutaj tą mniejszą z dwóch wartości:

$$M(1, 1) + M(2, 3) + cost_{13}^1 = 0 + 1200 + 1600 = 2800$$

$$M(1, 2) + M(3, 3) + cost_{13}^2 = 600 + 0 + 12000 = 12600$$

Podobnie $M(2, 4)$ jest mniejszą z dwóch wartości:

$$M(2, 2) + M(3, 4) + \text{cost}_{24}^2 = 0 + 2400 + 120 = 2520$$

$$M(2, 3) + M(4, 4) + \text{cost}_{24}^3 = 1200 + 0 + 320 = 1520$$

Wreszcie znajdujemy

$$M(1, 4) = 1680$$

jako najmniejszą z trzech wartości:

$$M(1, 1) + M(2, 4) + \text{cost}_{14}^1 = 0 + 1520 + 160 = 1680$$

$$M(1, 2) + M(3, 4) + \text{cost}_{14}^2 = 600 + 2400 + 1200 = 4200$$

$$M(1, 3) + M(4, 4) + \text{cost}_{14}^3 = 2800 + 0 + 3200 = 6000$$

Tak więc minimalny koszt wyrażony liczbą mnożeń wynosi 1680, ale jeszcze musimy znaleźć odpowiadającą mu kolejność mnożeń macierzy. Znalezienie takiej najlepszej kolejności wymaga użycia dodatkowej struktury danych $O(i, j)$, przeznaczonej do przechowywania indeksu najlepszego punktu podziału. Innymi słowy, $O(i, j) = k$ wtedy i tylko wtedy, gdy $M(i, j)$ osiąga minimalną wartość dla $M(i, k) + M(k + 1, j) + \text{cost}_{ij}^k$. Indeksy umieszczone w tablicy O ukazują poszukiwaną przez nas kolejność: $A_1((A_2 A_3) A_4)$.

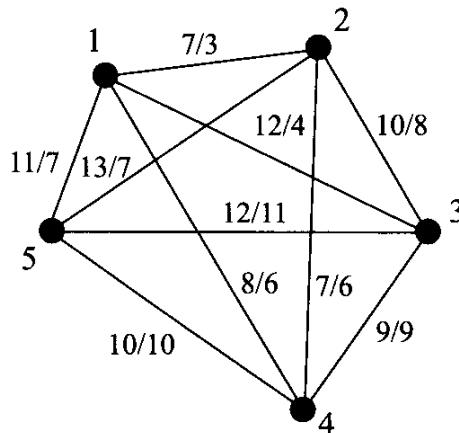
Zakończymy ten podrozdział przykładem zastosowania programowania dynamicznego do TSP. Ostrzegamy jednak – przykład jest długi. Aby jego opis był zrozumiały, będziemy się tutaj zajmować zadaniem TSP dla pewnych pięciu miast. Oto macierz L zawierająca odległości między miastami:

$$L = \begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 3 & 0 & 10 & 7 & 13 \\ 4 & 8 & 0 & 9 & 12 \\ 6 & 6 & 9 & 0 & 10 \\ 7 & 7 & 11 & 10 & 0 \end{bmatrix}$$

Koszt podróży z miasta i do miasta j jest podany w i -tym wierszu i j -tej kolumnie tej macierzy. Na przykład odległość między miastami 1 i 3 wynosi $L(1, 3) = 12$. Odległość ta jest różna od odległości między miastami 3 i 1, która wynosi $L(3, 1) = 4$. Mamy zatem tutaj do czynienia z asymetrycznym TSP. Zauważmy, że $L(i, i) = 0$ dla wszystkich $1 \leq i \leq n$. Całą tę sytuację przedstawiono na rys. 4.8.

Trasa w TSP jest cyklem, który odwiedza *każde* miasto raz i tylko raz, a zatem jest nieistotne, od którego miasta zaczynamy. Przypuśćmy, że zaczynamy od miasta 1. W tym momencie jesteśmy gotowi do podzielenia problemu na mniejsze problemy. Niech $g(i, S)$ oznacza długość najkrótszej ścieżki od miasta i do miasta 1, przechodzącej dokładnie raz przez *każde* miasto ze zbioru S . W szczególności oznacza to, że

$$g(4, \{5, 2, 3\})$$



Rysunek 4.8. TSP z pięcioma miastami. Dla każdej pary miast są podane dwie liczby oddzielone ukośnikiem. Pierwsza z nich określa odległość od miasta o mniejszym numerze do miasta o większym numerze, druga zaś oznacza odległość w przeciwnym kierunku

przedstawia najkrótszą ścieżkę, która prowadzi z miasta 4 przez miasta 5, 2 i 3 (w jakiejś bliżej nieokreślonej kolejności) do miasta 1. Obliczenie długości najkrótszej całej trasy TSP sprowadza się do znalezienia

$$g(1, V - \{1\})$$

przy czym V oznacza zbiór *wszystkich* miast w TSP. Innymi słowy, próbujemy znaleźć najkrótszą ścieżkę, która zaczyna się w mieście 1, przechodzi przez każde miasto ze zbioru $V - \{1\}$ dokładnie raz i wraca do miasta 1.

Sformułowanie problemu jako programowania dynamicznego oznacza pojęcie związku między rozwiązaniami mniejszych problemów a rozwiązaniem większego problemu. Twierdzimy, że mamy tutaj

$$g(i, S) = \min_{j \in S} \{L(i, j) + g(j, S - \{j\})\}$$

Innymi słowy, wybranie najkrótszej ścieżki zaczynającej się od miasta i oraz prowadzącej przez każde miasto w S przed powrotem do miasta 1 wymaga znalezienia w zbiorze S miasta j , dla którego suma odległości $L(i, j)$ oraz długości pozostałą drogi $g(j, S - \{j\})$ jest minimalna. Jeśli znamy rozwiązania mniejszych problemów (tj. problemów, w których rozmiar S wynosi k), to możemy uzyskać rozwiązanie większego problemu (tj. takiego, gdzie rozmiar S wynosi $k + 1$).

Wróćmy do naszego przykładu. Zadanie polega na znalezieniu

$$g(1, \{2, 3, 4, 5\})$$

Zwróćmy także uwagę, że:

$$g(2, \emptyset) = L(2, 1) = 3$$

$$g(3, \emptyset) = L(3, 1) = 4$$

$$g(4, \emptyset) = L(4, 1) = 6$$

$$g(5, \emptyset) = L(5, 1) = 7$$

gdyż zbiór S jest tutaj pusty i przechodzimy bezpośrednio od miasta i ($i = 2, 3, 4, 5$) do miasta 1. Następny krok jest równie prosty – musimy znaleźć rozwiązania dla wszystkich problemów, w których rozmiar S wynosi jeden. Wykonanie tego wymaga rozwiązań 12 podproblemów, gdyż będziemy zaczynać od miast 2, 3, 4 oraz 5 i dla każdego z nich będziemy rozważali trzy możliwe zbiory jednoelementowe. Na przykład dla miasta 2 mamy:

$$\begin{aligned} g(2, \{3\}) &= L(2, 3) + g(3, \emptyset) = 10 + 4 = 14 \\ g(2, \{4\}) &= L(2, 4) + g(4, \emptyset) = 7 + 6 = 13 \\ g(2, \{5\}) &= L(2, 5) + g(5, \emptyset) = 13 + 7 = 20 \end{aligned}$$

Zwróćmy uwagę, że wynik $g(2, \{3\}) = 14$ oznacza, iż długość najkrótszej ścieżki z miasta 2 do 1, która przed dojściem do 1 przechodzi przez miasto 3, wynosi 14. Wynik ten jest oczywisty, ponieważ S składa się tylko z jednego elementu i nie są tu możliwe żadne wybory. Podobnie, dla miasta 3 mamy:

$$\begin{aligned} g(3, \{2\}) &= L(3, 2) + g(2, \emptyset) = 8 + 3 = 11 \\ g(3, \{4\}) &= L(3, 4) + g(4, \emptyset) = 9 + 6 = 15 \\ g(3, \{5\}) &= L(3, 5) + g(5, \emptyset) = 12 + 7 = 19 \end{aligned}$$

dla miasta 4 mamy:

$$\begin{aligned} g(4, \{2\}) &= L(4, 2) + g(2, \emptyset) = 6 + 3 = 9 \\ g(4, \{3\}) &= L(4, 3) + g(3, \emptyset) = 9 + 4 = 13 \\ g(4, \{5\}) &= L(4, 5) + g(5, \emptyset) = 10 + 7 = 17 \end{aligned}$$

oraz dla miasta 5 mamy:

$$\begin{aligned} g(5, \{2\}) &= L(5, 2) + g(2, \emptyset) = 7 + 3 = 10 \\ g(5, \{3\}) &= L(5, 3) + g(3, \emptyset) = 11 + 4 = 15 \\ g(5, \{4\}) &= L(5, 4) + g(4, \emptyset) = 10 + 6 = 16 \end{aligned}$$

Jesteśmy teraz gotowi do wykonania następnego kroku naszego dynamicznego programu, gdy S wynosi 2. Musimy tutaj rozwiązać również 12 podproblemów, gdyż każde z czterech miast może być rozważane wraz z dwoma innymi miastami (dwoma spośród trzech). I tak dla miasta 2 mamy:

$$\begin{aligned} g(2, \{3, 4\}) &= \min\{L(2, 3) + g(3, \{4\}), L(2, 4) + g(4, \{3\})\} = \\ &= \min\{10 + 15, 7 + 13\} = \min\{25, 20\} = 20 \\ g(2, \{3, 5\}) &= \min\{L(2, 3) + g(3, \{5\}), L(2, 5) + g(5, \{3\})\} = \\ &= \min\{10 + 19, 13 + 15\} = \min\{29, 28\} = 28 \\ g(2, \{4, 5\}) &= \min\{L(2, 4) + g(4, \{5\}), L(2, 5) + g(5, \{4\})\} = \\ &= \min\{7 + 17, 13 + 16\} = \min\{24, 29\} = 24 \end{aligned}$$

Wynik $g(2, \{3, 4\}) = 20$ oznacza, że długość najkrótszej ścieżki z miasta 2 do 1, która przed dojściem do 1 przechodzi do miasta 3 i 4 (w dowolnej kolejności), wynosi 20. Tym razem musimy rozważyć dwie możliwości (możemy pojechać

najpierw do miasta 3 lub najpierw do miasta 4) i wybrać lepszą z nich. Podobnie, dla miasta 3 mamy:

$$\begin{aligned} g(3, \{2, 5\}) &= \min\{L(3, 2) + g(2, \{5\}), L(3, 5) + g(5, \{2\})\} = \\ &= \min\{8 + 20, 12 + 10\} = \min\{28, 22\} = 22 \\ g(3, \{2, 4\}) &= \min\{L(3, 2) + g(2, \{4\}), L(3, 4) + g(4, \{2\})\} = \\ &= \min\{8 + 13, 9 + 9\} = \min\{21, 18\} = 18 \\ g(3, \{4, 5\}) &= \min\{L(3, 4) + g(4, \{5\}), L(3, 5) + g(5, \{4\})\} = \\ &= \min\{9 + 17, 12 + 16\} = \min\{26, 28\} = 26 \end{aligned}$$

dla miasta 4 mamy:

$$\begin{aligned} g(4, \{2, 3\}) &= \min\{L(4, 2) + g(2, \{3\}), L(4, 3) + g(3, \{4\})\} = \\ &= \min\{6 + 14, 9 + 15\} = \min\{20, 24\} = 20 \\ g(4, \{2, 5\}) &= \min\{L(4, 2) + g(2, \{5\}), L(4, 5) + g(5, \{2\})\} = \\ &= \min\{6 + 20, 10 + 10\} = \min\{26, 20\} = 20 \\ g(4, \{3, 5\}) &= \min\{L(4, 3) + g(3, \{5\}), L(4, 5) + g(5, \{3\})\} = \\ &= \min\{9 + 19, 10 + 15\} = \min\{28, 25\} = 25 \end{aligned}$$

i dla miasta 5 mamy:

$$\begin{aligned} g(5, \{2, 3\}) &= \min\{L(5, 2) + g(2, \{3\}), L(5, 3) + g(3, \{2\})\} = \\ &= \min\{7 + 14, 11 + 11\} = \min\{21, 22\} = 21 \\ g(5, \{2, 4\}) &= \min\{L(5, 2) + g(2, \{4\}), L(5, 4) + g(4, \{2\})\} = \\ &= \min\{7 + 13, 10 + 19\} = \min\{20, 29\} = 20 \\ g(5, \{3, 4\}) &= \min\{L(5, 3) + g(3, \{4\}), L(5, 4) + g(4, \{3\})\} = \\ &= \min\{11 + 15, 10 + 13\} = \min\{26, 23\} = 23 \end{aligned}$$

Jesteśmy już gotowi do wykonania następnego kroku, w którym rozmiar zbioru S wynosi trzy. Musimy się tutaj zmierzyć tylko z czterema podproblemami – po jednym dla każdego miasta. W ten sposób dla miasta 2 musimy wybrać najkrótszą spośród trzech ścieżek:

$$\begin{aligned} g(2, \{3, 4, 5\}) &= \\ &= \min\{L(2, 3) + g(3, \{4, 5\}), L(2, 4) + g(4, \{3, 5\}), L(2, 5) + g(5, \{3, 4\})\} = \\ &= \min\{10 + 26, 7 + 25, 13 + 23\} = \min\{36, 32, 34\} = 32 \end{aligned}$$

Wynik $g(2, \{3, 4, 5\}) = 32$ oznacza, że długość najkrótszej ścieżki z miasta 2 do 1, która przed dojściem do 1 przechodzi przez miasta 3, 4 i 5 (w dowolnej kolejności), wynosi 32. Musimy teraz rozważyć trzy możliwości: możemy pojechać najpierw do miasta 3, do miasta 4 lub do miasta 5, i wybrać najlepszą z nich. Podobnie, dla miast 3, 4 i 5 mamy:

$$\begin{aligned} g(3, \{2, 4, 5\}) &= \\ &= \min\{L(3, 2) + g(2, \{4, 5\}), L(3, 4) + g(4, \{2, 5\}), L(3, 5) + g(5, \{2, 4\})\} = \\ &= \min\{8 + 24, 9 + 20, 12 + 20\} = \min\{32, 29, 32\} = 29 \end{aligned}$$

$$\begin{aligned}
 g(4, \{2, 3, 5\}) &= \\
 &= \min\{L(4, 2) + g(2, \{3, 5\}), L(4, 3) + g(3, \{2, 5\}), L(4, 5) + g(5, \{2, 3\})\} = \\
 &= \min\{6 + 28, 9 + 22, 10 + 21\} = \min\{34, 31, 31\} = 31 \\
 g(5, \{2, 3, 4\}) &= \\
 &= \min\{L(5, 2) + g(2, \{3, 4\}), L(5, 3) + g(3, \{2, 4\}), L(5, 4) + g(4, \{2, 3\})\} = \\
 &= \min\{7 + 20, 11 + 18, 10 + 20\} = \min\{27, 29, 30\} = 27
 \end{aligned}$$

Wreszcie nadszedł czas na końcowy krok, w którym zaczynamy od miasta 1 i zamykamy cykl. Wracamy w tym momencie do pierwotnego problemu:

$$g(1, \{2, 3, 4, 5\}) = ?$$

Zaczynając od miasta 1, mamy cztery możliwości: pójść najpierw do miasta 2, miasta 3, miasta 4 lub miasta 5. Odpowiedzi dla pojawiających się tutaj podproblemów (tzn. znajdowanie najkrótszych długości dla wszystkich możliwych ciągów dalszych wymienionych początków) są już znane, możemy więc podjąć ostateczną decyzję:

$$\begin{aligned}
 g(1, \{2, 3, 4, 5\}) &= \min\{L(1, 2) + g(2, \{3, 4, 5\}), L(1, 3) + g(3, \{2, 4, 5\}), \\
 &\quad L(1, 4) + g(4, \{2, 3, 5\}), L(1, 5) + g(5, \{2, 3, 4\})\} = \\
 &= \min\{7 + 32, 12 + 29, 8 + 31, 11 + 27\} = \min\{39, 41, 39, 38\} = 38
 \end{aligned}$$

Najkrótsza trasa ma zatem długość 38.

Jak jednak ta trasa przebiega? Podobnie jak w wypadku problemu mnożenia macierzy, znaleźliśmy optymalną wartość funkcji oceny, ale musimy jeszcze ustalić, która trasa ją dała. I znowu zadanie to jest łatwe, jeśli śledzimy nasze obliczenia. Potrzebujemy w tym celu dodatkowej struktury danych W , która przechowuje informacje, jakie w następnej kolejności wybrać miasto, żeby ścieżka miała minimalną długość. Zatem na przykład

$$W(5, \{2, 3, 4\}) = 2$$

gdyż najkrótsza ścieżka z miasta 5 do miasta 1, która musi przejść przez miasta 2, 3 i 4 przed dojściem do 1, musi najpierw przechodzić przez miasto 2. Podobnie

$$W(1, \{2, 3, 4, 5\}) = 5$$

Stąd globalne rozwiązanie dla analizowanego TSP daje trasę

$$1 - 5 - 2 - 4 - 3 - 1$$

której długość wynosi

$$11 + 7 + 7 + 9 + 4 = 38$$

Czy myśl o użyciu programowania dynamicznego do rozwiązania problemu TSP z 50 miastami budzi grozę?

4.4. Metoda podziału i ograniczeń

Stwierdziliśmy już, że rozmiar problemów w świecie rzeczywistym może wzrosnąć bardzo szybko wraz ze zwiększeniem się liczby pojawiających się w nich zmiennych. Przypomnijmy, że dla symetrycznego TSP istnieje $(n - 1)!/2$ różnych rozwiązań. Przeszukiwanie wyczerpujące nie wchodzi w grę już dla $n > 20$, przydałoby się zatem mieć jakąś heurystykę, która wyeliminowałaby części przestrzeni przeszukiwania, co do których wiemy, że na pewno nie dadzą optymalnego rozwiązania.

Metoda podziału i ograniczeń jest jedną z heurystyk, które korzystają z koncepcji kolejnego dzielenia przestrzeni przeszukiwania. Najpierw musimy mieć jakiś sposób na określanie dolnego ograniczenia kosztu dowolnego z rozwiązań (lub górnego ograniczenia, w zależności od tego, czy pracujemy nad minimalizacją, czy maksymalizacją). Pomysł polega tutaj na tym, że jeśli mamy rozwiązanie o koszcie, powiedzmy, c jednostek, a następne dostępne rozwiązanie ma dolne ograniczenie większe niż c , i dokonujemy minimalizacji, to nie musimy obliczać, ile wynosi prawdziwa jego wartość. Możemy od razu pominąć ten przypadek i przejść do następnego.

Warto myśleć o przestrzeni przeszukiwania jak o drzewie. W heurystyce podziału i ograniczeń usuwamy gałęzie, które nas nie interesują. Rozważmy na przykład symetryczny problem TSP z pięcioma miastami. Całą przestrzeń przeszukiwania \mathcal{S} można podzielić, powiedzmy, na podstawie tego, czy krawędź (1 2) należy do trasy, czy nie. Podobnie, przestrzeń można dalej podzielić w zależności od tego, czy krawędź (2 3) należy do trasy, czy nie, itd. Na rysunku 4.9 widać drzewo przedstawiające działanie tej zasady. Liście drzewa przedstawiają $(5 - 1)!/2 = 12$ możliwych tras.

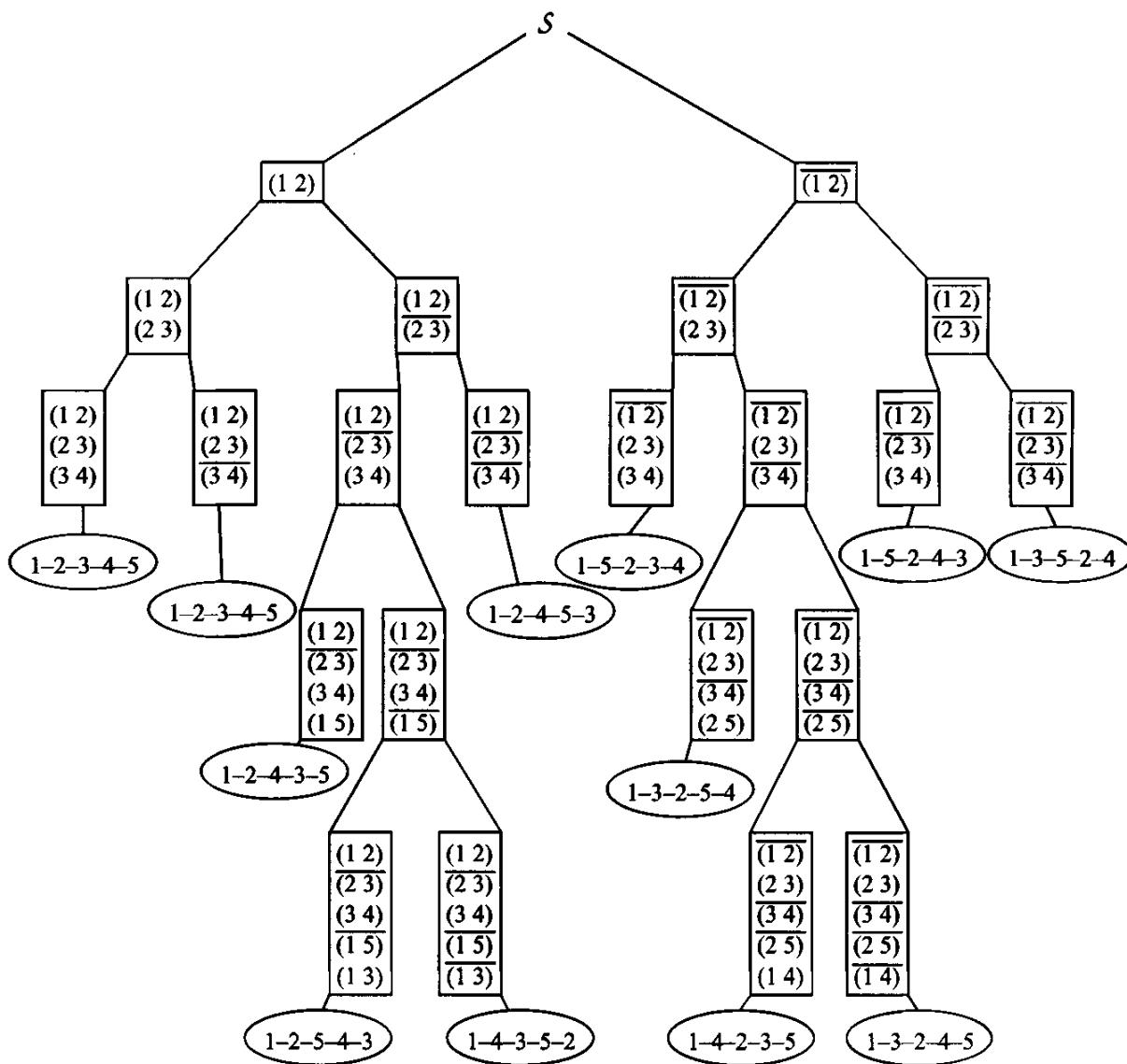
Przypuśćmy, że koszty podróży między miastami są opisane za pomocą następującej macierzy:

$$\begin{bmatrix} 0 & 7 & 12 & 8 & 11 \\ 7 & 0 & 10 & 7 & 13 \\ 12 & 10 & 0 & 9 & 12 \\ 8 & 7 & 9 & 0 & 10 \\ 11 & 13 & 12 & 10 & 0 \end{bmatrix}$$

gdzie każda pozycja określa koszt podróży z miasta w i -tym wierszu do miasta w j -tej kolumnie. Zera na głównej przekątnej wskazują, że nie możemy podróżować z danego miasta do niego samego.

Przy danej organizacji przestrzeni przeszukiwania w postaci drzewa jest nam potrzebna heurystyka umożliwiająca oszacowanie dolnego ograniczenia kosztu dowolnego końcowego rozwiązania lub nawet dowolnego wierzchołka drzewa. Jeśli takie dolne ograniczenie jest większe niż koszt najlepszego dotychczas znalezionego rozwiązania, to możemy zrezygnować z nowego rozwiązania i podążyć dalej, bez konieczności obliczania jego dokładnego kosztu.

Oto prosta, choć może nie najlepsza metoda obliczania dolnego ograniczenia długości trasy. Rozważmy jakiś przypadek pełnego rozwiązania TSP. Każda trasa zawiera dla każdego miasta dwie krawędzie przyległe: jedna krawędź



Rysunek 4.9. Podział przestrzeni przeszukiwania \mathcal{S} dla TSP z pięcioma miastami w zależności od tego, czy dana krawędź (i, j) należy do trasy, czy nie. W tym drugim przypadku nad krawędzią widnieje pozioma kreska. Krawędzie były wybierane w sposób losowy. W ovalach podano trasy wynikające z wyborów

wprowadza nas do tego miasta, druga krawędź wyprowadza z niego i wiedzie do następnego. Jeśli zatem weźmiemy dwie najkrótsze krawędzie związane z każdym z miast i obliczymy sumę wszystkich ich długości podzieloną przez dwa, to otrzymamy pewne dolne ograniczenie długości trasy. Lepsze rozwiązanie nie jest możliwe, gdyż w regule tej dla każdego miasta wybieramy najlepsze krawędzie. Przy powyższej macierzy dolne ograniczenie długości wszystkich możliwych tras wynosi:

$$\frac{1}{2}[(7+8)+(7+7)+(9+10)+(7+8)+(10+11)] = \frac{1}{2} \cdot 84 = 42$$

Zauważmy, że wartości 7 i 8 w pierwszym nawiasie odpowiadają długościom dwóch najkrótszych krawędzi związanych z miastem 1, a z kolei wartości 7 i 7 w drugim nawiasie odpowiadają dwóm najkrótszym krawędziom połączonym z miastem 2 itd.

Jeśli dla jakiejś trasy pewne krawędzie są już określone, to możemy wziąć pod uwagę wynikające z tego informacje i obliczyć dolne ograniczenie takiego częściowego rozwiązania. Gdybyśmy, na przykład, wiedzieli, że krawędź (2 3) została wybrana, a krawędź (1 2) nie, to dolne ograniczenie dla takiego częściowego rozwiązania wynosiłoby

$$\frac{1}{2}[(8 + 11) + (7 + 10) + (9 + 10) + (7 + 8) + (10 + 11)] = \frac{1}{2} \cdot 91 = 45,5$$

Możemy polepszyć dolne ograniczenie, włączając w jego obliczanie wybrane krawędzie i wykluczając te, które nie mogą się już pojawić. Gdyby do trasy zostały włączone krawędzie (1 2) oraz (2 4), to powyższe obliczenia dałyby dolne ograniczenie 42. W tym wypadku jednak po bliższym przyjrzeniu się możemy wykluczyć także krawędź (1 4) i uzyskać lepsze ograniczenie:

$$\frac{1}{2}[(7 + 11) + (7 + 7) + (9 + 10) + (7 + 9) + (10 + 11)] = \frac{1}{2} \cdot 88 = 44$$

Im lepsze jest takie ograniczenie, tym szybszy jest algorytm przeszukujący drzewo – taki algorytm po prostu eliminuje więcej rozwiązań. Trzeba jednakże uwzględnić koszt obliczania takich dolnych ograniczeń. Metoda może być użyteczna, jeśli czas obliczania ograniczeń nie przekracza czasu, jaki oszczędzamy na przycinaniu drzewa. W związku z tym takie dolne ograniczenia muszą być tak precyzyjnie dobrane, jak jest to możliwe.

Istnieje wiele sposobów ulepszania dolnego ograniczenia w problemie TSP, omawianie ich wykracza jednak poza zakres tej książki (więcej znajdziemy w [344]). Skupmy się teraz raczej znowu na optymalizacji numerycznej i zobaczymy, jak można tam zastosować heurystykę podziału i ograniczeń.

Założymy, że zadanie polega na

minimalizacji $f(\mathbf{x})$

przy czym $\mathbf{x} \in D \subseteq \mathbb{R}^n$. Innymi słowy, szukamy $\mathbf{x}^* = \{\mathbf{x} \in D \mid f(\mathbf{x}) = f^*\}$, gdzie $f^* = \inf_{\mathbf{x} \in D} f(\mathbf{x})$. Stosując zapis z [296], mamy, co następuje:

- D_i oznacza n -wymiarowe pudełko w \mathbb{R}^n , tzn. D_i jest określone przez \mathbf{l}^i (dolny róg) i \mathbf{u}^i (górny róg), przy czym $\mathbf{x} \in D_i$ wtedy i tylko wtedy, gdy $l_k^i \leq x_k \leq u_k^i$ dla wszystkich $1 \leq k \leq n$;
- C , określane jako zbiór kandydatów, jest zbiorem pudełek D_i , $i = 1, \dots, p$; dla każdego pudełka D_i obliczamy dolne ograniczenie wartości funkcji f w pudełku, tzn. dolne ograniczenie f na D_i ;
- f_{bound} reprezentuje bieżące górne ograniczenie f^* znalezione przez nasz algorytm; za wartość tę możemy wziąć na przykład najmniejszą dotychczas obliczoną wartość funkcji f .

Koncepcja algorytmu polega na tym, że zaczynamy od $C = \{D\}$ i za pomocą metody podziału i ograniczeń redukujemy zbiór C w taki sposób, żeby był zbieżny do zbioru \mathbf{x}^* punktów odpowiadających globalnym minimom. Zauważmy jednak, że sprawa zbieżności nie jest tutaj taka oczywista. Proces ten zależy od mechanizmu wyboru pudełek. Jeśli zawsze jest wybierane pudełko

o najmniejszym ograniczeniu dolnym, to algorytm może nigdy nie dać wyniku. Z kolei, jeśli zawsze jest wybierane największe pudełko – zgodnie z pewną miarą na rozmiar problemu, taką jak całkowita długość wszystkich wymiarów pudełek – algorytm zakończy pracę, lecz czas jego działania może być niedopuszczalnie duży. Co więcej, także stwierdzenie, że algorytm ma się zatrzymać, zależy od podziału topologii, gdyż ciąg pudełek może jedynie przybliżać samą krzywą.

Na rysunku 4.10 pokazano szkic omawianego tutaj algorytmu rozgałęziania i ograniczeń. Przy każdym kroku C zawiera zbiór aktywnych pudełek D_i ; w każdej chwili zbiór minimów globalnych \mathbf{x}^* jest zawarty w sumie teoriomnościovowej tych pudełek:

$$\mathbf{x}^* \in \bigcup D_i \subseteq D$$

```

procedure podział i ograniczanie
begin
    inicjuj  $C$ 
    inicjuj  $f_{bound}$ 
    while (nie zachodzi warunek zakończenia) do
        usuń najlepsze pudełko  $C \rightarrow D_i$ 
        zredukuj lub podziel  $D_i \rightarrow D'_i$ 
        uaktualnij  $f_{bound}$ 
         $C \leftarrow C \cup D'_i$ 
        for all  $D_i \in C$  do
            if (dolne ograniczenie  $f(D_i)$ ) >  $f_{bound}$  then usuń  $D_i$  z  $C$ 
    end
```

Rysunek 4.10. Algorytm podziału i ograniczeń

W każdej iteracji algorytmu ze zbioru kandydatów C jest wybierane i usuwane jedno pudełko D_i . Istnieje wiele metod wybierania, które pudełko należy usunąć; najczęściej stosowaną metodą jest wybór pudełka o najmniejszym ograniczeniu dolnym. W następnej kolejności próbujemy albo zmniejszyć rozmiar D_i , albo w ogóle usunąć pudełko. Operację tę można przeprowadzić, sprawdzając przebieg monotoniczności funkcji f w D_i (tzn. określenie, czy jej pochodne cząstkowe są zawsze dodatnie, czy ujemne). Jeśli funkcja f jest monotoniczna w wymiarze i , to pudełko możemy wzduż tego wymiaru zredukować do jednego punktu znajdującego się na brzegu pudełka¹. Jeśli redukcja D_i nie jest możliwa, to pudełko jest zastępowane dwoma lub więcej pudełkami. W każdym wypadku zbiór kandydatów C jest rozszerzany o D'_i , co oznacza albo zredukowane, albo podzielone D_i .

¹Możemy też wykonać inne testy i na przykład zastosować redukcję Newtona [296]. Może to dać wniosek, że D_i nie zawiera punktów stacjonarnych – wszystkie iteracyjne ciągi poszukiwań wychodzą poza D_i – lub że obszar ten zawiera jeden punkt stacjonarny, tzn. wszystkie iteracyjne ciągi są zbieżne do tego samego punktu.

Zanim D'_i trafi do C , uzyskujemy nowe dolne ograniczenia. Zauważmy jeszcze, że jeśli dolne ograniczenie dla pudełka jest większe niż f_{bound} , to pudełko to może zostać usunięte ze zbioru kandydatów.

Istnieje wiele wariantów tego ogólnego algorytmu podziału i ograniczeń. Możemy na przykład zastosować wersję wykorzystującą arytmetykę na przedziałach, gdzie wszystkie obliczenia są wykonywane na przedziałach, a nie na liczbach rzeczywistych, lub też wersję stochastyczną, w której wartości f są obliczane w pewnej liczbie punktów losowych. Więcej szczegółów na ten temat można znaleźć w [296]. Ogólne informacje na temat metod przedziałowych w optymalizacji globalnej podano w [378].

4.5. Algorytm A^*

Stwierdziliśmy już, że wykonywanie najlepszego dostępnego ruchu w danym momencie może prowadzić do kłopotów. Algorytmy zachłanne nie zawsze działają najlepiej. Rzecz w tym, że to, co jest teraz lepsze, może okazać się później nie całkiem tym, czego potrzebujemy. Gdybyśmy jednak mieli funkcję oceny, która byłaby na tyle „pouczająca”, żeby zapewnić unikanie takich pułapek, moglibyśmy korzystać z zalet takich metod zachłannych. Ten prosty pomysł prowadzi do koncepcji zwanej przeszukiwaniem typu „najpierw najlepszy” (ang. *best-first search*) i jej rozszerzenia zwanego algorytmem A^* .

Przy organizacji przestrzeni przeszukiwania w postaci drzewa, które jest specjalnym przypadkiem grafu skierowanego (niedługo wróćmy jeszcze do tej obserwacji), mamy do wyboru różne możliwości przeszukiwania tej struktury. Możemy zastosować przeszukiwanie w głąb i przechodzić w dół drzewa aż do jakiegoś wyznaczonego poziomu, zanim wykonamy jakieś obliczenia, a następnie wrócić tą samą drogą w górę. Możemy też zamiast tego spróbować uporządkować dostępne wierzchołki według jakiejś heurystyki, która odpowiada naszym oczekiwaniom co do tego, jakiej sytuacji spodziewamy się na głębszych poziomach naszego przeszukiwania. Chcielibyśmy zaczynać przeszukiwanie od wierzchołków, które dają najlepsze szanse na znalezienie czegoś dobrego. Zarys algorytmu przeszukiwania typu „najpierw najlepszy” pokazano na rys. 4.11.

```

procedure best-first( $v$ )
begin
    odwiedź  $v$ 
    for każde dostępne  $w$  do
        przypisz  $w$  wartość zgodnie z heurystyką
         $q \leftarrow$  najlepszy dostępny wierzchołek
        best-first( $q$ )
    end

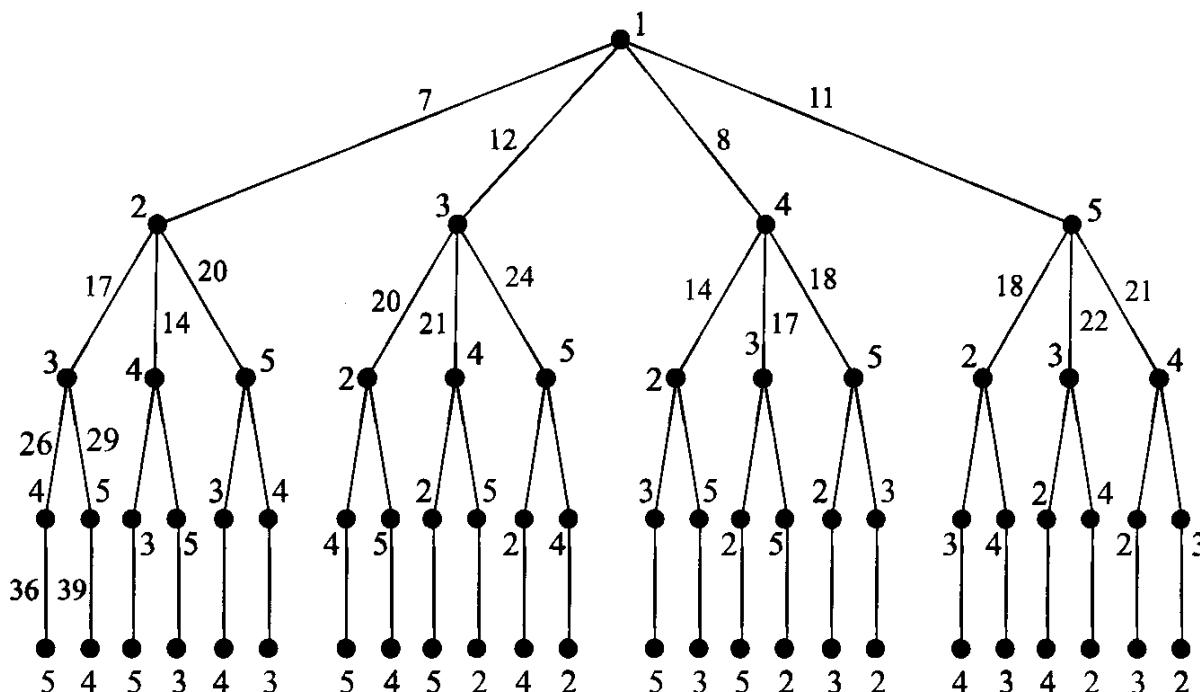
```

Rysunek 4.11. Algorytm przeszukiwania typu „najpierw najlepszy”

Zasadnicza sprawa jest ukryta w pojęciu *dostępnych* wierzchołków. Algorytm korzysta z dwóch list wierzchołków: *otwartych* i *zamkniętych*. Pierw-

sza lista zawiera wszystkie dostępne wierzchołki, druga zaś wierzchołki, które już zostały przetworzone. Gdy procedura przeszukiwania typu „najpierw najlepszy” zaczyna od określonego wierzchołka, który jest umieszczony na liście wierzchołków *zamkniętych*, wszystkie jego wierzchołki potomne są przenoszone na listę wierzchołków *otwartych* (oznacza to, że wierzchołki te stają się dostępne). Wierzchołki te są oceniane za pomocą jakiejś heurystyki, a najlepszy z nich jest wybierany do dalszego przetwarzania.

Jeśli przestrzeń przeszukiwania jest zorganizowana jako graf skierowany ogólnej postaci, to jeden z wierzchołków potomnych może już być obecny na liście wierzchołków otwartych lub zamkniętych. Gdy się to zdarzy, wierzchołek musi zostać ponownie oceniony i może zajść potrzeba przeniesienia go z listy wierzchołków *zamkniętych* na listę wierzchołków *otwartych*. Będziemy pomijać tę możliwość i skupimy się jedynie na przestrzeniach przeszukiwania mających postać drzewa. Na rysunku 4.12 widać drzewo odpowiadające problemowi TSP z rys. 4.8.



Rysunek 4.12. Przestrzeń przeszukiwania dla TSP z pięcioma miastami. Znajdujące się przy niektórych krawędziach liczby określają koszt związań z nimi częściowych rozwiązań obejmujących wierzchołek na końcu krawędzi. Wartości te definiujemy dalej jako wartość c dla wierzchołka. Zwróćmy uwagę, że koszt końcowego rozwiązania wymaga dodania końcowej krawędzi zamkającej cykl

Oto główne różnice między przeszukiwaniem w głąb i bardzo podobnym przeszukiwaniem wszerz a przeszukiwaniem typu „najpierw najlepszy”:

- Przeszukiwanie typu „najpierw najlepszy” bada najbardziej obiecujący wierzchołek kolejny, podczas gdy przeszukiwanie w głąb wchodzi tak głęboko, jak to tylko jest możliwe zgodnie z dowolnym wzorcem, a przeszukiwanie wszerz bada wszystkie wierzchołki na jednym poziomie przed przejściem do następnego poziomu.

- Przeszukiwanie typu „najpierw najlepszy” korzysta z heurystyki określającej wartość wierzchołka, natomiast przeszukiwanie w głąb i przeszukiwanie wszerz niczego takiego nie używają.

Efektywność algorytmu przeszukiwania typu „najpierw najlepszy” jest ściśle związana z jakością zastosowanej heurystyki oceny, a zatem prawidłowe jej opracowanie wymaga trochę więcej uwagi.

Przy ocenianiu częściowo utworzonego rozwiązania q powinniśmy brać pod uwagę dwa jego elementy:

- wartość decyzji już wykonanych $c(q)$;
- potencjał związany z pozostałymi decyzjami $h(q)$.

Tak więc funkcja oceny $eval$ dla częściowego rozwiązania q ma postać

$$eval(q) = c(q) + h(q)$$

Czasami z postawienia problemu bezpośrednio wynika dokładna wartość c . W TSP, na przykład, jest stosunkowo łatwo ocenić wartość już podjętych decyzji. Częściowa trasa

$$q = 1 - 3 - 5$$

może zostać oceniona jako

$$c(q) = dist(1, 3) + dist(3, 5)$$

przy czym $dist$ oznacza odległość od miasta w pierwszym argumentie do miasta w drugim. O wiele trudniejsze jest jednak ocenienie potencjalnej jakości pozostałych decyzji. Dlatego potrzebna jest nam dobra heurystyka h . Jak jednak ocenić jakość konkretnej heurystyki?

Jedno z kryteriów oceny heurystyki h opiera się na pojęciu *dopuszczalności*. Algorytm jest *dopuszczalny*, jeśli zawsze kończy pracę z optymalnym rozwiązaniem. Chcielibyśmy dobierać h tak, aby gwarantowała dopuszcjalność. Przypuśćmy, że dysponujemy *wyrocnią*, która potrafi podać prawdziwy koszt h^* kontynuowania pracy, żeby uzyskać wynik. Możemy teraz zapisać *idealną* funkcję oceny

$$eval^*(q) = c(q) + h^*(q)$$

która podaje prawdziwy koszt optymalnego rozwiązania, jakie możemytrzymać, wychodząc z wierzchołka q . Łatwo możemy wykazać, że algorytm przeszukiwania typu „najpierw najlepszy” z użyciem funkcji $eval^*$ jest dopuszcjalny. Taką funkcją $eval^*$ jednak nie dysponujemy, gdyż zwykle niemożliwe jest znalezienie takiej wszystkowiedzącej wyrocni h^* . Oznacza to, że musimy jakość szacować wartość h^* . Możemy zastąpić h^* pewnym oszacowaniem h minimalnego kosztu kontynuowania pracy, żeby uzyskać wynik. Jeśli h jest ograniczone z góry¹, tzn. nie osiąga nigdy wartości większej niż prawdziwy koszt najlepszej kontynuacji:

$$h(q) \leq h^* \quad \text{dla wszystkich wierzchołków } q$$

¹W przypadku problemów minimalizacji.

to uzyskany za pomocą tego oszacowania algorytm przeszukiwania typu „najpierw najlepszy” jest zawsze dopuszczalny i w tym konkretnym przypadku nazywany algorymem A^* . Powód, dla którego A^* gwarantuje uzyskanie globalnego rozwiązania, jest ukryty w powyższej nierówności. W związku z tym, że h ocenia pozostały koszt rozwiązania z niedomiarem, niemożliwe jest ominięcie najlepszej ścieżki!

Czasami istnieje wiele heurystyk h_i , które szacują pozostały koszt osiągnięcia celu. Przy dwóch heurystykach:

$$h_1(q) \leq h^*(q) \text{ oraz } h_2(q) \leq h^*(q)$$

spełniających $h_1(q) \leq h_2(q)$ dla wszystkich wierzchołków q , mówimy, że heurystyka h_2 jest lepsza¹, gdyż daje nam ścisłejsze ograniczenie.

Ciekawe jest wskazanie pewnych związków między różnymi wcześniejszymi przedstawionymi metodami przeszukiwania. Na przykład przeszukiwanie typu „najpierw najlepszy” z funkcją oceny

$$\text{eval}(q) = c(q) + h(q)$$

dla rozwiązania częściowego q może być uznane za przeszukiwanie wszerz, jeśli:

$$h(q) = 0 \text{ dla wszystkich } q$$

$$c(q') = c(q) + 1 \text{ przy czym } q' \text{ jest następcą } q$$

oraz może być uznane za przeszukiwanie w głąb, jeśli:

$$h(q) = 0 \text{ dla wszystkich } q$$

$$c(q') = c(q) - 1 \text{ przy czym } q' \text{ jest następcą } q$$

Zauważmy jeszcze, że jeśli

$$\begin{aligned} h(q) &= 0 \text{ dla wszystkich } q \text{ i } c(q) = \text{prawdziwy koszt ścieżki} \\ &\text{od początku do wierzchołka } q \end{aligned}$$

to otrzymujemy metodę podziału i ograniczeń!

Nasze rozważania ograniczyliśmy do przypadku, gdy przestrzeń przeszukiwania jest zorganizowana w postaci drzewa. Powyższe algorytmy można jednak uogólnić tak, żeby działały na dowolnych grafach [354]. Takie podejście wymaga obsługiwanego sytuacji, gdy ponownie odwiedzamy jakiś wierzchołek, aktualizowania wartości funkcji oceny, gdy koszt nowo odkrytej ścieżki jest lepszy niż koszt najlepszej dotychczas znanej.

4.6. Podsumowanie

Jeśli jesteś spostrzegawczy, to mogłeś zauważyć, że temat „metody tradycyjne” zajmuje 54 strony, czyli dwa rozdziały tej książki. Z całą mocą podkreślaliśmy w przedmowie i wstępie, że metody tradycyjne pozostawiają wiele do życzenia.

¹W terminologii z dziedziny sztucznej inteligencji h_2 ma więcej informacji.

Długość rozdziałów 3 i 4 nie przekłada się na ich wartość. W tym wypadku raczej oznacza słabość wszystkich tu opisanych metod. Każda z nich została zaprojektowana dla określonego rodzaju problemów i tylko dla tych problemów jest ona efektywna. Po zmianie problemu metody tradycyjne zaczynają się uginać pod ciężarem zadania, a czasem wręcz się załamują.

Jeśli masz do czynienia z problemem o kwadratowej funkcji oceny, powinieneś użyć metody Newtona-Gaussa. Jeśli masz do czynienia z liniową funkcją oceny z liniowymi ograniczeniami równościowymi i ograniczeniami w postaci nierówności, to najlepsze wyniki uzyskasz, stosując metodę sympleks. Powinieneś wiedzieć, kiedy metody z rozdziałów 3 i 4 dadzą globalnie optymalne rozwiązanie, a kiedy się to nie powiedzie.

Od nauczenia się kilku konkretnych metod ważniejsze jest jednak, żebyś zauważył, że (pominawszy szczegóły) są dwa główne sposoby podchodzenia do rozwiązywania problemów. Można pracować albo z wykorzystaniem pełnych rozwiązań, albo z rozwiązaniami częściowymi. To pierwsze podejście ma tę zaletę, że w każdej chwili mamy do dyspozycji gotowe rozwiązanie, które możemy zastosować w praktyce nawet wówczas, gdy zabrakło już nam czasu albo gdy algorytm nie zatrzymuje się z najlepszym rozwiązaniem. Drugie podejście daje czasem istotną możliwość skorzystania ze struktury wewnętrznej niektórych problemów. Możemy uzyskać ogromne przyspieszenie w generowaniu odpowiedzi dla wydawałoby się skomplikowanego problemu przez rozłożenie go na proste podproblemy. Możemy też zorganizować przestrzeń przeszukiwania w postaci drzewa, a następnie użyć czegoś w rodzaju algorytmu A^* i w ten sposób znaleźć najlepsze rozwiązanie w najkrótszym czasie.

Musisz jednak wiedzieć, że po rozłożeniu problemu może się okazać, że nie jest łatwo zbudować pełną odpowiedź na podstawie uzyskanych odpowiedzi częściowych. Możesz rozłożyć TSP z 100 miastami na 20 TSP z 5 miastami, z których każdy jest łatwy do rozwiązania. Ale co zrobisz z tymi 20 odpowiedziami?! Większość problemów ze świata rzeczywistego nie daje się rozwiązać metodami tradycyjnymi. Stwierdzenie to jest niemalże tautologią – gdyby problemy te można było rozwiązać klasycznymi metodami, w ogóle nie stanowiłyby problemu. W związku z tym zwykle stajemy wobec sytuacji, w której mamy do czynienia z problemem niemożliwym do rozwiązania, mającym wiele lokalnie optymalnych rozwiązań w większości nie do zaakceptowania. Dla tego rodzaju problemów są potrzebne narzędzia wykraczające poza metody tradycyjne.

V. Jakiego koloru jest niedźwiedź?

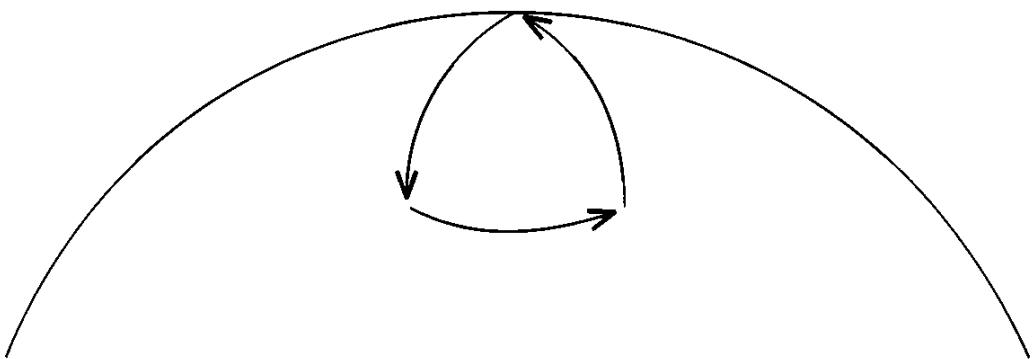
Gdy już znajdziesz rozwiązanie problemu, istnieje wielka pokusa, żeby dać sobie spokój. W końcu problem jest rozwiązany, prawda? No cóż, niezupełnie. Może się tak zdarzyć, że problem ten ma wiele rozwiązań, a niektóre z nich mogą być lepsze od innych. Podobnie jak w wypadku rozwiązań lokalnie optymalnych, posiadane rozwiązanie może się wydawać w porządku, lecz gdybyś tylko znalazł drogę do uzyskania jakiegoś innego rozwiązania, mógłbyś uciec z pułapki, w którą wpadłeś, i trafić na coś lepszego. Czasami w znalezieniu innych rozwiązań pomaga odwrócenie problemu do góry nogami. Mając to na uwadze, spróbuj rozwiązać zagadkę o kolorze niedźwiedzia i słynnym odkrywcy o nazwisku Brunatny.

Pewnego ranka pan Brunatny wyszedł ze swojego namiotu i przeszedł kilometr na południe. Następnie skręcił na wschód i przebył kolejny kilometr, po czym skręcił na północ i po przebyciu kolejnego kilometra znalazł się przed swoim namiotem. Podczas wędrówki pan Brunatny zobaczył niedźwiedzia. Jakiego był koloru?

Podobnie jak w wypadku innych problemów, możemy mieć trudności ze znalezieniem punktu wyjściowego. Szybko jednak dojdziemy, że klucz do zagadki musi tkwić w specyficznych cechach miejsca pobytu pana Brunatnego. Gdybyśmy wiedzieli, gdzie pan Brunatny się znajdował, może byłaby to informacja prowadząca do rozwiązania.

Pomyślawszy przez chwilę o geometrii kuli ziemskiej, masz dużą szansę stwierdzić, że namiot pana Brunatnego znajdował się na biegunie północnym. Wydaje się to jedynym miejscem, w którym można zatoczyć pętlę, idąc kilometr na południe, a następnie wschód i północ (zob. rys. V.1).

Oczywiście niedźwiedź musiał być biały: nie ma innych niedźwiedzi w okolicach bieguna północnego. Gdybyś miał się zakładać, myślałbyś: „Niedźwiedź chyba nie może być brunatny, skoro facet nazywa się Brunatny, prawda?”

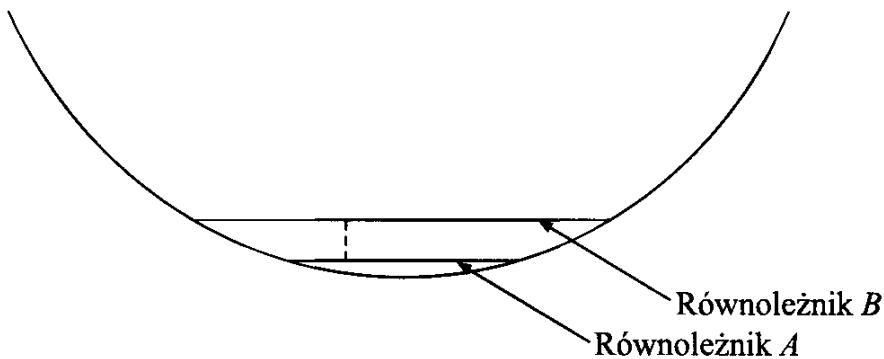


Rysunek V.1. Trasa pana Brunatnego

Ale czy to jest jedynie rozwiązań? Na pierwszy rzut oka wydaje się, że tak. Posuwając się na południe, nie ma możliwości zatoczenia pętli przez przejście kilometra na południe, wschód i północ.

Zastanowiwszy się dłużej, możemy jednak wpaść na inne rozwiązań!

Rozważmy równoleżnik na półkuli południowej mający obwód jednego kilometra (równoleżnik *A* na rys. V.2). Możemy wybrać dowolny punkt na tym równoleżniku i przejść kilometr na północ do kolejnego równoleżnika (równoleżnika *B* na rys. V.2). Jest to możliwa lokalizacja namiotu pana Brunatnego. Po opuszczeniu go idzie on kilometr na południe (docierając w ten sposób do równoleżnika *A*), skręca na wschód i maszeruje kilometr (zataczając pełne koło), po czym idzie na północ prosto do swojego namiotu. Namiot pana Brunatnego może stać w dowolnym miejscu na równoleżniku *B*. Tak więc okazuje się, że problem ma nie jedno rozwiązań, ale *nieskończonie wiele rozwiązań!*



Rysunek V.2. Możliwa lokalizacja namiotu pana Brunatnego

Czy teraz to już koniec? Czy znaleźliśmy całą nieskończoność rozwiązań?

Wcale nie! Po chwili namysłu okazuje się, że wciąż są możliwe inne rozwiązania, gdyż obwód równoleżnika *A* może wynosić $1/2$ kilometra, $1/3$ kilometra itd. W każdym z tych przypadków pan Brunatny okrążałby go kilka razy, zawsze wracając do punktu wyjścia. Ponieważ jednak w okolicach bieguna południowego nie ma żadnych niedźwiedzi, dodatkowe nieskończonie wiele rozwiązań nie zmienia odpowiedzi na postawione w zagadce pytanie, które – przypomnijmy – dotyczyło koloru niedźwiedzia napotkanego przez pana Brunatnego.

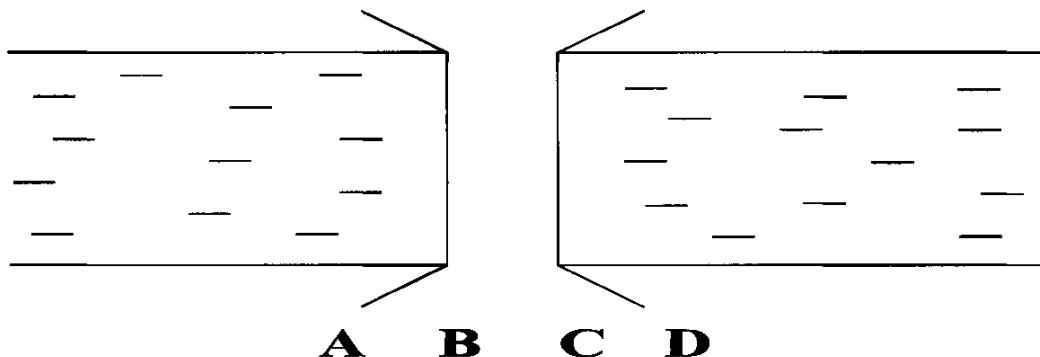
Jest jeszcze inna słynna (i bardzo błędna) wersja powyższej zagadki. Brzmi ona następująco: Pan Brunatny, słynny odkrywca, świętował swoje siedemdziesiąte urodziny. Podczas przyjęcia jakaś dama spytała go, co robił do-

kładnie dziesięć lat wcześniej. Po chwili namysłu pan Brunatny odparł: „Pamiętam ten dzień bardzo dobrze. Wyszedłem z namiotu o świcie i przeszedłem kilometr na południe. Następnie skręciłem na wschód i przebyłem kolejny kilometr, po czym skręciłem na północ i po przebyciu kolejnego kilometra znalazłem się przed swoim namiotem”. Pytanie brzmi: kiedy pan Brunatny ma urodziny?

W zagadce założono, że namiot może być tylko w jednym miejscu: na biegunie północnym. Ponieważ słońce wschodzi tam tylko raz w roku, 21 marca, pan Brunatny musi obchodzić urodziny tego dnia. Jak już jednak wiemy, istnieje wiele możliwych lokalizacji jego namiotu!

Morał z tej zagadki jest taki: szukaj dalej, nawet gdy masz już rozwiązanie. Myśl! Sprawdź, czy naprawdę wykorzystałeś wszystkie możliwości. Możesz natrafić na coś, czego początkowo nie zauważyleś, a co może okazać się lepsze od Twojego pierwszego rozwiązania. Nie masz na to szansy, jeśli oddasz się od razu. Pomocne jest także naszkicowanie sobie problemu w głowie. Gdy wyobrażisz sobie pana Brunatnego wędrującego z czekanem i teleskopem przez dzikie ostupy, sceneria ta szybciej może przekształcić się w krajobraz pełen kry i lodu. Jeśli nie masz zbyt bujnej wyobraźni, narysuj obrazek na kartce papieru. Sama możliwość przyglądania się czemuś może pomóc w sformułowaniu problemu i rozmyślaniach nad jego rozwiązaniem.

Wypróbujmy ten sposób i zacznijmy opis kolejnej zagadki od obrazka.



Rysunek V.3. Czterech podróżnych przed mostem

Czterech podróżnych (nazwijmy ich panami A, B, C i D) musi przejść przez most nad głębokim wąwozem (rys. V.3). Jest ciemna noc, a mężczyźni mają tylko jedno źródło światła: starą lampę naftową. Światło jest niezbędne do szczęśliwego przejścia mostu, ponieważ jest on bardzo stary i ma mnóstwo dziur oraz ruchomych desek. Co gorsza, jego nadwyróżona konstrukcja i opłakany stan sprawiają, że mogą na nim przebywać jednocześnie najwyższej dwie osoby. Pytanie brzmi: jak podróżni powinni zorganizować przeprawę? W lampie jest ograniczona ilość nafty, a czas ucieka.

Jeśli na moście może przebywać jednocześnie tylko dwóch podróżnych, to znaczy, że muszą pokonywać go pojedynczo lub parami. Ponieważ jednak muszą też mieć ze sobą lampa, żeby uniknąć wpadnięcia w jakąś dziurę, nie mogą przechodzić przez most pojedynczo. Niezależnie więc od tego, jak wygląda ostateczne rozwiązanie, musi ono składać się z ciągu par przechodzących przez most. Ale jakich par?

Dochodzimy teraz do bardzo interesującej części zagadki. Okazuje się, że każdy z podróżnych potrzebuje innego czasu do przejścia mostu. Pan A jest młody i zdrowy i potrzebuje tylko minuty, żeby znaleźć się po drugiej stronie wąwozu. Pan D z kolei to starszy jegomość, który ostatnio przeszedł operację biodra i potrzebuje dziesięciu minut, żeby przejść przez most. Panom B i C przeprawa zajmie odpowiednio dwie i pięć minut. Ponieważ każdy podróżny potrzebuje światła podczas wędrówki, w każdej parze przechodzącej przez most mężczyzna idący wolniej przesądza o czasie potrzebnym na ukończenie przejścia.

Teraz masz już wszystkie informacje. (Pomyśl: Czy naprawdę wszystkich potrzebujesz? A może chciałbyś wiedzieć coś jeszcze?) Jak zaplanować przeprawę wszystkich czterech mężczyzn przez most w najkrótszym czasie?

Twoim pierwszym pomysłem może być wysłanie mężczyzny idącego najszybciej, czyli pana A, z każdym z pozostałych podróżnych po kolej. Pan A mógłby nieść lampa. Panowie A i B mogliby przejść przez most razem, co zajęłoby dwie minuty, następnie pan A wróciłby z lampą w ciągu kolejnej minuty. Potem pan A przeprowadziłby się z panem C, żeby wrócić po pana D. Wszystko to zabraloby w sumie 19 minut. Jest jednak lepszy sposób wykonania tego zadania. Czy potrafisz go znaleźć?

Gdy już rozwiążesz tę zagadkę, rozwiązywanie innych podobnych powinno być łatwe. Weźmy na przykład zadanie, w którym sześciu podróżnych zbliża się do mostu, a ich czasy przechodzenia go wynoszą odpowiednio 1, 2, 4, 6, 8 i 9 minut. Jak tym razem wygląda najlepsza, czyli dająca najkrótszy czas, kolejność przechodzenia ich na drugą stronę? Przypuśćmy jeszcze, że zamiast sześciu mamy siedmiu podróżnych o czasach przechodzenia mostu: 1, 2, 6, 7, 8, 9 i 10 minut, ale tym razem most jest nowocześniejszy i może na nim przebywać jednocześnie trzech podróżnych...

5. Unikanie lokalnych optimów

O, gdybym miała konia skrzydłatego!

William Shakespeare: *Cymbelin*¹

Omówiliśmy kilka tradycyjnych metod rozwiązywania problemów. Niektóre z nich gwarantują znalezienie globalnego rozwiązania, a inne nie, ale wszystkie działają podobnie. Albo gwarantują znalezienie globalnego rozwiązania, ale są zbyt drogie (tzn. zbyt czasochłonne) przy rozwiązywaniu rzeczywistych problemów, albo mają tendencję do „utykania” w lokalnych optimach. Ponieważ właściwie nie ma sposobu na przyspieszenie algorytmów gwarantujących znalezienie globalnego rozwiązania, to znaczy, nie ma sposobu na znalezienie algorytmów działających w czasie wielomianowym dla większości rzeczywistych problemów (gdź z reguły są one NP-trudne), jedyne, co nam pozostaje, to projektowanie algorytmów zdolnych do unikania lokalnych optimów.

Jak to zrobić? Jak zaprojektować „konia skrzydłatego”? Poznaliśmy już jedną możliwość w rozdziale 2, gdzie omawialiśmy procedurę zwaną „iteracyjnym wspinaniem się”. Po osiągnięciu lokalnego optimum jest wybierany nowy punkt wyjściowy i poszukiwanie rozpoczyna się od nowa. Tę metodę możemy zastosować przy każdym innym algorytmie, rozważmy jednak kilka możliwości unikania lokalnych optimów w ramach jednego uruchomienia algorytmu.

Dwa główne sposoby, które przedstawimy w tym rozdziale, są oparte na: 1) dodatkowym parametrze (zwany *temperaturą*), który zmienia prawdopodobieństwo przejścia z jednego punktu w przestrzeni przeszukiwania do innego lub 2) pamięci, która zmusza algorytm do badania nowych obszarów przestrzeni przeszukiwania. Sposoby te nazywamy odpowiednio: *symulowanym wyżarzaniem i poszukiwaniem z tabu*. Omawiamy je dokładnie w kolejnych dwóch podrozdziałach, najpierw jednak zajmijmy się pewną intuicją związaną z tymi metodami i porównajmy je z prostą procedurą poszukiwania lokalnego.

¹*Dzieła dramatyczne*, tom 6, przekład Leon Ulrich, PIW 1973 (przyp. tłum.).

Procedurę iteracyjnego wspinania się omówiliśmy w rozdziale 2. Przepiszmy ją teraz, pomijając kilka wierszy programistycznych szczegółów i pozostawiając tylko najważniejsze koncepcje. Ograniczymy też algorytm do jednego ciągu ulepszających kroków (tzn. $MAX = 1$; rys. 2.5). W takim wypadku jeden przebieg prostej procedury *poszukiwania lokalnego* można opisać następująco:

```
procedure poszukiwanie lokalne
begin
     $x$  = pewien punkt początkowy w  $\mathcal{S}$ 
    while ulepsz( $x$ ) ≠ „nie” do
         $x$  = ulepsz( $x$ )
    return( $x$ )
end
```

Podprocedura $ulepsz(x)$ daje w wyniku nowy punkt y z otoczenia punktu x , tzn. $y \in N(x)$, jeśli y jest lepsze od x , w przeciwnym razie daje napis „nie” i wówczas x jest lokalnym optimum w \mathcal{S} .

Z kolei procedurę *symulowanego wyżarzania* (omówioną szczegółowo w podrozdziale 5.1) można opisać następująco:

```
procedure symulowane wyżarzanie
begin
     $x$  = pewien punkt początkowy w  $\mathcal{S}$ 
    while not warunek zakończenia do
         $x$  = ulepsz?( $x, T$ )
        uaktualnij( $T$ )
    return( $x$ )
end
```

Są trzy zasadnicze różnice między symulowanym wyżarzaniem a poszukiwaniem lokalnym. Pierwszą z nich jest sposób zatrzymywania się procedury. Symulowane wyżarzanie jest wykonywane dopóty, dopóki nie zostanie spełniony jakiś zewnętrzny „warunek zakończenia”, w przeciwieństwie do wymogu lokalnego poszukiwania, żeby znaleźć jakieś ulepszenie. Drugą różnicę stanowi to, że funkcja „ulepsz?(x, T)” nie musi dać lepszego punktu z otoczenia x ¹. Daje ona jedynie zaakceptowane rozwiązanie y z otoczenia x , przy czym akceptacja jest oparta na aktualnej temperaturze T . Trzecia różnica polega na tym, że w symulowanym wyżarzaniu parametr T jest uaktualniany okresowo, a jego wartość ma wpływ na wynik procedury „ulepsz?”. Cechą ta nie występuje w poszukiwaniu lokalnym. Zauważmy też, że powyższy zarys symulowanego wyżarzania jest bardzo uproszczony, żeby odpowiadał uproszczeniom, które poczyniliśmy dla poszukiwania lokalnego. Pominęliśmy, na przykład, proces inicjowania i częstotliwość zmiany parametru temperatury T . Naszym głównym celem było podkreślenie podobieństw i różnic.

¹Dlatego nazwę tej procedury opatrzyliśmy znakiem zapytania: „ulepsz?”.

Poszukiwanie z tabu (omówione szczegółowo w podrozdziale 5.2) jest prawie takie samo jak symulowane wyżarzanie, z wyjątkiem struktury algorytmu. Podobnie jak w symulowanym wyżarzaniu, funkcja „ulepsz?(x, H)” daje *zaakceptowane rozwiązanie* y z otoczenia x , które nie musi być lepsze od x , ale akceptacja jest oparta na historii poszukiwania H . Wszystko inne jest takie samo (przynajmniej z ogólnego punktu widzenia). Procedurę można opisać następująco:

```

procedure poszukiwanie z tabu
begin
     $x$  = pewien punkt początkowy w  $S$ 
    while not warunek zakończenia do
         $x$  = ulepsz?( $x, H$ )
        uaktualnij( $H$ )
        return( $x$ )
    end
```

Mając na uwadze te przewidywania i rozumiejąc podstawowe założenia, możemy przejść do bardziej szczegółowego omówienia tych dwóch interesujących metod poszukiwania.

5.1. Symulowane wyżarzanie

Przypomnimy teraz dokładną strukturę iteracyjnego wspinania się (rozdział 2), ponieważ symulowane wyżarzanie niewiele się od niego różni.

Zauważ, że wewnętrzna pętla procedury (rys. 5.1) *awsze* daje lokalne optimum. Procedura ta może „uciec” z lokalnego optimum tylko poprzez rozpoczęcie nowego poszukiwania (zewnętrzna pętla) z nowego (przypadkowego) miejsca. Wykonywanych jest tutaj *MAX* prób. Najlepsze rozwiązanie (*best*) uważane jest za ostateczny wynik algorytmu.

Zmodyfikujmy tę procedurę w następujący sposób:

- zamiast sprawdzać *wszystkie* ciągi w otoczeniu bieżącego punktu v_c i wybierać najlepszy, wybierzmy z tego otoczenia tylko jeden punkt v_n ;
- zaakceptujmy ten nowy punkt, tzn. $v_c \leftarrow v_n$ z prawdopodobieństwem zależnym od wartości względnej tych dwóch punktów, tzn. różnicę między wartościami danymi przez funkcję oceny dla tych dwóch punktów.

Dzięki tej prostej modyfikacji otrzymujemy nowy algorytm zwany *stochastycznym wspinaniem się* (rys. 5.2):

Omówmy niektóre cechy tego algorytmu (zauważ, że probabilistyczna formuła akceptowania nowego rozwiązania jest oparta na maksymalizowaniu funkcji oceny). Po pierwsze, stochastyczne wspinanie się ma tylko jedną pętlę. Nie musimy powtarzać iteracji, startując z różnych losowo wybranych punktów. Po drugie, nowo wybrany punkt jest „akceptowany” z pewnym prawdopodobieństwem p . Oznacza to, że zasada przechodzenia z bieżącego

```

procedure iteracyjne wspinanie się
begin
     $t \leftarrow 0$ 
    inicjuj best
    repeat
        local  $\leftarrow$  FALSE
        wybierz losowo bieżący punkt  $\mathbf{v}_c$ 
        oceń  $\mathbf{v}_c$ 
        repeat
            wybierz wszystkie nowe punkty z otoczenia  $\mathbf{v}_c$ 
            wybierz punkt  $\mathbf{v}_n$  ze zbioru nowych punktów
            o najlepszej wartości funkcji oceny eval
            if eval( $\mathbf{v}_n$ ) jest lepsze od eval( $\mathbf{v}_c$ )
                then  $\mathbf{v}_c \leftarrow \mathbf{v}_n$ 
                else local  $\leftarrow$  TRUE
            until local
             $t \leftarrow t + 1$ 
            if  $\mathbf{v}_c$  jest lepsze od best
                then best  $\leftarrow \mathbf{v}_c$ 
        until  $t = MAX$ 
    end

```

Rysunek 5.1. Struktura iteracyjnego wspinania się

```

procedure stochastyczne wspinanie się
begin
     $t \leftarrow 0$ 
    wybierz losowo bieżący ciąg  $\mathbf{v}_c$ 
    oceń  $\mathbf{v}_c$ 
    repeat
        wybierz ciąg  $\mathbf{v}_n$  z otoczenia  $\mathbf{v}_c$ 
        wybierz  $\mathbf{v}_n$  z prawdopodobieństwem  $\frac{1}{1+e^{\frac{\text{eval}(\mathbf{v}_c)-\text{eval}(\mathbf{v}_n)}{T}}}$ 
         $t \leftarrow t + 1$ 
    until  $t = MAX$ 
end

```

Rysunek 5.2. Struktura stochastycznego wspinania się

punktu \mathbf{v}_c do nowego sąsiada \mathbf{v}_n jest probabilistyczna. Nowo zaakceptowany punkt może być *gorszy* od punktu bieżącego. Zauważ jednak, że prawdopodobieństwo akceptacji zależy od różnicy wartości tych dwóch punktów, tzn. $\text{eval}(\mathbf{v}_c) - \text{eval}(\mathbf{v}_n)$ i wartości dodatkowego parametru T . Co więcej, T pozosta-

je stałe podczas wykonywania algorytmu. Przyjrzyjmy się prawdopodobieństwu akceptacji:

$$p = \frac{1}{1 + e^{\frac{\text{eval}(\mathbf{v}_c) - \text{eval}(\mathbf{v}_n)}{T}}}$$

i zastanówmy się, jaka jest rola parametru T ? Założymy, na przykład, że ocena dla bieżącego punktu i następnego punktu wynosi odpowiednio: $\text{eval}(\mathbf{v}_c) = 107$ i $\text{eval}(\mathbf{v}_n) = 120$ (pamiętaj, że formula ta odnosi się tylko do problemów maksymalizacji; podczas minimalizowania musisz zamienić miejscami odjemną i odjemnik podczas odejmowania). Różnica w naszym przykładzie wynosi $\text{eval}(\mathbf{v}_c) - \text{eval}(\mathbf{v}_n)$, czyli -13 , co oznacza, że nowy punkt \mathbf{v}_n jest lepszy od \mathbf{v}_c . Jakie jest prawdopodobieństwo zaakceptowania tego nowego punktu na podstawie różnych wartości T ? W tabeli 5.1 wyszczególniono kilka konkretnych przypadków.

Tabela 5.1. Prawdopodobieństwo p akceptacji jako funkcja zależna od T wypadku, gdy nowa próba \mathbf{v}_n jest o 13 punktów lepsza od bieżącego rozwiązania \mathbf{v}_c

T	$e^{\frac{-13}{T}}$	p
1	0,000002	1,00
5	0,0743	0,93
10	0,2725	0,78
20	0,52	0,66
50	0,77	0,56
10^{10}	0,9999...	0,5...

Wniosek jest oczywisty: im większa jest wartość T , tym mniejsze znaczenie ma wartość względna rywalizujących punktów \mathbf{v}_c i \mathbf{v}_n ! W szczególności, jeśli wartość T jest duża (np. $T = 10^{10}$), to prawdopodobieństwo akceptacji jest bliskie 0,5. Poszukiwanie staje się losowe. Z kolei, jeśli wartość T jest bardzo mała (np. $T = 1$), to stochastyczne wspinanie się przechodzi w zwykłe wspinanie się! A zatem zawsze musimy znaleźć odpowiednią wartość parametru T dla konkretnego problemu, która nie będzie ani za mała, ani za duża.

Żeby uzyskać jeszcze szerszą perspektywę, założymy, że $T = 10$ dla danego przebiegu algorytmu. Przyjmijmy też, że bieżące rozwiązanie \mathbf{v}_c zostało ocenione na 107, tzn. $\text{eval}(\mathbf{v}_c) = 107$. Wówczas prawdopodobieństwo akceptacji zależy jedynie od wartości nowego ciągu \mathbf{v}_n , jak pokazano w tab. 5.2.

Mamy teraz już pełny obraz: jeśli nowy punkt ma taką samą wartość jak punkt bieżący, tzn. $\text{eval}(\mathbf{v}_c) = \text{eval}(\mathbf{v}_n)$, to prawdopodobieństwo akceptacji wynosi 0,5. Ma to dużo sensu. Nie ma znaczenia, który z tych punktów wybierzemy, gdyż oba są jednakowej jakości. Ponadto, jeśli nowy punkt jest lepszy, to prawdopodobieństwo akceptacji jest większe od 0,5. Co więcej, prawdopodobieństwo akceptacji wzrasta wraz z (ujemną) różnicą między ocenami obu punktów. W szczególności, jeśli nowe rozwiązanie jest dużo lepsze od bieżącego (powiedzmy: $\text{eval}(\mathbf{v}_n) = 150$), to prawdopodobieństwo akceptacji jest bliskie 1.

Tabela 5.2. Prawdopodobieństwo akceptacji jako funkcja $eval(\mathbf{v}_n)$ dla $T = 10$ i $eval(\mathbf{v}_c) = 107$

$eval(\mathbf{v}_n)$	$eval(\mathbf{v}_c) - eval(\mathbf{v}_n)$	$e^{\frac{eval(\mathbf{v}_c) - eval(\mathbf{v}_n)}{10}}$	p
80	27	14,88	0,06
100	7	2,01	0,33
107	0	1,00	0,50
120	-13	0,27	0,78
150	-43	0,01	0,99

Główną różnicą między stochastycznym wspinaniem się a symulowanym wyżarzaniem jest to, że druga z tych metod zmienia parametr temperatury T w trakcie działania. Rozpoczyna się ona z dużą wartością T , co czyni procedurę zbliżoną do czysto losowego poszukiwania, a następnie stopniowo obniża wartość T . Pod koniec działania wartości T są całkiem małe, więc ostatnie etapy symulowanego wyżarzania przypominają zwyczajne wspinanie się. Co więcej, zawsze akceptujemy nowe punkty, jeśli są one lepsze od punktu bieżącego. Procedurę *symulowanego wyżarzania* pokazano na rys. 5.3 (ponownie założyliśmy, że mamy do czynienia z problemem maksymalizacji).

```

procedure symulowane wyżarzanie
begin
     $t \leftarrow 0$ 
    inicjuj  $T$ 
    wybierz losowo bieżący punkt  $\mathbf{v}_c$ 
    oceń  $\mathbf{v}_c$ 
repeat
repeat
    wybierz nowy punkt  $\mathbf{v}_n$ 
    w otoczeniu  $\mathbf{v}_c$ 
    if  $eval(\mathbf{v}_c) < eval(\mathbf{v}_n)$ 
        then  $\mathbf{v}_c \leftarrow \mathbf{v}_n$ 
    else if  $random[0, 1] < e^{\frac{eval(\mathbf{v}_n) - eval(\mathbf{v}_c)}{T}}$ 
        then  $\mathbf{v}_c \leftarrow \mathbf{v}_n$ 
    until (warunek zakończenia)
     $T \leftarrow g(T, t)$ 
     $t \leftarrow t + 1$ 
until (kryterium zatrzymania)
end

```

Rysunek 5.3. Struktura symulowanego wyżarzania

Symulowane wyżarzanie, znane również jako wyżarzanie typu Monte Carlo, schładzanie statystyczne, probabilistyczne wspinanie się, relaksacja sto-

chastyczna czy probabilistyczny algorytm wymiany, jest oparte na analogii zaczerpniętej z termodynamiki. Źeby wyhodować kryształ, ogrzewa się najpierw szereg materiałów do stanu ciekłego. Następnie obniża się temperaturę tego *kryształowego wytopu* dopóty, dopóki struktura kryształu nie zostanie *zamrożona*. Niedobrze jest, jeśli chłodzenie postępuje zbyt szybko. W strukturze kryształu pojawiają się pewne nieregularności, a poziom uwiezionej energii jest znacznie wyższy niż w idealnie uformowanym krysztale¹. Analogia między tym fizycznym procesem a problemem optymalizacji powinna być oczywista. Podstawowe „odpowiadające sobie” pojęcia są wymienione w tab. 5.3.

Tabela 5.3. Analogie między procesem fizycznym a problemem optymalizacji

Proces fizyczny	Problem optymalizacji
stan	dopuszczalne rozwiązanie
energia	funkcja oceny
stan kryształu	optymalne rozwiązanie
szbkie schładzanie	poszukiwanie lokalne
temperatura	parametr sterujący T
ostrożne wyżarzanie	symulowane wyżarzanie

Podobnie jak w wypadku dowolnego algorytmu poszukiwania, symulowane wyżarzanie wymaga znalezienia odpowiedzi na następujące pytania, bezpośrednio związane z rozważanym problemem (zob. rozdział 2):

- Jakie jest rozwiązanie?
- Jacy są sąsiedzi rozwiązania?
- Jaki jest koszt rozwiązania?
- Jak ustalić początkowe rozwiązanie?

Odpowiedzi na te pytania dostarczają nam strukturę przestrzeni przeszukiwania wraz z definicją otoczenia, funkcję oceny oraz punkt początkowy. Zauważmy jednak, że symulowane wyżarzanie wymaga także odpowiedzi na dodatkowe pytania:

- Jak ustalić początkową „temperaturę” T ?
- Jak ustalić współczynnik schładzania $g(T, t)$?
- Jak ustalić warunek zakończenia iteracji?
- Jak ustalić kryterium zatrzymania algorytmu?

Temperatura początkowa T musi być ustalona przed wykonaniem procedury. Czy powinniśmy zacząć od $T = 100$, $T = 1000$, czy może od jeszcze innej wartości? W jaki sposób powinniśmy ustalić warunek zakończenia iteracji, po którym temperatura maleje i procedura wyżarzania powtarza się? Czy powinniśmy wykonać pewną liczbę iteracji, czy zastosować w zamian jakieś inne kryterium?

¹Podobny problem występuje w metalurgii podczas ogrzewania i schładzania metali.

Poza tym, jak bardzo czy też o ile powinna zmaleć temperatura? O jeden procent, a może mniej? I wreszcie, kiedy algorytm powinien się zatrzymać, tzn. jaka jest temperatura zamrażania?

W większości implementacji symulowanego wyżarzania jest stosowana prosta sekwencja kroków:

KROK 1: $T \leftarrow T_{max}$

wybierz losowo \mathbf{v}_c

KROK 2: wybierz punkt \mathbf{v}_n z otoczenia \mathbf{v}_c

if $eval(\mathbf{v}_n)$ jest lepsze od $eval(\mathbf{v}_c)$

then wybierz ten punkt ($\mathbf{v}_c \leftarrow \mathbf{v}_n$)

else wybierz go z prawdopodobieństwem $e^{\frac{-\Delta eval}{T}}$

repeat ten krok k_T razy

KROK 3: ustaw $T \leftarrow rT$

if $T \geq T_{min}$

then goto KROK 2

else goto KROK 1

Musimy ustalić wartości parametrów T_{max} , k_T , r i T_{min} , które oznaczają odpowiednio: temperaturę początkową, liczbę iteracji, współczynnik schładzania i temperaturę zamrażania. Zbadajmy kilka możliwości, stosując metodę symulowanego wyżarzania kolejno do SAT, TSP i NLP.

Spears [444] zastosował symulowane wyżarzanie (ang. *simulated annealing* – SA) do trudnych problemów spełnialności. Procedurę SA-SAT opisana w [444] przedstawiono na rys. 5.4. Jej struktura sterująca jest podobna do tej z GSAT (rozdział 3). Zmienna „tries” sterująca zewnętrzną pętlą SA-SAT odpowiada zmiennej i w GSAT (zob. rys. 3.5 w rozdziale 3). Zmienne te aktualniają na bieżąco liczbę niezależnych prób rozwiązyania problemu. Na początku każdej próby w SA-SAT wartość T jest ustawiana na T_{max} (ponieważ zmienna j jest ustawiona na zero) i jest losowane nowe przypadkowe wartościowanie boolowskie. Wewnętrzna pętla repeat wypróbowuje różne podstawienia przez probabilistyczne przełączanie zmiennych boolowskich. Prawdopodobieństwo przełączenia zależy od bieżącej temperatury i od ulepszenia δ , jakie daje to przełączenie. Jak zwykle, jeśli ulepszenie jest ujemne, istnieje mała szansa, że przełączenie zostanie zaakceptowane, i odwrotnie: dodatnie ulepszenie zwiększa prawdopodobieństwo akceptacji przełączenia.

Istnieje zasadnicza różnica między GSAT a SA-SAT, będąca podstawową różnicą między lokalnym poszukiwaniem a symulowanym wyżarzaniem: GSAT może wykonać ruch wstecz (tzn. zmniejszenie liczby spełnionych klauzul), jeśli inne ruchy nie są możliwe. Jednocześnie jednak GSAT nie może wykonać dwóch ruchów wstecz z rzędu, gdyż jeden ruch wstecz zakłada istnienie kolejnego ruchu ulepszającego! SA-SAT z kolei może wykonać dowolny ciąg ruchów wstecz, może więc unikać lokalnych optimów!

Pozostałe parametry SA-SAT mają takie samo znaczenie jak w innych implementacjach symulowanego wyżarzania. Parametr r wyraża szybkość spa-

```

procedure SA-SAT
begin
    tries  $\leftarrow 0$ 
    repeat
        v  $\leftarrow$  losowe wartościowanie boolowskie
         $j \leftarrow 0$ 
        repeat
            if v spełnia klauzule then return(v)
             $T = T_{max} \cdot e^{-j \cdot r}$ 
            for  $k = 1$  to liczba zmiennych do
            begin
                oblicz wartość ulepszenia (pogorszenia)  $\delta$  liczby
                klauzul, które stały się prawdziwe po przełączeniu  $v_k$ 
                przełącz zmienną  $v_k$  z prawdopodobieństwem
                 $(1 + e^{-\frac{\delta}{T}})^{-1}$ 
                v  $\leftarrow$  nowe wartościowanie, jeśli zostało wykonane
                przełączenie
            end
             $j \leftarrow j + 1$ 
        until  $T < T_{min}$ 
        tries  $\leftarrow$  tries +1
    until tries = MAX-TRIES
end

```

Rysunek 5.4. Struktura algorytmu SA-SAT

dania temperatury, tzn. szybkość, z jaką temperatura spada z T_{max} do T_{min} . Spadek ten jest spowodowany przez zwiększone j , gdyż

$$T = T_{max} \cdot e^{-j \cdot r}$$

Spears [444] użył

$$T_{max} = 0,30 \text{ i } T_{min} = 0,01$$

odpowiednio dla maksymalnej i minimalnej temperatury. Szybkość spadania temperatury r zależała od liczby zmiennych w problemie oraz od liczby zmiennych „tries” (r było odwrotnością iloczynu liczby zmiennych i liczby prób). Spears skomentował wybór tych parametrów następująco:

Wybór parametrów wymaga pewnych kompromisów. Dla danego ustawienia MAX-TRIES redukcja T_{min} i/lub zwiększenie T_{max} umożliwia przeprowadzenie większej liczby testów w trakcie jednej niezależnej próby, zmniejszając w ten sposób liczbę razów, gdy jest zwiększana częstość zmiennej przed osiągnięciem progu MAX-

-TRIES¹. W ten sposób, zwiększając lub zmniejszając zakres temperatur (lub zmniejszając szybkość spadania temperatury), zmniejszamy liczbę niezależnych prób, ale za to przy każdej z nich szukamy dokładniej. Sytuacja się odwraca, gdy zmniejszamy zakres (lub zwiększamy szybkość spadania temperatury). Nie jest jasne, czy na ogół lepiej jest wykonywać więcej niezależnych prób, czy też szukać dokładniej przy każdej z nich.

Doświadczalne porównanie GSAT i SA-SAT dla trudnych problemów spełnialności pokazuje, że SA-SAT zdaje się doprowadzać do spełniania przy najmniej tak wielu formuł jak GSAT, ale przy mniejszej ilości pracy [444]. Spears dostarczył również doświadczalnych dowodów, że względna przewaga SA-SAT wynikała z jego ruchów wstecz, umożliwiających unikanie lokalnych optimów.

Warto odnotować, że TSP był jednym z pierwszych problemów, do których zastosowano symulowane wyżarzanie! Co więcej, na TSP testowano też wiele nowych wariantów tej metody. Standardowy algorytm symulowanego wyżarzania jest w istocie taki sam jak przedstawiony na rys. 5.3, gdzie v_c jest trasą, a $eval(v_c)$ opisuje długość trasy v_c . Różnice między implementacjami symulowanego wyżarzania występują w: 1) metodach generowania początkowego rozwiązania, 2) definicji otoczenia danej trasy, 3) wyborze sąsiada, 4) metodach obniżania temperatury, 5) warunku zakończenia, 6) warunku zatrzymania algorytmu oraz 7) istnieniu fazy dodatkowej obróbki końcowej.

Wszystkie powyższe decyzje są ważne, a nie są przy tym wcale łatwe ani niezależne. Na przykład liczba kroków w każdej temperaturze powinna być proporcjonalna do wielkości otoczenia. Dla każdego z wymienionych aspektów symulowanego wyżarzania istnieje wiele możliwości. Możemy zacząć od przypadkowego rozwiązania lub za początkowe rozwiązanie wziąć wynik działania algorytmu poszukiwania lokalnego. Możemy zdefiniować otoczenia różnej wielkości i dołączyć fazę dodatkowej obróbki końcowej w miejscu, w którym algorytm lokalnego poszukiwania osiągnie lokalny szczyt (gdyż symulowane wyżarzanie nie gwarantuje, że zostanie wykonane w przypadku wszystkich schematów zmniejszania T).

Aby poznać bardziej szczegółowo różne możliwości zastosowania symulowanego wyżarzania dla TSP oraz przeczytać bardzo dobry opis metod przyspieszania algorytmu i innych ulepszeń, zajrzyj do [242].

Symulowane wyżarzanie można też z łatwością zastosować dla NLP. Ponieważ mamy tu do czynienia ze zmiennymi ciągłymi, otoczenie często jest definiowane na podstawie rozkładów Gaussa (dla każdej zmiennej), dla których średnia wypada w punkcie bieżącym, a standardowe odchylenie jest ustalone na jedną szóstą długości dziedziny zmiennej (w taki sposób, że długość od

¹Komentarz ten dotyczy takiej sytuacji, gdy w algorytmie SA-SAT staramy się zachować przynajmniej w przybliżeniu stałą liczbę ocen wartościowania; takie założenie umożliwia sensowne porównywanie jakości działania tej procedury przy różnych układach parametrów (przyp. tłum.).

środka dziedziny do każdego z brzegów równa się trzem standardowym odchyleniom). Tak więc, jeśli punkt bieżący wynosi

$$\mathbf{x} = (x_1, \dots, x_n)$$

przy czym $l_i \leq x_i \leq u_i$ dla $i = 1, \dots, n$, to sąsiad \mathbf{x} zwany \mathbf{x}' jest wybierany w taki sposób, że

$$x'_i \leftarrow x_i + N(0, \sigma_i)$$

przy czym $\sigma_i = (u_i - l_i)/6$ i $N(0, \sigma_i)$ jest niezależną liczbą losową uzyskaną zgodnie z rozkładem Gaussa o średniej zero i odchyleniu standardowym σ_i .

Na tym etapie możemy obliczyć zmianę wartości funkcji:

$$\Delta eval = eval(\mathbf{x}) - eval(\mathbf{x}')$$

Jeśli $\Delta eval > 0$ (założymy, że mamy do czynienia z problemem minimalizacji), to nowy punkt \mathbf{x}' jest akceptowany jako nowe rozwiązanie. W przeciwnym razie jest akceptowany z prawdopodobieństwem

$$e^{\frac{\Delta eval}{T}}$$

Parametr T ponownie oznacza bieżącą temperaturę.

W dziedzinie optymalizacji numerycznej kwestie związane z wygenerowaniem początkowego rozwiązania, zdefiniowaniem otoczenia danego punktu i wyborem konkretnych sąsiadów są proste. W typowej procedurze są używane przypadkowy start i rozkład Gaussa przy pracy z otoczeniami. Implementacje różnią się jednak pod względem metod obniżania temperatury, warunku zakończenia iteracji, kryterium zatrzymania algorytmu oraz istnienia fazy dodatkowej obróbki końcowej (np. możemy tam zastosować metodę gradientową, która szybko zlokalizuje lokalne optimum). Zauważ także, że ciągle dziedziny zapewniają dodatkową elastyczność: wielkość otoczenia może maleć wraz z temperaturą. Jeśli parametry σ_i maleją z czasem, poszukiwanie koncentruje się wokół punktu bieżącego, dając lepsze dostrojenie.

5.2. Poszukiwanie z tabu

Główna koncepcja leżąca u podstaw poszukiwania z tabu jest bardzo prosta. „Pamięć” zmusza algorytm poszukiwania do badania nowych obszarów przestrzeni przeszukiwania. Możemy zapamiętać kilka z analizowanych ostatnio rozwiązań – stają się one punktami tabu (zakazanymi) i są pomijane podczas podejmowania decyzji o wyborze kolejnego punktu. Zauważ, że poszukiwanie z tabu jest w gruncie rzeczy deterministyczne (w przeciwieństwie do symulowanego wyżarzania), ale można je wzbogacić o elementy probabilistyczne [186].

Najlepiej będzie wyjaśnić podstawowe pojęcia poszukiwania z tabu na przykładzie. Zaczniemy od zastosowania poszukiwania z tabu dla SAT. Następnie omówimy ogólnie różne interesujące aspekty tej metody, żeby zakończyć kolejnym przykładem zastosowania tej metody, tym razem dla TSP.

Załóżmy, że mamy do rozwiązania problem SAT z $n = 8$ zmiennymi. Dla danej formuły logicznej F poszukujemy zatem takiego wartościowania boolowskiego dla wszystkich ośmiu zmiennych, żeby cała formuła F dała wartość TRUE. Założymy dalej, że przyjęliśmy następujące początkowe wartościowanie dla $\mathbf{x} = (x_1, \dots, x_8)$:

$$\mathbf{x} = (0, 1, 1, 1, 0, 0, 0, 1)$$

Jak zwykle potrzebujemy jakieś funkcji oceny, która umożliwi korygowanie przebiegu poszukiwania. Możemy na przykład obliczyć sumę ważoną liczby spełnionych klauzul, przy czym waga zależy od liczby zmiennych w klauzuli. W takim wypadku funkcja oceny powinna być zmaksymalizowana (tzn. próbujemy sprawić, żeby wszystkie klauzule były spełnione). Przypuśćmy, że powyższe przypadkowe wartościowanie da wartość 27. Następnie musimy zbadać otoczenie \mathbf{x} składające się z ośmiu innych rozwiązań, z których każde można osiągnąć przez przełączenie jednego bitu w wektorze \mathbf{x} . Na tym etapie poszukiwania jest to działanie tożsame z procedurą wspinania się.

Załóżmy, że przełączenie trzeciej zmiennej daje najlepszą ocenę (powiedzmy, 31), tak więc ten nowy wektor daje bieżące najlepsze rozwiązanie. Nadszedł więc czas na wprowadzenie nowego aspektu poszukiwania z tabu, a mianowicie pamięci. Żeby zapisywać nasze działania (ruchy), potrzebujemy jakichś struktur danych. Będziemy zapamiętywać indeks przełączanej zmiennej, jak również „czas” wykonania przełączenia, żebyśmy mogli rozróżnić starsze i nowsze przełączenia. W przypadku problemu SAT potrzebujemy rejestracji znacznika czasu dla każdej pozycji wektora rozwiązania: wartość znacznika czasu dostarczy nam informacji o najnowszym przełączeniu na tej konkretnej pozycji. W ten sposób wektor M posłuży nam za pamięć. Wektor ten jest zainicjowany na 0, a następnie na dowolnym etapie poszukiwań pozycja

$$M(i) = j \quad (\text{gdy } j \neq 0)$$

może być interpretowana jako „ j jest najnowszą iteracją, podczas której i -ty bit został przełączony” (oczywiście $j = 0$ oznacza, że i -ty bit nigdy nie został przełączony). Przydatna może być zmiana tej interpretacji w taki sposób, żebyśmy mogli zamodelować dodatkowy aspekt pamięci: po pewnym czasie (tzn. po pewnej liczbie iteracji) informacja przechowywana w pamięci zostaje wykasowana. Przy założeniu, że dowolna informacja może być przechowywana w pamięci przez, powiedzmy, najwyżej pięć iteracji, nowa interpretacja pozycji

$$M(i) = j \quad (\text{gdy } j \neq 0)$$

może brzmieć: „ i -ty bit został przełączony $5 - j$ iteracji temu”. Przy takiej interpretacji na rys. 5.5 przedstawiono zawartość pamięci M po jednej iteracji. Przypomnijmy, że przełączenie na trzeciej pozycji dało najlepszy wynik. Zauważmy, że wartość „5” może być odczytana jako „przez następnych pięć iteracji bit na trzeciej pozycji jest niedostępny (tzn. tabu)”.

Warto zauważyć, że główną różnicą między tymi dwiema równoważnymi interpretacjami jest kwestia implementacji. W drugim podejściu jest odczytywana wartość jako liczba iteracji, podczas których dana pozycja jest

0	0	5	0	0	0	0	0
---	---	---	---	---	---	---	---

Rysunek 5.5. Zawartość pamięci po jednej iteracji

niedostępna dla jakiegokolwiek przełączenia. Przy takiej interpretacji wymagamy, żeby *wszystkie* niezerowe pozycje w pamięci zmniejszały się o 1 przy każdej iteracji, w celu ułatwienia procesu wymazania ich z pamięci po pięciu iteracjach. Z kolei pierwsza interpretacja po prostu przechowuje numer iteracji, w której nastąpiło najświeższe przełączenie na konkretnej pozycji, wymaga więc licznika bieżącej iteracji t , który jest porównywany z wartościami umieszczonymi w pamięci. Konkretnie, jeśli $t - M(i) > 5$, tzn. przełączenie na i -tej pozycji nastąpiło pięć iteracji temu (jeśli w ogóle), to powinno zostać zapomniane. Dlatego też interpretacja ta wymaga jedynie aktualizacji *jednej* pozycji w pamięci w trakcie jednej iteracji i zwiększenia licznika iteracji. W naszym przykładzie przyjmiemy drugą z opisanych powyżej interpretacji, ale w większości *implementacji* poszukiwania z tabu jest stosowana pierwsza z nich, ponieważ jest efektywniejsza.

Przyjmijmy, że po czterech kolejnych iteracjach wyboru najlepszego sąsiada (który niekoniecznie jest lepszy od punktu bieżącego i dlatego właśnie możemy uniknąć lokalnych optimów) oraz po wykonaniu odpowiedniego przełączenia zawartość pamięci jest taka, jak pokazano na rys. 5.6.

3	0	1	5	0	4	2	0
---	---	---	---	---	---	---	---

Rysunek 5.6. Zawartość pamięci po pięciu iteracjach

Liczby umieszczone w pamięci (rys. 5.6) dostarczają nam następujących informacji:

Bit 2, 5 i 8 są dostępne do przełączenia w dowolnym momencie. Bit 1 jest niedostępny przez trzy kolejne iteracje, bit 3 jest niedostępny tylko przez jedną iterację, bit 4 (który właśnie został przełączony) jest niedostępny przez pięć kolejnych iteracji itd.

Innymi słowy, najnowsze przełączenie (iteracja 5) miało miejsce na pozycji 4 (tzn. $M(4) = 5$: bit 4 został przełączony $5 - 5 = 0$ iteracji temu). Inne przełączone wcześniej bity to:

- bit 6 (w czwartej iteracji), (gdyż $M(6) = 4$)
- bit 1 (w trzeciej iteracji), (gdyż $M(1) = 3$)
- bit 7 (w drugiej iteracji), (gdyż $M(7) = 2$)
- bit 3 (w pierwszej iteracji)

W rezultacie bieżące rozwiązanie to $\mathbf{x} = (1, 1, 0, 0, 0, 1, 1, 1)$ i powiedzmy, że jego ocena wynosi 33. Zbadajmy teraz dokładnie otoczenie \mathbf{x} . Składa się ono z ośmiu rozwiązań:

$$\begin{aligned}\mathbf{x}_1 &= (0, 1, 0, 0, 0, 1, 1, 1) \\ \mathbf{x}_2 &= (1, 0, 0, 0, 0, 1, 1, 1) \\ \mathbf{x}_3 &= (1, 1, 1, 0, 0, 1, 1, 1) \\ \mathbf{x}_4 &= (1, 1, 0, 1, 0, 1, 1, 1) \\ \mathbf{x}_5 &= (1, 1, 0, 0, 1, 1, 1, 1) \\ \mathbf{x}_6 &= (1, 1, 0, 0, 0, 0, 1, 1) \\ \mathbf{x}_7 &= (1, 1, 0, 0, 0, 1, 0, 1) \\ \mathbf{x}_8 &= (1, 1, 0, 0, 0, 1, 1, 0)\end{aligned}$$

które odnoszą się odpowiednio do przełączeń na bitach od pierwszego do ósmego. Po ocenieniu każdego z rozwiązań znamy ich indywidualne wartości, ale poszukiwanie z tabu wykorzystuje pamięć, żeby algorytm poszukiwania badał nowe obszary przestrzeni przeszukiwania. Zapamiętane nowe przełączenia są tabu (zakazane) do wyboru nowego rozwiązania. Z tego powodu podczas następnej iteracji (iteracji 6) niemożliwe jest przełączenie bitów 1, 3, 4, 6 i 7, ponieważ wszystkie one były „na nowo” testowane. Te zakazane (tabu) rozwiązania (tzn. rozwiązania uzyskane z zakazanych przełączeń) nie są brane pod uwagę, tak więc nowe rozwiązanie jest wybierane spośród \mathbf{x}_2 , \mathbf{x}_5 i \mathbf{x}_8 .

Przypuśćmy, że najlepszą ocenę spośród tych trzech możliwości uzyskało \mathbf{x}_5 i wynosi ona 32. Zauważ, że wartość ta jest mniejsza niż wartość oceny bieżącego najlepszego rozwiązania. Po tej iteracji zawartość pamięci zmienia się następująco: wszystkie niezerowe wartości zmniejszają się o jeden, wskazując w ten sposób, że wszystkie zapisane przełączenia miały miejsce jedną generację wcześniej. W szczególności, wartość $M(3) = 1$ zmienia się w $M(3) = 0$, co oznacza, że po pięciu generacjach fakt przełączenia trzeciego bitu zostaje usunięty z pamięci. Ponieważ zaś wybrane nowe aktualne rozwiązanie wynikło z przełączenia piątego bitu, wartość $M(5)$ ulega zmianie z 0 na 5 (przez kolejnych pięć iteracji ta pozycja jest tabu). Zawartość pamięci po szóstej iteracji przedstawiono na rys. 5.7.

2	0	0	4	5	3	1	0
---	---	---	---	---	---	---	---

Rysunek 5.7. Zawartość pamięci po sześciu iteracjach

Następne iteracje są przeprowadzane w podobny sposób. Na każdym etapie jest przetwarzane bieżące rozwiązanie. Wyznacza ono określone otoczenie, z którego są eliminowane rozwiązania tabu.

Po dłuższym namyśle taka polityka może wydawać się zbyt restrykcyjna. Może się zdarzyć, że jeden z sąsiadów objętych tabu, powiedzmy \mathbf{x}_6 , daje świetny wynik w ocenie, lepszy od wyniku jakiegokolwiek rozważanego wcześniej rozwiązania. Może powinniśmy uelastycznić poszukiwanie? Jeśli znajdziemy znakomite rozwiązanie, możemy zapomnieć o zasadach¹! Żeby uczynić

¹Jest to typowe zachowanie w różnych rzeczywistych sytuacjach: na widok świetnej okazji łatwo zapomina się o pewnych zasadach!

poszukiwanie bardziej elastycznym, w poszukiwaniu z tabu rozważa się rozwiązania z całego otoczenia, ocenia je wszystkie i w „typowej” sytuacji wybiera rozwiązanie, które nie jest tabu, jako kolejne bieżące rozwiązanie, nie bacząc na to, czy rozwiązanie to ma większą wartość według funkcji oceny od bieżącego rozwiązania, czy też nie. Jeśli jednak sytuacja nie jest „typowa”, tzn. w otoczeniu zostało znalezione znakomite rozwiązanie tabu, to lepsze rozwiązanie jest brane jako kolejny punkt. Takie pogwałcenie klasyfikacji tabu zachodzi, gdy zostaje spełnione tak zwane *kryterium aspiracji*.

Istnieją także inne sposoby zwiększenia elastyczności poszukiwania. Możemy, na przykład, zamienić wcześniejszą deterministyczną procedurę selekcji na metodę probabilistyczną, w której lepsze rozwiązania mają większą szansę na wybranie. Możemy na dodatek zmienić horyzont pamięci podczas poszukiwania: czasami może się opłacać pamiętać „więcej”, a czasami „mniej” (np. kiedy algorytm wspina się w obiecującej części przestrzeni przeszukiwania). Możemy także powiązać ten horyzont pamięci z rozmiarem problemu (np. pamiętając ostatnich \sqrt{n} ruchów, przy czym n jest rozmiarem problemu).

Kolejna możliwość jest jeszcze bardziej interesująca! Omówiona dotychczas struktura pamięci może być określona jako pamięć *niedawnych wartości*, ponieważ zapisuje jedynie pewne działania z kilku ostatnich iteracji. Pamięć ta może być powiększona o tak zwaną pamięć *częstości użycia*, która działa z uwzględnieniem znacznie szerszego horyzontu. Na przykład (wracając do omawianego wcześniej problemu SAT) wektor H może służyć jako pamięć długotrwała. Wektor ten jest zainicjowany na 0, a następnie na dowolnym etapie poszukiwania pozycja

$$H(i) = j$$

jest interpretowana jako: „podczas ostatnich h iteracji algorytmu i -ty bit został przełączony j razy”. Z reguły wartość horyzontu h jest dość duża, przynajmniej w porównaniu z horyzontem pamięci niedawnych wartości. Stąd po – dajmy na to – stu iteracjach z $h = 50$ pamięć długotrwała H może zawierać wartości przedstawione na rys. 5.8.

5	7	11	3	9	8	1	6
---	---	----	---	---	---	---	---

Rysunek 5.8. Zawartość pamięci częstości użycia po stu iteracjach (horyzont $h = 50$)

Te liczniki częstości pokazują rozkład ruchów w ciągu ostatnich 50 iteracji. Jak możemy wykorzystać te informacje? Zasady poszukiwania z tabu wskazują, że ten typ pamięci może być przydatny do różnicowania poszukiwania. Na przykład pamięć częstości użycia informuje nas, które przełączenia rzadko się pojawiały lub nie pojawiały się wcale, a my możemy różnicować poszukiwanie przez badanie tych możliwości.

Użycie pamięci długotrwałej w poszukiwaniu z tabu ogranicza się zwykle do pewnych szczególnych sytuacji. Możemy, na przykład, spotkać się z sytuacją, w której wszystkie ruchy nieobjęte tabu prowadzą do gorszego rozwiązania.

Jest to rzeczywiście sytuacja wyjątkowa: wszystkie dozwolone ruchy prowadzą do gorszych rozwiązań! Aby zatem podjąć sensowną decyzję co do dalszego kierunku poszukiwań, warto odnieść się do zawartości pamięci długotrwałej.

Istnieje wiele możliwości spożytkowania tych informacji w procesie decyzyjnym. Najbardziej typowe podejście czyni najczęstsze ruchy najmniej atrakcyjnymi. Zazwyczaj wartość oceny jest pomniejszana o karę związaną z częstotliwością, a ostateczny wynik wyłania zwycięzcę.

Żeby zilustrować tę koncepcję przykładem, założymy, że wartość bieżącego rozwiązania \mathbf{x} dla problemu SAT wynosi 35. Wszystkie przełączenia niebędące tabu, powiedzmy na bitach 2, 3 i 7, dają odpowiednio wartości 30, 33 i 31, a żadne z przełączeń z tabu nie daje wartości większej niż 37 (czyli największej znalezionej dotychczas wartości), nie możemy więc zastosować kryterium aspiracji. W takim wypadku odwołujemy się do pamięci częstotliwości użycia (zob. rys. 5.8). Założymy, że użytką w tej sytuacji formuła oceny nowego rozwiązania \mathbf{x}' jest

$$\text{eval}(\mathbf{x}') - \text{penalty}(\mathbf{x}')$$

przy czym eval daje w wyniku wartość pierwotnej funkcji oceny (tzn. odpowiednio: 30, 33 i 31 dla rozwiązań powstały przez przełączenie drugiego, trzeciego i siódmego bitu), a

$$\text{penalty}(\mathbf{x}') = 0,7 \cdot H(i)$$

przy czym 0,7 jest współczynnikiem, a $H(i)$ jest wartością pobraną z pamięci długotrwałej H :

7 – dla rozwiązania powstałego przez przełączenie drugiego bitu

11 – dla rozwiązania powstałego przez przełączenie trzeciego bitu

1 – dla rozwiązania powstałego przez przełączenie siódmego bitu

Nowe wyniki dla tych trzech możliwych rozwiązań wynoszą zatem

$$30 - 0,7 \cdot 7 = 25,1, \quad 33 - 0,7 \cdot 11 = 25,3 \quad \text{i} \quad 31 - 0,7 \cdot 1 = 30,3$$

tak więc wybieramy trzecie rozwiązanie (tzn. przełączenie siódmego bitu).

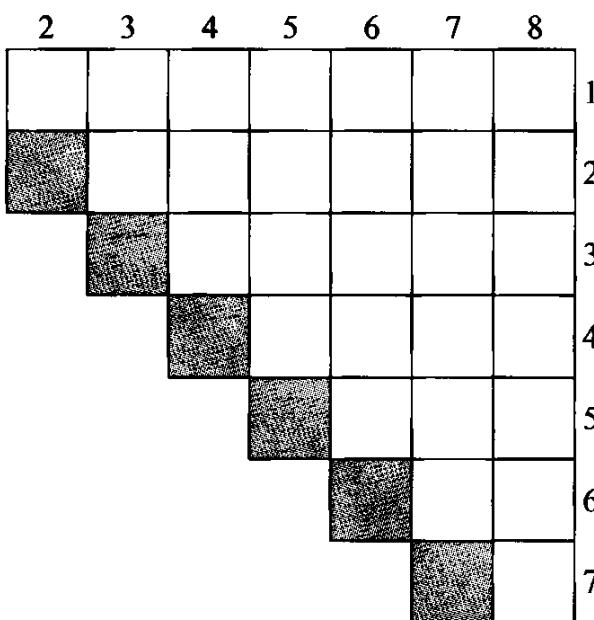
Powyższy sposób wykorzystania częstotliwości do pomniejszania o karną wartość wyniku oceny różnicuje poszukiwanie. W poszukiwaniu z tabu możemy rozważyć wiele innych możliwości. Jeśli musimy na przykład wybrać jakiś ruch objęty tabu, możemy zastosować dodatkową zasadę (tak zwaną *aspirację zgodną z domyślnym ruchem*), żeby wybrać „najstarszy” ze wszystkich rozważanych ruchów. Dobrym pomysłem może okazać się także zapamiętywanie nie tylko zbioru niedawnych ruchów, ale także informacji, czy ruchy te ulepszyły poszukiwanie. Informacje te mogą być następnie użyte przy podejmowaniu dalszych decyzji dotyczących poszukiwania (tak zwana *aspiracja zgodna z kierunkiem poszukiwań*). Idąc dalej, warto jest wprowadzić pojęcie *wpływ*, który określa stopień zmian nowego rozwiązania na podstawie odległości między starym a nowym rozwiązaniem albo zmianę w dopuszczalności rozwiązania, jeśli mamy do czynienia z problemem z ograniczeniami. Intuicja związana z wpływem jest taka, że dany ruch ma większy wpływ, jeśli odpowiadający mu „większy”

krok dzieli stare rozwiązanie od nowego. Tę informację również można wykorzystać w trakcie poszukiwania (tak zwana *aspiracja zgodna z wpływem*). Więcej szczegółów dotyczących licznych dostępnych możliwości w poszukiwaniu z tabu, a także praktyczne wskazówki dotyczące implementacji zawiera [186].

Na koniec tego podrozdziału podamy jeszcze jeden przykład struktur możliwych do użycia w poszukiwaniu z tabu w zastosowaniu do TSP. W przypadku tego problemu możemy rozważyć ruchy, które zamienią dwa miasta w danym rozwiązaniu¹. Następujące rozwiązanie (dla TSP złożonego z ośmiu miast):

$$(2, 4, 7, 5, 1, 8, 3, 6)$$

ma 28 sąsiadów, ponieważ istnieje 28 różnych par miast, tzn. $\binom{8}{2} = (7 \cdot 8)/2 = 28$, które możemy ze sobą zmienić. Tak więc dla pamięci niedawnych wartości możemy wykorzystać strukturę zaprezentowaną na rys. 5.9, gdzie zamiana miast i i j jest zarejestrowana jest w i -tym wierszu i j -tej kolumnie (dla $i < j$). Zauważ, że i oraz j interpretujemy jako *miasta*, a nie jako ich pozycje w wektorze rozwiązań, ale możemy rozważyć i taką możliwość. Zauważ także, że ta sama struktura może być zastosowana dla pamięci częstości użycia.



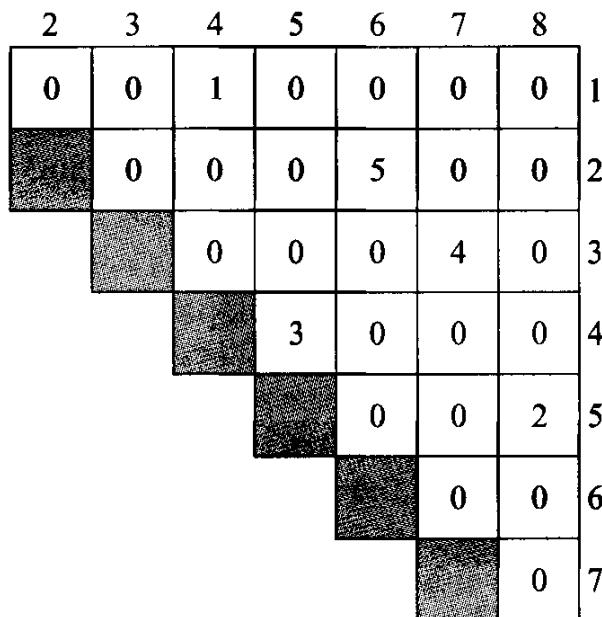
Rysunek 5.9. Struktura pamięci niedawnych wartości dla TSP

Dla jasności, będziemy utrzymywali w pamięci liczbę iteracji, przez które dana zmiana pozostała jeszcze na liście tabu (pamięć niedawnych wartości), podobnie jak w problemie SAT, podczas gdy pamięć częstości użycia wskaże nam liczby wszystkich zmian, jakie miały miejsce w obrębie jakiegoś horyzontu h . Przypuśćmy, że obie pamięci zostały zainicjowane na 0, po czym wykonano 500 iteracji poszukiwania. Bieżący stan poszukiwania może przedstawać się następująco: bieżące rozwiązanie to

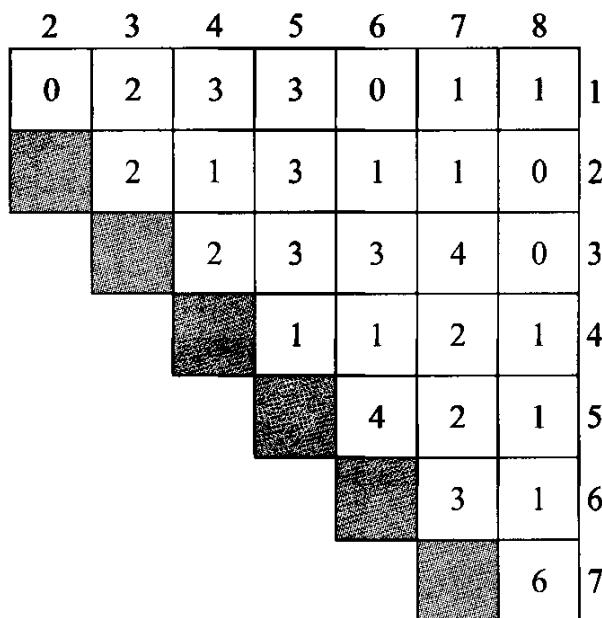
$$(7, 3, 5, 6, 1, 2, 4, 8)$$

¹WCale nie uważamy, że jest to najlepsza decyzja. Częściej na przykład bardziej zrozumiała jest zamiana krawędzi w trasie.

z całkowitą długością trasy równą 173. Najlepsze rozwiązanie znalezione podczas tych 500 iteracji daje wartość 171. Stan pamięci niedawnych wartości i pamięci częstości użycia przedstawiono na rys. 5.10 i 5.11.



Rysunek 5.10. Zawartość pamięci niedawnych wartości M dla TSP po 500 iteracjach. Horyzont ma zasięg pięciu iteracji



Rysunek 5.11. Zawartość pamięci częstości użycia F dla TSP po 500 iteracjach. Horyzont ma zasięg 50 iteracji

Podobnie jak w wypadku problemu SAT, możemy łatwo zinterpretować liczby w tych pamięciach. Wartość $M(2, 6) = 5$ oznacza, że najnowsza zamiana dotyczyła miast 2 i 6, czyli że poprzednie bieżące rozwiązanie prezentowało się następująco:

$$(7, 3, 5, 2, 1, 6, 4, 8)$$

Tak więc zamiana miast 2 i 6 stanowi *tabu* przez kolejnych pięć iteracji. Podobnie zamiany miast 1 i 4, 3 i 7, 4 i 5 oraz 5 i 8 znajdują się na liście tabu. Najstarsza spośród nich jest zamiana miast 1 i 4 (tzn. miała ona miejsce pięć iteracji temu) i zostanie usunięta z listy tabu po kolejnej iteracji. Zauważ, że tylko 5 zamian (z 28 możliwych) jest zakazanych (*tabu*).

Pamięć częstości użycia dostarcza nam dodatkowych danych statystycznych na temat poszukiwania. Dowiadujemy się z niej, że najczęstsza była zamiana miast 7 i 8 (zdarzyła się 6 razy podczas ostatnich 50 zamian), a niektóre pary miast (np. 3 i 8) nie zostały zamienione w trakcie ostatnich 50 iteracji.

Otoczenie trasy zostało zdefiniowane przez operację zamiany dwóch miast w trasie. Otoczenie to nie jest najlepszym wyborem ani dla poszukiwania z tabu, ani dla symulowanego wyżarzania. Knox [263] na przykład rozważał otoczenia oparte na zamianie *dwukrawędziowej* (zob. rys. 3.6), polegającej na usunięciu dwóch niemającecych punktów wspólnych krawędzi w bieżącej trasie i dodaniu dwóch innych krawędzi, żeby otrzymać nową dopuszczalną trasę. Zarys konkretnej implementacji poszukiwania z tabu, opisanej w [263], przedstawiono na rys. 5.12.

```

procedure poszukiwanie z tabu
begin
    tries ← 0
    repeat
        wygeneruj trasę
        count ← 0
        repeat
            zidentyfikuj zbiór  $T$  ruchów zamiany dwukrawędziowej
            wybierz najlepszy dopuszczalny ruch z  $T$ 
            wykonaj odpowiednią zamianę dwukrawędziową
            uaktualnij listę tabu i inne zmienne
            if nowa trasa jest dotychczas najlepsza przy danej wartości
                „tries”
            then uaktualnij informacje o lokalnej najlepszej trasie
            count ← count + 1
        until count = ITER
        tries ← tries + 1
        if bieżąca trasa jest dotychczas najlepsza (dla wszystkich
            „tries”)
        then uaktualnij informacje o globalnej najlepszej trasie
    until tries = MAX-TRIES
end
```

Rysunek 5.12. Konkretna implementacja poszukiwania z tabu dla TSP

Knox [263] uznawał trasę za tabu, jeśli *obie* dodane krawędzie biorące udział w zamianie znajdowały się na liście tabu. Lista tabu była uaktualniana przez umieszczanie na niej dodanych krawędzi (usunięte krawędzie były pominięte). Co więcej, lista tabu miała stały rozmiar. Gdy stawała się pełna, najstarszy element na liście był zastępowany nową usuniętą krawędzią. W momencie inicjowania lista była pusta, a wszystkie elementy listy aspiracyjnej były ustalone na duże wartości. Zauważ, że algorytm sprawdza *wszystkich* sąsiadów, to znaczy wszystkie zamiany dwukrawędziowe.

Knox [263] wykazał, że najlepsze wyniki były osiągane, gdy

- długość listy tabu wynosiła $3n$ (przy czym n oznacza liczbę miast w problemie);
- sprawdzana trasa mogła pominąć wynik tabu, jeśli obie krawędzie przeszły pomyślnie przez test aspiracji, porównujący długość trasy z wartościami aspiracyjnymi dla obu dodanych krawędzi; test był zdany, jeśli długość trasy była lepsza (tzn. mniejsza) niż *obie* wartości aspiracyjne;
- wartości na liście aspiracyjnej były kosztami podróży przed zamianą;
- liczba poszukiwań (MAX-TRIES) i liczba zamian (ITER) zależały od wielkości problemu; dla problemów ze 100 lub mniej miastami MAX-TRIES wynosiło 4, a ITER było ustawione na $0,0003 \cdot n^4$.

Istnieje wiele innych sposobów implementacji list tabu, kryteriów aspiracji, generowania początkowych tras itp. Podany przykład ilustruje tylko jedną z możliwych implementacji poszukiwania z tabu dla TSP. Możesz generować początkowe trasy losowo lub w jakiś inny sposób zapewniający różnorodność. Maksymalna liczba iteracji może być funkcją zależną od poczynionych dotychczas ulepszeń. Niewątpliwie możesz wyobrazić sobie dowolną liczbę innych możliwości.

5.3. Podsumowanie

Zarówno symulowane wyżarzanie, jak i poszukiwanie z tabu opracowano, żeby uniknąć lokalnych optimów. Różnią się one jednak sposobem osiągania tego celu. Poszukiwanie z tabu z reguły wykonuje ruchy do góry tylko wówczas, gdy utknie w lokalnym optimum, podczas gdy symulowane wyżarzanie może wspinać się w dowolnym momencie. Ponadto symulowane wyżarzanie jest algorytmem stochastycznym, poszukiwanie z tabu zaś – deterministycznym.

W porównaniu z klasycznymi algorytmami zaprezentowanymi w rozdziale 4 widzimy, że zarówno symulowane wyżarzanie, jak i poszukiwanie z tabu operują na pełnych rozwiązaniach (podobnie jak metody z rozdziału 3). Możesz zatrzymać te algorytmy na dowolnej iteracji i masz gotowe rozwiązanie. Możesz jednak zauważać subtelną różnicę między tymi metodami a metodami klasycznymi. Symulowane wyżarzanie i poszukiwanie z tabu mają więcej parametrów, które trzeba kontrolować, takich jak: temperatura, stopień redukcji,

pamięć itp. Klasyczne metody opracowano tak, aby po prostu „kręcić korbą”, żeby uzyskać odpowiedź, teraz musisz zacząć myśleć – nie tylko o tym, czy dany algorytm ma sens w przypadku Twojego problemu, ale także o tym, jak dobrać parametry algorytmu w taki sposób, żeby zapewnić mu optymalne działanie. Jest to zagadnienie dotyczące ogromnej większości algorytmów, które potrafią unikać lokalnych optimów. Im bardziej jest wyrafinowana metoda, tym bardziej musisz polegać na własnej ocenie co do sposobu, w jaki powinna ona być zastosowana.

VI. Jak dobrą masz intuicję?

W ludzkiej naturze leży próbowanie odgadnięcia rozwiązania skomplikowanego problemu. Niektóre z tych prób są bardziej intuicyjne niż inne. Nawet niektórzy ludzie wydają się obdarzeni lepszą intuicją niż inni! Ale nie są, prawda? Tak nam się tylko wydaje z powodu malej próbki, którą oceniamy. Mamy tendencję do zapamiętania szczęścia, który trafnie obstawił ruletkę trzy razy z rzędu, a zapominamy o setkach tych, którym podobny wyczyn się nie udał. Ta szczęśliwa osoba nie jest jednak jasnowidzem – to po prostu kwestia losowego próbowania. Gdybyś namówił naszego szczęścia, na ponowną grę i poprosił go o obstawienie trzech kolejnych numerów, prawdopodobieństwo trafnego wyboru będzie dokładnie takie samo jak przedtem: $(1/38)^3$, zakładając koło ruletki z 0 i 00.

A oto jedna ze słynnych sztuczek, na którą nabiera się ludzi żądnych wzbogacenia się na giełdzie. Nieuczciwy makler giełdowy wybiera 1024 osoby na swych potencjalnych klientów – choć słowo „frajerzy” byłoby tu bardziej na miejscu. Przez 10 kolejnych dni codziennie wysyła do każdego z nich list z informacją o przewidywanym wzroście lub spadku na giełdzie następnego dnia. Trik polega na tym, że w ciągu 10 dni istnieje 2^{10} różnych możliwych wyników i – wyobraźcie sobie – nasz makler wysyła każdemu jedno z tych prawdopodobnych rozwiązań. Po 10 dniach jedna pechowa ofiara doświadczy czegoś niebywałego: zobaczy, że makler za każdym razem przewidział poprawnie wzrost lub spadek na giełdzie! Ofiara będzie przekonana, że makler musi być jasnowidzem! Nawet Ci, którzy otrzymali 8 czy 9 poprawnych typowań, muszą być pod wrażeniem umiejętności maklera. A co z tymi, którzy otrzymali tyle samo dobrych i złych przewidywań, czy też szczęściami, którzy byli świadkami dziesięciu pomyłek maklera z rzędu? Zapomną o nim, ma się rozumieć! Teraz makler koncentruje się na ludziach, którym dostarczył właściwe przewidywania i próbuje wyłudzić od nich pieniądze. Uważaj więc na swoją intuicję i pilnuj portfela!

Czasem trzeba się nieźle namęczyć, żeby nie pójść za głosem intuicji. Nawet pozornie proste, codzienne wydarzenia wymagają uważnego zbadania. Spróbuj się zmierzyć z kilkoma takimi zadaniami.

Pierwsza zagadka jest bardzo prosta, ale pokazuje, że pojęcie „średniej” często jest źle rozumiane. Prowadzisz samochód ze stałą prędkością 40 km/h z Waszyngtonu do Nowego Jorku, po czym wracasz z większą prędkością – 60 km/h. Jaka jest Twoja średnia prędkość podczas całej podróży?

Prawdopodobnie przynajmniej 90% ludzi powie, że średnia prędkość wynosi 50 km/h, ponieważ intuicyjne wyczucie „średniej” nie jest do końca jasne. Prawidłowa odpowiedź – 48 km/h – wydaje się sprzeczna z intuicją! Zauważ, że „średnia prędkość” oznacza stosunek odległości do czasu, a więc

$$v_{avg} = \frac{D}{t}$$

przy czym D i t oznaczają odpowiednio całkowitą odległość i czas podróży. W takim razie $D = 2d$ (przy czym d oznacza odległość między Waszyngtonem a Nowym Jorkiem); zaś $t = t_{WN} + t_{NW}$ (przy czym t_{WN} i t_{NW} oznaczają odpowiednio czas podróży z Waszyngtonu do Nowego Jorku i z Nowego Jorku do Waszyngtonu). Zauważ też, że

$$t_{WN} = \frac{d}{40} \text{ oraz } t_{NW} = \frac{d}{60}$$

tak więc

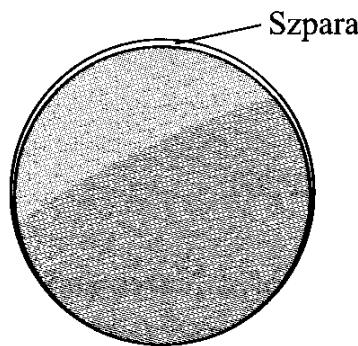
$$v_{avg} = \frac{D}{t} = \frac{2d}{t_{WN} + t_{NW}} = \frac{2d}{\frac{d}{40} + \frac{d}{60}} = \frac{2}{\frac{1}{40} + \frac{1}{60}} = 48$$

A oto inna wersja tej zagadki: wyobraź sobie, że jedziesz z Waszyngtonu do Nowego Jorku ze stałą prędkością 40 km/h. Jaka powinna być Twoja stała prędkość podczas podróży powrotnej, jeśli chcesz uzyskać średnią prędkość 80 km/h?

Przed nami kolejny przykład, który przeczy intuicji. Założmy, że w czasach starożytnej Grecji Zeus zlecił kowalowi wykonanie żelaznego pierścienia, który opasałby Ziemię. Zadanie polegało na tym, żeby średnica pierścienia odpowiadała dokładnie średnicy Ziemi¹. Kowal popełnił jednak błąd: wykuł pierścień, którego obwód był o metr dłuższy, niż powinien. Mimo to Zeus umieścił pierścień „dokoła” Ziemi w taki sposób, że dotykał jej w jednym punkcie (rys. VI.1). Pytanie brzmi: Jak bardzo pierścień „odstawał” po drugiej stronie Ziemi? Jakie zwierzę mogłoby przecisnąć się przez szparę między pierścieniem a Ziemią? Mrówka? Mysz? Szczur? Kot?

Okazuje się, że szpara ma prawie 32 cm i jest to rozwiązanie sprzeczne z intuicją! To naprawdę dużo! Nawet małe dziecko mogłoby przecisnąć się pod

¹Zakładamy tu, że Ziemia jest idealną kulą!



Rysunek VI.1. Ziemia i pierścień

pierścieniem! Szpara jest tak duża, ponieważ różnica obwodów Ziemi i pierścienia wynosi 100 cm:

$$2\pi r_1 - 2\pi r_2 = 100$$

różnica ich średnic ($2r_1 - 2r_2$) wynosi więc $100/\pi \approx 31,83$ cm.

Trzecia zagadka jest jeszcze bardziej podchwytliwa i ilustruje trudność, jaką wielu ludzi ma z odróżnieniem „średniej” od „prawdopodobieństwa”. Które z podanych zdarzeń jest bardziej prawdopodobne: wyrzucenie jednej szóstki podczas sześciu rzutów kostką, czy wyrzucenie dwóch szóstek podczas dwunastu rzutów kostką? Pytanie możemy rozszerzyć o „wyrzucenie trzech szóstek podczas osiemnastu rzutów kostką” itd. Możemy też zadać podobne pytanie: które z podanych zdarzeń jest bardziej prawdopodobne – wyrzucenie *przynajmniej* jednej szóstki podczas sześciu rzutów kostką, czy wyrzucenie *przynajmniej* dwóch szóstek podczas dwunastu rzutów kostką?

Z reguły intuicja nas tu zawodzi. Wydaje się, że nie ma dużej różnicy między tymi dwoma scenariuszami, skoro oczekiwany wynik to odpowiednio jedna szóstka i dwie szóstki, a jednak prawdopodobieństwo tych dwóch zdarzeń jest w rzeczywistości bardzo różne!

W pierwszym przypadku prawdopodobieństwo wyrzucenia dokładnie jednej szóstki podczas sześciu rzutów wynosi

$$\binom{6}{1} \cdot \frac{1}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6} \cdot \frac{5}{6}$$

ponieważ

- $\binom{6}{1}$ jest liczbą kombinacji sześciu rzutów, które dały dokładnie jedną szóstkę; może ona wypaść podczas pierwszego rzutu, drugiego rzutu itd.;
- $1/6$ jest prawdopodobieństwem wyrzucenia szóstki w dowolnym rzucie;
- $5/6$ jest prawdopodobieństwem *niewyrzucenia* szóstki w konkretnym rzucie; szóstka musi zostać wyrzucona w którymś z pozostałych pięciu rzutów.

Tak więc prawdopodobieństwo wyrzucenia dokładnie jednej szóstki podczas sześciu rzutów wynosi 0,4019.

Z kolei prawdopodobieństwo wyrzucenia dokładnie dwóch szóstek podczas dwunastu rzutów wynosi

$$\binom{12}{2} \cdot \left(\frac{1}{6}\right)^2 \cdot \left(\frac{5}{6}\right)^{10} = 0,2961$$

Jak widzimy, różnica jest spora! Nawiąsem mówiąc, prawdopodobieństwo wyrzucenia dokładnie k szóstek podczas n rzutów wynosi

$$\binom{n}{k} \cdot \left(\frac{1}{6}\right)^k \cdot \left(\frac{5}{6}\right)^{n-k}$$

Przed Tobą ostatnia szansa! Mamy trzy identyczne pudełka. Jedno z nich zawiera cenny przedmiot (np. klasyczną corvette kabriolet z 1953 roku z czerwoną tapicerką i mnóstwem bajerów); w dwóch pozostałych pudełkach znajdują się rzeczy o znacznie mniejszej wartości (np. ołówek i kilka cukierków). Nie masz pojęcia, które pudełko ma jaką zawartość.

Zostajesz poproszony o wskazanie jednego z pudełek i robisz to. W tym momencie osoba prowadząca widowisko i doskonale zorientowana, które pudełko zawiera samochód, otwiera jedno z pozostałych pudełek. Z ulgą przekonujesz się, że nie zawiera samochodu. Teraz masz dokonać ostatecznego wyboru: możesz pozostać przy wybranym na wstępie pudełku lub wybrać pozostałe zamknięte pudełko. To już finał, zawartość wybranego pudełka będzie więc Twoja. Co powinieneś zrobić? Pozostać przy pierwotnym wyborze, czy zmienić go?

Na pierwszy rzut oka wydaje się, że to wszystko jedno. W końcu samochód może być w każdym z dwóch zamkniętych pudełek, jaki więc sens miałaby zmiana pudełka wybranego na początku? Proste rozumowanie powinno jednak przekonać Cię, że zmiana decyzji podwoi Twoje szanse na wygranie samochodu! Gdy wybierałeś pudełko po raz pierwszy, prawdopodobieństwo, że zawiera samochód wynosiło dokładnie $1/3$. Tak więc prawdopodobieństwo, że samochód był w którymś z pozostałych pudełek wynosiło $2/3$. Gdy jedno z nich zostało otwarте, szansa, że samochód jest w tym nadal nieotwartym, wynosi $2/3$. Innymi słowy, jeśli pozostaniesz przy swoim pierwotnym wyborze, Twoja szansa na wygranie samochodu wynosi $1/3$, jeśli jednak zdecydujesz się na pozostałe zamknięte pudełko, Twoja szansa na wygraną wzrośnie do $2/3$. Zadziwiające? Możesz to łatwo sprawdzić, biorąc trzy karty do gry: króla i dwie słabsze karty, tasując je i kładąc zakryte na stole. Wybierz kartę, a następnie powtóż procedurę zastosowaną do pudełek. Wykonaj dwadzieścia prób, w których pozostaniesz przy swoim pierwotnym wyborze, i dwadzieścia, w trakcie których zmienisz wybór. Do eksperymentu potrzebujesz przyjaciela, który będzie znał położenie wygrywającej karty (króla) i odsłaniał pozostałą kartę. Najlepiej, żeby był to posiadacz klasycznej corvetty z 1953 roku!

6. Podejście ewolucyjne

Wszystko, co jest naturalne, godne jest szacunku.

Marek Tulliusz Ciceron: *De Senectute*¹

W poprzednich trzech rozdziałach omówiliśmy wiele różnych klasycznych metod rozwiązywania problemów, w tym programowanie dynamiczne, metodę podziału i ograniczeń oraz algorytmy lokalnego przeszukiwania, a także niektóre nowoczesne metody, takie jak symulowane wyżarzanie czy poszukiwanie z tabu. Niektóre z tych metod były postrzegane jako deterministyczne. W metodach tych w zasadzie „przekręca się wajchę” i pojawia się odpowiedź. Pewne z metod dla zadanej przestrzeni przeszukiwania i funkcji oceny zawsze dają to samo rozwiązanie (np. programowanie dynamiczne), inne zaś mogą dawać różne rozwiązania w zależności od początkowej konfiguracji lub punktu początkowego (np. algorytm zachłanny lub metoda wspinania się). Jeszcze inne metody miały charakter probabilistyczny i wprowadzały losowe odmiany do poszukiwania optymalnego rozwiązania. Metody te (np. symulowane wyżarzanie) mogą dać różne wyniki nawet wówczas, gdy są uruchamiane dla tej samej konfiguracji początkowej. Nie należy się spodziewać, żeby jakiekolwiek dwa uruchomienia takich algorytmów przebiegały tak samo. Każda próba przypomina odcisk ludzkiego palca i chociaż są duże podobieństwa między odciskami, żaden z dwóch odcisków nie jest taki sam.

W przypadku wszystkich tych metod możemy zaobserwować, że każda z nich jest oparta na jednym rozwiązaniu, które jest podstawą przyszzej eksploracji w kolejnych iteracjach. Metody te albo przetwarzają pełne rozwiązania, albo budują je z mniejszych cegiełek. Algorytmy zachłanne, na przykład, kolejno budują rozwiązania przez maksymalizację lokalnie dostępnych ulepszeń. W programowaniu dynamicznym rozwiązuje się wiele mniejszych podproblemów, zanim dojdzie się do końcowego pełnego rozwiązania. Metody podzia-

¹Oryginał łaciński: *Omnia quae secundum naturam sunt, aestimatione digna sunt.* (przyp. tłum.).

łu i ograniczeń organizują przestrzeń przeszukiwania w wiele podprzestrzeni, a następnie przeszukują i eliminują niektóre z nich w sposób systematyczny. Dla kontrastu, metody lokalnego przeszukiwania, symulowanego wyżarzania i poszukiwania z tabu przetwarzają pełne rozwiązania, w wyniku czego otrzymujemy potencjalną odpowiedź (choć prawdopodobnie nieoptymalną) przy zatrzymaniu implementującego je algorytmu po wykonaniu pewnej liczby jego kroków. Mają one zawsze zapamiętane *jedno „najlepsze bieżące” rozwiązanie*. Rozwiązanie to próbują one dalej polepszać w następnych krokach. Mimo tych różnic każda z tych metod działa na jednym rozwiążaniu lub buduje jedno takie rozwiązanie.

Ogólnie mówiąc, podejście polegające na utrzymywaniu najlepszego znalezionej dotychczas rozwiązania i na próbowaniu ulepszenia go jest intuicyjnie poprawne i bardzo proste, a przy tym często dość efektywne. Możemy tu zastosować: 1) reguły deterministyczne, na przykład, w metodzie wspinania korzysta się z reguły: jeśli jest sprawdzany sąsiad lepszy, to przechodzimy do niego i od niego kontynuujemy poszukiwanie; w przeciwnym wypadku kontynuujemy poszukiwanie w bieżącym otoczeniu; 2) reguły probabilistyczne, na przykład w symulowanym wyżarzaniu korzysta się z reguły: jeśli sprawdzany sąsiad jest lepszy, to przyjmujemy jego miejsce jako bieżącą pozycję; w przeciwnym wypadku albo probabilistycznie przyjmujemy tę nową, słabszą pozycję, albo kontynuujemy poszukiwanie w bieżącym otoczeniu; lub nawet 3) reguły korzystające z historii poszukiwania aż do bieżącej pozycji włącznie, na przykład w poszukiwaniu z tabu korzysta się z reguły: weźmy najlepszego dostępnego sąsiada, który nie musi być lepszy od bieżącego rozwiązania, ale który nie jest wymieniony w pamięci jako ruch wykluczony bądź „tabu”.

Przyjrzyjmy się teraz rewolucyjnemu pomysłowi, który nie pojawił się w żadnej dotychczas prezentowanej metodzie. Porzućmy pomysł przetwarzania tylko jednego rozwiązania. Co by się stało, gdybyśmy zamiast tego utrzymywali wiele rozwiązań jednocześnie? To znaczy, co by się stało, gdyby nasz algorytm poszukiwania pracował na *populacji rozwiązań*?

W pierwszej chwili może się wydawać, że pomysł ten nie daje nam nic nowego. Możemy oczywiście przetwarzać wiele rozwiązań równolegle, jeśli mamy do dyspozycji komputery równoległe! Gdyby tak było, moglibyśmy zaimplementować metodę przeszukiwania równoległego, na przykład równoległego symulowanego wyżarzania lub równoległego poszukiwania z tabu, w którym każdy procesor zajmowałby się jednym rozwiązaniem i każdy z procesorów wykonywałby równolegle ten sam algorytm. Może się wydawać, że wszystko, co możemy tutaj uzyskać, to zwiększenie szybkości przetwarzania – zamiast uruchamiać algorytm k razy, żeby zwiększyć prawdopodobieństwo napotkania globalnego optimum, komputer z k procesorami mógłby to samo zadanie wykonać za jednym razem i zużyć na to mniej rzeczywistego czasu.

Istnieje jednak tutaj dodatkowy element, który decyduje o tym, że algorytmy korzystające z populacji istotnie różnią się od innych metod rozwiązywania problemów – koncepcja rywalizacji między rozwiązaniami w populacji. Zasymulujmy zatem proces ewolucyjny z rywalizacją oraz doborem naturalnym (selekcją) i niech kandydaci na rozwiązania w naszej populacji walczą o miejsce

w przyszłych pokoleniach! Co więcej, możemy wykorzystać losowe mutacje, żeby poszukiwać nowych rozwiązań, naśladując w ten sposób procesy naturalnej ewolucji.

Jeśli jeszcze nie słyszałeś o tej analogii, to zróbmy na chwilę dygresję i opiszmy pewien, trochę zaskakujący, przykład z królikami i lisami. W każdej populacji królików¹ niektóre z nich są szybsze i sprytniejsze niż inne. Dla tych szybszych i sprytniejszych prawdopodobieństwo zjedzenia przez lisy jest mniejsze, a zatem więcej ich przeżywa, żeby móc robić to, co króliki robią najlepiej – rozmnażać się! Populacja przeżywających królików zaczyna się rozmnażać. W wyniku rozmnażania materiał genetyczny królików ulega wymieszaniu, a zachowania sprytniejszych i szybszych królików są dziedziczone. Niektóre powolne króliki łączą się z szybkimi, niektóre szybkie z szybkimi, niektóre sprytne z niezbyt sprytnymi itd. Do tego natura dorzuca raz na jakiś czas „dziką sztukę”, gdyż wszystkie geny królicze mogą ulec z pewnym prawdopodobieństwem mutacji. Powstające w ten sposób królicze dzieci nie są zwykłymi kopiami ich rodziców, ale są ich losowymi kombinacjami. Wiele z tych potomków wyrośnie w taki sposób, że będą wykazywały zachowania, które umożliwiają im lepsze konkurowanie z lisami i innymi królikami! W ten sposób po wielu pokoleniach możemy się spodziewać, że króliki będą sprytniejsze i szybsze niż te, które przeżyły w początkowych pokoleniach. Nie należy też zapomnieć, że jednocześnie ewolucji podlegają i lisy, co wywiera nacisk na króliki. Takie współgranie między tymi populacjami popływa je ku ich fizycznym granicom. Lisy muszą być coraz lepsze w znajdowaniu posiłków, króliki zaś w unikaniu szansy na stanie się czymś obiadem!

Opowiedzmy jeszcze raz tę historyjkę, ale w kategoriach przestrzeni przeszukiwania, funkcji oceny i potencjalnych rozwiązań. Założymy, że zaczynamy od populacji początkowych rozwiązań, być może wygenerowanej przez losowe próbkowanie przestrzeni przeszukiwania *S*. Czasami, żeby podkreślić związek z genetyką, rozwiązania te nazywamy *chromosomami*, nie ma jednak konieczności używania terminów biologicznych². Algorytmy, których będziemy poszukiwać, to tylko algorytmy – procedury matematyczne oparte na analogiach do naturalnych procesów ewolucyjnych i nie ma potrzeby, żeby nasze tak zwane chromosomy zachowywały się w jakikolwiek sposób zbliżony do naturalnych chromosomów. Możemy po prostu opisywać kandydatów na rozwiązania jako wektory lub jakiekolwiek inne struktury danych, które wybierzemy.

Funkcji oceny możemy użyć do określenia relatywnych zalet każdego z naszych początkowych rozwiązań. Nie zawsze jest to proste, gdyż należy tutaj w sensowny sposób połączyć wiele elementów. Na przykład: który królik jest lepszy – szybki i głupi czy wolny, ale sprytny? Przez dobór naturalny (selekcję) podejmuje się odpowiednią decyzję, wykorzystując metodę kontroli jakości,

¹Jeden królik reprezentuje rozwiązanie problemu. Jest to jeden punkt w przestrzeni poszukiwań.

²W rzeczywistości używanie terminologii biologicznej może dawać złudzenie, że mamy tutaj do czynienia z bardziej bezpośrednim związkiem z ewolucją naturalną, niż można to w danym przypadku uzasadnić.

która możemy nazwać „wypróbowywaniem przez lisy”. Stwierdzenie, który królik jest lepszy w spełnianiu wymagań narzuconych przez otoczenie, polega na sprawdzaniu, które z nich mogą przeżyć spotkania z lisami lub ich uniknąć. Natura nie ma funkcji matematycznej określającej jakość zachowań osobników; my nierzadko musimy taką funkcję określić. Wybrana funkcja oceny musi rozróżniać dwa różne rozwiązania, tzn. musi wycenić jedne rozwiązania lepiej niż inne. Rozwiązania, które są lepsze zgodnie z funkcją oceny, mają większe szanse na stanie się rodzicami następnego pokolenia potomstwa.

Jak powinniśmy generować to potomstwo? W istocie widzieliśmy już podstawową koncepcję w większości pozostałych algorytmów – nowe rozwiązania są generowane probabilistycznie w otoczeniu starych rozwiązań. Gdy stosujemy algorytmy ewolucyjne, nie musimy zdawać się wyłącznie na otoczenia poszczególnych rozwiązań. Możemy także przeglądać otoczenia par rozwiązań. Oznacza to, że możemy użyć więcej niż jednego rozwiązania-rodzica do wygenerowania nowego kandydata na rozwiązanie. Jednym ze sposobów osiągnięcia tego jest wybranie części dwóch rodziców i połączenie ich, żeby utworzyć potomka. Moglibyśmy na przykład wziąć pierwszą połowę od jednego rodzica, a drugą połowę od innego. Pierwsza połowa pierwszego może być postrzegana jako „pomysł” i podobnie z drugą połową drugiego. Czasami taka rekombinacja może być bardzo użyteczna – łączymy biszkopty z lodami i otrzymujemy smakołyk. Czasami nie działa to tak dobrze – łączymy pierwszą połowę *Hamleta* Szekspira z drugą połową *Króla Lira* i w dalszym ciągu mamy Szekspira, ale wynik zupełnie nie ma sensu. Inny sposób stosowania dwóch rozwiązań do wygenerowania nowej możliwości pojawia się przy rozwiązywaniu ciągłych problemów optymalizacyjnych. W tego typu problemach możemy mieszać parametry obu rodziców, w istocie obliczając średnią ważoną współrzędną po współrzędnej. Czasem może to być użyteczne, a czasem może to być stratą czasu. Odpowiedniość każdego operatora poszukiwania zależy od rozwiązywanego problemu.

W związku z tym, że nasze algorytmy ewolucyjne funkcjonują w komputerach, nie musimy zastanawiać się nad rzeczywistymi ograniczeniami biologicznymi. W znaczącej większości organizmów płciowych (z wyjątkiem bakterii) łączenie się zachodzi między parami osobników; nasze poszukiwanie ewolucyjne może działać na zasadzie „kojarzenia” lub „mieszania” więcej niż dwojga rodziców, być może z uwzględnieniem *wszystkich* osobników w populacjach wszystkich generacji, żeby określić prawdopodobieństwo selekcji każdego nowego kandydata na rozwiązanie.

W każdym pokoleniu osobniki konkurują – albo ze sobą, albo ze swoimi rodzicami – o przejście do następnej iteracji. Bardzo często po przejściu ciągu pokoleń obserwujemy, że są dziedziczone lepsze cechy testowanych rozwiązań i populacja jest zbieżna do otoczenia rozwiązania bliskiego optimum.

Bez wątpienia jest to wspaniała koncepcja! Dlaczego mamy pracować nad rozwiązywaniem trudnych matematycznych formuł lub opracowywać skomplikowane programy komputerowe, opisujące przybliżone modele problemu, jeśli możemy zasymulować istotny i podstawowy proces ewolucji oraz odkryć rozwiązania bliskie optimum za pomocą znacznie bardziej wiarygodnych modeli, zwłaszcza gdy trudna część procesu poszukiwania jest „za darmo”. Zwykle coś

za darmo to rzecz zbyt piękna, żeby była prawdziwa, celowe jest zatem zadanie tutaj kilku pytań testowych. Jakiego rodzaju wysiłek jest wymagany od użytkownika algorytmu ewolucyjnego? Innymi słowy, ile pracy wymaga zaimplementowanie tych pomysłów w ramach algorytmu? Czy jest to efektywne kosztowo? I czy *naprawdę* możemy rozwiązywać rzeczywiste problemy za pomocą takiego ewolucyjnego podejścia? Odpowiemy na te pytania w pozostałą części książki.

W następnych podrozdziałach przejrzymy jeszcze raz trzy wprowadzone w rozdziale 1 problemy: problem spełnialności formuł boolowskich (SAT), problem komiwojażera (TSP) oraz problem programowania nieliniowego (NLP). Przedstawimy też pewne możliwe konstrukcje algorytmów ewolucyjnych służących do rozwiązania tych zadań.

6.1. Podejście ewolucyjne do SAT

Przypuśćmy, że chcielibyśmy zastosować algorytm ewolucyjny do generowania rozwiązań dla problemu SAT ze 100 zmiennymi. Powiedzmy, że problem ten jest w zasadzie podobny do omawianego w podrozdziale 1.1, gdzie obliczaliśmy złożone wyrażenie boolowskie postaci

$$F(\mathbf{x}) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97})$$

i że musimy znaleźć wektor przypisań wartości logicznych dla wszystkich 100 zmiennych x_i , dla $i = 1, \dots, 100$, o takiej własności, że $F(\mathbf{x}) = 1$ (TRUE). W związku z tym, że mamy do czynienia ze zmiennymi, które mogą przybierać tylko dwa stany, TRUE bądź FALSE, naturalną reprezentacją rozwiązań jest binarny ciąg (wektor) o długości 100. Każdy element ciągu oznacza przypisanie wartości logicznej zmiennej odpowiadającej umieszczeniu tego elementu w ciągu. Na przykład, ciąg (1010 … 10) przypisywałby TRUE wszystkim nieparzystym zmiennym x_1, x_3, \dots, x_{99} , a FALSE zmiennym parzystym x_2, x_4, \dots, x_{100} , przy zastosowaniu kodowania TRUE jako 1 oraz FALSE jako 0. Każdy ciąg binarny 100 zmiennych może być możliwym rozwiązaniem problemu. Możemy użyć $F(\mathbf{x})$ do sprawdzenia, czy taki wektor spełnia wszystkie warunki konieczne, aby sprawić, że $F(\mathbf{x}) = 1$.

Potrzebujemy początkowej populacji, żeby zacząć całe obliczenie, a co za tym idzie, naszym następnym punktem do zastanowienia się jest rozmiar populacji. Ile osobników chcielibyśmy mieć? Powiedzmy, że chcemy wykorzystywać 30 wektorów-rodziców. Jednym ze sposobów ich wybrania jest losowanie symetryczną monetą wartości każdego bitu w tych wektorach. Możemy mieć nadzieję, że da nam to zróżnicowany zestaw rodziców. Zwykle taka różnorodność jest pożądana, choć czasami możemy chcieć zainicjować wszystkich dostępnych rodziców tym samym, najlepszym dotychczas znanym rozwiązaniem i zacząć przetwarzanie od tego punktu. Na potrzeby naszego przykładu przyjmijmy, że do określenia populacji początkowej skorzystaliśmy z tej procedury losowej. Warto zwrócić uwagę, że natychmiast otrzymujemy, iż przy każdej próbie

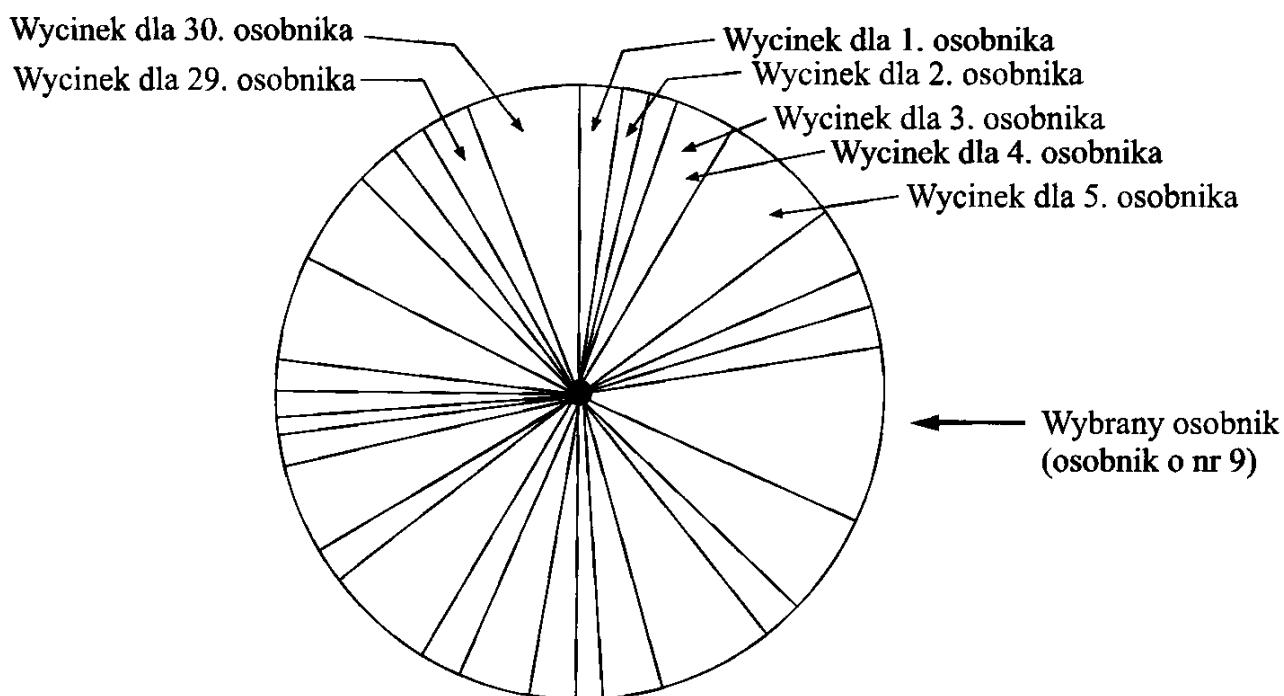
uruchomienia algorytmu ewolucyjnego będziemy mieli potencjalnie inną populację początkową w zależności od tego, jaką próbkę losową uda nam się uzyskać.

Przypuśćmy, że mamy początkową losową populację. Następną czynnością, jaką mamy wykonać, jest ocenienie każdego wektora, żeby sprawdzić, czy odkryliśmy już optymalne rozwiązanie, oraz żeby określić jakość posiadanych przez nas rozwiązań. Oznacza to, że potrzebujemy funkcji oceny. Tu pojawia się trudność. Proste podejście polega na użyciu $F(\mathbf{x})$ do sprawdzenia, które z naszych początkowych wektorów są optymalnymi rozwiązaniami. Przypuśćmy jednak, że żaden z początkowych wektorów nie jest doskonały, tzn. każdy wybrany losowo wektor początkowy \mathbf{x} daje $F(\mathbf{x}) = 0$. Co teraz? Nie mamy żadnej użytecznej informacji na temat tego, jak blisko konkretnego wektora znajduje się pełne rozwiązanie. Gdybyśmy zastosowali $F(\mathbf{x})$ jako funkcję oceny, a rozwiązaniem doskonałym było $\langle 11 \dots 11 \rangle$, wówczas dla rozwiązań $\langle 00 \dots 00 \rangle$ i $\langle 11 \dots 10 \rangle$ mielibyśmy sytuację, że oba te rozwiązania otrzymałyby ocenę 0, podczas gdy pierwszy wektor ma 100 nieprawidłowo ustawionych zmiennych, a odległość drugiego od rozwiązania idealnego wynosi jeden! W oczywisty sposób $F(\mathbf{x})$ nie daje informacji, które są potrzebne przy poszukiwaniu prowadzącym do uzyskania lepszych ciągów.

Mamy tutaj do czynienia z sytuacją, w której cel nie podpowiada, jak powinna wyglądać odpowiednia funkcja oceny. Cel polegający na sprawieniu, że złożone wyrażenie boolowskie $F(\mathbf{x}) = 1$, jest niewystarczający. Co zatem jest wystarczające? W pierwszej chwili możemy chcieć porównywać liczbę bitów różniących kandydata na rozwiązanie od jakiegoś idealnego rozwiązania i korzystać z tego jak z funkcji opisującej błąd. Im więcej bitów jest nieprawidłowo podanych, tym większy jest błąd, możemy zatem próbować minimalizować jego wielkość. Problem z tym podejściem polega na tym, że założono, iż mamy już jakieś rozwiązanie, które umożliwia nam wykonywanie takiego porównania; gdybyśmy jednak znali doskonały wektor, nie mielibyśmy szukać rozwiązania! Zamiast tego moglibyśmy skupić się na częściach wyrażenia boolowskiego $F(\mathbf{x})$, które są oddzielone symbolem (\wedge), na przykład $(x_{17} \vee \bar{x}_{37} \vee x_{73})$ i zliczać, ile z tych oddzielnych podwyrażeń ma wartość TRUE. Moglibyśmy następnie wykorzystać liczbę tych podwyrażeń dających TRUE jako funkcję oceny, mając na celu maksymalizację liczby podwyrażeń dających TRUE, ponieważ jeśli dla $F(\mathbf{x})$ danego jak powyżej wszystkie podwyrażenia dawały TRUE, to mielibyśmy, że $F(\mathbf{x}) = 1$ – dokładnie to, czego potrzebujemy. Czy wiemy, że taka funkcja oceny doprowadzi nas do idealnego rozwiązania? Niestety, nie. Wszystko, co możemy powiedzieć przed przystąpieniem do eksperymentów, to tyle, że wydaje się to być rozsądny kandydat na funkcję oceny.

Załóżmy, że stosujemy tę funkcję do określenia jakości naszych 30 początkowych wektorów i założmy dalej, że $F(\mathbf{x})$ składa się z 20 podwyrażeń, a co za tym idzie najlepszy wynik, jaki możemy uzyskać, wynosi 20, najgorszy zaś 0. Przypuśćmy, że najlepszy wektor w populacji poddany tej ocenie dał wartość 10, najgorszy zaś 1 oraz że średnia wyników wypadła gdzieś między 4 a 5. Następny problem, o którym możemy chcieć pomyśleć, to sposób, w jaki należy wygenerować osobniki, które mają przeżyć jako rodzice do następnego pokolenia. To znaczy, musimy zastosować funkcję selekcji.

Jedna z metod osiągnięcia tego celu polega na przypisaniu każdemu osobnikowi prawdopodobieństwa wyboru i to proporcjonalnie do ich wzajemnej wartości. To znaczy, dla rozwiązania o wyniku 10 prawdopodobieństwo wybrania powinno być 10 razy większe niż dla rozwiązania o wyniku 1. Tę selekcję proporcjonalną nazywamy czasami też selekcją według koła ruletki (selekcją ruletkową), gdyż często, realizując tę procedurę, myślimy o obracaniu koła ruletki po jednym razie dla każdego dostępnego miejsca w następnej populacji, przy czym każde rozwiązanie ma przypisany wycinek koła proporcjonalny do wyniku jego oceny (rys. 6.1). Jeślibyśmy chcieli utrzymać populację 30 osobników, to na kole umieścilibyśmy 30 wycinków – po jednym dla każdego osobnika – i obracalibyśmy kołem 30 razy – po jednym razie dla każdego dostępnego w następnej populacji miejsca. Zauważmy, że możemy przy tym uzyskiwać wiele kopii pewnych rozwiązań oraz że pewne rozwiązania w ogóle nie zostaną wybrane. W rzeczywistości nawet najlepsze rozwiązanie w bieżącej populacji może nie zostać wybrane tylko ze względu na losowy charakter tej procedury. Koło ruletki jedynie przechyla szanse wyboru w kierunku lepszych rozwiązań; jak w wypadku koła ruletki w kasynie – nie ma tutaj jednak żadnych gwarancji!



Rysunek 6.1. Koło ruletki z 30 wycinkami. Rozmiar każdego z nich odpowiada jakości odpowiedniego osobnika. Na tym rysunku najlepszą wartość ma osobnik o numerze 9

Przypuśćmy, że pokręciliśmy kołem 30 razy i wybraliśmy naszych 30 kandydatów do następnego pokolenia. Prawdopodobnie najlepsze rozwiązanie jest teraz reprezentowane 3 razy, najgorsze zaś rozwiązanie w ogóle nie zostało wybrane. Zostało usunięte z dalszych rozważań. Niezależnie od wynikowego rozkładu naszych rozwiązań, nie wygenerowaliśmy niczego nowego, jedynie zmodyfikowaliśmy początkowy rozkład naszych poprzednich rozwiązań. Teraz musimy skorzystać z losowego różnicowania, żeby znaleźć nowe rozwiązania.

Jednym ze sposobów zrobienia tego mogłoby być losowe wprowadzanie możliwości, by niektóre osobniki spośród 30 kandydatów uległy rekombinacji. Dla każdego z nich moglibyśmy mieć prawdopodobieństwo, powiedzmy 0,9, że do następnego pokolenia przejdzie zmieniony, to znaczy zostanie on podzielony i połączony z częścią jednego z pozostałych 29 rozwiązań. Taki działający na parze rodziców operator byłby procedurą cięcia i łączenia, która zapewniałaby, że kolejne składowe ciągów-kandydatów, które pochodzą z lepszych rozwiązań, mają szansę na pozostawanie razem. Na ile ta metoda jest efektywna, zależy częściowo od stopnia niezależności składowych. Jeśli składowe wektorów-kandydatów nie są w świetle funkcji oceny niezależne, to operator rekombinacji może, ujmując to metaforycznie, skończyć na rozcinaniu i składaniu *Hamleta z Królem Lirem*. Inna potencjalna trudność polega na tym, że operator rekombinacji może generować nowe rozwiązania, jeśli wśród osobników bieżącej populacji jest wystarczająco duża różnorodność. Jeśli nie będziemy mieli szczęścia i koło ruletki sprawi, że powstała populacja będzie jednorodna, to nastąpi zastój, nawet jeśli nasze bieżące rozwiązanie nie jest optimum lokalnym! Jako swego rodzaju zabezpieczenie możemy dodać jeszcze jeden operator, który w wybranym ciągu przełącza losowo jeden bit, i stosować go z małym prawdopodobieństwem. W ten sposób zapewniamy sobie potencjalną możliwość wprowadzania nowych odmian.

Tych pięć kroków polegających na ustaleniu: 1) reprezentacji, 2) populacji początkowej, 3) funkcji oceny, 4) procedury selekcji (doboru) oraz 5) losowych operatorów różnicowania określa istotne aspekty jednego z możliwych ewolucyjnych podejść do problemu SAT. Powtarzając na kolejnych pokoleniach ocenę, selekcję i różnicowanie, mamy nadzieję, że będziemy odkrywać wektory, które sprawiają, że coraz więcej podwyrażeń $F(\mathbf{x})$ jest spełnionych i że w końcu zostanie spełniony warunek $F(\mathbf{x}) = 1$.

6.2. Podejście ewolucyjne do TSP

Przypuśćmy, że chcemy zastosować podejście ewolucyjne do generowania rozwiązań dla TSP ze 100 miastami. Uprościmy sytuację, zakładając, że możemy podróżować z każdego miasta do każdego innego oraz że odległości między miastami są ustalone, celem zaś jest zminimalizowanie odległości pokonywanej przez komiwojażera na trasie prowadzącej przez każde miasto dokładnie raz i wracającej do jego bazy wypadowej.

Naszym pierwszym punktem analizy jest znowu reprezentacja wszystkich możliwych rozwiązań. Mamy wiele możliwości; naturalne wyjście, które przychodzi natychmiast na myśl, polega na użyciu uporządkowanej listy miast, które należy odwiedzić. Na przykład wektor

$$[17, 1, 43, 4, 6, 79, \dots, 100, 2]$$

mogłby oznaczać, że komiwojażer zaczyna podróż w mieście 17, jedzie do miasta 1, a następnie udaje się do miast 43, 4, 6, 79 i tak dalej przez wszystkie miasta aż do miasta 100, po którym następuje miasto 2. Komiwojażer musi

jeszcze dotrzeć do miasta 17, nie musimy jednak wprowadzać żadnej modyfikacji do naszej reprezentacji, żeby to obsłużyć. Gdy komiwojażer dojdzie do ostatniego miasta na liście, zakładamy, że ma jeszcze przejechać do jednego miasta, mianowicie do tego na początku listy. Łatwo możemy zauważyć, że każda permutacja tej listy generuje nowego kandydata na rozwiązanie, który spełnia ograniczenie polegające na konieczności odwiedzenia wszystkich miast dokładnie raz i powrotu do domu.

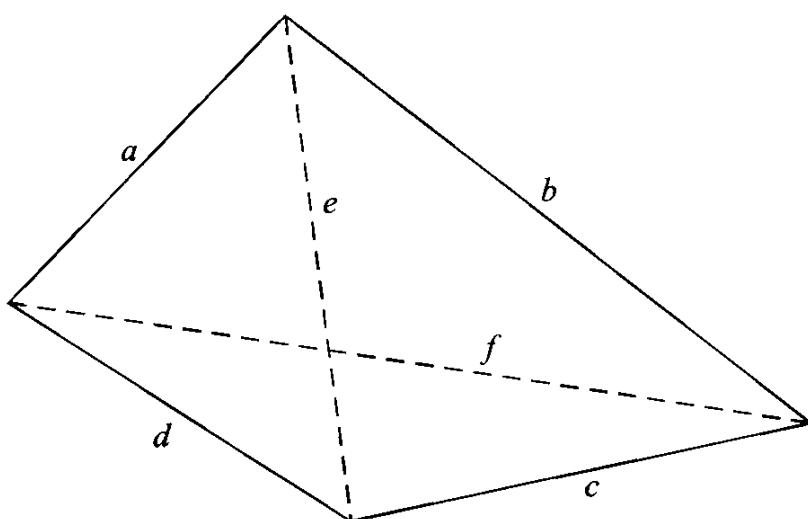
Na początku musimy zainicjować populację, ustalmy więc znowu, że mamy 30 tras-kandydatów. Możemy wygenerować 30 losowych permutacji liczb naturalnych z zakresu $[1 \dots 100]$, które będą stanowiły początkową populację. Gdybyśmy mieli jakieś informacje o dobrych rozwiązaniach, być może, pochodzące od jakiegoś komiwojażera, który już wcześniej korzystał z jakichś tras, moglibyśmy je również uwzględnić, ale na potrzeby tego przykładu przyjmijmy, że nie mamy żadnych takich wstępnych informacji.

Po wygenerowaniu 30 tras-kandydatów musimy ocenić ich jakość. W tym wypadku konieczność minimalizacji całkowitej długości trasy wskazuje, żeby zastosować funkcję oceny, która każdej trasie przypisuje wartość równoważną z jej długością. Im dłuższa jest trasa, tym gorsza jest ocena, a naszym celem jest znalezienie trasy, która minimalizuje funkcję oceny. Ta część metody ewolucyjnej wydaje się tutaj o wiele prostsza niż w wypadku SAT.

Założymy, że do oceny naszych 30 tras użyliśmy ich długości. Pozostało nam wybranie operatorów selekcji oraz losowego różnicowania populacji, ale nie musimy ich wykonywać w tej kolejności. Przypuśćmy, że inaczej niż w naszym przykładzie z SAT, zdecydowaliśmy się najpierw generować nowe rozwiązania, potem je oceniać, a następnie dokonywać selekcji.

Nasza pierwsza decyzja dotyczy zatem tego, jak generować nowe rozwiązania na podstawie istniejących rozwiązań-kandydatów. Zawsze należy korzystać z wiedzy na temat problemu, który rozwiązuje [309], i tutaj mamy taką sytuację, w której możemy to zrobić. Nasza funkcja oceny jest oparta na odległościach w przestrzeni euklidesowej, a zatem rozważmy następujące pytanie: Co jest dłuższe: suma długości przekątnych czworoboku czy suma długości każdej pary przeciwnych boków? Sytuację tę przedstawiono na rys. 6.2. W geometrii euklidesowej suma przekątnych (e i f) jest zawsze dłuższa niż suma przeciwnych boków (a i c lub b i d), a co za tym idzie, znaczy to, że trasy, które się krzyżują są dłuższe niż te, w których taka sytuacja nie zachodzi. Możemy zatem wprowadzić operator różnicowania, który będzie usuwał osobniki, w których trasa się krzyżała.

Rzeczywiście, przy naszej reprezentacji operator ten możemy bardzo łatwo zrealizować. Dla dowolnej uporządkowanej listy miast, jeśli wybierzemy na niej dwa miasta i odwrócić kolejność wszystkich miast między tymi dwoma punktami, to mamy możliwość naprawienia trasy, która przecina własną ścieżkę (krzyżuje się). Oczywiście, jeśli zastosujemy ten operator do rozwiązań, które nie przecinają swoich ścieżek, to możemy do rozwiązania wprowadzić opisany tutaj problem. Nie możemy jednak z góry przewidzieć, które rozwiązania będą się krzyżować, a które nie, a więc jedną z możliwości jest po prostu losowe wybranie dwóch miast na liście i wygenerowanie nowego rozwiązania. Jeśli takie



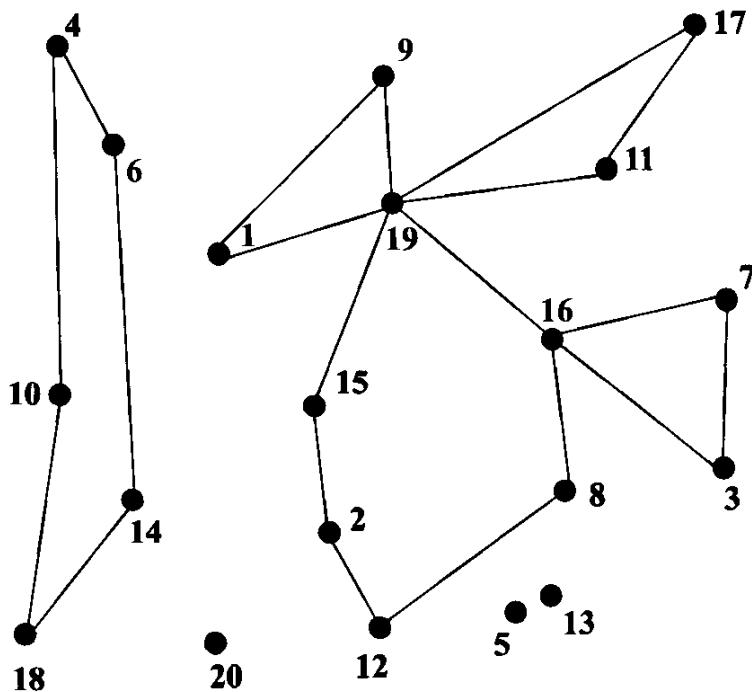
Rysunek 6.2. Czworobok. Zauważ, że $a + c < e + f$ oraz $b + d < e + f$

różnicowanie jest nieudane, to selekcja sprawi, że zostanie ono wyeliminowane z dalszych rozważań.

Podobnie jak w poprzednim przykładzie ilustrującym SAT, generując nowe rozwiązanie możemy myśleć o dwóch lub więcej rozwiązaniach początkowych; rozwiązanie takie wymaga jednak uważnego przemyślenia. Nie możemy po prostu wziąć pierwszej części jednego rozwiązania i połączyć jej z drugą częścią innego. Przy takim podejściu z dużym prawdopodobieństwem stworzymy rozwiązańe, które miałyby wiele wystąpień dla pewnych miast oraz żadnych wystąpień dla innych (rys. 6.3). Naruszałoby to ograniczenie, żeby odwiedzać każde miasto dokładnie raz. Moglibyśmy z kolei po prostu eliminować tego rodzaju rozwiązania, nadając im błędą wartość, nieskończoność, byłyby one jednak generowane tak często, że większość naszego czasu tracilibyśmy bezużytecznie, odkrywając niedopuszczalne rozwiązania i odrzucając je. Przy wybranej przez nas reprezentacji musielibyśmy być pomysłowi, żeby utworzyć operatory różnicowania korzystające z dwóch rodziców i generujące dopuszczalne rozwiązania.

Moglibyśmy, na przykład, wziąć dwóch rodziców i zacząć od pierwszego miejsca każdej trasy, wybierając losowo pierwsze miasto potomka jako pierwsze miasto albo z pierwszego, albo z drugiego rodzica. Moglibyśmy następnie przejść do drugiej pozycji w obu rodzicach i znowu wybrać losowo jednego z nich, po czym kontynuować do końca. Na każdym kroku musielibyśmy się upewnić, że nie kopujemy miasta, które już trafiło na listę w którymś z poprzednich kroków. W tym celu zamiast kopiować miasto z drugiego rodzica (lub w sytuacji gdy u obu rodziców miasta w danym miejscu już zostały umieszczone na liście), moglibyśmy po prostu wybierać nowe miasto losowo, tak żeby nie naruszyć ograniczenia związanego z jednokrotnym odwiedzaniem każdego miasta.

Przypuśćmy, że zamiast generowania potomstwa w liczbie równej liczbie rodziców zdecydowaliśmy się na tworzenie po trzech potomków na każdego rodzica. Możemy generować ich dowolną liczbę, uznaną za odpowiednią. Dla każdego z 30 rodziców trzy razy po kolei losowo wybieramy w reprezentacji rodzica dwa miasta, odwracamy listy miast między tymi punktami i wstawiamy uzyskane nowe rozwiązanie do populacji. W końcu otrzymujemy 90 potomnych tras.



Rysunek 6.3. Niedopuszczalne rozwiązanie TSP. Przedstawiona trasa jest nieprawidłowa, ponieważ nie spełnia ograniczenia mówiącego, że każde miasto na trasie musi być odwiedzone dokładnie jeden raz

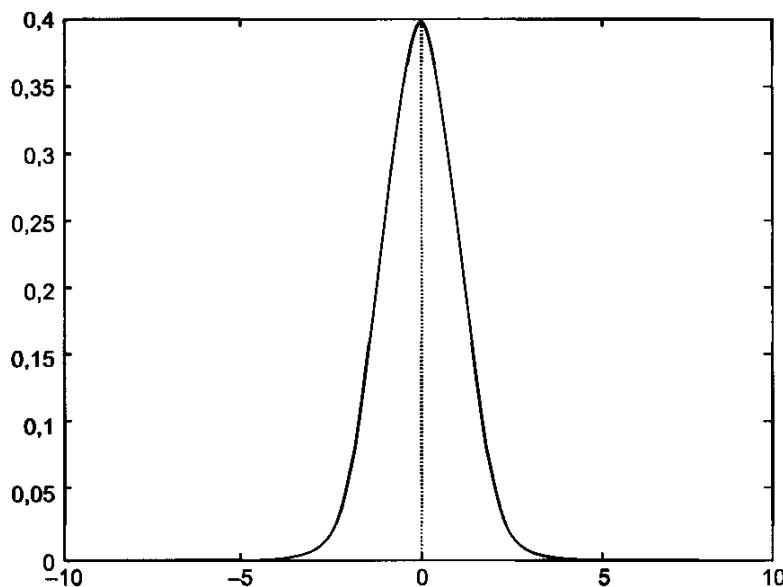
W tym momencie stajemy przed wyborem: możemy wprowadzić 90 potomków, aby konkurencieli z 30 rodzicami, i wybrać nowych rodziców spośród wszystkich $30 + 90 = 120$ rozwiązań lub też możemy odrzucić pierwotnych rodziców i skupić się na potomstwie. Chociaż odrzucanie rodziców wydaje się nierozważne, są powody podjęcia takiej decyzji; z jej wyjaśnieniem musimy poczekać do rozdziału 7. Na razie powiedzmy, że skupimy się na 90 potomkach i spośród nich zachowamy 30 najlepszych jako rodziców następnego pokolenia. Tego rodzaju selekcja jest opisywana parą (μ, λ) , przy czym: μ oznacza liczbę rodziców, λ – liczbę potomstwa, a selekcja jest wykorzystywana do utrzymania najlepszych μ rozwiązań przy rozważaniu samego potomstwa. Powyżej opisaliśmy procedurę $(30, 90)$.

Proces losowego różnicowania możemy zrealizować, stosując na każdym rodzicu procedurę odwracania kolejności (transpozycji) dwóch miast lub też zmodyfikowany operator działający na parze rodziców i generujący po trzech potomkach dla każdego z nich, a następnie stosując selekcję w celu wyeliminowania najgorszych 60 nowych osobników. Nasze oczekiwanie jest takie, że z czasem populacja będzie zmierzać do tras bliskich optymalnym.

6.3. Podejście ewolucyjne do NLP

Przypuśćmy, że chcemy rozwiązać NLP podane w rozdziale 1 (rys. 1.2) w przypadku 10 zmiennych. Znając wcześniejsze opisy możliwych podejść do SAT i TSP, opracowanie algorytmu dla NLP powinno być dość proste. W każdym razie pierwsza rzecz, jaką trzeba określić, to reprezentacja i tutaj najbardziej intuicyjnym rozwiązaniem jest wybranie wektora 10 liczb zmiennopozycyjnych,

z których każda odpowiada parametrowi problemu. Gdy zaczniemy myśleć o populacji początkowej, musimy jakoś zatroszczyć się o ograniczenia, ale nawet mimo tego możemy w sposób losowy wybrać ujednolicone wartości parametrów z przedziału $[0, 10]$, a następnie upewnić się, czy nie naruszyliśmy ograniczeń problemu. Gdybyśmy naruszyli, moglibyśmy odrzucić uzyskany wektor początkowy i wygenerować nowy. Postępując tak, możemy zapełnić całą populację początkową. Objętość części kostki $[0, 10]^{10}$ zawierającej elementy naruszające ograniczenia jest dość mała, takie podejście nie powinno więc powodować dużej utraty czasu obliczeniowego.



Rysunek 6.4. Rozkład normalny $f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-(x-m)^2/(2\sigma^2)}$ o odchyleniu standardowym $\sigma = 1$ i średniej $m = 0$

Następnie musimy wygenerować potomstwo. W przypadku wektorów liczb zmiennopozycyjnych często stosowanym operatorem różnicowania jest dodawanie do każdego parametru zmiennej losowej Gaussa¹. Przypomnijmy sobie, że funkcja gęstości zmiennej Gaussa ma kształt dzwonu (rys. 6.4). Zeby zdefiniować zmienną losową Gaussa, musimy znać jej średnią i wariancję. W naszym przykładzie przyjmiemy, że średnia wynosi 0, a wariancja 1 (jest to standar-dowa zmienna Gaussa). Później będziemy mogli pokazać, że możliwe są lepsze wybory, na razie jednak to powinno nam wystarczyć. Potomstwo generujemy tutaj, dodając różne zmienne losowe Gaussa do poszczególnych elementów rodzica, wykonując w ten sposób 10 wzajemnie niezależnych zmian. Po zakończeniu mamy potomka, który w średnim przypadku wygląda bardzo podobnie do rodzica, a prawdopodobieństwo, że będzie dla jakiegoś parametru radykalnie różny od rodzica, jest nieznaczne, gdyż większość gęstości zmiennej Gaussa jest skupiona w odległości odchylenia standardowego od średniej. Często taka własność jest bardzo pożądana, gdyż musimy mieć możliwość określenia związku między rodzicami a potomstwem. Jeśli ustalimy zbyt dużą wariancję, to związek ten może zostać zniszczony, a uzyskane w ten sposób poszukiwanie ma

¹ Nazywaną również zmienną *normalną*.

wtedy charakter zupełnie losowy. Takiej sytuacji należy unikać, gdyż nie bierze się wówczas pod uwagę tego, czego się nauczyliśmy w ciągu kolejnych pokoleń ewolucji. Różnicowanie Gaussa ma tę dodatkową zaletę, że za jego pomocą jest możliwe wygenerowanie dowolnego nowego rozwiązania, dzięki czemu proces ewolucyjny nigdy nie utknie na zawsze w rozwiązaniu nieoptymalnym. Po odpowiednio długim okresie oczekiwania wszystkie punkty przestrzeni zostaną przebadane.

Po wygenerowaniu całego potomstwa musimy je ocenić. Tutaj znowu, jak w wypadku TSP, cel wskazuje, jak powinna wyglądać funkcja oceny. Możemy jako funkcji oceny użyć bezpośrednio wyrażenia matematycznego, które mamy zmaksymalizować, przy czym musimy jeszcze uwzględnić ograniczenia. I chociaż wyrażenie opisujące cel jest zdefiniowane dla wszystkich liczb rzeczywistych we wszystkich wymiarach, obszar dopuszczalny jest małym skończonym podzbiorem całej przestrzeni liczb rzeczywistych. Jednym z możliwych rozwiązań w takiej sytuacji jest zastosowanie funkcji kary, gdzie wynik osobnika to wartość wygenerowana przez połączenie tej funkcji z wyrażeniem matematycznym opisującym cel. Cel ten jest dodatkowo tak modyfikowany za pomocą kary, żeby uwzględnić stopień naruszenia zadanych ograniczeń. Im dalej osobnik się znajduje w głąb obszaru niedopuszczalnego, tym większa jest kara. Odpowiednie wyskalowanie funkcji kary to kwestia oceny i doświadczenia; na razie możemy po prostu powiedzieć, że funkcja oceny to wyrażenie matematyczne celu minus odległość osobnika od granicy dopuszczalnego obszaru (jeśli osobnik jest w obszarze niedopuszczalnym). Będziemy więc dążyć do maksymalizacji tej sumy¹.

Jeśli chodzi o selekcję, to moglibyśmy zastosować rodzaje selekcji, które poznaliśmy przy SAT i TSP, możemy też wprowadzić trzeci jej rodzaj: selekcję rankingową (rangową) z wykorzystaniem turnieju. Przypuśćmy, że zaczeliśmy od 30 osobników i wygenerowaliśmy następnych 30 za pomocą różnicowania Gaussa (po jednym potomku dla każdego rodzica). Mamy w tym momencie zestaw 60 wektorów. Przypuśćmy dalej, że utrzymujemy populację, która na początku każdego pokolenia ma stale 30 osobników. Wybór tej trzydziestki możemy zorganizować w postaci turnieju, w którym, na przykład, losowo wybieramy pary rozwiązań i określamy w parach, które jest lepsze, i to rozwiązanie przechodzi do następnego pokolenia. Po 30 takich ujęciach w pary mamy nową populację, w której osobniki średnio lepsze mają większą szansę na przeżycie, nie mamy jednak pewności, że przetrwają do następnego pokolenia.

Po wielokrotnym powtarzaniu różnicowania, oceniania (z karami) oraz tego rodzaju selekcji turniejowej możemy się spodziewać, że w wyniku ewolucji powstanie osobnik, który spełnia ograniczenia i znajduje się blisko globalnego optimum. Istnieje wiele sposobów, które mogą zwiększyć szybkość, z jaką dobre rozwiązanie mogą być odkrywane. Polegają one na zmienianiu parametrów operatorów ewolucyjnych; zajmiemy się nimi bliżej w dalszych rozdziałach.

¹Z kolei dla problemu minimalizacji kara będzie raczej dodawana, a nie odejmowana.

6.4. Podsumowanie

Na rysunku 6.5 przedstawiono strukturę algorytmu ewolucyjnego.

```

procedure algorytm ewolucyjny
begin
     $t \leftarrow 0$ 
    inicjuj  $P(t)$ 
    oceń  $P(t)$ 
    while (not warunek zakończenia) do
        begin
             $t \leftarrow t + 1$ 
            wybierz  $P(t)$  spośród  $P(t - 1)$ 
            zmień  $P(t)$ 
            oceń  $P(t)$ 
        end
    end

```

Rysunek 6.5. Struktura algorytmu ewolucyjnego

Algorytm ewolucyjny w iteracji t dysponuje populacją osobników $P(t) = \{x_1^t, \dots, x_n^t\}$. Każdy osobnik reprezentuje potencjalne rozwiązanie danego problemu i w każdym algorytmie ewolucyjnym jest zaimplementowany jako struktura danych S . Każde rozwiązanie x_i^t jest oceniane, żeby określić miarę jego „przystosowania”. Nowa populacja w iteracji $t + 1$ jest utworzona przez wybranie (w kroku „wybierz”) bardziej przystosowanych osobników. Niektóre osobniki tej nowej populacji podlegają przekształceniom (w kroku „zmień”) za pomocą operatorów różnicowania, dzięki czemu powstają nowe rozwiązania. Działają tu unarne przekształcenia u_i , które dają w wyniku nowe osobniki przez wprowadzenie zmian do poszczególnych rozwiązań ($m_i : S \rightarrow S$), oraz przekształcenia wyższego poziomu, które dają w wyniku nowe osobniki przez łączenie części wielu (dwóch lub więcej) z nich ($c_j : S \times \dots \times S \rightarrow S$). Algorytm jest wykonywany do momentu, aż zostanie spełniony zdefiniowany zawsze warunek zatrzymania, który może albo sprowadzać się do wygenerowania określonej liczby rozwiązań-kandydatów, albo do przekroczenia określonego czasu.

Historia stosowania tego rodzaju algorytmów jest znana od wczesnych lat pięćdziesiątych XX wieku i była w istocie wymyślona koło dziesięciu razy niezależnie przez różnych naukowców [149]. Każda z tych procedur różniła się odrobinę od pozostałych. W latach sześćdziesiątych XX wieku niektórzy z naukowców nadali swoim algorytmom różne nazwy, na przykład: *algorytmy genetyczne* (ang. *genetic algorithms*), *strategie ewolucyjne* (ang. *evolution strategies*) lub *programowanie ewolucyjne* (ang. *evolutionary programming*); jednak nie każdy z nich nazwał swój wynalazek. Z czasem opracowano bardziej wyspecjalizowane metody, które były związane konkretnymi strukturami danych, takie jak *systemy klasyfikatorów* (ang. *classifier systems*) i *programowanie*

genetyczne (ang. *genetic programming*). W latach dziewięćdziesiątych XX wieku rozważano teoretyczne i empiryczne argumenty, świadczące o tym, że żadne z tych kanonicznych podejść nie daje niczego, czego nie dałoby się uzyskać za pomocą innego podejścia. Co więcej, ponieważ zainteresowanie algorytmami genetycznymi znaczaco wzrosło, zaczęto zapożyczać pomysły, wymieniać się nimi i wprowadzać we wszystkich wspomnianych podejściach związane z tym modyfikacje. W wyniku tego nie ma już żadnych naukowych podstaw do rozgraniczania tych różnych podejść ewolucyjnych. W świetle współczesnej wiedzy w dalszej części tekstu będziemy stosować termin *algorytm ewolucyjny* na określenie dowolnego z algorytmów, który wykorzystuje oparte na populacji losowe różnicowanie oraz selekcję. Zachęcamy Cię do myślenia w podobnie szeroki sposób.

Podsumujmy niektóre koncepcje, które omawialiśmy we wcześniejszych podrozdziałach tego rozdziału. Przy okazji wskażemy kilka dodatkowych możliwości. Na początku chcielibyśmy podkreślić, że algorytmy ewolucyjne mogą działać na dowolnej reprezentacji, która wydaje się odpowiednia dla zadania, jakie mamy do rozwiązania, niezależnie od tego, czy jest to ciąg binarny, jak to występuje w problemie SAT, permutacja liczb naturalnych, jak w TSP, czy też wektor liczb zmiennopozycyjnych, jak w wypadku NLP. Nie jesteśmy ograniczeni do tych wymienionych sposobów reprezentacji. Do reprezentacji potencjalnych rozwiązań moglibyśmy użyć tablic, grafów, automatów ze skończoną liczbą stanów i innych bardziej skomplikowanych struktur danych. W szczególności osobniki mogą reprezentować zestaw reguł (mających strukturę o zmiennej długości), schemat jakiejś skomplikowanej konstrukcji lub nawet program komputerowy. W istocie możliwości są nieograniczone!

Zauważmy jednak, że wybór operatorów różnicowania, które odpowiadają za zmienianie rodziców w potomstwo, zależy silnie od wybranej reprezentacji. Widzieliśmy już (podrozdział 6.1) mutację przez przełączenie, które przełącza wartości jednego bitu, oraz tak zwane „krzyżowanie jednopunktowe” – dwa operatory, które są często stosowane przy reprezentacji binarnej. Widzieliśmy też (podrozdział 6.2) operatory zamiany lub odwracania (transpozycji) stosowane przy reprezentacji całkowitoliczbowej oraz specjalny rodzaj krzyżowania, który na podstawie dwóch różnych permutacji tworzy jedną permutację liczb całkowitych. Opisaliśmy też (podrozdział 6.3) mutację Gaussa, stosowaną przy reprezentacjach zmiennopozycyjnych.

Mimo znacznych różnic w podejściu do tych problemów, implementacja algorytmów ewolucyjnych była uważana za dość łatwą. Niezależnie od tego, czy mamy do czynienia z SAT, TSP czy NLP, zasada była taka sama: 1) utworzyć populację osobników, które reprezentują potencjalne rozwiązania zadanego problemu, 2) ocenić je, 3) wprowadzić jakiś nacisk selekcyjny, który będzie promował lepsze osobniki lub eliminował osobniki gorszej jakości oraz 4) stosować pewne operatory różnicowania, które generują nowe, przeznaczone do oceniania rozwiązania. Następnie należy powtarzać wiele razy tę ewolucyjną pętlę oceny, selekcji i różnicowania (kroki 2–4)¹.

¹ Pętla ta, w zależności od konkretnej implementacji, może też wykonywać po kolei różnicowanie (wariację), selekcję i ocenianie.

Jest jeszcze kilka ciekawych obserwacji, które możemy zrobić. Nie będziemy tutaj przedstawiać ich szczegółowej analizy, ale przedstawimy kilka uwag na ich temat. Przede wszystkim powinno być jasne to, że podejście ewolucyjne jest bardzo szerokie. Możemy na przykład przytaczać argumenty świadczące o tym, że symulowane wyżarzanie jest szczególnym przypadkiem algorytmu ewolucyjnego, w którym rozmiar populacji jest ograniczony do jednego osobnika, operator różnicowania korzysta z jednego rodzica, a procedura selekcji jest oparta na zewnętrznym parametrze zwanym *temperaturą*. W rozdziale 10 omawiamy algorytmy ewolucyjne, które zmieniają zakres lub częstotliwość operatorów mutacji, a wspomniane powiązanie między symulowanym wyżarzaniem a algorytmami ewolucyjnymi będzie przedstawione bardziej bezpośrednio.

Poszukiwanie z tabu, w przeciwnieństwie do wyżarzania, jest oparte na zupełnie innych pomysłach, ale nawet i tutaj możliwe jest rozszerzenie algorytmu ewolucyjnego tak, żeby korzystał z pamięci, a wtedy poszukiwanie z tabu staje się jego szczególnym przypadkiem. Jest dość łatwo wyobrazić sobie dodanie albo jednostki pamięci do każdego osobnika (np. przez dodanie dodatkowych parametrów do reprezentowanej struktury oraz jakieś reguły interpretującej, która by operowała na tych parametrach i dzięki nim kierowała poszukiwaniem; zob. dodatkowo rozdział 11), albo pamięci globalnej (czegoś w rodzaju biblioteki wcześniejszych pomysłów) dla całej populacji (np. algorytmy kulturowe [388]). Algorytmy ewolucyjne mogą w rzeczywistości dość naturalnie przyjmować pomysły wykorzystujące poszukiwanie z tabu.

W rozdziale 3 przekonywaliśmy, że jedna z możliwych klasyfikacji algorytmów optymalizacyjnych może być oparta na tym, jak algorytm „postrzega” przestrzeń przeszukiwania. Algorytmy z jednej klasy (np. algorytmy zachłanne, programowanie dynamiczne, podziału i ograniczeń, A^*) oceniają raczej podprzestrzenie, a nie poszczególne rozwiązania. Algorytmy z innej klasy (np. algorytmy wspinające się, symulowane wyżarzanie, poszukiwanie z tabu) traktują całą przestrzeń przeszukiwania jako jednorodny zbiór potencjalnych rozwiązań. Tylko pojedyncze pełne rozwiązania są oceniane i nie występuje tutaj pojęcie podprzestrzeni¹. Z drugiej strony algorytmy ewolucyjne mogą łączyć te dwa podejścia, umożliwiając osobnikom opisywanie podprzestrzeni i konkretnego rozwiązania.

Istnieją także różne dodatkowe możliwości, które możemy przebadać. Wiele z nich omawiamy szeroko w dalszej części tego tekstu, ale o niektórych z nich wspomnimy już tutaj, żeby dać Ci lepszą perspektywę na tę fascynującą dziedzinę!

Większość algorytmów wymaga ustawienia pewnych parametrów działania (np. żeby zastosować symulowane wyżarzanie, musisz ustalić początkową temperaturę, szybkość schładzania, maksymalną liczbę wykonania głównej pętli dla danej temperatury itp.). Podobnie algorytmy ewolucyjne wymagają również ustalenia kilku z nich. Parametry te obejmują rozmiar populacji, prawdopodobieństwa (częstości) dla różnych operatorów, parametry sterujące naciiskiem selekcyjnym w systemie, współczynniki kary (jeśli zastosowaliśmy funkcję

¹Oprócz oczywiście otoczenia, które jest pewną podprzestrzenią.

kary dla rozwiązania problemu optymalizacyjnego z ograniczeniami) oraz inne, na przykład rozmiar kroku mutacji, liczbę punktów krzyżowania itp. Wartości takich parametrów mogą określić, czy algorytm znajdzie rozwiązanie bliskie optymalnemu i, co jest również ważne, czy rozsądne rozwiązanie znajdzie efektywnie. Dobranie odpowiednich wartości parametrów jest zajęciem czasochłonnym i w wypadku wielu problemów poświęcono dużo wysiłku, żeby opracować dobre heurystyki do ich znajdowania.

Przypomnijmy jednak, że algorytmy ewolucyjne realizują koncepcję ewolucji i że sama ewolucja musiała przejść pewien proces, żeby osiągnąć aktualny poziom skomplikowania. Naturalne jest zatem, żeby spodziewać się, że możemy użyć adaptacji nie tylko do odnajdowania rozwiązań problemów, ale także do dostosowania algorytmu do konkretnego problemu. Formalnie rzecz biorąc, sprawdza się to do modyfikacji wartości parametrów w czasie wykonywania algorytmu z uwzględnieniem rzeczywistego procesu poszukiwania. Zgodnie z wnioskami z późniejszej części tekstu, sterowanie wartościami różnych parametrów algorytmu ewolucyjnego potencjalnie umożliwia dostosowanie algorytmu do problemu w trakcie jego rozwiązywania!

Co więcej, wiele problemów świata rzeczywistego wymaga wielu rozwiązań. Jest to częste, gdy spełnionych ma być wiele uwarunkowań i trudno jest uwzględnić je za pomocą funkcji normalizującej. Byłoby dobrze, gdybyśmy mogli mieć algorytm, który daje więcej niż jedno dobre rozwiązanie! Dysponowanie kilkoma alternatywnymi wynikami, z których możemy wybierać, jest często bardzo użyteczne!

Algorytmy ewolucyjne możemy łatwo tak dostosować, żeby odpowiadały temu nowemu wymaganiu. Możliwe jest wykorzystanie informacji zwrotnej na temat różnorodności populacji i w szczególności na temat obecności innych osobników w otoczeniu któregoś z nich. Takie sprzężenie zwrotne odpowiednio wprowadzone do funkcji oceny może być wykorzystane do sterowania liczbą uzyskiwanych rozwiązań.

W rozdziale 1 wspomnieliśmy, że wiele problemów wymaga zmieniających się w czasie rozwiązań. Podczas rozwiązywania problemu często otrzymujemy nową информацию, która w trakcie rozwiązywania problemu wpływa na funkcję oceny. Może się tak zdarzyć w trakcie pokonywania najlepszej trasy TSP. Możemy przecież odkryć, że dzięki nieprzewidzianym okolicznościom jeden z odcinków trasy nie jest już dostępny. Lub też podczas opracowywania planu pracy w fabryce możemy się dowiedzieć, że jedna z maszyn uległa awarii. Takie niespodziewane zdarzenia w świecie rzeczywistym pojawiają się przez cały czas i wymagają szybkiego dostosowania aktualnego rozwiązania, żeby sprostać nowemu wyzwaniu.

Ogromna większość klasycznych algorytmów optymalizacyjnych zakłada ustaloną funkcję oceny. Każda zmiana funkcji oceny wymaga wystartowania algorytmu od początku! Dla kontrastu algorytmy ewolucyjne są z zasady swojej adaptacyjne – osobniki w populacji *adaptują* się do aktualnego otoczenia, a jakość tej adaptacji jest mierzona za pomocą funkcji oceny. W wyniku tego niespodziewana zmiana modelu lub nawet samej funkcji oceny może spowodować krótkotrwałe zakłócenie procesu ewolucyjnego, ale po pewnym

czasie, zależnym od konkretnego problemu, osobniki w populacji mogą dostosować się do wymogów nowego układu. Metody ewolucyjne nie wymagają wznowienia algorytmu przy każdej zmianie. Zamiast tego kontynuują swój proces adaptacyjny w świetle podobieństw między dotychczasowym doświadczeniem a obecnymi wymaganiami. Ilustruje to wielki potencjał algorytmów ewolucyjnych.

Inną zaletą podejścia ewolucyjnego przy rozwiązywaniu problemów jest możliwość połączenia go z innymi metodami. Możemy do niego bezpośrednio wprowadzić specjalne operatory różnicowania i operatory selekcji, które wykonują działania właściwe dla wspinania się bądź innej wybranej metody.

I jeszcze jedna możliwa zaleta. Przypomnijmy nasze rozważania z podrozdziału 1.3 dotyczące właściciela sieci supermarketów, który musi zadecydować, gdzie umieścić nowy sklep, gdy jego konkurenci również podejmują podobną decyzję. Oczywiście próbują oni maksymalizować swoje zyski, co z dużym prawdopodobieństwem zmniejsza zyski naszego właściciela sieci supermarketów. W ten sposób proces podejmowania decyzji jest bardzo skomplikowany (on myśli, że oni myślą, że on myśli, że oni myślą...), ale my możemy zamodelować tę sytuację, uruchamiając dwa konkurujące ze sobą procesy ewolucyjne. Pierwszy z nich optymalizowałby strategię właściciela sieci supermarketów, a drugi obsługiwałby strategię jego przeciwnika. Zwróć uwagę, że ocena strategii każdej ze stron jest oparta na bieżącej (lub być może przewidywanej) strategii oponenta. Tego rodzaju ewolucyjne współzawodnictwo jest nazywane *modelem koewolucyjnym*. Takie modele mogą być często wykorzystywane do odnajdowania rozwiązań problemów, które ze względu na relacje między graczami wydają się bardzo trudne do opisania za pomocą precyzyjnych matematycznych formuł.

Jeszcze jedną rzecz warto wspomnieć przy omawianiu zalet algorytmów ewolucyjnych: równoległość. Zrównoleglanie umożliwia osiągnięcie rozwiązań problemów, z którymi do tej pory nie można było sobie poradzić. Algorytmy ewolucyjne doskonale oddają się równoległym implementacjom, gdyż są z natury swej jawnie równolegle! Jest dość łatwo przypisać każdemu osobnikowi w populacji jeden procesor albo podzielić populację na kilka podpopulacji (to ostatnie możemy rozszerzyć dalej przez wprowadzenie pojęcia migracji osobników między podpopulacjami).

Podczas wymieniania potencjalnych zalet algorytmów ewolucyjnych warto wspomnieć „argument” ostateczny, który jest bardziej związany z psychologią programowania niż z efektywnością tego podejścia. Gdy uruchamiamy sztuczny proces ewolucyjny, stawiamy się w roli *stwórcy* i to my określamy reguły symulacji, co przypomina ustalanie fizyki świata. Możemy chcieć zwrócić szczególną uwagę na konkretne mechanizmy naturalnej ewolucji albo rozważyć elementy niewystępujące w systemach biologicznych (np. kojarzenie więcej niż dwóch rodziców). Możemy nawet zupełnie odejść od praw natury i wprowadzić *ewolucję lamarkowską*, w której osobniki nabierają cech w trakcie swojego „życia”, a następnie przekazują je swoim potomkom. Bariera między *fenotypem* (zachowaniem osobnika) a *genotypem* (genetycznym opisem osobnika) nie musi dla nas istnieć!

Możemy wprowadzić pojęcie płci osobników, przypisać im wiek (pewną liczbę pokoleń), a następnie używać tego do określenia, jak długo mogą one przeżyć. Możemy do osobników wprowadzić pamięć lub też badać pomysł rodzin osobników i uczenia się społecznego. Moglibyśmy nawet wprowadzić konkurencje między całymi populacjami i, być może, migrację osobników między nimi. Moglibyśmy zastanowić się nad dodaniem *efektu Baldwina*, w którym osobniki uczą się w trakcie swojego życia i w ten sposób mogą mieć wpływ na swoją jakość w procesie ewolucji. Wszystkie te drożki prowadzą do różnych wyborów. Na przykład, jakie rozwiązania powinny migrować: najlepsze, przeciętne, najgorsze? Rzeczywiście możliwości, które stoją przed nami jako programistami i osobami rozwiązyającymi zadania, są w zasadzie nieograniczone!

VII. Jedna z tych rzeczy jest niepodobna do pozostałych

Wysoki mężczyzna wchodzi do sklepu jubilerskiego i stawia na ladzie torbę pełną starych monet. Jubiler bierze torbę i wysypuje jej zawartość.

- Ile da mi pan za te monety? – pyta mężczyzna.
- O, czerwony złoty z 1818 r., na dodatek w bardzo dobrym stanie – mruczy jubiler, oglądając pieniądze przez okular. Myślę, że każda z nich jest warta 100 złotych. Ma pan dwanaście takich monet, tak?
- No cóż, jest coś, o czym powinien pan wiedzieć. Widzi pan, gdy kupowałem te monety, facet, który mi je sprzedawał, powiedział, że jedna z nich jest fałszywa.
- Jak to? Uczciwy złodziej?! – wykrzykuje jubiler.

Mija chwila ciszy, w trakcie której wysoki mężczyzna próbuje rozeznać, czy jubiler miał na myśli jego, czy faceta, który sprzedał mu monety.

- Dobrze – mówi wreszcie lekko zdenerwowany. – W jaki sposób znajdziemy tę fałszywą monetę?
- No cóż, mam pomysł – odpowiada jubiler. – Większość fałszywych monet jest wykonana z tańszego metalu, nie ważą więc tyle samo co prawdziwe monety. Mam wagę szalkową. Jedyne więc, co musimy zrobić, to zastanowić się, jak zważyć monety, żeby wykryć tę podrobioną.

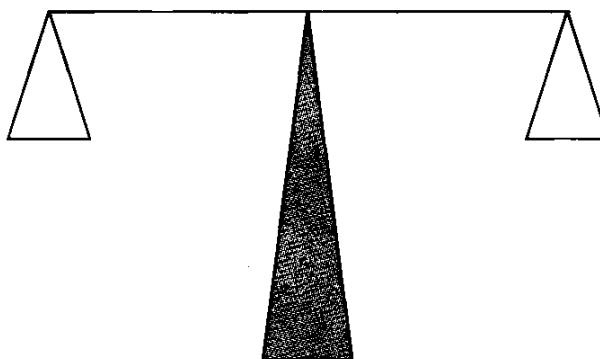
Wysoki mężczyzna uśmiecha się. – Och, to proste. Możemy to zrobić w trzech ważeniach. Nie tylko znajdziemy fałszywą monetę, ale będziemy też wiedzieć, czy jest ona cięższa, czy lżejsza od prawdziwej monety.

- Naprawdę? W jaki sposób? – pyta jubiler, zaskoczony zaprezentowanym opisem.
- Ostatnio czytałem bardzo dobrą książkę o heurystycznym podejściu do rozwiązywania problemów – odpowiada właściciel monet. Problem w tym, że nie pamiętam, jak to robiono...

Obaj mężczyźni pochylają się nad wagą i zaczynają kombinować. A Ty – czy wiesz, jak znaleźć fałszywą monetę?

Odlóż na chwilę książkę i sprawdź, czy potrafisz sam znaleźć początek rozwiązania.

Problem polega na znalezieniu pośród innych rzeczy konkretnego przedmiotu o cechach odróżniających go od reszty. Narzędziem, którym dysponujemy, jest waga szalkowa (rys. VII.1). Za pomocą tego urządzenia możemy jedynie stwierdzić, że jedna ze stron jest cięższa od drugiej lub że obie ważą tyle samo.



Rysunek VII.1. Waga szalkowa

Naturalnym sposobem podejścia do tego problemu jest podzielenie dwunastu monet na grupy i ważenie tych grup. Trudność polega jednak na tym, że jeśli podzielimy pieniądze na dwie grupy po sześć monet i porównamy je, to niezależnie od otrzymanego wyniku nie będziemy mieli pojęcia, jak podzielić te grupy dalej, żeby otrzymać rozwiązania w dwóch kolejnych ważeniu. Pamiętaj bowiem, że celem jest odnalezienie fałszywej monety przy trzech tylko użyciach wagi.

Rozważmy inny podział pieniędzy: na trzy grupy po cztery monety. Przyjmijmy, że dzielimy je następująco:

$$G_1 = (1, 2, 3, 4), \quad G_2 = (5, 6, 7, 8) \quad \text{i} \quad G_3 = (9, 10, 11, 12)$$

W pierwszym ważeniu porównujemy grupy G_1 i G_2 . Albo ważą tyle samo, albo jedna z nich jest cięższa. Rozważmy po kolejno te przypadki.

Jeśli monety z grup G_1 i G_2 ważą tyle samo, fałszywa moneta musi znajdować się w grupie G_3 . Jednocześnie dowiadujemy się, że wszystkie monety z grup G_1 i G_2 są prawdziwe. Dlatego też w drugim ważeniu możemy porównać trzy dowolne prawdziwe monety (powiedzmy: 1, 2 i 3) z trzema monetami z grupy G_3 :

$$(1, 2, 3) \text{ kontra } (9, 10, 11)$$

Znowu są dwa możliwe wyniki.

- Monety ważą tyle samo. Oznacza to, że fałszywą monetą jest 12, ponieważ jest to jedyna moneta z grupy G_3 , która nie brała udziału w drugim ważeniu. Trzecie ważenie (np. 1 kontra 12) określi, czy fałszywa moneta jest cięższa czy lżejsza od prawdziwej monety.

- Monety nie ważą tyle samo. Oznacza to, że fałszywa jest moneta 9, 10 lub 11. W tym momencie wiemy także, czy fałszywa moneta jest cięższa czy lżejsza od prawdziwej: jeśli (1, 2, 3) są cięższe niż (9, 10, 11), to znaczy, że fałszywa moneta jest cięższa, i odwrotnie. Trzecie ważenie (np. 9 kontra 10) określi, która moneta jest fałszywa. Jeśli monety ważą tyle samo, fałszywą monetą jest 11. Jeśli ich waga się różni, to na podstawie uzyskanej wcześniej informacji, czy fałszywa moneta jest cięższa czy lżejsza, możemy dokonać odpowiedniego wyboru.

Jeśli więc poszczęści się nam i monety z grup G_1 i G_2 ważą tyle samo, wszystko jest bardzo proste.

Co jednak zrobić, jeśli nie uzyskamy równowagi w pierwszym ważeniu?

W takim wypadku mamy dwie informacje: 1) fałszywka znajduje się w grupie G_1 lub G_2 i 2) monety 9, 10, 11 i 12 są prawdziwe.

Przed nami chyba najtrudniejszy krok na drodze do rozwiązania, musimy bowiem sensownie zaplanować drugie ważenie. Możemy rozważyć zważenie rozdzielonych monet z grupy G_1 (powiedzmy monety 1 i 2 na lewej szalce kontra monety 3 i 4 na prawej), ale to nie wystarczy. Jeśli będą ważyć tyle samo, dowiemy się, że fałszywa moneta znajduje się w grupie G_2 ; będziemy także wiedzieli, czy fałszywa moneta jest cięższa czy lżejsza. Nie będziemy jednak mogli odnaleźć jej w trzecim i ostatnim ważeniu, ponieważ na tym etapie będziemy mieli cztery monety: 5, 6, 7 i 8. Musimy więc wyeliminować jedną z nich.

Mogemy to zrobić, dokładając w drugim ważeniu jedną monetę z grupy G_2 (powiedzmy 5) na lewą szalkę i dodając jedną prawdziwą monetę (powiedzmy 12) do 3 i 4 na prawej szalce. Drugie ważenie wygląda więc następująco:

(1, 2, 5) kontra (3, 4, 12)

Założmy, że w pierwszym ważeniu monety (1, 2, 3, 4) okazały się cięższe od (5, 6, 7, 8). Mamy wówczas trzy możliwe rezultaty drugiego ważenia.

- Monety (1, 2, 5) są cięższe. Oznacza to, że monety 3, 4 i 5 są prawdziwe, gdyż mimo zmiany ich położenia na wadze uzyskaliśmy taki sam wynik ważenia (tzn. zawartość lewej szalki jest cięższa). Ponieważ moneta 12 też jest prawdziwa, fałszywa jest moneta 1 lub 2. Wiemy też, że fałszywa moneta jest cięższa, wskaże ją więc trzecie ważenie (1 kontra 2).
- Monety (3, 4, 12) są cięższe. Ponieważ wyniki pierwszego i drugiego ważenia są różne (tzn. najpierw zawartość lewej szalki była cięższa, a teraz – prawej), fałszywa moneta musiała zostać przełożona z jednej szalki na drugą. Tak więc albo moneta 3 lub 4 jest fałszywa i cięższa, albo moneta 5 jest fałszywa i lżejsza. Trzecie ważenie (3 kontra 4) wskaże fałszywą monetę. Jeśli będą ważyły tyle samo, fałszywa jest moneta 5.
- Monety (1, 2, 5) i (3, 4, 12) ważą tyle samo. W takim wypadku fałszywa moneta nie brała udziału w drugim ważeniu, jest więc to moneta 6, 7 lub 8. Z pierwszego ważenia wiemy też, że fałszywa moneta jest lżejsza, więc trzecie ważenie (powiedzmy: 6 kontra 7) wskaże poszukiwaną monetę.

W ten sposób w trzech ważeniach jubiler i jego klient mogą przekonać się, która moneta jest fałszywa oraz czy jest ona cięższa, czy lżejsza od pozostałych.

Właściwe podejście do tego problemu polega na takim podziale przestrzeni rozwiązań, żeby zastosowany ciąg porównań wyeliminował wiele możliwości i zostawił tylko kilka, które wymagają bliższego zbadania. Strategia ta jest często bardzo skuteczna. Kto chciałby tracić czas na szukanie odpowiedzi, jeśli odpowiedź leży gdzie indziej? Uważaj tylko na naturalną skłonność do podziału przedmiotów na połowę. W wielu wypadkach inny podział daje dużo lepsze efekty.

– Fantastyczne! – wykrzyknął jubiler. – Jednak przypomniał pan sobie rozwiązanie!

– Miałem sporo szczęścia, ale wszystko dobre, co się dobrze kończy – uśmiechnął się wysoki mężczyzna, wychodząc ze sklepu z 1100 złotymi i fałszywą monetą.

W drzwiach zderzył się z wchodzącą starszą panią.

– Proszę mi wybaczyć! – krzyknął przepraszać, gdy zbliżała się do lady.

– W czym mogę służyć? – spytał ją jubiler.

– Widzi pan, mam tu 120 złotych dziesięciozłotówek z 1925 r. – powiedziała kobieta, wskazując na pobrękującą w jej rękach torbę.

– I chce je pani sprzedać?

– Tak, ale widzi pan, jedna z tych monet jest fałszywa...

Sprawdź, czy potrafisz odnaleźć fałszywą monetę w pięciu ważeniach i stwierdzić, czy jest ona cięższa czy lżejsza od pozostałych monet!

7. Projektowanie algorytmów ewolucyjnych

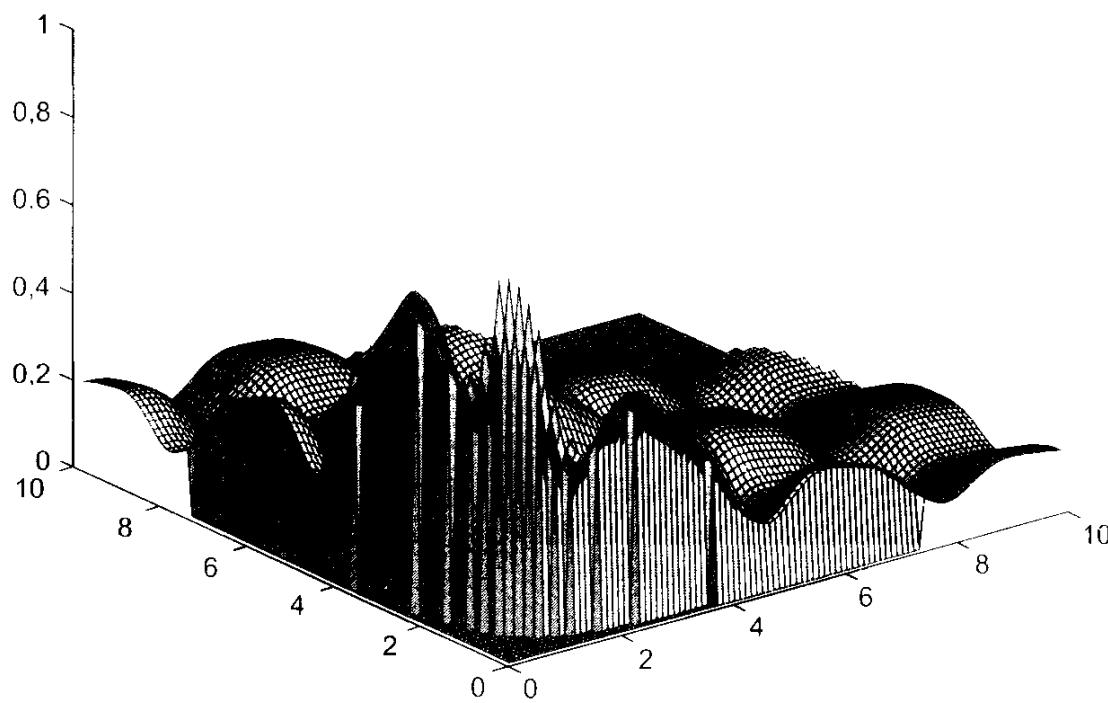
Tysiąc rzeczy idzie naprzód,
dziewięćset dziewięćdziesiąt dziewięć się cofa:
oto postęp.

Henri-Frédéric Amiel: *Dziennik intymny*¹

Podstawowa koncepcja ewolucyjnego rozwiązywania problemów jest dość prosta. Populacja kandydatów na rozwiązania naszego zadania podlega kolejnym krokom ewolucji realizowanym przez różnicowanie i selekcję. Losowe różnicowanie daje mechanizm odkrywania nowych rozwiązań. Selekcja określa, które rozwiązania należy zachować jako podstawę do dalszych poszukiwań. Ujmując to metaforycznie, poszukiwanie jest prowadzone w obszarze złożonym z pagórków i dolin (rys. 7.1), który jest również „nazywany powierzchnią odpowiedzi”, gdyż wskazuje odpowiedź funkcji oceny na każde możliwe rozwiązanie próbne. Celem poszukiwań jest najczęściej lokalizacja rozwiązania lub zestawu rozwiązań, który ma wystarczającą jakość mierzoną za pomocą funkcji oceny. Nie wystarczy jednak znaleźć te rozwiązania, musimy wyszukiwać je szybko. W końcu, stosując wyliczanie, również możemy je znaleźć, ale wtedy, zanim się doczekamy, postarzejemy się. Szybkość, z jaką odpowiednie rozwiązania mogą być znalezione, częściowo zależy od dokonanych przez nas wyborów przy określaniu reprezentacji potencjalnych rozwiązań, funkcji oceny, konkretnych operatorów różnicowania i operatorów selekcji oraz rozmiaru i sposobu inicjowania populacji początkowej, a także wielu innych aspektów obliczenia. Przy opracowywaniu dających wyniki algorytmów ewolucyjnych najważniejsze jest dokonanie dla wspomnianych elementów odpowiednich wyborów, biorąc pod uwagę rozwiązywany problem.

Czyż nie byłoby wspaniale mieć jeden najlepszy wybór tych wszystkich parametrów, który dawałby nam optymalne działanie algorytmu niezależnie od problemu, który usiłujemy rozwiązać? Po znalezieniu takiego układu parametrów moglibyśmy go bezpośrednio zaimplementować i nie zajmować się już

¹Przekład Joanna Guze, Czytelnik, Warszawa 1997, wydanie I (przyp. tłum.).



Rysunek 7.1. Funkcja oceny może wyglądać jak obszar z pagórkami i dolinami (zob. rys. 1.2)

problemami związanymi z reprezentacją rozwiązań, sposobem tworzenia operatorów różnicowania lub sposobem określania, które rozwiązania należy pozostawić jako podstawę do dalszych poszukiwań. Niestety, możemy udowodnić (przy pewnych słabych założeniach), że w rzeczywistości taki optymalny zestaw wyborów nie istnieje [499]. W istocie, wszystkie procedury poszukiwania, które nie dokonują zmiany próbki punktów przestrzeni poszukiwań, średnio na wszystkich problemach działają dokładnie tak samo. Chociaż z początku może wydawać się to nieintuicyjne, to przy poszukiwaniu maksimum algorytm wspinający się, który nigdy nie dokonuje zmiany próbki punktów przestrzeni, będzie średnio, dla wszystkich możliwych funkcji, działał tak samo dobrze jak losowe poszukiwanie na ślepo. Zdanie to jest również prawdziwe dla wszystkich algorytmów ewolucyjnych. Jeśli nie wiesz czegoś o problemie, z którym masz do czynienia, to nie masz żadnej przesłanki umożliwiającej wybranie jakiegoś konkretnego algorytmu poszukiwania.

W pierwszej chwili ten matematyczny wynik może wydawać się trochę przygnębiający. Oznacza to, że jeśli przy rozwiązywaniu problemu nie wykorzystasz jakiejś wynikającej z niego wiedzy, to nie ma znaczenia, jakiej użyjesz reprezentacji, jakich operatorów różnicowania, jak duża jest Twoja populacja, jaki rodzaj operatora selekcji zaimplementujesz i tak dalej. W pewnym sensie sytuacja jest nawet gorsza. Możemy pokazać, że żadna bijektywna¹ reprezentacja nie daje możliwości dostępnej przy innej reprezentacji [156]. Wszystko, co potrafisz zrobić na ciągach binarnych, możesz zrobić przy reprezentacji ósemkowej, szesnastkowej czy też przy podstawie 13. Co więcej, niezależnie od tego,

¹Odwzorowanie bijektywne między dwoma zbiorami to takie, które jest różnowartościowe i „na”.

jakie operatory poszukiwania wybierzesz, ich działanie nie będzie jedynym możliwym. Zawsze istnieje inny operator różnicowania, który będzie naśladował ten, który wybrałeś, niezależnie od tego, czy jest oparty na jednym rodzicu, dwóch czy więcej. Jeśli wybrałeś, powiedzmy, jednopunktową rekombinację na ciągach binarnych, to ktoś inny (jeśli jest sprytny) może opracować i utworzyć dokładnie taki sam algorytm przy użyciu kodowania o podstawie, powiedzmy, 17 oraz inny operator różnicowania dla dwóch rodziców, który nie jest oparty na rekombinacji. W ten sposób nie tylko nie ma najlepszego sposobu określenia parametrów algorytmu poszukiwania, średnio dla wszystkich problemów, ale także nie ma najlepszego sposobu poszukiwania rozwiązania dla wszystkich zadań. Ujmując to metaforycznie, jest wiele sposobów na obdarcie kota ze skóry, pod warunkiem że kot w końcu zostaje pozbawiony skóry; nie ma znaczenia, jak się to robi!

Zamiast postrzegać te wyniki jako kubel zimnej wody na nasz entuzjazm przy poszukiwaniu dobrych algorytmów poszukiwania, spójrzmy na sprawy bardziej optymistycznie. Po pierwsze, wiemy, że musimy użyć wiedzy na temat naszego problemu, żeby opracować algorytm, który będzie bardziej efektywny niż powiedzmy próbowanie na ślepą. Oznacza to, że musimy się dowieść jak najwięcej o problemie i spróbować włączyć tę wiedzę w użyteczne procedury. Po drugie, wiemy, że nie będzie najlepszego sposobu rozwiązania tego problemu. Rzeczywiście, może być wiele najlepszych dróg i jeszcze więcej sposobów lepszych niż oczekiwane! Niezależnie od tego, z jakim problemem będziemy mieli do czynienia, nie musimy trapić się wymuszaniem określonej reprezentacji lub operatora poszukiwania dla tego problemu. Naszym zadaniem jest opracowanie odpowiedniego narzędzia, które będzie dostosowane do problemu.

Algorytmy ewolucyjne są nieprawdopodobnie wszechstronne. Zmieniając reprezentację, operatory różnicowania, rozmiar populacji, mechanizm selekcji, inicjowanie, funkcję oceny i inne ich aspekty, uzyskujemy dostęp do zróżnicowanego zakresu procedur poszukiwania. Algorytmy ewolucyjne przypominają szwajcarski scyzoryk [125]: poręczny zestaw narzędzi, które możemy stosować do najrozmaitszych zadań. Jeśli kiedykolwiek używałeś takiego narzędzia, to wiesz, że ta wszechstronność nie jest za darmo. Dla każdego zastosowania zwykle istnieje lepsze narzędzie opracowane specjalnie na jego potrzeby. Jeśli miałbyś wyciągnąć gwóźdź z deski, to obcęgami prawdopodobnie zrobisz to lepiej niż czymkolwiek, co jesteś w stanie znaleźć w szwajcarskim scyzoryku. Podobnie, jeśli miałbyś usunąć drzazgę z palca, to starannie wykonaną pęsetą zrobisz to lepiej niż jakimś prowizorycznym narzędziem dostępnym w scyzoryku. Jeśli jednak nie wiesz dokładnie, jakie przed Tobą stoi zadanie, to elastyczność noża szwajcarskiego staje się bardzo poręczna. Wyobraźmy sobie wyjmowanie gwoździa za pomocą pęsety lub wyciąganie drzazgi za pomocą obcęgów! Posiadanie szwajcarskiego noża daje Ci możliwość szybkiego i efektywnego radzenia sobie z wieloma problemami, chociaż może dla danego zadania istnieć lepsze narzędzie. Poza tym nie musisz nosić obcęgów, pęsety, śrubokręta, otwieracza, szydła, pilnika, wykałaczki itd.

Analogia ta przenosi się bezpośrednio na stosowanie algorytmów ewolucyjnych. Przyjrzyjmy się problemowi znajdowania minimum n -wymiarowej funkcji kwadratowej

$$f(\mathbf{x}) = \sum_{i=1}^n x_i^2$$

Przypuśćmy, że miałbyś użyć prostego algorytmu ewolucyjnego, w którym miałbyś jednego rodzica \mathbf{x} , i że wygenerowałeś potomka \mathbf{x}' w ten sposób:

$$x'_i = x_i + N(0, \sigma)$$

dla każdego wymiaru $i = 1, \dots, n$, przy czym $N(0, \sigma)$ jest losową zmienną Gaussa o wartości średniej zero i odchyleniu standardowym σ . Dla tego algorytmu Rechenberg udowodnił ponad 25 lat temu, że maksymalna szybkość zbieżności jest osiągana, gdy σ jest proporcjonalna do wartości $\sqrt{f(\mathbf{x})}$. W szczególności [383]

$$\sigma^* = \frac{1,224}{n} \sqrt{f(\mathbf{x})}$$

Co więcej, spodziewana szybkość zbieżności przy tych uwarunkowaniach jest geometryczna ze względu na liczbę iteracji [18].

Ciekawe jest porównanie tego z szybkością zbieżności algorytmu, który został opracowany specjalnie dla funkcji kwadratowych. Najlepszym algorytmem dla takich funkcji jest procedura Gaussa-Newtona (rozdział 3). Jest on tak zaprojektowany, żeby przechodzić w jednym kroku z dowolnego punktu funkcji kwadratowej do jej minimum. To naprawdę szybka zbieżność! W istocie mówienie o szybkości zbieżności takiego algorytmu nie ma sensu. Nie możemy uzyskać lepszego wyniku jak osiągnięcie doskonałego rozwiązania w ramach jednej iteracji! W ten sposób wiemy, że procedura Gaussa-Newtona jest tą procedurą, którą trzeba wybrać, jeśli wykonujemy poszukiwanie dla funkcji kwadratowej. Algorytmy ewolucyjne, nawet po zastosowaniu innych trików przyspieszających lokalną zbieżność (zob. [46]), nie mogą dorównać szybkości działania tej konkretnej funkcji.

Zobaczmy jednak, co się stanie, gdy przejdziemy do funkcji przedstawionej na rys. 7.1 i zastosujmy dla niej procedurę Gaussa-Newtona. W procedurze tej założono, że każda przeszukiwana funkcja ma powierzchnię kwadratową. W dowolnym punkcie działa tak, jakby cała funkcja była kwadratowa, oblicza hipotetyczne optimum globalne, a następnie kontynuuje swoje działanie, aż osiągnie lokalne optimum. Przy pagórkowatej funkcji takiej jak na rys. 7.1 takie lokalne optimum może nie być wystarczające. Z kolei algorytm ewolucyjny może pokonać te optima lokalne, uciec ze związków z nimi pułapek i odkryć optimum globalne, przy czym może tego dokonać dla wielu rodzajów funkcji.

W istocie procedura Gaussa-Newtona ma bardzo ostre założenie: wszystko na świecie wygląda jak miska. W wypadku algorytmów ewolucyjnych mamy szersze spojrzenie i często nie czynimy żadnych założeń przed ujęciem problemu poszukiwania w zestaw struktur danych oraz operatorów różnicowania

i operatorów selekcji. W wyniku tego, gdy przeszukiwana funkcja wygląda rzeczywiście jak miska, działają one dalej efektywnie, ale nie tak efektywnie, jakbyśmy chcieli. Z kolei, jeśli nasza funkcja nie wygląda jak funkcja kwadratowa, to procedura Gaussa-Newtona może w ogóle nie działać. Staje się pęsetą, za pomocą której próbuje się wyciągnąć gwóźdź z deski. Wybierając struktury, operatory wyszukiwania i mechanizmy selekcji dla swojego algorytmu, przyjmujesz istotne założenia dotyczące rodzaju problemów, z jakimi masz nadzieję się zetknąć. Jeśli Twoje założenia zbyt zawężą zakres problemów, to możesz uzyskać bardzo efektywne poszukiwanie, które jest zupełnie nieadekwatne do zadania.

Dobrze by było, gdybyśmy mogli umieścić wszystkie problemy świata rzeczywistego w ramach poszukiwania w funkcji kwadratowej! Moglibyśmy wtedy rozwiązywać nasze problemy w jednym kroku! Niestety wszystkie tego typu problemy zostały rozwiązane dawno temu. To były te łatwe problemy. Oczywiście nie jest niemożliwe przekształcenie trudnych problemów w trywialne, ale rzadko jest to praktyczne. Zwróćmy znów uwagę na funkcję z rys. 7.1. Istnieje przekształcenie dające się zastosować do każdego punktu w obszarze dopuszczalnych rozwiązań, w którego wyniku wszystkie te punkty znajdują się na powierzchni funkcji kwadratowej. Ale jest to odwzorowanie punkt-do-punktu wymagające w istocie znajomości odpowiedzi na pytanie, zanim zaczniemy je wykonywać! W takim wypadku nie trzeba byłoby przeprowadzać żadnego poszukiwania choćby przez jedną iterację. Nie powinniśmy wykluczać możliwości stosowania funkcji oceny, które sprawiają, że problemy, z którymi mamy do czynienia, są odrobinę łatwiejsze, ale nie możemy też liczyć na rozwiązywanie problemów świata rzeczywistego za pomocą metody Gaussa-Newtona.

Podobnie nie możemy mieć nadziei, że za pomocą *jakiejkolwiek* jednej procedury czy operatora rozwiążemy wszystkie problemy. Na przykład tak jak metoda Gaussa-Newtona działa przy określonych założeniach, tak przy pewnych założeniach działają jednopunktowa rekombinacja, dwupunktowa rekombinacja, mieszanie, różnicowanie Gaussa itd. Z każdym operatorem wiąże się zestaw założeń. Najważniejsze jest zrozumienie, jak szerokie lub wąskie są te niejawne założenia dla operatorów, a następnie, które kombinacje funkcji oceny, reprezentacji itp. umożliwiają radzenie sobie z konkretnymi problemami. Przy efektywnym rozwiązywaniu problemów sztuka polega na tym, żeby przenieść jak najwięcej danych z problemu do metody rozwiązania, ale nie przenieść ich zbyt dużo. Jeśli przełożymy zbyt mało informacji, to ucierpi na tym szybkość działania. Jeśli zbyt dużo, to możemy stracić szanse na uzyskanie sensownego rozwiązania.

Przyjrzyjmy się teraz bliżej każdemu z tych problemów związanych z projektowaniem algorytmów ewolucyjnych. Zanim podejmiemy ten wysiłek, podkreślimy jeszcze raz, że żaden z wyborów podejmowanych przy którymkolwiek z elementów nie może być dokonywany w izolacji od wyborów dla innych. Wszystkie elementy są blisko powiązane. Na przykład własność funkcji oceny nie może być oceniana w oderwaniu od reprezentacji oraz przyjętych operatorów różnicowania. Przy czym operatory poszukiwania mogą być ocenione tylko wówczas, gdy mamy podaną reprezentację i funkcję oceny. Co więcej,

opisanie funkcji oceny i operatorów poszukiwania może wydawać się trudne bez zrozumienia reprezentacji. Badanie działania każdego z elementów algorytmu ewolucyjnego oddzielnie może dawać jedynie ograniczoną wiedzę. Co gorsza, takie badanie może być mylące.

7.1. Reprezentacja

Reprezentacją nazywamy odwzorowanie przestrzeni wszystkich możliwych rozwiązań na przestrzeń rozwiązań zakodowanych w konkretnej strukturze danych. Przyjrzyjmy się na przykład TSP. Przestrzeń wszystkich możliwych rozwiązań zawiera trasy, które może przebyć komiwojażer. Sposób reprezentacji tych tras na komputerze nie jest wcale oczywisty. Jedna z wybranych przez nas możliwości polega na utworzeniu uporządkowanej listy miast, które mają być odwiedzone przy założeniu, że na końcu następuje powrót do początkowego miasta. W ten sposób trasa przez dziesięć miast może być reprezentowana jako

[1 3 5 10 4 2 8 9 7 6]

Tutaj reprezentację stanowi permutacja pierwszych dziesięciu liczb naturalnych i każda taka permutacja będzie dawała dopuszczalną trasę, nie jest to jednak jedyna możliwa reprezentacja.

W problemie tym mamy 181 440 różnych tras. Moglibyśmy każdej z nich przypisać liczbę od 1 do 181 440. Tak uzyskalibyśmy reprezentację dziesiętną. W istocie moglibyśmy to zrobić na 181 440! sposobów. Zatem dla reprezentacji o podstawie 10 dla TSP z dziesięcioma miastami moglibyśmy rozważać 181 440! różnych odwzorowań. Co więcej, moglibyśmy zakodować każdą liczbę dziesiętną jako liczbę dwójkową (lub liczbę przy innej podstawie) i moglibyśmy znowu 181 440! różnych odwzorowań przy tej wielkości zbioru¹. Musimy jednak przyznać, że tego rodzaju reprezentacje wyglądają na niezbyt dobrze przystosowane do naszego problemu. Nie możemy łatwo wyobrazić sobie operatora poszukiwania, który przechodziłby od jednej trasy lub większej ich liczby do innej z zachowaniem jakości rozwiązań uzyskanej w trakcie kolejnych iteracji algorytmu. Natychmiast stajemy wobec problemu integracji reprezentacji z operatorami poszukiwania przy pewnej danej funkcji oceny!

„Dobra” reprezentacja umożliwia zastosowanie operatorów poszukiwania, które utrzymują powiązanie między rodzicami a ich potomstwem. Między rodzicami a potomstwem musi istnieć użyteczny związek wykorzystywany przez operatory poszukiwania. Niezgodność operatorów z reprezentacją prowadzi do algorytmu, który jest równoważny z poszukiwaniem na ślepo, sprowadzającym się w istocie do losowego próbkowania przestrzeni rozwiązań bez troszczenia się o wyniki poprzednich próbek. Jest możliwe, że tego typu procedura będzie działała gorzej niż czyste losowe poszukiwanie. W zależności od wyboru reprezentacji użyteczność może przybierać postaci: 1) dostępnego zakresu stopnio-

¹Znowu jest oczywiste, że sama wielkość alfabetu nie wystarczy, żeby wybrać reprezentację.

wanych operatorów różnicowania, które prowadzą do odpowiedniego stopniowanego zakresu różnych zachowań potomstwa (tzn. możliwe jest dostosowanie efektywnego „rozmiaru kroku” operatora poszukiwania), 2) łączenia użytkowych niezależnych części różnych rozwiązań, 3) zapewniania operatorów, które zawsze generują dopuszczalne rozwiązania itd.

Przypomnijmy, że Fogel i Ghozel [156] udowodnili, iż w klasie reprezentacji, które są bijekcjami, żaden wybór reprezentacji nie jest lepszy niż inny. W związku z tym najlepsze, co możemy zasugerować w kwestii wyboru reprezentacji, to zastosować strukturę, która wydaje się intuicyjna na podstawie sformułowania problemu. Jeśli jesteś zainteresowany optymalizacją parametrów ciągłych, takich jak temperatura bądź ciśnienie, to intuicyjną strukturą danych są wektory liczb zmiennopozycyjnych. Jeśli z kolei chcesz określić optymalny zestaw cech, których należy użyć w klasyfikatorze wzorców, to bardziej intuicyjne może być użycie ciągów binarnych, w których każdy bit wskazuje, czy dana cecha jest wykorzystywana, czy nie, lub też możesz pomyśleć o zastosowaniu zmiennej długości listy cech, jakie mają być uwzględnione (np. [1, 3, 5] lub [1, 4, 7, 9]). W każdym wypadku, gdy tylko potrafisz w myślach określić reprezentację, możesz też określić w myślach możliwe operatory różnicowania, gdyż jeśli to ostatnie nie jest możliwe, to obrana reprezentacja jest zła.

Poniżej omawiamy różne reprezentacje powszechnie stosowane w algorytmach ewolucyjnych. Są one pogrupowane i połączone z ilustrującymi przykładami opartymi na problemach, z którymi możesz się spotkać. Podana lista zawiera większość powszechnie spotykanych reprezentacji, które możemy znaleźć w literaturze¹.

7.1.1. Wektory symboli o ustalonej długości

Przyjrzyjmy się kilku możliwościom.

Możliwość (i). Przypuśćmy, że masz rozwiązać problem wyboru podzbioru, w którym musisz określić, ile tych elementów chcesz wybrać. Pojawia się on w modelowaniu statystycznym (tzn. które niezależne zmienne losowe należy uwzględnić), optymalizacji transportu (tzn. które przedmioty powinny być przewożone poszczególnymi pojazdami) itd. Typową reprezentacją w tego typu problemach jest lista bitów $\{0, 1\}$ w wektorze o ustalonej długości (tzn. ciąg binarny). Długość wektora odpowiada liczbie przedmiotów lub parametrów, jaką dysponujemy (np. 1 oznacza, że przedmiot ma zostać wybrany, a 0 pominięty). Wybór ten jest także intuicyjny w wypadku problemów, w których ewolucji podlega kod cyfrowy w języku maszynowym, co ma szczególne znaczenie w implementacjach sprzętowych.

¹Istnieją bardziej skomplikowane reprezentacje, uwzględniające samoadaptujące się parametry, które określają operatory poszukiwania użyte przez algorytm, ale omówimy je w rozdziale 10.

Możliwość (ii). Przypuśćmy, że masz problem optymalizacji zestawu kątów obrotowych związku chemicznego. W tym wypadku możliwy zakres kątów to $[0, 2\pi)$ radianów, a zatem naturalną reprezentacją jest lista wartości zmienno-pozycyjnych ograniczona do $[0, 2\pi)$. Z kolei dla danej listy niezależnych zmiennych, które mają znaleźć się w modelu, oraz celu polegającego na znalezieniu wartości ciągłych parametrów tych zmiennych o takiej własności, że pewna funkcja błędu określona na wyniku działania modelu oraz zmiennej zależnej jest minimalna, można byłoby znowu wybrać reprezentację zmiennopozycyjną, przy czym tym razem wartości nie byłyby ograniczone i pochodziłyby z zakresu $(-\infty, +\infty)$.

Możliwość (iii). Łącząc pomysły z punktów (i) i (ii), przypuśćmy, że musisz znaleźć liniowy model przewidywania, który działa na pewnym zestawie przeszłych wartości zmiennej. Chcesz na przykład przewidzieć cenę akcji IBM na podstawie ich historii. Można byłoby zaproponować model w postaci

$$y[t] = a_1 y[t-1] + \dots + a_k y[t-k] \quad (7.1)$$

przy czym: $y[i]$ jest ciągiem cen akcji, t – czasem, k – liczbą wartości cen sprzed bieżącej wartości, które będą włączone do modelu, a_1, \dots, a_k są współczynnikami rzeczywistymi. W problemie tym należy dobrać optymalną liczbę k tak, żeby następne optymalne wartości a_1, \dots, a_k dały minimalny błąd w przewidywaniu $y[t]$. Możemy tutaj przyjąć złożoną reprezentację w postaci

$$[k, a_1, \dots, a_{max}]$$

przy czym max jest maksymalną liczbą wcześniejszych wartości, które będziesz rozważał. Pierwsza współrzędna wektora k określa, ile kolejnych współrzędnych będzie uwzględnianych w modelu z (7.1). Możesz tu zwrócić uwagę na sposób implementowania operatorów poszukiwania dla k różniący się od sposobu dla a_1, \dots, a_{max} .

7.1.2. Permutacje

Przypuśćmy, że masz rozwiązać problem ułożenia planu działania. Musisz uporządkować listę j zadań, które muszą zostać wykonane przez fabrykę. Każde z nich musi zostać wykonane, a po wykonaniu jest usuwane z dalszych rozważań. W takiej sytuacji permutacja zadań

$$[1, 2, \dots, j]$$

jest wygodną reprezentacją. Porządek w permutacji odpowiada porządkowi wykonywania zadań. Zauważ, że może być konieczne korzystanie z wielu przykładów pewnych zadań – każde z nich może być reprezentowane przez inną liczbę lub po prostu przez uwzględnienie którejś liczby kilka razy. Permutacje są także użyteczne w różnych odmianach TSP.

7.1.3. Automaty ze skończoną liczbą stanów

Powiedzmy, że masz rozwiązać problem przewidywania ciągu symboli pojawiających się z upływem czasu i musisz opracować procedurę umożliwiającą wykonywanie przydatnego przewidywania. Jedna z takich procedur może polegać na użyciu automatu ze skończoną liczbą stanów (np. automatu Mealy'ego), z których jeden jest zdefiniowany jako stan startowy, a dla każdego możliwego symbolu wejściowego oraz stanu istnieje związany z nimi symbol wyjściowy oraz przejście do następnego stanu. Dla danych alfabetów symboli wejściowych i wyjściowych, odpowiednio $A = \{a_1, \dots, a_m\}$ oraz $B = \{b_1, \dots, b_n\}$, wygodną reprezentacją takiego automatu jest

$$(O_{ij}, S_{ij}, s)$$

przy czym: O_{ij} i S_{ij} – odpowiednio macierze symboli wyjściowych oraz przejść do następnego stanu, łączące każdy symbol wejściowy i ze stanem j ; zaś s – stan początkowy, który przyjmuje wartości od 1 do maksymalnej liczby stanów. Zwróć uwagę, że reprezentację tę możemy rozszerzyć do

$$(O_{ij}, S_{ij}, s, k)$$

przy czym k oznacza liczbę stanów automatu; zatem k określa wymiar macierzy O_{ij} i S_{ij} , a także maksymalną możliwą wartość s . W tym wypadku reprezentacja może mieć zmienną długość.

7.1.4. Wyrażenia symboliczne

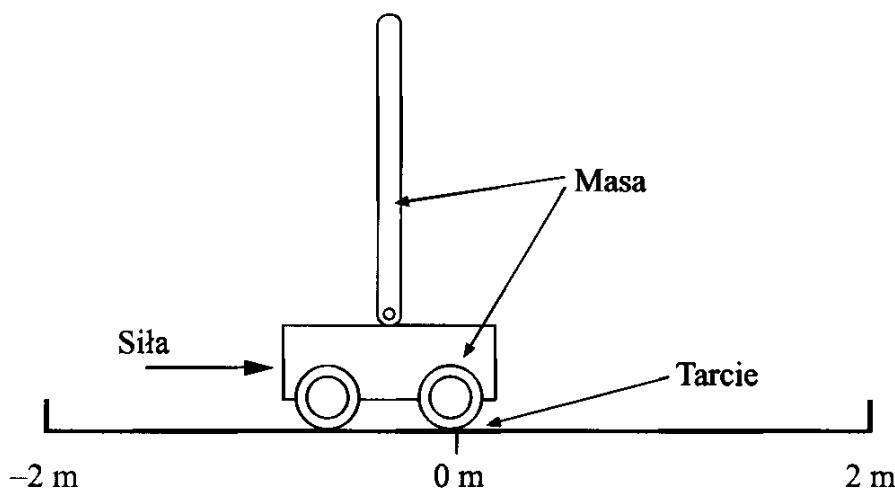
Przypuśćmy, że zostałeś postawiony wobec problemu stworzenia funkcji wykonującej konkretne zadanie odwzorowania, na przykład implementującego sekwencję sterującą umożliwiającą utrzymanie w równowadze miotły ustawionej na wózku do góry nogami (rys. 7.2) i działającą jedynie na zasadzie popchania wózka. Funkcja musi działać przy danym bieżącym stanie systemu $x, \dot{x}, \theta, \dot{\theta}$, przy czym widoczne zmienne oznaczają położenie i prędkość wózka oraz kija od miotły, wynikiem zaś jest siła, jaką należy przyłożyć do wózka. W takim wypadku możemy zastosować wyrażenie symboliczne zapisane w postaci jego drzewa składni. Drzewo to ma konkretną strukturę, łączącą funkcje i symbole terminalne. Funkcjami tutaj mogłyby być $+$, $-$, $*$, $/$, \sin , \cos itd., symbolami terminalnymi $x, \dot{x}, \theta, \dot{\theta}$ oraz r – zmienna rzeczywista mogąca przyjąć dowolną określoną wartość. Funkcje działają na innych funkcjach i symbolach terminalnych. Oto prosty przykład

$$(*(+ \sin(\dot{x}) 2,13)(\theta))$$

który można zapisać w postaci wyrażenia matematycznego

$$(\sin(\dot{x}) + 2,13) * \theta$$

Podobnie jak w przypadku maszyn ze skończoną liczbą stanów łatwo możemy sobie wyobrazić te struktury w postaci występujących w różnych konfiguracjach reprezentacjach drzewowych o różnych długościach.



Rysunek 7.2. Wózek z kijem od miotły przyczepionym na górze na zawiasach.

Celem jest takie popychanie i pociąganie wózka, żeby powrócił na środek obszaru, a przy okazji kij nie upadł i wózek nie zderzył się z żadną granicą obszaru

Powyższa lista nie jest dokładna i łatwo sobie wyobrazić sposoby łączenia różnych reprezentacji. Przypuśćmy na przykład, że mamy pewną liczbę celów T , do których będziemy strzelali z różnych broni W . Każda broń może w danym kroku wystrzelić tylko do pewnego podzbioru celów. Jeśli broń wystrzeli, to musi zostać przeładowana, co uniemożliwia korzystanie z niej przez pewien czas. W ten sposób reprezentacją dla strategii sterującej mogłaby być macierz złożona z $|W|$ wierszy, po jednym dla każdej broni, oraz z T kolumn, po jednej dla każdej chwili. Elementy w tej macierzy byłyby permutacjami wektorów wskazującymi kolejność priorytetów strzelania do dostępnych celów lub informującymi, żeby nie strzelać. Zauważmy, że wektory permutacji w macierzy miałyby różne długości odpowiadające liczbie celów dostępnych dla odpowiedniej broni. (Reprezentacja ta została użyta w [147]). Wszystkie powszechnie stosowane reprezentacje są w istocie wektorami o ustalonej lub zmiennej długości albo macierzami składającymi się z symboli. Interpretacja tych struktur może prowadzić do implementacji sieci neuronowych, sterowników rozmytych, automatów komórkowych lub innych struktur obliczeniowych.

7.2. Funkcja oceny

Funkcja oceny to nasz jedyny sposób ocenienia jakości rozwiązań¹ podlegających ewolucji. Relacje między ocenianiem rozwiązań oraz operatorami różnicowania w dużym stopniu określa efektywność całego poszukiwania. Staranne opracowanie odpowiednich funkcji oceny wymaga rozpatrzenia, jak najpraw-

¹Niektóre algorytmy ewolucyjne nie korzystają jednak z zewnętrznej funkcji oceny. Zamiast tego oceniają wzajemną wartość rozwiązań przez realizowaną parami rywalizację (polegającą na przykład na wykonywaniu rozgrywki w warcaby lub tryktraka). W takim *koewolucyjnym* modelu funkcja oceny jest wewnętrznym elementem samego procesu ewolucyjnego.

dopodobnie będzie się zachowywał algorytm. Jest jednak co najmniej jedna wskazówka, której należy się trzymać:

Optymalne rozwiązanie(a) powinny otrzymywać optymalną(e) ocenę(y).

Implementacje algorytmów ewolucyjnych dążą zwykle w granicy do optimum funkcji oceny. Wydaje się sensowne, żeby te optima globalne odpowiadały rozwiązaniom, których poszukujemy. Co więcej, chciałoby się mieć silną korelację między naszą subiektywną oceną zaproponowanego rozwiązania a oceną określoną za pomocą funkcji oceny.

Dla ilustracji zajmijmy się prostym problemem znajdowania ciągu dziesięciu symboli ze zbioru $\{0, 1\}$ o takiej własności, że suma symboli w wektorze jest maksymalna. Ewentualnie najlepszym rozwiązaniem jest ciąg [1111111111], a funkcja oceny, która natychmiast przychodzi na myśl, to

$$f(\mathbf{x}) = \sum_{i=1}^n x_i$$

przy czym im więcej jest jedynek w wektorze, tym jest on „bliższy” rozwiązaniu doskonałemu i tym większa obliczona jakość. Dla kontrastu przypatrzmy się funkcji

$$f(\mathbf{x}) = \prod_{i=1}^n x_i$$

Tutaj rozwiązanie dające globalne optimum daje również największą możliwą wartość oceny, ale wszystkie rozwiązania różne od [1111111111] otrzymują ocenę zero. Nie ma tutaj żadnej korelacji między postrzeganą jakością rozwiązań niedoskonałych a ich obliczonym „przystosowaniem”. Korzystanie z pierwszej funkcji oceny umożliwia prowadzenie poszukiwania w kierunku lepszych rozwiązań, korzystanie z drugiej przypomina poszukiwanie igły w stogu siana – zadanie możemy rozwiązać tylko wówczas, gdy znajdziemy prawidłową odpowiedź.

W praktyce rzadko możemy znaleźć doskonałe rozwiązanie problemu ze świata rzeczywistego. Nie mamy czasu, żeby pozwolić algorytmowi ewolucyjnemu działać wystarczająco długo. Musisz zająć się celem polegającym na tym, aby uzyskać rozwiązanie wysokiej jakości w czasie na tyle krótkim, żeby jeszcze było użyteczne, a nie upierać się przy doskonałości. Oznacza to, że konieczne jest, żeby funkcja oceny dawała informacje, które mogą poprowadzić poszukiwania przez rozwiązania, które są gorsze od idealnych. Funkcja oceny biorąca za wzór zasadę „wszystko albo nic” nie jest praktyczna.

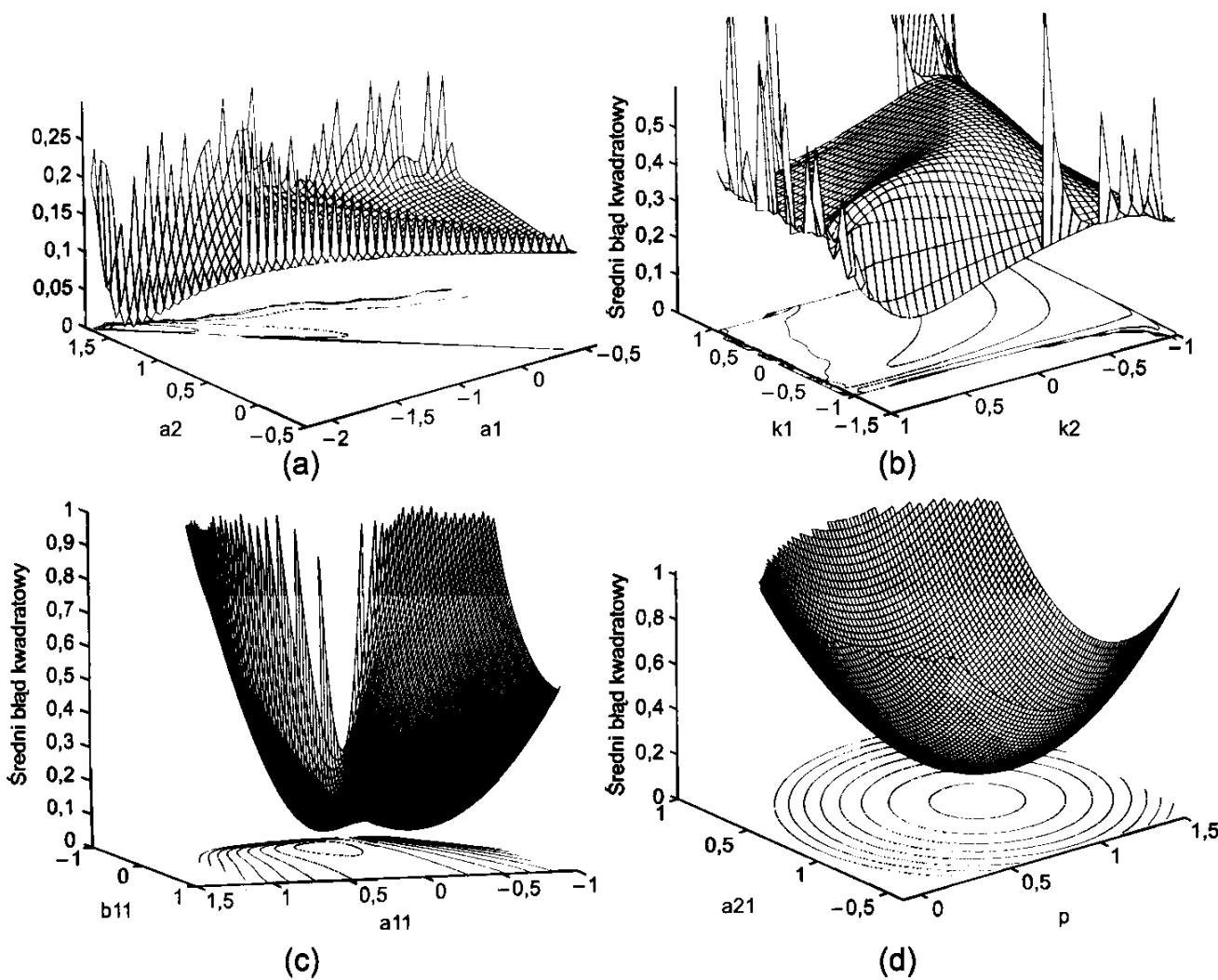
Opracowanie funkcji oceny, które umożliwiają szybką optymalizację ewolucyjną, wymaga obsługi współdziałania operatorów różnicowania z funkcją oceny. Chociaż zdarzają się wyjątki, często pożądane jest, żeby małe zmiany w potencjalnych rozwiązań dawały małe zmiany w ocenie powstałego potomstwa. Takie rozwiązanie zapewnia zwykle, że możemy wykonywać co najmniej lokalne poszukiwanie statystyczne. Przydatne też może być oddzielenie w ramach funkcji oceny różnych niezależnych czynników wpływających na

ogólną jakość. W takiej sytuacji można opracować specjalne operatory różnicowania (np. rekombinacji), które mieszają lub łączą użyteczne części pełnych rozwiązań.

Następnie musisz rozważyć w świetle funkcji oceny współgranie reprezentacji z operatorami poszukiwania. Przypuśćmy bowiem, że chcesz opracować filtr cyfrowy o skończonej odpowiedzi impulsowej, który mógłby być użyteczny przy przewidywaniu sygnału wyjściowego dla danej funkcji przenoszenia i danego sygnału wejściowego [70]. Naturalna funkcja oceny prowadziłaby do minimalizacji sumy kwadratów błędów między przewidzianym a rzeczywistym wynikiem funkcji przenoszenia dla pewnego zestawu sygnałów wejściowych. Umożliwiłoby to coraz ścisłejsze zbliżanie się do filtra o najlepszych odpowiedziach, w miarę jak zmniejszałby się kwadrat błędu. „Powierzchnia” tej funkcji oceny zależy od reprezentacji. Istnieje wiele struktur danych zaproponowanych do reprezentacji tego rodzaju filtrów. Na rysunku 7.3 przedstawiono kształty funkcji oceny, działającej na podstawie kwadratów błędów, dla każdej z tych reprezentacji. Uwidoczniono fragmenty wykresów w pobliżu optymalnego filtra dla problemu rozważanego w [70]. W zależności od tego, jaką decyzję podejmiesz, wybierając jedno rozwiązanie, niektóre z tych powierzchni będą bardziej „przyjazne” dla prowadzonej ewolucji, a inne mniej.

W praktyce ocenianie jakości rozwiązań w populacji jest najkosztowniejszą czasowo operacją przy wykonywaniu algorytmu ewolucyjnego. Widać to szczególnie, gdy ocena musi być przeprowadzona na dużej próbie rozwiązań (np. przy problemie rozpoznawania wzorców) lub gdy do ustalenia efektywności określonej strategii w określonym czasie należy wykonać symulację (np. przy podejmowaniu decyzji w warunkach bojowych). Proces ten możemy czasami przyspieszyć, stosując niepełną ocenę, w której nie określono dokładnej wartości oceny, ale oszacowano ją lub obliczono wartość względową.

Przypuśćmy na przykład, że Twoja procedura ma jako rodziców następnego pokolenia wykorzystywać najlepszych 10 rozwiązań w populacji 100 kandydatów. Po ocenieniu pierwszych 10 rozwiązań musisz kontynuować ocenę dowolnego następnego rozwiązania tylko wówczas, gdy potencjalnie może ono dać wartość lepszą niż najgorsze spośród tych 10. Jeśli możesz szybko stwierdzić, że tak nie jest, to nie musisz kończyć oceny. Powiedzmy, że masz do czynienia z problemem rozpoznawania wzorców z tysiącem przykładów treningowych, a bieżące rozwiązanie z 10. miejsca klasyfikuje poprawnie 950 przypadków, gdzie funkcja oceny to liczba poprawnych rozpoznań. Gdy dowolne kolejne rozwiązanie źle zaklasyfikuje 51. raz, to nie ma znaczenia, ile jeszcze przypadków zaklasyfikuje prawidłowo – nie będzie wpływać na następne pokolenie. Możesz przejść w populacji do następnego kandydata. Zauważ jednak, że w metodach selekcji, w których istnieje niezerowe prawdopodobieństwo przejścia do następnej generacji dla osobników o małej wartości, możesz dokończyć ocenę, żeby określić, jak dobre albo jak źle jest dane rozwiązanie.



Rysunek 7.3. Powierzchnie funkcji średniego błędu kwadratowego wokół optymalnego rozwiązania dla postaci: (a) bezpośredniej; (b) kratowej; (c) kaskadowej; (d) modelu równoległego filtru o trzech zerach i trzech biegunach [70]. Można pokazać, że każda postać filtru jest równoważna pozostałym, a jak widać, powierzchnie funkcji błędu są dla nich znaczowo różne. Niektóre z nich poddają się ewolucji lepiej od innych

Inna możliwość pójścia na skróty przy zastanawianiu się nad funkcją oceny polega na wykonywaniu porównywania tylko dwóch kandydatów na rozwiązania i określaniu, które z nich jest lepsze. Możesz nie dowiedzieć się, o ile jedno rozwiązanie jest lepsze od drugiego, a tylko które jest lepsze. Zastanówmy się nad problemem zaprojektowania sterownika dla niestabilnego systemu symulującego. Jedną z możliwych funkcji oceny jest liczba kroków czasu symulacji, dla których sterowanie może zapewnić, że system nie wyjdzie poza zakres tolerancji. Wymaga to uruchomienia każdego kandydata na sterownik do momentu, aż ten zawiedzie. Z kolei, gdybyś wykonał ciąg porównań sterowników parami, to musiałbyś wykonać symulację tylko do momentu, gdy jeden ze sterowników zawiedzie i wybrać w tym momencie ten drugi. Procedurę tę należy powtarzać do momentu zapełnienia następnej populacji. Nie musisz wiedzieć, jak dobry jest dowolny konkretny sterownik, wystarczy, że wiesz, iż jest lepszy niż jego przeciwnik, który zawiódł.

7.3. Operatory różnicowania

Praktyczna implementacja operatorów różnicowania zależy od wybranej reprezentacji oraz Twojej intuicji co do „krajobrazu” określonego przez funkcję oceny przy wybranej reprezentacji i operatorach różnicowania. Każdy taki krajobraz wymaga określenia metryki odległości, a tutaj możemy zdefiniować ją jako liczbę użyć operatorów różnicowania potrzebną do przejścia od jednego punktu (lub wielu punktów) do innego lub jako prawdopodobieństwo przejścia między nimi. Chociaż jesteśmy przyzwyczajeni do myślenia o krajobrazach jako o trójwymiarowych pagórkach i dolinach o euklidesowej metryce, taka metafora jest uzasadniona jedynie w celu zrozumienia, jak działa poszukiwanie ewolucyjne przy reprezentacjach ciągłych i ciągłych operatorach różnicowania (np. zakłóceniach Gaussa). Wyobrażenie sobie krajobrazu dla problemu optymalizacji kombinatorycznej takiego jak TSP może być pewnym wyzwaniem.

Na ogół wybór operatora różnicowania powinien wynikać z wybranej przez Ciebie reprezentacji. Możesz rozważyć różne operatory dla jednego lub dwóch rodziców bądź dla więcej niż dwóch. Nie istnieje żaden dobry wybór dla wszystkich problemów, musisz więc uważnie rozważyć implikacje wszystkich dokonywanych wyborów i próbować określić ich wpływ na poszukiwanie coraz lepszych rozwiązań mierzonych za pomocą obranej funkcji oceny. Rzućmy okiem na kilka typowych operatorów różnicowania w świetle opisanych wcześniej reprezentacji.

7.3.1. Wektory symboli o ustalonej długości

Przy wykorzystywaniu reprezentacji binarnej często jako istotny jest wybierany operator, w którym z określonym prawdopodobieństwem poszczególne bity są przełączane z 0 na 1 i na odwrót. Często operator ten jest stosowany do każdego bitu, dzięki czemu jest możliwe przejście od dowolnego ciągu do dowolnego innego, ale jest to mało prawdopodobne. W istocie ten operator różnicowania oznacza, że mamy do czynienia z krajobrazem opartym na odległości Hamminga. Im więcej bitów musi zostać przełączone, żeby przekształcić jeden ciąg w drugi, tym dalej te ciągi są od siebie. Chociaż w wielu zastosowaniach korzystających z ciągów binarnych używamy stałego prawdopodobieństwa przełączenia bitów, większość nietrywialnych problemów przy optymalnym poszukiwaniu wymaga zmiennego prawdopodobieństwa (tzn. prawdopodobieństwo przełączenia bitu musi się zmieniać odpowiednio w ciągu trwania ewolucji z uwzględnieniem bieżącej populacji). Określenie możliwie najlepszego sposobu zmieniania tych prawdopodobieństw jest złożonym problemem, który rozwiązano tylko dla prostych przypadków.

Inny, często w tym wypadku stosowany operator działa przez połączenie dwóch lub więcej rozwiązań za pomocą krzyżowania. Polega to na tym, że układają się obok siebie kilka rozwiązań, a potem wybiera ich kawałki (lub poszczególne bity) z jednego rozwiązania bądź drugiego, tworząc nowe rozwiązanie, które będzie podlegać testom. Te operatory możemy stosować do dowolnego ciągu symboli o ustalonej długości, należy jednak pamiętać, że użyteczność wyniku zależy od konkretnego problemu. Warto też zwrócić uwagę, że różne operato-

ry krzyżowania prowadzą do różnych krajobrazów. Liczba punktów krzyżowania, które należy wykorzystać, żeby z dwóch lub więcej rozwiązań zbudować nowe, zależy od konkretnego operatora, a krajobraz z niego uzyskany raczej odbiega od krajobrazu wynikającego z opisanego wyżej operatora przełączania bitów.

Jeśli symbolami w ciągu są zmienne ciągłe, to często stosowana procedura polega na zakłócaniu symboli rozwiązania przez dodanie zmiennej losowej Gaussa o średniej zero i określonym odchyleniu standardowym. Zgodna z intuicją metryka to tutaj euklidesowa odległość między dwoma punktami przestrzeni poszukiwań, odchylenie standardowe wyznacza prawdopodobieństwo wykonania dużego lub małego kroku w tym krajobrazie. Inne typowe typy zakłócania jednego rozwiązania polegają na dodaniu do niego zmiennej Cauchy'ego lub zmiennej jednostajnej. Użycie zmiennej Cauchy'ego daje dłuższe kroki niż w przypadku zmiennych Gaussa, zmienne jednostajne nie powodują żadnego przechylenia generowanych rozwiązań w kierunku otoczenia rozwiązania-rodzica (w obszarze stosowania jednostajnej funkcji gęstości). Inny często stosowany operator różnicowania zmiennych ciągłych działa przez mieszanie wartości współrzędnych różnych rozwiązań dla każdej współrzędnej, biorąc średnią (ważoną) odpowiednich współrzędnych rodziców. Ten operator arytmetyczny przypomina branie „środka masy” dla wielu rozwiązań i może niekiedy przyspieszyć zbieżność do lokalnego optimum (takie przyspieszone zbieganie może być dobre i złe w zależności od jakości odkrytego optimum). Istnieje tutaj jeszcze wiele innych możliwych rodzajów rekombinacji, które możemy rozważać (np. krzyżowanie geometryczne zamiast arytmetycznego).

Przy poszukiwaniu w zbiorze liczb całkowitych często użyteczne jest zmienianie wartości symbolu przez jego zwiększenie lub zmniejszenie z tym samym prawdopodobieństwem, ale o wartość zgodną ze zmienią losową Poissona o parametrze λ oznaczającym średnią. Na przykład przy poddawaniu ewolucji liczby odstępów w modelu autoregresywnym, możemy z tym samym prawdopodobieństwem zwiększyć „rząd modelu” (do pewnej maksymalnej wartości wynikającej z ograniczeń pamięciowych) lub zmniejszyć (do zerowej liczby odstępów), dodając lub odejmując wartość zgodnie z rozkładem Poissona. Jeśli liczby całkowite mogą przybierać wartości ze skończonego zakresu, a zwłaszcza ze skończonego małego zakresu, to często wygodnie jest używać również jednostajnego próbkowania możliwości lub próbkowania zgodnie z rozkładem dwumianowym (wykorzystywany często jako dyskretny odpowiednik rozkładu Gaussa).

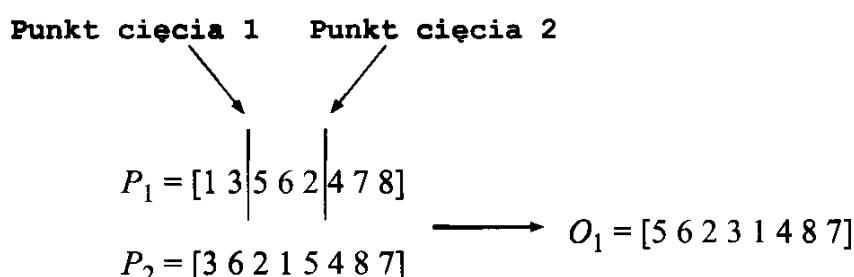
7.3.2. Permutacje

Stosowanie permutacji często wymaga wymyślenia sprytnych operatorów różnicowania, które korzystają z wiedzy na temat interpretacji permutacji. Rozważmy na przykład problem wyboru operatora różnicowania, gdy permutacja reprezentuje trasę w TSP. Jedna z możliwości polega na usunięciu losowo wybranego jednego miasta i wstawieniu go w losowe miejsce listy. Inna możliwość polega na wybraniu dwóch pozycji na liście i odwróceniu kolejności miast między tymi punktami (podobnie jak w operatorze zamiany dwukrawędziowej

[284]). W wypadku TSP operator odwracania okazuje się lepszym wyborem, gdyż wykorzystuje własności związane z odległościami euklidesowymi między wierzchołkami czworoboku umieszczonego w kartezjańskim układzie współrzędnych. Jak widzieliśmy w rozdziale 6, dla danego czworoboku suma odległości po przekątnych jest zawsze większa niż suma długości każdej pary przeciwnie skierowanych boków (rys. 6.2). Wynika z tego, że każda trasa, która krzyżuje się ze sobą, jest nieoptimalna. Zastosowanie odwracania dla wierzchołków za takim skrzyżowaniem usuwa związaną z nim niedoskonałość. Tego rodzaju korzyść nie pojawia się przy zastosowaniu operatora usuwającego i wstawiającego i nie uzyskamy nic w TSP po wybraniu dwóch pozycji i zamianie ich miejscami (bez odwracania całego segmentu między nimi). Zauważ jednak, że w innych problemach tego rodzaju operatory mogą być bardziej odpowiednie niż odwracanie – wszystko zależy od zadania. Podobnie jak możemy sobie wyobrazić użycie procedury zamiany dwukrawędziowej jako operatora różnicowania, możemy sobie wyobrazić zastosowanie operatora k -krawędziowego, wymaga jednak on dodatkowego czasu przetwarzania.

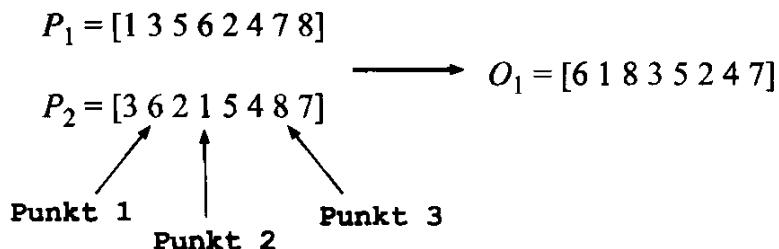
Używanie dwóch lub więcej permutacji do generowania permutacji potomnej wymaga ostrożnego planowania. Użycie prostych operacji polegających na cięciu i łączeniu wprowadza ryzyko wygenerowania listy z wielokrotnymi wystąpieniami pewnych elementów i bez wystąpień innych. W literaturze istnieje wiele różnych metod rekombinacji dla permutacji. Wspomnijmy tutaj dwie, które są oparte na zachowaniu kolejności pojawiania się pewnych pozycji w każdym z rodziców.

W pierwszej z nich bierzemy dwie permutacje P_1 i P_2 , a następnie obieramy losowo dwa punkty cięcia w P_1 . Kopiujemy pozycje między punktami cięcia w P_1 do O_1 . Sprawdzamy, które pozycje w P_2 nie są uwzględnione w O_1 , a następnie wstawiamy je kolejno, począwszy od drugiego punktu cięcia (rys. 7.4) [85].



Rysunek 7.4. Jeden z możliwych operatorów wykonujących rekombinację dla dwóch permutacji. Elementy między dwoma punktami cięcia w pierwszej permutacji są kopowane do potomka. Pozostałe elementy potomka są kopowane kolejno z drugiej permutacji

Porównaj to z taką procedurą: Weź dwie permutacje P_1 i P_2 i wybierz losowo n punktów z P_2 . Znajdź w P_1 pozycje o takich samych wartościach dla wszystkich n pozycji z P_2 . Zmień ich kolejność w P_1 zgodnie z ich kolejnością w P_2 . Bezpośrednio przekopiuj pozostałe pozycje z P_1 (rys. 7.5) [462].



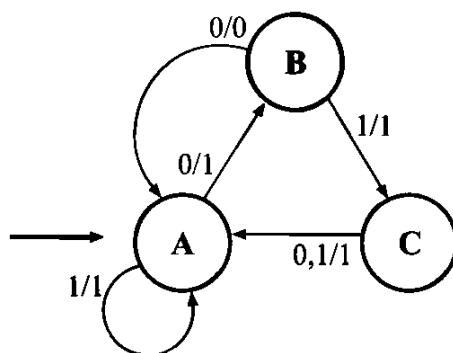
Rysunek 7.5. Inny możliwy operator wykonujący rekombinację dla permutacji.

Wybieramy losowo trzy punkty w drugiej permutacji. Elementy z tych pozycji są kopiowane do potomka. Pozostałe elementy są kopiowane po kolej z pierwszej permutacji

Ta druga procedura nie gwarantuje, że jakikolwiek ciągły odcinek dowolnego rodzica zostanie przekazany do potomka bez naruszenia, nie ma też jednak konieczności zachowywania ciągłego fragmentu któregokolwiek z rodziców. Jak w wypadku operatorów na pojedynczych rodzicach musisz, żeby stwierdzić, czy dany operator korzystający z wielu rodziców jest adekwatny do rozważanego problemu, określić wpływ jego zastosowania.

7.3.3. Automaty ze skończoną liczbą stanów

Jeśli pracujesz z automatami ze skończoną liczbą stanów o zdefiniowanych dla wszystkich wejść we wszystkich stanach symbolach wyjściowych i stanach wynikowych oraz jeśli wybrałeś zaproponowaną powyżej reprezentację macierzową, to zastosowanie operatorów różnicowania jest dość naturalne. Istnieje pięć trybów różnicowania konkretnego automatu ze skończoną liczbą stanów: 1) dodanie stanu, 2) usunięcie stanu, 3) zmiana stanu początkowego, 4) zmiana symbolu wyjściowego, 5) zmiana przejścia do następnego stanu (rys. 7.6). Sprawy stają się tutaj trochę bardziej skomplikowane, gdyż musisz uważać na zmieniający się rozmiar i strukturę automatu. Operatory różnicowania mają tutaj pewne ograniczenia: nie możemy dodać więcej stanów niż określona wartość maksymalna (ograniczona ilością pamięci lub czasu) i nie możemy usunąć ani zmienić stanu, gdy automat ma tylko jeden stan. Różnicowanie, które tutaj możemy zastosować, zależy od rozwiązania. Co więcej, istnieje wiele różnych sposobów implementacji tych operatorów. Na przykład przy dodawaniu stanu musisz zdecydować, czy pozwolić, żeby z któregoś z istniejących stanów można było do niego przejść. Przy usuwaniu stanu musisz określić pewne reguły przedstawiania połączeń, które do tej pory prowadziły do usuwanego stanu. Nawet przy wykonywaniu tak prostej czynności jak zmienianie symbolu wyjściowego możesz wprowadzić losowy rozkład wyboru spośród wszystkich możliwych zmian. Zamiast wybierania nowych symboli w sposób jednostajny możesz wprowadzić pojęcie bliskości między znaczeniem dostępnymi symboli i dzięki niemu zapewnić, że symbole bliższe staremu mają większe prawdopodobieństwo wybrania. Musisz też zdecydować, ile operacji różnicowania wykonać; liczba ta może być wyznaczana za pomocą zmiennej losowej Poissona o dostrajanej wartości parametru.



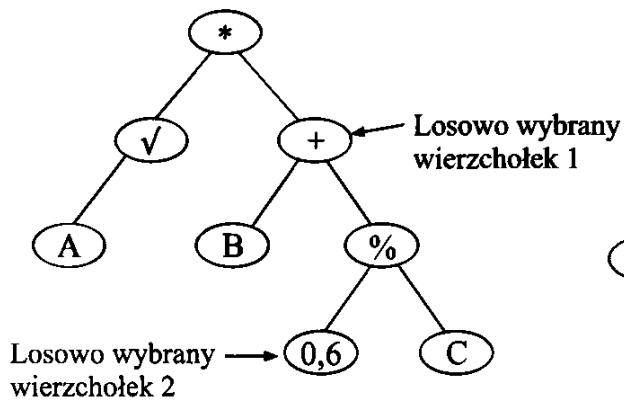
Rysunek 7.6. Automat ze skończoną liczbą stanów o trzech stanach. Automat zaczyna w stanie A. Dla każdego symbolu wejściowego (0, 1) każdy stan odpowiada symbolowi wyjściowemu (0, 1) oraz przechodzi do następnego stanu. Symbole wejściowe są wskazane na lewo od ukośnika, symbole wyjściowe na prawo. Zauważ, że alfabet wejściowy i wyjściowy nie muszą być równoważne. Naturalne tryby różnicowania automatu ze skończoną liczbą stanów wynikają z diagramu

Do automatów ze skończoną liczbą stanów możemy też stosować operatory rekombinacji, znów jednak musisz myśleć w kategoriach funkcjonalności automatów wynikowych. Jeśli wezmiesz z jednego automatu trzy stany, które ewoluowały w ten sposób, żeby działać w jego kontekście, i połączysz je z trzema stanami z innego automatu o odmiennej architekturze, to jest mało prawdopodobne, żeby nowy automat działał podobnie do któregokolwiek z rodziców. Przypomina to wzięcie pierwszej połowy z *Hamleta* i połączenie jej z drugą połową *Króla Lira*. Funkcjonalnie jest to „makromutacja”, czasami jednak użycie czegoś, co sprowadza się do wykonywania dużych skoków w przestrzeni możliwych rozwiązań, może dać pożądanego efekt (przynajmniej do pewnego stopnia). Inny pomysł mógłby polegać na wykonywaniu na wielu automatach (więcej niż dwóch) operacji logicznej brania większości. Dla każdego wejścia w każdym układzie stanów w każdym rozważanym automacie są zapisywane wyjścia, a jako wyjście automatu wynikowego bierze się wyjście, które zdobyło większość głosów przy tych zapisach. W ten sposób implementowane jest demokratyczne decydowanie wśród rodziców. Wadą tego podejścia jest to, że rozmiar powstającego potomka jest iloczynem rozmiarów rodziców, a taka wartość może bardzo szybko wzrastać.

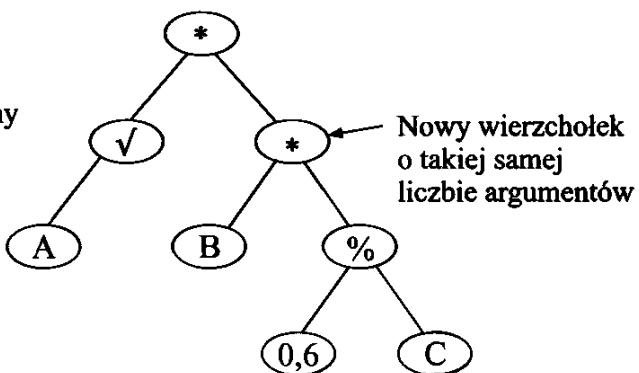
7.3.4. Wyrażenia symboliczne

W przypadku wyrażeń symbolicznych przedstawionych za pomocą drzewa analizy syntaktycznej, podobnie jak w przypadku automatów ze skończoną liczbą stanów, jesteśmy w sytuacji, gdy musimy zastosować strukturę o zmiennej długości. Możemy myśleć o powiększaniu lub skracaniu drzewa przez losowe rozszerzanie albo przycinanie gałęzi. Możemy sterować kolejnością wykonywanych działań przez zamianianie wierzchołków na gałęzi lub przez zamianianie kolejności w całych gałęziach (rys. 7.7). Możemy też modyfikować elementy w poszczególnych wierzchołkach, zmieniając powiedzmy + na *.

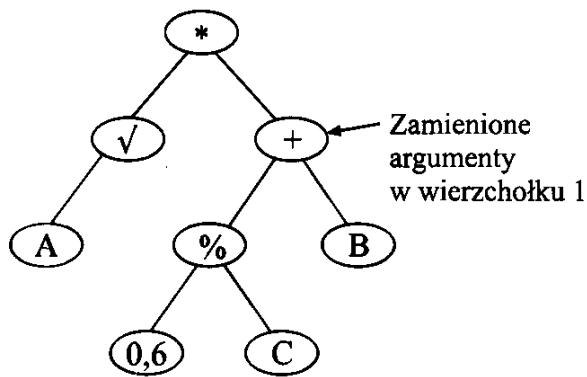
(a) Pierwotny program w postaci drzewa analizy składniowej



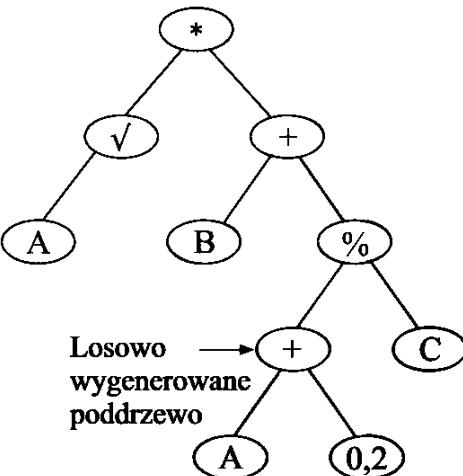
(b) Mutacja jednowierzchołkowa w wierzchołku 1



(c) Mutacja zamieniająca w wierzchołku 2

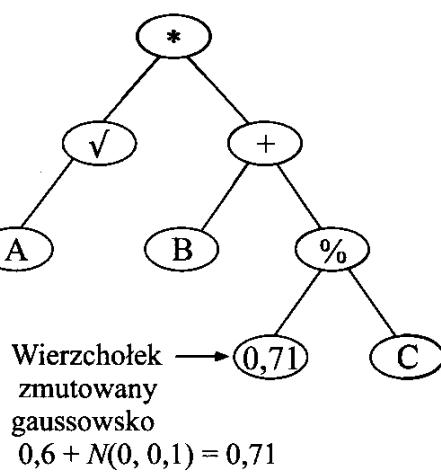
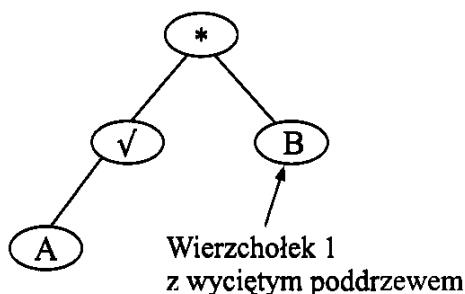


(d) Mutacja powiększająca w wierzchołku 2



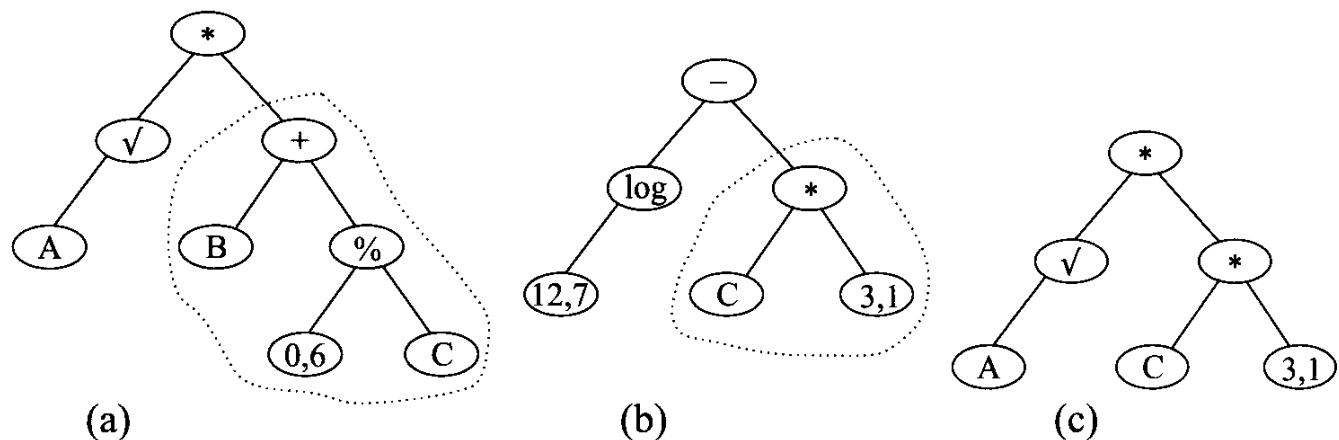
(f) Mutacja Gaussa

(e) Mutacja obcinająca w wierzchołku 1



Rysunek 7.7. Przykłady przedstawiające sześć operatorów różnicowania stosowanych przy reprezentacji drzewowej. Pierwotne wyrażenie to $\sqrt{A}*(B+0,6/C)$: (a) pierwotne drzewo; (b) różnicowanie w jednym wierzchołku; (c) zamiana wierzchołków; (d) powiększanie poddrzewa; (e) obcinanie poddrzewa; (f) różnicowanie Gaussa (stosowane do liczb rzeczywistych)

Podobnie jak w przypadku automatów ze skończoną liczbą stanów, możemy wymieniać elementy lub gałęzie między dwoma lub więcej drzewami. Przy zamianianiu gałęzi, jeśli każda z nich wykonuje podprogram, który jest ważny dla rozwiązania większego problemu, tego rodzaju rekombinacja może być pomocna. Może to być doskonały sposób na zgromadzenie użytecznych modułów pochodzących z różnych drzew (rys. 7.8). Musimy być uważni i nie myśleć, że tylko z tego powodu, że jakaś gałąź jest użyteczna w jednym drzewie, będzie przydatna w innym. Użyteczność podprocedury lub dowolnego modułu zawsze zależy od zintegrowanego skomplikowanego systemu, w którym funkcjonuje. Wyobraźmy sobie wymienianie płyt głównych między dowolnie wybranymi komputerami osobistymi. Dwa komputery mogą działać doskonale przed zamianą, ale po jej wykonaniu może się zdarzyć, że żaden nie będzie funkcjonował. Nawet zamiana poszczególnych rezystorów w dwóch różnych komputerach może dać katastrofalny wynik. Istnieją eksperymentalne dowody [267], że takie zamianianie gałęzi między (zwykle dwoma) drzewami wyrażeń symbolicznych może dać pozytywny wynik. Ponownie efektywność tej operacji zależy od problemu.



Rysunek 7.8. Wymienianie gałęzi między dwoma drzewami; wymieniane obszary są zaznaczone *przerywanymi liniami*: (a) pierwsze drzewo; (b) drugie drzewo; (c) jedno z drzew wynikowych

7.4. Selekcja

Operator selekcji działa na osobnikach z bieżącej populacji. Na podstawie ich jakości określa następną populację. Selekcja zwykle przybiera dwie postaci: 1) pewne osobniki są eliminowane z rozważań, a pozostałe osobniki przeżywają i stają się rodzicami kolejnych pokoleń lub 2) osobniki są wybierane losowo, zastępowanie odbywa się na podstawie ich wzajemnej jakości. Przy pierwszej postaci każdy osobnik do kolejnego pokolenia może dokładać najwyżej jedną swoją kopię; w drugiej postaci każdy osobnik może być wybrany wiele razy. Chociaż istnieje wiele sposobów określania punktu stosowania selekcji w algorytmie ewolucyjnym, główna koncepcja pozostaje taka sama: selekcja stanowi filtr, za pomocą którego jest określany skład kolejnego pokolenia. Dla wy-

gody przeanalizujemy teraz różne procedury selekcji przy założeniu, że każda z nich będzie stosowana po wygenerowaniu wszystkich nowych potomków i po przypisaniu im miary jakości na podstawie funkcji oceny.

Metody selekcji możemy w szerokim sensie zaklasyfikować albo jako deterministyczne, albo jako stochastyczne. Dla danej populacji (rodziców i potomków) selekcja deterministyczna zawsze wyeliminuje te same osobniki, metody stochastyczne generują funkcję masy prawdopodobieństwa określoną na możliwych składach następnej populacji. Selekcja deterministyczna ma tę zaletę, że często jej implementacja jest szybsza niż implementacja selekcji stochastycznej (lub probabilistycznej), a w powszechnie stosowanych implementacjach selekcji deterministycznej również ma miejsce szybsza zbieżność populacji, ale może to być zarówno zaletą, jak i wadą w zależności od tego, do czego ta zbieżność zachodzi!

Przy powszechnie stosowanej w algorytmach ewolucyjnych notacji μ oznacza liczbę rodziców, λ – liczbę potomków. Wygląda to trochę jak szyfr, ale żeby pozostać w zgodzie z ustaleniami w literaturze, będziemy stosować właśnie te symbole. Był może, najprostsza postać selekcji jest oznaczana $(\mu + \lambda)$, przy czym λ potomków jest tworzone z μ rodziców, wszystkie $\mu + \lambda$ osobników jest oceniane, a następnie jest stosowana selekcja, żeby zachować jedynie najlepsze μ spośród tak uzyskanego zbioru. Zauważ, że λ potomków może być uzyskane za pomocą dowolnych operatorów różnicowania i, co więcej, nie wszystkie μ rodziców musi brać udział w generowaniu potomstwa. Inna często pojawiająca się metoda selekcji deterministycznej jest określana jako (μ, λ) , przy czym selekcja wybiera μ najlepszych rozwiązań, ale tylko spośród λ potomków (oczywiście $\lambda \geq \mu$). Może wydawać się, że odrzucanie rodziców z poprzedniego pokolenia jest niepożądane, takie rozwiązanie zmniejsza jednak szansę na stagnację przy nieoptimalnym rozwiązaniu.

W przeciwnieństwie do selekcji deterministycznej istnieją w zasadzie trzy różne metody selekcji stochastycznej (tzn. istnieje wiele innych, ale większość z nich może być przedstawiona jako pewien wariant jednej z tych trzech metod). Jedną z metod, która była stosowana już we wczesnych latach historii algorytmów ewolucyjnych, choć teraz jest darzona mniejszą uwagą, nazywamy selekcją proporcjonalną. W metodzie tej dla każdego osobnika musi być obliczona dodatnia wartość jakości, a następnie liczba kopii każdego osobnika przechodzących do następnego pokolenia jest proporcjonalna do jego relatywnej przydatności. Oznacza to, że przy założeniu, że w następnej populacji ma być μ rodziców, z bieżącej populacji jest pobierane μ próbek, przy czym prawdopodobieństwo wybrania i -tego osobnika

$$P_i = \frac{F_i}{\bar{F}}$$

gdzie: F_i jest „przystosowaniem” i -tego osobnika, a \bar{F} jest średnią ze wszystkich osobników w bieżącej populacji. Osobniki o przystosowaniu większym od średniej będą średnio obdarzane większą uwagą niż te o przystosowaniu mniejszym od średniej. Takie rozwiązanie przesuwa (obciąża) działanie algorytmu ewolucyjnego tak, żeby przy dalszych poszukiwaniach używało lepszych rozwiązań, ale

jednocześnie nie umożliwiał utrzymywania wyłącznie najlepszego rozwiązania. W rzeczywistości przy tej metodzie jest możliwe, że w wyniku losowego próbkowania przetrwają tylko najgorsze rozwiązania, ale takie zdarzenie jest bardzo mało prawdopodobne.

Dwie pozostałe metody selekcji stochastycznej są oparte na losowych turniejach. W pierwszej metodzie z bieżącej populacji jest wyciągana losowa próba osobników, jest z niej wybierany najlepszy, a pozostałe osobniki wracają do pierwotnej populacji, żeby można je było wykorzystać w następnych próbkach. Proces ten jest powtarzany dopóty, dopóki nie zostanie określona odpowiednia liczba rodziców dla następnej generacji. Rozmiar próbki określa zmienność działania procedury. Łatwo możemy zaobserwować, że jeśli rozmiar próbki będzie równy rozmiarowi populacji, to metoda ta jest równoważna z selekcją deterministyczną ($\mu + \lambda$).

W drugiej metodzie turniejowej każdemu osobnikowi jest przypisywany wynik zastępczy oparty na porównaniu go z losowym podzbiorem populacji. Przy każdym porównaniu wartość wyniku rozwiązania jest zwiększa o jeden punkt, jeśli jest co najmniej tak dobry, jak jego przeciwnik. Najlepsze rozwiązanie w populacji będzie zawsze otrzymywać punkty przy każdym porównaniu, ale osobniki o mniejszych wartościach też mogą zarabiać punkty. Nawet rozwiązania o małych wynikach mogą mieć szczęście i zmierzyć się ze słabszymi osobnikami, zarabiając w ten sposób dodatkowe punkty. W podzbiorze rozmiaru q maksymalna liczba punktów, jakie możemy zarobić, wynosi q . Następnie zaczyna działać deterministyczna procedura selekcji, w której jest brana pod uwagę przypisana każdemu osobnikowi liczba punktów. W pierwszej metodzie turniejowej pewne osobniki mogą być w ogóle pominięte, w tej drugiej natomiast jest wykonywana probabilistyczna ocena każdego z nich. I znowu, jak łatwo możemy sobie wyobrazić, jeśli rośnie rozmiar q podzbioru, to ta metoda turniejowa staje się równoważna z selekcją deterministyczną typu ($\mu + \lambda$).

Czasami wygodnie jest porównywać te różne metody pod względem ich nacisku selekcyjnego. Może to być zmierzone na wiele sposobów: jako średnie zmniejszanie się wariancji populacji, zmiana średniego przystosowania populacji w stosunku do jej wariancji lub też jako średni czas potrzebny najlepszemu rozwiązaniu na przejęcie całej populacji. Niestety bardzo trudno jest analizować operatory selekcji ze względu na te kryteria przy połączeniu z losowym różnicowaniem. W związku z tym większość analiz dotyczy wyłącznie powtarzania operacji selekcji przy zwykłym kopiowaniu wybranych rozwiązań bez jakichkolwiek ich modyfikacji. Bäck w [20] pokazał, że najsurowsze są procedury deterministyczne ($\mu + \lambda$) i (μ, λ). Najsłabszą (tzn. najwolniej zbieżną) metodą spośród tych, które badał, okazała się selekcja proporcjonalna. Selekcję turniejową możemy przez zmianę liczby rund dostroić tak, żeby była silna lub słaba (tzn. więcej rund prowadzi do silniejszego dobierania).

Terminu „selekcja” możemy używać w dwóch różnych kontekstach. Może się on odnosić tak do procesu wyboru rodziców, jak i do procesu wybierania osobników do zastąpienia. Metody selekcji zwykle definiują oba te aspekty, ale nie zawsze tak jest. Na przykład w algorytmie ewolucyjnym *stanu stabilnego* w każdym pokoleniu tylko kilku (w skrajnym wypadku tylko jeden) potomków

jest tworzonych przed selekcją. W wyniku tego populacja przy każdej selekcji przechodzi mniejszą zmianę. W tym miejscu musimy zająć się dwoma aspektami selekcji. Po wybraniu rodzica(ów) i stworzeniu potomka musimy wybrać osoby, które należy odrzucić z populacji. Tworzy to miejsce na nowego potomka, przy założeniu, że nie zdecydowaliśmy się, żeby potomstwo od razu wyeliminować. Wybór rodziców i wybór, jakie osobniki usunąć, mogą być niezależne i w rzeczywistości robione zupełnie innymi sposobami. Możemy, powiedzmy, użyć selekcji proporcjonalnej przy wyborze rodziców i selekcji turniejowej do usuwania osobników. Możemy też zastosować regułę selekcji *elitarnej*, gdzie najlepsze osobniki w populacji mają gwarancję przetrwania do następnego pokolenia. Gwarantuje to, że przystosowanie najlepszego rozwiązania nigdy nie maleje. Własności asymptotycznej zbieżności algorytmów ewolucyjnych na ogół opierają się na elitarności reguły selekcji [399, 145], a algorytmy ewolucyjne oparte na regułach elitarnej często są zbieżne szybciej niż te, które z niej nie korzystają.

7.5. Inicjowanie

Przypomnijmy, że większość algorytmów ewolucyjnych może być zapisana równaniem

$$x[t+1] = s(v(x[t]))$$

przy czym: $x[t]$ – populacja przy reprezentacji x w chwili t , $v(\cdot)$ – operator(y) różnicowania; $s(\cdot)$ – operator selekcji. To równanie różnicowe opisuje probabilistyczny przebieg ewolucji we wszystkich pokoleniach. Zauważmy jednak, że, zanim uruchomimy ten proces, musimy określić $x[0]$, populację początkową. W porównaniu z innymi zagadnieniami dość mało uwagi poświęcono inicjowaniu algorytmów ewolucyjnych. Najprostsze rozwiązanie polega na pobraniu losowej próbki z przestrzeni wszystkich rozwiązań – dzięki temu uzyskujemy nieobciążoną populację początkową. Takie rozwiązanie jest często użyteczne przy porównywaniu algorytmów ewolucyjnych; przy rozwiązywaniu prawdziwych problemów mamy często pewne informacje, które mogą nam pomóc w lepszym ukształtowaniu populacji początkowej. Jeślibyśmy na przykład szukali modelu liniowego o rzeczywistych współczynnikach, który wiązałby wysokość drzewa z jego grubością i wiekiem

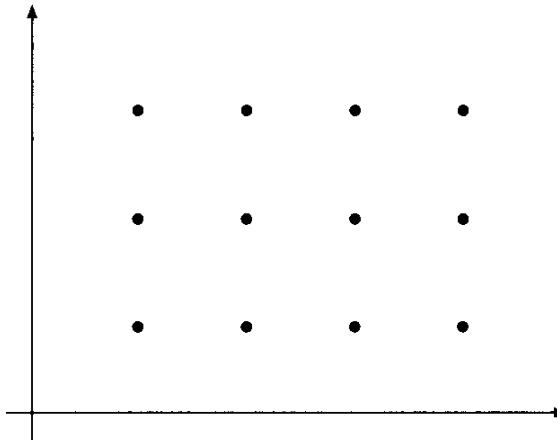
$$\text{Wysokość} = a_1 \cdot \text{Grubość} + a_2 \cdot \text{Wiek}$$

to prawdopodobnie skłaniałybyśmy się ku ograniczeniu wartości początkowych a_1 i a_2 do liczb dodatnich. Nie miałoby większego sensu myślenie o modelach, w których im grubsze jest drzewo, tym jest ono mniejsze, lub w których maleje ono z wiekiem. Deformacja populacji początkowej w kierunku lepszych rozwiązań może często oszczędzić dużo czasu, który w przeciwnym razie spędzielibyśmy na „wymyślaniu koła”. Dlaczego bowiem algorytm ewolucyjny ma się uczyć samego, co już wiemy?

Algorytmy ewolucyjne działające na populacjach dają dodatkową korzyść, polegającą na tym, że możemy zainicjować co najmniej jednego osobnika w po-

pulacji najlepszym rozwiązaniem (lub rozwiązaniami) uzyskanymi za pomocą innych metod (np. algorytmów zachłannych). Metoda ta może być stosowana jako argument za tym, że „algorytmy ewolucyjne nigdy nie będą działały gorzej niż metoda XYZ” (np. „ewolucyjny optymalizator dostarczania surowców nigdy nie może działać gorzej niż system dostarczania na właściwy czas”). Potencjalny użytkownik może znać system XYZ i ufać mu, w związku z czym dodatkowe zabezpieczenie wiedzą, że efektywność algorytmu ewolucyjnego jest ograniczona z dołu przez wyniki uzyskiwane inną metodą, jest uspokajające.

Jeśli nie wiemy wiele na temat problemu i nie możemy dać algorytmowi żadnej podpowiedzi, to należy rozważyć inne sposoby inicjowania populacji niż losowe próbkowanie. Przy rozwiązywaniu ciągłych (lub całkowitoliczbowych) problemów optymalizacji zmiennych jednym ze sposobów jest ułożenie początkowej populacji w siatkę lub inny regularny wzór (rys. 7.9). Zabezpiecza to nas przed uzyskaniem losowej próbki, która skupi się zbytnio w jakimś obszarze. Inna możliwość polega na wymuszeniu, żeby przy wybieraniu następnego osobnika zapewnić pewną minimalną odległość od wszystkich pozostałych. Chociaż takie postępowanie jest czasochłonne, zabezpiecza nas przed nieumyślnym skupieniem początkowych kandydatów na rozwiązania przy zachowaniu losowego charakteru procedury.



Rysunek 7.9. Jednym ze sposobów zapewnienia różnorodności w populacji początkowej jest ustawienie jej w siatkę

7.6. Podsumowanie

W porównaniu z klasycznymi metodami algorytmy ewolucyjne dają nowe możliwości rozwiązywania problemów. Ich podstawową zaletą jest pojęciowa prosta. Podstawowa procedura jest bardzo prosta: wygeneruj populację potencjalnych rozwiązań problemu, opracuj pewne operatory różnicowania, które mogą tworzyć nowe rozwiązania na bazie starych, i zastosuj mechanizm selekcji, żeby zatrzymać te rozwiązania, które dotychczas sprawdzały się najlepiej.

Najpierw musisz mieć jasny, zapisany za pomocą matematycznych pojęć opis tego, co usiłujesz osiągnąć, żeby każdy kandydat na rozwiązanie mógł

być oceniony. Musisz też przemyśleć sposób reprezentacji i przechowywania rozwiązań na komputerze. W przypadku niektórych problemów możesz chcieć skorzystać z ciągów bitów. W innych bardziej intuicyjna może być macierz liczb rzeczywistych. Dla jeszcze innych odpowiednia może być struktura zmiennej długości, na przykład maszyna ze skończoną liczbą stanów lub sieć neuronowa zakodowana w zmiennej wielkości macierzy liczb rzeczywistych. Niestety nie ma najlepszej reprezentacji, która będzie odpowiednia dla wszystkich problemów, musisz zatem użyć swojej intuicji i wybrać strukturę, która zapewnia wgląd w sposób rozwiązania problemu.

Jeśli uruchamiasz algorytm ewolucyjny przy ograniczonym czasie rzeczywistym jego działania i musisz znaleźć rozwiązania w krótkim przedziale czasu, to najprawdopodobniej będziesz chciał użyć deterministycznej metody selekcji kilku najlepszych rozwiązań, żeby służyły za rodziców w następnych pokoleniach. Jeśli rozwiążujesz problem z wieloma lokalnymi optimami, to metoda ta ma większe szanse na utknięcie w którymś z punktów optymalnych, ale jedynie lokalnie. Nie możesz tutaj skorzystać z luksusu podjęcia próby ucieczki, który pojawia się przy doborze probabilistycznym.

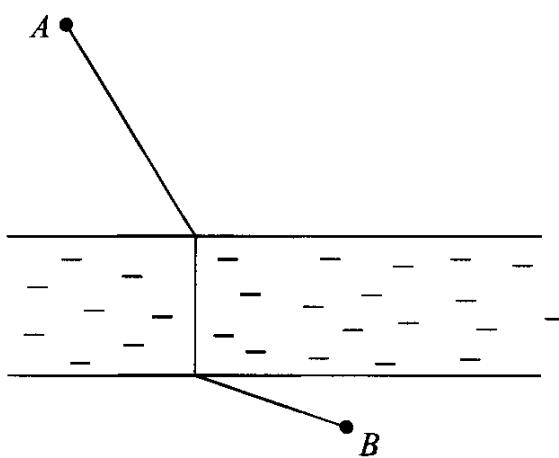
Jeśli dysponujesz wiedzą na temat tego, gdzie szukać dobrych rozwiązań, to możesz i powinieneś włączyć tę wiedzę do poszukiwania ewolucyjnego. Możesz to uwzględnić zarówno przy określeniu populacji początkowej, jak i przy opracowywaniu operatorów różnicowania. W końcu, żeby uzyskać najlepsze działanie Twojego poszukiwania ewolucyjnego, musisz rozważyć, jak wszystkie te aspekty algorytmu pasują do siebie, gdyż żadna z części algorytmu nie może zostać oddzielona od reszty i optymalizowana niezależnie od reszty algorytmu.

W tym rozdziale przedstawiliśmy pierwsze spojrzenie na problemy związane z opracowywaniem praktycznych algorytmów ewolucyjnych. Mamy tu do czynienia z wieloma subtelnymi punktami, które wymagają większej uwagi. Na przykład, jak możemy użyć ewolucji do znalezienia użytecznych układów parametrów operatorów różnicowania jako funkcji zależnej od tego, czego algorytm się nauczył w trakcie trwania samego procesu ewolucji? Jak możemy dawać sobie radę z problemami, w których ograniczenia sprawiają, że wielkie obszary przestrzeni rozwiązań są niedopuszczalne? Jak zamiast jednego znajdować wiele rozwiązań problemu? Te i inne ważne zagadnienia są omawiane w następnych rozdziałach.

VIII. Jaka jest najkrótsza droga?

Wiele skomplikowanych problemów wymaga znalezienia najkrótszej drogi z jednego punktu do drugiego, przy jednoczesnym spełnieniu pewnych ograniczeń. W tym rozdziale omówimy najsłynniejsze z tych problemów. Zanim to jednak nastąpi, rozważmy dwie zagadki, związane z minimalizowaniem długości ścieżki.

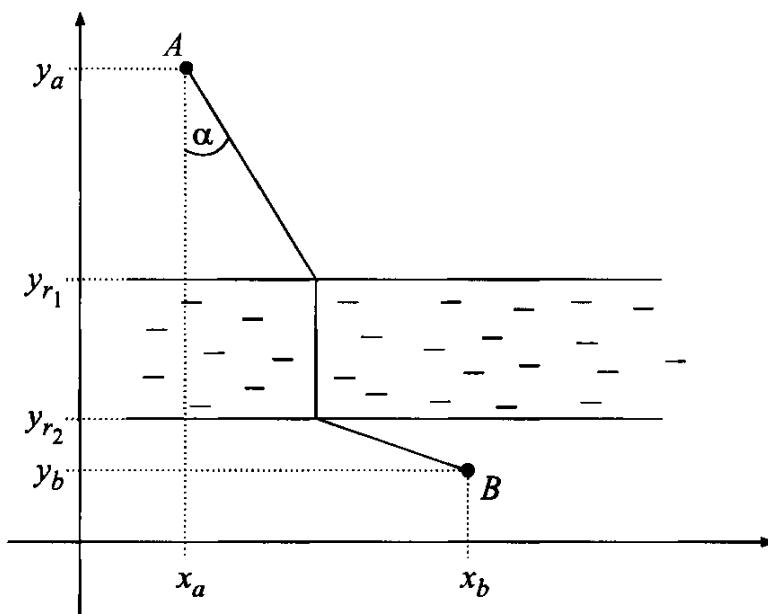
Wyobraźmy sobie, że mamy zbudować drogę z miasta A do miasta B . Miasta te oddziela przepływająca rzeka, a my chcielibyśmy, żeby długość drogi między nimi była jak najmniejsza. Trzeba więc zbudować most prostopadły do brzegów rzeki, co pokazano na rys. VIII.1.



Rysunek VIII.1. Rzeka, dwa miasta A i B oraz łącząca je droga

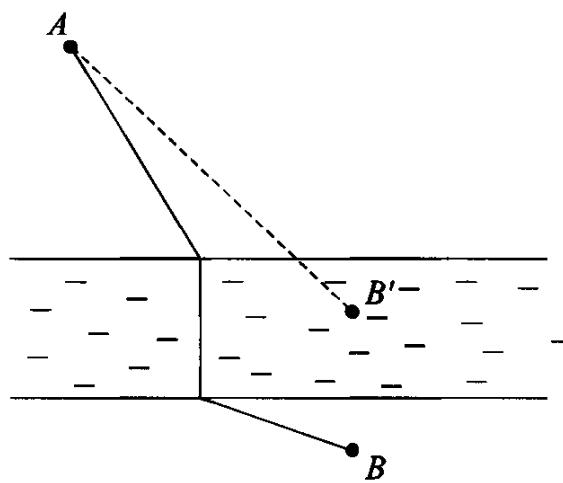
Pytanie brzmi: gdzie wybudować most, żeby zminimalizować całkowitą długość drogi? Istnieje wiele sposobów rozwiązania tego problemu, ale niektóre z nich wymagają mnóstwa obliczeń. Możemy, na przykład, przyjąć jakieś współrzędne (x_a, y_a) i (x_b, y_b) odpowiednio dla miast A i B . Możemy także przyjąć, że rzeka płynie poziomo i jest ograniczona przez y_{r1} i y_{r2} (przy czym

$y_a > y_{r_1} > y_{r_2} > y_b$). Następnie możemy sformułować wzór na długość połączenia między miastami A i B , będącego funkcją kąta α (rys. VIII.2), i znaleźć minimalną długość.



Rysunek VIII.2. Obliczanie minimalnej odległości między miastami A i B

Wszystko stanie się prostsze, jeśli skupimy się na istotnych elementach modelu, a odrzucimy „szum”, czyli elementy, które tarasują naszą drogę do celu. Założymy, że nie ma żadnej rzeki – zostaje zredukowana do linii (o szerokości zero), a miasto B jest przeniesione w górę o odległość równą pierwotnej szerokości rzeki (rys. VIII.3). Tak postawiony problem ma niesamowicie proste rozwiązanie: jest nim linia prosta między miastami A i B' !



Rysunek VIII.3. Dwa miasta A i B' , bez rzeki

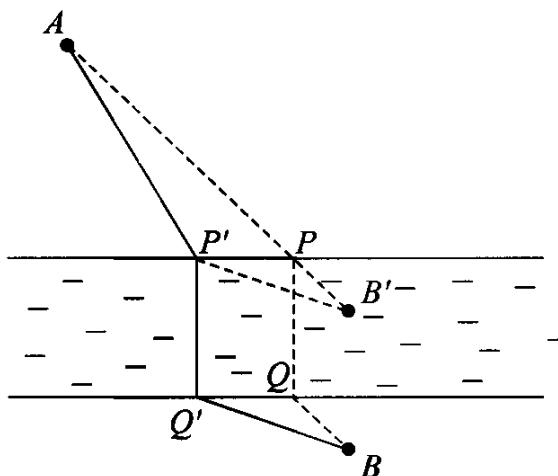
Rozwiązanie to sugeruje także rozwiązanie pierwotnego problemu. Linia biegająca od miasta A do miasta B' przecina brzeg rzeki w jakimś punkcie P . W tym miejscu powinien powstać most, którego drugi koniec znajdzie się w punkcie Q (leżącym na sąsiednim brzegu rzeki) (rys. VIII.4). Odcinki QB i AP są równoległe, tak więc całkowita odległość

$$AP + PQ + QB$$

jest najkrótszą z możliwych. Innymi słowy, w wypadku każdego innego połączenia, na przykład $A - P' - Q' - B$, mamy

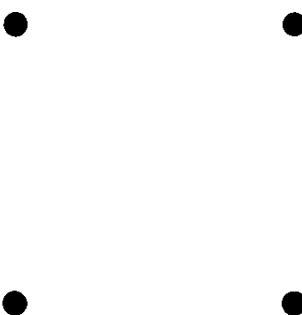
$$\begin{aligned} AP' + P'Q' + Q'B &= AP' + P'Q' + P'B' > \\ &> AB' + P'Q' = AP + PQ + PB' = AP + PQ + QB \end{aligned}$$

gdyż $P'Q' = PQ$ i $AP' + P'B' > AB'$ (nierówność trójkąta). Najlepszym sposobem na rozwiązanie problemu, jak przedostać się przez rzekę, jest zignorowanie jej istnienia!



Rysunek VIII.4. Najkrótsza droga $A - P - Q - B$ między miastami A i B

Rozważmy kolejny problem, w którym jest wymagane znalezienie najkrótszych połączeń. Mamy cztery miasta położone w wierzchołkach kwadratu (rys. VIII.5).

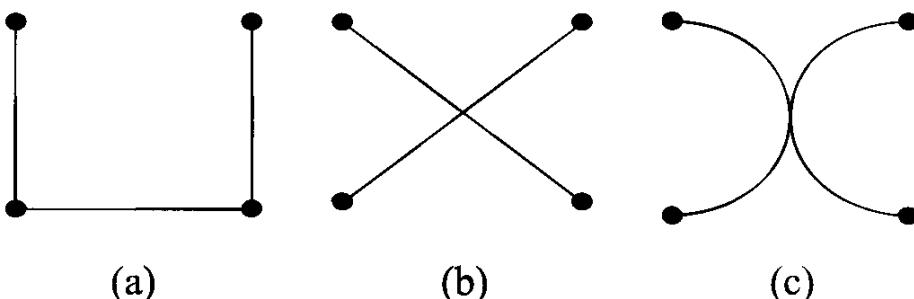


Rysunek VIII.5. Położenie czterech miast

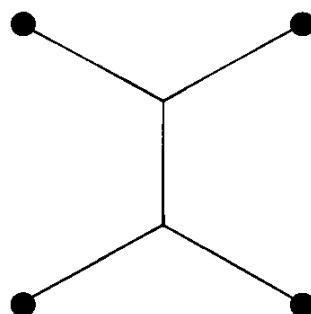
Zadanie polega na zaprojektowaniu takiej sieci dróg, żeby: 1) każde miasto było połączone z każdym innym, 2) całkowita długość dróg była jak najmniejsza. Drogi mogą się przecinać, tzn. jest możliwa zmiana dróg podczas podróży między miastami. Istnieje wiele możliwości, które spełniają warunek 1). Na rysunku VIII.6 przedstawiono niektóre z nich.

Zauważ jednak, że każda z przedstawionych sieci ma inną długość całkowitą. Przyjmując, że długość boku kwadratu wynosi 1, całkowita długość dróg na rys. VIII.6a wynosi 3; na rys. VIII.6b wynosi $2\sqrt{2} = 2,828427$; na rys. VIII.6c wynosi $\pi \approx 3,14159$. To spora różnica!

Optymalne rozwiązanie nie jest wcale oczywiste. Nawet gdy już je poznamy (rys. VIII.7), udowodnienie jego optymalności nie jest oczywiste!



Rysunek VIII.6. Różne sieci dróg



Rysunek VIII.7. Rozwiązanie optymalne. Wszystkie kąty między przecinającymi się drogami wynoszą 120° . Całkowita długość takiej sieci wynosi $1 + \sqrt{3} \approx 2,732$

Powyższy problem można uogólnić następująco: mając n punktów na płaszczyźnie, zaprojektuj taką sieć połączeń między nimi, żeby jej całkowita długość była jak najkrótsza. Zadanie to jest znane jako problem Steinera i zostało rozwiązane tylko w pewnych szczególnych przypadkach. Nie jest znany żaden efektywny algorytm, za pomocą którego można by rozwiązać ogólne przypadki tego problemu. Jednocześnie jest wiele technicznych zastosowań (np. tworzenie sieci telefonicznych, wytyczanie tras samolotów), w których korzystne byłoby użycie takiego algorytmu. Czy możesz go wymyślić? Jeśli tak, będziesz milionerem!

8. Problem komiwojażera

Nikt nie zdaje sobie sprawy z tego, jak pięknie jest podróżować,
dopóki nie wróci do domu i nie ułoży swojej głowy na swojej starej,
dobrze znanej poduszce.

Lin Yutang: *A trip to Anhwei*

Pierwszy z problemów omawianych w rozdziale VIII jest związany z problemem komiwojażera (TSP) tylko powierzchownie. Oba problemy wymagają znalezienia rozwiązania minimalnej długości. Na tym jednak podobieństwa się kończą, gdyż dla TSP nie istnieją żadne sztuczki, które umożliwiłyby znajdowanie doskonałych rozwiązań. Problem ten jest NP-trudny [179] – nie są znane żadne algorytmy znajdowania doskonałych rozwiązań, których złożoność wzrasta wielomianowo ze względu na liczbę miast. W wyniku tego musimy polegać na generowaniu rozwiązań, które są gorsze od doskonałych, ale tak im bliskie, jak to tylko jest możliwe w ramach dostępnego nam czasu. Zadanie to jest bardzo trudne i dlatego cały ten rozdział poświęcimy TSP. Co więcej, TSP jest związane z innymi problemami spokrewnionymi z układaniem planów czy z wyznaczaniem podziałów, a jego ogólność sprawia, że jest on podstawowym elementem optymalizacji kombinatorycznej. Niektóre z pomysłów, które okazały się użyteczne przy TSP, mogą się również przydać w wielu innych zastosowaniach. TSP służy także ilustracji wielu istotnych koncepcji zawartych w tej książce: elastyczności ewolucyjnego podejścia do rozwiązywania problemów, łatwości włączania wiedzy zależnej od problemu oraz łatwości hybrydyzacji.

W informatyce problem komiwojażera jest problemem o specjalnym znaczeniu. Sam problem jest pojęciowo bardzo prosty. Komiwojażer musi odwiedzić każde miasto na swoim terytorium dokładnie raz, a następnie wrócić do miejsca, z którego wyruszył – do swojego domu. Zadanie polega na tym, żeby przy danym koszcie podróży między parami miast opracować pełną trasę o minimalnym koszcie całkowitym. Jedną z możliwych przestrzeni przeszukiwania dla TSP jest zbiór permutacji n miast. Każda permutacja n miast daje rozwiązanie, które jest interpretowane jako pełna trasa, przy czym niejawnie przyjmujemy, że

po ostatnim mieście z listy następuje powrót do domu. Rozmiar tej przestrzeni przeszukiwania wynosi $n!$ ¹.

TSP ma bardzo długą historię. Został wspomniany już przez Eulera w 1759 roku, choć wtedy funkcjonował pod inną nazwą. Euler interesował się problemem trasy konika szachowego. Poprawne rozwiązanie tego problemu polegało na podaniu takiej trasy konika, w której odwiedza on wszystkie 64 pola szachownicy dokładnie raz. Termin „komiwojażer” został pierwszy raz użyty w niemieckiej książce z 1932 roku napisanej przez doświadczonego obwoźnego sprzedawcę: *Komiwojażer, jak i co powinien robić, żeby dostać zapłatę i odnieść sukces w swojej pracy* [279]. Chociaż książka ta nie skupiała się na TSP, problem ten został tam omówiony w ostatnim rozdziale wraz z innymi problemami związanymi z planowaniem.

TSP został wprowadzony przez RAND Corporation w 1948 roku. Reputacja tej firmy pomogła w popularyzacji tego problemu. Został także zauważony ze względu na nowy temat programowania liniowego i próby wykorzystania tej metody do rozwiązywania problemów kombinatorycznych.

TSP pojawia się w wielu zastosowaniach, a rozmiar problemu może być bardzo duży [239]:

W [287] wspomniano zastosowanie TSP o liczbie miast do 17 000 do wiercenia otworów w płytach drukowanych, w [49] wymieniono zagadnienia z krystalografii rentgenowskiej o liczbie miast do 14 000, a w [266] podano, że przykłady związane z wytwarzaniem układów VLSI sięgają rozmiarami 1,2 miliona miast. Co więcej, 5 godzin obliczeń wykonywanych na komputerze za miliony dolarów w celu znalezienia optymalnego rozwiązania może nie być efektywne pod względem kosztu, jeśli gorsze od niego o kilka procent można uzyskać w sekundy na zwykłym PC. Stąd pojawia się potrzeba istnienia heurystyk.

W ciągu ostatnich kilku dziesięcioleci wielu ludzi podało algorytmy generujące przybliżone rozwiązania dla TSP. Należą do nich algorytmy: najbliższego sąsiada, zachłanny, wstawiania najbliższego miasta, wstawiania najdalszego miasta, podwójnego minimalnego drzewa rozpinającego, podziału na pasy, krzywych wypełniających przestrzeń i inne zaproponowane przez Karpa, Litkego, Christofidesa itd. [239]. W niektórych z tych algorytmów założono, że miasta odpowiadają punktom na płaszczyźnie o jakiejś standardowej metryce. Inna grupa algorytmów (np. 2-opt, 3-opt, Lina-Kernighana) skupia się na lokalnej optymalizacji: poprawia trasę, wprowadzając lokalne zaburzenia. Problem komiwojażera w latach osiemdziesiątych i dziewięćdziesiątych XX wieku stał się celem dla społeczności naukowców zajmujących się metodami ewolucyjnymi.

¹ Jak już zauważyliśmy we wcześniejszych rozdziałach, ponieważ problem jest symetryczny i każda trasa jest pełnym cyklem, przestrzeń tę możemy zmniejszyć do rozmiaru $(n - 1)!/2$. Dowolny algorytm poszukiwania, który działa po prostu na permutacjach, musi jednak w dalszym ciągu obsługiwać $n!$ możliwości.

W pierwszych latach tego zainteresowania podejście ewolucyjne opisano w pracach [142], [174], [192], [203], [202], [238], [282], [330], [338], [430], [452], [457], [471] oraz [494]. Co więcej, na prawie każdej konferencji dotyczącej algorytmów ewolucyjnych wciąż przedstawia się kilka prac dotyczących TSP. Poszukiwanie „świętego Graala” – idealnego algorytmu ewolucyjnego rozwiązującego TSP – wciąż trwa! Oczywiście taki algorytm powinien być konkurencyjny w stosunku do obecnie stosowanych metod.

Ciekawe jest porównanie tych podejść ze szczególnym zwróceniem uwagi na reprezentację i używane operatory różnicowania. W tym rozdziale prześledzimy ewolucję algorytmów ewolucyjnych dla TSP, nie będziemy jednak podawać dokładnych wyników różnych eksperymentów dla rozmaitych systemów opracowanych przez te lata. Bardzo mało publikacji podaje czas obliczeniowy potrzebny do rozwiązania konkretnych przykładów TSP. Zamiast tego podaje się liczbę wyliczeń funkcji oceny. Utrudnia to porównywanie podejść, gdyż różne operatory mają różne złożoności czasowe. Eshelman w [129] podał, że rozwiązywanie TSP z 532 miastami zajęło mu 2,5 godziny na jednoprocesorowym komputerze Sun SPARCstation. Gorges-Schleuter [195] podał dla tego samego problemu czas między 1 a 2 godzinami, ale zastosował implementację równoległą na 64 transputerach T800. Braun w [56] podał czas 30 minut dla problemu z 431 miastami przy rozwiązywaniu na stacji roboczej Sun. We wszystkich tych opisach pominięto porównanie jakości końcowego rozwiązania, przy czym wskazano, że algorytmy ewolucyjne mogą być zbyt wolne przy rozwiązywaniu dużych zadań TSP (np. o ponad 10 000 miast). Zacytowane wyniki efektywnościowe zależą silnie od wielu szczegółów, w tym od rozmiaru populacji, liczby pokoleń, liczby miast itd. Co więcej, wiele wyników dotyczyło tylko relatywnie małych zagadnień TSP (tzn. do 100 miast). Jak zanotowano w [239]:

Wygląda na to, że zadania o wielkości 100 miast muszą być obecnie uważane za znajdujące się w zakresie metod optymalizacji globalnej, a przykłady problemów muszą być dużo większe, żeby mieć pewność, że naprawdę należy stosować podejście heurystyczne.

Tego rodzaju problemy są obecnie rutynowo rozwiązywane w czasie kilku godzin [241].

W większości cytowanych w tym rozdziale prac proponowane podejście jest porównywane z innymi podejściami. Przy takich porównaniach są wykorzystywane dwa główne rodzaje danych testowych:

- Miasta w badanym przykładzie TSP są rozłożone losowo, zwykle zgodnie z jednostajną zmienną losową; założono przy tym metrykę euklidesową. Przydatny jest tutaj empirycznie określony wzór, podający oczekiwana długość L^* minimalnej trasy:

$$L^* = k\sqrt{n \cdot R}$$

przy czym: n oznacza liczbę miast, R jest polem prostokąta, wewnątrz którego miasta są ułożone losowo, k jest doświadczalnie wyznaczoną

stałą. Wartość oczekiwana k dla ograniczenia Helda-Karpa do \sqrt{n} dla losowego euklidesowego TSP (dla $n \geq 100$) wynosi

$$k = 0,70805 + \frac{0,52229}{\sqrt{n}} + \frac{1,31572}{n} - \frac{3,07474}{n\sqrt{n}}$$

Bonomi i Lutton [51] polecają stosowanie $k = 0,749$.

- Miasta w badanym przykładzie TSP są pobierane z jakiegoś publicznie dostępnego zbioru danych testowych (TSPLIB [385]) z udokumentowanymi optymalnymi lub najlepiej znanyimi rozwiązaniami¹.

8.1. W poszukiwaniu dobrych operatorów różnicowania

Problem komiwojażera ma bardzo łatwą i naturalną funkcję oceny: dla każdego potencjalnego rozwiązania – permutacji miast – możemy określić wszystkie odległości między uporządkowanymi miastami i po $n - 1$ operacjach dodawania obliczyć całkowitą długość trasy. W populacji tras możemy łatwo porównać dowolne dwie z nich, ale wybór sposobu reprezentacji trasy i wybór operatorów różnicowania nie są już tak oczywiste.

Wcześniej zauważyliśmy, że jedną z naturalnych reprezentacji dla TSP jest permutacja. Wydaje się, że oddaje ona istotę postępu w podróży od jednego miasta do drugiego. Na początkowych etapach rozwoju algorytmów ewolucyjnych panował jednak pogląd, że niezależnie od problemu korzystna byłaby reprezentacja binarna [228]. Z perspektywy czasu brzmi to trochę nieprawdopodobnie, ale w istocie można potwierdzić dowodem brak zaufania do tego poglądu. Żadna bijektywna reprezentacja nie jest lepsza od jakiejkolwiek innej przy rozwiązyaniu wszystkich problemów. Pojawia się zatem pytanie, czy reprezentacja binarna daje jakąkolwiek widoczną korzyść przy zajmowaniu się TSP?

Po chwili zastanowienia ciągi binarne wydają się słabo pasować do reprezentacji rozwiązań zadań TSP. Nietrudno spostrzec, dlaczego tak jest. W końcu interesuje nas najlepsza permutacja miast, tzn.

$$(i_1, i_2, \dots, i_n)$$

przy czym (i_1, i_2, \dots, i_n) jest permutacją ciągu $\{1, 2, \dots, n\}$. Kodowanie binarne tych miast nie czyni życia łatwiejszym. W istocie prawdziwe jest odwrotne stwierdzenie: Reprezentacja binarna wymagałaby specjalnych operatorów *naprawy*, które usuwałyby błędy powstające przy generowaniu nielegalnych (tzn. niedopuszczalnych) tras. Jak zaobserwowano w [494]:

Niestety, nie istnieje praktyczny sposób kodowania TSP w postaci ciągów binarnych, w których nie ma zależności porządkowych lub

¹Zobacz <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/index.html>.

do których można zastosować operatory w sensowny sposób. Proste krzyżowanie ciągów miast wprowadza duplikaty i pominięcia. Aby zatem rozwiązać ten problem, należy użyć jakiejś odmiany standardowego... krzyżowania. Idealny operator rekombinacji powinien wymieniać krytyczne informacje ze struktur rodziców w niedestrukcyjny, zrozumiały sposób.

Chociaż ogólna koncepcja zawarta w tej opinii jest słuszna, zauważmy, że jest ona nieco nagięta w kierunku poszukiwania operatorów rekombinacji. To często popełniany błąd przy opracowywaniu algorytmów ewolucyjnych. Zamiast skupiać się na jakiejś klasie operatorów różnicowania, należy szerzej myśleć o sposobach tworzenia operatorów różnicowania odpowiednich dla rozwiązywanego zadania. Mogą one uwzględniać jakiś rodzaj rekombinacji, ale nie muszą. Operator różnicowania, który z dużym prawdopodobieństwem generuje niedopuszczalne rozwiązania, jest również z dużym prawdopodobieństwem operatorem, którego nie chcemy stosować¹.

W latach osiemdziesiątych XX wieku rozważano dla TSP trzy reprezentacje wektorowe: *w postaci listy sąsiedztwa* (przyległościową), *porządkową* i *ścieżkową*. Każda z tych reprezentacji ma związane ze sobą własne operatory różnicowania – omówimy je po kolej. A ponieważ jest stosunkowo łatwo wymyślić pewien rodzaj operatora mutacji, który wprowadza niewielką zmianę do trasy, skoncentrujemy się najpierw na projektowaniu możliwych użytecznych operatorów rekombinacji. We wszystkich trzech przypadkach trasa jest opisana jako lista miast. W przykładach będziemy używali układu dziesięciu miast ponumerowanych intuicyjnie od jednego do dziesięciu.

Reprezentacja w postaci listy sąsiedztwa

W reprezentacji w postaci listy sąsiedztwa trasa jest zakodowana jako lista n miast. Miasto j jest wymienione na pozycji i wtedy i tylko wtedy, gdy trasa prowadzi z miasta i do miasta j . Na przykład wektor

$$(2 \ 4 \ 8 \ 3 \ 9 \ 7 \ 1 \ 5 \ 6)$$

reprezentuje następującą trasę:

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

Jeśli to nie jest jasne, to wróć i sprawdź, czy potrafisz stwierdzić, dlaczego każde z miast w wektorze jest umieszczone na swoim miejscu. Każda trasa ma tylko jedną reprezentację w postaci listy sąsiedztwa; niektóre z list mogą jednak reprezentować trasy niedopuszczalne; na przykład

$$(2 \ 4 \ 8 \ 1 \ 9 \ 3 \ 5 \ 7 \ 6)$$

¹Lidd w [282] osiągnął pewien sukces, stosując dla TSP reprezentację binarną oraz proste operatory krzyżowania i mutacji; trasy niedopuszczalne były „naprawiane” za pomocą procedury zachłannej, nie jest więc jasne, jaki zysk dawał ten algorytm ewolucyjny. Co więcej, w żadnym z omawianych tam zadań nie było więcej niż 100 miast.

prowadzi do

$$1 - 2 - 4 - 1$$

tzn. częściowej trasy ze zbyt wcześnie kończącym się (przedwczesnym) cyklem.

Reprezentacja w postaci listy sąsiedztwa nie wspiera prostego operatora krzyżowania przez „cięcie i łączenie”¹. Konieczne może być dołożenie kolejnego operatora naprawy, który poprawia niepoprawne pozycje. Zamiast tego Grefenstette i in. [203] zaproponowali operatory rekombinacji, które pasują do reprezentacji w postaci listy sąsiedztwa: *krzyżowanie przez naprzemienne wybieranie krawędzi*, *krzyżowanie przez wymianę podtras* i *krzyżowanie heurystyczne*.

Krzyżowanie przez naprzemienne wybieranie krawędzi. Operator ten tworzy potomka przez losowe wybranie krawędzi z pierwszego rodzica, następnie wybranie odpowiedniej krawędzi z drugiego itd. – operator rozszerza trasę przez naprzemienne wybieranie krawędzi z dwóch rodziców. Jeśli nowa krawędź od jednego z rodziców wprowadza cykl do bieżącej, wciąż częściowej trasy, to operator zamiast niej wybiera losowo jedną z pozostałych krawędzi, które nie wprowadzają cyklu. Na przykład pierwszy potomek dwóch rodziców

$$p_1 = (2 \ 3 \ 8 \ 7 \ 9 \ 1 \ 4 \ 5 \ 6)$$

$$p_2 = (7 \ 5 \ 1 \ 6 \ 9 \ 2 \ 8 \ 4 \ 3)$$

mogłby wyglądać tak:

$$o_1 = (2 \ 5 \ 8 \ 7 \ 9 \ 1 \ 6 \ 4 \ 3)$$

przy czym proces zaczął się od krawędzi (1 2) z rodzica p_1 , a jedyną losową krawędzią wprowadzoną podczas procesu naprzemennego wybierania krawędzi jest krawędź (7 6) zamiast (7 8), która wprowadziłaby cykl przedwczesny.

Krzyżowanie przez wymianę podtras. Operator ten tworzy potomka przez wybranie podtrasy o losowej długości z jednego rodzica, następnie z drugiego itd. – operator rozszerza trasę, pobierając krawędzie na przemian z różnych rodziców. I znowu, jeśli jakaś krawędź od jednego z rodziców wprowadza cykl do bieżącej, wciąż częściowej trasy, to operator zamiast niej wybiera losową krawędź spośród jeszcze niewybranych, która nie wprowadza cyklu.

Krzyżowanie heurystyczne. Operator ten tworzy potomka przez wybór losowego miasta jako punktu początkowego dla trasy potomka. Następnie porównuje dwie krawędzie, które wychodzą z tego miasta w obu rodzicach, i wybiera lepszą (tzn. krótszą) z nich. Miasto na drugim końcu wybranej krawędzi

¹Nawet jeśli wspierała, to nie byłoby powodu z niego korzystać. Musi istnieć pewnego rodzaju dopasowanie między operatorem, reprezentacją i rozwiązywanym problemem. Sam fakt, że operator daje rozwiązania dopuszczalne, nie jest wystarczającym uzasadnieniem jego stosowania.

służy za punkt początkowy przy wyborze krótszej z dwóch krawędzi wychodzących z tego miasta itd. Jeśli na którymś etapie nowa krawędź wprowadza cykl do częściowej trasy, to trasa ta jest rozszerzana o losową krawędź wybraną spośród pozostałych, niewprowadzających cyklu.

To krzyżowanie heurystyczne zostało zmodyfikowane w [238], gdzie zmieniono dwie reguły: 1) jeśli krótsza krawędź rodzica wprowadza cykl w trasie potomka, to sprawdź dłuższą. Jeśli dłuższa krawędź nie wprowadza cyklu, to zaakceptuj ją, w przeciwnym razie 2) wybierz najkrótszą spośród q losowo wybranych krawędzi (q jest parametrem tej metody). W wyniku działania tego operatora są sklejane krótkie podścieżki wybrane z tras rodziców. Jego stosowanie może jednak wprowadzać niepożądane krzyżowanie się krawędzi, a zatem tego rodzaju krzyżowanie heurystyczne nie nadaje się do drobnych lokalnych optymalizacji potencjalnych rozwiązań.

Suh i Gucht w [457] wprowadzili dodatkowy operator heurystyczny oparty na algorytmie 2-opt [284], który nadaje się do wspomnianego lokalnego ulepszania. Operator ten losowo wybiera dwie krawędzie $(i \ j)$ oraz $(k \ m)$ i sprawdza, czy

$$dist(i, j) + dist(k, m) > dist(i, m) + dist(k, j)$$

gdzie $dist(a, b)$ oznacza odległość między miastami a i b . Jeśli ten warunek zachodzi, to krawędzie $(i \ j)$ oraz $(k \ m)$ są zastępowane krawędziami $(i \ m)$ oraz $(k \ j)$.

Jedną z możliwych zalet reprezentacji w postaci listy sąsiedztwa jest to, że umożliwia nam wprowadzanie wzorców związanych z dobrymi rozwiązaniami. Na przykład wzorzec

$(* * * 3 * 7 * *)$

oznacza zbiór wszystkich tras z krawędziami $(4 \ 3)$ i $(6 \ 7)$. Dzięki temu możemy zastosować jakiś rodzaj procedury podziału i ograniczeń i skupić naszą uwagę na takim wąskim podzbiorze wszystkich możliwych rozwiązań. Rezultaty uzyskane za pomocą wszystkich standardowych operatorów różnicowania okazały się niezadowalające. Krzyżowanie przez naprzemienne wybieranie krawędzi w wyniku działania polegającego na zmienianiu rodzica co krawędź często psuje dobre trasy. Przy tym operatorze trudno jest wygenerować ciągi krawędzi i przechowywać je w jakimś rodzicu. Krzyżowanie przez wymianę podtras działa tutaj lepiej niż krzyżowanie przez naprzemienne wybieranie krawędzi, najprawdopodobniej dlatego, że z mniejszą częstością narusza pojawiające się wzorce. Nawet przy tym operatorze wyniki doświadczalne nie były jednak zadowalające. Z kolei zastosowanie do TSP krzyżowania heurystycznego wypada lepiej, gdyż te pierwsze dwa operatory są ślepe, tzn. nie biorą pod uwagę rzeczywistych długości krawędzi. Przy tym krzyżowanie heurystyczne wybiera lepszą z dwóch możliwych krawędzi. Pod pewnymi względami korzysta ono z informacji lokalnej, żeby poprawić szanse na wygenerowanie lepszych rozwiązań. Trzeba jednak przyznać, że nawet krzyżowanie heurystyczne nie dawało zbyt dobrych wyników. W trzech eksperymentach dotyczących TSP z 50, 100 i 200 miastami

algorytm ewolucyjny znalazł trasy odległe o 25, 16 i 27% od optimum przy odpowiednio 15 000, 20 000 i 25 000 pokoleń [203].

Reprezentacja porządkowa

W reprezentacji porządkowej trasa jest zakodowana jako lista n miast; i -ty element listy jest liczbą z zakresu od 1 do $n - i + 1$. Podstawą jest tu następująca koncepcja. Mamy do dyspozycji pewną uporządkowaną listę miast C , która służy za punkt odniesienia dla list w reprezentacji porządkowej. Założymy na przykład, że taka uporządkowana lista (punkt odniesienia) to

$$C = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

Trasa

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

jest reprezentowana jako lista l zawierająca wskaźniki:

$$l = (1 \ 1 \ 2 \ 1 \ 4 \ 1 \ 3 \ 1 \ 1)$$

i powinna być interpretowana w następujący sposób:

Pierwszą liczbą na liście l jest 1, zatem jako pierwsze miasto na trasie bierzemy pierwsze miasto z listy C (miasto o numerze 1) i usuwamy je z C . Tak powstała trasa częściowa to

$$1$$

Następną liczbą na liście l jest także 1, zatem jako następne miasto na trasie bierzemy pierwsze miasto z bieżącej listy C (miasto o numerze 2) i usuwamy je z C . Tak powstała trasa częściowa to

$$1 - 2$$

Następną liczbą na liście l jest 2, zatem jako następne miasto na trasie bierzemy drugie miasto z bieżączej listy C (miasto o numerze 4) i usuwamy je z listy C . Nowa trasa częściowa to

$$1 - 2 - 4$$

Następną liczbą na liście l jest 1, zatem jako następne miasto na trasie bierzemy pierwsze miasto z bieżącej listy C (miasto o numerze 3) i usuwamy je z C . Trasa częściowa to

$$1 - 2 - 4 - 3$$

Następną liczbą na liście l jest 4, zatem jako następne miasto na trasie bierzemy czwarte miasto z bieżącej listy C (miasto o numerze 8) i usuwamy je z C . Trasa częściowa to

$$1 - 2 - 4 - 3 - 8$$

Następną liczbą na liście l jest znowu 1, zatem jako następne miasto na trasie bierzemy pierwsze miasto z bieżącej listy C (miasto o numerze 5) i usuwamy je z C . Trasa częściowa to

$$1 - 2 - 4 - 3 - 8 - 5$$

Następną liczbą na liście l jest 3, zatem jako następne miasto na trasie bierzemy trzecie miasto z bieżącej listy C (miasto o numerze 9) i usuwamy je z C . Trasa częściowa to

$$1 - 2 - 4 - 3 - 8 - 5 - 9$$

Następną liczbą na liście l jest 1, zatem jako następne miasto na trasie bierzemy pierwsze miasto z bieżącej listy C (miasto o numerze 6) i usuwamy je z C . Trasa częściowa to

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6$$

Ostatnią liczbą na liście l jest 1, zatem jako następne miasto na trasie bierzemy pierwsze miasto z bieżącej listy C (miasto o numerze 7 – ostatnie dostępne miasto) i usuwamy je z C . Ostateczna trasa to

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

Główna (i chyba jedyna) zaleta reprezentacji porządkowej polega na tym, że działa tutaj klasyczne krzyżowanie przez cięcie i łączenie! Dla danych dwóch tras w reprezentacji porządkowej, jeśli wykonamy cięcie po jakiejś pozycji i weźmiemy pierwszą część trasy z pierwszego rodzica, drugą zaś z drugiego (lub na odwrót), to w wyniku otrzymamy trasę dopuszczalną. Na przykład dla dwóch rodziców:

$$p_1 = (1 \ 1 \ 2 \ 1 \mid 4 \ 1 \ 3 \ 1 \ 1)$$

$$p_2 = (5 \ 1 \ 5 \ 5 \mid 5 \ 3 \ 3 \ 2 \ 1)$$

którzy odpowiadają trasom

$$1 - 2 - 4 - 3 - 8 - 5 - 9 - 6 - 7$$

$$5 - 1 - 7 - 8 - 9 - 4 - 6 - 3 - 2$$

z punktem krzyżowania (cięcia) zaznaczonym jako „|”, otrzymamy następujące potomstwo:

$$o_1 = (1 \ 1 \ 2 \ 1 \mid 5 \ 3 \ 3 \ 2 \ 1)$$

$$o_2 = (5 \ 1 \ 5 \ 5 \mid 4 \ 1 \ 3 \ 1 \ 1)$$

Osobniki te odpowiadają trasom

$$1 - 2 - 4 - 3 - 9 - 7 - 8 - 6 - 5$$

$$5 - 1 - 7 - 8 - 6 - 2 - 9 - 3 - 4$$

Łatwo zauważyc, że częściowe trasy na lewo od punktu krzyżowania nie zmieniają się, natomiast częściowe trasy na prawo od punktu krzyżowania są

w zasadzie ułożone losowo. To niezbyt dobrze. Mamy jednak nadzieję, iż czytając powyższą uwagę, że przy tej reprezentacji działa proste krzyżowanie, pomyślałeś „i co z tego?”. Tworzenie poprawnych tras znowu nie wystarczy, potomstwo musi dziedziczyć po rodzicach dobre cechy i to w efektywniejszy sposób, niż jak to wynika z poszukiwania na ślepo. Nic dziwnego, że po połączeniu tej reprezentacji z prostym krzyżowaniem jednopunktowym wyniki były kiepskie [203].

Reprezentacja ścieżkowa

Reprezentacja ścieżkowa jest chyba najbardziej naturalną reprezentacją trasy. Na przykład trasa

$$5 - 1 - 7 - 8 - 9 - 4 - 6 - 2 - 3$$

jest reprezentowana jako

$$(5 \ 1 \ 7 \ 8 \ 9 \ 4 \ 6 \ 2 \ 3)$$

Najbardziej znane operatory krzyżowania dla reprezentacji ścieżkowej to: krzyżowanie z częściowym odwzorowaniem (ang. *partially-mapped crossover* – PMX), krzyżowanie z zachowaniem porządku (ang. *order crossover* – OX) i krzyżowanie cykliczne (ang. *cycle crossover* – CX).

PMX. Operator ten tworzy potomstwo, wybierając podciąg trasy z jednego rodzica i zachowując porządek i pozycje tak wielu miast z drugiego rodzica, jak jest to tylko możliwe [192]. Podciąg trasy wybiera się, typując dwa losowe punkty cięcia, które służą za granice operacji zamiany. Na przykład dla pary rodziców (dwa punkty cięcia są zaznaczone za pomocą „|”)

$$\begin{aligned} p_1 &= (1 \ 2 \ 3 \mid 4 \ 5 \ 6 \ 7 \mid 8 \ 9) \\ p_2 &= (4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3) \end{aligned}$$

potomstwo zostanie utworzone w następujący sposób. Najpierw są wymieniane segmenty między punktami cięcia (symbol „x” można interpretować jako „obecnie nieznane”):

$$\begin{aligned} o_1 &= (\mathbf{x} \ \mathbf{x} \ \mathbf{x} \mid 1 \ 8 \ 7 \ 6 \mid \mathbf{x} \ \mathbf{x}) \\ o_2 &= (\mathbf{x} \ \mathbf{x} \ \mathbf{x} \mid 4 \ 5 \ 6 \ 7 \mid \mathbf{x} \ \mathbf{x}) \end{aligned}$$

Taka wymiana prowadzi do ciągu odwzorowań

$$1 \leftrightarrow 4, \quad 8 \leftrightarrow 5, \quad 7 \leftrightarrow 6 \text{ oraz } 6 \leftrightarrow 7$$

Następnie możemy wstawić dodatkowe miasta z pierwotnych rodziców, dla których nie występują konflikty:

$$\begin{aligned} o_1 &= (\mathbf{x} \ 2 \ 3 \mid 1 \ 8 \ 7 \ 6 \mid \mathbf{x} \ 9) \\ o_2 &= (\mathbf{x} \ \mathbf{x} \ 2 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3) \end{aligned}$$

Wreszcie pierwszy x w potomku o_1 (który powinien być równy 1, ale wystąpił konflikt) zostaje zastąpiony przez 4 ze względu na odwzorowanie $1 \leftrightarrow 4$. Podobnie drugi x w potomku o_1 zostaje zastąpiony przez 5, a odpowiednie x w potomku o_2 są zastępowane przez 1 i 8. Zatem potomkowie wyglądają tak:

$$\begin{aligned} o_1 &= (4 \ 2 \ 3 \mid 1 \ 8 \ 7 \ 6 \mid 5 \ 9) \\ o_2 &= (1 \ 8 \ 2 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3) \end{aligned}$$

Operator PMX, przy użyciu odpowiedniego planu reprodukcji, wykorzystuje jednocześnie podobieństwa w uporządkowaniu i wartości [192]. Algorytm ewolucyjny w [142] dał jednak lepsze wyniki przy mniejszej liczbie wyliczeń funkcji oceny. Zastosowano tam po prostu działający na jednym rodzicu operator różnicowania typu „usuń i zastąp” przy reprezentacji porządkowej dla zadań TSP ze 100 miastami.

OX. Operator ten tworzy potomstwo, wybierając podciąg trasy z jednego rodzica i zachowując wzajemne uporządkowanie miast z drugiego [86]. Na przykład dwóch rodziców (o punktach cięcia zaznaczonych „|”)

$$\begin{aligned} p_1 &= (1 \ 2 \ 3 \mid 4 \ 5 \ 6 \ 7 \mid 8 \ 9) \\ p_2 &= (4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3) \end{aligned}$$

dałoby potomstwo w następujący sposób. Najpierw segmenty między punktami cięcia są kopiowane do potomków:

$$\begin{aligned} o_1 &= (\text{x } \text{x } \text{x} \mid 4 \ 5 \ 6 \ 7 \mid \text{x } \text{x}) \\ o_2 &= (\text{x } \text{x } \text{x} \mid 1 \ 8 \ 7 \ 6 \mid \text{x } \text{x}) \end{aligned}$$

Następnie, począwszy od drugiego punktu cięcia w jednym z rodziców, miasta z drugiego rodzica są kopiowane w takim samym porządku, ale z pominięciem symboli już obecnych. Po osiągnięciu końca ciągu kontynuujemy kopowanie od pierwszego miejsca ciągu. Ciąg miast w drugim rodzicu (pożyczony od drugiego punktu cięcia) wygląda tak:

$$9 - 3 - 4 - 5 - 2 - 1 - 8 - 7 - 6$$

Po usunięciu miast 4, 5, 6 i 7, które już się znajdują w pierwszym potomku, uzyskujemy

$$9 - 3 - 2 - 1 - 8$$

Ciąg ten jest umieszczany w pierwszym potomku (pożyczony od drugiego punktu cięcia):

$$o_1 = (2 \ 1 \ 8 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3)$$

W podobny sposób uzyskujemy drugiego potomka:

$$o_2 = (3 \ 4 \ 5 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 2)$$

Operator OX wykorzystuje własność reprezentacji ścieżkowej polegającą na tym, że ważne jest wzajemne uporządkowanie miast (w odróżnieniu od ich konkretnych pozycji), tzn. dwie trasy

$$\begin{aligned} 9 - 3 - 4 - 5 - 2 - 1 - 8 - 7 - 6 \\ 4 - 5 - 2 - 1 - 8 - 7 - 6 - 9 - 3 \end{aligned}$$

są w rzeczywistości identyczne.

CX. Operator ten tworzy potomstwo w ten sposób, że każde miasto wraz z jego pozycją pochodzi od jednego z rodziców [338]. Krzyżowanie cykliczne działa następująco. Dwóch rodziców, powiedzmy:

$$\begin{aligned} p_1 &= (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \\ p_2 &= (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5) \end{aligned}$$

tworzy pierwszego potomka, biorąc pierwsze miasto z pierwszego rodzica:

$$o_1 = (1 \ x \ x \ x \ x \ x \ x \ x \ x)$$

Ponieważ każde miasto w potomku powinno być wzięte od jednego z rodziców (z tej samej pozycji), w tym momencie nie mamy żadnego wyboru. Następnym rozważanym miastem musi być miasto 4, gdyż jest to miasto z rodzica p_2 , znajdujące się tuż „pod” wybranym miastem 1. W p_1 miasto to znajduje się na pozycji „4”, mamy zatem

$$o_1 = (1 \ x \ x \ 4 \ x \ x \ x \ x \ x)$$

To z kolei wyznacza miasto 8, gdyż jest to miasto w rodzicu p_2 , które występuje tuż „pod” wybranym miastem 4. Mamy zatem

$$o_1 = (1 \ x \ x \ 4 \ x \ x \ x \ 8 \ x)$$

Stosując tę regułę dalej, otrzymujemy jako następne miasta 3 i 2. Zauważmy jednak, że wybranie miasta 2 wymaga wybrania miasta 1, które już się pojawiło na liście. W ten sposób zamknęliśmy cykl

$$o_1 = (1 \ 2 \ 3 \ 4 \ x \ x \ x \ 8 \ x)$$

Pozostałe miasta są wyprowadzane z drugiego rodzica:

$$o_1 = (1 \ 2 \ 3 \ 4 \ 7 \ 6 \ 9 \ 8 \ 5)$$

Podobnie

$$o_2 = (4 \ 1 \ 2 \ 8 \ 5 \ 6 \ 7 \ 3 \ 9)$$

CX zachowuje bezwzględne pozycje elementów z ciągów rodziców.

Jest też możliwe określenie innych operatorów działających przy reprezentacji ścieżkowej. Syswerda [461], na przykład, zdefiniował dwie zmodyfikowane wersje operatora krzyżowania z zachowaniem porządku. W pierwszej modyfikacji (zwanej krzyżowaniem opartym na kolejności – ang. *order-based crossover*)

wybiera się losowo kilka pozycji w wektorze, a kolejność miast na wybranych pozycjach w jednym rodzicu jest narzucona odpowiednim miastom w drugim rodzicu. Jako przykład weźmy dwóch rodziców:

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

$$p_2 = (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5)$$

Załóżmy, że wybrane pozycje to pozycje 3, 4, 6 i 9. Uporządkowanie miast na tych pozycjach w rodzicu p_2 zostanie narzucone rodzicowi p_1 . Miasta znajdujące się w p_2 na tych pozycjach to (w kolejności występowania) 2, 8, 6 i 5. W rodzicu p_1 miasta te występują na pozycjach 2, 5, 6 i 8. W potomku elementy na tych pozycjach są przedstawiane tak, żeby zgadzały się co do kolejności z tymi samymi elementami z p_2 (kolejność to 2 – 8 – 6 – 5). Pierwszy potomek jest kopią p_1 na wszystkich pozycjach oprócz 2, 5, 6 i 8:

$$o_1 = (1 \ x \ 3 \ 4 \ x \ x \ 7 \ x \ 9)$$

Wszystkie pozostałe elementy są umieszczone zgodnie z kolejnością ich występowania w p_2 , tzn. 2, 8, 6, 5, tak więc w końcu uzyskujemy

$$o_1 = (1 \ 2 \ 3 \ 4 \ 8 \ 6 \ 7 \ 5 \ 9)$$

Podobnie możemy utworzyć drugiego potomka

$$o_2 = (3 \ 1 \ 2 \ 8 \ 7 \ 4 \ 6 \ 9 \ 5)$$

Druga zmodyfikowana wersja (zwana krzyżowaniem opartym na pozycjach – ang. *position-based crossover*) jest bardziej podobna do pierwotnego krzyżowania z zachowaniem porządku. Jedyna różnica polega na tym, że w krzyżowaniu opartym na pozycjach zamiast wybierać jeden podciąg miast do skopiowania, wybiera się (losowo) w tym celu kilka miast.

Warto zauważyć, że te dwa operatory (krzyżowanie oparte na kolejności i krzyżowanie oparte na pozycjach) są w pewnym sensie równoważne. Krzyżowanie oparte na kolejności z pewną liczbą pozycji wybranych jako punkty cięcia i krzyżowanie oparte na pozycjach z pozostałymi pozycjami jako punktami cięcia zawsze dadzą ten sam wynik. Oznacza to, że jeśli średnia liczba punktów krzyżowania wynosi $n/2$, przy czym n jest liczbą wszystkich miast, to te dwa operatory powinny zapewniać tę samą efektywność. Jeśli jednak średnia liczba punktów krzyżowania wynosi, powiedzmy, $n/10$, to operatory te będą się zachowywać inaczej. Więcej informacji na temat tych operatorów oraz ich teoretyczne i empiryczne porównania można znaleźć w [174], [338], [452] i [460].

Przeglądając różne operatory zmiany uporządkowania, które ostatnio się pojawiły, powinniśmy wspomnieć też o operatorze inwersji. Prosta inwersja polega na wyborze dwóch punktów w permutacji, która jest potem cięta w tych punktach, i odwróceniu kolejności we fragmencie między nimi. A zatem operator ten działa tylko na jednym rodzicu. Na przykład permutacja

$$(1 \ 2 \mid 3 \ 4 \ 5 \ 6 \mid 7 \ 8 \ 9)$$

o dwóch punktach cięcia oznaczonych przez „|” jest przekształcana na

$$(1 \ 2 \ | \ 6 \ 5 \ 4 \ 3 \ | \ 7 \ 8 \ 9)$$

Taka prosta inwersja gwarantuje, że wynikowy potomek jest trasą dopuszczalną. Umożliwia także usunięcie skrzyżowania na trasie, choć może też wprowadzić skrzyżowanie tam, gdzie go nie było. Niektóre doświadczenia dla TSP z 50 miastami pokazały, że inwersja działała lepiej niż algorytm ewolucyjny stosujący operator „krzyżowania i poprawiania” (ang. *cross and correct*) [494]. Zwiększenie liczby punktów cięcia pogarsza jakość działania algorytmu. Inne przykłady, w których stosowano inwersję, podano w [147]. Pod koniec tego rozdziału wrócimy do operatora inwersji i omówimy jego najnowszą wersję – tak zwany operator *inver-over*.

W tym miejscu powinniśmy wspomnieć inne rozwiązania, w których przy tworzeniu potomstwa skorzystano albo z jednego, albo z więcej niż dwóch rodziców. W pracy [213] Herdy prowadził eksperymenty z czterema różnymi operatorami różnicowania:

- *inwersją* – (ang. *inversion*) opisaną powyżej;
- *wstawianiem* – (ang. *insertion*) wybiera się miasto i wstawia w losowe miejsce;
- *przenoszeniem* – (ang. *displacement*) wybiera się fragment trasy i wstawia go w losowe miejsce;
- *wzajemną wymianą* – (ang. *reciprocal exchange*) zamienia się miejscami dwa miasta.

Herdy użył także wersji krzyżowania heurystycznego, w której przy tworzeniu potomstwa bierze udział kilku rodziców. Po losowym wybraniu pierwszego miasta w trasie potomka sprawdza się wszystkich lewych i prawych sąsiadów tego miasta we wszystkich rodzicach. Wybiera się to miasto, do którego odległość jest najmniejsza. Proces ten jest kontynuowany, dopóki nie zakończy się trasy.

Inny sposób kodowania tras i wykonywania na nich operacji polega na zastosowaniu wektora n liczb zmiennopozycyjnych, przy czym n odpowiada liczbie miast. Elementy wektora są posortowane, a ich kolejność określa trasę. Na przykład wektor

$$\mathbf{v} = (2,34, -1,09, 1,91, 0,87, -0,12, 0,99, 2,13, 1,23, 0,55)$$

odpowiada trasie

$$2 - 5 - 9 - 4 - 6 - 8 - 3 - 7 - 1$$

gdyż najmniejsza liczba $-1,09$ jest drugim elementem wektora \mathbf{v} , druga z kolei najmniejsza liczba $-0,12$ jest piątym elementem wektora \mathbf{v} itd. Nowe rozwiązania możemy generować różnymi metodami, w których stosujemy wektory liczb rzeczywistych, ale znowu powiązanie (lub co najmniej intuicja) między reprezentacją a problemem może być stracone.

W większości omawianych do tej pory operatorów braliśmy pod uwagę miasta (tzn. ich pozycje i uporządkowanie), a nie krawędzie – połączenia między miastami. To, co może być istotne, to nie pozycja miasta na trasie, ale raczej jego powiązanie z innymi miastami. Homaifar i Guan w [230] zaproponowali:

Po starannym rozważeniu problemu możemy stwierdzić, że podstawowymi cegiełkami w TSP są krawędzie, a nie reprezentacja pozycji miast. Miasto lub krótka ścieżka na danej pozycji bez informacji o sąsiedztwie ma niewielkie znaczenie przy budowaniu dobrej trasy. Trudno jest przecież twierdzić, że wstawienie miasta a na pozycji 2 jest lepsze niż na pozycji 5. Chociaż jest to skrajny przypadek, podstawowe założenie jest takie, że dobry operator powinien wydobywać z rodziców tyle informacji o krawędziach, ile się da. Założenie to można częściowo wyjaśnić na podstawie wyników doświadczeń, podanych w pracy Olivera [338], które wskazują, że operator OX działa 11% lepiej niż PMX i 15% lepiej niż krzyżowanie cykliczne.

Grefenstette [202] opracował klasę operatorów heurystycznych, w której kładzie się nacisk na krawędzie. Działają one według podanego tu schematu.

1. Wybierz losowo miasto, które będzie bieżącym miastem c w potomku.
2. Wybierz cztery krawędzie (po dwie z każdego rodzica) wchodzące lub wychodzące z bieżącego miasta c w każdym z rodziców.
3. Określ rozkład prawdopodobieństwa na wybranych krawędziach na podstawie ich kosztu. Prawdopodobieństwo dla krawędzi związanej z poprzednio odwiedzonym miastem wynosi 0.
4. Wybierz krawędź. Jeśli przynajmniej jedna krawędź ma niezerowe prawdopodobieństwo, to wybór jest oparty na wcześniej określonym rozkładzie; w przeciwnym razie jest losowo wybierane któreś nieodwiedzone miasto.
5. Miasto na „drugim końcu” wybranej krawędzi staje się bieżącym miastem c .
6. Jeśli trasa jest zakończona, to zatrzymaj się; w przeciwnym razie przejdź do kroku 2.

Jak podano w [202], takie operatory przenoszą około 60% krawędzi z rodziców – oznacza to, że 40% jest wybierane losowo. Ponownie, procedura ta bardziej wygląda jak poszukiwanie na ślepo niż poszukiwanie ewolucyjne.

Z kolei operator *z rekombinacją krawędzi* (ang. *edge recombination* – ER) z [494] przenosi z rodziców do jednego potomka ponad 95% krawędzi. Operator ER bada informacje o krawędziach w trasie. Na przykład dla trasy

(3 1 2 8 7 4 6 9 5)

krawędziami są (3 1), (1 2), (2 8), (8 7), (7 4), (4 6), (6 9), (9 5) oraz (5 3). W końcu w TSP to z krawędziami, a nie z miastami, są związane wartości (odległości). Minimalizowana funkcja oceny to sumaryczny koszt wszystkich

krawędzi wchodzących w skład trasy dopuszczalnej. Pozycja miasta na trasie nie jest ważna – trasy są cykliczne. Kierunek krawędzi także nie jest istotny: obie krawędzie (3 1) i (1 3) wskazują tylko, że miasta 1 i 3 są połączone.

Ogólna koncepcja krzyżowania ER polega na tym, że potomek powinien być tworzony wyłącznie z krawędzi występujących u obu rodziców. Wykonuje się to za pomocą listy krawędzi utworzonej z trasy obu rodziców. Lista krawędzi podaje dla każdego miasta c wszystkie inne miasta, które są z nim połączone w co najmniej jednym z rodziców. Dla każdego miasta c na liście znajdują się co najmniej dwa i co najwyżej cztery miasta. Na przykład dla dwóch rodziców

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

$$p_2 = (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5)$$

lista krawędzi jest następująca:

- Miasto 1:** krawędzie do innych miast: 9 2 4
- Miasto 2:** krawędzie do innych miast: 1 3 8
- Miasto 3:** krawędzie do innych miast: 2 4 9 5
- Miasto 4:** krawędzie do innych miast: 3 5 1
- Miasto 5:** krawędzie do innych miast: 4 6 3
- Miasto 6:** krawędzie do innych miast: 5 7 9
- Miasto 7:** krawędzie do innych miast: 6 8
- Miasto 8:** krawędzie do innych miast: 7 9 2
- Miasto 9:** krawędzie do innych miast: 8 1 6 3

Tworzenie potomka zaczyna się od wybrania z jednego z rodziców początkowego miasta. W [494] autorzy wybierali jedno z miast początkowych (np. 1 lub 4 w naszym przykładzie). Jest wybierane miasto o najmniejszej liczbie krawędzi na liście. Jeśli jest kilka takich miast, to jest wybierane jedno z nich losowo. Zwiększa to szanse na to, że zakończymy trasę ze wszystkimi krawędziami pochodzący od rodziców. Przy losowym wyborze szanse na napotkanie „błędu krawędziowego”, tzn. na natrafienie na miasto, które nie ma dalszych krawędzi, są dużo większe. Założmy, że wybraliśmy miasto 1. Miasto to jest bezpośrednio połączone z trzema innymi miastami: 9, 2 i 4. Spośród tych trzech jest wybierane następne miasto. W naszym przykładzie miasta 4 i 2 mają trzy krawędzie, a miasto 9 cztery. Wybieramy losowo spośród miast 4 i 2. Założmy, że zostało wybrane miasto 4. I znowu, kandydatami na następne miasto budowanej trasy są miasta 3 i 5, gdyż są one bezpośrednio połączone z ostatnim miastem 4. Znowu, wybieramy miasto 5, gdyż ma ono tylko trzy krawędzie (miasto 3 ma cztery krawędzie). Jak dotąd potomek ma następujące elementy:

$$(1 \ 4 \ 5 \ x \ x \ x \ x \ x \ x)$$

Kontynuujemy tę procedurę, aby w końcu otrzymać potomka

$$(1 \ 4 \ 5 \ 6 \ 7 \ 8 \ 2 \ 3 \ 9)$$

który w całości jest zbudowany z krawędzi wziętych z dwóch jego rodziców. Z przeprowadzonej serii eksperymentów wynika, że błędy krawędziowe pojawiają się bardzo rzadko (1–1,5% przypadków) [494].

Krzyżowanie z rekombinacją krawędzi zostało rozszerzona w [452]. Główny pomysł był oparty na obserwacji, że „wspólne podciągi” nie były zachowywane przez krzyżowanie ER. Na przykład, jeśli lista krawędzi zawiera wiersz z trzema krawędziami

Miasto 4: krawędzie do innych miast: 3 5 1

to jedna z tych krawędzi się powtarza. W poprzednim przykładzie byłaby to krawędź (4 5), występująca w obu rodzicach. Jest ona jednak zapisana jak inne krawędzie, np. (4 3) i (4 1), które występują tylko w jednym z rodziców. W zaproponowanym rozwiążaniu [452] modyfikuje się listę krawędzi w ten sposób, że są na niej pamiętane „zaznaczone” miasta:

Miasto 4: krawędzie do innych miast: 3 –5 1

przy czym znak „–” oznacza, że zaznaczone miasto 5 powinno być wymienione na liście dwa razy. W poprzednim przykładzie z dwoma rodzicami

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

$$p_2 = (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5)$$

rozszerzona lista krawędzi jest następująca:

Miasto 1: krawędzie do innych miast: 9 –2 4

Miasto 2: krawędzie do innych miast: –1 3 8

Miasto 3: krawędzie do innych miast: 2 4 9 5

Miasto 4: krawędzie do innych miast: 3 –5 1

Miasto 5: krawędzie do innych miast: –4 6 3

Miasto 6: krawędzie do innych miast: 5 –7 9

Miasto 7: krawędzie do innych miast: –6 –8

Miasto 8: krawędzie do innych miast: –7 9 2

Miasto 9: krawędzie do innych miast: 8 1 6 3

Algorytm tworzący nowego potomka w pierwszej kolejności korzysta z elementów zaznaczonych. Ma to znaczenie tylko w przypadkach, gdy na liście są wymienione trzy krawędzie – w obu pozostałych przypadkach albo żadne miasto nie jest zaznaczone, albo oba są zaznaczone. To rozszerzenie połączone z modyfikacją związaną z dokonywaniem lepszego wyboru, gdy konieczny jest wybór losowy, dodatkowo polepszyło działanie systemu [452].

Operator z rekombinacją krawędzi wskazują, że reprezentacja ścieżkowa może nie wystarczać do przedstawienia ważnych własności tras – dlatego została uzupełniona o listę krawędzi. Czy istnieją jeszcze inne reprezentacje, które są bardziej odpowiednie dla problemu komiwojażera? Pytanie to wymaga, żebyśmy się także zapytali o operatory różnicowania, jakie możemy zastosować do

takiej reprezentacji. Warto poeksperymentować z innymi, być może, niewektorowymi reprezentacjami, choćby po to, żeby przekonać się, co możemy przy ich pomocy odkryć.

Przeprowadzono co najmniej trzy niezależne próby stosowania metod ewolucyjnych do znajdowania rozwiązań dla TSP przy reprezentacji macierzowej [174, 430, 230]. Fox i McMahon [174] zastosowali reprezentację trasy w postaci binarnej macierzy pierwszeństwa M . Element macierzy m_{ij} w i -tym wierszu i j -tej kolumnie zawiera tutaj 1 wtedy i tylko wtedy, gdy miasto i występuje na trasie przed miastem j . Na przykład trasa

$$(3 \ 1 \ 2 \ 8 \ 7 \ 4 \ 6 \ 9 \ 5)$$

jest reprezentowana za pomocą macierzy z rys. 8.1.

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1
4	0	0	0	0	1	1	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1
7	0	0	0	1	1	1	0	0	1
8	0	0	0	1	1	1	1	0	1
9	0	0	0	0	1	0	0	0	0

Rysunek 8.1. Reprezentacja trasy w postaci macierzy. Element m_{ij} macierzy znajdujący się w i -tym wierszu i j -tej kolumnie zawiera 1 wtedy i tylko wtedy, gdy miasto i występuje na trasie przed miastem j

Przy tej reprezentacji przedstawiająca trasę (tzn. porządek liniowy na zbiorze miast) macierz M o wymiarze $n \times n$ ma podane niżej własności.

1. Liczba jedynek wynosi dokładnie $n(n - 1)/2$.
2. $m_{ii} = 0$ dla wszystkich $1 \leq i \leq n$.
3. Jeśli $m_{ij} = 1$ oraz $m_{jk} = 1$, to $m_{ik} = 1$.

Jeśli liczba jedynek w macierzy jest mniejsza niż $n(n - 1)/2$ oraz są spełnione dwa poprzednie wymagania, to miasta są uporządkowane częściowo. Oznacza to, że możemy uzupełnić taką macierz (co najmniej na jeden sposób) tak, żeby uzyskać trasę dopuszczalną. Fox i McMahon [174] proponowali:

Reprezentacja ciągu w postaci macierzy boolowskiej zamyka całą informację o ciągu, włączając w to mikrotopologię poszczególnych połączeń między miastami i makrotopologię poprzedników i następników. Reprezentacja w postaci macierzy boolowskiej może być używana w celu zrozumienia działania istniejących operatorów oraz do

opracowywania nowych, które będzie można zastosować do ciągów, żeby uzyskać pożądane efekty przy zachowaniu niezbędnych własności ciągu.

Opracowane w [174] dwa nowe operatory to *przecięcie* i *złączenie*. Operator przecięcia jest oparty na obserwacji, że znalezienie iloczynu (logicznego) bitów z dwóch macierzy daje macierz, w której: 1) liczba jedynek jest nie większa niż $n(n - 1)/2$ oraz 2) pozostałe dwa wymagania są spełnione. Dzięki temu możemy uzupełnić taką macierz tak, żeby otrzymać trasę dopuszczalną.

Na przykład dwóch rodziców

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \text{ oraz } p_2 = (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5)$$

jest reprezentowanych przez dwie macierze (rys. 8.2).

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	1	1	1	1	1	1
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	1	1	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	1	0	1	0	0	0	0

Rysunek 8.2. Reprezentacja dwóch rodziców w postaci macierzy boolowskich

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0

Rysunek 8.3. Pierwsza faza działania operatora przecięcia

Pierwszy etap przecięcia tych dwóch macierzy daje macierz przedstawioną na rys. 8.3.

Częściowy porządek wynikający z przecięcia wymaga, żeby miasto 1 poprzedzało miasta 2, 3, 5, 6, 7, 8 oraz 9. Miasto 2 poprzedzało miasta 3, 5, 6,

7, 8 oraz 9. Miasto 3 poprzedzało miasto 5. Miasto 4 poprzedzało miasta 5, 6, 7, 8 i 9. Miasta 6, 7 i 8 poprzedzały miasto 9. W trakcie następnego etapu działania operatora przecięcia jest wybierany jeden z rodziców i są dodawane pewne jedynki, które występują tylko w nim, a następnie macierz jest uzupełniana do ciągu w wyniku analizy sum wierszy i kolumn. Na przykład na rys. 8.4 przedstawiono jeden z możliwych wyników operacji z drugiego etapu. Macierz ta reprezentuje trasę (1 2 4 8 7 6 3 5 9).

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	1	1	1	1	1
2	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	0	0	0	1
4	0	0	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	1
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	0	0	0	0	0	0	0

Rysunek 8.4. Ostateczny wynik działania operatora przecięcia

Operator złączenia jest oparty na obserwacji, że podzbiór bitów z jednej macierzy można bezpiecznie połączyć z podzbiorem bitów z drugiej macierzy, pod warunkiem że mają one puste przecięcie. Operator dzieli zbiór miast na dwie rozłączne grupy (w [174] użyto w tym celu specjalnej metody). Dla pierwszej z grup są kopowane bity z pierwszej macierzy, a dla drugiej – z drugiej. Wreszcie macierz jest uzupełniana do pełnego ciągu przez analizę sum wierszy i kolumn (podobną do przeprowadzanej przy operatorze przecięcia). Na przykład dwóch rodziców p_1 i p_2 przy podziale miast $\{1, 2, 3, 4\}$ i $\{5, 6, 7, 8, 9\}$ daje macierz (rys. 8.5), która jest uzupełniana tak jak w wypadku operatora przecięcia.

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	x	x	x	x	x
2	0	0	1	1	x	x	x	x	x
3	0	0	0	1	x	x	x	x	x
4	0	0	0	0	x	x	x	x	x
5	x	x	x	x	0	0	0	0	0
6	x	x	x	x	1	0	0	0	1
7	x	x	x	x	1	1	0	0	1
8	x	x	x	x	1	1	1	0	1
9	x	x	x	x	1	0	0	0	0

Rysunek 8.5. Pierwsza faza działania operatora złączenia

Podane wyniki eksperymentalne dla różnych układów miast (np. losowych, pogrupowanych, ułożonych w koncentryczne okręgi) wykazały, że połączenie operatorów złączenia i przecięcia umożliwiło ulepszanie rozwiązań w kolejnych pokoleniach nawet wówczas, gdy nie była stosowana selekcja elitarna. (Przypomnijmy, że selekcja elitarna zawsze zachowuje najlepsze dotychczas znalezione rozwiązanie). Nie stwierdzono natomiast takiego zachowania ani dla operatora ER, ani PMX. Bardziej szczegółowe porównanie kilku operatorów dwu- i jednoargumentowych (zamiany, cięcia i inwersji) pod względem ich efektywności, złożoności i czasu wykonania podano w [174].

Drugie podejście do wykorzystania reprezentacji macierzowej zaproponował Seniw w [430]. Element macierzy m_{ij} w wierszu i i kolumnie j zawiera 1 wtedy i tylko wtedy, gdy trasa wiedzie bezpośrednio z miasta i do miasta j . Oznacza to, że w każdym wierszu i w każdej kolumnie macierzy jest tylko jeden niezerowy element (dla każdego miasta i istnieje dokładnie jedno miasto odwiedzone tuż przed i i dokładnie jedno odwiedzone tuż po i). Na przykład macierz na rys. 8.6a przedstawia trasę, która prowadzi przez miasta (1 2 4 3 8 6 5 7 9) w tej właśnie kolejności. Zauważmy także, że przy tej reprezentacji unikamy problemu związanego z określeniem miasta początkowego, tzn. rys. 8.6a przedstawia także trasy (2 4 3 8 6 5 7 9 1), (4 3 8 6 5 7 9 1 2) itd.

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1
4	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0
9	0	0	1	0	0	0	0	0	0

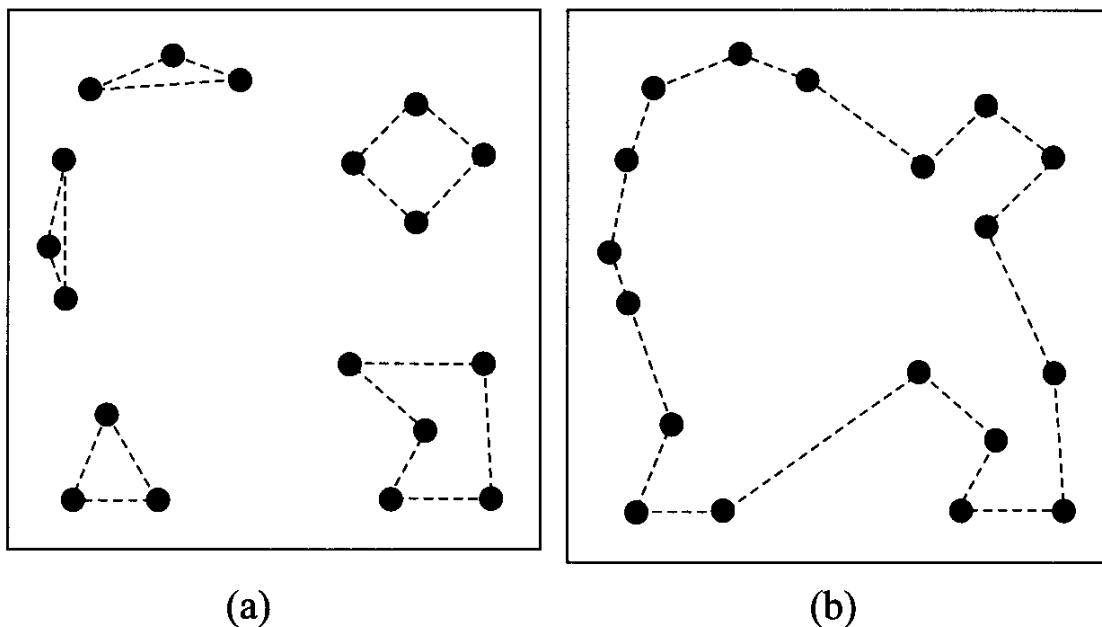
(b)

Rysunek 8.6. Trasy zapisane w postaci macierzy binarnych

Warto zwrócić uwagę, że każda pełna trasa jest reprezentowana przez macierz binarną, dla której w każdym wierszu i w każdej kolumnie jest ustawiony na jedynkę dokładnie jeden bit. Nie każda jednak macierz o takiej własności reprezentuje jedną trasę. Macierze binarne mogą reprezentować kilka rozłącznych podtras. Każda z nich tworzy odrębną pętlę (cykl), nie łącząc się z żadną inną podtrasą w macierzy. Na przykład macierz z rys. 8.6b reprezentuje dwie podtrasy: (1 2 4 5 7) oraz (3 8 6 9).

Takie podtrasy były dopuszczane w nadziei, że będzie występowało naturalne grupowanie. Po zakończeniu algorytmu ewolucyjnego najlepsza macierz jest sprowadzana do jednej trasy przez kolejne łączenie ze sobą par podtras za pomocą algorytmu deterministycznego. Podtrasy złożone z jednego miasta

(trasa wychodzi z miasta, żeby zaraz powrócić do niego), mające długość zero, nie były dopuszczane. Nałożono dolne ograniczenie $q = 3$ na liczbę miast w podtrasie, żeby uniemożliwić algorytmowi ewolucyjnemu sprowadzenie TSP do dużej liczby podtras o bardzo małej liczbie miast.



Rysunek 8.7. Podtrasy wynikające z przykładowego przebiegu algorytmu na pewnej liczbie miast celowo rozłożonych w grupach. Zgodnie z oczekiwaniem algorytm doszedł do kilku rozłącznych podtras (a); trasa po połączeniu podtras (b)

Zdefiniowano dwa operatory różnicowania. Jeden z nich pobierał macierz, losowo wybierał kilka wierszy i kolumn, a następnie usuwał zbiór bitów na przecięciach tych kolumn i wierszy, zastępując usunięte bity losowo w, być może, odmiennej konfiguracji. Rozważmy na przykład trasę z rys. 8.6a reprezentowaną przez

$$(1 \ 2 \ 4 \ 3 \ 8 \ 6 \ 5 \ 7 \ 9)$$

Załóżmy, że do operacji różnicowania wybrano losowo wiersze 4, 6, 7, 9 oraz kolumny 1, 3, 5 i 9. Obliczane są sumy elementów tych wierszy i kolumn. Bitы na przecięciach tych wierszy i kolumn są usuwane i losowo zastępowane, ale wstawione wartości muszą się zgadzać z zapamiętanymi sumami. Innymi słowy, podmacierz odpowiadająca wierszom 4, 6, 7 i 9 oraz kolumnom 1, 3, 5, 8 i 9 z pierwotnej macierzy (rys. 8.8b) jest zastępowana inną podmacierzą (rys. 8.8b). Wynikowa macierz reprezentuje rozwiązanie z dwoma podtrasami

$$(1 \ 2 \ 4 \ 5 \ 7) \text{ oraz } (3 \ 8 \ 6 \ 9)$$

i jest przedstawiona na rys. 8.6b.

Drugi operator zaczyna od macierzy, która ma wszystkie bitы ustawione na zero. Najpierw przegląda dwie macierze wybrane z populacji i gdy napotyka na ustawiony bit w tym samym miejscu (ten sam wiersz i ta sama kolumna) obu rodziców, to ustawia odpowiedni bit w nowej macierzy, która będzie opisywała potomka. Tak wygląda pierwsza faza działania operatora. Następnie

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
1	0	0	0	0

(a)

0	0	1	0	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0

(b)

Rysunek 8.8. Część macierzy przed (a) i po różnicowaniu (b)

operator kopiuje na przemian po jednym ustawnionym bicie z każdego rodzica, aż w żadnym z nich nie będzie już bitów, które można przenieść bez naruszania podstawowych ograniczeń na postać macierzy. To jest druga faza działania operatora. Na koniec w każdym z wierszy potomka, który nie zawiera ustawionego bitu, jest wstawiany losowy bit. Operator ten można uruchamiać wielokrotnie w zależności od liczby potrzebnych potomków, można przy tym zmieniać kolejność macierzy rodziców.

W poniższym przykładzie działania tego operatora zaczynamy od macierzy pierwszego rodzica z rys. 8.9a reprezentującej dwie podtrasy

(1 5 3 7 8) oraz (2 4 9 6)

i macierzy drugiego rodzica (rys. 8.9b) reprezentującej jedną trasę

(1 5 6 2 7 8 3 4 9)

Pierwsze dwie fazy tworzenia pierwszego potomka przedstawiono na rys. 8.10.

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	1	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	1	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

(b)

Rysunek 8.9. Pierwszy (a) i drugi rodzic (b)

Pierwszego potomka wyznaczonego za pomocą tego operatora, po ostatniej fazie, przedstawiono na rys. 8.11. Reprezentuje on podtrasę

(1 5 6 2 3 4 9 7 8)

Drugi potomek reprezentuje podtrasy

(1 5 3 4 9) oraz (2 7 8 6)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0

(b)

Rysunek 8.10. Potomek w czasie wykonywania krzyżowania: (a) po pierwszej fazie i (b) po drugiej fazie

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0
4	0	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	1	0	0
6	0	1	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0
8	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	1	0	0

Rysunek 8.11. Potomek otrzymany za pomocą operatora działającego na dwóch rodzicach po ostatniej fazie

Zauważ, że w obu potomkach występują wspólne segmenty macierzy rodziców.

Procedura ta wykazywała sensowne działanie na wielu przykładach testowych o rozmiarach od 30 do 512 miast, jednak wpływ działania parametru q , minimalnej liczby miast w podtrasie, jest wciąż niejasny. W dodatku procedury łączenia podtras w jedną pełną trasę nie są wcale oczywiste. Metoda jest jednakże podobna do algorytmu rekurencyjnego grupowania Litkego [287], który rekurencyjnie zastępuje grupy rozmiaru B jednym reprezentującym je miastem, aż pozostałe mniej niż B miast. Taki mniejszy problem jest następnie rozwiązywany optymalnie. Wszystkie grupy są rozwijane jedna po drugiej, algorytm ustawa po kolej rozwijane elementy, wstawiając je między dwóch sąsiadów w bieżącej trasie. Podejście to może też być użyteczne przy rozwiązywaniu problemu wielu komiwojażerów, w którym kilku komiwojażerów musi przebyć oddzielne nienakładające się na siebie trasy.

Trzeci sposób stosowania macierzy do reprezentowania rozwiązań TSP został zaproponowany w [230]. Jak w poprzednim podejściu, element m_{ij} macierzy binarnej M jest ustawiony na 1 wtedy i tylko wtedy, gdy istnieje krawędź z miasta i do miasta j ; tutaj jednak zostały użyte inne operatory różnicowania.

Homaifar i Guan w [230] zdefiniowali dwa operatory krzyżowania macierzowego (ang. *matrix crossover* – MX). Operatory te wymieniają wszystkie elementy dwóch macierzy rodziców znajdujące się albo za jednym punktem krzyżowania (krzyżowanie jednopunktowe), albo między dwoma punktami krzyżowania (krzyżowanie dwupunktowe). Wprowadza się jeszcze dodatkowo „algorytm naprawy”, który: 1) usuwa duplikaty, tzn. zapewnia, że każdy wiersz i każda kolumna mają dokładnie jedną jedynkę, oraz 2) rozcina i łączy cykle (jeśli jakieś są), żeby utworzyć trasę dopuszczalną.

Krzyżowanie dwupunktowe zilustrujemy następującym przykładem. Dwie macierze rodziców przedstawiono na rys. 8.12. Reprezentują one dwie dopuszczalne trasy

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9) \text{ oraz } p_2 = (1 \ 4 \ 3 \ 6 \ 5 \ 7 \ 2 \ 8 \ 9)$$

Wybieramy dwa punkty krzyżowania. Są to punkty między kolumnami 2 i 3 (pierwszy punkt) oraz między kolumnami 6 i 7 (drugi punkt). Punkty krzyżowania przecinają macierze pionowo. Dla każdej macierzy pierwsze dwie kolumny stanowią pierwszą część podziału, kolumny 3, 4, 5 i 6 – środkową część, a ostatnie trzy kolumny – trzecią część.

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0

(b)

Rysunek 8.12. Macierze binarne z zaznaczonymi punktami krzyżowania (po dwójne linie)

Po wykonaniu pierwszego kroku dwupunktowego operatora MX w obu macierzach są wymieniane elementy występujące między punktami krzyżowania (tzn. elementy w kolumnach 3, 4, 5 i 6). Wynik pośredni podano na rys. 8.13.

Obaj potomkowie (a) i (b) są niedopuszczalni. Jednak całkowita liczba jedynek w każdej z pośrednich macierzy jest poprawna (tzn. wynosi 9). Pierwszy krok „algorytmu naprawy” przenosi niektóre jedynki w macierzach tak, że

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

(a)

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	1
9	1	0	0	0	0	0	0	0	0

(b)

Rysunek 8.13. Dwie postacie pośrednie potomków po wykonaniu pierwszego kroku operatora MX

każdy wiersz i każda kolumna mają dokładnie jedną jedynkę. Na przykład w potomku na rys. 8.13a podwójne jedynki występują w wierszach 1 i 3. Algorytm może przenieść element $m_{14} = 1$ do m_{84} , a element $m_{38} = 1$ do m_{28} . Podobnie, dla drugiego potomka (rys. 8.13b) podwójne jedynki występują w wierszach 2 i 8. Algorytm może przenieść element $m_{24} = 1$ do m_{34} , a element $m_{86} = 1$ do m_{16} . Po zakończeniu pierwszego kroku algorytmu naprawy pierwszy potomek reprezentuje już trasę dopuszczalną

$$(1 \ 2 \ 8 \ 4 \ 3 \ 6 \ 5 \ 7 \ 9)$$

a drugi potomek reprezentuje trasę, która składa się z dwóch podtras

$$(1 \ 6 \ 5 \ 7 \ 2 \ 8 \ 9) \text{ oraz } (3 \ 4)$$

Drugi krok algorytmu naprawy powinien być zastosowany tylko do drugiego potomka. Na tym etapie procedura rozcina i łączy podtrasy, żeby utworzyć trasę dopuszczalną. W fazie cięcia i łączenia bierze się pod uwagę istniejące krawędzie w pierwotnych rodzicach. Na przykład krawędź (2 4) została wybrana, żeby połączyć wspomniane dwie podtrasy, gdyż krawędź ta występuje w jednym z rodziców. Tak więc pełna trasa (dopuszczalny drugi potomek) to

$$(1 \ 6 \ 5 \ 7 \ 2 \ 4 \ 3 \ 8 \ 9)$$

W pracy [230] zastosowano także jako uzupełnienie krzyżowania MX operator inwersji heurystycznej. Operator odwraca porządek miast między dwoma punktami cięcia, tak samo jak przy prostej inwersji (zobacz wcześniej). Jeśli odległość między dwoma punktami cięcia jest duża (inwersja wysokiego rzędu), to operator przeszukuje połączenia między „dobrymi” ścieżkami, w przeciwnym wypadku wykonuje przeszukiwanie lokalne (inwersja niskiego rzędu). Istnieją dwie różnice między klasyczną a zaproponowaną tutaj inwersją. Pierwsza z nich polega na tym, że wynikowy potomek jest akceptowany tylko wówczas, gdy nowa trasa jest lepsza od pierwotnej. Druga różnica polega na tym, że procedura inwersji wybiera jedno miasto na trasie i szuka ulepszenia, wykonując inwersje

najniższego możliwego rzędu: dwa. Pierwsza inwersja, która daje ulepszenie, jest akceptowana i procedura inwersji kończy się. Jeśli nie ma ulepszenia, to rozważane są inwersje rzędu trzy itd.

Wyniki przedstawione w [230] wskazują, że algorytm ewolucyjny korzystający z dwupunktowego MX i inwersji działa dobrze dla zadań TSP o rozmiarach od 30 do 100 miast. W jednym z eksperymentów wynik uzyskany za pomocą tego algorytmu dla problemu z 318 miastami był tylko o 0,6% gorszy od rozwiązania optymalnego.

8.2. Uzupełnianie o metody poszukiwania lokalnego

Główne wysiłki społeczności naukowców, zajmujących się algorytmami ewolucyjnymi w związku z TSP, były ukierunkowane na wymyślanie odpowiednich reprezentacji połączonych z operatorami rekombinacji, które nie prowadziłyby do powstawania częściowych tras. Operatory działające na pojedynczych rodzicach (tzn. operatory jednoargumentowe) nie wzbudziły takiego zainteresowania. Żadne z dotychczas opisanych podejść nie dorównuje innym heurystykom (np. Lina-Kernighana) i to zarówno pod względem jakości generowanych rozwiązań, jak i czasu ich uzyskiwania. Podejścia oparte na krzyżowania wymagały dużo obliczeń, a podejścia oparte wyłącznie na mutacji (tzn. operatory prostej inwersji czy zamiany) nie zapewniały w efektywny sposób unikania lokalnych optimów przy braku dodatkowych probabilistycznych mechanizmów selekcji.

Te trudności spowodowały, że wiele osób do ewolucyjnego schematu dokładają dodatkowe operatory poszukiwania lokalnego. Tego rodzaju hybrydyzacja była badana już na wczesnych etapach rozwoju algorytmów ewolucyjnych [202]¹. Przy opracowywaniu rozwiązań problemów świata rzeczywistego często użyteczne okazuje się sprawdzenie, jak można połączyć deterministyczne heurystyki ze stochastycznym poszukiwaniem ewolucyjnym. Na rysunku 8.14 przedstawiono możliwą strukturę algorytmu ewolucyjnego rozszerzonego o procedurę lokalnej optymalizacji (dodaliśmy * do wyrażenia „algorytm ewolucyjny”)².

Różnica między podstawowym algorytmem ewolucyjnym (zob. rys. 6.5) a jego rozszerzeniem o lokalny optymalizator jest niewielka, ale znacząca. Każdy osobnik w populacji, czy właśnie zainicjowany, czy utworzony z użyciem jednego lub więcej rodziców, jest poddawany przed oceną lokalnemu ulepszeniu. W ten sposób odkryte lokalne optima zastępują pierwotne osobniki. (Możesz o tym myśleć jako o swego rodzaju ewolucji lamarkowskiej). Przestrzeń przesz-

¹Faktycznie połączenie poszukiwania lokalnego z ewolucyjnym zaproponowali Howard i Kaufman już w 1967 roku [253].

²Termin *algorytm memetyczny* odnosi się do algorytmu, w którym lokalną optymalizację stosuje się do każdego rozwiązania przed dokonaniem jego oceny. Można myśleć, że tego rodzaju algorytm działa tak, jakby algorytm ewolucyjny był stosowany do podprzestrzeni lokalnych optimów, a optymalizacja lokalna działała jako mechanizm naprawiania potomków, którzy znajdują się poza tą podprzestrzenią (tzn. nie są lokalnie optymalne).

```

procedure algorytm ewolucyjny*
begin
     $t \leftarrow 0$ 
    inicjuj  $P(t)$ 
    zastosuj lokalną optymalizację do  $P(t)$ 
    oceń  $P(t)$ 
    while (not warunek zakończenia) do
        begin
             $t \leftarrow t + 1$ 
            wybierz  $P(t)$  spośród  $P(t - 1)$ 
            zmień  $P(t)$ 
            zastosuj lokalną optymalizację do  $P(t)$ 
            oceń  $P(t)$ 
        end
    end

```

Rysunek 8.14. Struktura algorytmu ewolucyjnego z lokalnym optymalizatorem

kiwania algorytmu ewolucyjnego jest ograniczana jedynie do rozwiązań lokalnie optymalnych przy danej funkcji oceny i heurystyce znajdowania lokalnych ulepszeń.

Istnieje wiele sposobów opracowania algorytmów ewolucyjnych, które zawierają operatory poszukiwania lokalnego. W wypadku TSP możemy brać pod uwagę metody Lina-Kernighana, 2-opt, 3-opt itd. Dodatkowo możemy opracować inne operatory działające na jednym lub wielu rodzicach i połączyć je z różnymi metodami selekcji. Jest wiele możliwości do rozważenia.

Pewne wysiłki były nakierowane na stosowanie różnych operatorów krzyżowania (np. operatora MPX [329]) do lokalnie optymalnych osobników [56, 195, 329, 302, 471, 129], tzn. rozwiązań, które są uzyskane w wyniku ulepszenia za pomocą lokalnego poszukiwania. W niektórych z tych badań metody ewolucyjne po rozszerzeniu o operatory lokalne działały lepiej niż algorytmy lokalnego poszukiwania z wieloma punktami początkowymi i dawały rozwiązania bliskie optymalnych dla zadań testowych z 442, 532 i 666 miastami, przy czym odstępstwo od optymalnej długości trasy było mniejsze niż jeden procent.

Przedstawimy niektóre z tych podejść. Mühlenbein i in. w [330] popierali „inteligentną ewolucję” osobników. Ich algorytm

- używa procedury 2-opt do zastąpienia każdej trasy w bieżącej populacji trasą lokalnie optymalną;
- kładzie większy nacisk na rozwiązania wysokiej jakości, umożliwiając im generowanie większej ilości potomstwa;
- używa rekombinacji i mutacji;
- poszukuje minimum za pomocą lokalnego poszukiwania dla każdego osobnika;

- powtarza ostatnie trzy kroki dopóty, dopóki nie zostanie spełniony warunek zatrzymania.

Używają oni wersji krzyżowania z zachowaniem porządku (OX), w którym para rodziców (z dwoma punktami cięcia zaznaczonymi „|”), powiedzmy

$$p_1 = (1 \ 2 \ 3 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 8 \ 9)$$

$$p_2 = (4 \ 5 \ 2 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 9 \ 3)$$

tworzy potomka w następujący sposób. Najpierw segmenty między punktami cięcia są kopowane do potomka:

$$o_1 = (\mathbf{x} \ \mathbf{x} \ \mathbf{x} \ | \ 4 \ 5 \ 6 \ 7 \ | \ \mathbf{x} \ \mathbf{x})$$

$$o_2 = (\mathbf{x} \ \mathbf{x} \ \mathbf{x} \ | \ 1 \ 8 \ 7 \ 6 \ | \ \mathbf{x} \ \mathbf{x})$$

Następnie, zamiast zaczynać od drugiego punktu cięcia w jednym z rodziców, jak to miało miejsce w wypadku OX, miasta z drugiego rodzica są kopowane w tej samej kolejności od początku ciągu z pominięciem symboli, które już się znajdują w potomku:

$$o_1 = (2 \ 1 \ 8 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 9 \ 3)$$

$$o_2 = (2 \ 3 \ 4 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 5 \ 9)$$

Wyniki eksperymentalne dla zadania z 532 miastami okazały się zachęcające, gdyż za pomocą tej metody znaleziono rozwiązanie o 0,06% gorsze od optymalnego (znalezionego przez Padberga i Rinaldiego [342]).

Warto wspomnieć o jeszcze jednym podejściu. Craighurst i Martin [81] skoncentrowali się na badaniu powiązania między zapobieganiem „kazirodztwu” a jakością działania algorytmu ewolucyjnego dla TSP. Oprócz wprowadzenia 2-opt jako operatora lokalnego wprowadzili dodatkowy pomysł, polegający na tym, że rozwiązanie nie może wchodzić w rekombinację z innymi rozwiązaniami z nim „spokrewnionymi”. Prawo dotyczące kazirodztwa k -tego stopnia zabraniało krzyżowania się osobników o $k - 1$ stopniu pokrewieństwa. To znaczy dla $k = 0$ ograniczenia nie występują. Dla $k = 1$ osobnik nie może łączyć się w parę ze sobą. Dla $k = 2$ nie może łączyć się w parę ze sobą, ze swoimi rodzicami, dziećmi, rodzeństwem itd. Przeprowadzono szereg eksperymentów z użyciem sześciu zadań testowych wziętych z TSPLIB o rozmiarach od 48 do 101 miast. Wyniki wskazywały na silną i interesującą zależność między prawami zabraniającymi kazirodztwa a częstością mutacji. Dla małych częstości mutacji zabranianie kazirodztwa poprawiało wyniki; jednakże dla większych częstości mutacji znaczenie mechanizmu zakazu kazirodztwa malało, aż w końcu mechanizmy te zaczęły wpływać negatywnie na jakość. Wypróbowywano również inne prawa zapobiegania kazirodztwu, w których podobieństwo między dwoma osobnikami było mierzone jako stosunek różnicy między całkowitą liczbą krawędzi w rozwiązaniu a liczbą krawędzi wspólnie dzielonych do liczby wszystkich krawędzi. Ciekawe, reguły te nie miały dużego wpływu na różnorodność populacji. Czasami rezultaty okazują się sprzeczne z intuicją. Szersze omówienie znajduje się w [81].

8.3. Inne możliwości

Valenzuela i Jones [473] zaproponowali interesujące rozszerzenie algorytmów ewolucyjnych oparte na metodzie dziel i rządź z rozwiązujących TSP algorytmów Karpa-Steele'a [251, 453]. Przedstawiony algorytm ewolucyjny dziel i rządź (ang. *evolutionary divide and conquer* – EDAC) może być zastosowany do dowolnego problemu, w którym pewna wiedza na temat dobrych rozwiązań podproblemów jest użyteczna przy tworzeniu rozwiązania globalnego. Zastosowali oni tę metodę dla geometrycznego TSP (tzn. takiego, w którym dla odległości między miastami jest spełniona nierówność trójkąta: $dist(A, B) + dist(B, C) \geq dist(A, C)$ dla wszystkich miast A, B i C).

Można tutaj rozważyć różne metody bisekcji. W metodach tych tnie się prostokąt z n miastami na dwa mniejsze prostokąty. Na przykład w jednej z tych metod dzieli się zadanie, przecinając obszar prostokąta dokładnie na pół linią równoległą do krótszego boku. W drugiej metodzie cięcie odbywa się przez $\lfloor n/2 \rfloor$ miasto, licząc od krótszego boku, dzięki czemu dysponujemy miastem „wspólnym” dla dwóch prostokątów. Końcowe podproblemy są dość małe (zwykle mają od 5 do 8 miast) i stosunkowo łatwe do rozwiązania. Wybrano tutaj procedurę 2-opt ze względu na jej szybkość i prostotę. Użyty dalej algorytm latania (tzn. naprawy) następuje pewne krawędzie w dwóch rozłącznych trasach, aby uzyskać jedną większą trasę. Główna rola algorytmu ewolucyjnego polega na określeniu kierunku podziału na każdym etapie (pionowo czy poziomo).

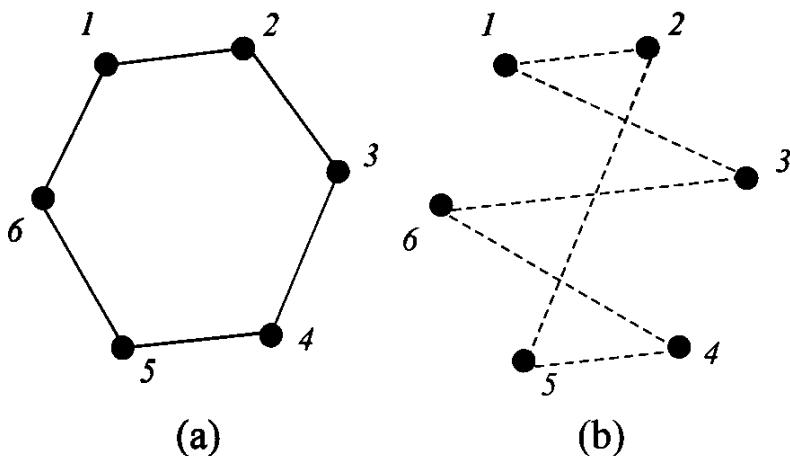
Istnieją także jeszcze inne możliwości. W [474] użyto systemu ewolucyjnego do ulepszenia prostego algorytmu heurystycznego dla TSP. Zadanie ewolucji polegało na zakłócaniu współrzędnych miasta. Podano wyniki działania dla problemów o rozmiarach do 500 miast. W [306] omówiono nowy operator poszukiwania lokalnego, w [297] wprowadzono nową metodę selekcji, a także nowe operatory krzyżowania (krzyżowanie z wymianą krawędzi – ang. *edge exchange crossover*, EEX, oraz krzyżowanie z wymianą podtras – ang. *subtour exchange crossover*, SX).

Zakończymy ten rozdział opisem dwóch niedawno wprowadzonych metod ewolucyjnych rozwiązywania TSP. Pierwsza z nich, zaproponowana przez Nagatę i in. [334], wprowadza lokalne poszukiwanie w ciekawy sposób, który nie jest tak jawnym jak w innych systemach. Drugie podejście to „czysty” algorytm ewolucyjny (tzn. nie występuje tutaj żadna metoda lokalnego poszukiwania), który jest oparty na nowym operatorze łączącym inwersję i krzyżowanie. Oba systemy mają pewne cechy wspólne.

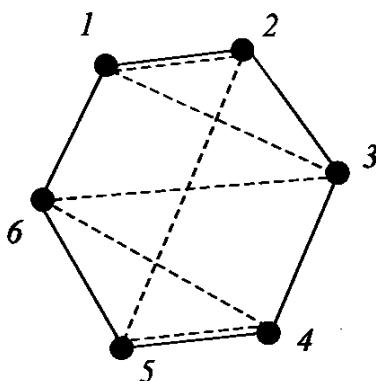
8.3.1. Krzyżowanie ze składaniem krawędzi

Nagata i in. w [334] przedstawili nowy operator krzyżowania ze składaniem krawędzi (ang. *edge assembly crossover* – EAX). Proces tworzenia potomka jest tutaj dość złożony i wymaga wykonania kilku kroków. Założymy, że wybrano dwóch rodziców: rodzica A i rodzica B (rys. 8.15). W pierwszej kolejności jest tworzony graf G zawierający krawędzie obu rodziców (rys. 8.16). Na podstawie

grafu G jest tworzony zbiór tak zwanych AB -cykli. Taki AB -cykl jest określony jako parzystej długości podcykl w G z krawędziami pochodząymi na przemian z A i B . Zwróć uwagę, że AB -cykl może powtarzać miasta, ale nie krawędzie. Po dodaniu jednej krawędzi sprawdzamy, czy AB -cykl jest obecny jako podzbiór wybranych krawędzi. Zauważ także, że niektóre z początkowych krawędzi wybranych do budowy AB -cyklu mogą nie znaleźć się w końcowym AB -cyklu. Stwierdzenia te zilustruje dalszy ciąg opisu naszego przykładu.



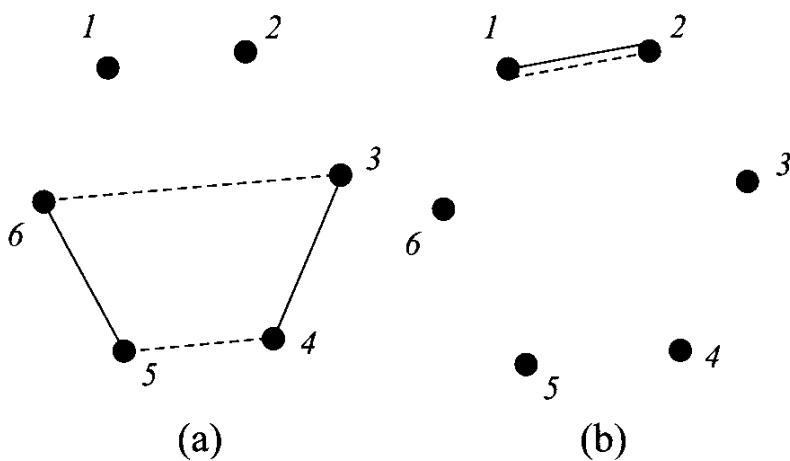
Rysunek 8.15. Rodzice A (a) i B (b) dla zadania TSP z sześcioma miastami



Rysunek 8.16. Graf G : zawiera krawędzie rodzica A i rodzica B

Załóżmy, że zaczynamy od miasta 6 i że pierwsza krawędź AB -cyklu to (6 1). Krawędź ta pochodzi z rodzica A , a zatem następna wybrana krawędź powinna pochodzić z rodzica B i zaczynać się w mieście 1. Powiedzmy, że wybieramy krawędź (1 3). Kolejne wybrane krawędzie to po kolej (3 4) z rodzica A , (4 5) z rodzica B , (5 6) z rodzica A oraz (6 3) z rodzica B . W tym momencie mamy już pierwszy AB -cykl (rys. 8.17a). Zauważ, że pierwsze dwie krawędzie (6 1) i (1 3) nie są tu obecne. Krawędzie znalezionego AB -cyklu są usuwane z grafu G i rozpoczyna się poszukiwanie następnego cyklu. Zauważ, że w trakcie trwania tego procesu możemy natrafić na „nieudane” AB -cykle, które składają się tylko z dwóch miast (rys. 8.17).

Następny krok polega na wybraniu podzbioru AB -cykli (zwanego E -zbiorem). Nagata i in. [334] rozważali dwa mechanizmy selekcji: 1) konkretną deterministyczną metodę heurystyczną oraz 2) metodę losową, w której każdy AB -cykl był wybierany z prawdopodobieństwem 0,5.



Rysunek 8.17. AB -cykl (a) i nieudany AB -cykl, składający się tylko z dwóch miast (b)

Po ustaleniu E -zbioru jest tworzony pośredni potomek C zgodnie z poniższą procedurą:

```

set  $C \leftarrow A$ 
for każda krawędź  $e \in E$ -zbiór do
    if  $e \in A$  then  $C \leftarrow C - \{e\}$ 
    if  $e \in B$  then  $C \leftarrow C \cup \{e\}$ 

```

Pośredni potomek C reprezentuje zbiór rozłącznych podtras, które łącznie przechodzą przez wszystkie miasta (tak jak to miało miejsce we wcześniej omawianych metodach, zob. rys. 8.7). Na tym etapie pośrednie dziecko jest przekształcane w jedną trasę dopuszczalną za pomocą prostego algorytmu zachłannego. Ta procedura zachłanna zawiera „poszukiwanie lokalne”, gdyż podczas jej wykonywania są rozważane długości wielu połączeń i jest wybierane najlepsze połączenie. A konkretne, procedura ta zaczyna od podtrasy T o najmniejszej liczbie krawędzi. Następnie wybiera dwie krawędzie: jedną z wybranej podtrasy T i drugą spośród wszystkich krawędzi w pozostałych podtrasach. Założymy, że

$$(v_q v_{q+1}) \in T \text{ oraz } (v'_r v'_{r+1}) \notin T$$

(wartości $q+1$ i $r+1$ są obliczane modulo liczba wierzchołków w odpowiednich podtrasach). Usuwając te dwie krawędzie, redukujemy koszt o

$$\text{cięcie}(q, r) = L(v_q, v_{q+1}) + L(v'_r, v'_{r+1})$$

przy czym $L(a, b)$ oznacza długość krawędzi (a, b) . Następnie zastępujemy je krótszym połączeniem:

$$\text{połączenie}(q, r) = \min\{L(v_q, v'_r) + L(v_{q+1}, v'_{r+1}), L(v_q, v'_{r+1}) + L(v_{q+1}, v'_r)\}$$

Szukamy zatem

$$\min_{r,q} \{\text{połączenie}(q, r) - \text{cięcie}(q, r)\}$$

W pracy [334] użyto jeszcze dodatkowej heurystyki redukującej liczbę rozważanych par krawędzi. Po utworzeniu połączenia liczba podtras jest zmniejszana o jeden i proces jest powtarzany (tzn. jest wybierana podtrasa o najmniejszej liczbie krawędzi). W końcu jest tworzona dopuszczalna trasa, która staje się właściwym potomkiem rodziców A i B . System korzystający z EAX dał wyniki, które były o mały ułamek procenta większe od optimum dla zadań testowych o rozmiarach od 100 do ponad 3000 miast.

Przedstawiony algorytm ma wiele interesujących cech [334]. Po pierwsze należy nadmienić, że zastosowano bardzo silny nacisk selekcyjny. Potomek następuje jednego z rodziców tylko wówczas, gdy jest lepszy od obu! Jeśli potomek nie jest lepszy od obu rodziców, to operator krzyżowania ze składaniem krawędzi jest stosowany wielokrotnie (do $N = 100$ razy), żeby zwiększyć szanse na wygenerowanie potomka lepszego od rodziców. EAX przy każdej próbie może wygenerować innego potomka, gdyż przy pierwszej próbie potomek jest generowany za pomocą deterministycznej, heurystycznej metody wyboru elementu E -zbioru, a przy dalszych próbach (jeśli są konieczne) jest stosowana metoda losowego wyboru elementu tego zbioru. Ta metoda wielokrotnego generowania dziecka (ang. *iterative child generation* – ICG) zwiększa prawdopodobieństwo znalezienia ulepszonego potomka dla tego samego zbioru rodziców.

Algorytm ten nie używa bezpośrednio żadnej podobnej do procedury 2-opt metody lokalnego poszukiwania; operator EAX łączy podtrasy, wykorzystując informację lokalną przy wyborze krawędzi, jakie mają być używane do rozcinania i łączenia. Co więcej, operator EAX wprowadza niejawną mutację, dokładając do potomka nowe krawędzie w trakcie trwania procesu łączenia podtras. W pracy [485] próbowało ocenić wpływ poszczególnych składników tej metody na jakość końcowych rozwiązań:

Do potomków są wprowadzane krawędzie, które nie pojawiają się w rodzicach lub nawet w całej populacji. Jesteśmy w tej chwili przekonani, że dobry operator dla TSP musi mieć możliwość wprowadzania nowych krawędzi do potomków.

Jest to istotne odejście od typowego sposobu tworzenia operatorów różnicowania. We wszystkich początkowych podejściach (podrozdział 8.1) próbowało zachować jak najwięcej krawędzi z pierwotnych rodziców!

W kolejnym punkcie opiszemy kolejny algorytm rozwiązujący TSP. Zachowuje on wiele cech powyższego algorytmu, takich jak konkurowanie potomstwa ze swoimi rodzicami oraz wprowadzanie nowych krawędzi, nie korzysta jednak z żadnych lokalnych informacji.

8.3.2. Operator inver-over

Przypuśćmy, że chcielibyśmy utworzyć „czysty” algorytm ewolucyjny, radzący sobie z TSP. To znaczy, nasz algorytm w ogóle nie będzie używał lokalnego poszukiwania. Jak moglibyśmy zaprojektować taką procedurę, żeby mogła konkurować z metodami lokalnymi? Po przyjrzeniu się wielu próbom ewoluowania rozwiązań możemy dojść do wniosku, że TSP powinno być rozwiązywane

z użyciem stosunkowo silnego nacisku selekcyjnego. Moglibyśmy też chcieć dodać efektywny obliczeniowo operator różnicowania, który daje potomków z zachowaniem możliwości uciekania z lokalnych optimów. Operatory jednoargumentowe wymagają mniej czasu niż operatory działające na dwóch rodzicach, moglibyśmy jednak chcieć połączyć zalety obu tych rodzajów operatorów.

Mając to na względzie, zastanówmy się nad następującym, podanym w [463], algorytmem ewolucyjnym dla TSP. Ma on następujące cechy:

- każdy osobnik konkuje wyłącznie ze swoimi potomkami;
- jest tylko jeden operator różnicowania, ale ten operator *inver-over* dostosowuje się do sytuacji;
- liczba zastosowań operatora do osobnika w jednym pokoleniu jest zmienna.

Algorytm ten możemy postrzegać jako zestaw równoległych procedur wspinania się, które są podobne do algorytmu Lina-Kernighana. Każdy ze wspinających się wątków wykonuje zmienną liczbę zamian krawędzi. Ten operator *inver-over* ma pewne elementy adaptacyjne: 1) liczbę inwersji wykonywanych na jednym osobniku i 2) segment podlegający inwersji jest określany przy użyciu innego losowo wybranego osobnika. Możemy zatem patrzeć na ten algorytm jak na algorytm ewolucyjny z silnym naciskiem selekcyjnym oraz z samodostosowującym się operatorem różnicowania.

Na rysunku 8.18 znajduje się bardziej szczegółowy opis całego algorytmu *inver-over*, a także zaproponowanego tutaj operatora. Z małym prawdopodobieństwem¹ p drugi punkt inwersji jest wybierany losowo z tego samego osobnika. Element ten jest konieczny. Bez możliwości tworzenia nowych połączeń algorytm szukałby tylko wśród połączeń obecnych już w populacji początkowej. Jeśli $rand() > p$, to informacja potrzebna do wykonania inwersji pochodzi z innego losowo wybranego osobnika. Dzięki temu operator inwersji faktycznie przypomina krzyżowanie, gdyż część wzoru (co najmniej dwa miasta) drugiego osobnika pojawia się w potomku.

Zilustrujmy jedną iterację tego operatora na następującym przykładzie. Założymy, że bieżący osobnik S' to

$$S' = (2 \ 3 \ 9 \ 4 \ 1 \ 5 \ 8 \ 6 \ 7)$$

a bieżące miasto c to 3. Jeśli wygenerowana liczba losowa $rand()$ nie przekracza p , to z tego samego osobnika S' wybierane jest inne miasto c' (powiedzmy, że c' to 8) i odpowiedni segment jest odwracany, co daje potomka

$$S' \leftarrow (2 \ 3 \ 8 \ 5 \ 1 \ 4 \ 9 \ 6 \ 7)$$

Pozycje punktów cięcia dla wybranego segmentu są umiejscowione za miastami 3 i 8. W odwrotnej sytuacji (tzn. gdy $rand() > p$) z populacji jest wybierany

¹Co ciekawe, doświadczenia pokazały, że wartość ta jest niezależna od liczby miast w zadaniu testowym. Funkcja $rand()$ z rys. 8.18 generuje losową liczbę zmiennopozycyjną z przedziału [0..1].

```

procedure inver-over
  losowe inicjowanie populacji  $P$ 
  while (niespełniony warunek zakończenia) do
    begin
      for każdy osobnik  $S_i \in P$  do
        begin
           $S' \leftarrow S_i$ 
          wybierz (losowo) miasto  $c$  z  $S'$ 
          repeat
            if ( $rand() \leq p$ )
              wybierz miasto  $c'$  z pozostałych miast w  $S'$ 
            else
              wybierz (losowo) osobnika z  $P$ 
              przypisz do  $c'$  miasto „za” miastem  $c$  w wybranym osobniku
              if (następne lub poprzednie miasto w stosunku do  $c$  w  $S'$  to  $c'$ )
                exit z pętli repeat
              odwróć w  $S'$  ciąg od miasta za  $c$  do miasta  $c'$ 
               $c \leftarrow c'$ 
              if ( $eval(S') \leq eval(S_i)$ )
                 $S_i \leftarrow S'$ 
        end
    end

```

Rysunek 8.18. Zarys algorytmu inver-over

losowo inny osobnik. Przypuśćmy, że jest to $(1\ 6\ 4\ 3\ 5\ 7\ 9\ 2\ 8)$. Wyszukujemy w tym osobniku miasto c' , które jest „za” miastem 3 (w tym wypadku jest to miasto 5). A zatem segment podlegający inwersji w S' zaczyna się za miastem 3 i kończy za miastem 5. W związku z tym nowy potomek to

$$S' \leftarrow (2\ 3\ 5\ 1\ 4\ 9\ 8\ 6\ 7)$$

Zauważ, że podciąg $3 - 5$ przyszedł z „drugiego rodzica”. Zauważ też, że w każdym wypadku ciąg wynikowy jest pośredni w tym sensie, że powyższy operator inwersji jest stosowany wiele razy, zanim zostanie wykonana ocena. Proces ten kończy się, gdy c' , miasto za bieżącym miastem c w losowo wybranym osobniku, jest także miastem „obok” c w modyfikowanym osobniku. Powiedzmy, że po kilku inwersjach bieżący osobnik S' to

$$S' = (9\ 3\ 6\ 8\ 5\ 1\ 4\ 2\ 7)$$

a bieżące miasto c to 6. Jeśli $rand() > p$, to z losowo wybranego osobnika w populacji uzyskujemy miasto „za” miastem 6. Założmy, że miastem tym jest 8 (jeśli $rand() \leq p$, to jest wybierane losowe miasto, a więc i tak może się zdarzyć, że miastem tym jest 8). Ponieważ miasto 8 już znajduje się za miastem 3, więc ciąg inwersji się kończy.

Wyniki przedstawione w [463] pokazują efektywność tego algorytmu: 1) dla zadania testowego ze 144 miastami średnie rozwiązanie było tylko 0,04% powyżej minimum, 2) dla zadania testowego z 442 miastami było 0,63% powyżej minimum, 3) dla zadania testowego z 2392 miastami było 2,66% gorsze. Co więcej, dla losowo wybranego zadania testowego z 10 000 miast średnie rozwiązanie pozostawało w odległości 3,56% od dolnej granicy Helda-Karpa, podczas gdy najlepsze rozwiązanie znalezione przy testowych 10 uruchomieniach algorytmu było o mniej niż 3% powyżej tej dolnej granicy. Czas działania algorytmu też był sensowny: wynosił kilka sekund dla problemów do 105 miast, poniżej 3 minut dla zadania z 442 miastami i poniżej 90 minut dla zadania z 2392 miastami.

Na podstawie tych eksperymentów można poczynić kilka interesujących spostrzeżeń.

- Zaproponowany system jest prawdopodobnie najszybszym dotychczas opracowanym algorytmem ewolucyjnym dla TSP. Wszystkie inne algorytmy, które są oparte na operatorach krzyżowania dają o wiele gorsze wyniki w znacznie dłuższym czasie.
- Zaproponowany system ma tylko trzy parametry: rozmiar populacji, prawdopodobieństwo p wygenerowania losowej inwersji oraz liczbę iteracji w warunku zakończenia. Większość innych systemów ewolucyjnych ma wiele dodatkowych parametrów, a każdy z nich wymaga przemyślenia, jak go ustawić.
- Precyzja i stabilność algorytmu ewolucyjnego jest dość duża dla relatywnie małych zadań testowych (prawie 100% dokładności dla każdego rozważanego zadania testowego o wielkości do 105 miast), czas obliczania też był dopuszczalny (3–4 sekundy).
- System wprowadza nowy, interesujący operator, który łączy cechy inwersji (lub mutacji) i krzyżowania. Przedstawione w tym podrozdziale wyniki eksperymentów wskazują, że algorytm ewolucyjny inver-over dał wyniki, które są znaczco lepsze od wyników dawanych przez algorytmy korzystające z losowej inwersji.
- Parametr prawdopodobieństwa p (który we wszystkich eksperymentach był utrzymywany na stałym poziomie 0,02) określa proporcje inwersji wykonywanych na ślepo i inwersji sterowanych (dostosowanych do problemu).

8.4. Podsumowanie

Problem komiwojażera jest podstawowym przykładem tego, jak na pierwszy rzut oka prosty problem może rozkwitnąć mnogością niespodziewanych wyzwań. Wydaje się, że znalezienie uporządkowanej listy miast o takiej własności, żeby całkowita długość ścieżki była minimalna, nie jest tak trudne. Dlaczego miałyby to być trudne? A jednak TSP, jak wiele innych problemów w informatyce, jest problemem NP-trudnym. Nie dysponujemy algorytmami wielo-

mianowymi, które mogą generować doskonałe rozwiązania. Możemy zdać się na wyliczeniowe wygenerowanie rozwiązania problemu, ale tylko dla małych, trywialnych zadań. Czasami możemy zastosować heurystyki podobne do metod podziału i ograniczeń, ale nawet wtedy musimy mieć możliwość znalezienia jakiejś metody, która znajduje pełne rozwiązanie dla TSP i której złożoność rośnie wielomianowo w zależności od liczby miast w zadaniu. Jeśli umiałbyś opracować taki algorytm, to nie tylko otrzymałbyś wiele nagród od różnych informatycznych i matematycznych towarzystw, ale Twoje nazwisko zostałoby na zawsze zapisane w podręcznikach historii! A przy okazji zarobiłbyś na tym jeszcze trochę pieniędzy.

W związku z brakiem doskonałych algorytmów, które generują odpowiedzi na tyle szybko, żeby były przydatne, poświęcono wiele wysiłków, żeby szybko generować rozwiązania bliskie optymalnych. Algorytmy ewolucyjne były intensywnie badane pod tym kątem. Z TSP wynikają proste kodowanie problemu oraz funkcja oceny. Po wybraniu reprezentacji główną trudność stanowi określenie operatorów różnicowania.

Jak przekonaliśmy się w tym przeglądzie, najwięcej uwagi poświęcono operatorom rekombinacji działającym na dwóch rodzicach. Te wizjonerskie wysiłki czynione w tym kierunku są naprawdę imponujące. Jest to jeden z najwspanialszych aspektów obliczeń ewolucyjnych. Metoda ta jest tak elastyczna, że pasuje do wielu reprezentacji i funkcji oceny. Z łatwością możemy określić, co jest przydatne w danym problemie. Z taką samą łatwością możemy wymyślać operatory różnicowania, przekształcające rozwiązania-rodziców w potomków. Bez konieczności kłopotania się o gładkość, ciągłość czy o podwójną różniczkowalność funkcji oceny i bez konieczności kodowania rozwiązań w jakiejś ustalonej strukturze danych, masz do dyspozycji nieograniczone możliwości okazania swojej kreatywności.

Dlaczego tak wiele wysiłku poświęcono stosowaniu rekombinacji? W istocie jest to historyczna zaszłość. Nacisk na operatory różnicowania działające na dwóch rodzicach wydaje się wynikać z tego, że wcześniej naukowcy skupiali się też na tych operatorach. Jest coś „seksownego” w tym, że rozwiązania zapisane w komputerze mają w sobie coś „płciowego”, nie oznacza to jednak, że operatory różnicowania oparte na tego rodzaju bazie są najlepsze. Przy dużych wysiłkach poświęconych na wymyślanie sposobów krzyżowania (tylko) par tras-rodziców dla TSP, pomijano inne użyteczne rodzaje operatorów różnicowania (np. inver-over). Możliwość wykorzystania więcej niż dwóch rodziców przy generowaniu potomstwa budziła stosunkowo małe zainteresowanie naukowców [115, 119]¹. Doświadczenia pokazały, że stosowanie więcej niż dwóch rodziców może dla pewnych problemów dać korzyść, a także, że nawet całkowite odrzucenie rekombinacji może mieć zalety [112].

Najbardziej istotne jest to, że jako rozwiązyjący problem musisz wziąć odpowiedzialność za własne podejście do jego rozwiązania. Nie wystarczy tutaj powiedzieć: „Cóż, ten i ten użył algorytmu xyz, a więc i ja tak zrobię”. Albo

¹Pomysł stosowania więcej niż dwóch rodziców w rekombinacji pojawił się w rzeczywistości w latach sześćdziesiątych XX wieku [253, 58, 161].

jeszcze gorzej: „Każdy używa algorytmu xyz, a więc jak ja go nie użyję, to będę głupi”. W istocie takie podejście to prawdziwy grzech ciężki. Twoim obowiązkiem jest poszukiwanie lepszych sposobów rozwiązywania problemów i pytanie się o założenia, jakie inni poczynili w poprzednich podejściach.

Dzięki temu będziesz mógł wymyślić nowe metody radzenia sobie z problemami. Musisz zawsze myśleć o sposobach uwzględnienia w algorytmie poszukującym rozwiązania wiedzy na temat problemu. Być może, istnieją operatory lokalne, które da się zaadaptować. Być może, algorytm ewolucyjny można zastosować do uciekania z optimów lokalnych. Być może, możesz wynaleźć jakiś nowy operator, jak inver-over. Być może, żadne poszukiwanie nie jest potrzebne, gdyż można znaleźć dokładny wzór na postać rozwiązania! Być może...

IX. Kto ma zebrę?

W świecie rzeczywistym często spotykamy tak zwane *problemy spełniania ograniczeń*, które polegają na znalezieniu dopuszczalnego rozwiązania. Innymi słowy, nie musimy niczego optymalizować, ale za to musimy spełnić wszystkie ograniczenia problemu.

Jednym z najbardziej znanych przykładów problemu spełniania ograniczeń jest słynny problem zebry. Brzmi on następująco: jest pięć domów, każdy innego koloru, zamieszkałych przez ludzi różnych narodowości, którzy mają różne zwierzęta domowe, ulubione napoje i samochody. Ponadto:

1. Anglik mieszka w czerwonym domu.
2. Hiszpan ma psa.
3. Mieszkaniec zielonego domu pija kakao.
4. Ukrainiec pija ajerkoniak.
5. Zielony dom stoi bezpośrednio po (Twojej) prawej stronie od beżowego domu.
6. Właściciel garbusa hoduje ślimaki.
7. Posiadacz forda mieszka w żółtym domu.
8. Mieszkaniec środkowego domu pija mleko.
9. Norweg mieszka w pierwszym domu od lewej.
10. Właściciel chevroleta i hodowca lisa mieszkają w sąsiednich domach.
11. Dom posiadacza forda stoi obok domu, gdzie trzymany jest koń.
12. Właściciel mercedesa pija sok pomarańczowy.
13. Japończyk jeździ volkswagenem.
14. Norweg mieszka obok niebieskiego domu.

A pytanie brzmi: kto ma zebrę? I jeszcze: kto pija wodę?

Kilka zdań dostarcza nam bezpośrednich informacji. Zdanie 8, na przykład, mówi nam, że mleko pija mieszkaniec domu 3 (tzn. środkowego). Zdanie 9 informuje nas, że Norweg mieszka w domu 1 (tzn. pierwszym domu od lewej). Zdanie 14 wskazuje, że dom 2 jest niebieski (ponieważ jest to jedyny dom stojący obok domu Norwega). Tak więc nasza dotychczasowa wiedza przedstawia się następująco:

Dom	1	2	3	4	5
Kolor			niebieski		
Napój				mleko	
Narodowość		Norweg			
Samochód					
Zwierzę					

Dwa inne zdania są bardzo interesujące. Zdanie 1 mówi nam, że Anglik zamieszkuje czerwony dom, a ze zdania 5 dowiadujemy się, że zielony dom stoi bezpośrednio po prawej stronie beżowego domu. Wnioski są następujące:

- dom 1 nie jest czerwony (ponieważ mieszka w nim Norweg);
- kolory beżowy i zielony mają odpowiednio albo domy 3 i 4, albo 4 i 5.

Oznacza to, że dom 1 musi być żółty. Skoro tak, to Norweg posiada fordą (zdanie 7), a koń jest hodowany w domu 2 (zdanie 11). Wiemy teraz, że:

Dom	1	2	3	4	5
Kolor	żółty		niebieski		
Napój				mleko	
Narodowość		Norweg			
Samochód		ford			
Zwierzę			koń		

Możemy teraz rozważyć dwa przypadki kolorów domów, ponieważ istnieją tylko dwie możliwości dla domów 3, 4 i 5:

- przypadek 1: (beżowy, zielony, czerwony);
- przypadek 2: (czerwony, beżowy, zielony).

Przyjrzyjmy się im po kolej. W pierwszym przypadku możemy wywnioskować następujące dodatkowe informacje: Anglik mieszka w domu 5, ponieważ dom jest czerwony (zdanie 1), kakao pija mieszkaniec domu 4, ponieważ dom jest zielony (zdanie 3), Ukrainiec pija ajerkoniak (zdanie 4), musi więc mieszkać w domu 2 (gdyż Norweg i Anglik mieszkają w domach 1 i 5, a mleko i kakao konsumują mieszkańcy domów 3 i 4), Anglik ma mercedesa i pija sok pomarańczowy (zdanie 12, ponieważ sok jest lubiany przez mieszkańca domu 1 lub 5, ale Norweg z domu 1 jest posiadaczem forda). W takim razie przypadek 1 wygląda tak:

Dom	1	2	3	4	5
Kolor	żółty	niebieski	beżowy	zielony	czerwony
Napój		ajerkoniak	mleko	kakao	sok pomar.
Narodowość	Norweg	Ukrainiec			Anglik
Samochód	ford				mercedes
Zwierzę		koń			

Teraz wiemy, że ten przypadek nie prowadzi do rozwiązania! Żeby to zobaczyć, spróbujmy zbadać, kto jest właścicielem garbusa. Nie jest nim Norweg ani Anglik, ponieważ znamy już marki posiadanych przez nich samochodów. Nie może nim być Japończyk, gdyż wiemy ze zdania 13, że jeździ volkswagenem. Nie jest nim też Ukrainiec, ponieważ właściciel garbusa hoduje ślimaki (zdanie 6), a Ukrainiec ma konia. Hiszpan też nie może go mieć, ponieważ jest właścicielem psa (zdanie 2). I tak doszliśmy do sprzeczności. W przypadku 1 niemożliwe jest spełnienie pozostałych ograniczeń problemu. Prawdziwy musi więc być przypadek 2. Wiemy już, że domy 3, 4 i 5 mają odpowiednio kolory czerwony, beżowy i zielony. Zdania 1 i 3 dostarczają nam nowych bezpośrednich informacji, a więc nasza aktualna wiedza przedstawia się następująco:

Dom	1	2	3	4	5
Kolor	żółty	niebieski	czerwony	beżowy	zielony
Napój			mleko		kakao
Narodowość	Norweg			Anglik	
Samochód	ford				
Zwierzę		koń			

Możemy teraz rozumować w ten sposób: skoro Ukrainiec pija ajerkoniak (zdanie 4), musi mieszkać w domu 2 lub 4 (mieszkaniec domu 5 pija kakao). Ale jeśli Ukrainiec mieszka w domu 4, to Hiszpan (posiadacz psa, jak mówi zdanie 2) musi mieszkać w domu 5, a Japończyk w domu 2. Mieszkaniec domu 2 musi też pić sok pomarańczowy (zdanie 12) i jeździć mercedesem. Prowadzi to do sprzeczności, gdyż Japończyk jeździ volkswagenem! Stąd wniosek, że Ukrainiec mieszka w domu 2 i wiemy już, że:

Dom	1	2	3	4	5
Kolor	żółty	niebieski	czerwony	beżowy	zielony
Napój		ajerkoniak	mleko		kakao
Narodowość	Norweg	Ukrainiec	Anglik		
Samochód	ford				
Zwierzę		koń			

Teraz łatwo już możemy znaleźć rozwiązanie. Ponieważ właściciel mercedesa pija sok pomarańczowy (zdanie 12), musi zamieszkiwać dom 4. Skoro Japończyk ma volkswagena, musi mieszkać w domu 5. Tak więc Hiszpan (posiadacz psa) jest lokatorem domu 4. Wiemy zatem, że:

Dom	1	2	3	4	5
Kolor	żółty	niebieski	czerwony	beżowy	zielony
Napój		ajerkoniak	mleko	sok pomar.	kakao
Narodowość	Norweg	Ukrainiec	Anglik	Hiszpan	Japończyk
Samochód	ford			mercedes	volkswagen
Zwierzę		koń			pies

Zdanie 6 mówi, że posiadacz garbusa hoduje ślimaki, musi więc mieszkać w domu 3. Zdanie 10 umożliwia nam poprawne wytypowanie właścicieli chevroleta i lisa. Teraz już wiemy, że zebrę ma Japończyk, a wodę pija Norweg:

Dom	1	2	3	4	5
Kolor	żółty	niebieski	czerwony	beżowy	zielony
Napój	woda	ajerkoniak	mleko	sok pomar.	kakao
Narodowość	Norweg	Ukrainiec	Anglik	Hiszpan	Japończyk
Samochód	ford	chevrolet	garbus	mercedes	volkswagen
Zwierzę	lis	koń	ślimaki	pies	zebra

A myślałeś, że to się nigdy nie uda!

A oto bardziej formalny sposób spojrzenia na tę zagadkę. W problemie mamy 15 zmiennych:

- a_i – kolor domu i ,
- b_i – ulubiony napój mieszkańców domu i ,
- c_i – narodowość mieszkańców domu i ,
- d_i – samochód mieszkańców domu i ,
- e_i – zwierzę trzymane przez mieszkańców domu i .

Każda z tych zmiennych ma dziedzinę:

$$\begin{aligned}
 \text{dom}(a_i) &= \{\text{żółty}(\dot{Z}), \text{niebieski}(N), \text{czerwony}(C), \text{beżowy}(B), \\
 &\quad \text{zielony}(Z)\}, \\
 \text{dom}(b_i) &= \{\text{woda}(W), \text{ajerkoniak}(A), \text{mleko}(M), \\
 &\quad \text{sok pomarańczowy}(S), \text{kakao}(K)\}, \\
 \text{dom}(c_i) &= \{\text{Norweg}(N), \text{Ukrainiec}(U), \text{Anglik}(A), \text{Hiszpan}(H), \\
 &\quad \text{Japończyk}(J)\}, \\
 \text{dom}(d_i) &= \{\text{ford}(F), \text{chevrolet}(C), \text{garbus}(G), \text{mercedes}(M), \\
 &\quad \text{volkswagen}(V)\}, \\
 \text{dom}(e_i) &= \{\text{lis}(L), \text{koń}(K), \text{ślimaki}(\dot{S}), \text{pies}(P), \text{zebra}(Z)\}.
 \end{aligned}$$

Mögemy przedstawić naszą wiedzę za pomocą tabeli z możliwymi dziedzinami dla każdej zmiennej. Po przeanalizowaniu zdań 8, 9, 14, 1 i 5 wiemy na przykład, że¹:

¹Zauważ, że po wypełnieniu tabeli domenami poszczególnych zmiennych dowiadujemy się natychmiast, że Norweg pija wodę. Trudniejsze jest znalezienie właściciela zebry.

Dom	1	2	3	4	5
Kolor	$\{\dot{Z}\}$	$\{N\}$	$\{C, B\}$	$\{B, Z\}$	$\{C, Z\}$
Napój	$\{W\}$	$\{S, A\}$	$\{M\}$	$\{K, S, A\}$	$\{K, S, A\}$
Narodowość	$\{N\}$	$\{U, J\}$	$\{A, H, J\}$	$\{H, U, J\}$	$\{A, H, U, J\}$
Samochód	$\{F\}$	$\{M, V, C\}$	$\{G, V, C\}$	$\{M, G, V, C\}$	$\{M, G, V, C\}$
Zwierzę	$\{L, Z\}$	$\{K\}$	$\{P, \dot{S}, L, Z\}$	$\{P, \dot{S}, L, Z\}$	$\{P, \dot{S}, L, Z\}$

Pozostałe ograniczenia są traktowane jako ograniczenia wartości konkretnych zmiennych. Zdanie 2 na przykład staje się:

$$\text{jeśli } c_i = H \text{ to } e_i = P \text{ (dla } i = 2, 3, 4, 5)$$

Zadanie polega więc na takim wybraniu odpowiedniej wartości dla każdej cechy (koloru, napoju, narodowości, samochodu i zwierzęcia) oraz każdego domu, żeby wszystkie ograniczenia zostały spełnione.

Dobrze jest ćwiczyć się w rozwiązywaniu podobnych problemów spełniania ograniczeń, ponieważ zmuszają do logicznego myślenia i rozstrzygania, co jest możliwe, a co nie. Zadania tego typu pojawiają się na egzaminach na studia, pamiętaj więc starą harcerską zasadę: „Bądź gotów!”. A jeśli już zakończyłeś swoją edukację – czy to nie wspaniałe, że masz je już za sobą?!

9. Metody radzenia sobie z ograniczeniami

Bieda daje człowiekowi dziwnych towarzyszów łóża.

William Shakespeare: *Burza*¹

Każdy rzeczywisty problem ma ograniczenia. Nie możemy się ich pozbyć. Problemy bez ograniczeń pojawiają się tylko w książkach. Dhar i Ranganathan [102] napisali:

Prawie wszystkie problemy decyzyjne są związane z ograniczeniami. Rzeczą, która odróżnia rozmaite typy problemów, jest rodzaj tych ograniczeń. W zależności od sposobu prezentacji problemu przyjmują one postać reguł, zależności między danymi, wyrażeń algebraicznych itp.

Poprawilibyśmy tylko tę opinię przez usunięcie słowa „prawie”. W najskrajniejszym przypadku, jeśli rzeczywisty problem jest wart rozwiązania, to jest on wart rozwiązania w czasie naszego życia lub też najwyżej w czasie przewidywalnego trwania rasy ludzkiej. Ostateczny wybuch naszego słońca w postaci gwiazdy nowej nakłada ograniczenie na wszystko.

Ale wracając do tego, co nas dotyczy, możemy zetknąć się z problemami życia codziennego o bardzo nieprzyjemnych ograniczeniach. Już od pierwszego rozdziału podkreślaliśmy znaczenie dołączania ograniczeń do metod rozwiązywania problemów. Teraz skupimy się na różnych metodach radzenia sobie z ograniczeniami, które możemy dołączyć do algorytmów ewolucyjnych. W tym ujęciu w ramach jednej ewoluującej populacji możemy mieć do czynienia z rozwiązaniami zarówno dopuszczalnymi, jak i niedopuszczalnymi.

W metodach obliczeń ewolucyjnych funkcja oceny, która wartościuje poszczególne rozwiązania, jest głównym powiązaniem między problemem a algorytmem. Lepszym osobnikom zwykle jest przyznawane większe prawdopodo-

¹Przekład Władysław Tarnawski, Zakład Narodowy im. Ossolińskich, wydanie I, 1958 (przyp. tłum.).

bieństwo przyciągania i reprodukcji. Zasadniczą sprawą jest tutaj takie zdefiniowanie funkcji oceny, żeby racjonalnie pokrywała i charakteryzowała dany problem. Dokonanie tego może być nie lada wyzwaniem, gdy stoimy wobec możliwości pojawienia się niedopuszczalnych rozwiązań. Nasze końcowe rozwiązanie musi być rozwiązaniem *dopuszczalnym*, w przeciwnym razie nie jest to w ogóle rozwiązanie. Przy poszukiwaniu rozwiązań dopuszczałnych wygodne może być jednak wykonywanie operacji na rozwiązańach niedopuszczalnych. Znalezienie odpowiedniej miary oceniającej osobniki dopuszczałne i niedopuszczalne ma tutaj kolosalne znaczenie. Taka miara może decydować o sukcesie lub porażce podejścia.

Ograniczenia często mają bardzo delikatny charakter. Na przykład, gdzieś na początku naszej matematycznej edukacji daje się nam następujące zadanie:

Ślimak wspina się po drewnianym słupie o długości 10 metrów. Za dnia wchodzi pięć metrów, nocą zaś zasypia i ześlizguje się cztery metry. Ile dni zajmie ślimakowi wspięcie się na szczyt słupa?

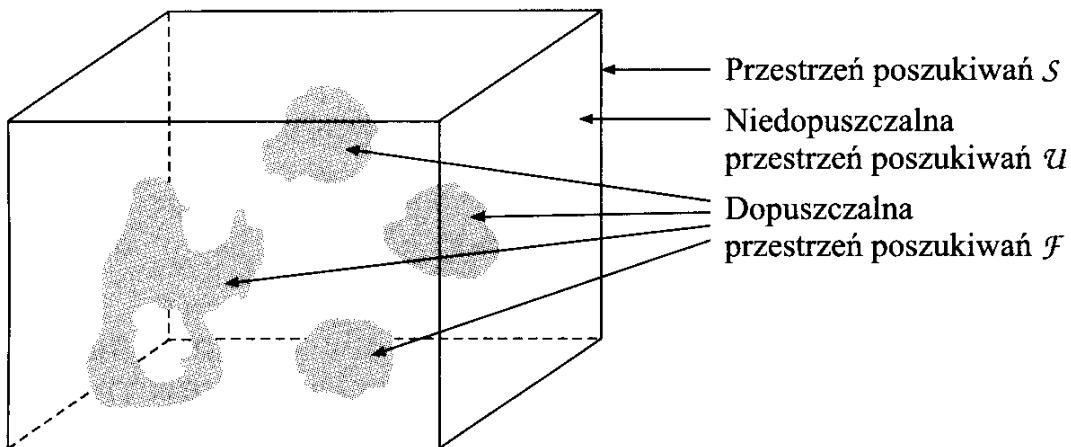
Większość dzieci (a także część dorosłych!) rozumie w ten sposób, że w ciągu 24 godzin (tzn. przez dzień i noc) ślimak „zyskuje” dokładnie jeden metr. Skoro słup ma 10 metrów, to dotarcie do czubka zajmie mu dziesięć dni.

Powyższe rozumowanie jest poprawne, ale tylko dla pierwszych pięciu dni, gdyż po tym czasie jedno z obecnych w zadaniu ograniczeń zostaje naruszone. Ograniczenie to mówi, że słup ma wysokość *tylko* dziesięciu metrów! Jeśli ślimak znajduje się na początku dnia przy kresce oznaczającej szósty metr, to nie ma już miejsca, żeby się mógł wspiąć jeszcze pięć metrów. Po pięciu dniach i nocach ślimak znajduje się przy kresce oznaczającej piąty metr i osiągnie szczyt słupa w ciągu szóstego dnia.

To proste ograniczenie wprowadza pewne utrudnienia do prostego rozumowania w innej sytuacji. Nie zawsze jednak tak musi być. Niekiedy ograniczenia są pomocne i mogą prowadzić Cię w prawidłowym kierunku. W dalszej części rozdziału pokażemy kilka sposobów korzystania z takich okoliczności. Najpierw zajmijmy się jednak ogólnymi zagadnieniami związanymi z obsługą ograniczeń w algorytmach ewolucyjnych. W podrozdziale 9.2 zilustrujemy wiele z tych zagadnień na przykładzie problemów programowania nieliniowego (NLP).

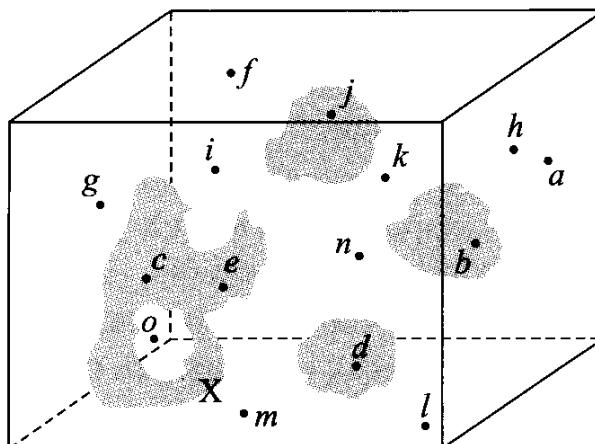
9.1. Ogólne rozważania

Przy rozwiązywaniu problemów optymalizacyjnych z ograniczeniami z zastosowaniem algorytmów ewolucyjnych ważne jest odpowiednie traktowanie niedopuszczalnych osobników [311]. W szczególności w rzeczywistych problemach opracowanie operatorów, które w pełni ich unikają i jednocześnie zapewniają dobrą lokalizację przydatnych rozwiązań dopuszczałnych, jest dość trudne. Na ogół przestrzeń poszukiwań \mathcal{S} składa się z dwóch rozłącznych podzbiorów dopuszczałnych (ang. *feasible*) i niedopuszczalnych (ang. *unfeasible*) podprzestrzeni oznaczanych odpowiednio \mathcal{F} i \mathcal{U} (zob. rys. 9.1).



Rysunek 9.1. Przestrzeń poszukiwań oraz jej części dopuszczalna i niedopuszczalna

Nie czynimy tutaj żadnych założeń co do tych podprzestrzeni. W szczególności nie muszą one być ani wypukłe, ani spójne (jak widać na rys. 9.1, część dopuszczalna F składa się z czterech rozłącznych podzbiorów). Rozwiązyując problemy optymalizacyjne, szukamy optimum *dopuszczalnego*. W czasie procesu poszukiwania mamy do czynienia z różnymi osobnikami dopuszczalnymi i niedopuszczalnymi. Na rysunku 9.2 pokazano populację na pewnym etapie ewolucji, która zawiera pewne osobniki dopuszczalne (b, c, d, e, j) oraz niedopuszczalne ($a, f, g, h, i, k, l, m, n, o$), a także (globalnie) optymalne rozwiązanie zaznaczone jako „X”.



Rysunek 9.2. Populacja z 15 osobnikami $a\text{--}o$

Konieczność pracy z rozwiązaniami dopuszczalnymi i niedopuszczalnymi może wpływać na sposób określania wielu aspektów algorytmu ewolucyjnego. Przypuśćmy, że używaliśmy jakiejś postaci selekcji elitarnej. Czy powinniśmy po prostu pamiętać najlepszego osobnika *dopuszczalnego*, czy też powinniśmy przechowywać jakiegoś osobnika *niedopuszczalnego*, który według funkcji oceny ma lepsze parametry? Czasami pytania pojawiają się przy opracowywaniu operatorów różnicowania. Niektóre operatory możemy stosować tylko do osobników dopuszczalnych. Bez wątpienia, najważniejszą trudnością jest tutaj zaprojektowanie odpowiedniej funkcji oceny, która będzie mogła obsłużyć rozwiązania zarówno dopuszczalne, jak i niedopuszczalne. Pokonanie tej trudności jest zdecydowanie niełatwwe.

Na ogół będziemy musieli opracować dwie funkcje oceny $eval_f$ i $eval_u$ działające odpowiednio na podprzestrzeni dopuszczalnej i niedopuszczalnej. Należy przy tym odpowiedzieć sobie na wiele ważnych pytań:

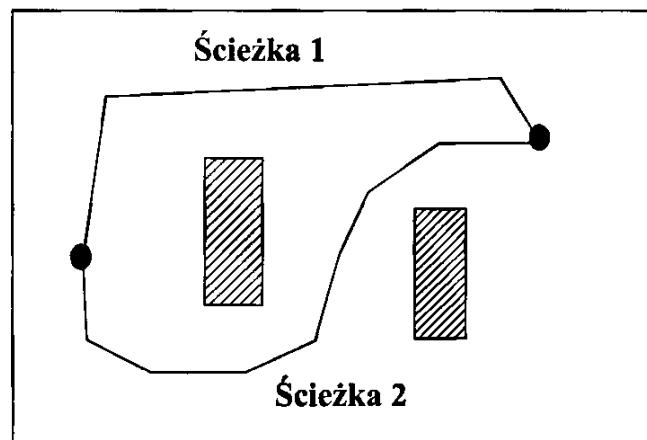
1. Jak powinniśmy porównywać dwa osobniki dopuszczalne, na przykład „ c ” i „ j ” z rys. 9.2? Innymi słowy, jak powinniśmy określić funkcję $eval_f$?
2. Jak powinniśmy porównywać dwa osobniki niedopuszczalne, na przykład „ a ” i „ n ”? Innymi słowy, jak powinniśmy określić funkcję $eval_u$?
3. W jaki sposób są ze sobą związane funkcje $eval_f$ i $eval_u$? Czy powinniśmy zakładać, na przykład, że $eval_f(s) \succ eval_u(r)$ dla dowolnego $s \in \mathcal{F}$ i dowolnego $r \in \mathcal{U}$? (Symbol \succ jest interpretowany jako „jest lepszy od”, tzn. „większy niż” w przypadku maksymalizacji oraz „mniejszy niż” w przypadku minimalizacji).
4. Czy powinniśmy uważać, że osobniki niedopuszczalne są szkodliwe i eliminować je z populacji?
5. Czy powinniśmy „poprawiać” osobniki niedopuszczalne przez przenoszenie ich do najbliższego punktu w przestrzeni dopuszczalnych rozwiązań (np. poprawioną wersją punktu „ m ” z rys. 9.2 mogłyby być optimum „ X ”)?
6. Jeśli już poprawiamy osobniki niedopuszczalne, to czy powinniśmy zastępować niedopuszczalnego osobnika w samej populacji, czy też tylko używać procedury poprawiania w czasie oceny?
7. Skoro naszym celem jest znalezienie optymalnego rozwiązania dopuszczalnego, czy powinniśmy jakoś karać osobniki niedopuszczalne?
8. Czy powinniśmy zacząć od początkowej populacji dopuszczalnych osobników i utrzymywać dopuszczalność ich potomstwa przez stosowanie specjalnych operatorów?
9. Czy powinniśmy zmieniać topologię przestrzeni poszukiwań za pomocą dekoderów, które tłumaczą rozwiązania niedopuszczalne na dopuszczalne?
10. Czy powinniśmy określić zestaw ograniczeń, które określają dopuszczalną przestrzeń poszukiwań i przetwarzać oddzielnie osobniki i ograniczenia?
11. Czy powinniśmy skoncentrować się na poszukiwaniu granicy między dopuszczalną a niedopuszczalną częścią przestrzeni poszukiwań?
12. Jak powinniśmy się zabrać do znajdowania dopuszczalnego rozwiązania?

W obliczeniach ewolucyjnych wyłoniło się kilka trendów w sposobie obsługi niedopuszczalnych rozwiązań. Większość z nich jednak pojawiła się niedawno, co sprawiło, że prace sprzed dziesięciu lat stały się w większości przestarzałe (np. [391]). Nawet jeśli ograniczymy się do podejścia z funkcją kary, która obniża jakość rozwiązań niedopuszczalnych, to w tym obszarze zastosowań znajdziemy wiele metod różniących się w wielu ważnych punktach. Inne

nowsze metody utrzymują dopuszczalność osobników w populacji za pomocą specjalnych operatorów lub dekoderów, nakładają ograniczenie, że każde dopuszczalne rozwiązanie jest „lepsze” od dowolnego niedopuszczalnego, zajmują się ograniczeniami po jednym na raz w określonej kolejności, naprawiają niedopuszczalne rozwiązania, używają metod optymalizacji wielokryterialnej, są oparte na algorytmach kulturowych lub też oceniają rozwiązania za pomocą określonego modelu koewolucyjnego. Przyjrzymy się tym metodom, przeglądając kolejno punkty 1–12.

9.1.1. Określanie funkcji $eval_f$

Określanie funkcji f dla podręcznikowych problemów jest zazwyczaj proste: funkcja ta jest zwykle po prostu dana. Przy rozwiązywaniu większości problemów z zakresu badań operacyjnych (problemu plecakowego, TSP, problemu pokrycia zbiorami itp.) funkcja oceny pojawia się jako istotna część samego sformułowania problemu. W rzeczywistych problemach sprawa nie jest taka prosta. Na przykład przy tworzeniu systemu ewolucyjnego do sterowania ruchomym robotem pojawia się potrzeba wprowadzenia sposobu oceniania różnych tras robota. Nie jest jednak jasne (zob. rys. 9.3), czy ścieżka 1, czy też ścieżka 2 powinna zostać uznana za lepszą po uwzględnieniu ich długości, obecności przeszkód i gładkości. Ścieżka 1 jest krótsza, ale ścieżka 2 jest gładsza. Gdy stajemy wobec takich problemów, pojawia się potrzeba włączenia do funkcji oceny elementów heurystycznych. Zwróć uwagę, że już nawet podzadania zmierzenia gładkości i braku przeszkód nie są takie proste.



Rysunek 9.3. Ścieżki wraz z ich otoczeniem. Która z nich jest lepsza – ta krótsza czy ta gładszta?

Podobne sytuacje spotykamy w wielu innych problemach związanych z projektowaniem, w których nie ma jasnej formuły umożliwiającej porównanie dwóch dopuszczalnych możliwości. W takich sytuacjach konieczne jest wprowadzenie pewnych zależnych od problemu heurystyk, które powinny określać liczbową miarę $eval_f$ jakości dopuszczalnego osobnika x .

Jeden z najlepszych przykładów ilustrujących trudności związane z oczną nawet dopuszczalnych osobników pojawia się w problemie SAT. Dla danej formuły w koniunkcyjnej postaci normalnej, powiedzmy

$$F(\mathbf{x}) = (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

trudne jest porównanie dwóch osobników dopuszczalnych $\mathbf{p} = (0, 0, 0)$ i $\mathbf{q} = (1, 0, 0)$ (w obydwu przypadkach $F(\mathbf{p}) = F(\mathbf{q}) = 0$). De Jong i Spears [95] przebadali kilka możliwości. Można byłoby na przykład zdefiniować $eval_f$ jako stosunek liczby iloczynów logicznych, które dają prawdę, do wszystkich iloczynów logicznych. W tym przykładzie byłoby to

$$eval_f(\mathbf{p}) = 0,666 \text{ i } eval_f(\mathbf{q}) = 0,333$$

Możemy też [347] zmienić zmienne boolowskie x_i na zmienne zmiennopozycyjne y_i i przyjąć

$$eval_f(\mathbf{y}) = |y_1 - 1||y_2 + 1||y_3 - 1| + |y_1 + 1||y_3 + 1| + |y_2 - 1||y_3 - 1|$$

lub

$$\begin{aligned} eval'_f(\mathbf{y}) = & (y_1 - 1)^2(y_2 + 1)^2(y_3 - 1)^2 + (y_1 + 1)^2(y_3 + 1)^2 + \\ & + (y_2 - 1)^2(y_3 - 1)^2 \end{aligned}$$

W powyższych przypadkach rozwiązanie problemu SAT odpowiada zbiorowi punktów, w których jest osiągane globalne minimum funkcji oceny – wartość TRUE dla $F(\mathbf{x})$ jest równoważna z otrzymaniem dla funkcji $eval_f(\mathbf{y})$ globalnego minimum o wartości 0.

Podobne trudności pojawiają się również w innych kombinatorycznych problemach decyzyjnych, jak na przykład w problemie pakowania pojemnika. Jak zaobserwowano w [134], gdzie opisano zadanie zapakowania największej liczby potencjalnie różnych przedmiotów, oczywista funkcja oceny licząca liczbę pojemników, które wystarczą do zapakowania wszystkich przedmiotów, nie jest zadowalająca. (Cel oczywiście polega na minimalizacji wartości tej właśnie funkcji oceny). Funkcja ta nie jest zadowalająca, gdyż uzyskany za jej pomocą „krajobraz” nie jest przyjazny. Istnieje niewielka liczba rozwiązań optymalnych i bardzo duża liczba rozwiązań, które dają liczbę tylko o jeden większą (tzn. wymagają tylko jeszcze jednego pojemnika). Wszystkie te nieoptymalne rozwiązania mają w świetle tej oceny tę samą wartość, co nie daje nam żadnej wskazówki na temat tego, jak poruszać się po przestrzeni poszukiwań. Problem opracowania „doskonałej” funkcji $eval_f$ jest nietrywialny.

W rzeczywistości istnieje całkowicie odrębna możliwość. W wielu wypadkach nie musimy definiować funkcji oceny $eval_f$ jako odwzorowania na liczby rzeczywiste. W pewnym sensie nie musimy jej definiować w ogóle! Konieczne jest tylko, żebyśmy stosowali metodę wyboru, która jest oparta na wartościach rozwiązań, jak to jest na przykład przy wyborze według proporcji. W wypadku innych rodzajów wyboru wystarczy sama relacja porządku mówiąca, które rozwiązania są lepsze. Jeśli relacja porządku \succ umożliwia określenie „czy osobnik dopuszczalny x jest lepszy niż osobnik dopuszczalny y ?”, to taka relacja \succ

wystarcza do zastosowania metod selekcji turniejowej i rankingowej, które wymagają odpowiednio wybrania najlepszego osobnika spośród pewnej ich liczby lub też uporządkowania wszystkich osobników w listę rankingową.

Zbudowanie takiego porządku \succ może wymagać użycia jakiejś heurystyki. W wypadku problemów optymalizacji wielokryterialnej, na przykład, określenie częściowego porządku wśród rozwiązań jest relatywnie łatwe. Konieczne jednak może być zastosowanie dodatkowych heurystyk przy porządkowaniu osobników, które nie są porównywalne z taką relacją częściową.

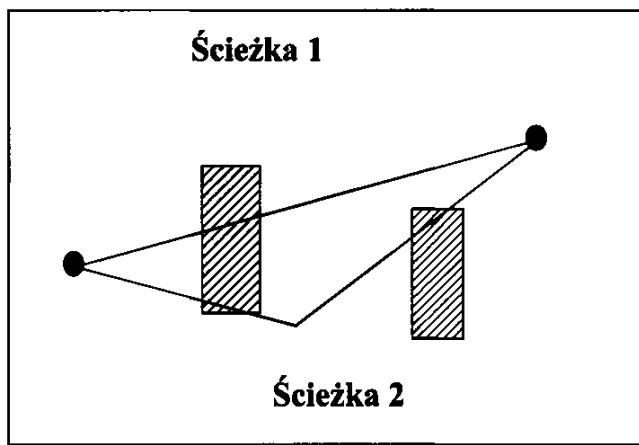
Podsumowując, wydaje się, że metody turniejowe i wyboru na podstawie rankingu dają pewną dodatkową elastyczność. Czasami łatwiej jest ustalić ranking rozwiązań, niż obliczyć numeryczną wartość każdego z nich i potem te wartości porównać. Przy metodach rankingowych musimy w dalszym ciągu rozwiązywać problemy związane z porównywaniem dwóch niedopuszczalnych osobników (zob. punkt 9.1.2), a także z porównywaniem osobników dopuszczalnych z niedopuszczalnymi (zob. punkt 9.1.3).

9.1.2. Określanie funkcji $eval_u$

Określanie funkcji oceny dla osobników niedopuszczalnych jest trudnym zadaniem. Pojawia się tutaj pokusa, żeby nie robić tego w ogóle i po prostu odrzucać wszelkie niedopuszczalne rozwiązania (zob. punkt 9.1.4). Inne rozwiązanie polega na rozszerzeniu dziedziny funkcji $eval_f$ tak, żeby obejmowała niedopuszczalne osobniki, tzn. $eval_u(x) = eval_f(x) \pm Q(x)$, przy czym $Q(x)$ oznacza albo karę związaną z niedopuszczalnym osobnikiem x , albo koszt poprawienia takiego osobnika (tzn. przekształcenia jej w dopuszczalne rozwiązanie, zob. punkt 9.1.7). Jeszcze inną możliwość polega na określaniu oddzielnej funkcji $eval_u$, która działa niezależnie od funkcji $eval_f$. Potem musimy jednak określić pewne zależności między tymi funkcjami (zob. punkt 9.1.3).

Ocenianie osobników niedopuszczalnych jest nie lada wyzwaniem. Przyjmiemy się problemowi plecakowemu, w którym naszym celem jest włożenie do plecaka o określonym rozmiarze tak dużej liczby przedmiotów, jak to tylko jest możliwe. Wielkość, o jaką jest przekroczony rozmiar plecaka, nie jest jednak zbyt dobrą miarą jakości rozwiązania (zob. punkt 9.1.7). Ta sama obserwacja dotyczy wielu problemów związanych z planowaniem lub układaniem rozkładów zajęć, a także nawet problemów planowania trasy. Nie jest jasne, czy ścieżka 1, czy ścieżka 2 jest lepsza (rys. 9.4); choć ścieżka 2 ma więcej punktów przecięcia z przeszkodami i jest dłuższa niż ścieżka 1, jednak przy użyciu takich kryteriów, większość niedopuszczalnych, choć być może bliskich dopuszczalnym, ścieżek jest „gorsza” niż linia prosta (ścieżka 1).

Podobnie jak w wypadku rozwiązań dopuszczalnych (punkt 9.1.1), możliwe jest określanie porządku dla osobników niedopuszczalnych (zamiast $eval_u$). W obu wypadkach konieczne jest określenie zależności między ocenami osobników dopuszczalnych i niedopuszczalnych (punkt 9.1.3).



Rysunek 9.4. Ścieżki niedopuszczalne wraz z ich otoczeniem. Która z nich jest lepsza?

9.1.3. Zależności między $eval_f$ a $eval_u$

Załóżmy, że zdecydowaliśmy się na korzystanie z osobników dopuszczalnych i niedopuszczalnych i że oceniamy ich za pomocą dwóch funkcji oceny $eval_f$ lub $eval_u$. Oznacza to, że dopuszczalny osobnik x jest oceniany za pomocą $eval_f(x)$, niedopuszczalny osobnik y za pomocą $eval_u(y)$. Teraz musimy ustalić zależność między tymi funkcjami oceny.

Jak wcześniejszy wspomnieliśmy, jedną z możliwości jest opracowanie $eval_u$ na podstawie $eval_f$ zgodnie ze wzorem $eval_u(y) = eval_f(y) \pm Q(y)$, przy czym $Q(y)$ oznacza albo karę związaną z niedopuszczalnym osobnikiem y , albo koszt poprawienia takiego osobnika (zob. punkt 9.1.7). Inne możliwe podejście polega na stworzeniu następującej globalnej funkcji oceny $eval$:

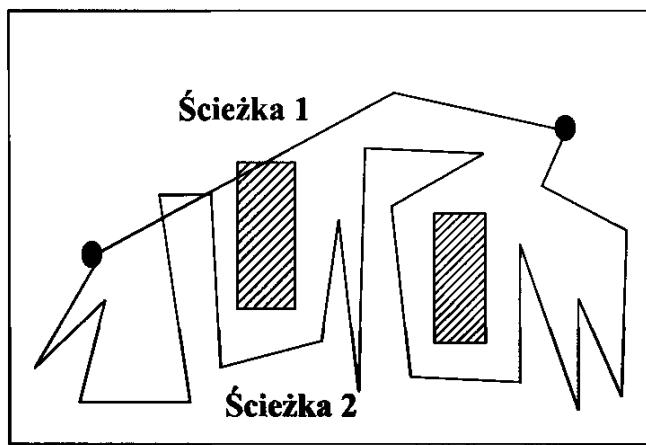
$$eval(p) = \begin{cases} q_1 \cdot eval_f(p) & \text{jeśli } p \in \mathcal{F} \\ q_2 \cdot eval_u(p) & \text{jeśli } p \in \mathcal{U} \end{cases}$$

Innymi słowy, mamy dwie wagi q_1 i q_2 , które służą do wyskalowania wzajemnego znaczenia $eval_f$ i $eval_u$.

Zwróćmy uwagę, że obie te metody dają możliwość, by pewne rozwiązania niedopuszczalne były „lepsze” niż dopuszczalne. To znaczy, że możliwe jest uzyskanie osobnika dopuszczalnego x oraz niedopuszczalnego y , dla których $eval(y) > eval(x)$. Może to prowadzić do sytuacji, gdy algorytm zatrzyma się z rozwiązaniem niedopuszczalnym. Takie zjawisko skłoniło wielu naukowców do eksperymentowania z dynamicznymi funkcjami kary Q (zob. punkt 9.1.7), które w miarę trwania ewolucyjnego poszukiwania zwiększały presję na rozwiązania niedopuszczalne. Problem wyboru $Q(x)$ (lub wag q_1 i q_2) może być jednak tak trudny, jak rozwiązanie pierwotnego problemu.

Niektórzy badacze [362, 501] uzyskali dobre wyniki, przyjmując proste założenie, że dowolne rozwiązanie dopuszczalne jest zawsze lepsze niż dowolne rozwiązanie niedopuszczalne. Heurystykę tę dla optymalizacyjnych problemów numerycznych zastosowali Powell i Skolnick [362] (zob. punkt 9.1.4). Rozwiązania dopuszczalne (w problemach minimalizacji) były mierzone wartościami z przedziału $(-\infty, 1)$, a niedopuszczalne – wartościami z przedziału $(1, \infty)$. Xiao

i Michalewicz [501] do rozwiązywania problemu planowania ścieżki dla rozwiązań dopuszczalnych i niedopuszczalnych użyli dwóch odrębnych funkcji oceny. Wartość $eval_u$ była zwiększana (co pogarszało atrakcyjność oceny) przez dodawanie do niej stałej o takiej własności, że najlepszy osobnik niedopuszczalny był gorszy od najgorszego dopuszczalnego. W rzeczywistości jest wątpliwe, żeby tego rodzaju kategoryczne rozgraniczenia były zawsze odpowiednie. W szczególności, czy dopuszczalny osobnik „ b ” (rys. 9.2) powinien rzeczywiście być oceniany jako lepszy niż niedopuszczalny „ m ”, który znajduje się „tuż przy” optymalnym rozwiązaniu? Podobny przykład możemy uzyskać, zastanawiając się nad problemem planowania ścieżki. Nie jest jasne, czy ścieżka dopuszczalna 2 (zob. rys. 9.5) zasługuje na lepszą ocenę niż ścieżka 1, która nie jest dopuszczalna! Nie ma żadnych ogólnych użytkowych heurystyk umożliwiających określenie zależności między ocenami rozwiązań dopuszczalnych i niedopuszczalnych.



Rysunek 9.5. Ścieżki niedopuszczalna i dopuszczalna wraz z ich otoczeniem.
Która z nich jest lepsza?

9.1.4. Odrzucanie rozwiązań niedopuszczalnych

W algorytmach ewolucyjnych bardzo popularna jest heurystyka „kary śmierci”. Przy każdym kroku należy „zabić” wszystkie rozwiązania niedopuszczalne. Zwróćmy uwagę, że odrzucanie rozwiązań niedopuszczalnych bardzo upraszcza całą pracę. Chociażby nie ma potrzeby określania $eval_u$ ani porównywania jej z $eval_f$.

Eliminowanie rozwiązań niedopuszczalnych może działać nieźle, gdy przestrzeń rozwiązań dopuszczalnych jest wypukła i stanowi rozsądną część całej przestrzeni poszukiwań. W innych sytuacjach podejście to ma poważne ograniczenia. Istnieje na przykład wiele problemów poszukiwania, w których losowe wyznaczanie początkowych rozwiązań może dać w wyniku całą populację samych osobników niedopuszczalnych. W takich okolicznościach kluczowe może być wykonanie procedury poprawiania rozwiązań zamiast ich odrzucania. Oczyszczanie populacji tutaj nie pomaga, gdyż prowadzi do miejsca, z którego zaczynaliśmy. Co więcej, wiele operatorów różnicowania ma tę własność, że

szycie osiągają optymalne rozwiązanie, jeśli mogą „przekraczać” niedostępne obszary (jest tak zwłaszcza dla przestrzeni dopuszczalnych rozwiązań, które nie są wypukłe).

9.1.5. Poprawianie osobników niedopuszczalnych

Naukowcy zajmujący się algorytmami ewolucyjnymi są zainteresowani koncepcją poprawiania rozwiązań niedopuszczalnych. Jest tak szczególnie wśród osób zajmujących się pewnymi kombinatorycznymi problemami optymalizacyjnymi (np. TSP, problemem plecakowym lub problemem pokrycia zbiorami). W przypadku tych problemów jest dość łatwo naprawić rozwiązanie niedopuszczalne i uczynić je dopuszczalnym. Taką poprawioną wersję możemy użyć albo przy ocenie, tzn. w taki sposób, że

$$\text{eval}_u(y) = \text{eval}_f(x)$$

przy czym x jest poprawioną (tzn. dopuszczalną) wersją y , albo jako element, którym możemy zastąpić oryginalnego osobnika w populacji (operację tę możemy wykonywać z określonym prawdopodobieństwem, zob. punkt 9.1.6). Zwrót uwagę, że poprawioną wersją rozwiązania „ m ” (rys. 9.2) może być optimum „X”.

Proces poprawiania osobników niedopuszczalnych ma wiele wspólnego z pewnym połączeniem uczenia się i ewolucji (tzw. *efektem Baldwina* [491]). Uczenie się (jak lokalne poszukiwanie w ogólności i lokalne poszukiwanie najbliższego dopuszczalnego rozwiązania w szczególności) oraz ewolucja oddziałują na siebie. Wartość uzyskana w ramach lokalnego ulepszania jest przekazywana genom osobnika. W ten sposób lokalne poszukiwanie odpowiada uczeniu się zachodzącemu w ramach jednego pokolenia.

Słabość tych metod wynika z ich zależności od konkretnego problemu. Dla każdego problemu konieczne jest opracowanie innej procedury naprawiania. Nie istnieją standardowe heurystyki ich uzyskiwania. Czasami przy poprawianiu możliwe jest zastosowanie metod zachłannych, kiedy indziej możemy użyć jakiejś procedury losowej lub też którejś z wielu innych dostępnych metod. Czasami naprawianie rozwiązań niedopuszczalnych okazuje się tak trudne jak pierwotny problem. Tak się często dzieje w nieliniowych problemach transportowych, szeregowaniu zadań czy układaniu planów. Mimo to jeden z systemów ewolucyjnych służący do rozwiązywania problemów z ograniczeniami – GENOCOP III (opisany w podrozdziale 9.2) – jest oparty na algorytmach naprawiających.

9.1.6. Zastępowanie osobników ich poprawionymi wersjami

Pomysł zastępowania osobników ich poprawionymi wersjami jest związany z takową ewolucją lamarkowską (ang. *Lamarckian evolution*) [491], w której założono, że osobnik w czasie swego życia doskonali się i wynikające z tego ulepszenia są przenoszone do jego kodu genetycznego. Oczywiście natura nie

działa w ten sposób, pamiętajmy jednak, że opracowujemy algorytmy ewolucyjne będące procedurami rozwiązywania problemów i nie musimy być ograniczeni sposobem, w jaki działa natura.

Orvosh i Davis [339] przedstawili tak zwaną regułę 5%, według której w wielu kombinatorycznych problemach optymalizacyjnych przy włączaniu procedury poprawiania do algorytmu ewolucyjnego najlepsze wyniki uzyskuje się, gdy 5% poprawionych osobników zastępuje ich pierwotne wersje niedopuszczalne. Wiemy już, że reguła ta nie sprawdza się we wszystkich dziedzinach, stanowi jednak co najmniej punkt wyjścia przy rozwiązywaniu nowego problemu. W zastosowaniach ciągłych pojawia się nowa reguła zastępowania. W systemie GENOCOP III (podrozdział 9.2), korzystającym z funkcji poprawiającej dla pewnych problemów, dobre rezultaty daje ustawienie poziomu zastępowania osobników na 15%. Większe i mniejsze wartości dawały gorsze rezultaty. Tego rodzaju wartości ściśle zależą od rozwiązywanego problemu i mogą się zmieniać nawet w trakcie jego rozwiązywania (można byłoby się pozastanawiać, jak dopasować takie parametry – zob. rozdział 10).

9.1.7. Nakładanie kar na osobniki niedopuszczalne

Najczęściej stosowanym sposobem radzenia sobie z niedopuszczalnymi rozwiązaniami jest wprowadzenie kary za ich niedopuszczalność. Rozszerzamy tutaj dziedzinę $eval_f$ i zakładamy, że

$$eval_u(p) = eval_f(p) \pm Q(p)$$

przy czym $Q(p)$ oznacza albo karę związaną z niedopuszczalnym osobnikiem p , albo koszt jego poprawienia. Podstawowe pytanie, które się tutaj pojawia, dotyczy sposobu opracowywania takiej funkcji kary $Q(p)$. Intuicja podpowiada, że kara powinna być jak najniższa – większa tylko na tyle, żeby przekroczyć próg, poniżej którego niedopuszczalne rozwiązania mogą okazać się optymalne (tzw. *reguła minimalnej kary* [280]). Efektywna implementacja tej reguły jest jednak często trudna do uzyskania.

Zależność między osobnikiem niedopuszczalnym „ p ” a częścią dopuszczalną \mathcal{F} przestrzeni poszukiwań \mathcal{S} odgrywa istotną rolę w karaniu takich osobników. Osobnik może zostać ukarany za sam fakt bycia niedopuszczalnym, za „wielkość” jego niedopuszczalności lub też proporcjonalnie do ilości pracy koniecznej do jego poprawienia. Na przykład dla problemu plecakowego z maksymalną wagą 99 kilogramów możemy mieć dwa niedopuszczalne rozwiązania, które dają ten sam zysk (obliczony na podstawie wartości przedmiotów, które się zmieściły w plecaku), ale które mają sumaryczną wagę wszystkich przedmiotów równą odpowiednio 100 i 105 kilogramów. Trudno jest uzasadnić, że pierwszy osobnik o wadze 100 jest lepszy niż drugi o wadze 105, choć dla tej pierwszej ograniczenie wagi zostało przekroczone o mniejszą wartość. Jest tak dlatego, że pierwsze rozwiązanie może być złożone z pięciu elementów, z których każdy waży 20 kilogramów, drugie może zawierać (oprócz innych przedmiotów) przedmiot o małej wartości i wadze sześciu kilogramów – jego usunięcie spowoduje uzyskanie dopuszczalnego rozwiązania, które może być o wiele lepsze

od dowolnej poprawionej wersji pierwszego osobnika. W takich wypadkach odpowiednia funkcja kary powinna uwzględniać łatwość poprawiania osobnika, a także jakość poprawionej wersji. I znów konkretne rozwiązanie zależy tutaj od problemu.

Wydaje się, że właściwy wybór metody karania zależy od: 1) stosunku wielkości przestrzeni dopuszczalnych rozwiązań do wielkości całej przestrzeni, 2) własności topologicznych dopuszczalnej przestrzeni rozwiązań, 3) rodzaju funkcji oceny, 4) liczby zmiennych, 5) liczby ograniczeń, 6) rodzaju ograniczeń oraz 7) liczby ograniczeń aktywnych w optimum. Oznacza to, że stosowanie funkcji kary nie jest łatwe i że jest dostępna tylko częściowa analiza własności takich funkcji. Obiecującym kierunkiem rozwoju badań związanych z funkcjami kary jest analiza samodostosowujących się kar – czynniki decydujące o karze mogą zostać włączone do poszczególnych rozwiązań i być zmieniane wraz ze składnikami stanowiącymi właściwe rozwiązanie (zob. rozdział 10).

9.1.8. Utrzymywanie populacji dopuszczalnej za pomocą specjalnych sposobów reprezentacji i operatorów różnicowania

Wydaje się, że jedna z najsensowniejszych heurystyk pomocnych przy zajmowaniu się zagadnieniem dopuszczalności polega na zastosowaniu wyspecjalizowanego sposobu reprezentacji oraz operatorów różnicujących, które będą utrzymywać dopuszczalność osobników w populacji.

Opracowano wiele specjalizowanych systemów umożliwiających rozwiązywanie konkretnych problemów optymalizacyjnych. Te algorytmy ewolucyjne korzystają z jedynych w swoim rodzaju sposobów reprezentacji oraz z wyspecjalizowanych operatorów różnicowania. Niektóre przykłady opisano w [91], a wiele innych omówiliśmy także tutaj. W GENOCOP (podrozdział 9.2) założono na przykład, że problem, z którym masz do czynienia, ma tylko liniowe ograniczenia i dopuszczalny punkt zakończenia (lub dopuszczalną początkową populację). Zamknięty zbiór operatorów różnicowania zachowuje dopuszczalność rozwiązań. Gdy pracujemy w ten sposób, nie ma potrzeby zajmowania się rozwiązaniami niedopuszczalnymi.

Często takie systemy są o wiele bardziej wiarygodne niż inne metody ewolucyjne oparte na stosowaniu funkcji kary. Wielu fachowców korzystało z zależnych od problemu reprezentacji oraz wyspecjalizowanych operatorów różnicowania przy rozwiązywaniu problemów optymalizacji numerycznej, uczenia się maszyn, optymalnego sterowania, modelowania kognitywnego, problemów z dziedziny klasycznych badań operacyjnych (TSP, problemu plecakowego, problemów transportowych, problemów przypisania do zadań, problemu pakowania pojemników, przydziału, rozdziału itp.), projektowania inżynierskiego, integracji systemów, rozgrywanych wielokrotnie gier, robotyki, przetwarzania sygnałów i wielu, wielu innych. Operatory różnicowania są w nich często dostosowane do użytej reprezentacji (zob. np. [167, 267]).

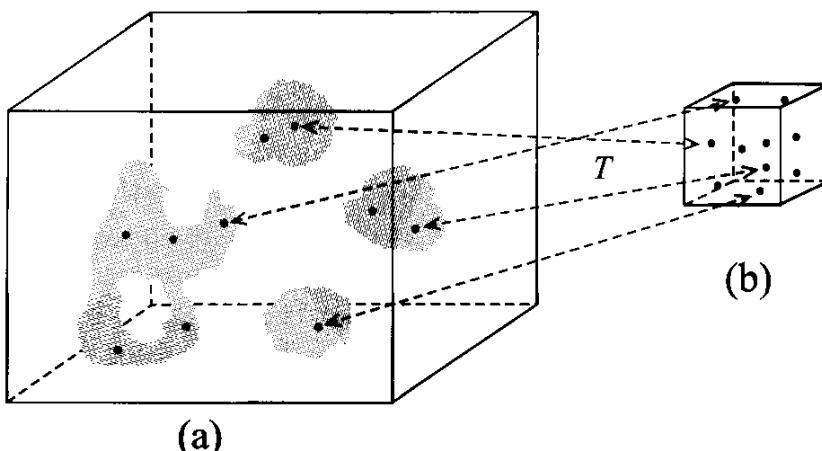
9.1.9. Stosowanie dekoderów

Zastosowanie jakiegoś rodzaju dekodera przy opracowywaniu algorytmu ewolucyjnego daje bardzo ciekawe możliwości. W tego rodzaju metodach struktura danych reprezentująca osobnika nie przedstawia bezpośrednio rozwiązania, ale podaje wskazówki, jak zbudować dopuszczalne rozwiązanie. Na przykład ciąg przedmiotów w problemie plecakowym może być interpretowany jako „weź przedmiot, jeśli jest to możliwe” – taka interpretacja będzie zawsze prowadziła do dopuszczalnego rozwiązania. Rozważmy następujący scenariusz. Musimy rozwiązać zerojedynkowy problem plecakowy (ten warunek zerojedynkowy dotyczy tutaj branych przedmiotów – albo bierzemy przedmiot, albo nie) dla n przedmiotów. Wartość i waga i -tego przedmiotu wynoszą odpowiednio p_i (ang. *profit*) i w_i (ang. *weight*). Przedmioty możemy posortować malejąco względem wartości p_i/w_i i interpretować ciąg binarny

$$\langle 1100110001001110101001010111010101 \dots 0010 \rangle$$

w następujący sposób. Weźmy pierwszy element listy (tzn. element o największym stosunku wartości do wagi), jeśli mieści się w plecaku. Powtarzajmy tę czynność dla drugiego, piątego, szóstego, dziesiątego itd. elementu listy aż do momentu, gdy plecak zostanie zapełniony. Zauważ, że ciąg złożony z samych 1 odpowiada rozwiązaniu zachłannemu. Każdy ciąg bitów jest odwzorowywany na rozwiązanie dopuszczalne, a każde rozwiązanie dopuszczalne może mieć wiele kodów. Możemy teraz zastosować klasyczne operatory binarne (krzyżowanie i mutacja), a powstałe potomstwo będzie w oczywisty sposób dopuszczalne. Efekt działania klasycznych operatorów jest tutaj jednak trudny do przewidzenia i niekoniecznie powinniśmy się spodziewać, że rezultat będzie podobny do rezultatów uzyskiwanych w innych problemach zerojedynkowych.

Warto zwrócić uwagę na kilka czynników, które muszą byćbrane pod uwagę przy korzystaniu z dekoderów. Każdy dekoder wyznacza pewną relację T między jakimś rozwiązaniem dopuszczalnym a rozwiązaniem odkodowanym (zob. rys. 9.6).



Rysunek 9.6. Przekształcenie T między rozwiązaniami w: (a) oryginalnej przestrzeni i (b) przestrzeni dekodera

Ważne jest tutaj zapewnienie spełnienia kilku warunków: 1) dla każdego rozwiązania $s \in \mathcal{F}$ istnieje zakodowane rozwiązanie d , 2) każde zakodowane rozwiązanie d odpowiada rozwiązaniu dopuszczalnemu s i 3) wszystkie rozwiązania w \mathcal{F} powinny być reprezentowane przez tę samą liczbę kodowań d^1 . Dodatkowo mają sens wymagania, żeby 4) przekształcenie T mogło być szybko obliczane i 5) miało własność lokalności, tzn. małe zmiany w zakodowanym rozwiążaniu powodują małe zmiany w rozwiążaniu właściwym. Ciekawą analizę zastosowania drzew kodujących w algorytmach ewolucyjnych przedstawili Palmer i Kershenbaum [343], gdzie także zostały sformułowane powyższe warunki.

9.1.10. Oddzielanie osobników i ograniczeń

Oddzielanie osobników i ograniczeń jest ogólną i ciekawą heurystyką. Pierwszy sposób podejścia do niej obejmuje wielokryterialne metody optymalizacyjne, gdzie funkcja oceny f i funkcje mierzące stopień naruszenia ograniczeń f_j (dla m ograniczeń) stanowią $(m+1)$ -wymiarowy wektor \mathbf{v} :

$$\mathbf{v} = (f, f_1, \dots, f_m)$$

Za pomocą którejś z metod wielokryterialnej optymalizacji możemy podjąć próbę zminimalizowania tego wektora po współrzędnych. Idealne rozwiązanie x miałyby własności $f_j(x) = 0$ dla $1 \leq j \leq m$ oraz $f(x) \leq f(y)$ dla wszystkich dopuszczalnych y (problemy minimalizacyjne). Surry [458] przedstawił udaną implementację tego podejścia.

Inne podejście jest oparte na koncepcji obsługiwania ograniczeń w określonej kolejności. Schoenauer i Xanthakis [416] nazwali tę metodę podejściem z „pamięcią behawioralną”. Omówimy tę metodę w punkcie 9.2.3.

9.1.11. Badanie granicy między dopuszczalną a niedopuszczalną częścią przestrzeni przeszukiwania

Jedna z najnowszych koncepcji rozwiązywania problemów optymalizacyjnych z ograniczeniami nosi nazwę *oscylacji strategicznej*. Pierwotnie metoda ta została zaproponowana w połączeniu z *poszukiwaniem rozproszonym*, a ostatnio

¹ Jak zaobserwował Davis w [90], wymaganie, żeby wszystkie rozwiązania w \mathcal{F} były reprezentowane przez tyle samo kodów, jest zbyt duże. Istnieją sytuacje, w których takie rozwiązanie nie jest najlepsze. Przypuśćmy, że mamy procedurę kodowania i dekodowania, która nie umożliwia reprezentowania pewnych rozwiązań nieoptymalnych, ale umożliwia reprezentowanie wszystkich rozwiązań optymalnych. Taka procedura mogłaby świetnie się spisywać. Przykładem mogłaby być oparta na porządku struktura kolorowania dla grafu z procedurą dekodowania podającą dla każdego wierzchołka pierwszy dopuszczalny dla niego kolor. Przy takiej reprezentacji nie można zakodować rozwiązań, w których pewne wierzchołki, w sytuacji gdy można było je pokolorować, nie zostały pokolorowane, ale takie ograniczenie świetnie tutaj pasuje!

została zastosowana do różnych problemów z zakresu optymalizacji kombinatorycznej i nieliniowej (zob. na przykład przeglądową pracę Glovera [184]). W podejściu tym, którego opisanie wymaga trochę wysiłku, jest wyznaczany *poziom krytyczny*, o którym możemy myśleć, że reprezentuje granicę między dopuszczalnością a niedopuszczalnością, ale który może także obejmować takie elementy jak etap konstrukcji lub wybrany przedział wartości funkcji. W kontekstach związanych z dopuszczalnością i niedopuszczalnością podstawowa strategia polega na zbliżeniu się do granicy dopuszczalności i przekroczeniu jej dzięki mechanizmowi zaimplementowanemu albo za pomocą adaptacyjnych kar i „zachęt” (które możemy stopniowo rozluźniać lub zacieśniać w zależności od tego, czy bieżący kierunek poszukiwań prowadzi w głąb konkretnego obszaru, czy też z powrotem w kierunku granicy), albo przez zastosowanie odpowiednio dostosowanych metod gradientowych lub podgradientowych prowadzących w pożądanym kierunku. W kontekście przeszukiwania otoczenia reguły określające następny ruch są zwykle projektowane tak, żeby brały pod uwagę miejsce, przez które przechodzi poszukiwanie, oraz kierunek przechodzenia przez przestrzeń. Przy wielokrotnym przekraczaniu granicy między obszarami pojawia się możliwość powtórzenia przebytej już raz drogi. Tego typu sytuacji unika się za pomocą mechanizmu z pamięcią tras i metodami probabilistycznymi. Korzystając z nich, otrzymujemy trasy, w których przekracza się granice, wędrując z różnych kierunków.

Stosowanie różnych reguł (w zależności od miejsca i kierunku przeszukiwania) jest zwykle związane z przekraczaniem granic na różną głębokość po różnych stronach granicy. Jedną z możliwości jest tutaj na przemian zbliżanie się do granicy i odchodzenie od niej po jej jednej stronie, bez przekraczania. Takie jednostronne oscylatory są szczególnie przydatne w wielu problemach związanych z szeregowaniem oraz z teorią grafów, gdzie jakaś pomocnicza struktura może być utrzymywana tylko do pewnego momentu, po czym jest tracona (np. gdy zabraknie zadań, które możemy przypisać lub gdy zostaną naruszone warunki definiujące drzewo, trasę itp.). W takich sytuacjach tworzonemu procesowi budowania, aż do osiągnięcia określonego poziomu, towarzyszy proces destrukcyjny demontowania stworzonej struktury.

W oscylacji strategicznej często ważne jest, żeby poświęcić trochę czasu na przeszukiwanie obszarów w pobliżu granicy. Można to osiągnąć, wprowadzając ciąg niewielkich oscylacji w okolicach granicy przed każdą większą oscylacją na większą głębokość. Jeśli jest dopuszczalne poświęcenie większego wysiłku na każdy ruch, to dzięki tej metodzie można wykonywać bardziej skomplikowane ruchy (takie jak różne postacie „wymian”) dokładniej przeszukujące okolice granicy. Tego rodzaju ruchy można, na przykład, stosować do osiągania lokalnego optimum za każdym razem, gdy zostanie osiągnięta pewna krytyczna odległość od granicy. Strategia stosowania takich ruchów na dodatkowych poziomach wynika z *zasady zblżonej optymalności*, według której dobre konstrukcje na jednym poziomie z dużą dozą prawdopodobieństwa będą bliskie dobrym konstrukcjom na innych poziomach. Tego typu podejście można znaleźć w pracach [176, 256, 483, 185].

Podobne podejście zaproponowano w algorytmach ewolucyjnych, w których można opracować operatory różnicowania specjalizowane w poszukiwaniu granicy (i tylko granicy) między rozwiązaniami dopuszczalnymi i niedopuszczalnymi [318]. Główna koncepcja operatorów granicznych polega tutaj na tym, że początkowa populacja znajduje się na granicy między obszarami dopuszczalnymi i niedopuszczalnymi, a operatory różnicowania są zamknięte ze względu na tę granicę – dzięki temu całe potomstwo trafia także na granicę. Ponieważ operatory te pojawiły się w kontekście optymalizacji parametrów z ograniczeniami, podejście to omawiamy w podrozdziale 9.2.

9.1.12. Znajdowanie rozwiązań dopuszczalnych

W wypadku pewnych problemów już znalezienie jakiegokolwiek rozwiązania dopuszczalnego ma wartość. Tego rodzaju problemy są naprawdę trudne! W ich wypadku nie chodzi nam o żadną optymalizację (znajdowanie *najlepszego* rozwiązania dopuszczalnego). Musimy za to znaleźć dowolny punkt w dopuszczalnej przestrzeni \mathcal{F} . Tego typu problemy nazywamy problemami *spełniania ograniczeń* (ang. *constraint satisfaction problems*). Klasycznym przykładem problemu z tej kategorii jest znany problem N królowych, w którym zadanie polega na rozmieszczeniu N królowych na szachownicy o N rzędach i N kolumnach w taki sposób, żeby żadna królowa nie biła innej. Albo się to uda, albo nie – nie ma żadnego pośredniego wyjścia.

Opracowano pewne algorytmy ewolucyjne, które rozwiązują takie problemy. Zwykle śledzą one liczbę ograniczeń, które są naruszone, i na tej podstawie określają jakość ewoluujących rozwiązań (np. [52]). Dodatkowo niektóre stosują tak zwane *skanujące* operatory rekombinacji, które przy tworzeniu nowego rozwiązania mogą korzystać z fragmentów dowolnego rozwiązania istniejącego w populacji. W ten sposób potencjalnie mogą korzystać ze wszystkich dostępnych cegiełek [116].

Paredis przeprowadził eksperymenty z dwoma różnymi podejściami do problemów spełniania ograniczeń. W pierwszym z nich [348, 349] zastosował sprytną reprezentację osobników, w której każda współrzędna oprócz wartości z dziedziny poszukiwań mogła przyjmować wartość „?”, która oznaczała niepodjęte wybory. Początkowa populacja składała się z ciągów złożonych z samych znaków zapytania. Następnie w każdym cyklu wybór-przypisanie-propagacja pewne znaki „?” były zastępowane wartościami z odpowiedniej dziedziny (przypisanie było sprawdzane ze względu na zgodność z analizowaną współrzędną). Jakość takich częściowo określonych osobników była definiowana jako wartość funkcji oceny dla najlepszego pełnego rozwiązania znalezione go przy poszukiwaniu rozpoczętym od badanego rozwiązania częściowego. Operatory różnicowania były rozszerzane tak, żeby uwzględniały proces naprawy (uwzględnianie ograniczeń). Powyższy system został zaimplementowany i użyty do rozwiązania kilku problemów N królowych [349], a także pewnych problemów szeregowania [348].

W drugim podejściu Paredis [350] badał model koewolucyjny, w którym populacja potencjalnych rozwiązań koewoluowała wraz z populacją ograniczeń.

Przy tym podejściu rozwiązania lepiej przystosowane spełniają więcej ograniczeń, ograniczenia lepiej dostosowane są łamane przez więcej rozwiązań. Oznacza to, że osobniki z populacji rozwiązań mogą pochodzić z całej przestrzeni rozwiązań i że nie ma rozróżnienia między osobnikami dopuszczalnymi i niedopuszczalnymi. Ocena osobników odbywa się na podstawie miar naruszenia ograniczeń f_j , przy czym lepsze f_j (np. aktywne ograniczenia) mają większy wpływ na ocenę rozwiązań. Podejście to sprawdzono na problemie N królowych i porównano z innymi podejściami opartymi na jednej populacji [351, 350].

9.2. Optymalizacja numeryczna

Podejmowano wiele wysiłków, żeby zastosować algorytmy ewolucyjne do optymalizacji numerycznej z ograniczeniami [310, 313]. Wykorzystywane przy tym metody można pogrupować w pięć podstawowych kategorii.

1. Metody oparte na zachowaniu dopuszczalności rozwiązań.
2. Metody oparte na funkcjach kary.
3. Metody korzystające z jasnego rozgraniczenia rozwiązań dopuszczalnych i niedopuszczalnych.
4. Metody oparte na dekoderach.
5. Inne metody hybrydowe.

Opisaliśmy już wiele z zagadnień związanych z tymi kategoriami metod. Obecnie przyjrzymy się im jeszcze raz, ale dokładniej. Co więcej, dla większości przedstawionych tutaj metod zaprezentujemy testowy przykład wraz z wynikiem działania metody dla tego przykładu.

9.2.1. Metody oparte na zachowywaniu dopuszczalności rozwiązań

Istnieją dwie metody, które należą do tej kategorii. Przedstawimy je tutaj jedną po drugiej.

Stosowanie specjalnych operatorów. Główny pomysł zastosowany w systemie GENOCOP¹ [316, 307] polega na korzystaniu ze specjalnych operatorów różnicowania, które przekształcają osobniki dopuszczalne w inne osobniki dopuszczalne. Operatory te są zamknięte ze względu na dopuszczalną część \mathcal{F} przestrzeni przeszukiwania. Wcześniej zauważliśmy, że w metodzie tej założono, że mamy do czynienia wyłącznie z liniowymi ograniczeniami oraz że początek punkt (lub populacja) jest dopuszczalny. Za pomocą równań liniowych

¹Skrót GENOCOP pochodzi od określenia angielskiego „GEnetic algorithm for Numerical Optimization of COnstrained Problems” i wymyślono go, zanim różne wątki z dziedziny algorytmów ewolucyjnych stały się funkcjonalnie podobne.

niektóre ze zmiennych są eliminowane – są zastępowane liniowymi kombinacjami pozostałych. Nierówności liniowe są także odpowiednio uaktualniane. Jeśli, na przykład, określona współrzędna x_i w wektorze rozwiązania \mathbf{x} ma zostać zmieniona, to algorytm ewolucyjny określa jej bieżącą dziedzinę $dom(x_i)$, będącą funkcją zależną od liniowych ograniczeń oraz pozostałych wartości wektora \mathbf{x} , i z tej dziedziny jest pobierana nowa wartość x_i (albo z prawdopodobieństwem o jednostajnym rozkładzie, albo o innych rozkładach, gdy mamy do czynienia z różnicowaniem typu niejednostajnego lub granicznego). Niezależnie jednak od jej rozkładu wartość ta jest wybierana tak, żeby potomstwo wektora, będącego rozwiązaniem, było zawsze dopuszczalne. Podobnie dla dwóch wektorów rozwiązań \mathbf{x} i \mathbf{y} arytmetyczne krzyżowanie $a\mathbf{x} + (1 - a)\mathbf{y}$ w wypukłych przestrzeniach poszukiwań daje zawsze dopuszczalne rozwiązanie (przy $0 \leq a \leq 1$). Ponieważ ten system ewolucyjny przyjmuje tylko ograniczenia liniowe, otrzymujemy, że część dopuszczalna \mathcal{F} przestrzeni przeszukiwania jest też wypukła.

Dla wielu funkcji testowych system GENOCOP przy ograniczeniach liniowych dawał zadziwiająco dobre wyniki [307]. Rozważmy następujący problem [137] minimalizacji funkcji

$$G1(\mathbf{x}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5 \sum_{i=1}^4 x_i^2 - \sum_{i=5}^{13} x_i$$

przy ograniczeniach

$$\begin{aligned} 2x_1 + 2x_2 + x_{10} + x_{11} &\leq 10 \\ 2x_1 + 2x_3 + x_{10} + x_{12} &\leq 10 \\ 2x_2 + 2x_3 + x_{11} + x_{12} &\leq 10 \\ -8x_1 + x_{10} &\leq 0 \\ -8x_2 + x_{11} &\leq 0 \\ -8x_3 + x_{12} &\leq 0 \\ -2x_4 - x_5 + x_{10} &\leq 0 \\ -2x_6 - x_7 + x_{11} &\leq 0 \\ -2x_8 - x_9 + x_{12} &\leq 0 \end{aligned}$$

i zakresach $0 \leq x_i \leq 1$, $i = 1, \dots, 9$, $0 \leq x_i \leq 100$, $i = 10, 11, 12$, $0 \leq x_{13} \leq 1$. W problemie tym mamy 13 zmiennych i 9 ograniczeń liniowych. Funkcja $G1$ jest kwadratowa i ma globalne minimum w punkcie

$$\mathbf{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$$

przy czym $G1(\mathbf{x}^*) = -15$. Sześć spośród dziewięciu ograniczeń jest aktywnych w tym globalnym optimum (tzn. wszystkie oprócz $-8x_1 + x_{10} \leq 0$, $-8x_2 + x_{11} \leq 0$ i $-8x_3 + x_{12} \leq 0$).

System GENOCOP uzyskał to globalne rozwiązanie przy liczbie pokoleń mniejszej niż 1000¹.

¹Szczegółowe omówienie systemu GENOCOP wraz z zastosowanymi tam operatorami, metodą selekcji, procedurą inicjowania itp. znajduje się w [307].

Gdy mamy do czynienia z ograniczeniami liniowymi, a co za tym idzie z wypukłymi przestrzeniami rozwiązań dopuszczalnych, wysiłek wkładany w poszukiwanie jest mniejszy niż w przypadku zadań o przestrzeniach niebędących zbiorami wypukłymi, gdzie obszary dopuszczalne mogą być rozłączne i nieregularne. Zwrócić uwagę na to, że metoda ta może być uogólniona do sytuacji, gdy mamy do czynienia z ograniczeniami nieliniowymi, ale przy wypukłej części dopuszczalnej \mathcal{F} . Słabość tej metody polega na tym, że nie radzi sobie z niewypukłymi przestrzeniami przeszukiwania (tzn. z obsługiwaniem ograniczeń nieliniowych ogólnej postaci).

Przeszukiwanie granicy obszaru dopuszczalnego. Poszukiwanie na granicach między obszarami dopuszczalnymi a niedopuszczalnymi może być bardzo istotne, gdy mamy do czynienia z problemami optymalizacyjnymi z nieliniowymi ograniczeniami równościowymi lub z nieliniowymi ograniczeniami aktywnymi w docelowym optimum. Algorytmy ewolucyjne stwarzają szerokie pole manewru przy używaniu specjalizowanych operatorów, które mogą efektywnie przeszukiwać takie graniczne regiony. Podamy dwa przykłady zastosowania tego podejścia; więcej szczegółów można znaleźć w [415].

Rozważmy wprowadzony w rozdziale 1 problem optymalizacji numerycznej (zob. podrozdział 1.1 i rys. 1.2). Problem polega na maksymalizacji funkcji

$$G2(\mathbf{x}) = \left| \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n ix_i^2}} \right|$$

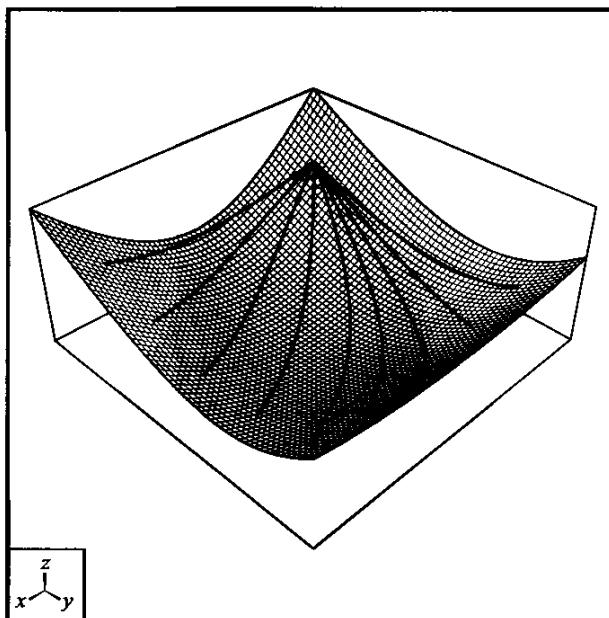
przy ograniczeniach

$$\prod_{i=1}^n x_i \geq 0,75, \quad \sum_{i=1}^n x_i \leq 7,5n \quad \text{i zakresach } 0 \leq x_i \leq 10 \text{ dla } 1 \leq i \leq n$$

Funkcja $G2$ jest nieliniowa, a jej maksimum globalne jest nieznane, choć wiadomo, że leży blisko początku układu współrzędnych. W problemie tym mamy jedno ograniczenie nieliniowe i jedno liniowe. To ostatnie jest nieaktywne w okolicy początku układu współrzędnych, będziemy więc je pomijać.

Pewne potencjalne trudności, które możemy napotkać przy rozwiązywaniu tego zadania, przedstawiono na rys. 1.2, na którym punktom niedopuszczalnym przypisano wartość zero. Granica między obszarem rozwiązań dopuszczalnych i niedopuszczalnych jest określona za pomocą równania $\prod x_i = 0,75$. Ten problem jest bardzo trudny i żadne standardowe metody (deterministyczne czy ewolucyjne) nie dawały zadowalających wyników. Użyta tu funkcja była pierwszym przykładem, na którym przetestowano pomysł szukania wyłącznie granicy obszarów [318]. Dzięki prostej postaci ograniczeń do tego problemu możemy dostosować wiele specjalnych procedur inicjujących i operatorów różnicowania.

- **Inicjowanie.** Wybieramy losowo dodatnią liczbę x_i i jako wartość x_{i+1} przyjmujemy jej odwrotność. Ostatnia zmienna ma wartość 0,75 (dla n nieparzystego) lub jakąś wielokrotność 0,75 (jeśli n jest parzyste), co powoduje, że punkt leży na powierzchni granicznej.
- **Krzyżowanie.** Operator różnicowania krzyżowania geometrycznego jest określony jako $(x_i)(y_i) \rightarrow (x_i^\alpha y_i^{1-\alpha})$, przy czym parametr α jest losowo wybrany z przedziału $[0, 1]$. Na rysunku 9.7 przedstawiono potomstwo dwóch rodziców dla wszystkich możliwych wartości α .
- **Mutacja.** Wybieramy losowo dwie zmienne i mnożymy jedną z nich przez losowy współczynnik $q > 0$, drugą zaś przez $1/q$ (ograniczamy q tak, żeby były zachowane ograniczenia dotyczące zmiennych).



Rysunek 9.7. Krzyżowanie geometryczne dla hiperboloidy przeprowadzone wokół jej centrum. Wraz ze zmianą wartości α od 0 do 1 potomstwo tworzy widoczną na rysunku linię łączącą rodziców

Opisany powyżej prosty algorytm ewolucyjny (przy standardowej zasadzie selekcji proporcjonalnej rozszerzonej o sukcesję elitarną) dał niezwykłe wyniki. Dla $n = 20$ system przy każdym uruchomieniu osiągał wartość 0,80 w ciągu mniej niż 4000 pokoleń (przy rozmiarze populacji 30, prawdopodobieństwie krzyżowania $p_c = 1,0$ i prawdopodobieństwie mutacji $p_m = 0,06$). Najlepsza znaleziona wartość 0,803553 była lepsza niż jakakolwiek wartość znaleziona za pomocą wcześniej omawianych metod, najgorsza zaś wartość wyniosła 0,802964. Podobnie dla $n = 50$ wszystkie wyniki (w ciągu 30 000 pokoleń) były lepsze niż 0,83, najlepszy zaś wynosił 0,8331937. Warto zauważyć, jakie znaczenie ma tutaj rekombinacja geometryczna. Przy ustalonym rozmiarze populacji wynoszącym 30 rozwiązań lepsze wyniki były otrzymywane przy większych prawdopodobieństwach krzyżowania (geometrycznego) i mniejszych prawdopodobieństwach mutacji.

W związku z tym, że przeszukiwanie dopuszczalnych obszarów jest nowym elementem w optymalizacji z ograniczeniami, zilustrujemy to podejście jeszcze jednym przykładem testowym. W tym zadaniu testowym zastosujemy *krzyżowanie sferyczne* [318, 415]. Zadanie polega na minimalizacji

$$G3(\mathbf{x}) = (\sqrt{n})^n \cdot \prod_{i=1}^n x_i$$

przy czym

$$\sum_{i=1}^n x_i^2 = 1 \quad i \quad 0 \leq x_i \leq 1 \quad \text{dla } 1 \leq i \leq n$$

Funkcja $G3$ ma w $(x_1, \dots, x_n) = (1/\sqrt{n}, \dots, 1/\sqrt{n})$ globalnie optymalne rozwiązanie, a wartość tej funkcji w tym punkcie wynosi 1. Nasz algorytm ewolucyjny składa się tutaj z następujących elementów:

- **Inicjowanie.** Wybieramy losowo n zmiennych y_i , obliczamy $s = \sum_{i=1}^n y_i^2$ i inicjujemy osobnika (x_i) zgodnie ze wzorem $x_i = y_i/s$ dla $i \in [1, n]$.
- **Krzyżowanie.** Operator różnicowania krzyżowania sferycznego daje jednego potomka (z_i) z dwóch rodziców (x_i) i (y_i) zgodnie ze wzorem

$$z_i = \sqrt{\alpha x_i^2 + (1 - \alpha)y_i^2} \quad i \in [1, n]$$

dla α obranego losowo z przedziału $[0, 1]$.

- **Mutacja.** Właściwy dla tego problemu operator mutacji przekształca (x_i) przez wybranie dwóch współrzędnych $i \neq j$ oraz losowej liczby p z przedziału $[0, 1]$ oraz przypisanie

$$x_i \rightarrow p \cdot x_i \quad \text{oraz} \quad x_j \rightarrow q \cdot x_j$$

przy czym $q = \sqrt{\left(\frac{x_i}{x_j}\right)^2 (1 - p^2) + 1}$

Ten prosty algorytm ewolucyjny (znowu z selekcją proporcjonalną rozszerzoną o sukcesję elitarną) dał bardzo dobre wyniki. Dla $n = 20$ system osiągnął wartość 0,99 w każdej próbie w ciągu mniej niż 6000 pokoleń (przy rozmiarze populacji 30, prawdopodobieństwie krzyżowania $p_c = 1,0$ i prawdopodobieństwie mutacji $p_m = 0,06$). Najlepsza wartość osiągnięta po 10 000 pokoleniach wynosiła 0,999866.

9.2.2. Metody oparte na funkcjach kary

Główne wysiłki związane z ewolucyjnym rozwiązywaniem optymalizacji z ograniczeniami były związane z wykorzystaniem funkcji kary, które zmniejszały

wartość rozwiązań niedopuszczalnych. Dzięki nim problem z ograniczeniami stawał się problemem bez ograniczeń, ale o zmodyfikowanej funkcji oceny

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{jeśli } \mathbf{x} \in \mathcal{F} \\ f(\mathbf{x}) + kara(\mathbf{x}) & \text{w przeciwnym razu} \end{cases}$$

przy czym $kara(\mathbf{x})$ jest równa zeru, jeśli ograniczenie nie zostało naruszone, i jest dodatnia, jeśli zostało naruszone (przy założeniu, że celem jest minimalizacja). Funkcja $kara$ jest zwykle oparta na jakiegoś rodzaju odległości wyznaczającej, jak daleko znajduje się dane rozwiązanie niedopuszczalne od obszaru \mathcal{F} rozwiązań dopuszcjalnych, lub też, jak dużo wysiłku wymaga „nareperowanie” rozwiązania, tzn. przekształcenie go w coś z \mathcal{F} . Wiele metod przy tworzeniu kary korzysta ze zbioru funkcji f_j ($1 \leq j \leq m$), przy czym funkcja f_j mierzy stopień naruszenia j -go ograniczenia:

$$f_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\} & \text{jeśli } 1 \leq j \leq q \\ |h_j(\mathbf{x})| & \text{jeśli } q+1 \leq j \leq m \end{cases}$$

(definicje g_j i h_j znajdziesz w punkcie 3.2.1). Metody te różnią się w wielu istotnych szczegółach, jeśli chodzi o sposób tworzenia funkcji kary, a także jeśli chodzi o sposób, w jaki jest stosowana do niedopuszczalnych rozwiązań. Część z tych szczegółów przedstawimy poniżej.

Metody kar statycznych. W pracy [231] Homaifar wraz z zespołem zaproponował tworzenie pewnej rodziny przedziałów dla każdego danego ograniczenia. Przedziały te określają odpowiedni współczynnik kary. Pomyśl ten sprowadza się do takich działań.

- Dla każdego ograniczenia stwórz kilka (l) stopni (ang. *level*) jego naruszenia.
- Dla każdego stopnia naruszenia i dla każdego ograniczenia przyjmij współczynnik kary R_{ij} ($i = 1, 2, \dots, l$, $j = 1, 2, \dots, m$). Wyższe stopnie naruszenia wymagają współczynników o większej wartości (znowu, zakładamy, że wykonujemy minimalizację).
- Zacznij od losowej populacji osobników (dopuszczalnych lub niedopuszczalnych).
- Przeprowadź ewolucję populacji. Oceniaj osobniki za pomocą funkcji

$$eval(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^m R_{ij} f_j^2(\mathbf{x})$$

Słabość tej metody jest związana z liczbą parametrów. Przy m ograniczeniach metoda ta wymaga w sumie $m(2l+1)$ parametrów: m określa liczbę przedziałów dla każdego ograniczenia, $l+1$ wartości służy do określenia końców przedziałów (stopni naruszenia ograniczenia) dla każdego ograniczenia, następnie l służy do określenia współczynników kary R_{ij} . W szczególności dla $m = 5$ ograniczeń

i $l = 4$ poziomów ograniczeń musimy określić 45 parametrów! Otrzymane wyniki w oczywisty sposób zależą od tych parametrów. Dla dowolnego konkretnego problemu może istnieć optymalny zestaw parametrów, przy którym możemy otrzymywać dopuszczalne rozwiązania bliskie optymalnym, taki zestaw jednak może być bardzo trudny do uzyskania.

Niewielki zestaw eksperymentów przedstawiony w [308] wskazuje, że metoda ta może dać dobre rezultaty, jeśli stopnie naruszenia ograniczeń oraz współczynniki kary R_{ij} są dostosowane do problemu. Rozważmy, na przykład, opisany w [219] problem minimalizacji następującej funkcji pięciu zmiennych:

$$G4(\mathbf{x}) = 5,3578547x_3^2 + 0,8356891x_1x_5 + 37,293239x_1 - 40792,141$$

przy trzech poniższych podwójnych nierównościach:

$$0 \leq 85,334407 + 0,0056858x_2x_5 + 0,0006262x_1x_4 - 0,0022053x_3x_5 \leq 92$$

$$90 \leq 80,51249 + 0,0071317x_2x_5 + 0,0029955x_1x_2 + 0,0021813x_3^2 \leq 110$$

$$20 \leq 9,300961 + 0,0047026x_3x_5 + 0,0012547x_1x_3 - 0,00190853x_3x_4 \leq 25$$

i zakresach

$$78 \leq x_1 \leq 102, \quad 33 \leq x_2 \leq 45, \quad 27 \leq x_i \leq 45 \quad \text{dla } i = 3, 4, 5$$

Najlepsze rozwiązanie uzyskane w 10 próbach przez Homaifara i in. [219] to

$$\mathbf{x} = (80,49, 35,07, 32,05, 40,33, 33,34)$$

przy czym $G4(\mathbf{x}) = -30005,7$, podczas gdy rozwiązanie optymalne [219] to

$$\mathbf{x}^* = (78,0, 33,0, 29,995, 45,0, 36,776)$$

przy $G4(\mathbf{x}^*) = -30665,5$. W tym optimum aktywne są dwa ograniczenia (górnego ograniczenie z pierwszej nierówności i dolne z trzeciej).

Homaifar i in. podali w [231], że współczynnik kary, który odpowiadał naruszeniu dolnego ograniczenia z pierwszej nierówności, wynosił 50, a współczynnik kary dla górnego ograniczenia tej nierówności wynosił 2,5. Jest zrozumiałe, że możemy uzyskać lepsze wyniki, jeśli współczynniki kary możemy dobrać do problemu lub jeśli możemy je dostosowywać na bieżąco w trakcie rozwiązywania problemu (zob. rozdział 10).

Metoda kar dynamicznych. Joines i Houck w pracy [247] odeszli od metod zastosowanych w [231] i zastosowali kary dynamiczne. W ich podejściu osobniki są oceniane w każdej iteracji (pokoleniu) t za pomocą formuły

$$\text{eval}(\mathbf{x}) = f(\mathbf{x}) + (C \times t)^\alpha \sum_{j=1}^m f_j^\beta(\mathbf{x})$$

przy czym C , α i β są stałymi. Rozsądne ich wartości wynoszą $C = 0,5$, $\alpha = \beta = 2$ [247]. Metoda ta nie wymaga określenia tak wielu parametrów

jak w opisanej wcześniej metodzie kar statycznych i zamiast wielu stopni naruszenia ograniczeń mamy tutaj w niedopuszczalnych rozwiązaniach zwiększający się we wzorze na karę współczynnik $(C \times t)^\alpha$ – w miarę wzrostu t współczynnik ten także rośnie.

Joines i Houck [247] eksperymentowali z testowym przykładem [227], w którym minimalizowali funkcję

$$G5(\mathbf{x}) = 3x_1 + 0,000001x_1^3 + 2x_2 + \frac{0,000002}{3}x_2^3$$

przy ograniczeniach

$$x_4 - x_3 + 0,55 \geq 0, \quad x_3 - x_4 + 0,55 \geq 0$$

$$1000 \sin(-x_3 - 0,25) + 1000 \sin(-x_4 - 0,25) + 894,8 - x_1 = 0$$

$$1000 \sin(x_3 - 0,25) + 1000 \sin(x_3 - x_4 - 0,25) + 894,8 - x_2 = 0$$

$$1000 \sin(x_4 - 0,25) + 1000 \sin(x_4 - x_3 - 0,25) + 1294,8 = 0$$

$$0 \leq x_i \leq 1200, \quad i = 1, 2 \quad \text{oraz} \quad -0,55 \leq x_i \leq 0,55, \quad i = 3, 4$$

Najlepsze znane rozwiązanie jest następujące:

$$\mathbf{x}^* = (679,9453, 1026,067, 0,1188764, -0,3962336)$$

przy $G5(\mathbf{x}^*) = 5126,4981$.

Najlepszy wynik podany w [247] (uzyskany po wielu próbach i przy różnych wartościach α oraz β) dał wartość 5126,6653. Co ciekawe, żadne z uzyskanych rozwiązań nie było w pełni dopuszczalne ze względu na trzy ograniczenia równościowe, lecz suma naruszeń była jednak dość mała (10^{-4}). Wydaje się, że metoda ta czasem przypisuje zbyt wysokie kary. Często współczynnik $(C \times t)^\alpha$ wzrasta zbyt szybko, żeby można było z niego korzystać. W wyniku tego ewolucja może się zatrzymać w jakimś lokalnym optimum. Eksperymenty, w których wykorzystano tę metodę w [308], wykazały, że najlepszy osobnik był często znajdowany we wcześniejszych pokoleniach. Metoda ta dała bardzo dobre wyniki dla przykładów testowych, w których korzystano z kwadratowych funkcji oceny.

Metoda kar wyżarzanych. W innym sposobie dynamicznego określania kar skorzystano z algorytmu wyżarzania – możliwe, że możemy wyżarzyć wartości kar za pomocą jakiegoś parametru analogicznego do „temperatury”. Ta-ka procedura została włączona do drugiej wersji GENOCOP (GENOCOP II) [312, 307].

- Podziel wszystkie ograniczenia na cztery podzbiory: równania liniowe, nierówności liniowe, równania nieliniowe i nierówności nieliniowe.
- Wybierz losowo jeden punkt początkowy. (Populacja początkowa składa się z kopii tego jednego osobnika). Ten punkt początkowy spełnia ograniczenia liniowe.
- Ustaw temperaturę początkową $\tau = \tau_0$.

- Przeprowadź ewolucję populacji. Oceniaj osobniki za pomocą funkcji

$$eval(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \sum_{j=1}^m f_j^2(\mathbf{x})$$

- Jeśli $\tau < \tau_f$, to się zatrzymaj; w przeciwnym razie:
 - zmniejsz temperaturę τ ,
 - użyj najlepszego dostępnego rozwiązania jako punktu początkowego w następnej iteracji,
 - powtóż poprzedni główny krok algorytmu.

W metodzie tej inaczej traktujemy ograniczenia liniowe i nieliniowe. Algorytm za pomocą zestawu specjalnych operatorów utrzymuje dopuszczalność ze względu na ograniczenia liniowe. Operatory te przekształcają rozwiązania dopuszczalne ze względu na te ograniczenia w inne rozwiązania, również ze względu na nie dopuszczalne. Algorytm w każdym pokoleniu bierze pod uwagę tylko aktywne ograniczenia, a nacisk selekcyjny na rozwiązania niedopuszczalne zwiększa się ze względu na zmniejszające się wartości temperatury τ .

Metoda ta ma dodatkową jedyną w swoim rodzaju własność: zaczyna pracę od jednego punktu¹. Dzięki temu możemy relatywnie łatwo porównać ją z innymi klasycznymi metodami optymalizacyjnymi, dla których efektywność jest testowana na danym problemie przy jakimś punkcie początkowym. Metoda ta wymaga temperatury początkowej τ_0 i temperatury „zamarzania” τ_f (ang. *freezing*) oraz schematu schładzania, który zmniejsza temperaturę τ . Standarde wartości [312] wynoszą $\tau_0 = 1$, $\tau_{i+1} = 0,1 \cdot \tau_i$ i $\tau_f = 0,000001$. Jednym z przykładów testowych dla tej metody [137] był zbadany przez Michalewicza i Attię w [312] problem:

$$\text{minimalizacji } G6(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

przy nieliniowych ograniczeniach

$$\begin{aligned} c1 : \quad & (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geqslant 0 \\ c2 : \quad & -(x_1 - 6)^2 - (x_2 - 5)^2 + 82,81 \geqslant 0 \end{aligned}$$

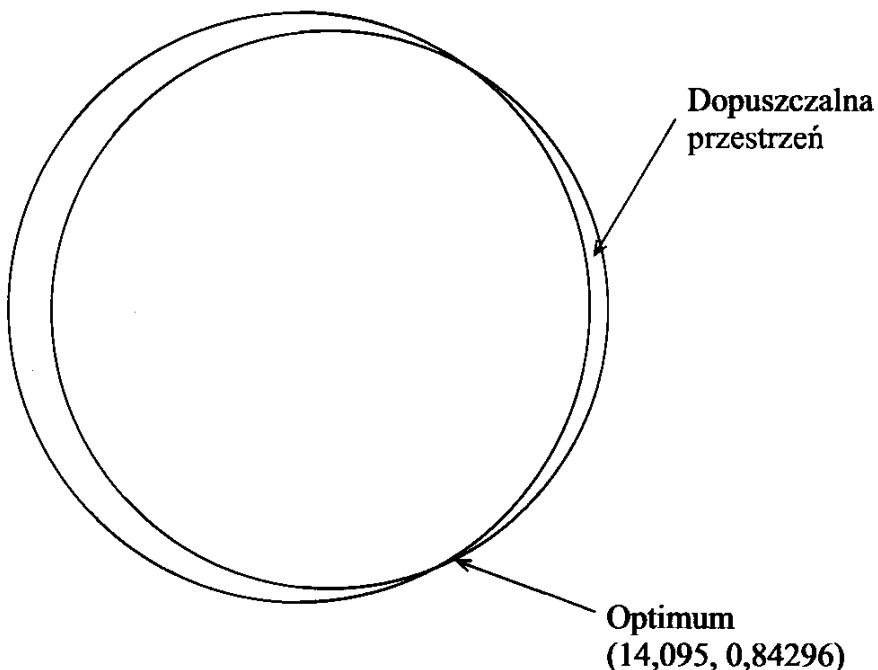
i zakresach

$$13 \leqslant x_1 \leqslant 100 \text{ oraz } 0 \leqslant x_2 \leqslant 100$$

Znane globalne rozwiązanie wynosi $\mathbf{x}^* = (14,095, 0,84296)$ oraz $G6(\mathbf{x}^*) = -6961,81381$ (rys. 9.8). W tym optimum aktywne są oba ograniczenia. Punkt początkowy, tym razem niedopuszczalny, był równy $\mathbf{x}_0 = (20,1, 5,84)$.

GENOCOP II zbliżył się bardzo do optimum już przy 12. iteracji. Rozwój systemu przedstawiono w tab. 9.1.

¹Własność ta nie jest jednak istotna. Jedyne ważne wymaganie polega na tym, że następna populacja zawiera najlepszego osobnika z poprzedniej populacji.



Rysunek 9.8. Dopuszczalna przestrzeń dla przykładu testowego G6

Tabela 9.1. Rozwój GENOCOP II przy analizie funkcji G6. Dla iteracji 0 najlepszym punktem jest punkt początkowy

Numer iteracji	Najlepszy punkt	Aktywne ograniczenia
0	(20,1, 5,84)	c_1, c_2
1	(13,0, 0,0)	c_1, c_2
2	(13,63, 0,0)	c_1, c_2
3	(13,63, 0,0)	c_1, c_2
4	(13,73, 0,16)	c_1, c_2
5	(13,92, 0,50)	c_1, c_2
6	(14,05, 0,75)	c_1, c_2
7	(14,05, 0,76)	c_1, c_2
8	(14,05, 0,76)	c_1, c_2
9	(14,10, 0,87)	c_1, c_2
10	(14,10, 0,86)	c_1, c_2
11	(14,10, 0,85)	c_1, c_2
12	(14,098, 0,849)	c_1, c_2

W pracy [308] przedstawiono wyniki eksperymentu świadczące o tym, że ograniczenia liniowe problemu mogą nie dopuszczać systemu do zbliżania się do optimum. Jest to istotny przykład szkodliwego wpływu, jaki może mieć ograniczanie populacji tylko do obszaru dopuszczalnego, przynajmniej jeśli chodzi o dopuszczalność ze względu na ograniczenia liniowe. Dodatkowe badania wykazały, że skuteczność tej metody bardzo silnie zależy od zastosowanego schematu schładzania.

Metody kar dostosowujących się. Zamiast korzystać z ustalonego schematu zmniejszania temperatury, możemy wprowadzić informację zwrotną, pochodzącą z procesu poszukiwania i wykorzystać ją do korygowania kar (zob. też rozdział 10). Omówimy teraz dwa sposoby włączania tej informacji.

Bean i Hadj-Alouane w pracach [36, 205] wprowadzili metodę, w której funkcja kary uwzględnia informacje zwrotne pochodzące z poszukiwania. Każdy osobnik jest oceniany za pomocą wzoru

$$eval(\mathbf{x}) = f(\mathbf{x}) + \lambda(t) \sum_{j=1}^m f_j^2(\mathbf{x})$$

przy czym $\lambda(t)$ jest uaktualniane przy każdym pokoleniu t :

$$\lambda(t+1) = \begin{cases} \frac{1}{\beta_1} \cdot \lambda(t) & \text{jeśli } \mathbf{b}^i \in \mathcal{F} \text{ dla } t-k+1 \leq i \leq t \\ \beta_2 \cdot \lambda(t) & \text{jeśli } \mathbf{b}^i \in \mathcal{S} - \mathcal{F} \text{ dla } t-k+1 \leq i \leq t \\ \lambda(t) & \text{w przeciwnym wypadku} \end{cases}$$

przy czym: \mathbf{b}^i oznacza osobnika najlepszego zgodnie z funkcją $eval$ w pokoleniu i , $\beta_1, \beta_2 > 1$ i $\beta_1 \neq \beta_2$, żeby uniknąć cykli. Innymi słowy: 1) jeśli wszystkie najlepsze osobniki w ostatnich k pokoleniach były dopuszczalne, to w metodzie tej jest zmniejszana wartość kary $\lambda(t+1)$ dla pokolenia $t+1$ oraz 2) jeśli wszystkie najlepsze osobniki w ostatnich k pokoleniach były niedopuszczalne, to metoda zwiększa karę. Jeśli w ciągu ostatnich k pokoleń mamy trochę osobników dopuszczalnych i trochę niedopuszczalnych, to wartość $\lambda(t+1)$ pozostaje niezmieniona.

W tej metodzie wprowadzono trzy dodatkowe parametry: β_1 , β_2 oraz horyzont czasowy k . Z intuicyjnego rozumowania uzasadniającego metodę Beana i Hadja-Alouana wynika, że jeśli ograniczenia nie stanowią problemu, to algorytm powinien działać dalej z mniejszymi karami; w przeciwnym wypadku kary powinny zostać zwiększone. Obecność osobników dopuszczalnych i niedopuszczalnych w zbiorze najlepszych osobników w ostatnich k pokoleniach oznacza, że aktualna wartość współczynnika kary $\lambda(t)$ jest ustawiona poprawnie, a przy najmniej mamy taką nadzieję.

W innym podejściu zaproponowanym przez Smitha i Tate'a w [435] dla każdego ograniczenia $1 \leq j \leq m$ skorzystano z progu „bliskości dopuszczalności” q_j . Progi te oznaczają odległości od obszaru dopuszczalnego \mathcal{F} , które uznaje się za „sensowne” (lub innymi słowy, które umożliwiają uznanie rozwiązań niedopuszczalnych za „interesujące” ze względu na ich bliskość od obszaru dopuszczalnego). W związku z tym funkcja oceny jest tutaj zdefiniowana jako

$$eval(\mathbf{x}, t) = f(\mathbf{x}) + F_{feas}(t) - F_{all}(t) \sum_{j=1}^m \left(\frac{f_j(\mathbf{x})}{q_j(t)} \right)^k$$

przy czym: $F_{all}(t)$ oznacza wartość najlepszego dotychczas znalezionego (do pokolenia t) rozwiązania, pozbawioną obciążenia związanego z karą, $F_{feas}(t)$ oznacza wartość najlepszego dotychczas znalezionego rozwiązania dopuszczalnego,

k jest stałą. Zauważmy, że progi bliskości obszarów dopuszczalnych $q_j(t)$ są dynamiczne. Ich wartość w trakcie poszukiwania wykorzystującego sprzężenie zwrotne zmienia się w zależności od jego przebiegu. Możemy na przykład zdefiniować $q_j(t) = q_j(0)/(1+\beta_j t)$, dzięki czemu współczynnik kary będzie się zwiększał w czasie. Zgodnie z naszą najlepszą wiedzą żadnej z metod dostosowujących się nie zastosowano do ciągłych problemów programowania nieliniowego.

Metoda kary śmierci. W metodzie wykorzystującej karę śmierci niedopuszczalne rozwiązania są usuwane, czyli, innymi słowy, są natychmiast zabijane. Ta prosta metoda może dać przy niektórych problemach bardzo dobre wyniki. Jeden z testowych przykładów opisany w [308] dotyczył następującego problemu [227]. Należało zminimalizować funkcję

$$\begin{aligned} G7(\mathbf{x}) = & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + \\ & + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + \\ & + (x_{10} - 7)^2 + 45 \end{aligned}$$

przy ograniczeniach:

$$\begin{aligned} 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 &\geq 0 \\ -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 &\geq 0 \\ -10x_1 + 8x_2 + 17x_7 - 2x_8 &\geq 0 \\ -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 &\geq 0 \\ 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 &\geq 0 \\ -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 &\geq 0 \\ 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} &\geq 0 \\ -0,5(x_1 - 8)^2 - 2(x_2 - 4)^4 - 3x_5^2 + x_6^2 + 30 &\geq 0 \end{aligned}$$

i zakresach

$$-10,0 \leq x_i \leq 10,0, \quad i = 1, \dots, 10$$

Problem ma trzy ograniczenia liniowe i pięć nieliniowych. Funkcja $G7$ jest kwadratowa i ma globalne minimum w punkcie

$$\begin{aligned} \mathbf{x}^* = & (2,171996, 2,363683, 8,773926, 5,095984, 0,9906548, \\ & 1,430574, 1,321644, 9,828726, 8,280092, 8,375927) \end{aligned}$$

przy czym $G7(\mathbf{x}^*) = 24,3062091$.

W tym globalnym optimum aktywnych jest sześć spośród ośmiu ograniczeń (wszystkie oprócz dwóch ostatnich). System GENOCOP rozszerzony o metodę wykorzystującą „karę śmierci” znalazł rozwiązania dobrej jakości, z których najlepsze dawało wartość 25,653.

Metoda ta wymaga inicjowania za pomocą rozwiązań dopuszczalnych, a zatem porównania z innymi metodami mogą być trudne; z eksperymentów przeanalizowanych w [308] wyłania się bardzo ciekawy obraz. Metody usuwające rozwiązania niedopuszczalne działały dość słabo i nie były tak silne jak inne metody (tzn. standardowe odchylenie wartości rozwiązań było dość duże).

Algorytmy ewolucyjne z segregacją. Przy dostrajaniu współczynników kary zbyt małe poziomy kary prowadzą do uzyskiwania rozwiązań niedopuszczalnych, gdyż pewne rozwiązania obciążone karą osiągają wciąż większe wartości niż niektóre rozwiązania dopuszcjalne. Z drugiej strony zbyt duży poziom kary ogranicza przeszukiwanie tylko do obszarów dopuszcjalnych i powoduje, że są omijane pewne skróty przez obszary niedopuszcjalne. Prowadzi to często do przedwczesnego zatrzymania się na dostępnych rozwiązańach o mniejszej wartości. Pewna metoda radzenia sobie z tymi problemami została opisana w [280].

Pomysł polega na tym, żeby opracować dwie różne funkcje oceny z karami ze statycznymi współczynnikami kary p_1 i p_2 . Wartość p_1 jest celowo ustalona zbyt nisko, a wartość p_2 – zbyt wysoko. Każdy osobnik w bieżącej populacji przechodzi rekombinację i mutację (lub różnicowanie w jakiejś postaci). Wartości dwóch funkcji oceny $f_i(\mathbf{x}) = f(\mathbf{x}) + p_i(\mathbf{x})$, $i = 1, 2$, są obliczane dla każdego potomka (z uniknięciem dodatkowego kosztu związanego z wielokrotnym wywoływaniem właściwej funkcji oceny) i na podstawie uzyskanych wartości są tworzone dwie listy rankingowe, zawierające rodziców i potomstwo wraz z wartościami odpowiednich funkcji oceny.

Rodzice następnego pokolenia są wybierani przez pobieranie kolejno po jednym elemencie na przemian z jednej listy i z drugiej zgodnie z kolejnością ich występowania. Główny pomysł polega tu na tym, że w takim schemacie selekcji będą utrzymywane dwie podpopulacje: te osobniki, które zostały wybrane dzięki f_1 , będą częściej leżały w obszarze rozwiązań niedopuszcjalnych, a wybrane dzięki f_2 będą częściej leżały w obszarze rozwiązań dopuszcjalnych. Na ogół poszukiwanie ewolucyjne będzie osiągało dopuszcjalne rozwiązania, zbliżając się do nich zarówno od strony obszaru rozwiązań dopuszcjalnych, jak i rozwiązań niedopuszcjalnych. Metoda ta dała doskonałe wyniki w optymalizacji stosowanej przy projektowaniu płyt drukowanych [280], nie została jednak jeszcze, według najlepszej wiedzy autorów, zastosowana do problemów ciągłego programowania nieliniowego.

9.2.3. Metody oparte na poszukiwaniu rozwiązań dopuszcjalnych

W niektórych metodach ewolucyjnych szczególnie uwypukla się problem rozróżnienia rozwiązań dopuszcjalnych i niedopuszcjalnych. W jednej z metod rozważamy ograniczenia po kolei. Gdy zostanie znaleziona wystarczająca liczba rozwiązań dopuszcjalnych ze względu na dane ograniczenie, analizowane jest następne. W innej metodzie z kolei przyjmujemy, że dowolne rozwiązanie dopuszcjalne jest lepsze niż jakiekolwiek rozwiązanie niedopuszcjalne (co już wcześniej rozważaliśmy). W jeszcze innej poprawiamy osobniki niedopuszcjalne. Przyjrzyjmy się po kolei tym metodom.

Metoda pamięci behawioralnej. W pracy [416] Schoenauer i Xanthakis zaproponowali podejście, które nazwali „pamięcią behawioralną”.

- Zaczni od losowej populacji osobników (dopuszczalnych lub niedopuszczalnych).
- Ustaw $j = 1$ (j jest licznikiem ograniczeń).
- Przeprowadź ewolucję populacji. Oceniaj osobniki za pomocą funkcji $eval(\mathbf{x}) = f_j(\mathbf{x})$. Ewolucja kończy się, gdy dany procent populacji stanie się dopuszczalny ze względu na przetwarzane ograniczenie¹ (zostanie osiągnięty tak zwany próg przełączenia ϕ).
- Ustaw $j = j + 1$.
- Bieżąca populacja jest punktem wyjścia dla następnej fazy ewolucji, w której $eval(\mathbf{x}) = f_j(\mathbf{x})$ (definicja pochodzi z punktu 9.2.2). W tej fazie punkty, które nie spełniają pierwszego, drugiego, ... lub $(j-1)$ -ego ograniczenia, są usuwane z populacji. Kryterium zatrzymania polega znowu na przekroczeniu procentowego progu przełączenia ϕ populacji zachowującej j -te ograniczenie.
- Jeśli $j < m$, to należy powtarzać dwa poprzednie kroki, w przeciwnym wypadku ($j = m$) należy wykonać optymalizację funkcji oceny, tzn. $eval(\mathbf{x}) = f(\mathbf{x})$, jednocześnie odrzucając osobniki niedopuszczalne.

Metoda ta wymaga ustalenia kolejności rozpatrywania wszystkich ograniczeń. Wpływ uporządkowania ograniczeń na końcowy wynik nie jest jasny, a różne uporządkowania mogą dawać różne rezultaty i to zarówno jeśli chodzi o czas działania algorytmu, jak i o dokładność wyniku. Niezależnie od porządku metoda ta wymaga określenia trzech parametrów: współczynnika współdzielienia σ , progu przełączenia ϕ oraz porządku ograniczeń. Metoda ta różni się znacznie od wielu innych metod i dość mocno odbiega od pozostałych algorytmów wykorzystujących kary, gdyż w danym momencie bierze ona pod uwagę tylko jedno ograniczenie. W dodatku w końcowej fazie jest stosowana taktyka „kary śmierci”. Wszystko to powoduje, że nie możemy jej łatwo zaszufladkować do tej lub innej kategorii.

W [416] Schoenauer i Xanthakis sprawdzili działanie omawianej tutaj procedury na następującym problemie. Należało zmaksymalizować funkcję

$$G8(\mathbf{x}) = \frac{\sin^3(2\pi x_1) \cdot \sin(2\pi x_2)}{x_1^3 \cdot (x_1 + x_2)}$$

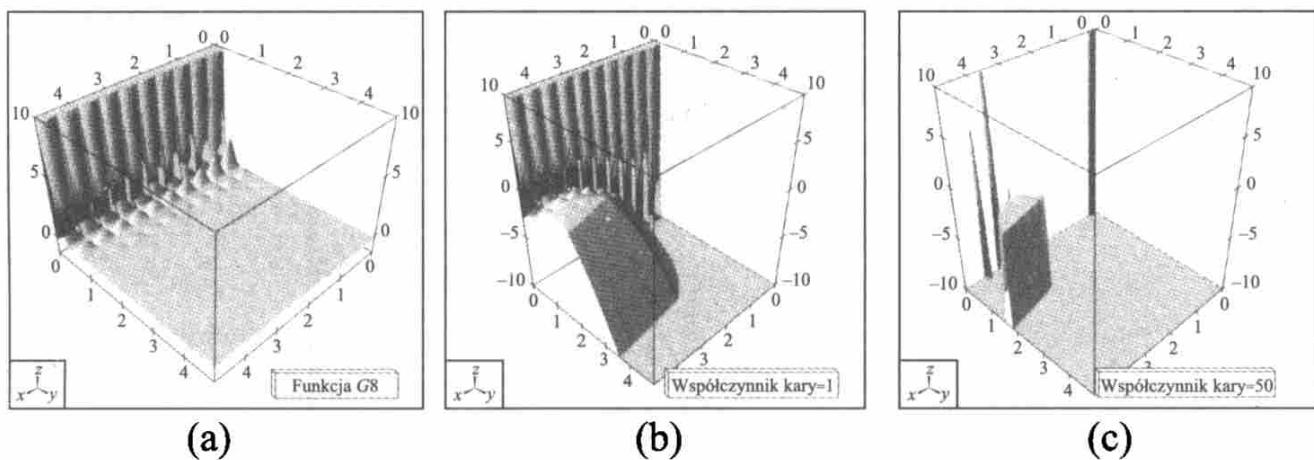
przy ograniczeniach

$$\begin{aligned} c1(\mathbf{x}) &= x_1^2 - x_2 + 1 \leq 0 \\ c2(\mathbf{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{aligned}$$

i zakresach

$$0 \leq x_1 \leq 10 \text{ oraz } 0 \leq x_2 \leq 10$$

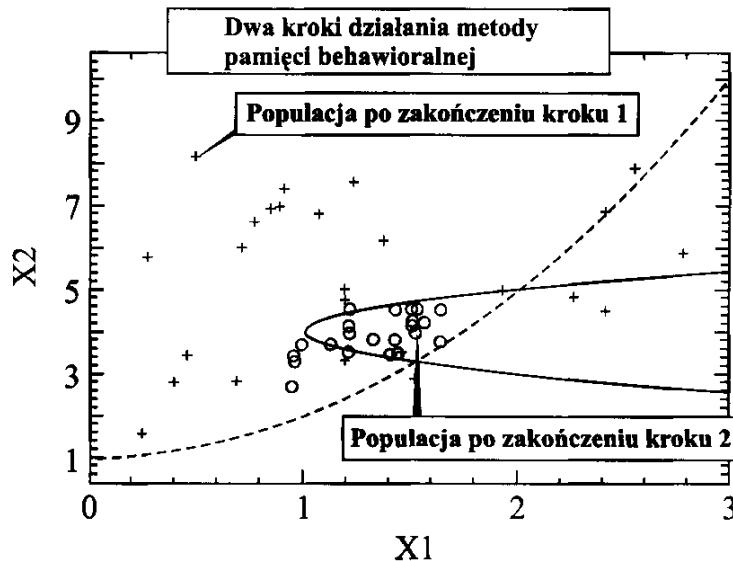
¹W metodzie tej powinno się zastosować tak zwany schemat współdzielenia wartości (zob. rozdział 11), który umożliwia zachowanie różnorodności populacji.



Rysunek 9.9. Funkcja $G8$ (a) wraz z dwoma krajobrazami (wykresami) funkcji przystosowania, które odpowiadają dwóm różnym parametrom kary α ; (b) mała wartość α sprawia, że najwyższe szczyty zostają poza obszarem dopuszczalnym; (c) duża wartość α sprawia, że obszar dopuszczalny wygląda jak igła w stogu siana (wszystkie rysunki, żeby zapewnić dobrą czytelność, przedstawiają tylko wartości między -10 a 10)

Funkcja $G8$ ma wiele optimów lokalnych. Najwyższe jej wzniesienia są rozmieszczone wzdłuż osi x , np. $G8(0,00015, 0,0225) > 1540$. Funkcja $G8$ ma w dopuszczalnym obszarze dwa maksima o prawie tej samej wartości $0,1$. Na rysunku 9.9 przedstawiono krajobraz funkcji przystosowania dla różnych parametrów kary, tzn. wykresy funkcji $G8(\mathbf{x}) - \alpha(c1(\mathbf{x})^+ + c2(\mathbf{x})^+)$ dla różnych wartości α . Wszystkie wartości funkcji na rysunku są ograniczone do obszaru między -10 a 10 , dzięki czemu możemy rozpoznać wartości w obszarze dopuszczalnym wokół 0 . W tej sytuacji każda próba użycia parametru kary jest trudnym zadaniem. Małe wartości parametru kary powodują, że obszar dopuszczalny jest schowany między o wiele wyższymi wzniesieniami, obrazującymi przystosowanie po uwzględnieniu kary (rys. 9.9b), parametry na tyle duże, żeby obszar dopuszczalny naprawdę wybijał się wśród wartości obłożonych karą, oznaczającą pionowe brzegi na granicy obszarów (rys. 9.9c). Wyobrażenie sobie algorytmu ewolucyjnego, który potrafi wspinać się po takim pionowym klifie, jest trudne. Znalezienie maksimum globalnego takiej obciążonej karami funkcji przystosowania wymaga szczęśliwego ruchu wychodzącego poza duży płaski obszar rozwiązań o małym przystosowaniu.

Z drugiej strony, jak pokazano na rys. 9.10, metoda pamięci behawioralnej bez problemu doprowadza do umieszczenia 80% populacji najpierw w obszarze dopuszczalnym ze względu na ograniczenie $c1$ (krok 1), a potem w obszarze dopuszczalnym dla obu ograniczeń (krok 2). Optymalizacja $G8$ w tak uzyskanym obszarze jest już łatwa (zauważmy, że, jak widać na rys. 9.9, na tym obszarze mamy do czynienia z prostym krajobrazem funkcji przystosowania o dwóch optimach). Przy każdym kroku należy tutaj jednak zachowywać różnorodność populacji. Populacja wynikowa będzie stosowana jako podstawa następnej fazy ewolucji, a zatem próbki z docelowego obszaru dopuszczalnego powinny być w niej rozłożone w miarę równomiernie. Ponieważ funkcja oceny w tym obszarze dopuszczalnym ma dwa optima, musimy uważać, żeby zlokalizować oba wznie-



Rysunek 9.10. Ilustracja przykładowego przebiegu działania metody pamięci behawioralnej dla funkcji G_8 ; powiększenie okolic obszaru dopuszczalnego. Linia przerywana oznacza ograniczenie c_1 , linia ciągła ograniczenie c_2 . W populacji po zakończeniu kroku 1 znajdują się głównie próbki z obszaru dopuszczalnego ze względu na c_1 , zaś populacja po zakończeniu kroku 2 zawiera głównie próbki z obszaru dopuszczalnego ze względu na c_1 i c_2

sienia. Uzyskujemy to za pomocą schematu współdzielenia (zob. rozdział 10). Zagadnienie to jest jeszcze ważniejsze, gdy obszar dopuszczalny składa się z wielu różnych spójnych składowych, co pokazano w [416] na przykładzie podobnego problemu o odrobinę zmienionym ograniczeniu c_2 .

Metoda z zapewnianiem przewagi punktów dopuszczalnych. Powell i Skolnick w [362] użyli metody opartej na klasycznym podejściu z karami, ale z jednym ważnym dodatkiem. Każdy osobnik był oceniany zgodnie ze wzorem

$$eval(\mathbf{x}) = f(\mathbf{x}) + r \sum_{j=1}^m f_j(\mathbf{x}) + \theta(t, \mathbf{x})$$

przy czym r jest stałą. Wprowadzony tutaj składnik $\theta(t, \mathbf{x})$ jest dodatkową funkcją zależną od iteracji, która ma wpływ na ocenę rozwiązań niedopuszczalnych. W metodzie tej chodzi o to, że osobniki dopuszczalne i niedopuszczalne są rozróżniane dzięki dodatkowej heurystyce (wcześniej zaproponowanej w [391]). Dla każdego dopuszczalnego osobnika \mathbf{x} i każdego niedopuszczalnego \mathbf{y} mamy $eval(\mathbf{x}) < eval(\mathbf{y})$, tzn. każde rozwiązanie dopuszczalne jest lepsze niż jakiekolwiek niedopuszczalne. Można to osiągnąć na wiele sposobów. Jedną z możliwości jest ustawienie

$$\theta(t, \mathbf{x}) = \begin{cases} 0 & \text{jeśli } \mathbf{x} \in \mathcal{F} \\ \max\{0, \max_{\mathbf{x} \in \mathcal{F}}\{f(\mathbf{x})\} - \\ - \min_{\mathbf{x} \in \mathcal{S}-\mathcal{F}}\{f(\mathbf{x}) + r \sum_{j=1}^m f_j(\mathbf{x})\}\} & \text{w przeciwnym wypadku} \end{cases}$$

Osobniki niedopuszczalne są karane w taki sposób, że nie mogą być lepsze od najgorszego dopuszczalnego (tzn. $\max_{\mathbf{x} \in \mathcal{F}} \{f(\mathbf{x})\}$)¹.

Michalewicz w [308] sprawdził tę metodę na jednym z problemów pochodzących z [227]. Należy zminimalizować funkcję

$$\begin{aligned} G9(\mathbf{x}) = & (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^7 + 7x_6^2 + \\ & + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \end{aligned}$$

przy ograniczeniach:

$$\begin{aligned} 127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 &\geq 0 \\ 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 &\geq 0 \\ 196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 &\geq 0 \\ -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 &\geq 0 \end{aligned}$$

i zakresach

$$-10,0 \leq x_i \leq 10,0, \quad i = 1, \dots, 7$$

Problem ten ma cztery ograniczenia nieliniowe. Funkcja $G9$ jest nieliniowa i ma globalne minimum w punkcie

$$\begin{aligned} \mathbf{x}^* = & (2,330499, 1,951372, -0,4775414, 4,365726, -0,6244870, \\ & 1,038131, 1,594227) \end{aligned}$$

przy czym $G9(\mathbf{x}^*) = 680,6300573$. W tym globalnym optimum aktywne są dwa ograniczenia – pierwsze i ostatnie. Metoda Powella i Skolnicka w tym przykładzie zadziałała bardzo przyzwoicie. Najlepsze znalezione rozwiązanie dawało 680,934. Stosując tę metodę, napotykamy różne trudności przy znajdowaniu rozwiązań dopuszczalnych dla innych testowych przykładów z [308].

W niedawno opublikowanej pracy [97] podejście to zmodyfikowano przez wykorzystanie doboru turniejowego z użyciem funkcji oceny:

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{jeśli } \mathbf{x} \text{ jest dopuszczalne} \\ f_{\max} + \sum_{j=1}^m f_j(\mathbf{x}) & \text{w przeciwnym wypadku} \end{cases}$$

przy czym f_{\max} jest wartością funkcji dla najgorszego rozwiązania dopuszczalnego w populacji. Główna różnica między tym podejściem a podejściem Powella i Skolnicka polega na tym, że tutaj wartość funkcji oceny nie jest brana pod uwagę przy ocenie rozwiązania niedopuszczalnego. Dodatkowo wprowadzono tutaj schemat niszowania (rozdział 10), który umożliwia utrzymanie różnorodności rozwiązań dopuszczalnych. W związku z tym poszukiwanie początkowo skupia się na odnajdywaniu rozwiązań dopuszczalnych, a następnie, gdy już zostanie znaleziona odpowiednia ilość rozwiązań dopuszczalnych, algorytm znajduje lepsze rozwiązanie dopuszczalne przez utrzymywanie odpowiednio różnorodnego

¹Powell i Skolnick w [362] osiągnęli ten sam wynik, odwzorowując rozwiązania dopuszczalne w odcinek $(-\infty, 1)$, a niedopuszczalne w odcinek $(1, \infty)$. Ta różnica implementacji nie jest istotna dla metod rankingowych i turniejowych.

zbioru rozwiązań w dopuszczalnym obszarze. Nie ma potrzeby wprowadzania tutaj współczynników kary, gdyż rozwiązania dopuszczalne są zawsze oceniane lepiej niż niedopuszczalne, a rozwiązania niedopuszczalne są porównywane tylko na podstawie naruszonych ograniczeń. Sugerujemy tutaj wykonanie normalizacji ograniczeń $f_j(\mathbf{x})$.

Poprawianie osobników niedopuszczalnych. GENOCOP III (następca wcześniejszych systemów GENOCOP) dysponuje możliwością poprawiania rozwiązań niedopuszczalnych oraz umożliwia korzystanie z pewnych mechanizmów koewolucyjnych [317]. Podobnie jak w pierwotnym GENOCOP (punkt 9.2.1) są tutaj eliminowane równania liniowe, jest zmniejszana liczba zmiennych, a nierówności liniowe są odpowiednio modyfikowane. Wszystkie punkty początkowej populacji spełniają ograniczenia liniowe. W kolejnych pokoleniach specjalne operatory zachowują dopuszczalność ze względu na te ograniczenia. Zbiór punktów, które spełniają ograniczenia liniowe, oznaczamy $\mathcal{F}_l \subseteq \mathcal{S}$.

W równaniach nielinowych jest konieczny dodatkowy parametr (γ) , który określa precyzję działania systemu. Wszystkie równania nieliniowe $h_j(\mathbf{x}) = 0$ (dla $j = q + 1, \dots, m$) są zastępowane parą nierówności:

$$-\gamma \leq h_j(\mathbf{x}) \leq \gamma$$

W ten sposób mamy do czynienia wyłącznie z nierównościami nieliniowymi. Te nierówności nieliniowe nakładają dalsze ograniczenia na zbiór \mathcal{F}_l . Określają one część $\mathcal{F} \subseteq \mathcal{F}_l$ przestrzeni poszukiwań \mathcal{S} zawierającą rozwiązania w pełni dopuszczalne.

Tworząc GENOCOP III, rozszerzono GENOCOP tak, by utrzymywał dwie oddzielne populacje, takie że rozwój jednej wpływa na ocenę osobników w drugiej. Pierwsza populacja P_s składa się z tak zwanych punktów poszukiwania (ang. *search points*), które pochodzą z \mathcal{F}_l i spełniają ograniczenia liniowe problemu. Jak wspomnieliśmy wcześniej, dopuszczalność tych punktów w sensie ograniczeń liniowych jest utrzymywana za pomocą specjalnych operatorów (zob. punkt 9.2.1).

Druga populacja P_r składa się z tak zwanych punktów odniesienia z \mathcal{F} (ang. *reference points*). Punkty te są w pełni dopuszczalne, tzn. spełniają *wszystkie* ograniczenia¹. Punkty odniesienia \mathbf{r} z P_r , jako punkty dopuszczalne, są oceniane bezpośrednio przez funkcję oceny, tzn. $eval(\mathbf{r}) = f(\mathbf{r})$. Jednakże punkty z P_s są „poprawiane” na potrzeby oceny, a sam proces poprawiania działa następująco. Założymy, że mamy punkt poszukiwania $\mathbf{s} \in P_s$. Jeśli $\mathbf{s} \in \mathcal{F}$, to $\mathbf{s} = f(\mathbf{s})$, gdyż \mathbf{s} jest w pełni dopuszczalne. W przeciwnym wypadku (tzn. gdy $\mathbf{s} \notin \mathcal{F}$) system wybiera² jeden z punktów odniesienia, powiedzmy \mathbf{r} z P_r ,

¹ Jeśli GENOCOP III ma kłopot ze zlokalizowaniem takich punktów odniesienia w trakcie inicjowania, to prosi użytkownika o podanie ich. W sytuacjach, gdy stosunek $|\mathcal{F}|/|\mathcal{S}|$ jest bardzo mały, może się zdarzyć, że początkowy zbiór punktów odniesienia może się składać z wielu kopii jednego punktu dopuszczalnego.

² Lepsze punkty odniesienia mają większe szanse na wybranie. Użyto tutaj metody nieliniowego wyboru na podstawie rankingu.

i tworzy ciąg losowych punktów \mathbf{z} z odcinka między \mathbf{s} a \mathbf{r} , generując losowe liczby a z zakresu $\langle 0, 1 \rangle$ i stosując wzór $\mathbf{z} = a\mathbf{s} + (1 - a)\mathbf{r}$ ¹. Gdy zostanie znaleziony w pełni dopuszczalny punkt \mathbf{z} , obliczamy $eval(\mathbf{s}) = eval(\mathbf{z}) = f(\mathbf{z})$ ².

Dodatkowo, jeśli $f(\mathbf{z})$ jest lepsze niż $f(\mathbf{r})$, to punkt \mathbf{z} zastępuje \mathbf{r} jako nowy punkt odniesienia w populacji punktów odniesienia P_r . Punkt \mathbf{z} także zastępuje \mathbf{s} w populacji punktów poszukiwania P_s , przy czym dzieje się to z określonym prawdopodobieństwem zastąpienia p_r .

Strukturę GENOCOP III przedstawiono na rys. 9.11, procedurę oceny punktów poszukiwania (które nie zawsze są w pełni dopuszczalne) z populacji P_s podano na rys. 9.12.

```

procedure GENOCOP III
begin
     $t \leftarrow 0$ 
    inicjuj  $P_s(t)$ 
    inicjuj  $P_r(t)$ 
    oceń  $P_s(t)$ 
    oceń  $P_r(t)$ 
    while (not warunek zakończenia) do
        begin
             $t \leftarrow t + 1$ 
            wybierz  $P_s(t)$  z  $P_s(t - 1)$ 
            zmodyfikuj  $P_s(t)$ 
            oceń  $P_s(t)$ 
            if  $t \bmod k = 0$  then
                begin
                    zmodyfikuj  $P_r(t)$ 
                    wybierz  $P_r(t)$  z  $P_r(t - 1)$ 
                    oceń  $P_r(t)$ 
                end
            end
        end
    end

```

Rysunek 9.11. Struktura GENOCOP III

Zauważ, że istnieje pewna asymetria przetwarzania populacji punktów poszukiwania P_s i przetwarzania populacji punktów odniesienia P_r . Podczas gdy do populacji P_s stosujemy operatory selekcji i różnicowania w każdym pokoleniu, populacja P_r jest modyfikowana co k pokoleń (pewne dodatkowe zmiany

¹Zwróć uwagę na to, że tak wygenerowany punkt należy do \mathcal{F}_l .

²Ten sam punkt przestrzeni \mathcal{S} może w różnych populacjach otrzymywać różne oceny ze względu na losowość w procesie poprawiania.

```

procedure oceń  $P_s(t)$ 
begin
    dla każdego  $\mathbf{s} \in P_s(t)$  do
        if  $\mathbf{s} \in \mathcal{F}$ 
            then oceń  $\mathbf{s}$  (jako  $f(\mathbf{s})$ ) else
                begin
                    wybierz  $\mathbf{r} \in P_r(t)$ 
                    wygeneruj  $\mathbf{z} \in \mathcal{F}$ 
                    oceń  $\mathbf{s}$  (podając  $f(\mathbf{z})$ )
                    if  $f(\mathbf{r}) > f(\mathbf{z})$  then zastąp  $\mathbf{r}$  przez  $\mathbf{z}$  w  $P_r$ 
                    z prawdopodobieństwem  $p_r$  zastąp  $\mathbf{s}$  przez  $\mathbf{z}$  w  $P_s$ 
                end
            end
        end

```

Rysunek 9.12. Ocena populacji P_s

w P_r są możliwe przy okazji oceny punktów poszukiwania, zob. rys. 9.12). Takie rozwiązanie zastosowano ze względu na efektywność systemu. Poszukiwanie w ramach części dopuszczalnej przestrzeni poszukiwań \mathcal{F} jest traktowane jako zdarzenie w tle. Zauważ też, że kroki „wybór” (selekция) i „modyfikacja” występują w odwrotnej kolejności w pętli ewolucyjnej P_r . Ponieważ często prawdopodobieństwo uzyskania potomstwa dopuszczalnego jest małe, najpierw reprodukowane są osobniki rodzicielskie, a następnie najlepsze osobniki (zówno spośród rodziców, jak i potomstwa) są wybierane jako te, które mają przetrwać.

W GENOCOP III funkcja oceny jest wykorzystywana tylko do określania jakości w pełni dopuszczalnych osobników, a zatem funkcja ta nie jest poprawiana ani zmieniana, jak to było w metodach opartych na funkcjach kary. Wprowadzono tutaj tylko kilka dodatkowych parametrów (np. rozmiar populacji punktów odniesienia, prawdopodobieństwo zastąpienia, częstotliwość stosowania operatorów różnicowania do populacji punktów odniesienia, stopień precyzji γ). Wynikiem tej procedury jest zawsze rozwiązanie dopuszczalne. Przestrzeń rozwiązań niedopuszczalnych \mathcal{F} jest przeszukiwana (populacja P_r) przez wykonywanie odwołań z punktów poszukiwań i stosowanie operatorów różnicowania co k pokoleń (zob. rys. 9.11). Otoczenia lepszych punktów odniesienia są badane częściej. Niektóre z w pełni dopuszczalnych punktów są przenoszone do populacji punktów poszukiwań P_s (proces zastępowania), gdzie podlegają dodatkowym przekształceniom za pomocą specjalnych operatorów.

Jednym z najciekawszych parametrów przedstawianego systemu jest prawdopodobieństwo zastąpienia p_r (zastąpienie \mathbf{s} przez \mathbf{z} w populacji punktów poszukiwań P_s , zob. rys. 9.12). Z eksperymentów opisanych w [317] wynika, że dla obranego zestawu testującego najlepsze wyniki dawało prawdopodobieństwo $p_r = 0,2$. Na przykład, następujący problem z [227] okazał się bardzo

trudny dla wszystkich wcześniej opisanych w literaturze metod. Problem ten polega na minimalizacji

$$G10(\mathbf{x}) = x_1 + x_2 + x_3$$

przy ograniczeniach:

$$1 - 0,0025(x_4 + x_6) \geq 0$$

$$1 - 0,0025(x_5 + x_7 - x_4) \geq 0$$

$$1 - 0,01(x_8 - x_5) \geq 0$$

$$x_1 x_6 - 833,33252 x_4 - 100 x_1 + 83\,333,333 \geq 0$$

$$x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0$$

$$x_3 x_8 - 1\,250\,000 - x_3 x_5 + 2500 x_5 \geq 0$$

i zakresach

$$100 \leq x_1 \leq 10\,000, \quad 1000 \leq x_i \leq 10\,000 \text{ dla } i = 2, 3$$

$$10 \leq x_i \leq 1000 \text{ dla } i = 4, \dots, 8$$

Problem ma trzy ograniczenia liniowe i trzy nieliniowe. Funkcja $G10$ jest liniowa i ma minimum globalne w punkcie

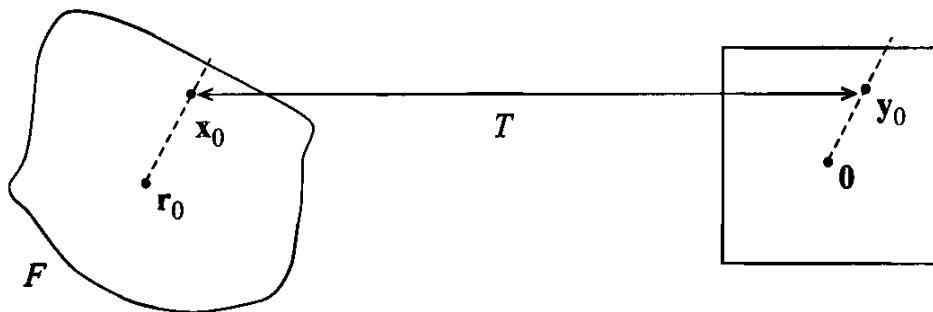
$$\begin{aligned} \mathbf{x}^* = & (579,3167, 1359,943, 5110,071, 182,0174, 295,5985, \\ & 217,9799, 286,4162, 395,5979) \end{aligned}$$

przy czym $G10(\mathbf{x}^*) = 7049,330923$. W tym optimum globalnym jest aktywnych sześć ograniczeń. Dla tego problemu najlepszy uzyskany przez GENOCOP III wynik wynosił 7286,650 – co jest lepszą wartością od dowolnej z najlepszych wartości przedstawionych w pracy [308].

Podobną efektywność uzyskano dla dwóch innych problemów: $G9$ (z wynikiem 680,640) i $G7$ (z wynikiem 25,883). Co ciekawe, odchylenie standardowe wyników uzyskanych za pomocą GENOCOP III jest bardzo małe. Dla problemu $G9$, na przykład, wszystkie wyniki były między 680,640 a 680,889. Z kolei inne systemy dawały wyniki bardzo zróżnicowane (między 680,642 a 689,660; zob. [308]). Wszystkie uzyskane wartości \mathbf{x} były oczywiście dopuszczalne, a nie zawsze tak było w wypadku innych systemów (np. GENOCOP II dał wartość 18,917 dla $G7$ – systemy oparte na metodach zespołu Homaifara [231] oraz Powella i Skolnicka [362] dały dla problemu $G10$ wyniki odpowiednio 2282,723 i 2101,367, które oczywiście były niedopuszczalne).

9.2.4. Metody oparte na dekoderach

Zastosowanie dekoderów stanowi ciekawe alternatywne podejście przy opracowywaniu algorytmów ewolucyjnych, ale do problemów ciągłych zostały użyte niedawno [272, 273]. Jest łatwo określić odwzorowanie wzajemnie jednoznaczne dowolnej wypukłej przestrzeni rozwiązań dopuszczalnych \mathcal{F} na n -wymiarową kostkę $[-1, 1]^n$ (zob. rys. 9.13).



Rysunek 9.13. Odwzorowanie T przestrzeni \mathcal{F} na kostkę $[-1, 1]^n$ (przypadek dwuwymiarowy)

Zauważ, że dowolny punkt (różny od $\mathbf{0}$) $\mathbf{y}_0 = (y_{0,1}, \dots, y_{0,n}) \in [-1, 1]^n$ określa odcinek od $\mathbf{0}$ do brzegu kostki. Odcinek ten jest opisany wzorem

$$\mathbf{y}_i = \mathbf{y}_{0,t} \cdot t \quad \text{dla } i = 1, \dots, n$$

przy czym t zmienia się od 0 do $t_{max} = 1 / \max\{|y_{0,1}|, \dots, |y_{0,n}|\}$. Dla $t = 0$ mamy $\mathbf{y} = \mathbf{0}$, a dla $t = t_{max}$ mamy $\mathbf{y} = (y_{0,1}t_{max}, \dots, y_{0,n}t_{max})$ – punkt brzegowy kostki $[-1, 1]^n$. W rezultacie odpowiedni (dla $\mathbf{y}_0 \in [-1, 1]^n$) punkt dopuszczalny $\mathbf{x}_0 \in \mathcal{F}$ (brany względem pewnego punktu odniesienia¹ \mathbf{r}_0) jest określony jako

$$\mathbf{x}_0 = \mathbf{r}_0 + \mathbf{y}_0 \cdot \tau$$

przy czym $\tau = \tau_{max}/t_{max}$ oraz τ_{max} jest określone z dowolną dokładnością za pomocą procedury wyszukiwania binarnego, tak żeby punkt

$$\mathbf{r}_0 + \mathbf{y}_0 \cdot \tau_{max}$$

był punktem brzegowym zbioru rozwiązań dopuszczalnych \mathcal{F} .

Powyzsze odwzorowanie spełnia wszystkie wymagania stawiane „dobremu” dekoderowi. Przekształcenie to, oprócz tego, że jest wzajemnie jednoznaczne, jest szybkie i ma własność „lokalności” (tzn. punkty bliskie siebie przed wykonaniem odwzorowania są bliskie po jego wykonaniu). Istnieje kilka dodatkowych cech, które sprawiają, że przedstawiona metoda jest interesująca.

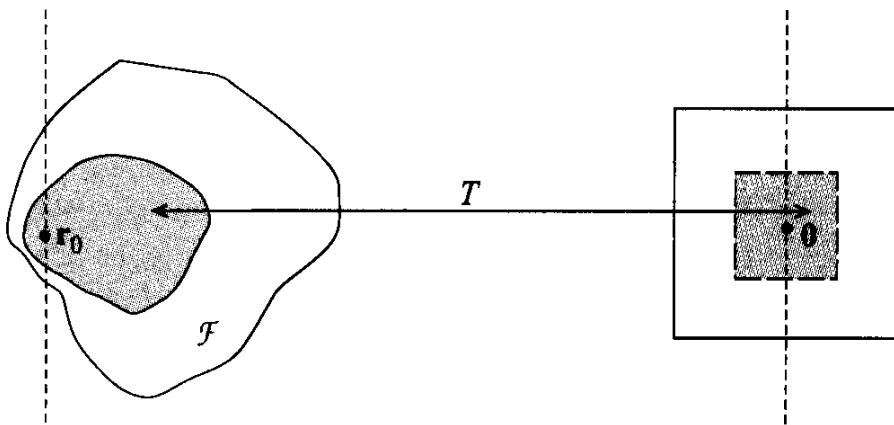
- Inaczej niż w wypadku innych metod postępowania z ograniczeniami, nie ma tutaj potrzeby ustalania żadnych dodatkowych parametrów (np. częstotliwości siedmiu operatorów w GENOCOP [307] lub współczynników kary). Są wymagane tylko podstawowe parametry algorytmu ewolucyjnego (np. rozmiar populacji, rodzaj zastosowanych operatorów różnicowania).
- Inaczej niż w niektórych innych metodach postępowania z ograniczeniami, nie ma potrzeby stosowania żadnych specjalnych operatorów, które

¹Punkt odniesienia \mathbf{r}_0 to dowolny punkt wewnętrzny zbioru wypukłego \mathcal{F} . Zauważ, że założenie o wypukłości \mathcal{F} nie jest konieczne. Wystarczy istnienie punktu odniesienia \mathbf{r}_0 , dla którego każdy odcinek zaczynający się w \mathbf{r}_0 przecina brzeg \mathcal{F} dokładnie w jednym punkcie. Warunek ten jest spełniony przez każdy wypukły zbiór \mathcal{F} .

utrzymywałyby dopuszczalność rozwiązań (np. operatorów GENOCOP [307] zapewniających zachowanie ograniczeń liniowych czy specjalnych operatorów granicznych, które umożliwiają poszukiwanie granicy między częścią dopuszczalną a niedopuszczalną przestrzeni poszukiwań). Każdy algorytm ewolucyjny może być użyty wraz z zaproponowanym tutaj odwzorowaniem.

- Inaczej niż w większości innych metod postępowania z ograniczeniami (tzn. w metodach, które zachowują rozwiązania niedopuszczalne), nie ma potrzeby wykonywania oceny rozwiązań niedopuszczalnych (w szczególności nie ma potrzeby obciążania ich karami, dostosowywania współczynników kary czy poprawiania rozwiązań).
- Inaczej niż w innych metodach postępowania z ograniczeniami (np. w metodach opartych na funkcjach kary lub w metodach hybrydowych), przedstawiona tutaj metoda daje zawsze dopuszczalne rozwiązanie.

Przedstawione tutaj podejście możemy rozszerzyć o dodatkowe iteracyjne poprawianie rozwiązań oparte na wzajemnej zależności lokalizacji punktu odniesienia oraz efektywności proponowanego podejścia. Umiejscowienie punktu r_0 ma wpływ na stopień i rodzaj „deformacji” dziedziny optymalizowanej funkcji. Użyty tutaj algorytm ewolucyjny nie optymalizuje funkcji oceny, ale raczej pewną inną funkcję równoważną topologicznie z pierwotną funkcją. Rozważmy na przykład przypadek, gdy punkt odniesienia znajduje się w pobliżu brzegu obszaru rozwiązań dopuszczalnych \mathcal{F} – łatwo możemy stwierdzić, że przekształcenie T jest bardzo nieregularne. Część kostki $[-1, 1]^2$, która znajduje się na lewo od pionowej linii, jest przekształcana w o wiele mniejszą część zbioru \mathcal{F} niż część na prawo od niej (zob. rys. 9.14).



Rysunek 9.14. Wpływ umiejscowienia punktu odniesienia na przekształcenie T

W świetle tych obserwacji wydaje się intuicyjnie słuszne umieszczanie punktu odniesienia w otoczeniu spodziewanego optimum, zwłaszcza jeśli to optimum znajduje się niedaleko brzegu zbioru \mathcal{F} . W takim przypadku obszar między krawędzią \mathcal{F} a punktem odniesienia r_0 jest przeszukiwany z większą dokładnością. Gdy jednak nie mamy informacji na temat przybliżonej lokalizacji optymalnego rozwiązania, powinniśmy umieścić punkt odniesienia blisko

geometrycznego środka zbioru \mathcal{F} . Możemy to zrobić, próbując zbiór \mathcal{F} i przyjmując

$$\mathbf{r}_0 = \frac{1}{k} \sum_{i=1}^k \mathbf{x}_i$$

przy czym \mathbf{x}_i to próbki ze zbioru \mathcal{F} . Możemy też wykorzystać wspomniany efekt w ramach iteracyjnego ulepszania najlepszego dotychczas znalezionej rozwiązania. W tym celu musimy powtórzyć proces optymalizacji z użyciem nowego punktu odniesienia \mathbf{r}'_0 znajdującego się na odcinku między bieżącym punktem odniesienia \mathbf{r}_0 a najlepszym dotychczas znalezionym rozwiązaniem \mathbf{b}

$$\mathbf{r}'_0 = t \cdot \mathbf{r}_0 + (1 - t) \cdot \mathbf{b}$$

przy czym $t \in (0, 1]$ powinno być bliskie zeru. Taka zmiana umiejscowienia punktu odniesienia oznacza, że w następnej iteracji otoczenie znalezionej optimum zostanie przeszukane dokładniej niż pozostała część obszaru dopuszczalnego. Metoda ta, jak pokazują doświadczenia, może dawać dobre wyniki dla problemów, w których optymalne rozwiązania są rozmieszczone w pobliżu granicy obszaru dopuszczalnego [272, 273]. Podejście to możemy także rozszerzyć tak, żeby było użyteczne dla zadań z niewypukłymi przestrzeniami poszukiwań [272, 273].

9.2.5. Metody hybrydowe

Opracowywanie metod hybrydowych, łączących algorytmy ewolucyjne z procedurami deterministycznymi, nie jest trudne¹. Waagen i in. w pracy [484] połączycyli na przykład reprezentację zmiennopozycyjną oraz różnicowanie Gaussa z opracowaną przez Hooke'a-Jeevesa metodą zbioru kierunków. Otrzymaną procedurę przetestowano na trzech ciągłych funkcjach bez ograniczeń. Myung i in. w pracy [332] zastosowali podobne podejście, ale przeprowadzili testy na ciągłych problemach z ograniczeniami. Zastosowali reprezentację zmiennopozycyjną oraz różnicowanie Gaussa, jednak połączycyli je z metodą Lagrange'a, którą opracowali Maa i Shanblatt [294], tworząc dwufazowy algorytm poszukiwania. W trakcie pierwszej fazy algorytm ewolucyjny optymalizuje funkcję

$$eval(\mathbf{x}) = f(\mathbf{x}) + \frac{s}{2} \left(\sum_{j=1}^m f_j^2(\mathbf{x}) \right)$$

przy czym s jest stałe. Po zakończeniu tej fazy zaczyna się druga działająca zgodnie z metodą, którą podali Maa i Shanblatt. Procedura ta na wejściu otrzymuje najlepsze rozwiązanie uzyskane w pierwszej fazie i prowadzi obliczenia dopóty, dopóki system

$$\mathbf{x}' = -\nabla f(\mathbf{x}) - \left[\sum_{j=1}^m \nabla f_j(\mathbf{x})(s f_j(\mathbf{x}) + \lambda_j) \right]$$

¹Robiono to już w latach sześćdziesiątych XX wieku [253].

nie osiągnie stanu równowagi, przy czym mnożniki Lagrange'a są uaktualniane zgodnie ze wzorem $\lambda'_j = \epsilon s f_j$ dla pewnej małej dodatniej stałej ϵ .

Tę metodę hybrydową pomyślnie zastosowano do kilku testowych przykładów. Zminimalizujmy za jej pomocą

$$G11(\mathbf{x}) = x_1^2 + (x_2 - 1)^2$$

przy ograniczeniu nieliniowym

$$x_2 - x_1^2 = 0$$

oraz zakresach

$$-1 \leq x_i \leq 1, \quad i = 1, 2$$

Znanym globalnym rozwiązaniem jest $\mathbf{x}^* = (\pm 0,70711, 0,5)$, a $G11(\mathbf{x}^*) = 0,75000455$. Przy rozwiązywaniu tego problemu pierwsza faza procesu ewolucyjnego po 100 pokoleniach dała wynik $(\pm 0,70711, 0,5)$, druga faza doprowadziła system do rozwiązania globalnego. Metoda ta jednak nie została przetestowana na problemach z większą liczbą wymiarów.

Istnieje wiele innych metod postępowania z ograniczeniami, które są godne uwagi. W niektórych z nich na przykład są używane wartości funkcji oceny f oraz funkcji kar f_j ($j = 1, \dots, m$) jako elementy wektora i są stosowane metody wielokryterialne do minimalizacji wszystkich elementów wektora. W pracy [410], na przykład, Schaffer wybiera część populacji odpowiadającą procentowo ułamkowi $1/(m+1)$ na podstawie celu. Parmee i Purchase w [353] włączyli to podejście do metod rozwiązywania problemów związanych z silnie ograniczonymi przestrzeniami projektowymi. Z kolei Surry i in. [458] zastosowali metodę, w której wszystkie osobniki w populacji są umieszczane na liście rankingowej zgodnie z naruszanymi przez nie ograniczeniami. Miejsce na tej liście r razem z wartością funkcji oceny f dają w rezultacie problem optymalizacji dwukryterialnej. Podejście to dało dobre wyniki przy optymalizacji dostawczych sieci gazowych [458].

W pracy [222] Hinterding i Michalewicz zastosowali wektor naruszeń ograniczeń do wspomagania procesu wyboru rodziców. Długość wektora odpowiada liczbie ograniczeń. Jeśli, na przykład, pierwszy rodzic spełnia ograniczenia 1, 2 i 4 (powiedzmy z pięciu), to jest kojarzony z osobnikiem spełniającym ograniczenia 3 i 5. Można uwzględnić wiedzę na temat ograniczeń problemu w przestrzeni przekonań w algorytmach kulturowych [388]. Algorytmy te umożliwiają przeprowadzenie efektywnego przeszukiwania przestrzeni poszukiwań [389].

Ostatnio Kelly i Laguna [257] opracowali nową procedurę rozwiązywania trudnych problemów optymalizacyjnych z ograniczeniami, która

... w ciągu kilku minut generowała rozwiązania optymalne i bliskie optymalnym złożonych problemów nieliniowych. Rozwiązywanie tych problemów innymi metodami zajmowało wiele dni z powodu weryfikowania rozwiązań optymalnych. W problemach tych występowały niewypukłe, nieróżniczkowalne oraz mieszane funkcje dyskretnie na obszarach wyznaczonych przez ograniczenia. Przestrzeń

rozwiązań dopuszczalnych w sposób jawny zawiera ograniczenia programowania liniowego i ograniczenia programowania całkowitoliczbowego oraz niejawnie ograniczenia nieliniowe wprowadzone przez funkcje kary. Funkcje celu, które są przedmiotem procesu optymalizacji, nie zawsze mają matematycznie wyrażalną reprezentację, a ich obliczenie może wymagać wykonania symulacji. Dzięki takiej elastyczności podejście to może być także zastosowane do optymalizacji w problemach z niepewnością (co zwiększa związek z praktycznymi zastosowaniami).

Brzmi naprawdę nieźle! Metoda ta łączy algorytmy ewolucyjne z: 1) przeszukiwaniem rozproszonym, 2) procedurami mieszanej optymalizacji całkowitoliczbowej oraz 3) strategiami pamięci adaptacyjnej w przeszukiwaniu z tabu. Możesz sobie bez wątpienia wyobrazić wiele więcej sposobów konstrukcji hybrydowych metod obsługi ograniczeń.

9.3. Podsumowanie

Przedstawiliśmy wiele aspektów problemów optymalizacyjnych z ograniczeniami, a także podaliśmy przegląd pewnych podejść do rozwiązywania tych problemów. Chociaż może wydawać się, że przegląd ten był trochę przydługi, w rzeczywistości było to bardzo skrótowe przedstawienie zagadnienia. Poznaliśmy tylko powierzchniowo to, co zostało zrobione i co pozostaje do zrobienia. Musimy jednak przyznać, że nie jest jasne, jakie cechy sprawiają, że problem jest trudny dla algorytmów ewolucyjnych (lub innych metod) i to mimo że problemy można scharakteryzować za pomocą rozmaitych parametrów: liczby ograniczeń liniowych, liczby ograniczeń nieliniowych, liczby ograniczeń równościowych, liczby ograniczeń aktywnych, stosunku $\rho = |\mathcal{F}|/|\mathcal{S}|$ rozmiaru przestrzeni rozwiązań dopuszczalnych do rozmiaru całości oraz typu funkcji oceny określonego za pomocą liczby zmiennych, liczby lokalnych optimów, istnienia pochodnych itd.

W rozdziale 11 omówiliśmy przykłady testowe ($G1-G11$) numerycznych problemów optymalizacyjnych z ograniczeniami [319]. Przykłady te obejmują funkcje oceny wielu różnych rodzajów (liniowe, kwadratowe, sześcienne, wielomianowe, nieliniowe) o różnych liczbach zmiennych i różnych rodzajach ograniczeń (w postaci nierówności liniowych, równań i nierówności nieliniowych) oraz liczbach ograniczeń. Stosunek ρ rozmiaru przestrzeni rozwiązań dopuszczalnych \mathcal{F} do całej przestrzeni poszukiwań \mathcal{S} wahał się w tych przykładach od zera do prawie 100%. Własności topologiczne przestrzeni rozwiązań dopuszczalnych też były różne. Omówione przykłady zestawiono w tab. 9.2, w której dla każdego z nich są podane: liczba n zmiennych, typ funkcji f , względny rozmiar obszaru dopuszczalnego przestrzeni poszukiwań podany jako stosunek ρ , liczba ograniczeń każdego rodzaju (nierówności liniowych NL , równań nieliniowych RN oraz nierówności nieliniowych NN), a także liczba a ograniczeń aktywnych w optimum (łącznie z ograniczeniami równościowymi).

Tabela 9.2. Zestawienie 11 przykładów testowych. Stosunek $\rho = |\mathcal{F}|/|\mathcal{S}|$ został określony eksperymentalnie przez wygenerowanie 1 000 000 losowych punktów z \mathcal{S} i sprawdzenie, które z nich należą do \mathcal{F} (dla $G2$ i $G3$ założyliśmy $k = 50$). NL , RN i NN oznaczają odpowiednio liczbę nierówności liniowych, równań nieliniowych i nierówności nieliniowych

Funkcja	n	Typ funkcji f	ρ	NL	RN	NN	a
$G1$	13	kwadratowa	0,0111%	9	0	0	6
$G2$	k	nieliniowa	99,8474%	0	0	2	1
$G3$	k	wielomianowa	0,0000%	0	1	0	1
$G4$	5	kwadratowa	52,1230%	0	0	6	2
$G5$	4	sześcienna	0,0000%	2	3	0	3
$G6$	2	sześcienna	0,0066%	0	0	2	2
$G7$	10	kwadratowa	0,0003%	3	0	5	6
$G8$	2	nieliniowa	0,8560%	0	0	2	0
$G9$	7	wielomianowa	0,5121%	0	0	4	2
$G10$	8	liniowa	0,0010%	3	0	3	6
$G11$	2	kwadratowa	0,0000%	0	1	0	1

Chociaż odnotowano wiele sukcesów, wyniki wielu testów nie dały jasnego obrazu stopnia trudności problemów. Żaden konkretny parametr, na przykład liczba ograniczeń liniowych, nieliniowych czy aktywnych, nie wystarcza do określenia stopnia trudności problemu. Wiele metod ewolucyjnych doszło bardzo blisko optimów w przykładach $G1$ i $G7$ (gdzie odpowiednio $\rho = 0,0111\%$ i $\rho = 0,0003\%$), podczas gdy większość z nich miała trudności w przykładzie $G10$ (gdzie $\rho = 0,0010\%$). Dwie funkcje kwadratowe (przykłady $G1$ i $G7$) o podobnej liczbie ograniczeń (odpowiednio dziewięć i osiem) oraz o takiej samej liczbie (sześć) ograniczeń aktywnych w optimum okazały się prawdziwym wyzwaniem dla większości z omówionych metod.

Wiele z metod wykazało wrażliwość na obecność rozwiązania dopuszczalnego w początkowej populacji. Nie ma wątpliwości, że konieczne są tutaj głębsza analiza i szersze badania. Zdecydowanie możesz tutaj pomóc, wnosząc jakąś wiedzę. Jak można było się spodziewać, wyniki eksperymentalne z [319] wskazują, że pozostaje otwarte pytanie, jak a priori dokonać właściwego wyboru metody ewolucyjnej do badań problemu optymalizacyjnego. Wydaje się, że sensowną miarę trudności problemu mogą dać bardziej skomplikowane własności (np. cechy funkcji oceny wraz z własnościami topologicznymi obszaru dopuszczalnego). Oprócz tego pomocne mogą być pewne dodatkowe miary cech problemu ze względu na ograniczenia. Jak dotąd tego rodzaju informacje są nam nieznane.

W pracy [319] Michalewicz i Schoenauer napisali:

Wydaje się, że na obecnym etapie badań najbardziej obiecujące jest podejście eksperymentalne obejmujące opracowanie skalowalnego zestawu testowych problemów optymalizacyjnych z ograniczeniami

mi, w którym wiele [...] cech możemy łatwo dostosowywać. Powinno być przy tym możliwe testowanie nowych metod względem korpusu wszystkich dostępnych metod.

Jest oczywiste, że istnieje zapotrzebowanie na parametryzowany generator przykładów testowych, którego można by użyć do systematycznej analizy różnych metod i który zastąpiłby dotychczasowe testowanie ich na kilku wybranych przykładach. Co więcej, nie wiadomo, czy wprowadzenie kilku dodatkowych konkretnych przykładów testowych cokolwiek wnosi. Wspomniany generator przykładów testowych został niedawno opracowany [314, 315]. Czas pokaże, czy zdołamy pojąć istotne własności problemów z ograniczeniami na tyle, żeby umieć dostosowywać nasze algorytmy ewolucyjne do zadań, które przed nami stoją.

X. Czy potrafisz dostosować się do problemu?

Czasami trudno jest zastosować ogólną strategię rozwiązywania problemów do konkretnego zagadnienia, gdyż jest ono w jakimś sensie wyjątkowe. Musimy wówczas przeanalizować problem i opracować metodę, która będzie pasowała do tego przypadku. Rzecz dotyczy wielu zagadnień matematycznych, ponieważ rozwiązanie rzadko obejmuje większą liczbę problemów. Rozważmy, na przykład, następujące zadanie:

Wykaż, że $\lfloor (2 + \sqrt{3})^n \rfloor$ jest nieparzyste dla każdego naturalnego n .

Trudno zastosować tu jakąkolwiek ogólną metodę rozwiązywania problemów. Indukcja matematyczna, na przykład, nie wydaje się odpowiednia. Możemy łatwo wykazać, że dla $n = 1$ podane wyrażenie (przed obcięciem części ułamkowej) ma wartość 3,732, dla $n = 2$ wartość ta wynosi 19,928, dla $n = 3$ wynosi 51,981. Trudne może być jednak wykazanie indukcji: jeśli $\lfloor (2 + \sqrt{3})^n \rfloor$ jest nieparzyste, to $\lfloor (2 + \sqrt{3})^{n+1} \rfloor$ także musi być nieparzyste. Co jest takiego szczególnego w $\sqrt{3} = 1,7320508075688772935$, że każda następna potęga $(2 + \sqrt{3})$ daje liczbę nieparzystą z jakimś ułamkiem?

Inną możliwością użycia ogólnej metody do rozwiązywania tego problemu mogłoby być obliczenie n -tej potęgi i zapisanie otrzymanego wyrażenia:

$$(2 + \sqrt{3})^n = 2^n + n \cdot 2^{n-1} \sqrt{3} + \dots + \sqrt{3}^n$$

ale to niewiele pomaga. Niektóre składniki tego wyrażenia są parzyste, ale inne są pomnożone przez $\sqrt{3}$ w jakieś potędze, co czyni nasze rozumowanie dość mętnym. Stosując to podejście, raczej nie rozwiążemy naszego problemu.

Co zatem możemy zrobić? Zdaje się, że musimy poszukać jakiejś specyficznej cechy tego problemu, czegoś, co pomogłoby nam ruszyć z miejsca. Jest to prawdopodobnie najtrudniejszy moment w rozwiązywaniu problemów i nie ma żadnych zasad określających, jak prowadzić te poszukiwania. Jedyną wskazów-

ką, jaką możemy Ci dać, jest stwierdzenie, że sukces zależy tu w dużej mierze od eksperymentowania i doświadczenia.

Ze zwykłej ciekawości możemy sprawdzić inne potęgi n i zobaczyć, co się stanie. Zaobserwujemy coś interesującego. Zauważliśmy już, że:

$$(2 + \sqrt{3})^1 = 3,732$$

$$(2 + \sqrt{3})^2 = 13,928$$

$$(2 + \sqrt{3})^3 = 51,981$$

Następnie:

$$(2 + \sqrt{3})^4 = 199,995$$

$$(2 + \sqrt{3})^5 = 723,9986$$

$$(2 + \sqrt{3})^6 = 2701,9996$$

Wygląda na to, że większe wartości n dają większą część ułamkową liczby, która zbliża się do 1, podczas gdy część całkowita liczby pozostaje nieparzysta. Otrzymana liczba jest parzysta, jeśli pominiemy część ułamkową, która jest bliska zera dla większych wartości n . Możemy przecież zapisać te liczby jako:

$$(2 + \sqrt{3})^1 = 4 - 0,268$$

$$(2 + \sqrt{3})^2 = 14 - 0,072$$

$$(2 + \sqrt{3})^3 = 52 - 0,019$$

$$(2 + \sqrt{3})^4 = 200 - 0,005$$

$$(2 + \sqrt{3})^5 = 724 - 0,0014$$

$$(2 + \sqrt{3})^6 = 2702 - 0,0004$$

Gdybyśmy mogli wykazać, że $(2 + \sqrt{3})^n$ jest zawsze liczbą parzystą pomniejszoną o ułamek, dowód byłby zakończony. Dla $n = 1$ odejmowany ułamek wynosi $0,268 = 2 - \sqrt{3}$. Jest to dla nas pewna wskazówka. Czy to prawda, że $0,072 = (2 - \sqrt{3})^2$? Tak! Jest również jasne, że $(2 - \sqrt{3})^n$ jest zawsze ułamkiem z przedziału $[0..1]$.

Teraz wystarczy tylko udowodnić, że

$$(2 + \sqrt{3})^n + (2 - \sqrt{3})^n$$

jest parzyste dla wszystkich n . Jest to bardzo proste, ponieważ wszystkie składowe z $\sqrt{3}$ podniesione do nieparzystej potęgi zredukują się (przeciwne znaki w dwóch wyrażeniach), a parzyste potęgi $\sqrt{3}$ są albo mnożone przez 2 (w jakieś potędze), albo dodawane do siebie (np. $3^n + 3^n = 2 \cdot 3^n$ dla parzystego n). To kończy dowód.

Tak naprawdę dowód stanowi tylko ostatni akapit; wszystkie pozostałe rozważania pokazują jedynie rozumowanie prowadzące do uzyskania dowodu. Sam dowód jest bardzo krótki i elegancki, ale nie jest łatwo do niego dojść. Przyjęta metoda musiała być dostosowana do tego konkretnego problemu. Jest

bardzo mało prawdopodobne, że natrafimy kiedyś na podobny problem, w którym opracowana tu metoda znalazłaby zastosowanie.

Konieczność opracowywania metod dopasowanych do konkretnych problemów jest nieco frustrująca. Znacznie lepiej byłoby mieć metodę, która dostosowałaby się do poszczególnych problemów. Wprawdzie istnieją pewne pomysły, jak można używać w ten sposób algorytmów ewolucyjnych (opisujemy je w kolejnym rozdziale), ale zanim do nich przejdziemy, spróbuj zmierzyć się z następującymi problemami i zobacz, jak Ci pójdziesz.

Oto pierwszy z nich:

Znajdź wszystkie sześciocyfrowe liczby, które pomnożone przez dowolną liczbę od 2 do 6 włącznie dadzą liczbę składającą się z tych samych sześciu cyfr co liczba wyjściowa, ale w innej kolejności.

Przestrzeń poszukiwań składa się więc ze wszystkich liczb sześciocyfrowych, od 100 000 do 999 999. Które z nich mają opisaną w zadaniu własność? Przekonaj się najpierw, czy umiesz rozwiązać tę zagadkę bez naszej pomocy.

Wymagana własność zawiera przestrzeń poszukiwań. Tylko niektóre sześciocyfrowe liczby (jeśli w ogóle jakieś) spełniają opisany warunek, tzn. są dopuszczalne. W sumie jest pięć ograniczeń, po jednym dla każdego mnożenia (przez dwa, przez trzy itd.). W jaki sposób ograniczenia te mogą pomóc nam w rozwiązaniu problemu? Jak możemy ograniczyć poszukiwanie?

Przyjrzyjmy się pierwszej (najbardziej znaczącej) cyfrze poszukiwanej liczby. Ostatnie ograniczenie wskazuje, że powinno nią być 1, ponieważ jeśli będzie nią 2 lub więcej, to po pomnożeniu liczby przez 6 otrzymamy liczbę siedmiocyfrową! A zatem to jedno ograniczenie umożliwia zmniejszenie przestrzeni poszukiwań do zbioru wszystkich liczb od 100 000 do 199 999.

Pomyślmy jeszcze chwilę. Jakie są skutki występowania 1 na pierwszej pozycji? Jeśli oznaczymy poszukiwaną liczbę przez x , to liczba $2x$ rozpocznie się cyfrą 2 lub 3. Ogólnie rzecz biorąc, pierwsza cyfra $6x$ będzie większa od pierwszej cyfry $5x$, która będzie większa od pierwszej cyfry $4x$, która z kolei będzie większa od pierwszej cyfry $3x$, która będzie większa od pierwszej cyfry $2x$, która będzie większa od pierwszej cyfry x . Z tej prostej obserwacji możemy wyciągnąć następujące wnioski:

- skoro sześć cyfr tworzących liczby x , $2x$, ..., $6x$ jest takich samych, a każda z tych liczb rozpoczyna się cyfrą większą niż pierwsza cyfra poprzedniej liczby, to znaczy, że występuje sześć różnych cyfr;
- cyfra 0 nie występuje.

Rozumujmy dalej. Jaka jest ostatnia (najmniej znacząca) cyfra liczby x ? Nie może być parzysta, bo wtedy ostatnią cyfrą $5x$ byłoby 0, co nie jest możliwe. Nie może nią być 5, bo wtedy $2x$ miałoby 0 jako ostatnią cyfrę, co też nie jest możliwe. Nie może nią być 1, ponieważ pierwszą cyfrą jest 1, a wszystkie cyfry są różne. Tak więc jedynymi możliwymi cyframi na końcu liczby x są 3, 7 i 9.

Jeśli ostatnią cyfrą x jest 3, to ostatnimi cyframi liczb $2x$, ..., $6x$ są odpowiednio 6, 9, 2, 5 i 8. Jest to niemożliwe, ponieważ sześciocyfrowa liczba

składałaby się z cyfr $\{3, 6, 9, 2, 5, 8\}$, a wiemy już, że zbiór ten musi zawierać cyfrę 1 (gdyż x zaczyna się od 1). Jeśli ostatnią cyfrą x jest 9, to ostatnimi cyframi liczb $2x, \dots, 6x$ są odpowiednio 8, 7, 6, 5 i 4. Jest to niemożliwe z tych samych powodów co poprzednio.

Oznacza to, że ostatnią cyfrą liczby x musi być 7. Poszukujemy więc wśród liczb od 100 007 do 199 997 i tylko co dziesiąta liczba z tego zbioru wymaga rozważenia.

Wiedząc, że ostatnią cyfrą jest 7, wiemy także, że poszukiwane cyfry to $\{1, 2, 4, 5, 7, 8\}$, ponieważ ostatnimi cyframi liczb $2x, \dots, 6x$ są odpowiednio 4, 1, 8, 5 i 2. Umożliwia to nam ustalenie pierwszych cyfr liczb $2x, \dots, 6x$, gdyż muszą one tworzyć szereg rosnący. Wszystkie nasze dotychczasowe odkrycia zebraliśmy na rys. X.1.

$$\begin{aligned} x &= 1 - - - 7 \\ 2x &= 2 - - - 4 \\ 3x &= 4 - - - 1 \\ 4x &= 5 - - - 8 \\ 5x &= 7 - - - 5 \\ 6x &= 8 - - - 2 \end{aligned}$$

Rysunek X.1. Informacje znane na obecnym etapie rozwiązywania problemu.

Kreski oznaczają cyfry, które należy ustalić

Aby ustalić liczbę x , wystarczy już tylko znaleźć permutację zaledwie czterech cyfr: 2, 4, 5 i 8, gdyż pierwszą cyfrą x jest 1, a ostatnią – 7. Problem jest już teraz niewielki, ponieważ do sprawdzenia mamy $4! = 24$ możliwości. Zauważ także, że różnica między $5x$ a $2x$ wynosi $3x$, między $6x$ i $4x - 2x$ itd. Krótko mówiąc, różnica między dwiema różnymi liczbami od x do $6x$ jest liczbą z tego zbioru.

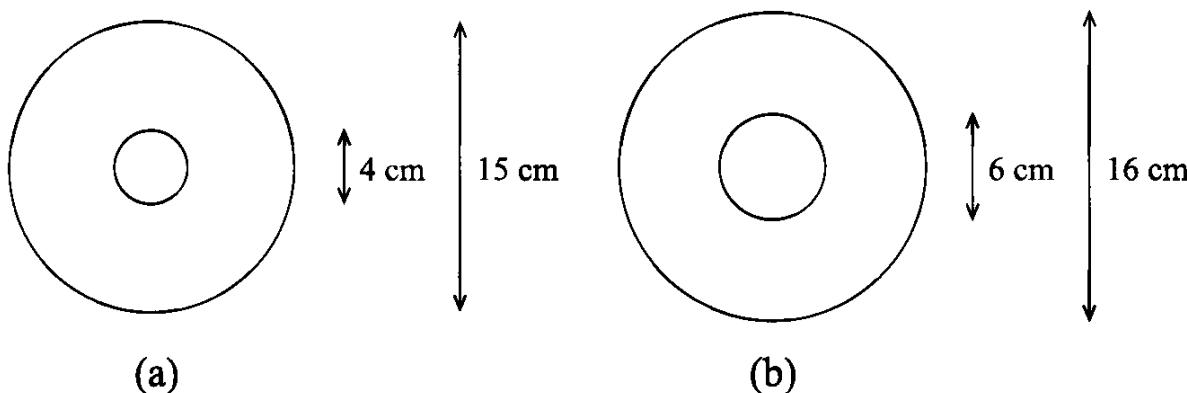
W czym może nam pomóc powyższe spostrzeżenie? Możemy przyjąć następujące twierdzenie: zbiór cyfr o określonym stopniu znaczenia (tzn. najbardziej znacząca, druga pod względem znaczenia itd. aż do najmniej znaczącej), tworzących liczby od x do $6x$, jest zawsze taki sam: $\{1, 2, 4, 5, 7, 8\}$. Innymi słowy, każda „pionowa” kolumna cyfr (rys. X.1) jest także permutacją tych sześciu cyfr $\{1, 2, 4, 5, 7, 8\}$. Powód takiego stanu rzeczy jest bardzo prosty: gdyby cyfra pojawiła się dwa razy w tej samej kolumnie, na przykład dla $5x$ i $2x$, to ich różnica (która jest $3x$) miałaby w tej kolumnie 0 albo 9, co jest niemożliwe, gdyż cyfry te nie należą do rozpatrywanego zbioru.

Te proste spostrzeżenie umożliwia nam zsumowanie sześciu równań z rys. X.1. Ponieważ wiemy, że $7 + 4 + 1 + 8 + 5 + 2 = 27$, możemy zsumować te równania kolumna po kolumnie, gdyż każda z kolumn daje w sumie 27. Lewe strony sumują się do $x + 2x + 3x + 4x + 5x + 6x = 21x$, tak więc

$$21x = 2\,999\,997$$

Z tego wynika, że $x = 142\,857$. Ograniczyliśmy przestrzeń poszukiwań do jednego punktu! Nieźle! A jak Tobie poszło?

I kolejne zadanie dla Ciebie. Jest wiele *praktycznych* problemów, które można by łatwo rozwiązać, używając właściwego modelu. Rozważmy taką sytuację: mamy dwie rolki papieru toaletowego. Średnice wewnętrznych i zewnętrznych kół podano na rys. X.2.



Rysunek X.2. Dwie rolki papieru toaletowego

Każda rolka składa się z 480 listków, ale długość jednego listka w rolce (a) wynosi 16 cm, podczas gdy w drugiej rolce (b) listki mają po 17 cm. Pytanie brzmi: która rolka ma grubsze listki?

Zauważ, że pytanie to jest bardzo ważne (!), a odpowiedź użyteczna. Wielu producentów podaje jedynie informację o liczbie listków. Jeśli jesteśmy zainteresowani ich grubością, musimy wykonać wszystkie niezbędne obliczenia! A zatem do dzieła!

Jest kilka sposobów rozwiązania tego zadania. Najprostszym z nich jest prawdopodobnie obliczenie powierzchni boku obu rolek:

$$\text{powierzchnia rolki (a)} = \pi(7,5^2 - 2^2) = 164,15$$

$$\text{powierzchnia rolki (b)} = \pi(8^2 - 3^2) = 172,79$$

oraz całkowitej długości wszystkich listków obu rolek:

$$\text{długość rolki (a)} = 480 \times 16 = 7680$$

$$\text{długość rolki (b)} = 480 \times 17 = 8160$$

Grubość listków każdej z rolek wynosi odpowiednio

$$\frac{164,15}{7680} = 0,0214 \text{ cm} \quad \text{oraz} \quad \frac{172,79}{8160} = 0,0212 \text{ cm}$$

tak więc rolka (a) ma odrobinę grubsze listki!

Zauważ, że możemy też odpowiedzieć na dodatkowe pytanie: jaka jest liczba warstw w każdej z rolek? Grubość wszystkich warstw w rolce (a) wynosi 5,5 cm, mamy więc:

$$\frac{5,5}{0,0855} \approx 64 \text{ warstwy}$$

(W rolce (b) jest tylko 59 warstw).

Czas na kolejny problem: Wyobraź sobie, że ktoś daje Ci worek złota, który waży między 1 a 40 kg. Żeby problem był bardziej interesujący, przyjmijmy, że ciężar worka może być tylko liczbą całkowitą. Masz do dyspozycji wagę szalkową, a Twoje zadanie polega na znalezieniu zbioru ciężarków o najmniejszej liczności, za pomocą którego można zważyć worek przez zrównoważenie wagi.

Jeśli spojrzymy na nasz problem przez pryzmat potęg dwójki, to dysponujemy ciężarkami

$$1, 2, 4, 8, 16 \text{ i } 32$$

Dowolny ciężar worka między $1 \leq x \leq 40$ możemy określić, umieszczając pewne ciężarki na jednej szalce wagi, a worek na drugiej, na przykład:

$$x = 35 = 1 + 2 + 32$$

Przypomina to zapisywanie liczby całkowitej w systemie dwójkowym. Żeby jednak mieć możliwość zważenia worka ważacego dowolną liczbę kilogramów między 1 a 40, potrzebujemy 6 ciężarków (zob. powyżej).

Jeśli z kolei rozważymy potęgi trójki, uzyskamy lepszy rezultat. Ciężarki, którymi dysponujemy, to

$$1, 3, 9 \text{ i } 27$$

Dowolna liczba całkowita $1 \leq x \leq 40$ może zostać wyrażona za pomocą jednoznacznie wyznaczonej sumy tych ciężarków, pod warunkiem że ciężarki mogą też przyjmować wartość ujemną. Dzieje się tak w wypadku, gdy ciężarek kładziemy na tej samej szalce co worek ze złotem, na przykład:

$$35 = -1 + 9 + 27$$

co oznacza, że na jednej szalce znajduje się worek i ciężarek 1, na drugiej zaś szalce – ciężarki 9 i 27. W ten sposób możemy zważyć worek za pomocą zaledwie czterech ciężarków!

Jeśli potęgi trójki sprawdziły się tak dobrze, co z potęgami czwórki?

Ostatnia zagadka w tym rozdziale brzmi następująco: z cyfr 1, ..., 9 możemy utworzyć dwie liczby, których stosunek będzie wynosił $1/2, 1/3, \dots, 1/9$.

Przykładowo:

$$\frac{7293}{14586} = \frac{1}{2}$$

oraz

$$\frac{6381}{57429} = \frac{1}{9}$$

Czy potrafisz znaleźć wszystkie pozostałe ułamki?

10. Dostosowywanie algorytmu do problemu

Kto chce się wzbogacić za dzień, będzie wisiał za rok.

Leonardo da Vinci: *Notatki*¹

Działanie prawie każdego praktycznego algorytmu poszukiwania heurystycznego jest sterowane pewnym zestawem parametrów. W symulowanym wyżarzaniu, na przykład, mamy do czynienia z parametrem temperaturowym, a co więcej, musimy określić schemat redukcji temperatury w czasie. We wspinaniu się występuje parametr sterujący rozmiarem lokalnego otoczenia, w którym szukamy ulepszeń. W poszukiwaniu z tabu musimy określić sposób realizacji reguł korzystania z pamięci. Żaden z tych algorytmów nie jest starannie zapakowanym prezentem, który wystarczy jedynie otworzyć, aby cieszyć się przyjemną niespodzianką!

Nie inaczej jest z algorytmami ewolucyjnymi. W rzeczywistości algorytmy te mają często więcej parametrów sterujących niż inne metody. Jest to zarówno 1) źródło ich siły w stosowaniu do różnych problemów, jak i 2) źródło frustracji dla kogoś, kto musi opracowywać takie algorytmy. Tak – tą osobą jesteś Ty. Musisz rozważyć zatem wiele elementów: 1) reprezentację, 2) funkcję oceny, 3) operatory różnicowania, 4) rozmiar populacji, 5) kryterium zatrzymania algorytmu itd. Znaczny wysiłek włożono w poszukiwanie takich ustawień parametrów, które mogłyby zapewnić sensowne zachowanie tych algorytmów dla wielu różnych problemów. W pewnych wypadkach starania te faktycznie zakończyły się powodzeniem, ale na ogólny poszukiwanie magicznego zestawu wartości parametrów, który będzie dawać najlepsze możliwe wyniki, to robota głupiego.

Co więcej, użycie do poszukiwania użytecznych wartości parametrów metody prób i błędów jest bardzo żmudną i czasochłonną przygodą. Dobrze by było, gdyby istniała teoria, która wskazywałaby, jak ustawać różne parametry algorytmów ewolucyjnych. W pewnych przypadkach taka teoria rzeczywiście istnieje, ich zakres jest jednak ograniczony. Przy braku takiej teorii dobrze by

¹Za: <http://www.sentencja.net/v.html> (przyp. tłum.).

było mieć do dyspozycji jakąś automatyczną metodę optymalizacji parametrów sterujących poszukiwaniem ewolucyjnym ulepszonych rozwiązań. Jak się przekonamy, istnieje wiele takich metod, a niektóre z nich są oparte na algorytmach ewolucyjnych (czyli są taką ewolucją ewolucji).

10.1. Parametry sterujące w algorytmach ewolucyjnych

Po określeniu reprezentacji i funkcji oceny, które w rzeczywistości są pomostem między Twoim pierwotnym problemem a schematem algorytmu ewolucyjnego, w dalszym ciągu стоisz przed wieloma wyzwaniami. Jak zamierzasz przekształcić rozwiązania-rodziców w potomstwo? To znaczy, jakiej postaci operatorów różnicowania zamierzasz użyć? Jakiegokolwiek wyboru byś dokonał, bez wątpienia stanieś przed decyzją, jak sparametryzować te operatory. Jeśli na przykład uważasz, że w Twoim problemie może być użyteczna jakaś postać rekombinacji, to musisz określić, czy będzie to krzyżowanie jednopunktowe (tzn. wybieramy losowo punkt i łączymy w jedną całość dane, które znajdują się przed tym punktem u pierwszego rodzica, z danymi, które znajdują się po tym punkcie u drugiego rodzica), czy analogicznie określone krzyżowanie dwupunktowe, krzyżowanie n -punktowe, krzyżowanie jednostajne (równomierne; gdzie każdy element jest wybierany losowo zgodnie z rozkładem jednostajnym od obu rodziców), krzyżowanie arytmetyczne (które polega na mieszaniu lub uśrednianiu elementów), operator logiki większościowej (porównujemy tutaj troje lub więcej rodziców i każdy element ustawiamy na wartość, która występuje w większości wybranych rodziców) itd. Powiedzmy, że wybrałeś operator logiki większościowej. Teraz masz do czynienia z kolejnym problemem związanym z parametrami. Ilu rodziców będziesz porównywał przy podejmowaniu decyzji? Minimalna sensowna liczba to trzy, możesz tu jednak wziąć pod uwagę badanie całej populacji niezależnie od tego, jak jest duża. Powiedzmy, że przyjęłeś, iż w operatorze większościowym dla każdego potomka będzie wybieranych losowo 20 rodziców. Tu pojawia się jeszcze jeden problem z parametrem sterującym. Czy powinieneś ustawić wartości potomków na podstawie głosu większości dla każdego elementu, czy też może powinieneś ustawić każdą wartość na podstawie prawdopodobieństwa, które odpowiada ułamkowi populacji mającej daną wartość dla interesującego nas elementu? Powiedzmy więc, gdybyś pracował z reprezentacją binarną i 11 spośród rodziców miało 1 jako pierwszy element, 9 zaś miało 0, to czy ustawilibyś pierwszy element potomka na 1, gdyż taka wartość występuje u większości, czy też może powiedziałbyś, że prawdopodobieństwo ustawienia tego elementu na 1 wynosi $11/20$, a prawdopodobieństwo ustawienia go na 0 wynosi $9/20$? Niezależnie od tego, jaki operator różnicowania wybierzesz, będziesz musiał określić mnóstwo parametrów.

Każda decyzja jest tutaj istotna. Zły wybór parametrów może dać katastrofalne wyniki. Widzieliśmy już wielu ludzi rezygnujących z algorytmów ewolucyjnych właśnie z tego powodu. Wypróbowałeś coś, o czym przeczytali w jakiejś książce na ten temat, doszli do wniosku, że to „nie działa”, lub

też zasugerowali się, że powinni używać reprezentacji binarnej, podczas gdy problem ze swojej natury nie pasował do tej reprezentacji i wykonanie kodowania wymagało dużego wysiłku, stracili więc zainteresowanie tą metodą. Powinieneś zatem wiedzieć, że gdy zaczynasz korzystać z dowolnego algorytmu poszukiwania ewolucyjnego, czy jakiegoś innego, wówczas powinieneś zawsze nastawić się na sprawdzenie, jak najlepiej dostosować ten algorytm do swojego problemu. Nigdy nie powinieneś przyjmować, że tylko dlatego, iż metoda przy danym zestawie parametrów zadziałała dobrze komuś innemu w jego konkretnym zastosowaniu, to i dla tego zestawu zadziała Tobie.

Oto klasyczny przykład. Na początkowym etapie badań nad algorytmami ewolucyjnymi De Jong [93] analizował zbiór pięciu problemów NLP i próbował znaleźć odpowiednie wartości parametryczne dla rozmiaru populacji, prawdopodobieństwa rekombinacji i mutacji, strategii selekcji itd. Każdy z tych problemów dotyczył wartości ciągłych, ale De Jong zastosował do nich kodowanie binarne, a do różnicowania użył krzyżowania jednopunktowego oraz prostej mutacji polegającej na przełączaniu bitów. Po przeprowadzeniu wielu eksperymentów doszedł do wniosku, że dla tych funkcji dobre parametry to: rozmiar populacji 50, prawdopodobieństwo krzyżowania 0,6 i mutacji 0,001 oraz selekcja elitarna zachowująca zawsze najlepsze dotychczas znalezione rozwiązanie.

Dziesięć lat później Grefenstette [201] użył algorytmu ewolucyjnego do poszukiwania wartości parametrów, które dla tych samych funkcji dobrał De Jong. Efektywność pierwotnego algorytmu ewolucyjnego była mierzona dla przypadków *online* i *offline*, przy czym w tym pierwszym brano pod uwagę jakość każdego rozwiązania wygenerowanego w eksperymencie ewolucyjnym, w tym drugim zaś tylko jakość najlepszego rozwiązania w danej chwili. Poszukiwanie ewolucyjne najlepszych parametrów dla poszukiwania ewolucyjnego doprowadziło do następujących wartości (wartości offline zostały podane w nawiasach): rozmiar populacji 30 (80), prawdopodobieństwo krzyżowania 0,95 (0,45), prawdopodobieństwo mutacji 0,01 (0,01) oraz selekcja elitarna (nieelitarna).

Oto kilka interesujących rzeczy, jakie wynikły z tych badań. Po pierwsze, większość publikacji, na temat algorytmów ewolucyjnych, z lat osiemdziesiątych i początku dziewięćdziesiątych XX wieku podawała wartości bardzo zbliżone do wartości podanych w którymś z tych badań. Publikacje te miały oczywiście duży wpływ na inne zastosowania. Ale czy było to właściwe? Większość tych innych zastosowań nie miała nic wspólnego z tymi pięcioma funkcjami badanymi we wspomnianych pracach. Po drugie, pięć lat później Davis [89] pokazał, że prosty algorytm wspinania się, który korzystał z inicjowania losowymi wartościami dla tych zadań, przewyższał algorytmy ewolucyjne:

Podstawowe założenie naszej dziedziny polegało na tym, że algorytmy genetyczne działają dobrze, gdyż łączą schematy – wieloczłonowe składniki dobrych rozwiązań. Mimo to jest algorytm [wspinania się z wykorzystaniem losowego bitu], który działa na jednym biecie na raz i znajduje rozwiązania od 3 do 23 razy szybciej niż algorytm genetyczny, dla tych właśnie problemów, którymi posłużyliśmy się, aby dostosować i porównać algorytmy genetyczne.

Nie ma wątpliwości, że wiele początkowych prób związanych z obliczeniami ewolucyjnymi, a nawet niektóre z ostatnio podejmowanych, było zbyt wąsko zakrojonych i dopuszczało tylko jeden konkretny sposób zakodowania problemów, co powodowało, że do określenia pozostawała tylko kwestia znalezienia najlepszych parametrów dla dostępnych tutaj operatorów. Teraz już wiemy, że przy każdym problemie trzeba dopuszczać możliwość dostosowywania dostępnych parametrów, a nawet współzależności operatorów różnicowania, operatora selekcji i reprezentacji. Zakres „optymalnych” ustawień parametrów jest z konieczności bardzo wąski. Każde poszukiwanie ogólnego (bliskiego) optymalnego zestawu parametrów jest skazane na porażkę już na początku.

Jedną z możliwości ręcznego dostosowania parametrów lub dostosowania na podstawie wielu prób użycia algorytmu ewolucyjnego jest skorzystanie z metod analizy matematycznej. Takie podejście daje dobre wyniki w pewnych ograniczonych dziedzinach. Na przykład, w pierwszych latach istnienia algorytmów ewolucyjnych Rechenberg [383] wykazał, że jeśli korzystamy z populacji złożonej z jednego rodzica, który generuje tylko jednego potomka, i jeśli używamy reprezentacji ciągłej (tzn. opartej na liczbach rzeczywistych) i jeśli próbujemy znaleźć minimum silnie wypukłej (np. kwadratowej) funkcji lub funkcji typu „płaszczyznowy korytarz” (ang. *planar corridor*), i jeśli wymiar tych funkcji jest duży (tzn. $n \rightarrow \infty$), i jeśli korzystamy z mutacji Gaussa o średniej zero i dostosowanym odchyleniu standardowym, to dla tych dwóch rodzajów funkcji najlepsze wartości odchylenia standardowego dają ulepszone rozwiązania z prawdopodobieństwem odpowiednio 0,27 i 0,18. Rechenberg zaproponował jako kompromis tak zwaną regułę jednej piątej sukcesów: jeśli częstość generowania rozwiązań lepszych od rodzica jest większa niż jedna piąta, to należy zwiększyć odchylenie standardowe; jeśli jest ona mniejsza niż jedna piąta, to należy je zmniejszyć. Chodzi tutaj o to, że jeśli generujemy za dużo lepszych potomków, to jesteśmy zbyt zachowawczy i wykonujemy za małe kroki. Jeśli jednak częstość ulepszania spada poniżej jednej piątej, to jesteśmy zbyt agresywni i powinniśmy zredukować rozmiar kroku. Ta analiza wskazuje na pewną procedurę online dostosowującą odchylenie standardowe w celu otrzymania maksymalnej spodziewanej szybkości polepszania rozwiązań.

Reguła ta, ściśle rzecz biorąc, działa tylko dla funkcji przebadanych przez Rechenberga, a i to tylko w granicy przy rosnącej liczbie wymiarów. Czy regułę jednej piątej sukcesów możemy zastosować do problemu komiwojażera? Fogel i Ghozeil w pracy [157] pokazali, że wyniki jej zastosowania są tutaj bardzo kiepskie, gdyż w niektórych sytuacjach maksymalne spodziewane ulepszenie pojawiało się dla prawdopodobieństwa ulepszenia równego 0,008, a wartość ta nie jest wcale taka bliska jednej piątej. Niemniej jednak pomysł, żeby przekazywać do algorytmu ewolucyjnego informacje na temat jego postępów i wykorzystywać je do poprawiania strategii jest kuszący i przydatny.

Dla kontrastu ręczne dostosowywanie parametrów to droga przez mękę. Zwykle jest dostrajany tylko jeden parametr na raz, co może spowodować, że uzyskamy nie najlepsze rozwiązanie, gdyż parametry oddziałują na siebie w skomplikowany sposób. Z kolei równolegle dostosowywanie wielu parametrów prowadzi do konieczności przeprowadzenia ogromnej liczby eksperymentów.

tów. Techniczne mankamenty dostosowywania parametrów na podstawie eksperymentów są następujące:

- parametry nie są niezależne, ale wypróbowanie wszystkich możliwych kombinacji jest praktycznie niemożliwe;
- proces dostosowywania parametrów jest czasochłonny, nawet jeśli parametry są optymalizowane po jednym na raz;
- wyniki są często niezadowalające, nawet jeśli poświęcono na ich dopracowanie bardzo wiele czasu.

Ręczne dostosowywanie parametrów i systematyczne wykonywanie eksperymentów to prawdziwy szczyt wyzwań – może się nam udać osiągnąć go lub możemy przy tym polec.

Inną możliwością wybrania odpowiednich ustawień parametrów dla algorytmu ewolucyjnego jest oparcie się na dostrzegalnej analogii między naszym problemem i podejściem a problemem kogoś innego i jego podejściem. Przy odrobinie szczęścia coś, co zadziałało dla niego, zadziała i dla nas. Nie jest jednak jasne, jak mierzyć podobieństwo między problemami, tak aby mieć przekonanie, że analogia między dwoma problemami lub metodami ich rozwiązania rzeczywiście zachodzi. Nie wystarczy wiedzieć, że obydwa podejścia są, na przykład, oparte na reprezentacji o tej samej liczebności. To, że w dwu podejściach korzysta się, powiedzmy, z reprezentacji w postaci ciągów binarnych, nic nie daje, jeśli chodzi o ustalenie parametrów.

Jeszcze jedną możliwością jest oparcie się na teorii wskazującej, jak ustawać parametry. Jednak nasz opis tej możliwości szybko się skończy, gdyż oprócz wyniku Rechenberga nie ma tutaj zbyt dużo wyników teoretycznych. W dalszej części tego rozdziału omówimy pewną pracę, która określiła optymalne prawdopodobieństwo mutacji dla problemów liniowo separowalnych, w rzeczywistości jednak nie są to problemy, dla których chciałbyś używać algorytmów ewolucyjnych. Było trochę badań teoretycznych nad optymalnym rozmiarem populacji [466, 207, 189] oraz optymalnymi prawdopodobieństwami operatorów [191, 467, 19, 413], dotyczyły one jednak prostych problemów i ich praktyczne znaczenie jest ograniczone.

Wracając do koncepcji znajdowania wartości parametrów, które są odpowiednie dla wielu problemów niezależnie od sposobu dostosowania, musimy sobie uświadomić, że każde wywołanie algorytmu ewolucyjnego to uruchomienie już ze swojej natury dynamicznego i adaptacyjnego procesu. Korzystanie ze stałych wartości parametrów nie jest więc zgodne z duchem tego rodzaju rozwiązań. Jest oczywiste, że w praktyce na różnych etapach procesu ewolucyjnego optymalne mogą być różne wartości parametrów [88, 461, 16, 17, 19, 215, 442]. Uświadomimy sobie to wyraźnie, gdy zauważymy, że populacja co pokolenie zmienia swój skład. To, co działa dla jednego pokolenia i jednego składu populacji, nie musi działać dla wszystkich innych pokoleń i układów. Może być tak, że w pewnym zastosowaniu algorytmów ewolucyjnych duże kroki mutacji są dobre w początkowych pokoleniach, gdyż takie rozwiązanie wspomaga badanie przestrzeni poszukiwań. Mniejsze mutacje mogą zaś przydawać się później przy

dostosowaniu wyniku. (Zauważ, że ta zasada przypomina schładzanie w symulowanym wyżarzaniu). Już samo zastosowanie parametrów statycznych może prowadzić do gorszej jakości działania metody.

Prosty sposób poradzenia sobie z tym problemem polega na użyciu parametru, który zmienia się w czasie, czyli na zastąpieniu parametru p funkcją $p(t)$, przy czym t oznacza licznik pokoleń, ale tak jak znalezienie optymalnych parametrów *statycznych* dla konkretnego zadania może być dość trudne, opracowanie optymalnej funkcji $p(t)$ może być jeszcze trudniejsze. Kolejna możliwa wada tego podejścia pojawia się, gdy wartość parametru $p(t)$ zmienia się wyłącznie jako funkcja czasu t i nie bierze się pod uwagę rzeczywistego postępu w rozwiązywaniu. Występuje wtedy rozdźwięk między algorytmem a problemem. Mimo to wielu naukowców (zob. podrozdział 10.4) ulepszyło swoje algorytmy ewolucyjne (tzn. poprawiło jakość uzyskiwanych przy ich użyciu wyników dla konkretnych problemów) za pomocą prostych deterministycznych reguł modyfikujących parametry. Wybranie nieoptymalnego $p(t)$ może często prowadzić do lepszych wyników niż nieoptymalny wybór p .

Musimy przyznać tutaj, że szukanie dobrych wartości parametrów algorytmów ewolucyjnych jest słabo ustrukturalizowanym, źle zdefiniowanym i skomplikowanym problemem. Tego rodzaju problemy są jednak tymi, do których algorytmy ewolucyjne doskonale pasują! Naturalny wydaje się pomysł, żeby użyć algorytmu ewolucyjnego nie tylko do znajdowania rozwiązań problemu, ale także do dostosowania tego samego algorytmu do konkretnego problemu. Formalnie rzecz biorąc, sprowadza się to do modyfikacji wartości parametrów w czasie działania algorytmu przy jednoczesnym braniu pod uwagę samego procesu poszukiwania. Istnieją w zasadzie dwa sposoby na osiągnięcie tego. Można albo użyć jakiejś reguły heurystycznej, która zbiera informacje o bieżącym stanie poszukiwania i odpowiednio modyfikuje wartości parametrów, albo można włączyć parametry do struktury danych reprezentującej rozwiązanie i w ten sposób uczynić je przedmiotem ewolucji, jakiej podlega samo rozwiązanie.

10.2. Objaśnienie zagadnienia za pomocą NLP

Załóżmy, że mamy do czynienia z problemem optymalizacji numerycznej:

$$\text{zoptymalizuj } f(\mathbf{x}) = f(x_1, \dots, x_n)$$

przy pewnych ograniczeniach w postaci równań i nierówności

$$g_i(\mathbf{x}) \leq 0 \quad (i = 1, \dots, q) \quad \text{oraz} \quad h_j(\mathbf{x}) = 0 \quad (j = q + 1, \dots, m)$$

i zakresach $l_i \leq x_i \leq u_i$ dla $1 \leq i \leq n$, określających dziedziny dla poszczególnych zmiennych.

Dla tak postawionego problemu optymalizacji numerycznej możemy rozważyć algorytm ewolucyjny oparty na reprezentacji zmiennopozycyjnej. Każdy osobnik \mathbf{x} w populacji jest reprezentowany jako wektor liczb zmiennopozycyjnych

$$\mathbf{x} = \langle x_1, \dots, x_n \rangle$$

Założymy, że do wytworzenia potomstwa w następnym pokoleniu używamy mutacji Gaussa. Przypomnijmy, że operator mutacji Gaussa wymaga dwóch parametrów: średniej, której wartość ustawiamy tutaj na zero, dzięki czemu średnio potomstwo nie różni się od rodziców, oraz odchylenia standaryzowanego σ , które możemy interpretować jako wielkość kroku mutacji. Mutacje są wtedy wprowadzane przez zastępowanie elementów wektora \mathbf{x} wartościami

$$x'_i = x_i + N(0, \sigma)$$

przy czym $N(0, \sigma)$ jest liczbą losową Gaussa o średniej zero i odchyleniu standardowym σ . Najprostszy sposób określenia mechanizmu mutacji polega na ustaleniu tego samego σ dla wszystkich wektorów w populacji, dla wszystkich zmiennych w każdym wektorze i na czas całego procesu ewolucyjnego, na przykład $x'_i = x_i + N(0, 1)$. Intuicja podpowiada jednak, że możemy zyskać, zmieniając rozmiar kroku mutacji¹.

Po pierwsze, możemy zastąpić parametr statyczny σ parametrem dynamicznym, tzn. funkcją $\sigma(t)$. Funkcja ta może być zdefiniowana w postaci jakiejś heurystyki, która przypisuje różne wartości w zależności od liczby pokoleń. Możemy na przykład zdefiniować rozmiar kroku mutacji jako

$$\sigma(t) = 1 - 0,9 \cdot \frac{t}{T}$$

przy czym t jest numerem bieżącego pokolenia, zmieniającym się od 0 do T , będącego maksymalnym numerem pokolenia. Rozmiar kroku mutacji $\sigma(t)$, używanego dla każdego wektora w populacji i dla każdego elementu każdego wektora, będzie się tutaj powoli zmniejszał od 1 na początku przebiegu algorytmu ($t = 0$) do 0,1, w miarę jak liczba pokoleń t będzie zbliżać się do T . Może to pomóc w dostosowaniu rozwiązań w miarę postępów algorytmu ewolucyjnego. W tym podejściu wartość interesującego nas parametru zmienia się zgodnie z w pełni deterministycznym schematem, a Ty masz pełną kontrolę nad parametrem i jego wartością w danej chwili t , gdyż jest ona w pełni określona i przewidywalna.

Po drugie, możliwe jest włączenie informacji zwrotnych pochodzących z poszukiwania przy zachowaniu tej samej wartości σ dla każdego wektora w populacji i dla każdego elementu wektora. Przykładem jest tutaj reguła jednej piątej Rechenberga. Implementacja online tej reguły jest następująca:

¹Istnieją formalne argumenty potwierdzające w kilku konkretnych sytuacjach tę myśl, na przykład [16, 17, 19, 215].

```

if ( $t \bmod n = 0$ ) then
     $\sigma(t) \leftarrow \begin{cases} \sigma(t-n)/c, & \text{jeśli } p_s > 1/5 \\ \sigma(t-n) \cdot c, & \text{jeśli } p_s < 1/5 \\ \sigma(t-n), & \text{jeśli } p_s = 1/5 \end{cases}$ 
else
     $\sigma(t) \leftarrow \sigma(t-1)$ 

```

przy czym p_s oznacza częstość względną mutacji zakończonych sukcesem, mierzoną w ciągu pewnej liczby pokoleń, $0,817 \leq c \leq 1$ [21]. (Schwefel w [418] wskazał wartość $c = 0,82$). Zmiany wartości parametrów są teraz oparte na informacji pochodzącej z poszukiwania, a adaptacja σ zachodzi co n pokoleń. Twój wpływ na wartość parametru jest tutaj znacznie mniejszy niż w poprzednim, deterministycznym schemacie. Mechanizm wprowadzający związek między procesem poszukiwań a wartościami parametrów jest w dalszym ciągu opracowaną przez człowieka heurystyką dyktującą, jak powinny odbywać się zmiany, ale wartości $\sigma(t)$ nie są przewidywalne.

Po trzecie, możliwe jest przypisanie do każdego wektora (tzn. każdego rozwiązania) jego własnego rozmiaru kroku mutacji. Możemy tutaj reprezentację osobnika rozszerzyć do długości $n + 1$:

$$\langle x_1, \dots, x_n, \sigma \rangle$$

Wartość σ określa, w jaki sposób wartości x_i będą mutowane, i sama też jest przedmiotem różnicowania. Typowe różnicowanie może tutaj mieć postać

$$\begin{aligned}\sigma' &= \sigma \cdot e^{N(0, \tau_0)} \\ x'_i &= x_i + N(0, \sigma')\end{aligned}$$

przy czym τ_0 jest parametrem metody, który często jest ustawiany na $1/\sqrt{n}$. Mechanizm ten jest zwykle nazywany mechanizmem *samoadaptujących się* rozmiarów kroków mutacji.

W powyższym schemacie wartość σ jest stosowana do jednego osobnika. Każdy osobnik ma własny przystosowalny parametr związany z rozmiarem kroku. Osobnik taki w miarę postępów w poszukiwaniu uczy się, jak przeszukiwać przestrzeń potencjalnych rozwiązań. Podejście to możemy łatwo rozszerzyć tak, żeby uczenie odbywało się dla każdego elementu każdego wektora rozwiązań niezależnie. Każde x_i może być zmieniane zgodnie z własnym rozmiarem kroku mutacji. Jeżeli osobnik jest reprezentowany jako

$$\langle x_1, \dots, x_n, \sigma_1, \dots, \sigma_n \rangle$$

to mutacje możemy przeprowadzać przez zastępowanie powyższego wektora wartościami pochodząymi ze wzorów podobnych do wcześniej już omawianych:

$$\begin{aligned}\sigma'_i &= \sigma_i \cdot e^{N(0, \tau_0)} \\ x'_i &= x_i + N(0, \sigma'_i)\end{aligned}$$

przy czym τ_0 jest znowu parametrem metody. Każdy element x_i ma teraz własny samoadaptujący się rozmiar kroku mutacji σ_i . Mechanizm ten ozna-

ca większy stopień swobody przy adaptacji strategii poszukiwań do topologii krajobrazu funkcji przystosowania¹.

Pokazaliśmy już, jak możesz kontrolować (adaptować) operator mutacji w czasie trwania procesu ewolucyjnego. Istnieje oczywiście wiele elementów i parametrów, które także mogą być zmieniane i dostosowywane w celu uzyskania optymalnego działania algorytmu. Na ogół trzy możliwości powyżej naszkicowane dla operatora mutacji działają dla dowolnego parametru algorytmu ewolucyjnego – dla rozmiaru populacji, kroku mutacji, współczynnika kary, nacisku selekcyjnego. W niektórych sytuacjach możemy chcieć podejście to wprowadzić w sposób hierarchiczny. Na przykład przy adaptacji rozmiaru populacji możemy chcieć, aby współzawodniczyły ze sobą różne populacje – każda z własnym parametrem określającym rozmiar.

Pojawia się tutaj potrzeba stworzenia taksonomii, która mogłaby użytecznie opisać wyniki dostosowywania różnych parametrów i stosowania procedur sterujących. Rozmiar kroku mutacji, na przykład, może mieć różne obszary wpływu, które nazywamy *zakresami*. W modelu z $\langle x_1, \dots, x_n, \sigma_1, \dots, \sigma_n \rangle$ określony rozmiar kroku mutacji dotyczy tylko jednej zmiennej jednego osobnika, czyli parametr σ_i działa na poziomie poniżej osobnika. Przy reprezentacji $\langle x_1, \dots, x_n, \sigma \rangle$ zakresem σ jest jeden osobnik, podczas gdy parametr dynamiczny $\sigma(t)$ wpływał na wszystkie osobniki, a zatem w swoim zakresie miał całą populację. Zajmijmy się teraz pełniejszą taksonomią metod sterowania parametrami.

10.3. Taksonomia metod sterowania

Istnieje wiele aspektów, które możemy brać pod uwagę podczas klasyfikowania różnych metod sterowania parametrami.

1. *Co dokładnie jest zmieniane* (np. reprezentacja, funkcja oceny, operatory, proces selekcji, częstość mutacji)?
2. *Jak jest taka zmiana przeprowadzana* (np. za pomocą heurystyki deterministycznej, heurystyki opartej na informacji zwrotnej czy też w sposób samoadaptujący się)?
3. *Jaki jest zakres/poziom zmiany* (jest ona np. na poziomie populacji, na poziomie osobnika)?
4. *Jakie wielkości statystyczne lub dane mają wpływ na zmianę* (np. ocena działania operatorów, różnorodność populacji)?

Każdemu z tych aspektów warto poświęcić uwagę.

Klasyfikacja metod sterowania parametrami ze względu na to, co jest zmieniane, wymaga przede wszystkim określenia listy *wszystkich* składników

¹ Jeśli znasz dobrze statystykę i rachunek prawdopodobieństwa, to warto, żebyś zauważał, że procedura ta nie umożliwia przeprowadzania mutacji skorelowanych. Sterowanie tworzeniem potomstwa w sytuacji, gdy kroki w jednym wymiarze są skorelowane z krokami w innych wymiarach, wymaga dodatkowych mechanizmów.

algorytmu ewolucyjnego. Samo to jest już trudnym zadaniem, ale możemy tutaj skoncentrować się na następujących rzeczach:

- reprezentacji osobników,
- funkcji oceny,
- operatorach różnicowania i związanych z nimi prawdopodobieństwach,
- operatorze selekcji (lub zastępowania) i związanych z nim regułach,
- populacji opisanej w kategoriach rozmiaru, topologii itp.

Dodatkowo każdy składnik może być sparametryzowany, a liczba tych parametrów nie jest jasno określona. Na przykład potomstwo uzyskane w wyniku *krzyżowania arytmetycznego* lub brania średniej z k rodziców $\mathbf{x}_1, \dots, \mathbf{x}_k$ może być zdefiniowane za pomocą wzoru

$$\mathbf{v} = a_1 \mathbf{x}_1 + \dots + a_k \mathbf{x}_k$$

przy czym: a_1, \dots, a_k i k mogą być uważane za parametry tego operatora. Parametry populacji mogą obejmować liczbę i rozmiary populacji składowych, szybkość migracji między populacjami składowymi itp. (to w wypadku ogólnym; sytuacje, gdy mamy do czynienia z więcej niż jedną populacją, są opisane w rozdziale 16).

Mimo tych ograniczeń możemy skupić się na tym, co jest zmieniane. Umożliwia to ustalenie, na co określony mechanizm ma bezpośredni wpływ (może on pośrednio wpływać na inne aspekty algorytmu ewolucyjnego). Jest to także metoda, za pomocą której wiele osób poszukuje sposobów ulepszania własnych algorytmów, na przykład: „Chciałbym poeksperymentować ze zmianieniem częstości mutacji. Zobaczmy, jak to robili inni”¹.

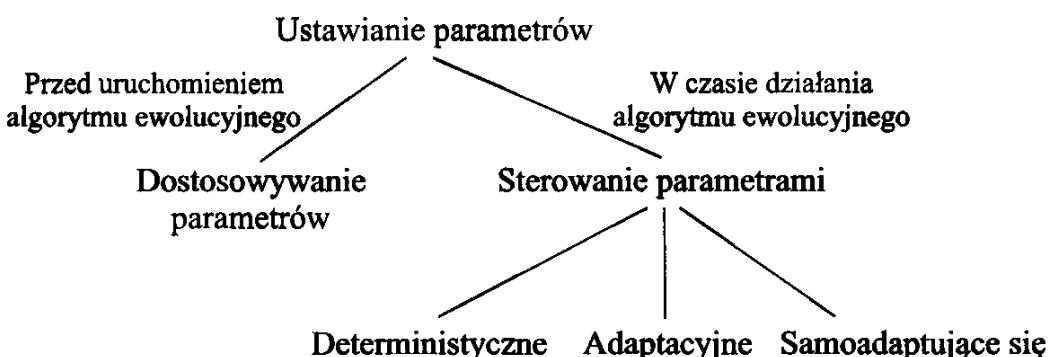
Jak już wcześniej zauważyliśmy, każda metoda zmieniania wartości parametru może być zaliczona do jednej z trzech kategorii.

- *Deterministyczne* sterowanie parametrem. Ma ono miejsce wówczas, gdy wartość parametru strategii (tzn. parametru, który steruje tym, jak poszukiwanie ewolucyjne jest wykonywane; np. takiego jak σ w poprzednim przykładzie) jest zmieniana zgodnie z pewną deterministyczną regułą. Reguła taka opisuje zmianę parametru strategii, w której nie bierze się pod uwagę informacji zwrotnej pochodzącej z poszukiwania. Często jest tutaj wykorzystywany schemat zmieniania wartości wraz z czasem – reguła jest stosowana, gdy przeminie określona liczba pokoleń od chwili poprzedniego jej użycia.

¹ Wykazałibyśmy się niedbałością, gdybyśmy nie wspomnieli, że tego rodzaju podejście z dołu do góry w rozwiązywaniu problemów jest prawie zawsze złe. Jest to trochę tak, jakbyśmy powiedzieli: „Hej, mam tu młotek, ciekawe, co inni ludzie robili za pomocą młotka?”. Gubi się tutaj zupełnie samo rozwiązywanie problemów. „Co jest naszym celem? Co chcemy osiągnąć?” Młotek to tylko środek do osiągnięcia celu, a nie cel sam w sobie.

- *Adaptacyjne* sterowanie parametrem. Ma ono miejsce wówczas, gdy istnieje jakaś informacja zwrotna pochodząca z poszukiwania, która umożliwia określenie kierunku i/lub wielkości zmiany parametru strategii. Przypisanie wartości parametru strategii może uwzględniać przypisanie nagrody, a działanie algorytmu ewolucyjnego może określać, czy nowa wartość powinna być zachowywana, czy rozprowadzana po całej populacji.
- *Samoadaptujące się* sterowanie parametrem. Ten rodzaj sterowania może być zrealizowany za pomocą ewolucji algorytmu ewolucyjnego. Parametry, które mają podlegać adaptacji, są uwzględniane w strukturach danych opisujących poszczególne osobniki i przechodzą różnicowanie (mutacje i rekombinacje). „Lepsze” wartości tych zakodowanych osobników prowadzą do kolejnych „lepszych” osobników, które z kolei z większym prawdopodobieństwem przeżywają i dają potomstwo, a co za tym idzie rozprowadzają te „lepsze” wartości parametrów.

Z tą terminologią wiąże się klasyfikacja przedstawiona na rys. 10.1. Inne taksonomie i pojęcia zaproponowano w pracy [6], my wierzymy jednak, że przedstawione tutaj podejście ma swoje zalety.



Rysunek 10.1. Globalna klasyfikacja sposobów ustawiania parametrów w algorytmach ewolucyjnych

Każda zmiana może mieć wpływ na element, całego osobnika, całą populację lub nawet funkcję oceny. Mamy tutaj do czynienia z aspektem odnoszącym się do zakresu lub poziomu adaptacji [6, 223, 439, 436]. Zauważ jednak, że zakres/poziom zależy zwykle od składnika algorytmu ewolucyjnego, w którym zachodzą zmiany, np. zmiana rozmiaru kroku mutacji może mieć wpływ na element, osobnika lub na całą populację w zależności od tego, jaką konkretne implementacja algorytmu została użyta. Z kolei zmiana współczynników kary za naruszenie ograniczeń zawsze dotyczy całej populacji. A zatem zakres/poziom jest zwykle drugorzędną cechą zależną zazwyczaj od tego, jak konkretny składnik algorytmu jest zrealizowany.

Kwestia dotycząca zakresu parametru jest tak naprawdę bardziej skomplikowana, niż to przedstawiono w podrozdziale 10.2. Przede wszystkim zakres

zależy od tego, jak dane parametry są interpretowane. I tak na przykład osobnika można reprezentować jako

$$\langle x_1, \dots, x_n, \sigma_1, \dots, \sigma_n, \alpha_1, \dots, \alpha_{n(n-1)/2} \rangle$$

przy czym wektor α oznacza kowariancje między zmiennymi $\sigma_1, \dots, \sigma_n$. W tym wypadku zakresem parametrów strategii z α jest cały osobnik, chociaż notacja może sugerować, że działają one na poziomie poniżej osobnika.

Następny przykład pokazuje, że ten sam parametr zakodowany w osobniku może być interpretowany na różne sposoby, które prowadzą do różnych wariantów algorytmu o różnych zakresach tego parametru. Spears [443] eksperymentował z osobnikami, które zawierały dodatkowy bit określający, czy ma być zastosowane krzyżowanie jednopunktowe czy jednostajne (bit 1 (0) oznaczał krzyżowanie jednopunktowe (jednostajne)). Rozważał on dwa rodzaje interpretacji tego bitu. Pierwsza z nich była oparta na wyborze operatora na podstawie wskazania rodziców. Jeśli bity obu rodziców były takie same, to był używany operator przez nie wskazany; w przeciwnym razie dokonywany był wybór losowy. A zatem przy takiej interpretacji parametr ten działał na poziomie osobnika. Druga interpretacja była oparta na rozkładzie bitów w całej populacji. Jeśli, na przykład, 73% populacji miało ustawiony bit 1, to prawdopodobieństwo krzyżowania jednopunktowego dla wybranego osobnika wynosiło 0,73. Przy tej interpretacji parametr ten działał na poziomie populacji. Zwróć uwagę, że te dwie interpretacje można z łatwością ze sobą połączyć. Na przykład podobnie jak w pierwszej interpretacji, jeśli bity obu rodziców były takie same, to mógłby być używany wskazany przez nie operator. Jeśli jednak były różne, to operator mógłby być wybrany zgodnie z rozkładem bitów – tak jak w drugiej interpretacji. Zakres/poziom tego parametru przy takiej interpretacji nie byłby ani poziomem osobnika, ani poziomem populacji, ale raczej byłby to oba poziomy. Przykład ten pokazuje, że pojęcie zakresu może być źle zdefiniowane i dowolnie złożone.

Inny sposób klasyfikacji konkretnego podejścia jest oparty na typie informacji lub wielkości statystycznej użytej do określenia zmiany wartości parametru [439, 436]. Najczęściej śledzoną wielkością jest postęp poszukiwań (np. efektywność operatorów różnicowania). Możemy też rozważać inne wielkości, takie jak różnorodność populacji. Informacje zgromadzone przy śledzeniu tych parametrów są następnie używane przy poprawianiu wartości parametrów. Chociaż jest to znaczące rozróżnienie, pojawia się tylko przy adaptacyjnym sterowaniu parametrami. Podobne rozróżnienie może być też wprowadzone przy deterministycznym sterowaniu parametrami, ale przy zastosowaniu licznika, który nie jest związany z postępem poszukiwania. Jedną z możliwości jest użycie liczby obliczeń funkcji przystosowania – jak wynika z powyższego opisu deterministycznego sterowania, liczba obliczeń funkcji przystosowania nie ma żadnego bezpośredniego związku z postępem poszukiwań. Istnieją jeszcze inne możliwości, na przykład zmienianie prawdopodobieństwa mutacji w zależności od liczby wykonywanych mutacji, nie będziemy jednak tutaj głębiej wnikać w te zagadnienia.

Tak więc główne kryteria umożliwiające klasyfikację metod zmieniających wartości parametrów strategii poszukiwania algorytmu podczas jego wykonywania są następujące:

1. *Co* jest zmieniane?
2. *Jak* zmiana jest przeprowadzana?

Klasyfikacja ta jest zatem dwuwymiarowa. Bierze się w niej pod uwagę rodzaj sterowania oraz składnik algorytmu ewolucyjnego, z którym jest związany parametr. *Rodzaj* i *składnik* są niezależne i obejmują typowe postacie sterowania parametrami w algorytmach ewolucyjnych. *Rodzaj* zmiany parametru określają trzy kategorie: zmiana może być deterministyczna, adaptacyjna i samoadaptująca się. *Składnik*, z którym jest związana zmiana parametrów, może należeć do pięciu kategorii – a są to: reprezentacja, funkcja oceny, operatory różnicowania (mutacja i rekombinacja), selekcja (zastępowanie) i populacja.

10.4. Możliwości sterowania parametrami

W tym podrozdziale przedstawimy przegląd wielu eksperymentalnych wysiłków związanych ze sterowaniem parametrami algorytmów ewolucyjnych. Opis jest podzielony w zależności od tego, co podlega adaptacji. W wyniku tego poniższe punkty odpowiadają wspomnianej wcześniej liście pięciu składników algorytmu ewolucyjnego, przy czym omówienie dotyczące sterowania operatorami różnicowania jest jeszcze podzielone na dwie części, poświęcone oddzielnie mutacji i rekombinacji.

10.4.1. Reprezentacja

Większość prób mających na celu dostosowywanie reprezentacji przeprowadzono w związku z binarnymi strukturami danych. Stało się tak prawdopodobnie dla tego, że historycznie podczas prób kodowania różnych problemów w postaci ciągów binarnych napotkano wiele trudności. Często populacje zbyt szybko dawały nieoptymalny wynik lub też pojawiały się tak zwane *bariery Hamminga*, przy których wartości zbliżone funkcjonalnie miały bardzo niewiele wspólnych elementów (np. liczby 7 i 8 są reprezentowane jako 0111 i 1000, a zatem chociaż jako liczby różnią się tylko o 1, to ich odpowiedniki binarne różnią się na wszystkich pozycjach). Warto jednak pamiętać, że adaptacja reprezentacji nie ogranicza się tylko do reprezentacji binarnej. Przedstawimy tutaj trzy główne sposoby wprowadzania reprezentacji adaptacyjnej i we wszystkich tych sposobach będziemy korzystali z adaptacyjnego sterowania parametrami.

Shaefer w [431] zaproponował strategię wykorzystującą elastyczne odwzorowywanie zmiennych funkcji na elementy osobnika (po jednym elemencie na zmienną). Jego mechanizm umożliwiał nie tylko zmianę liczby bitów przeznaczonych na poszczególne parametry, dzięki czemu było możliwe dostosowywanie rozdrobnienia, z jaką pracował algorytm, ale także zmianę zakresu każdego

z parametrów przez skracanie lub rozciąganie oraz przez przesuwanie środka wykorzystywanego przedziału. Należy pamiętać, że przy przechodzeniu z dziedziny ciągłej do binarnej, trzeba ustalić pewien poziom dokładności w obrębie wykorzystywanego przedziału (lub zestawu przedziałów), żeby możliwość rozszerzania, skracania i przesuwania środków tych przedziałów w trakcie poszukiwania była atrakcyjna¹. Reprezentacja była zmieniana w locie na podstawie reguł rządzących stopniem zbieżności elementów w całej populacji, wariancją tych elementów oraz tym, jak blisko granic zakresów znajdowały się wartości elementów dla każdej zmiennej.

Mathias i Whitley [492, 301] zaproponowali metodę, w której również zmienia się reprezentację, ale dokonuje się tego przy wielokrotnym uruchamianiu algorytmu. Pierwsza próba jest wykorzystywana do lokalizacji *rozwiązania tymczasowego*, a następne próby służą do interpretacji elementów jako odległości (tzw. *delt*) od ostatniego rozwiązania tymczasowego. Każde uruchomienie algorytmu tworzy nową hiperkostkę z rozwiązaniem tymczasowym w jej centrum. Rozdzielnosc *delt* też może być zmieniana przy każdym uruchomieniu algorytmu, co umożliwia rozszerzanie lub zawężanie przestrzeni przeszukiwania. Algorytm jest ponownie uruchamiany, gdy odległość Hamminga między najlepszym a najgorszym osobnikiem populacji wynosi jeden lub mniej.

Schraudolph i Belew [417] zaproponowali metodę, w której zmienia się interpretację bitów w ciągu. Algorytm zaczyna od ustalonych przedziałów wartości dla każdej zmiennej występującej w funkcji oceny. W miarę jak populacja staje się zbieżna w przedziale wartości jakieś zmiennej, przypisany na początku zakres wartości dla tej zmiennej jest zmniejszany, dzięki czemu algorytm ewolucyjny powiększa wybrany fragment przedziału, zachowując tę samą długość bitowego kodowania wartości. Nie zaproponowano jednak żadnej metody zmniejszania powiększenia.

Wśród innych metod, wartych wymienienia, znajduje się strategia opisana w pracy [190], w której struktura danych ma zmienną długość i może zawierać za mało lub za dużo bitów służących do zdefiniowania osobnika (tzn. może on być zbyt ściśle określony w wypadku niektórych zmiennych i zbyt luźno w wypadku innych, przy czym w ostatniej sytuacji braki są uzupełniane za pomocą dodatkowych reguł). Interpretacja osobnika zmienia się zatem, gdyż dla zmiennych, które są określone zbyt dokładnie, pierwsza pobrana specyfikacja będzie specyfikacją, na której będą wykonywane operacje. W miarę jak struktury danych przechodzą różnicowanie i selekcję, interpretacja każdego elementu rozwiązania może być włączona lub wyłączona w zależności od innych elementów tego rozwiązania. Oprócz tego niektóre z początkowych zastosowań reprezentacji adaptacyjnych dotyczyły samoadaptującego się sterowania mechanizmu dominacji w strukturze kodującej symulowane diploidy. W tym bardziej skomplikowanym układzie każda struktura we wszystkich osobnikach występuje podwójnie. Dodatkowa kopia koduje alternatywne wartości, a reguły dominacji

¹Zwrót uwagę na to, że jest to atrakcyjne tylko wówczas, gdy odwzorowujemy problem z dziedziny ciągłej do dziedziny o znacząco mniejszej liczności.

określają, które z rozwiązań będą realizowane. Dla tego typu układów istotną pracę wykonano w latach sześćdziesiątych i siedemdziesiątych XX wieku [28, 395, 229, 59], świezsze wyniki można znaleźć w [193, 196, 197].

10.4.2. Funkcja oceny

Włączenie funkcji kar do funkcji oceny potencjalnych rozwiązań umożliwia wykorzystanie potencjału znajdującego się w metodach adaptacji funkcji oceny w trakcie poszukiwania ewolucyjnego. W pracy [247] przedstawiono różne mechanizmy zmieniania kar zgodnie ze zdefiniowanymi z góry deterministycznymi schematami (omówiliśmy je w podrozdziale 10.2). W podobnym rozwiązaniu z pracy [312] skorzystano z następującego pomysłu. Funkcja oceny *eval* zawiera dodatkowy parametr τ :

$$\text{eval}(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \sum_j f_j^2(\mathbf{x})$$

który jest zmniejszany za każdym razem, gdy algorytm ewolucyjny jest zbieżny do jakiejś wartości (zwykle $\tau \leftarrow \tau/10$). Kopie najlepszego rozwiązania po uzyskaniu zbieżności są wykorzystywane jako początkowa populacja w następnej iteracji z nową zmniejszoną wartością τ . W ramach jednego uruchomienia systemu jest wykonywanych wiele takich cykli obliczeń. W ramach jednego cyklu funkcja oceny jest stała ($J \subseteq \{1, \dots, m\}$ jest zbiorem ograniczeń aktywnych w końcu cyklu), a nacisk związany z karą zwiększa się przez zmianę funkcji oceny, gdy algorytm ewolucyjny przechodzi z jednego cyklu do drugiego.

Eiben i Ruttakay w [118] zaproponowali metodę, która mieści się gdzieś między dostosowywaniem a adaptacyjnym sterowaniem funkcją przystosowania. Stosują oni metodę rozwiązywania problemów z ograniczeniami, która zmienia funkcję oceny w zależności od efektywności uruchomienia algorytmu. Zwiększone są kary (wagi) związane z ograniczeniami naruszonymi przez najlepszego osobnika po zakończeniu cyklu i tak uzyskane nowe wagi są używane w następnym uruchomieniu procedury. W ten sposób poszukiwanie ewolucyjne jest zmuszane do położenia większego nacisku na spełnianie ograniczeń, których do tej pory nie udało się spełnić (zob. też [123, 122]).

W raporcie technicznym [36] z 1992 roku przedstawiono coś, co możemy uważać za najwcześniejszego przykład adaptacyjnych funkcji oceny w zastosowaniu do spełniania ograniczeń, gdzie kary za naruszenie ograniczeń w optymalizacyjnym problemie z ograniczeniami były dostosowywane w trakcie wykonywania algorytmu (zob. podrozdział 10.2). Od tego czasu podjęto wiele wysiłków związanych z dostosowywaniem kar przypisanych do naruszeń ograniczeń. Kara może zmieniać się w zależności od liczby naruszonych ograniczeń [435] lub też w zależności od tego, czy stwierdzono, że algorytm ewolucyjny utknął w lokalnym optimum [108, 109] (za [327]). Bez wątpienia istnieje jeszcze wiele innych możliwości.

10.4.3. Operatory mutacji i ich prawdopodobieństwa

Poczyniono bardzo wiele wysiłków, żeby znaleźć optymalne wartości częstości mutacji oraz parametrów dla rozkładów prawdopodobieństwa mutacji. Skupimy się najpierw na dostosowanych optymalnych wartościach, a następnie przejdziemy do metod sterowania nimi na bieżąco.

De Jong [93] i Grefenstette [201], jak wcześniej powiedzieliśmy, przeanalizowali pięć ciągłych problemów optymalizacyjnych w kodowaniu binarnym. Na tej podstawie określili zalecane częstości mutacji p_m , które różniły się o rząd wielkości i wynosiły odpowiednio 0,001 i 0,01. Schaffer i in. [412] również zaproponowali prawdopodobieństwo mutacji bitowej, które mieściło się w zakresie [0,005, 0,01]. Propozycje te były oparte wyłącznie na wynikach eksperymentów, a zatem możliwość ich uogólniania była dość ograniczona.

O wiele wcześniej Bremermann [58, 57] udowodnił, że optymalna częstość mutacji dla opartych na funkcjach binarnych problemów optymalizacyjnych, które są liniowo separowalne, wynosi $1/L$, przy czym L jest liczbą zmiennych binarnych. Daje to maksymalną oczekiwana szybkość zbieżności do najlepszego rozwiązania. Co więcej, Bremermann w [57] wskazał, że częstość mutacji powinna zależeć od liczby zmiennych binarnych, które zostały poprawnie ustalone, zwracając w ten sposób po raz pierwszy uwagę na to, że częstość mutacji nie powinna być stała. Podobny wynik uzyskał później Mühlenbein [328]. Smith i Fogarty porównali tę częstość z kilkoma innymi stałymi częściami i potwierdzili, że była to najlepsza możliwa wartość stała dla p_m [437]. Bäck w [21] stwierdził, że $1/L$ jest również dobrą wartością p_m , gdy zastosuje się kodowanie Graya (które jest innym rodzajem kodowania binarnego).

Fogarty w [141] stosował deterministyczne schematy sterujące, które w miarę upływu czasu i z uwzględnieniem elementów zmniejszały p_m (w [22] znajdują się dokładne wzory). Jak wspomnieliśmy wcześniej w innym kontekście, koncepcja ta jest podobna do zmniejszania temperatury przy symulowanym wyżarzaniu. Hesser i Männer [215] opracowali teoretycznie optymalne schematy deterministycznej modyfikacji p_m przy rozwiązywaniu problemu zliczania jedynek¹. Zaproponowali oni wzór

$$p_m(t) = \sqrt{\frac{\alpha}{\beta}} \times \frac{\exp\left(\frac{-\gamma t}{2}\right)}{\lambda\sqrt{L}}$$

przy czym α, β, γ są stałymi, λ jest rozmiarem populacji, t jest czasem – licznikiem pokoleń (por. [57]).

Bäck także zaprezentował optymalny schemat zmniejszania częstości mutacji zgodnie z funkcją zależną od odległości do optymalnego wyniku w problemie zliczania jedynek (funkcja ta nie zależy od czasu) [16]. Jej wzór to:

$$p_m(f(\mathbf{x})) \approx \frac{1}{2(f(\mathbf{x}) + 1) - L}$$

¹Chodzi tutaj o znalezienie ciągu zer i jedynek, dla którego suma bitów jest maksymalna.

Bäck i Schütz [27] ograniczyli funkcję sterującą prawdopodobieństwem mutacji tak, żeby jego zmniejszanie zaczynało się od $p_m(0) = 0,5$ i kończyło na $p_m(T) = 1/L$, jeśli wykonano T obliczeń:

$$p_m(t) = \left(2 + \frac{L-2}{T} \cdot t \right)^{-1} \quad \text{jeśli } 0 \leq t \leq T$$

Janikow i Michalewicz [237] przedstawili eksperymenty z *mutacją niejednostajną* (nierównomierną), przy czym

$$x_k^{t+1} = \begin{cases} x_k^t + \Delta(t, r(k) - x_k) & \text{jeśli losowa cyfra binarna to 0} \\ x_k^t - \Delta(t, x_k - l(k)) & \text{jeśli losowa cyfra binarna to 1} \end{cases}$$

dla $k = 1, \dots, n$. Funkcja $\Delta(t, y)$ daje w wyniku wartość z zakresu $[0, y]$, taką że prawdopodobieństwo, iż $\Delta(t, y)$ jest bliskie zera wzrasta, w miarę jak rośnie t (t oznacza numer pokolenia). Takie rozwiązanie powoduje, że operator na początku przeszukuje przestrzeń poszukiwań jednostajnie (dla małego t), a na późniejszych etapach bardzo lokalnie. Janikow i Michalewicz w [237] użyli funkcji

$$\Delta(t, y) = y \cdot r \cdot \left(1 - \frac{t}{T} \right)^b$$

przy czym: r jest liczbą losową z przedziału $[0, 1]$, T jest maksymalną wartością licznika pokoleń, b jest parametrem systemu określającym stopień niejednostajności.

W tym momencie powinieneś się zatrzymać i zastanowić nad podobieństwem powtarzającym się w tych oddzielnich badaniach. Często przydaje się mieć na początku większą różnorodność, a później skupić działanie na drobniejszych zmianach. Twierdzenia o braku darmowych obiadów [499] wskazują, że takie postępowanie niewiele daje, gdy interesują nas wszystkie problemy. Ta sytuacja uświadamia nam dwie ważne rzeczy: 1) możemy skupić się na podzbiorze wszystkich możliwych problemów i znaleźć wzory, które umożliwiają efektywne sterowanie wartościami parametrów operatorów różnicowania i 2) typowe problemy wykorzystywane do testowania i analizowania operatorów różnicowania mają wiele istotnych podobnych cech. Ta ostatnia własność może nie być taka pożądana. Być może, badania skupią się na zbyt wąskim podzbiorze wszystkich możliwych problemów i dają metody, które dla rzeczywistych problemów nie muszą być tak użyteczne, jak by się nam wydawało. Ważne jest zatem, żeby zawsze testować procedury na problemach, które są tak podobne do problemów rzeczywistych, jak jest to tylko możliwe.

Wspomnieliśmy już dwa inne środki służące do sterowania parametrami ciągłego operatora mutacji; były to: 1) reguła jednej piątej Rechenberga i 2) samoadaptująca się wartość σ w mutacji Gaussa. Interpretacja σ może zostać rozszerzona tak, żeby stanowiła zmienną sterującą przy skalowaniu innych używanych przy mutacji rozkładów losowych, takich jak zmienna losowa Cauchy'ego czy też mieszanych rozkładów Cauchy'ego i Gaussa [65]. Co

więcej, uaktualnianie parametru σ nie musi być wykonywane za pomocą rozkładu logarytmiczno-normalnego, tzn. $e^N(0, \sigma)$. Fogel i in. [155] zaproponowali podejście, w którym dla każdego elementu osobnika jest używana inna zmenna losowa Gaussa do aktualizacji σ . Porównanie tych metod przedstawiono w [408, 7]. Jak mogliśmy się spodziewać, dla każdej metody istnieją sytuacje, w których ma ona przewagę, ale nie został dotychczas wynaleziony żaden wyraźny wzorzec umożliwiający określenie, kiedy którą z nich należy zastosować. Samoadaptujące się sterowanie rozmiarami kroków mutacji omówiono szeroko w pracach [21, 419]. Istnieje wiele zastosowań i wariantów samoadaptującego się poszukiwania parametrów w dziedzinach ciągłych (np. [220, 17, 16, 437]).

Samoadaptująca się mutacja była wykorzystywana również do problemów nienumerycznych. Fogel i in. [164] użyli samoadaptujących się metod do sterowania prawdopodobieństwami względnymi pięciu operatorów mutacji dla składników automatu skończonego. Hinterding w [221] użył reprezentacji z zastosowaniem wielu struktur, żeby zaimplementować samoadaptującą się metodę do problemu cięcia surowca (ang. *cutting stock problem*), gdzie metoda samoadaptacji określała prawdopodobieństwo użycia jednego z dwóch operatorów mutacji. Chellapilla i Fogel [66] użyli samoadaptacji do sterowania długością operatora inwersji w TSP. Pomysł wykorzystywania ewolucji do optymalizacji jej samej (tzn. samoadaptacja) ma powszechnie zastosowanie zarówno w problemach ciągłych, jak i dyskretnych.

10.4.4. Operatory krzyżowania i ich prawdopodobieństwa

W wypadku reprezentacji binarnych częstość krzyżowania p_c (inaczej niż częstość mutacji p_m , która jest stosowana do każdego bitu) dotyczy pary rozwiazań, podając prawdopodobieństwo, że wybrana para przejdzie rekombinację. Wysiłki De Jonga [93], Grefenstette'a [201] i Schaffera [412] wskazywały wartości p_c pochodzące z zakresu od 0,6 do 0,95. Nic dziwnego, że w tych opracowaniach, gdzie prawdopodobieństwo mutacji było zawsze bardzo małe, konieczne były duże wartości częstości rekombinacji. Przy innych schematach kodowania, które nie są oparte na ciągach binarnych, należy jednak uważać przy stosowaniu tych samych ustawań, gdyż efekt rekombinacji przy różnych reprezentacjach może być całkowicie inny.

W dalszej części tego punktu będziemy zajmować się mechanizmami sterowania prawdopodobieństwami rekombinacji i mechanizmami sterującymi samym przebiegiem rekombinacji. Zaczniemy od badań związanych ze sterowaniem prawdopodobieństwem stosowania operatora.

W pracy [91] Davis wprowadził adaptowalną częstość stosowania operatorów różnicowania (np. krzyżowania). Uczynił to przez nagradzanie tych z nich, które dawały lepsze potomstwo. Jest to swego rodzaju forma uczenia ze wzmacnieniem, gdzie nagroda trafia do operatorów, które zadziałyły kilka pokoleń temu, w nadziei że to właśnie one pomogły uzyskać obecne dobre wyniki. Nagrody są zmniejszane, w miarę jak przechodzimy do dalszej przeszłości i wcześniej używanych operatorów. W podobnym duchu Julstrom [249]

dostosowywał częstotliwość krzyżowania i mutacji, porównując odpowiednio ulepszenia, które uzyskano przy użyciu tych operatorów w poprzednim pokoleniu (lub dawniej). Prawdopodobieństwo krzyżowania było ustawiane na stosunek nagrody związanej z ulepszeniami wprowadzonymi przez krzyżowanie podzielonej przez liczbę wystąpień krzyżowania do tego samego ułamka zsumowanego z analogicznym ułamkiem uzyskanym dla mutacji:

$$p_c = \frac{Nagroda(Krzyż)/N(Krzyż)}{Nagroda(Krzyż)/N(Krzyż) + Nagroda(Mut)/N(Mut)}$$

przy czym *Krzyż* i *Mut* oznaczają odpowiednio krzyżowanie i mutację. Gdy krzyżowanie daje więcej sukcesów, zwiększa się prawdopodobieństwo jego użycia w przyszłości; podobnie jest też dla mutacji. Zauważ, że nie ma przekonującego rozumowania wskazującego, że prawdopodobieństwo krzyżowania powinno się uzupełniać z prawdopodobieństwem mutacji. Można tak samo dobrze prawdopodobieństwa te adaptować niezależnie. Tuson i Ross [470] przedstawili obszerne studium różnych opartych na koszcie mechanizmów adaptacji częstotliwości stosowania operatorów i stwierdzili, że ta ogólna procedura działała efektywnie dla problemów harmonogramowania, ale jednocześnie działała niezadowalająco dla wielu innych problemów. Istnieje wiele sposobów implementacji tej ogólnej koncepcji i każdy z nich będzie miał swoje zalety i ograniczenia. Inne badania oparte na tych pomysłach znajdziesz w [286, 486].

Spears w [443] zaproponował sposób samoadaptowania wyboru między dwoma rodzajami krzyżowania – krzyżowania dwupunktowego i krzyżowania jednostajnego. Dodał on jeden bit do każdego osobnika (zob. podrozdział 10.3). Ten dodatkowy bit określał, jaki rodzaj krzyżowania ma być użyty dla tego osobnika. Potomek dziedziczył swój rodzaj krzyżowania po rodzicach zgodnie z zastosowanym operatorem różnicowania. Wcześniej Fogel i Atmar [151] zastosowali adaptację krzyżowania i mutacji przez odpowiednie oznaczanie osobników, ale nie dopuszczali, żeby osobniki zmieniały sposób przechodzenia przez różnicowanie. Populacja była inicjowana losowo, jeśli chodzi o operatory różnicowania, a następnie dynamika ewolucji decydowała o tym, który operator miał przewagę, w miarę jak populacja dążyła do rozwiązania. Metodę tę zaimplementowano i zastosowano do znajdowania rozwiązań układów równań liniowych; w tym wypadku populacja była zbieżna do mutacji blisko dziesięć razy częściej niż do rekombinacji.

Schaffer i Morishima w [413] użyli metod samoadaptacji do określania liczby punktów krzyżowania. Do reprezentacji wprowadzono specjalne znaczniki, które śledziły umiejscowienie punktów krzyżowania. Eksperymenty przez nich przeprowadzone wskazywały, że samoadaptacja dawała na zestawie problemów testowych co najmniej tak dobre wyniki jak metody bez adaptacji.

Jedną z pierwszych prac na temat samoadaptacji prawdopodobieństw krzyżowania (i mutacji) była praca [384], w której osobniki reprezentowały strategie w prostej grze karcianej. Osobniki składały się z parametrów opisujących reguły gry, na przykład prawdopodobieństwo „postawienia wysoko” przy „dobréj karcie”. Miały one też parametry umożliwiające zmianę prawdopodobieństwa mutacji i wyłączenie lub wyłączenie możliwości krzyżowania z innymi

osobnikami. Ponadto w pracy tej wykorzystano koewolucję do określenia wartości każdej ze strategii, gdyż wszystkie strategie współzawodniczyły z innymi strategiami obecnymi w populacji. Warto zwrócić uwagę, że wszystko to zostało zrobione w 1967 roku¹!

Dotychczas rozważaliśmy tylko sterowanie rekombinacją dwóch rodziców. Przy rozszerzaniu rekombinacji na wielu rodziców w naturalny sposób pojawia się nowy parametr [111], a mianowicie liczba rodziców biorących w niej udział. Eiben i in. [121] użyli mechanizmu adaptacyjnego do ustalania liczby osobników w rekombinacji na podstawie zachowania współzawodniczących podpopulacji. Populację w tym badaniu podzielono na rozłączne podpopulacje, w których stosowano krzyżowanie dla różnych liczb osobników. Podpopulacje ewoluowały niezależnie przez pewien czas i wymieniały informacje przez dopuszczenie migracji po każdym z takich okresów (więcej na temat tych podpopulacji dowieś się w rozdziale 16). Migracja, zgodnie z dość naturalnym założeniem, była zorganizowana tak, żeby te populacje, które świadczyły o większym postępie w danym okresie, zwiększały się, natomiast te o małym postępie – zmniejszały. Istniał także mechanizm zabezpieczający przed zupełnym wymarciem podpopulacji (a co za tym idzie operatorów krzyżowania). Metoda ta dała algorytm wykazujący podobną efektywność jak tradycyjna wersja (jedna populacja i jedno krzyżowanie) wykorzystująca wariant, w którym zastosowano krzyżowanie sześciu rodziców. W mechanizmie tym nie udało się wyodrębnić (za pomocą zwiększenia odpowiednich podpopulacji) lepszych operatorów. Zgadza się to z wynikami opracowania Spearsa [443] dotyczącego samoadaptujących się schematów.

10.4.5. Selekcja rodziców

Rodzina tak zwanych mechanizmów selekcji Boltzmanna obejmuje metodę, która zmienia w trakcie ewolucji nacisk selekcyjny zgodnie z wstępnie zdefiniowanym „schematem schładzania” [298], bardzo podobnym do tego, co widzieliśmy przy symulowanym wyżarzaniu. Nazwa pochodzi od próbkowania Boltzmanna występującego w fizyce materii gęstej, gdzie przez przejścia między stanami jest poszukiwany minimalny poziom energetyczny. W stanie i szansa na przejście do stanu j wynosi

$$P[\text{przejście do } j] = \exp\left(\frac{E_i - E_j}{K_b \cdot T}\right)$$

przy czym: E_i , E_j są poziomami energetycznymi, K_b jest parametrem zwanym stałą Boltzmanna, a T jest temperaturą. Taka reguła przejścia jest nazywana kryterium Metropolis. W mechanizmie zaproponowanym przez de la Mazę

¹Pomysł adaptowania na bieżąco reguł związanych z operatorami podał Bledsoe w 1961 roku [50]: „Jedną z trudności przy stosowaniu takich metod w komputerach jest konieczność obrania reguł doboru partnerów. Jedną z interesujących możliwości jest pozwolenie, żeby sama reguła doboru ewoluowała, tzn. żeby w procesie selekcji była wybierana ta reguła doboru, która zapewniała najlepszą przeżywalność w populacji”.

i Tidora [96] zastosowano kryterium Metropolis do określenia oceny rozwiązania. W ten sposób kryterium selekcji zmienia się w czasie. Podobne podejście z zastosowaniem wielu populacji przedstawiono w [402].

W porównaniu z metodami adaptowania operatorów różnicowania lub reprezentacji metodom adaptacyjnego wyboru rodziców poświęcono stosunkowo mało uwagi. Jednak pewne metody selekcji mają parametry, które łatwo poddają się adaptacji. Rozważmy na przykład proporcjonalne szeregowanie, w którym każdemu osobnikowi przypisywano własne prawdopodobieństwo wyboru, które jest proporcjonalne do numeru kolejnego i na pewnej liście rankingowej¹:

$$p(i) = \frac{2 - b + 2i(b - 1)/(roz\text{m_}pop - 1)}{roz\text{m_}pop}$$

przy czym parametr b oznacza wartość oczekiwana liczby potomków, którzy mają pochodzić od najlepszego osobnika. Zmieniając wartość tego parametru w zakresie $[1, 2]$, możemy zmieniać nacisk selekcyjny w procedurze. Podobne możliwości istnieją dla innych metod rankingowych i selekcji turniejowej.

10.4.6. Populacja

Wiele wysiłku włożono w określenie odpowiedniego rozmiaru populacji w algorytmie ewolucyjnym, jednak większość analiz teoretycznych nie była bardzo użyteczna [188, 189, 441]. W rezultacie przeprowadzono empiryczne próby określenia optymalnych ustawień dla konkretnych problemów. Wspomnieliśmy już wcześnie opracowanie De Jonga [93] i późniejsze Grefenstette'a [201], w których autorzy, posługując się pięcioma problemami testowymi, szukali dobrych rozmiarów populacji. Grefenstette w zasadzie określał rozmiary populacji za pomocą algorytmu ewolucyjnego na metapozycie. W wielu eksperymentach testowano algorytmy ewolucyjne na populacjach różnych rozmiarów, żeby określić, które wartości są lepsze (np. [238, 63, 181]), nie wykonywano tam jednak adaptacji rozmiaru populacji w trakcie ewolucji.

Smith w pracy [440] przedstawił krótki opis dotyczący możliwości adaptacji rozmiaru populacji jako funkcji zależnej od zmienności jakości wzorców w poszczególnych rozwiązaniach. W zaproponowanej metodzie szacowano zmienność na podstawie pobranych z populacji różnych próbkoowych rozwiązań, a następnie na podstawie tak określonej zmienności zmieniano rozmiar populacji. Niestety krótki opis (jedna strona druku) nie pozwala szczegółowo wyjaśnić teorii ani mechanizmu, który został tu wykorzystany, ale jest dobrą podstawą przyszłych prac.

Arabas i in. [12] zaimplementowali regułę adaptacji rozmiaru populacji przez oznaczanie osobników parametrem sterującym czasem życia osobnika. Parametr określający czas życia zdefiniowano jako maksymalną liczbę pokoleń, jakie dany osobnik mógł przeżyć. Osobnik był automatycznie usuwany z populacji po przekroczeniu swego czasu życia, niezależnie od jego jakości. Czas

¹Numer najgorszego osobnika to zero, a numer najlepszego to $roz\text{m_}pop - 1$.

życia przypisywany osobnikowi przy jego tworzeniu był określany w zależności od jakości rozwiązania (tzn. lepsza jakość implikowała dłuższy czas życia). W ten sposób rozmiar populacji mógł wzrastać lub maleć w zależności od jakości osobników w poszczególnych pokoleniach.

10.5. Łączenie sposobów sterowania parametrami

W większości eksperymentów związanych ze sterowaniem parametrami algorytmów ewolucyjnych skupiano się na sterowaniu tylko jednym aspektem algorytmu na raz. Było tak prawdopodobnie dla tego, że: 1) badanie możliwości wynikających z adaptacji odbywało się eksperymentalnie i 2) łatwiej jest uzyskać pozytywne wyniki w prostszych sytuacjach. Łączenie sposobów sterowania jest o wiele trudniejsze, gdyż współzależność nawet statycznie ustawionych parametrów różnych składników algorytmu ewolucyjnego nie jest w pełni jasna. Często zależy ona od funkcji oceny [209], reprezentacji [464] i wielu innych czynników. Przeprowadzono wiele badań doświadczalnych, które miały na celu określenie związków między różnymi parametrami algorytmu ewolucyjnego [131, 412, 500]; żeby przybliżyć te wzajemne zależności, opracowano i przeanalizowano pewne modele stochastyczne oparte na łańcuchach Markowa [64, 336, 459, 482]. Mimo to wciąż brakuje wyników dotyczących łączenia metod adaptacyjnych dla algorytmów ewolucyjnych.

Przy łączeniu sposobów sterowania parametrami najczęściej są wykorzystywane metody związane z mutacją. Na przykład przy reprezentacji zmienno-pozycyjnej i stosowaniu mutacji Gaussa istnieje wiele możliwości sterowania jej działaniem. Możemy tutaj wyróżnić ustawianie odchylenia standardowego mutacji (rozmiaru kroku mutacji) na poziomie globalnym dla każdego rozwiązania lub dla elementów w rozwiązaniu. Możemy też sterować preferowanym kierunkiem mutacji, dodając rotacyjne parametry, które określają korelacje między krokami w poszczególnych kierunkach.

Inne przykłady łączenia adaptacji różnych parametrów mutacji podali Yao i in. [504] oraz Ghozeil i Fogel [183]. Yao i in. w [504] połączyli adaptację rozmiaru kroku z mieszanym mutacją Cauchy'ego i Gaussa. Przypomnijmy, że zmienna losowa Cauchy'ego jest bardzo podobna do zmiennej losowej Gaussa, tyle że ma ona znacznie grubsze ogony. W związku z tym zmienna losowa Cauchy'ego z większym prawdopodobieństwem generuje potomstwo bardziej odległe od rodzica. W podejściu tym samoadaptacji podlegał parametr skalujący rozmiar kroku, sam rozmiar kroku był używany do generowania dwóch nowych osobników na podstawie jednego rodzica: jeden był tworzony za pomocą mutacji Cauchy'ego, a drugi Gaussa. Gorszy z tych dwóch potomków był odrzucany. Wyniki empiryczne wskazywały, że metoda ta była co najmniej tak dobra, jak w przypadku użycia tylko mutacji Gaussa albo Cauchy'ego, chociaż rozmiar populacji został zmniejszony o połowę, żeby zrekompensować generowanie dwóch osobników z jednego rodzica. Ghozeil i Fogel porównali użycie współrzędnych biegunkowych do reprezentacji rozmiaru kroku mutacji i kierunku z powszechnie stosowaną reprezentacją kartezjańską. Chociaż wyniki te miały

wstępny charakter, wskazują na możliwość nie tylko sterowania wieloma aspektami różnicowania, ale także zmiany sposobu reprezentacji.

Znacznie rzadziej możemy znaleźć prace opisujące wykorzystanie wielu parametrów sterujących, rządzących różnymi składnikami algorytmu ewolucyjnego. Hinterding i in. w pracy [224] połączyli samoadaptację rozmiaru kroku mutacji z adaptacją rozmiaru populacji opartą na informacji zwrotnej. W podejściu tym grupa trzech algorytmów ewolucyjnych, z których każdy działał na populacji innego rozmiaru, była wykorzystana do korygowania rozmiaru populacji jednego lub więcej zastosowanych algorytmów. Korekcja taka była wykonywana w „epokach” co 1000 ocen. W każdym z jednostkowych algorytmów ewolucyjnych wykorzystywano samoadaptującą się mutację Gaussa. Procedura ta dochodziła do różnych strategii dla różnych rodzajów funkcji testowych. W przypadku funkcji jednomodalnych (z jednym ekstremum) dochodziła we wszystkich algorytmach do populacji o małych rozmiarach. Z kolei przy funkcjach wielomodalnych (z wieloma ekstremami) procedura doprowadzała do sytuacji, w której jeden z algorytmów używał populacji o dużym, ale oscylującym rozmiarze; miało to pomagać w uciekaniu z optimów lokalnych.

Smith i Fogharty w [438] poddali samoadaptacji rozmiar kroku mutacji oraz preferowane punkty krzyżowania. Każdy złożony element osobnika obejmował: 1) zakodowaną zmienną z wynikowego rozwiązania, 2)częstość mutacji dla tego elementu oraz 3) dwa znaczniki połączeniowe, po jednym na każdym końcu elementu, które były używane do łączenia elementów w większe *bloki*, gdy dwa sąsiednie elementy miały ustalone swoje sąsiadujące znaczniki. W podejściu tym krzyżowanie działało na wielu osobnikach i zachodziło na granicach bloków, a mutacja mogła mieć wpływ na wszystkie elementy w bloku, przy czym częstość mutacji była obliczana jako średnia wartości częstości wszystkich elementów bloku. Trudno jest opisać takie skomplikowane algorytmy w jednym akapicie, ale najważniejsza rzecz, jaką chcielibyśmy tutaj przekazać, to ta, że adaptacyjne mechanizmy sterowania różnymi parametrami można kodować w ramach jednego osobnika, a nawet w ramach jednego jego elementu.

Najbardziej wszechstronne połączenie sposobów sterowania zaproponowali państwo Lisowie [286], ponieważ połączyli oni adaptację prawdopodobieństwa mutacji, częstości krzyżowania i rozmiaru populacji. Aby oszacować efektywność działania algorytmu przy różnych ustawieniach parametrów, zastosowali statystyczne eksperymentalne projektowanie (Latin Squares) do pewnej liczby pokoleń. Uzyskane wyniki określały, jak zmieniać parametry dla następnego zbioru pokoleń. Po wyczerpaniu takiego zbioru pokoleń procedura była powtarzana.

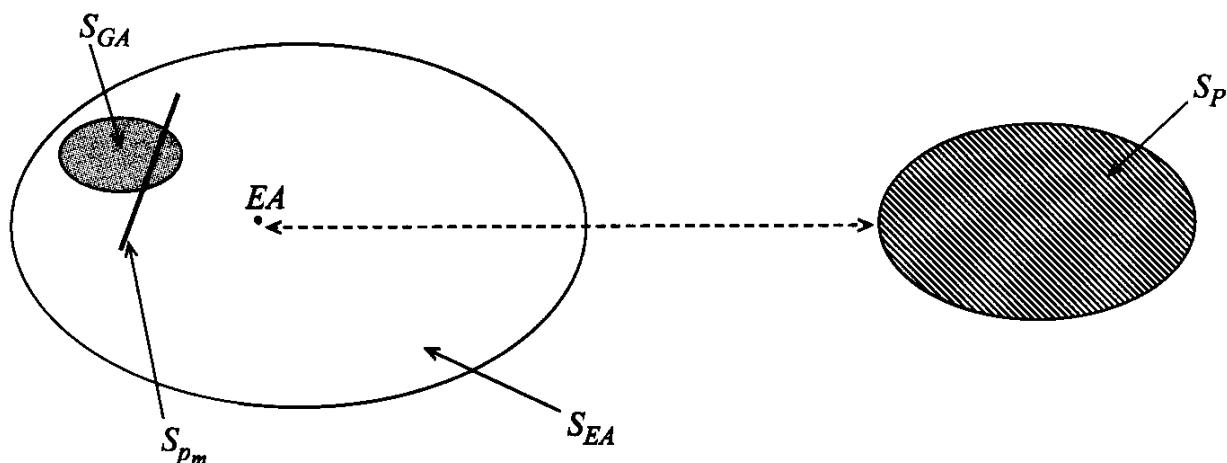
Jest interesujące, że we wszystkich, oprócz jednego, podejściach łączących różne rodzaje sterowania korzystano z samoadaptacji. Hinterding i in. w [224] do sterowania rozmiarem populacji zastosowali zamiast samoadaptacji adaptację na podstawie informacji zwrotnej. Chociaż już współzależność statycznie ustawionych parametrów różnych składników algorytmu ewolucyjnego jest skomplikowana, to współzależność dynamicznie adaptowanych parametrów za pomocą albo deterministycznej, albo opartej na informacji zwrotnej adap-

tacji jest jeszcze bardziej skomplikowana, a co za tym idzie znacznie trudniejsza do określenia. Samoadaptacja wydaje się najbardziej obiecującą metodą umożliwiającą łączenie różnych sposobów sterowania parametrami, gdyż zostawiamy samemu procesowi ewolucji określenie korzystnych związków między różnymi elementami podczas poszukiwania bliskiego optymalnego rozwiązania problemu.

10.6. Podsumowanie

Efektywność algorytmu ewolucyjnego zależy od wielu jego składników (np. reprezentacji, operatorów różnicowania) oraz ich wzajemnych zależności. Rozmaitość parametrów opisujących te składniki, wielość możliwych wyborów (np. czy zmieniać, czy nie zmieniać) oraz złożoność związków między różnymi składnikami i parametrami sprawiają, że wybór „idealnego” algorytmu ewolucyjnego jest bardzo trudny albo wręcz niemożliwy.

Jak możemy znaleźć „najlepszy” algorytm ewolucyjny dla danego problemu? Jak wynika z naszych wcześniejszych rozważań, możemy zdobyć się na pewien wysiłek i dostosować parametry algorytmu, próbując znaleźć dobre ich wartości przed uruchomieniem algorytmu. Ale nawet jeśli założymy przez chwilę, że istnieje coś takiego jak doskonała konfiguracja, to znalezienie jej jest naprawdę beznadziejnym zadaniem. Myśl tę zilustrowano na rys. 10.2.



Rysunek 10.2. Algorytm ewolucyjny EA dla problemu P jako jeden punkt w przestrzeni poszukiwań S_{EA} wszystkich możliwych algorytmów ewolucyjnych. EA przeszukuje (linia przerywana) przestrzeń rozwiązań S_P dla problemu P . S_{GA} reprezentuje podprzestrzeń klasycznych algorytmów genetycznych, a S_{p_m} reprezentuje podprzestrzeń algorytmów ewolucyjnych różniących się wyłącznie częstością mutacji p_m

Przestrzeń poszukiwań S_{EA} wszystkich możliwych algorytmów ewolucyjnych jest duża, o wiele większa od przestrzeni poszukiwań S_P dla danego problemu P , a zatem nasze szanse na *zgadnięcie* właściwej konfiguracji (znowu, jeśli taka istnieje!) dla algorytmu ewolucyjnego są skromne (np. znacznie mniejsze niż szanse na zgadnięcie optymalnej permutacji miast dla przypadku problemu komiwojażera o dużym rozmiarze). Nawet jeśli ograniczymy nasze zainteresowania do

stosunkowo wąskiej podklasy algorytmów ewolucyjnych, powiedzmy S_{GA} , która reprezentuje klasyczne algorytmy „genetyczne”, to w dalszym ciągu liczba możliwości jest ogromna¹. Zauważ, że w ramach tej (stosunkowo małej) klasy istnieje wiele możliwych algorytmów o różnych rozmiarach populacji, różnych częstościach stosowania dwóch operatorów różnicowania (czy to statycznych, czy dynamicznych) itd. Co więcej, zgadnięcie właściwych wartości parametrów może nie na wiele się przydać. Dowolny zbiór parametrów statycznych z dużą dozą prawdopodobieństwa okaże się nieodpowiedni, gdyż algorytmy ewolucyjne są ze swojej istoty dynamicznymi, adaptacyjnymi procesami. Stosowanie sztywnych parametrów, które nie zmieniają się w czasie ewolucji, jest zatem wyborem dalekim od optymalnego. Na różnych etapach procesu ewolucyjnego mogą być pożądane różne wartości parametrów.

Adaptacja daje możliwość dostosowywania algorytmu ewolucyjnego do problemu i modyfikowania parametrów konfiguracyjnych oraz parametrów strategii, które sterują przebiegiem poszukiwania ewolucyjnego w czasie trwania poszukiwania. Możemy tutaj dzięki adaptacji łatwiej wprowadzać do algorytmu ewolucyjnego informacje zależne od dziedziny oraz wiele operatorów różnicowania. Możemy także pozwolić, żeby sam algorytm wybierał te wartości i operatory, które dają lepsze wyniki. Jeśli w algorytmie ewolucyjnym dopuszczałyśmy jakiś poziom adaptowalności, to w zasadzie przeszukujemy jednocześnie dwie różne przestrzenie. Podczas rozwiązywania problemu P (przeszukiwania przestrzeni rozwiązań S_P) jest także przeszukiwana część przestrzeni S_{EA} w celu znalezienia najlepszego na danym etapie poszukiwania algorytmu ewolucyjnego EA . Dotychczas badania (zob. podrozdział 10.4) objęły tylko niewielką część przestrzeni S_{EA} . Na przykład, przy dostosowywaniu częstości mutacji p_m zajmujemy się tylko podprzestrzenią S_{p_m} (zob. rys. 10.2) składającą się ze wszystkich algorytmów ewolucyjnych, w których każdy parametr sterujący jest ustalony z wyjątkiem częstości mutacji. Pierwsze eksperymenty Grefenstette'a [201] były jeszcze bardziej ograniczone i brano w nich pod uwagę tylko powyższe algorytmy z podprzestrzeni S_{GA} .

Jedna z głównych przeszkód w optymalizacji ustawień parametrów algorytmów ewolucyjnych wynika z nieliniowego (tzw. *epistatycznego*) współdziałania tych parametrów. Wzajemne oddziaływanie różnych parametrów oraz łączny wpływ parametrów na zachowanie algorytmu ewolucyjnego są bardzo skomplikowane. Możemy tutaj wyciągnąć pesymistyczny wniosek, że takie podejście nie jest odpowiednie, gdyż zdolność algorytmów ewolucyjnych do radzenia sobie z tymi przeszkodami jest ograniczona. Ale znowu warto podkreślić, że możliwości obsługi dowolnych nieliniowych zależności między parametrami są zawsze ograniczone dla każdego algorytmu poszukiwania. Optymalizacja parametrów należy do kategorii źle zdefiniowanych, słabo ustukturalizowanych (lub co najmniej kiepsko wyjaśnionych) problemów, które nie oddają się me-

¹ Podprzestrzeń *klasycznych* algorytmów genetycznych, $S_{GA} \subset S_{EA}$, składa się z algorytmów ewolucyjnych, w których osobniki są reprezentowane przez ciągi binarne o stałej długości, i ma dwa operatory różnicowania: operator krzyżowania jednopunktowego oraz operator mutacji przez przełączenie bitu. Jest też tutaj stosowana selekcja proporcjonalna.

todom analitycznym. Algorytmy ewolucyjne są potencjalnie użyteczną metodą rozwiązywania tego rodzaju problemów. Z grubsza rzecz biorąc, na znalezienie odpowiednich ustawień parametrów możemy nie mieć lepszego sposobu niż pozwolenie, żeby algorytm ewolucyjny znalazł je sam. W tym świetle podejście związanie z samoadaptującymi się metodami stanowi najwyższy stopień uzależnienia od algorytmów ewolucyjnych. Bardziej ostrożne podejście polega tutaj na dostosowywaniu parametrów za pomocą jakiegoś rodzaju reguł heurystycznych opisujących sposób poprawiania parametrów i sprowadzających się do adaptacyjnego sterowania parametrami. Obecnie nie ma wystarczających eksperymentalnych ani teoretycznych wyników, które umożliwiłyby wyciągnięcie jakichkolwiek sensownych wniosków na temat zalet bądź wad różnych możliwych tutaj opcji.

Twierdzenie o braku darmowych obiadów [499] wskazuje teoretyczne ograniczenie samoadaptujących się algorytmów. Nie ma „najlepszej” metody samoadaptacji. Chociaż twierdzenie w oczywisty sposób może być zastosowane do samodostosowującego się algorytmu ewolucyjnego, opisuje ono jakość działania algorytmu z samoadaptacją w porównaniu z innymi algorytmami dostosowującymi parametry w zastosowaniu do wszystkich problemów. Twierdzenie to jednak w praktyce nie sprawdza się, gdyż tych „innych” algorytmów prawie nie ma. Właściwe porównanie powinno być przeprowadzane między samoadaptującymi się metodami sterowania parametrami a decyzjami podejmowanymi przez człowieka na zasadzie zgadywania, jakie wartości będą najlepsze. To ostatnie jest coraz częstszą i bolesną praktyką.

Można utrzymywać, że najlepszym sposobem projektowania algorytmu ewolucyjnego, w tym ustawiania jego parametrów, jest zdanie się na ludzkie doświadczenie i inteligencję. W końcu „inteligencja” algorytmu ewolucyjnego zawsze jest ograniczona przez niewielki, z góry określony rozmiar oglądanej przez algorytm przestrzeni poszukiwań, natomiast ludzie-projektanci mogą w całości zgłębić rozwiązywany problem. Nie oznacza to jednak, że zgłębieanie problemu prowadzi do lepszych ustawień parametrów (por. z dyskusją na temat dostosowywania parametrów i ustawiania parametrów w sposób analogiczny w podrozdziale 10.1). Co więcej, ludzka wiedza jest kosztowna i może nie być do niej dostępu w wypadku interesującego nas problemu, co sprawia, że często najpraktyczniejsze jest zdanie się na komputer. Obszar stosowalności ewolucyjnej metody rozwiązywania problemów mógłby być znacznie rozszerzony dzięki użyciu algorytmów ewolucyjnych, które mogą przynajmniej w części konfigurować się same.

Niektórzy naukowcy [37] wskazywali, że adaptacyjne sterowanie parametrami w istotny sposób komplikuje zadanie algorytmu ewolucyjnego i osiągane polepszenie jakości rozwiązań nie usprawiedliwia ich kosztu. Oczywiście z mechanizmami sterowania adaptacyjnego i samoadaptującego się jest związany pewien koszt uczenia się algorytmu – albo w trakcie działania algorytmu są zbierane pewne dane statystyczne, albo dodatkowe operacje są wykonywane na osobnikach o rozszerzonej reprezentacji. Porównywanie efektywności algorytmów z mechanizmami (samo)adaptacji i bez nich może być jednak mylące, gdyż nie uwzględnia czasu potrzebnego na dostosowanie procedury. Bardziej

sprawiedliwe porównanie musiałoby opierać się na modelu, w którym wzięto by pod uwagę czas potrzebny na konfigurację (tzn. dostosowanie) oraz sam czas działania algorytmu. W tym momencie jednak nie są nam znane żadne takie porównania.

Mechanizmy sterowania parametrami na bieżąco mogą mieć szczególne znaczenie w środowiskach niestabilnych, gdzie często musimy modyfikować aktualne rozwiązanie w związku z różnymi zmianami w środowisku (np. awariami maszyn, chorobami pracowników). Potencjalne możliwości algorytmu ewolucyjnego zapewniające obsługiwane takich zmian i efektywne śledzenie optimum były badane przez niektórych naukowców [8, 24, 477, 478]. Jak wcześniej opisywaliśmy, rozważono tutaj wiele mechanizmów. Niektórzy używali (samo)adaptacji dla różnych parametrów algorytmu, inni zaś stosowali mechanizmy oparte na zachowywaniu różnorodności populacji (niekiedy z wykorzystaniem innych schematów adaptacyjnych opartych na diploidalności i dominacji).

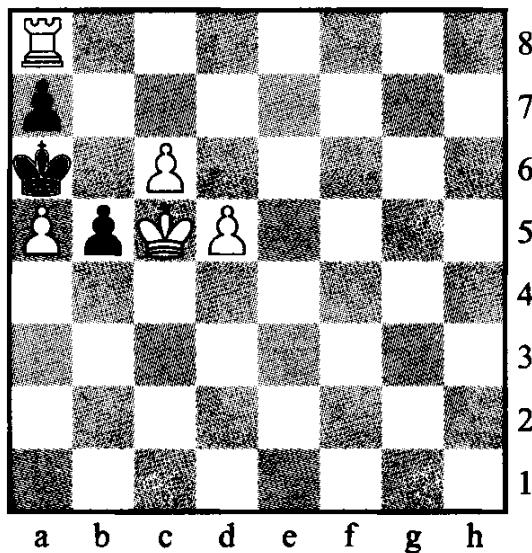
Istnieje kilka pasjonujących wątków badawczych związanych ze sterowaniem parametrami algorytmów ewolucyjnych. Oto one.

- Opracowywanie modeli umożliwiających porównywanie algorytmów z i bez mechanizmów (samo)adaptacji. Modele te powinny obejmować środowiska stabilne i dynamiczne.
- Wyjaśnianie użytkowniczości zmiennych wartości parametrów, a także zachowania algorytmu wynikającego z wzajemnego oddziaływanego parametrów, przy wykorzystaniu prostych reguł sterowania deterministycznego. Można byłoby tutaj, na przykład, rozważyć algorytm ewolucyjny ze stałym rozmiarem populacji oraz algorytm ewolucyjny z rozmiarem populacji rosnącym lub malejącym z określoną szybkością, przy czym w obu algorytmach sumaryczna liczba obliczeń funkcji oceny byłaby taka sama.
- Potwierdzanie lub dyskredytowanie popularnych heurystyk sterowania adaptacyjnego.
- Przeprowadzanie prób mających na celu znalezienie ogólnych warunków, przy których sterowanie adaptacyjne sprawdza się. Istnieją pewne wskazówki dotyczące rozmiarów kroków samoadaptujących się mutacji związane z liczbą potomstwa generowanego z jednego rodzica [419], ale nie sprawdzono, czy te lub inne wskazówki są poprawne bądź ogólnie przydatne.
- Wyjaśnianie efektów współdziałania adaptacyjnie sterowanych parametrów.
- Badanie wad i zalet związanych z samoadaptacją wielu (być może wszystkich) parametrów algorytmu ewolucyjnego.
- Opracowywanie formalnych podstaw matematycznych dla zaproponowanej taksonomii w sterowaniu parametrami algorytmów ewolucyjnych. Powinna ona być wyrażona w kategoriach funkcjonalów przekształcających operatory i wymagane przez nie zmienne.

Jest mnóstwo możliwości dostosowywania i naganiania algorytmu ewolucyjnego do potrzeb rozważanego przez nas problemu. Jeśli nie dysponujemy dostateczną wiedzą na temat tego, jak dobrze ustawić jego parametry, często użyteczne staje się zastosowanie heurystyki, która może pokierować przebiegiem procesu ewolucyjnego. Kierowanie to może być zrealizowane również jako algorytm ewolucyjny.

XI. Czy potrafisz dać mata w dwóch ruchach?

Istnieje wspaniała zagadka szachowa, która pokazuje, że problemy nie powinny być postrzegane jako statyczne. Koncepcja „czasu” może okazać się zasadnicza! Po kilku ruchach sytuacja na szachownicy przedstawia się następująco:

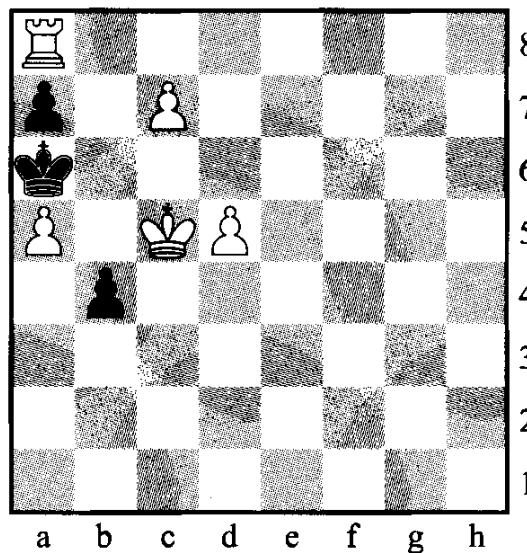


Kolej na ruch białych (poruszają się w góre). Czy możliwe jest danie mata czarnym w dwóch ruchach?

Na pierwszy rzut oka problem nie wydaje się trudny. Na szachownicy zostało tylko kilka bierek, drzewo poszukiwań nie będzie więc zbyt szerokie. Białe mają dać mata w zaledwie dwóch ruchach, a zatem drzewo poszukiwań nie może też być zbyt przepastne. A zatem spróbujmy!

Białe mają cztery możliwości wykonania pierwszego ruchu: ruch pionkiem z c6, ruch pionkiem z d5, ruch królem i ruch wieżą. Przeanalizujmy je po kolej.

- Ruch białym pionkiem c6 – c7. Ruch ten nie da mata w kolejnym posunięciu, ponieważ odpowiedzią czarnych może być ruch pionkiem b5 – b4. Uniemożliwia to danie mata królowi.

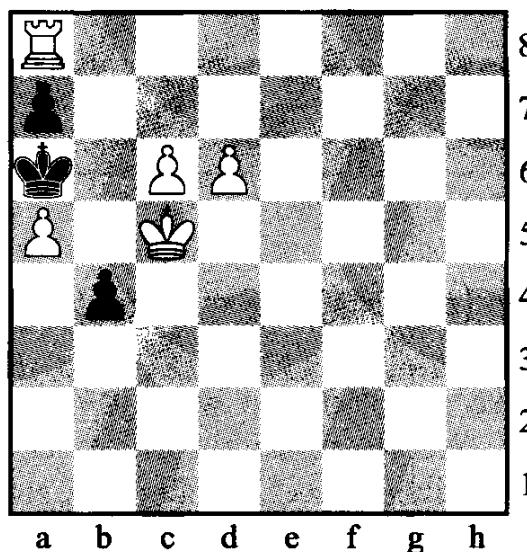


Białe mają dwie możliwości dania szacha:

- bicie wieżą a8 × a7 lub
- ruch pionkiem c7 – c8 i jego promocję na królową lub gońca.

Żaden z tych szachujących ruchów nie prowadzi do mata. W pierwszym przypadku czarny król może zbić białą wieżę, w drugim zaś – białego pionka z pola a5¹.

- Ruch białym pionkiem d5 – d6. Jest to pokojowy ruch, niewymagający żadnej obrony. Czarne mogą odpowiedzieć przesunięciem pionka b5 – b4, co dałoby następujący układ na szachownicy:



¹ Białe mogłyby wówczas dać mata czarnemu królowi w następnym ruchu (a8 × a7), ale byłby to ich trzeci ruch.

W drugim ruchu białe mogą zaszachować jedynie, bijąc wieżą ($a8 \times a7$), ale jak już przekonaliśmy się powyżej, nie doprowadzi to do mata (czarny król zbię wieżę).

- Białe mogą wykonać ruch królem. Jednak $c5 - b4$ daje pat (prowadzi do remisu), gdyż w takim wypadku czarne nie mogą wykonać żadnego prawidłowego posunięcia. Dwa inne możliwe ruchy białego króla ($c5 - d6$ i $c5 - d4$) także nie prowadzą do osiągnięcia celu: czarny król może spokojnie zbić biały pionek na $a5$. Jeśli białe zaszachują czarnego króla – można to znowu zrobić jedynie, bijąc wieżą $a8 \times a7$ – to czarne mogą odpowiedzieć ruchem $a5 - b4$ lub $a5 - b6$. Tak więc ruch białym królem nie prowadzi do rozwiązania.
- Białe mogą ruszyć wieżą. Znowu mamy tu trzy możliwości:
 - bicie $a8 \times a7$. Czarny król zbię wówczas białą wieżę. Następnie w drugim ruchu białe nie będą mogły go zaszachować;
 - ruch $a8 - b8$. Czarne mogą odpowiedzieć, ruszając pionkiem $b5 - b4$. Przy takim ustawieniu jedyną możliwością zaszachowania czarnego króla jest ruch wieżą ($b8 - b6+$). Nie doprowadzi to jednak do mata, ponieważ czarny król może ją zbić ($a7 \times b6$);
 - ruch $a8 - ?8$, co oznacza, że wieża może przejść na pole $c8, d8, e8, f8, g8$ lub $h8$. Czarne mogą wtedy wykonać ruch pionkiem $b5 - b4$ i ponownie białe nie będą miały możliwości zaszachowania króla w drugim ruchu.

Nasze analizy możemy podsumować prostym stwierdzeniem: *niemożliwe* jest danie mata czarnemu królowi tylko w dwóch ruchach!

Na czym zatem polega trik? Jedynym sposobem rozwiązania tej zagadki jest wzięcie pod uwagę czynnika czasu! Układ na szachownicy przedstawiony na początku tego rozdziału nie powinien być traktowany jak statyczne ujęcie, ale raczej jako pewien etap dynamicznie rozwijającej się gry. Kolej na ruch białych oznacza, że poprzedni ruch należał do czarnych. Stwierdzenie to jest banalne, ale jaki to był ruch? Na szachownicy są tylko trzy czarne bierki, możemy więc prześledzić wszystkie możliwości.

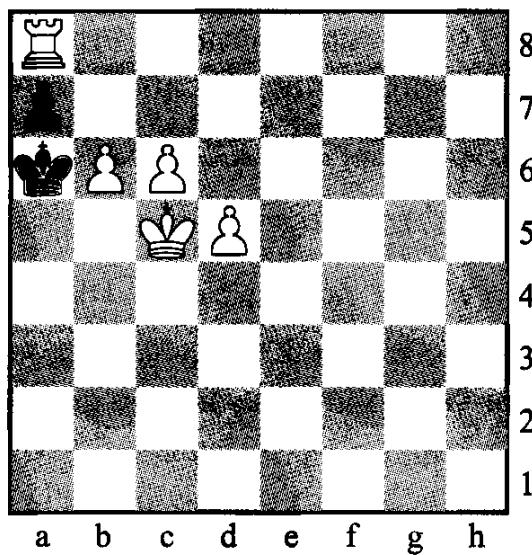
- Pionek na $a7$ jest wciąż na swojej pozycji wyjściowej, możemy więc wyeliminować możliwość jego ruchu.
- Król mógł przejść na pole $a6$ z pola $b6$ lub $b7$. Po głębszym zastanowieniu okazuje się to jednak niemożliwe. Czarny król nie mógł znajdować się na polu $b6$ podczas poprzedniego ruchu z powodu obecności białego króla na $c5$. Nie mógł też zajmować pola $b7$ z powodu obecności białego pionka na $c7$ (który nie mógł się ruszyć podczas poprzedniego ruchu).

Tak więc poprzedni ruch należał do czarnego pionka z pola $b5$. Jaki to był ruch? Albo $b7 - b5$, albo $b6 - b5$. Pierwsza opcja jest niemożliwa: pionek ten nie mógł być na polu $b6$, bo na $c5$ był biały król. Poprzedni ruch to w takim razie przesunięcie pionka $b7 - b5$.

Wiedząc o tym, możemy bez trudu rozwiązać zagadkę. Wymagany ruch białych to

a5 × b6 (*en passant*)

po którym sytuacja na szachownicy przedstawia się następująco:



Teraz czarne są zmuszone odpowiedzieć ruchem króla a6 – a5. Jest to jedynie możliwe posunięcie, po którym białe dają mata czarnemu królowi:

a8 × a7++.

Piękne, prawda?

11. Środowiska zmienne w czasie oraz szum

Zmienność to zasada życia.

Ci zaś, którzy patrzą tylko w przeszłość
lub teraźniejszość,
na pewno stracą przeszłość.

John F. Kennedy, przemówienie z 25 czerwca 1963 roku

Jak już się przekonaliśmy, rzeczywiste problemy są zwykle związane z dużymi przestrzeniami próbek i nieliniowymi, wielomodalnymi funkcjami oceny o wiele optimach. W wielu przypadkach nie istnieją tutaj matematyczne procedury w zamkniętej postaci, a jeśli nawet tego rodzaju metody istnieją, to są zbyt złożone obliczeniowo, żeby ich stosowanie było praktyczne. Ich złożoność obliczeniowa wzrasta tak szybko, że gdy próbujemy zastosować je do czegoś więcej niż trywialnych przypadków, stają się praktycznie bezużyteczne. Żeby zapewnić większą użyteczność tych metod, przyjmujemy pewne założenia upraszczające związane ze światem rzeczywistym, dopuszczając, być może, zastosowanie liniowej funkcji oceny z liniowymi ograniczeniami, wartościami całkowitoliczbowymi lub innymi powszechnie stosowanymi matematycznymi rozwiązaniami. Prawidłowość odpowiedzi poświęcamy tutaj po części po to, żeby uzyskać jakąkolwiek odpowiedź.

W rzeczywistości sprawy mają się o wiele gorzej, gdyż przy rozwiązywaniu problemów świata rzeczywistego nie wystarczy rozwiązanie problemu. Problem musi być rozwiązywany nieustannie, wciąż i wciąż, bez przerwy. Rozwiązanie problemu, z którym mamy do czynienia, staje się szybko rozwiązaniem wcześniejszego problemu. Zadowalanie się jednym rozwiązaniem dowolnego istotnego rzeczywistego problemu jest tożsame z zaakceptowaniem porażki.

11.1. Życie to dynamiczny krajobraz

Poznamy lepiej wszechogarniającą naturę zmienności problemów, gdy przyjmiemy się bliżej zadaniu przelecenia z Nowego Jorku do Los Angeles. Zanim wykonamy jakikolwiek ruch, musimy się zapytać: „Jaki jest cel?”. To znaczy, o jakie parametry musimy dbać przy ocenianiu alternatywnych rozwiązań tego

problemu? Jednym z oczywistych celów jest dotarcie do Los Angeles na czas lub może lepiej – dotarcie wcześniej. Zbyt późny przylot jest niepożądany, ale latanie to nie tylko dolatywanie na czas. Co z bezpieczeństwem? Opis tego jest łatwy – lot z pewnością powinien być bezpieczny. Lecz jakie ryzyko związane z lotem jesteśmy gotowi zaakceptować? W czasie gdy będziesz się zastanawiać nad tym parametrem, dodamy jeszcze jeden wymiar, który będzie nas interesować. Założmy, że jesteś posiadaczem samolotu, który będzie używany do przelotu. W tej sytuacji musisz się też zastanawiać nad kosztami. Na przykład nad kosztem paliwa, które zostanie spalone podczas lotu. Szybszy lot może być korzystny dla parametrów związanych z dotarciem na czas, ale będzie kosztować więcej. I oczywiście chciałbyś też, aby lot przebiegał spokojnie. Czasami, żeby to osiągnąć, trzeba zmienić wysokość przelotu, unikając w ten sposób obszaru turbulencji. Każda taka zmiana kosztuje trochę paliwa. Optymalny przelot wymaga odpowiedniego wyważenia każdego z tych parametrów.

Chociaż określenie funkcji normalizującej obsługującej różne poziomy znaczenia dla każdego ze wspomnianych powyżej parametrów jest samo w sobie trudnym wyzwaniem, założmy, że już taką funkcją dysponujesz i że opracowałeś już najlepszą możliwą trasę lotu. Startujesz. Plan lotu przewiduje, że będziesz przelatywał nad Denver w Kolorado, żeby przelecieć ponad przewidywanymi wysokimi prądami powietrznymi i w ten sposób zaoszczędzić paliwo oraz jednocześnie dolecieć na czas. W trakcie podróży, zanim doleciałeś do Denver, pojawiła się zła pogoda i musisz odstąpić od swego planu. Co teraz? Jak przeprowadzisz zmianę planu? Wydaje się, że utknąłeś. Utknąłeś, gdyż nie rozważyłeś tej możliwości i nie uwzględniłeś jej w swoim planowaniu. Najlepsze, na co można tutaj liczyć, to rozpoczęcie planowania od początku. Jak stąd, gdzie jestem (powiedzmy gdzieś nad stanem Ohio), mogę się optymalnie ze wzgledu na naszą funkcję oceny dostać do Los Angeles? Główny problem, który się tutaj pojawia, wiąże się z brakiem czasu na opracowanie najlepszego rozwiązania. Konieczne jest wykonanie zbyt dużego obliczenia. Musimy w związku z tym zastosować jakieś alternatywne rozwiązanie.

Gdy rozważyliśmy z pilotem, co robić, meteorolodzy podali, że przewidywania co do wysokich prądów powietrznych nie sprawdziły się i nie będzie też obszarów turbulencji. W odpowiedzi na tę sytuację zmuszony jesteś do zrobienia *czegoś*, w związku z czym wybierasz nową ścieżkę, która „wydaje się działać” i ruszasz nią, wiedząc, że z pewnym prawdopodobieństwem straciłeś w ten sposób kilka możliwości optymalizacji trasy.

Zbliżając się do Los Angeles, dowiadujesz się, że jest tam mgła, a pułap chmur znajduje się 66 metrów nad ziemią. Jest to bardzo bliskie dolnej granicy (60 metrów) wymaganej do lądowania zgodnie z przepisami. Czy taka pogoda utrzyma się do momentu Twojego przylotu? Jak można to ocenić? Pytasz, jaką była pogoda 20 minut temu i godzinę temu. Kierunek zmian wskazuje, że pogoda staje się coraz gorsza – mgła się obniża. Czy pozostałe powyżej minimalnego poziomu? Jak można to ustalić? Najlepsze, co można zrobić, to zgadnąć to. Konsekwencje tego zgadnięcia są bardzo istotne. Jeśli zdecydujesz się lecieć w stronę lotniska i okaże się, że chmury znajdują się poniżej minimalnego

bezpiecznego poziomu, zostaniesz zmuszony do „kręcenia się w kółko” i utraty co najmniej 30 minut na prowadzenie przez kontrolę lotów, zanim zostaniesz dopuszczony do kolejnej próby; innym rozwiązaniem jest wykonanie niezgodnego z przepisami i prawdopodobnie niebezpiecznego lądowania w warunkach poniżej standardów bezpieczeństwa. (Teraz widać, że pominęliśmy jeszcze jeden możliwy parametr: wykonanie lotu zgodnego z przepisami). Zamiast nastawiać się na lądowanie w Los Angeles, możesz uznać, że lepsze jest skierowanie się do innego pobliskiego lotniska z lepszymi warunkami pogodowymi, a następnie wypożyczenie samochodu, który dowiezie Cię do celu. Spóźnisz się nieco i będzie Cię to kosztowało więcej niż początkowo planowałeś, ale, być może, warto właśnie tak postąpić.

Całą tę sytuację moglibyśmy jeszcze „uatrakcyjnić”, wprowadzając awarię silnika, pożar na pokładzie, wadliwy instrument nawigacyjny i wiele, wiele innych okoliczności¹. Widać już w czym rzecz: rozwiązywanie problemów wymaga planowania z uwzględnianiem wszelkich ewentualności. I nie tylko tego, wymaga to uwzględniania ewentualności, które nie zostały przewidziane na czas.

Przyjrzyjmy się jeszcze jednemu przykładowi, który umożliwi nam zrozumienie wpływu tych zjawisk na granicę między zyskiem a stratą. Założmy, że prowadzisz fabrykę produkującą ubrania. Zatrudniasz tam 100 robotników, którzy obsługują dziesięć maszyn koniecznych przy szyciu pięciu różnych rodzajów ubrań. Pewna firma wykonała dla Ciebie symulację pracy fabryki, dzięki której masz do wyboru kilka schematów pracy. Zanim wykonasz pierwszy krok, musisz sobie zadać pytanie: „Jaki jest cel?”. Możliwą odpowiedzią jest wyprodukowanie jak największej liczby ubrań, prawdopodobnie jednak nie jest to pełna odpowiedź. Prawdopodobnie chcesz położyć większy nacisk na produkcję ubrań, które przynoszą Ci więcej pieniędzy. Nie chcesz, żeby ubrania zalegały w magazynach i nie były sprzedawane. Chcesz także maksymalnie obniżyć koszt wykonywanych operacji. Chcesz też zachować wysokie morale wśród robotników, co oznacza, że nie możesz im za często zmieniać planów pracy; w przeciwnym razie nie będą zadowoleni ze zlecanych zadań. Co więcej, każdy robotnik ma inne uzdolnienia, ważne jest więc, żeby zarówno na morale załogi, jak i na wydajność, zapewnić, by zadania przydzielane poszczególnym pracownikom odpowiadały ich umiejętnościom. Mają tutaj miejsce dodatkowe ograniczenia, gdyż nie każdy z robotników może pracować na każdym stanowisku przy maszynie. Możemy iść dalej? Co z zapewnieniem prawidłowego utrzymania parku maszyn? Nie można wykorzystywać ich przez 24 godziny na dobę, chyba że chcesz ryzykować niespodziewane awarie.

¹ Jeden z autorów (D.F.) jest posiadaczem licencji pilota. W trakcie ponad 500 godzin lotów, przy różnych okazjach przytrafiły mu się: 1) całkowita awaria radia pokładowego (podczas jego pierwszego samotnego lotu przez kraj), 2) otwarcie się luku bagażowego w trakcie nocnego lotu ponad zasłaniającymi ziemię chmurami, 3) krztuszenie się silnika podczas przelotu nad górami, 4) niezatrzaśnięcie się drzwi kabiny i otwarcie się ich w trakcie lotu i 5) wielokrotne sytuacje, gdy pomocne w trakcie lotu urządzenia nawigacyjne nie działały. Chociaż opis naszego fikcyjnego lotu z Nowego Jorku do Los Angeles zawiera wiele niespodziewanych sytuacji, nie są one wcale takie nietypowe.

Przypuśćmy, że udało Ci się odpowiednio wyważyć wszystkie te parametry i jakoś znalazłeś najlepszy harmonogram pracy ze względu na odpowiednią funkcję oceny. W trakcie przechodzenia do przekazywania planu pracy pracownikom dowiadujesz się, że: 1) jedna z maszyn nie działa i szacunkowy czas naprawy wynosi sześć do ośmiu godzin, ale może też potrwać „dużo dłużej” i 2) trzech ludzi zachorowało, a czterech innych spóźniło się, gdyż na autostradzie zdarzył się wypadek. Jak w związku z tym powinieneś zmodyfikować swoje plany? Pierwsza z informacji wskazuje, że musisz coś zmienić, ponieważ jedna z maszyn, które zamierzałeś użyć, nie jest dostępna. Twój plan nie pasuje do nowych ograniczeń. Druga z informacji nie wymusza zmiany planów, ze względu jednak na niedostępność jednej z maszyn, możesz wziąć pod uwagę to, że dysponujesz niewykorzystaną siłą roboczą.

Następnie dowiadujesz się, że telefonował jeden z Twoich przełożonych, by powiedzieć, że jest pilne zlecenie od ważnego klienta i że powinieneś zawiesić wszystkie prace i wykonać to zlecenie na maksymalnych obrotach. Jak to wpływa na Twój plan? Jeszcze później tego dnia jeden z dostawców dzwoni i mówi, że nie może dotrzymać terminu dostawy obejmującej dziesięć procent produkowanych ubrań. Jak zmienisz plan w świetle tej informacji? W czasie wykonywania planu niektórzy z robotników mogą zapominać dokładnego notowania zużywanych przez nich materiałów. W wyniku tego możesz uważać, że masz ich więcej, niż jest ich w rzeczywistości. Jak poprawisz plan, żeby dać sobie radę z tym problemem, gdy odkryjesz po trzech, czterech dniach, że zachodzi taka sytuacja? Bez prawidłowej strategii podejmowania decyzji sytuacja szybko może się pogorszyć, tak że każde działanie jest wykonywane tylko po to, żeby zapobiec ostatnio powstałej anomalii. Taka sytuacja pojawia się bardzo często w praktyce biznesowej i jest określana mianem „gaszenie pożarów”. Tak nie powinno się działać.

Jest zatem jasne, że rozwiązywanie problemów to więcej niż tylko znajdowanie jednego najlepszego rozwiązania. Jest to jednocześnie: 1) definiowanie problemu, 2) określenie reprezentacji możliwych kandydatów na rozwiązania, 3) przewidywanie, jak problem może zmieniać się w czasie, 4) szacowanie prawdopodobieństwa możliwych zmian oraz 5) poszukiwanie rozwiązań, które są odporne na takie zmiany. Dodatkowo dochodzą tutaj operacje ponownego rozwiązywania problemu w miarę jego zmieniania się w związku z otrzymywanyymi świeżymi informacjami. Rozwiązywanie problemu to nigdy niekończący się proces.

W związku z tym zajmiemy się teraz porównywaniem rozważanych przez nas metod heurystycznych ze względu na ich użyteczność do adaptacji rozwiązań w miarę zmieniających się okoliczności. Klasyczne metody rozwiązywania problemów planowania, opracowywania harmonogramów oraz innych problemów kombinatorycznych mają matematyczny charakter. Metody te obejmują programowanie liniowe i dynamiczne. Sprawdzają się wyraźnie słabo, gdy istotna jest adaptacja do zmieniających się warunków. Metody te obliczają tylko jedno rozwiązanie, a gdy zmieniają się warunki, znajdowanie nowego rozwiązania musi zacząć się od zera. Wykonana wcześniej praca nad rozwiązaniem problemu jest tracona.

Dla kontrastu metody, które *szukają* rozwiązania, a nie obliczają ich, często okazują się lepiej dostosowane do dawania sobie rady przy zmieniających się warunkach. Warto mieć tutaj w głowie odpowiednią analogię. Możemy sobie wyobrazić funkcję oceny jako krajobraz, będący wynikiem odwzorowywania rozwiązań na ich wartości funkcji oceny. Każde możliwe rozwiązanie odpowiada punktowi w tym krajobrazie. Gdy zmieniają się warunki związane z problemem, mamy dwie możliwości: 1) zmienia się krajobraz lub 2) zmieniają się ograniczenia określające obszar możliwych rozwiązań dopuszczalnych. Oba te elementy mogą oczywiście zmieniać się jednocześnie. Przyjrzymy się dokładniej wszystkim tym możliwościom.

Zmiana krajobrazu następuje, gdy jest modyfikowana funkcja oceny. Pamiętaj, że funkcja oceny jest właściwie subiektywnym opisem, co ma zostać osiągnięte (lub tego, czego należy unikać). Jeśli zmienia się nasze zadanie, to zmieni się też funkcja oceny. Jeśli, na przykład, względne znaczenie przylotu na czas do Los Angeles zmienia się w stosunku do kosztu paliwa, to funkcja oceny będzie musiała być poprawiona, żeby ta zmiana została uwzględniona. Wynikiem tej zmiany będzie nowy krajobraz funkcji przystosowania. Efekt ten możemy też zaobserwować, gdy zwiększa się lub zmniejsza koszt paliwa, gdyż ma to wpływ na poziom zysku związany z każdą trasą lotu. Za każdym razem, gdy zmienia się cena ropy, zmienia się też krajobraz funkcji oceny.

Ograniczenia związane z obszarem rozwiązań dopuszczalnych mogą się zmieniać jak funkcja dostępnych zasobów. W powyższym przykładzie z fabryką to, co zmienia się wraz z awarią jednej z 10 maszyn, to właśnie obszar rozwiązań dopuszczalnych. Pojawia się tutaj ostre ograniczenie wskazujące, że we wszystkich rozkładach pracy musi być używane dziewięć maszyn. Dotychczas dopuszczalne rozwiązania, które wykorzystywały wszystkie dostępne maszyny, są teraz niedopuszczalne. (Sposoby radzenia sobie z niedopuszczalnymi rozwiązaniami były rozważane w rozdziale 9).

Gdy cel lub warunki rozwiązywania zadania zmieniają się, zmienia się także krajobraz i/lub obszar rozwiązań dopuszczalnych. Jednak w niektórych przypadkach zmiana może nie być duża. Jeśli cena ropy zmienia się o 0,01 \$ na baryłce, to jest bardzo prawdopodobne, że przy takiej zmianie wartość rozwiązania z poprzedniej sytuacji będzie bardzo podobna od nowej wartości. Nasze wcześniejsze najlepsze znalezione rozwiązanie będzie w dalszym ciągu bardzo dobre i, co więcej, intuicyjnie rozwiązania dające ulepszenie będą znajdowały się w lokalnym otoczeniu dotychczasowego najlepszego rozwiązania. Oczywiście nie musi tak być. Może się okazać, że mała zmiana wysokich prądów powietrznych wymusi zupełnie inną trasę z Nowego Jorku do Los Angeles. To możliwe, lecz często w świecie rzeczywistym jest inaczej.

Nie wszystkie jednak zmiany są małe bądź nieznaczne, a gdy pojawiają się większe zmiany, podobieństwo między wcześniej znalezionymi rozwiązaniami a nowymi możliwościami może być nieznaczne. Nasze wcześniejsze rozwiązanie (lub rozwiązania) może mieć małą wartość lub może być bezwartościowe. Mogliśmy to zauważyć, obserwując przebieg naturalnej ewolucji. Historia odnotowała kilka przypadków masowej zagłady, z których najbardziej znana jest zagłada dinozaurów (zginęło wtedy ponad 60% wszystkich gatunków żyjących na zie-

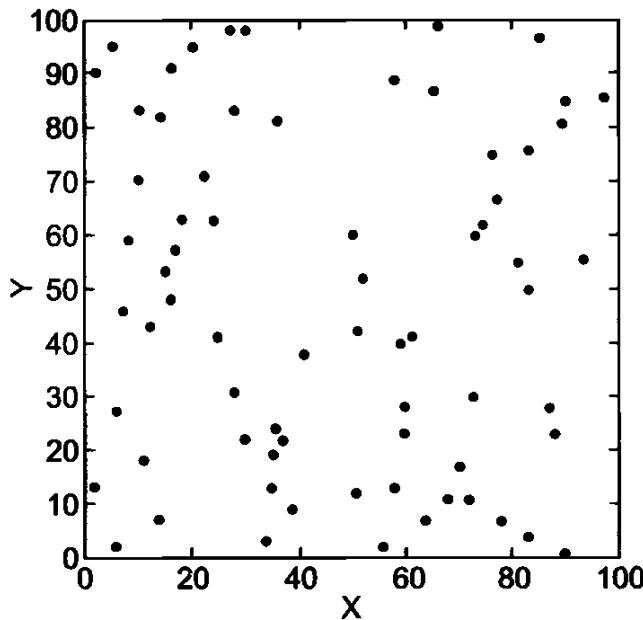
mi) po uderzeniu komety 65 milionów lat temu [379]. Zmiany w środowisku, jakie wywołało to uderzenie, były bardzo poważne. Większość gatunków była zarówno nieprzygotowana, jak i niezdolna do adaptacji do nowych warunków. Ewolucja nie gwarantuje pomyślnej adaptacji przy zmieniających się okolicznościach.

Mimo wszystko obliczenia ewolucyjne, jeśli próbujemy rozwiązywać problemy w zmieniającym się środowisku, mają wiele zalet w porównaniu z innymi heurystykami. Podstawowa zaleta polega na tym, że algorytmy ewolucyjne w każdym momencie utrzymują całą populację rozwiązań, a nie tylko jedno rozwiązanie. Pojawiają się w ten sposób potencjalne możliwości różnicowania podejść do rozwiązywania problemów. Gdy zmienia się problem, by przytoczyć banalne stwierdzenie, nie musimy postawić wszystkiego na jedną kartę. Jeśli ograniczenia się zmieniają i jedno z rozwiązań stanie się niedopuszczalne, to, być może, inne sensowne rozwiązanie wciąż będzie dopuszczone. Możemy przechodzić od rozwiązania do rozwiązania w populacji, by stwierdzić, czy któreś z innych dostępnych rozwiązań ma dla nas wartość.

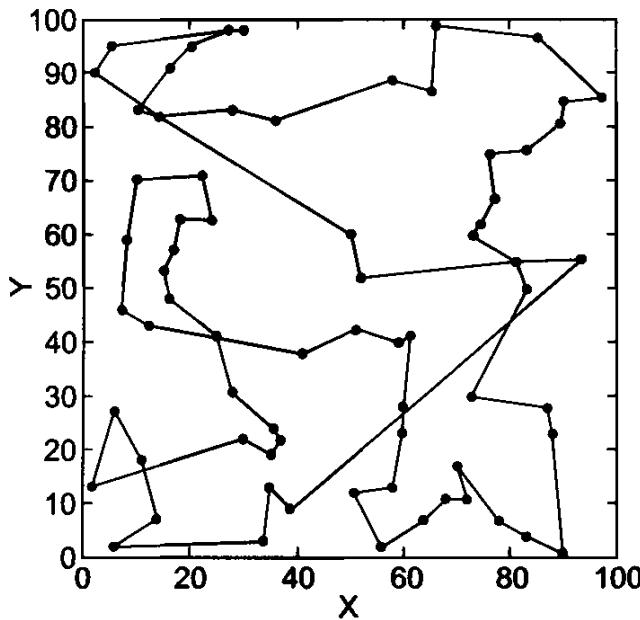
Druga, pokrewna zaleta takich algorytmów polega na tym, że możemy zacząć poszukiwanie nowych rozwiązań, korzystając z różnorodności osobników w populacji. Każde dostępne rozwiązanie-kandydat jest punktem początkowym do znalezienia nowych sposobów radzenia sobie z problemem, niezależnie od tego, jakie zmiany mogą zajść. Nie musimy tutaj polegać na jednym punkcie początkowym i nie musimy budować nowego rozwiązania, zaczynając od *tabula rasa*. Jeśli istnieją jakieś podobieństwa między starym a nowym problemem, to jest możliwe, że znajdą one swoje odbicie w rozwiązaniach obecnych w populacji.

Aby przyjrzeć się przykładowi potencjalnego zastosowania obliczeń ewolucyjnych do adaptacji przy zmieniającym się problemie, rozważmy alternatywne rozwiązania polegające na zastosowaniu do TSP albo algorytmu ewolucyjnego, albo algorytmu zachłanego rozważania najbliższych sąsiadów. Przypuśćmy, że chcieliśmy rozwiązać problem z 70 miastami. Na potrzeby tego przykładu rozmieściliśmy miasta jednostajnie losowo na kwadracie o boku o długości 100 jednostek. Pozycje miast przedstawiono na rys. 11.1. Patrząc na rysunek, trudno jest wyodrębnić jakiekolwiek oczywiste rozwiązania. Problem znalezienia odpowiedniej trasy dla tych miast wydaje się pewnym wyzwaniem.

Zacznijmy od zastosowania podejścia zachłanego. Przypuśćmy, że miasto na pozycji (6, 2) jest naszą bazą wypadową (jest to miasto w lewym dolnym rogu mapy). Nasza pierwsza decyzja polega na określeniu, które z pozostałych 69 miast znajduje się najbliżej. Patrząc na rysunek, moglibyśmy łatwo odrzucić niektóre z możliwości. Ponieważ jednak rozważamy zastosowanie algorytmu, musimy wykonać 69 porównań odległości między poszczególnymi miastami a naszą bazą w punkcie (6, 2). Po wykonaniu tej pracy stwierdzamy, że najbliższy jest punkt (14, 7). W następnej kolejności musimy określić, które z 68 miast jest najbliższe punktowi (14, 7). Odpowiedź brzmi (11, 18). I tak dalej. Na rysunku 11.2 pokazano powstałą w ten sposób trasę. Nie wydaje się to zbyt



Rysunek 11.1. Zestaw 70 miast, które ma odwiedzić komiwojażer, obejmujący jego bazę wypadową w punkcie (6, 2)



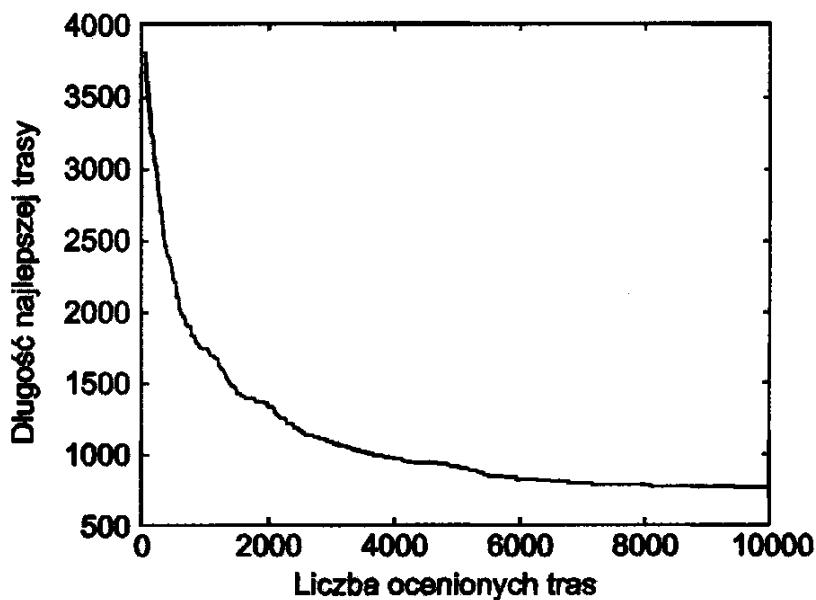
Rysunek 11.2. Rozwiążanie zachłanne rozpoczynające w punkcie (6, 2) i wybiegające za każdym razem najbliższego sąsiednie miasto. Zachłanność na krótką metę oznacza powstawanie kilku długich połączeń między miastami

dobre. Trasa ma bardzo długie ścieżki i krzyżuje się wiele razy. Sumaryczna długość całej trasy wynosi 843,95 jednostek¹.

Uzyskawszy to rozwiązanie, porównajmy je z rozwiązaniem, które możemy uzyskać w wyniku ewolucji. Przypuśćmy, że zastosowaliśmy reprezentację permutacyjną i ponumerowaliśmy miasta od 1 do 70 zgodnie z miejscami na liście uporządkowanej. Nasza początkowa populacja, żeby nie komplikować

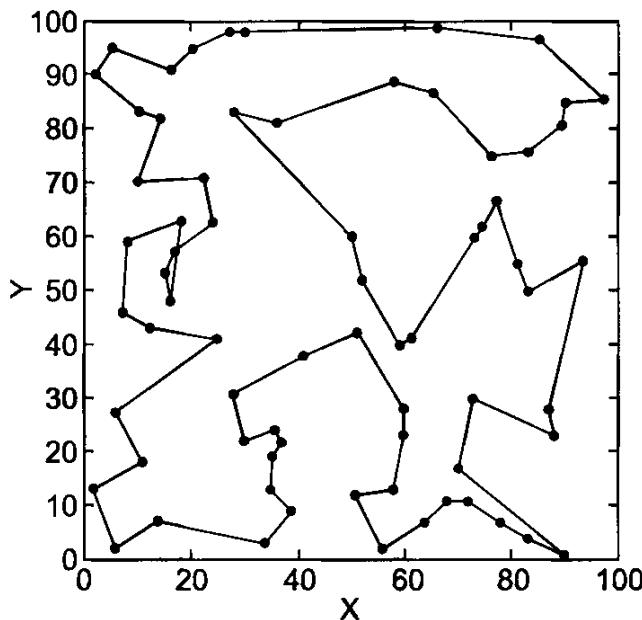
¹Stosując podany w rozdziale 8 wzór na wartość oczekiwana długości trasy komiwojażera, stwierdzamy, że rozwiązanie zachłanne jest około 35% gorsze od oczekiwanej najlepszej trasy.

opisu, będzie składała się z czterech tras rodzicielskich. Każda z nich jest tworzona losowo. Operator różnicowania, który będziemy stosowali, to opisana wcześniej w rozdziale 7 procedura transpozycji (odwracania). W procedurze tej z permutacji wybieramy dwie miejscowości i odwracamy kolejność ich występowania w permutacji (transponujemy je). I znowu, dla uproszczenia nasz operator różnicowania będzie dla danego rodzica generował tylko jednego potomka. Następnie operator selekcji dokona porównania długości wszystkich tras w populacji i zostawi cztery najlepsze z nich, które staną się rodzicami dla następnego pokolenia. Na rysunku 11.3 pokazano szybkość optymalizacji ewolucyjnej przy jednym uruchomieniu tej procedury. Długość najlepszej trasy w pierwszym pokoleniu wynosiła 3816,32; wartość ta jest o wiele gorsza od rozwiązania wcześniejszej znalezionej metodą zachłanną. Lecz po 2500 pokoleniach ewolucja doprowadziła do trasy z rys. 11.4, która ma długość 758,54 jednostek. Jest to o 21,04% gorzej niż wartość oczekiwana najlepszego rozwiązania, użyliśmy tutaj jednak populacji złożonej z czterech osobników, możemy zatem przewidywać, że osiągneliśmy lepszy rezultat, gdybyśmy zastosowali większą populację. Mimo to rozwiązanie z rys. 11.4 wygląda lepiej niż rozwiązanie z rys. 11.2 i rzeczywiście jest ono o 10,12% lepsze od rozwiązania zachłannego.



Rysunek 11.3. Szybkość ewolucyjnej optymalizacji długości znalezionej trasy przedstawiona jako funkcja liczby ocenionych tras. Przy czterech osobnikach rodzicielskich w każdej populacji 10 000 tras jest równoważne 2500 pokoleniom

Żeby ocena była sprawiedliwa, musimy jednak wziąć pod uwagę ilość obliczeń, które trzeba wykonać w obu procedurach. W wypadku algorytmu zachłannego obliczyliśmy 2415 odległości, żeby wykonać wszystkie konieczne porównania. W wypadku algorytmu ewolucyjnego, nie licząc obliczeń, które są konieczne do zaimplementowania operatora różnicowania, wykonaliśmy 10 000 sprawdzeń długości tras, co odpowiada obliczeniu około 700 000 odległości. W algorytmie ewolucyjnym wykonaliśmy zatem około 290 razy pracy więcej,



Rysunek 11.4. Najlepsza trasa uzyskana jako wynik ewolucji trwającej 2500 pokoleń. Chociaż rozwiązanie to nie jest idealne, jest ono o około 10% lepsze od rozwiązania uzyskanego zachłannie

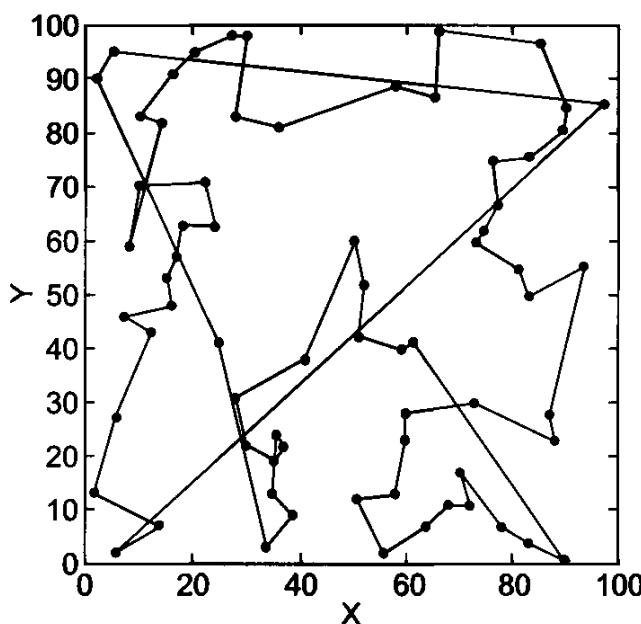
zakończyliśmy jednak na rozwiązaniu o 10% lepszym i o lepszych walorach estetycznych¹.

Powiedzmy teraz, że w chwili, gdy kierowaliśmy się ku drzwiom wyjściowym naszego domu w punkcie (6, 2), żeby zacząć naszą podróż po wszystkich miastach, zadzwonił do nas klient z miejscowości w punkcie (11, 18) i odwołał naszą dzisiejszą wizytę. Oznacza to, że musimy zmienić nasz plan.

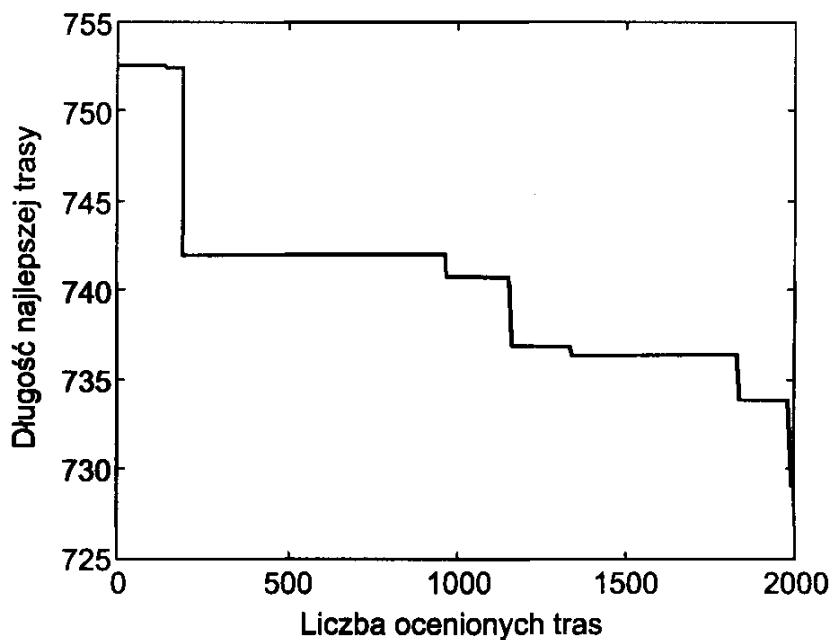
Przy stosowaniu algorytmu zachłannego musimy od początku zacząć nasze poszukiwanie trasy komiwojażera dla 69 miast. Musimy przy pierwszym wyborze wykonać 68 porównań itd. Po wyliczeniu 2346 odległości tworzymy nowe rozwiązanie przedstawione na rys. 11.5. Długość tej nowej trasy wynosi 962,64 jednostki. Zwróć uwagę, że jest to *więcej* niż w poprzednim zachłannym rozwiązaniu. Mimo że *usunęliśmy* miasto, długość trasy *wzrosła*. Nie wygląda to zachęcająco. Nasze nowe rozwiązanie jest o około 55% gorsze niż wartość oczekiwana najlepszego rozwiązania dla TSP z jednostajnie rozłożonymi 69 miastami.

Zamiast tego porównajmy, jak zmieni się plan, kiedy użyjemy algorytmu ewolucyjnego. Mamy już rozwiązanie poprzedniego problemu. Jeśli weźmiemy takie rozwiązanie i usuniemy miasto, które odpowiada punktowi (11, 18), to tak powstała nowa trasa może być punktem początkowym naszej populacji. Stworzymy trzy dodatkowe kopie tego rozwiązania, dzięki czemu będziemy mieli znowu

¹ W rzeczywistości uwaga ta jest bardzo serio. W świecie rzeczywistym często ludzie będą wołali rozwiązanie, które jest w sposób wymierny *gorsze* niż inne, a które ma pewne walory estetyczne. Dodatkowo, jeśli chodzi o sprawiedliwość, to algorytm ewolucyjny prześcignął rozwiązanie zachłanne po sprawdzeniu 96 250 tras, co stanowi około 160 razy więcej pracy, niż było wykonane przy rozwiązaniu zachłannym. Jest to i tak znaczco większy wysiłek.



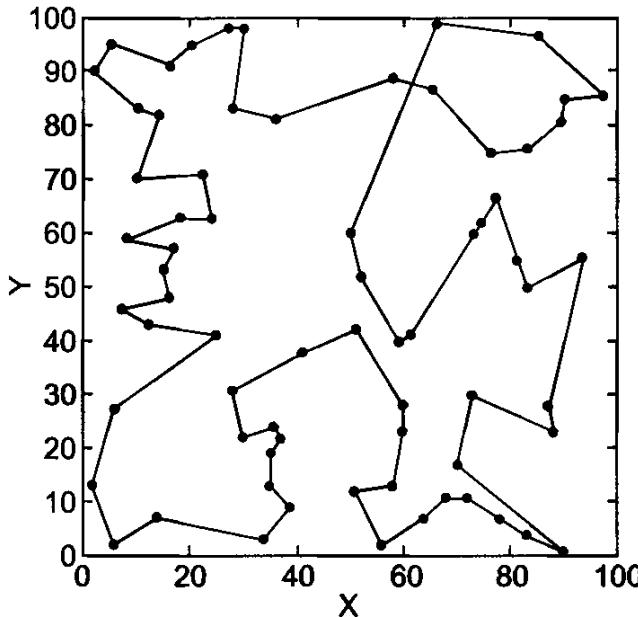
Rysunek 11.5. Po usunięciu klienta z punktu (11, 18) rozwiązanie zachłanne po-garsza się, chociaż rozmiar problemu jest mniejszy. Zauważ, że dwa ostatnie odcinki trasy są nadzwyczaj długie i że przechodzą przez prawie cały obszar obsługiwany przez komiwojażera



Rysunek 11.6. Ewolucyjna optymalizacja po zmianie problemu przez usunięcie klienta na pozycji (11, 18). Algorytm ewolucyjny zaczyna tam, gdzie poprzednio zakończył

cztery osobniki rodzicielskie w początkowej populacji. Jeśli teraz przez 500 pokoleń zastosujemy algorytm ewolucyjny, to uzyskamy krzywą optymalizacyjną z rys. 11.6. Nasze początkowe rozwiązanie w pierwszym pokoleniu miało długość 752,51 jednostek, a poprawiliśmy je do długości 729,01 jednostek. Nowe najlepsze uzyskane ewolucyjnie rozwiązanie pokazano na rys. 11.7.

Zauważmy, że gdy zaczynaliśmy od najlepszego wyniku ewolucji dla TSP z 70 miastami, już wówczas nasza początkowa populacja dla nowego TSP z 69



Rysunek 11.7. Najlepsze wyewoluowane rozwiązanie nowego TSP z 69 miastami.

Populacja początkowa została utworzona na podstawie populacji uzyskanej z poprzedniego TSP dla 70 miast. I znowu, po ocenieniu dalszych 2000 tras najlepsze znalezione rozwiązanie nie jest doskonałe (drogi się krzyżują), mamy tutaj jednak znacznie lepszy wynik niż w rozwiązaniu uzyskanym zachłannie. Co więcej, łatwo jest wyobrazić sobie heurystykę „oczyszczającą” trasę taką jak ta, która sugeruje, że są możliwe synergie wynikające z hybrydyzacji podejść (zob. rozdział 16)

miastami była lepsza od rozwiązania otrzymanego przez algorytm zachłanny. Dodatkowe przetwarzanie, które wykonaliśmy w algorytmie ewolucyjnym, służyło tylko dalszej poprawie trasy. Odwrotnie niż w algorytmie zachłannym, który po usunięciu miasta wygenerował rozwiązanie w rzeczywistości gorsze, algorytm ewolucyjny umiał użyć informacji, która została zawarta we wcześniejszym uporządkowaniu miast, i szybko dokonać ponownej optymalizacji dla nowego problemu. Co więcej, algorytm ewolucyjny z tego przykładu był bardzo ograniczony, gdyż wykorzystaliśmy tylko cztery osobniki. Gdyby rozmiar populacji był większy, moglibyśmy się spodziewać lepszego dostosowywania się do zmian w problemie.

Oprócz opisanej wyżej zdolności do adaptacji w dynamicznie zmieniającym się środowisku, algorytmy ewolucyjne dysponują potencjałem do uwzględniania możliwych zmian warunków bezpośrednio w ocenie rozwiązania. Jeśli możemy ograniczyć możliwe wyniki i przypisać im sensowne prawdopodobieństwa, to możemy optymalizować rozwiązania ze względu na wiele różnych kryteriów: 1) wartość oczekiwana wyników ważoną na podstawie prawdopodobieństwa, 2) wynik o maksymalnym prawdopodobieństwie, 3) najgorszy wynik itd. Na przykład, gdybyśmy opracowywali sieć komunikacyjną z wieloma połączonymi ze sobą węzłami, moglibyśmy oceniać każdego kandydata na rozwiązanie nie tylko w kategoriach jego funkcjonalności, gdy sieć działa bez zarzutu, ale także, gdy połączenia ulegają awariom. Jakość rozwiązania przy możliwych takich awariach systemu może zostać dołączona do ogólnej wartości rozwiązania.

Na końcu zaś rozwiążanie, będące wynikiem ewolucji, będzie przetestowane na okoliczność wystąpienia wielu „awarii” i będziemy mogli żyć w przekonaniu, że będzie mogło odpowiednio na nie reagować [423].

Przypominasz sobie pewnie, że w ostatnim akapicie rozdziału 9 sugerowaliśmy potrzebę powstania parametryzowanego generatora przykładów testowych, który umożliwiłby systematyczne analizowanie różnych metod obsługi ograniczeń. Podobna potrzeba pojawia się w przypadku dynamicznie zmieniającego się środowiska i pierwsze próby utworzenia takich generatorów przykładów testowych już się pojawiły [472].

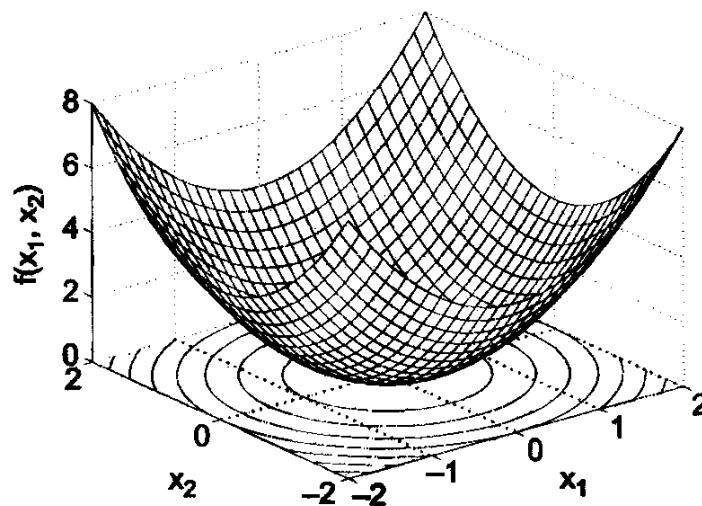
11.2. Świat rzeczywisty jest zaszumiony

Inna trudność, pojawiająca się przy rozwiązywaniu problemów, polega na tym, że wszystkie nasze obserwacje są obciążone błędem. Możemy wielokrotnie mierzyć tę samą wielkość i za każdym razem otrzymywać inne wyniki. Częściowo jest to rezultatem niewystarczającej dokładności naszych przyrządów. Dodatkowo istnieją zewnętrzne czynniki, które mogą mieć wpływ na wykonywane przez nas pomiary. Gdybyśmy mierzyli, na przykład, ilość ziarna zbieranego w maju z, powiedzmy, określonego akra ziemi w Kolumbii Brytyjskiej, moglibyśmy zapisać tę wartość corocznie i stwierdzić, że za każdym razem była inna liczba. Na ilość zboża ma wpływ temperatura w zimie, opady i wiele innych czynników, na które nie mamy wpływu. Typowy sposób radzenia sobie z takimi zewnętrznymi zmiennymi polega na ujęciu ich w ogólną zmienną odpowiadającą „szumowi”, a następnie zamodelowaniu tego szumu lub, być może, po prostu przyjęciu pewnych założeń na jego temat (np. że ma rozkład Gaussa o średniej zero). Jeśli nie możemy przyjąć takich założeń, to często nie mamy możliwości bezpośredniego obliczania odpowiedzi i musimy użyć algorytmu poszukiwania, który znajdzie zestaw parametrów, w odpowiednim sensie optymalny.

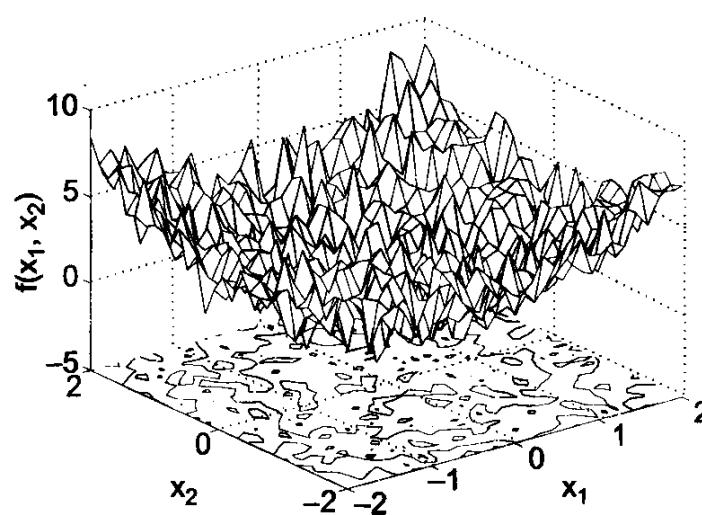
W ten sposób wracamy do koncepcji przeszukiwania krajobrazu przystosowania lub błędów w celu znalezienia maksymalnej lub minimalnej wartości, lecz wartość, którą uzyskamy dla każdego zestawu regułowanych parametrów, zmienia się przy każdym jej obliczeniu. Nawet jeśliśmy przeszukiwali przestrzeń parametrów pokazaną na rys. 11.8 (misa funkcji kwadratowej) przy narzucie spowodowanym przez szum, to podstawowa struktura funkcji może zostać utracona lub w najlepszym razie może być trudna do wyodrębnienia (rys. 11.9).

Stwarza to istotny problem przy stosowaniu metod gradientowych, gdyż jeśli podstawowa struktura funkcji nie jest znana z góry, to przybliżanie gradien- tu w dowolnym momencie staje się bardzo niepewne. Tak po prostu wygląda działanie szumu. Podobnie metody wspinania się (lub równoważnie zstępowa- nia) mogą mieć tutaj problemy, gdyż wykonywane kroki będą podejmowane na podstawie losowego szumu, który powinien być eliminowany.

W samej rzeczy tego typu ograniczenia dotyczą wszystkich metod opartych na próbkowaniu. Pojawiają się one zarówno we wspinaniu się, jak i, powiedzmy, w symulowanym wyżarzaniu lub algorytmach ewolucyjnych. Różnica jest taka, że w tym ostatnim wypadku mamy do czynienia nie z jednym rozwią-



Rysunek 11.8. Misa funkcji kwadratowej



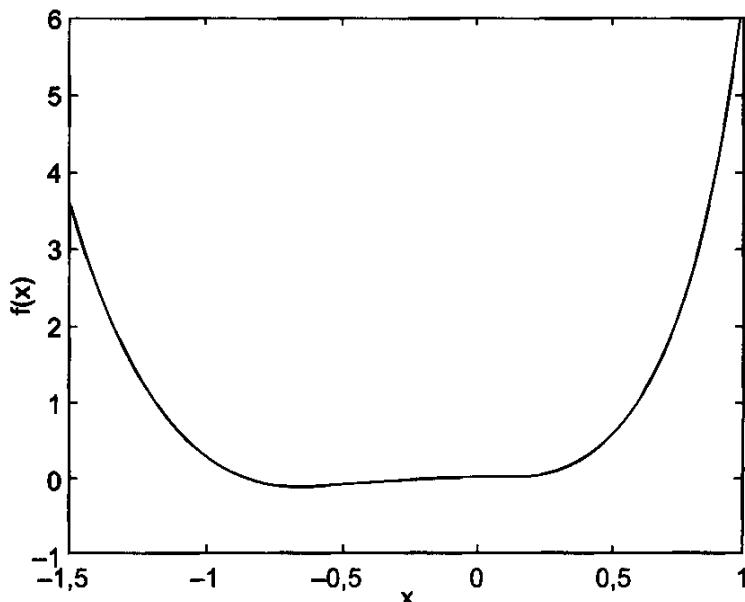
Rysunek 11.9. Misa funkcji kwadratowej, w której do każdego punktu próbkowania dodano standardowy szum Gaussa. Chociaż metody gradientowe mogą być bardzo efektywne przy znajdowaniu ekstremów funkcji takich jak na rys. 11.8, po dodaniu szumu do oceny rozwiązań-kandydatów metody te mogą zawieść

zaniem, ale z populacją konkurencyjnych rozwiązań, do których możemy się odwołać. W pewnym sensie populacja zawiera więcej informacji na temat kształtu krajobrazu, niż możemy zawrzeć w jednym punkcie. W kolejnych pokoleniach średnia populacji może odzwierciedlać uzyskane w wielu próbach statystyczne średnie przystosowanie. Inna zaleta algorytmów ewolucyjnych polega na tym, że rozwiązanie, aby być utrzymane, musi konkurować w sposób efektywny, ale nie tylko z jednym poprzednim najlepszym rozwiązaniem, ale raczej z grupą rozwiązań. To często oznacza, że szczęśliwy punkt, który za jakimś razem został dobrze oceniony, zostanie w następnym pokoleniu odrzucony. Dla kontrastu, gdy metoda wspinania się natknie się na punkt, który został dobrze oceniony w wyniku szczęśliwego wpływu losowego szumu, wówczas przejście w inne miejsce przez przeszukiwanie jego bezpośredniego otoczenia staje się trudne, chyba że zostaną zastosowane dodatkowe heurystyki lub ten punkt zostanie ponownie wybrany w wyniku próbkowania.

Stopień trudności, z jakim mamy tutaj do czynienia, zależy częściowo od stosunku sygnału do szumu. W miarę jak stosunek ten zmniejsza się, struktura pierwotnej powierzchni staje się mniej widoczna. Może to być poważnym wyzwaniem dla niektórych klasycznych metod optymalizacji, zwłaszcza gdy funkcja ma wiele ekstremów lub gdy ma głębokie rowki. Zilustrujemy ten problem za pomocą prostej funkcji przedstawionej na rys. 11.10:

$$x^4 + 2x^3e^x$$

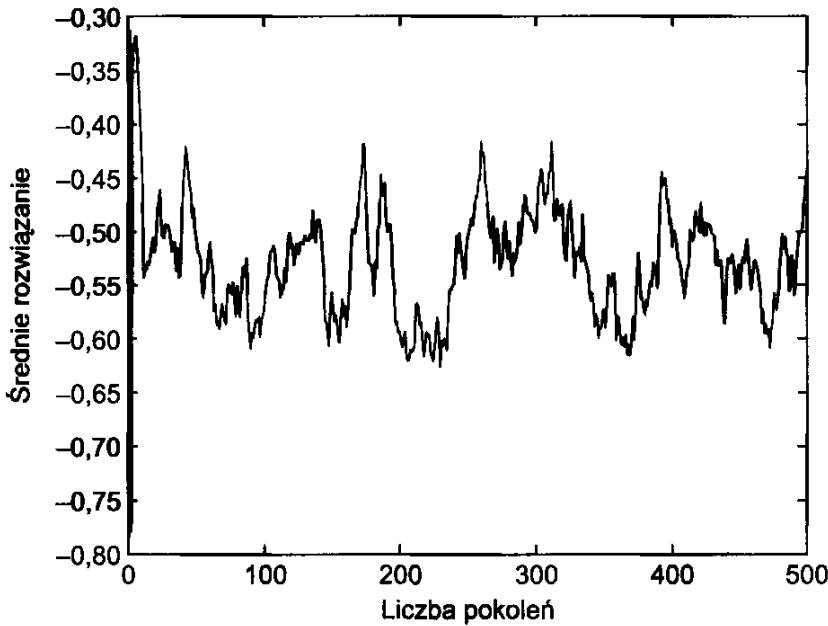
Zastosujemy tutaj metodę Brenta (zob. rozdział 3), polegającą na wykonywaniu kolejnych obliczeń funkcji w celu znalezienia jej minimum. Jeśli zaczniemy od przedziału $[-5, 5]$ z początkowym punktem środkowym 3, to minimum zostanie znalezione w punkcie $-0,630591$ z tolerancją błędu 10^{-6} . Co się jednak stanie, gdy przy każdym wywołaniu funkcji oceny metodą Brenta do funkcji dodamy standardowy szum Gaussa? Uśrednione minimum uzyskane w 100 niezależnych próbach znajduje się w punkcie $-0,396902$, a jego standardowe odchylenie wynosi 0,391084. Dodanie szumu ma duży wpływ na końcowy wynik.



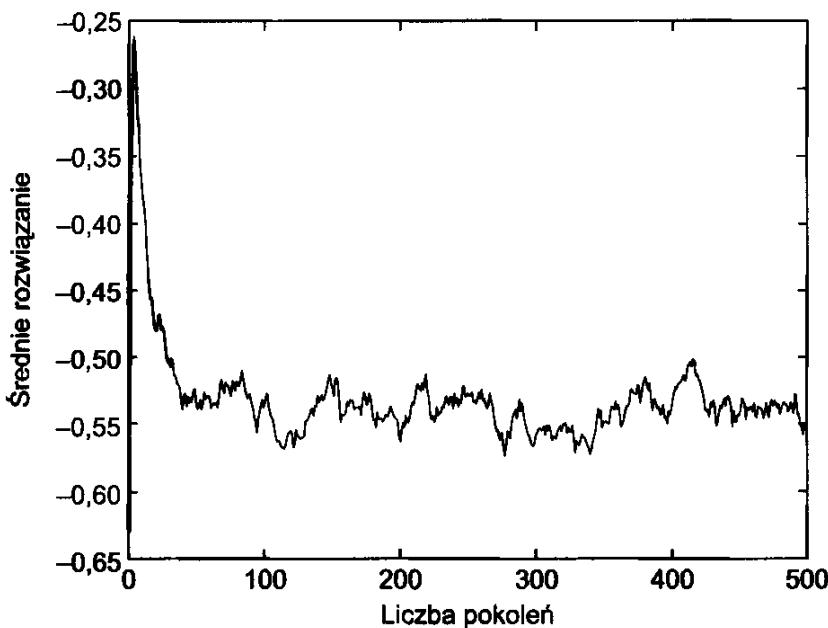
Rysunek 11.10. Funkcja $f(x) = x^4 + 2x^3e^x$

Dla porównania zastosujmy tutaj algorytm ewolucyjny z populacją o rozmiarze 100 osobników rodzicielskich. Osobniki w populacji początkowej są inicjowane wartościami z przedziału $[-5, 5]$, jeden potomek na rodzica jest generowany przez dodanie do rodzica zmiennej losowej Gaussa o wartości średniej równej zero i o ustalonym odchyleniu standardowym równym 0,01. W wyniku selekcji w każdym pokoleniu pozostanie 100 najlepszych osobników. Na rysunku 11.11 pokazano wykres wartości średniego rozwiązania w populacji w zależności od liczby pokoleń. Średnia przy 500 pokoleniach wynosi $-0,506095$ przy odchyleniu standardowym 0,152638. Na rysunku 11.12 przedstawiono średnie rozwiązanie w populacji w kolejnych pokoleniach przy rozmiarze populacji zwiększym do 1000 osobników rodzicielskich. Średnia po 500 pokoleniach wynosi $-0,528244$ przy standardowym odchyleniu 0,051627. Wprowadzenie losowego szumu do funkcji znieksztalcza proces ewolucji, oddalając go od

prawdziwego minimum, wpływ szumu nie jest jednak tak duży jak w przypadku metody Brenta.



Rysunek 11.11. Średnie rozwiązanie w populacji 100 kandydatów ewoluującej przez 500 pokoleń przy poszukiwaniu minimum funkcji $f(x) = x^4 + 2x^3e^x$ przy dodatkowym szumie Gaussa o średniej zero i odchyleniu standardowym 0,01



Rysunek 11.12. Średnie rozwiązanie w populacji 1000 kandydatów ewoluującej przez 500 pokoleń przy poszukiwaniu minimum funkcji $f(x) = x^4 + 2x^3e^x$ przy pewnym dodatkowym szumie

Chociaż dodanie szumu do funkcji miało negatywny wpływ na metodę Brenta, tutaj efekt ten nie był tak druzgocący. Użyta tu metoda mogła w dalszym ciągu wygenerować punkty, które możemy uważać za bliskie minimum, a średnie rozwiązanie przy 100 próbach znajdowało się w odległości od optimum mniejszej niż jedno odchylenie standardowe. Wpływ szumu na klasyczne

metody optymalizacyjne może być przerzążający, zwłaszcza gdy musimy uzyskać przybliżenie gradientu funkcji. Tego typu sytuacja pojawia się dość często, gdyż zwykle nie możemy zapisać funkcji oceny w taki sposób, żeby możliwe było bezpośrednie obliczenie pochodnej. Techniki wykorzystujące przybliżone gradienty oparte na próbkowaniu funkcji zwykle korzystają z różnicy w wysokości funkcji oceny podzielonej przez odległość punktów, w których były pobierane wartości:

$$\frac{g(x_1) - g(x_2)}{|x_1 - x_2|}$$

przy czym $|x_1 - x_2|$ powinna być jak najmniejsza. Po uwzględnieniu szumu $g(x_1)$ i $g(x_2)$ możemy zapisać jako

$$g(x_1) = f(x_1) + h(x_1) \text{ oraz } g(x_2) = f(x_2) + h(x_2)$$

przy czym: $f(\cdot)$ jest prawdziwą wartością rozwiązania, $h(\cdot)$ jest dodanym do niej szumem. Jeśli dodane wyrażenia oznaczające szum są statystycznie niezależne, to wariancja szumu dla różnicy $g(x_1) - g(x_2)$ jest *sumą* wariancji wartości $h(x_1)$ i $h(x_2)$. Jeśli $h(\cdot)$ jest, powiedzmy, standardową zmienną losową Gaussa o zerowej średniej i jednostkowej wariancji, to zmienna losowa $G = g(x_1) - g(x_2)$ będzie miała wariancję 2,0. Jeśli odchylenie standardowe zmiennej G jest większe od różnicy między $f(x_1)$ a $f(x_2)$, to trudne będzie uzyskanie wiarygodnego oszacowania nawet poprawnego znaku gradientu. W tej sytuacji metody, w których przy określaniu kierunku poszukiwań korzysta się z przybliżonego gradientu, można wyrzucić do kosza.

11.2.1. Zapewnianie różnorodności

Nierzadko cenna okazuje się różnorodność, którą możemy uzyskać dzięki rozważaniu zbioru potencjalnych rozwiązań. W typowym algorytmie ewolucyjnym nie ma jednak żadnego mechanizmu gwarantującego różnorodność populacji. Selekcja powoduje, że odrzucane są rozwiązania najmniej przystosowane. W wyniku tego populacja może być zbliżona do jednego punktu lub zestawu punktów, które znajdują się w pewnym małym otoczeniu najlepszego, dotychczas znalezionej rozwiązania. Początkowe równomierne pokrycie krajobrazu funkcji przystosowania, które uzyskujemy przez losowe rozłożenie punktów, bardzo szybko jest tracone, a po kilku pokoleniach wszystkie rozwiązania w populacji mogą być w zasadzie bardzo podobne. Jeśli rozwiązania są takie same, to w przypadku dynamicznych i zaszumionych środowisk zalety związane z populacją rozwiązań tracą na wartości. W związku z tym pojawia się naturalna potrzeba opracowania sposobów zapewniania różnorodności populacji.

Jedna z możliwości jest określana mianem *niszowania*. W ekologii nisza oznacza całość wykorzystywanych przez organizm ożywionych i nieożywionych zasobów jego środowiska. Elementami takiej niszy są na przykład zakres temperatur, umiejscowienie kryjówek czy źródło pokarmu. W obliczeniach ewolucyjnych niszowanie odnosi się do pomysłu, że współzawodniczące gatunki nie mogą współzestępować w tej samej niszy (zasada wykluczania konkurencyjnego). Jednym ze sposobów wprowadzenia takiego pojęcia do ewolucyjnego

algorytmu optymalizacyjnego jest uwzględnienie wymagania, żeby każde rozwiązanie współdzieliło swoje przystosowanie z innymi rozwiązaniami, które znajdują się na tym samym obszarze przestrzeni przeszukiwania. To tak zwane *współdzielenie przystosowania* (ang. *fitness sharing*) działa na zasadzie przyznania każdemu konkretному rozwiązaniu i przystosowania

$$F'_i = \frac{F(i)}{\sum_{j=1}^n sh(d(i, j))}$$

przy czym: n oznacza liczbę rozwiązań w populacji, $F(i)$ – wynik otrzymany z funkcji oceny (celem jest maksymalizacja), $d(i, j)$ – „odległość” od rozwiązania i -tego do j -tego, $sh(d(i, j))$ – funkcję współdzielenia zwykle określona jako

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^{\alpha} & \text{jeśli } d < \sigma_{share} \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

przy czym: σ_{share} oznacza rozmiar otoczenia wokół i -tego rozwiązania, a α jest parametrem skalującym. Im bliżej siebie są dwa rozwiązania, czy to co do ich struktury, czy co do ich funkcji, tym bardziej są karane za zajmowanie sąsiednich pozycji. Takie obciążenie wymusza na populacji, żeby pozostała rozproszona między wieloma ekstremami, przy czym każde wznieśienie krajobrazu funkcji może przyjąć liczbę osobników proporcjonalną do jego wysokości.

Chociaż opis tej metody brzmi zachęcająco, ma też ona pewne wady. Podstawową z nich jest to, że wpływ współdzielenia przystosowania jest określany za pomocą dostosowania parametrów σ_{share} i α . Różne ich wybory będą dawały w przestrzeni poszukiwań różne grupy rozwiązań. To, które z nich jest najlepsze, zależy od problemu oraz od funkcji oceny. Wartość σ_{share} powinna być wystarczająco mała, żeby można było rozróżnić szczyty, ale skąd mamy się dowiedzieć, jak mała powinna być, zanim zaczniemy rozwiązywać problem? Zwykle dochodzi do tego, że wykonujemy kilka doraźnych zmian parametrów, aż osiągnięta różnorodność okaże się akceptowalna. Z punktu widzenia praktyka zastosowania takiej procedury może być prawdziwym wyzwaniem, gdyż otrzymane wyniki zależą bardzo silnie od doboru parametrów, a podjęcie dobrej decyzji co do ich wartości może być dość trudne. Inną, drugorzędną wadą, jest koszt obliczeniowy związany z porównywaniem rozwiązań, określaniem ich wzajemnej odległości i obliczeniem współdzielonego przystosowania. Przy dużych rozmiarach populacji procedura ta może być obliczeniowo nieosiągalna.

Inna metoda wymuszania różnorodności jest nazywana *stłaczaniem* (ang. *crowding*). Przy przeprowadzaniu selekcji w celu określenia, które rozwiązania mają pozostać, nowe rozwiązania podobne do już istniejących po prostu zastępują stare. W ten sposób w populacji zanika tendencja do wytwarzania nadmiernej liczby podobnych rozwiązań. Co więcej, konieczne jest tutaj wprowadzenie miary odległości określającej stopień podobieństwa rozwiązań, a odpowiednie ustalenie tego kryterium zależy od problemu.

Każda z tych metod – niszowanie i stłaczanie – wymaga wprowadzenia miary odległości między dowolnymi nowymi rozwiązaniami. Przy zajmowaniu

się problemami, w których rozwiązania znajdują się w zbiorze liczb rzeczywistych, sprawia wydaje się prosta, gdyż możemy stosować znane nam postacie miary (np. miarę euklidesową). Przy problemach kombinatorycznych związanych dyskretnymi przestrzeniami sprawy nie są tak proste. Jeśli rozwiązania są reprezentowane jako ciągi binarne, to możemy mieć ochotę na zastosowanie odległości Hamminga, co jednak robić, gdy rozwiązania są permutacjami? Jaka jest odległość między $[1 \ 3 \ 2 \ 4 \ 5 \ 7 \ 6]$ a $[3 \ 6 \ 7 \ 5 \ 1 \ 2 \ 4]$? Na pytanie to nie ma oczywistej odpowiedzi. Co ważniejsze, odległość między dwoma rozwiązaniami może być lepiej wyrażana w kategoriach ich funkcjonalności niż w kategoriach kodowania. Na przykład, gdybyśmy rozwiązywali TSP, wówczas dwa rozwiązania $[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7]$ i $[4 \ 5 \ 6 \ 7 \ 1 \ 2 \ 3]$ reprezentujące uporządkowane listy miast, które mają być odwiedzone, z uwzględnieniem powrotu z ostatniego miasta do pierwszego, są funkcjonalnie równoważne. Obie listy opisują dokładnie tę samą trasę. Intuicyjnie odległość między tymi rozwiązaniami jest rzeczywiście równa zeru. Taka sytuacja stanowi naprawdę ważny problem. Moglibyśmy bez wątpienia zakodować obie te permutacje za pomocą jakiegoś języka binarnego. Gdybyśmy tak zrobili, chcielibyśmy do określania ich podobieństwa stosować odległość Hamminga. Byłaby to jednak sztuczna miara pokrewieństwa tych rozwiązań, która z dużym prawdopodobieństwem nie miałaby wiele wspólnego z określaniem podobieństwa co do funkcjonalności. Podobnych argumentów możemy użyć dla dowolnej innej reprezentacji, którą moglibyśmy obrać. Czasami najoczywitsza miara odległości nie nadaje się do użytku!

Jeszcze inna metoda utrzymywania różnorodności populacji polega na takim ustawieniu algorytmu ewolucyjnego, żeby konkurowały ze sobą całe populacje, z których każda byłaby oceniana nie tylko na podstawie wartości przystosowania osobników, które w niej przeżywają, ale także na podstawie ich różnorodności. Jeden z możliwych sposobów mierzenia jakości populacji jest opisany wzorem

$$Q(\mathbf{x}) = \alpha \bar{f} + (1 - \alpha) \sigma_{\mathbf{x}}^2$$

przy czym: \mathbf{x} jest wektorem rozwiązań w populacji, \bar{f} – średnią wartością wszystkich rozwiązań w \mathbf{x} , $\sigma_{\mathbf{x}}^2$ – wariancję rozwiązań w \mathbf{x} , α – współczynnikiem skalującym, który służy do wyważania udziału przystosowania rozwiązań i różnorodnością populacji. Każda populacja przyjmuje miejsce typowego jej osobnika. Selekcja jest tutaj wykorzystywana do pozbywania się całych populacji rozwiązań, a różnicowanie – do generowania zupełnie nowych populacji przez losowe zmiany w populacjach istniejących. Jednocześnie dla każdej populacji działa jej algorytm ewolucyjny, dzięki czemu mamy tutaj do czynienia z hierarchiczną strukturą, w której na niższym poziomie osobniki podlegają różnicowaniu i selekcji w ramach swojej populacji, na wyższym zaś tym samym procesom podlegają całe populacje. Koncepcja hierarchicznego ciągu algorytmów ewolucyjnych jest interesująca, jednak ewidentną wadą tego podejścia jest jego złożoność obliczeniowa i wymagania pamięciowe związane z koniecznością utrzymywania populacji.

Jeszcze jednym sposobem zapewnienia różnorodności populacji jest sprawienie, aby pamiętała, gdzie już była. Istnieją dwa sposoby realizacji tego pomysłu: albo za pomocą *jawnych* struktur pamięciowych, albo za pomocą struktur *ukrytych*. Jawne struktury danych polegają na tym, że są w nich zapisywane konkretne zbierane podczas trwania ewolucji informacje, które na późniejszym etapie możemy ponownie wprowadzić do populacji. Trzeba tu przy tym odpowiedzieć sobie na kilka pytań.

- Co jest zapamiętywane i kiedy?
- Co jest wprowadzane ponownie (przypominane) i kiedy?
- Jakie są struktury pamięciowe i jakiej są wielkości?
- Co jest (jeśli zachodzi taka konieczność) usuwane z pamięci?

Opublikowano wiele różnych raportów na temat eksperymentów z pamięcią jawną. Louis i Xu [292] rozważali, na przykład, rozwiązanie, w którym określono było zapisywane najlepsze rozwiązanie (tzn. na końcu *każdego* okresu do pamięci był dodawany najlepszy osobnik z populacji). Po zmianie w środowisku pewien (mały) procent *r*-zainicjowanych osobników był pobierany z pamięci, a reszta osobników była ponownie inicjowana losowo. Pozostaje tutaj otwarty problem wykrywania „zmiany” w środowisku. Ramsey i Grefenstette [376] opracowali system, który także zapamiętywał najlepszego osobnika w populacji, osobnik ten był jednak zapisywany wraz z pewnymi zmiennymi opisującymi środowisko, dzięki czemu było ustanawiane połączenie między stanem środowiska a rozwiązaniem w tym stanie. Po zmianie w środowisku około 50% ponownie inicjowanych osobników pobierano z pamięci. Uwaga była skupiana na tych osobnikach, które były zapisywane przy środowiskach *podobnych* do nowego. I znowu to, jak określać zmianę, która wymusza odwołanie się do zapisanych osobników, i jak określać podobieństwo między środowiskami, pozostaje sprawą otwartą.

Mori i in. [326] eksperymentowali z systemem, w którym najlepszy osobnik był zapamiętywany w pamięci *każdego* pokolenia. Zawsze jednak dysponujemy ograniczoną pamięcią, a więc w każdym pokoleniu jeden osobnik był usuwany z pamięci. Decyzja co do tego, który to miał być osobnik, była oparta na liczbie pokoleń, przez które osobnik był pamiętany oraz na jego wkładzie do różnorodności pamiętanych osobników. Osobniki z pamięci mogły być używane jako nowi rodzice w procesie ewolucyjnym. Trojanowski i in. w [468, 469] eksperymentowali ze strukturami pamięciowymi związanymi z poszczególnymi osobnikami. Osobnik „pamiętał”, ze względu na ograniczoną pamięć, swoich niedawnych przodków. Po zmianie w środowisku każdy osobnik był oceniany razem ze wszystkimi rozwiązaniami znajdującymi się w jego pamięci. Spośród wszystkich tych rozwiązań najlepsze rozwiązanie stawało się nowym bieżącym rozwiązaniem.

W przeciwnieństwie do pamięci jawnej pamięć ukryta pozostawia algorytmowi ewolucyjnemu sterowanie tym, co zapamiętywać, kiedy to przypominać itd. Jednym ze sposobów osiągania tego jest zastosowanie samoadaptacji, którą opisaliśmy wcześniej w rozdziale 10. Strategie generowania nowych rozwiązań,

które sprawdziły się w przeszłości, są w pewnym sensie „zapamiętywane” przez potomstwo, które wygenerowały. Inna metoda jest oparta na pojawiających się w genetyce koncepcjach związanych z diploidalnością i dominacją. W modelu tym osobniki utrzymują dwa lub więcej rozwiązań (lub struktur danych), funkcja dominacji wybiera fragmenty jednej lub więcej struktur, na podstawie których buduje końcowe rozwiązanie. Podstawowym zagadnieniem jest tutaj opracowanie takiej funkcji, żeby powstawały sensowne wyniki.

Nie wiadomo, jak to robić. Goldberg i Smith [193], na przykład, pracowali na reprezentacji binarnej z dodatkowym urozmaiceniem polegającym na tym, że stosowali trzy wartości: 0, recesywne 1 i dominujące 1. Wartość 0 była wybierana wtedy i tylko wtedy, gdy odpowiednia wartość w drugiej strukturze (każdy osobnik składał się z dwóch struktur) również wynosiła 0 lub było to recesywne 1. W przeciwnym razie wybierano 1. Ng i Wong [335] rozszerzyli to podejście, uwzględniając recesywne i dominujące zera i jedynki. Ryan [404] zaproponował schemat addytywnej dominacji, w którym odpowiednie wartości (które mogły być liczbowe, a nie tylko binarne) były dodawane między strukturami danych, przy czym wynik był ustawiany na 0, jeśli suma znajdowała się poniżej progu równego 10.

Każde z tych podejść jest podejściem ad hoc. Jest to, być może, łagodny sposób stwierdzenia, że nie istnieje teoria, która mówiłaby, jakiej funkcji dominacji należy użyć, ile struktur zawrzeć w osobniku ani czy pozwolić funkcji dominacji zmieniać się w czasie. Lewis i in. w [281] zaobserwowali, że zmienianie funkcji dominacji jest istotne w środowiskach niestabilnych, ale obserwacja ta jest zbyt ogólna, żeby miała praktyczne znaczenie. Obecnie obszar badań związanych ze strukturami diploidalnymi i poliploidalnymi oraz funkcjami dominacji jest dużym obszarem obejmującym niewiele teorii, która mogłaby wspomóc praktyków.

Każda z tych metod – niszowanie, współdzielenie, zapamiętywanie – może być pomyślnie wykorzystana do zapewnienia różnorodności populacji, stanowiącej ochronę przed dynamicznym lub zaszumionym środowiskiem. Wszystkie one mają kilka parametrów, które należy ustawić, a każdy z tych parametrów zwiększa złożoność obliczeniową procedur. Najlepsze wykorzystanie tych i innych metod generujących różnorodne rozwiązania wymaga wykonywania dużej liczby ocen. Nie istnieją efektywne heurystyki, które pomogłyby dokonywać tutaj wyboru, oraz które byłyby ogólne i które by nie zależały od rozwiązywanego problemu.

11.3. Modelowanie trasy statku

W rozdziale 9 krótko omówiliśmy trudności związane z opracowywaniem funkcji oceny dla osobników dopuszczalnych i niedopuszczalnych, pojawiające się w kontekście problemu planowania trasy (zob. punkt 9.1.3), tzn. pojawiające się w środowisku, w którym jest wiele przeszkód. Jeszcze ciekawsze jest rozważanie problemów planowania tras, gdy niektóre z przeszkód zmieniają się dynamicznie, poruszając się w różnych kierunkach z różnymi prędkościami.

Scenariusz, którym się tutaj będziemy zajmowali, był rozważany w [434] w kontekście modelowania tras statków w sytuacjach kolizyjnych. W artykule przedstawiono eksperymenty ze zmodyfikowaną wersją oprogramowania Evolutionary Planner/Navigator (EP/N), opracowanego wcześniej dla środowisk statycznych [501]. To nowe rozszerzenie EP/N oblicza „optimalną pod względem bezpieczeństwa” trasę dla statku płynącego w środowiskach ze statycznymi i dynamicznymi przeszkodami. Ten nowy system uwzględnia także czas, zmieniątą prędkość statku oraz zmieniające się w czasie ograniczenia, które odnoszą się do ruchomych statków (zwanych także „obcymi statkami” lub „celami”; są to inne jednostki pływające, których należy unikać, uwzględniając bezpieczną odległość między mijającymi się celami, ich prędkość oraz kierunek ruchu).

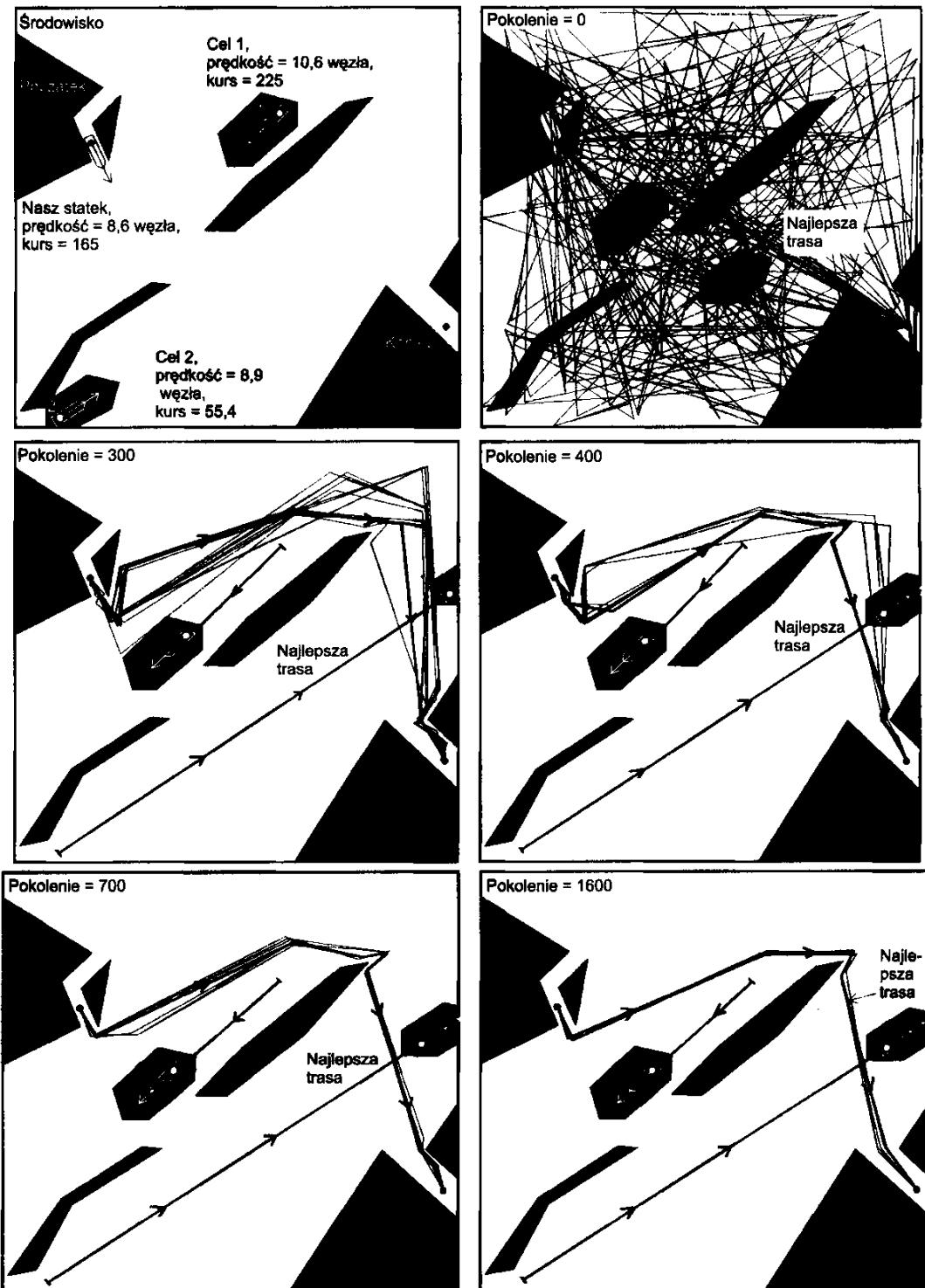
W nowej wersji EP/N użyto innej reprezentacji trasy, w której z każdym odcinkiem trasy jest związana dodatkowa wartość dotycząca prędkości obsługiwanej statku oraz oddzielna funkcja oceny, która uwzględnia czas konieczny do pokonania trasy, a także standardowe parametry związane z pomyślnym przebiegiem podróży oraz zachowaniem odległości od innych okrętów. W pracach [501] i [434] podano szczegóły pierwotnego systemu EP/N oraz jego rozszerzenia na dynamiczne środowiska.

Rozważmy teraz pewien interesujący problem nawigacyjny przejścia przez cieśninę i zobaczymy, jak EP/N go obsługuje. Założymy, że środowisko ma sześć statycznych ograniczeń nawigacyjnych (rys. 11.13) i dwa cele. Nasz statek wyrusza w pobliżu lewego górnego rogu mapy i musi osiągnąć punkt po przeciwnej stronie cieśniny (w pobliżu prawego dolnego rogu). Między dwiema wąskimi wyspami, znajdującymi się na środku cieśniny jest korytarz, jednak ruchy pierwszego i drugiego celu sprawiają, że przejście tą drogą może być trudne. Prędkość naszego statku wynosi $\vartheta = 8,6$ węzła.

Ponieważ cele 1 i 2 blokują naszemu statkowi przejście między dwiema wyspami, system opracowuje trasę (rys. 11.13), która „biegnie naokoło” wysp, a nie między nimi. W tym wypadku cel 1 nie stwarza zagrożenia kolizji, w dalszym ciągu należy jednak ominąć cel 2. Zadanie to jest realizowane przez mały manewr w okolicach górnego końca górnej wyspy – jest tam wykonywany ostry skręt w prawo, a następnie w lewo, dzięki czemu nasz statek omija sześciokąt celu 2. (Zwróć uwagę, że lokalizacje obszarów dynamicznych – czarne sześciokąty – są przedstawione w odniesieniu do najlepszej trasy. Rozmieszczenie ich zależy od momentu pierwszego skrzyżowania się trasy naszego statku z trasą celu).

Ciekawe jest, co się stanie, gdy eksperyment zostanie powtórzony, ale z mniejszą prędkością naszego statku, wynoszącą $\vartheta = 5,6$ węzła. Ponieważ nasz statek zbliża się wolniej do przesmyku między wyspami, możemy poświęcić trochę czasu na „manewry oczekujące” umożliwiające celowi 1 miniecie okolic przesmyku. Właśnie to robi EP/N. Jak widzimy na rys. 11.14, najlepsza trasa zaczyna się kilkoma małymi manewrami umożliwiającymi późniejsze przejście między wyspami i uniknięcie obu celów – nasz statek przepływa za rufami obu celów 1 i 2.

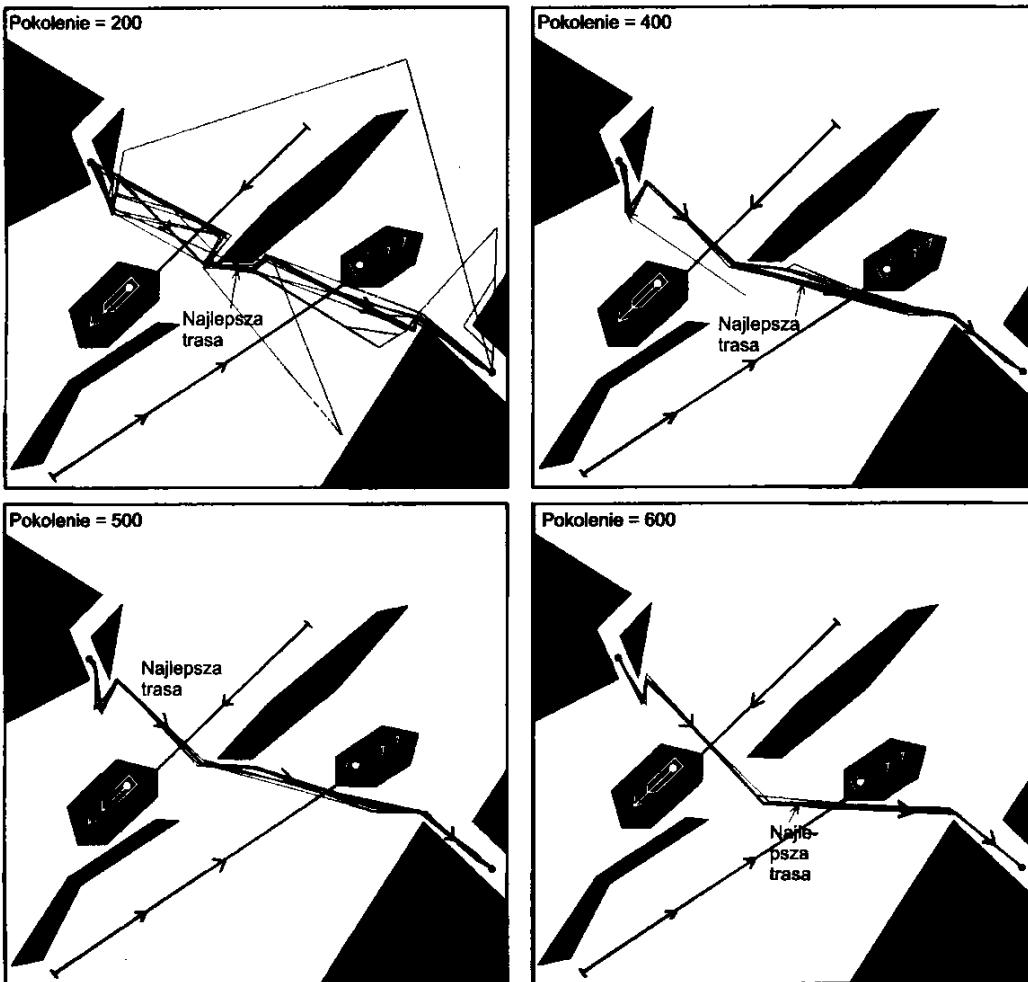
Jak widzimy, prędkość naszego statku może w istotny sposób wpływać na ostateczną najlepszą trasę. Co się zatem dzieje, jeśli nasz statek może zmieniać



Rysunek 11.13. Ewolucja trasy z uwzględnieniem dwóch poruszających się celów przy statycznych ograniczeniach nawigacyjnych. Nasz statek porusza się z prędkością $\vartheta = 8,6$ węzła

prędkość w trakcie manewrów? W rzeczywistości takie zmiany są dość ograniczone, gdyż zmiana trasy jest uważana przede wszystkim za manewr zapobiegający kolizji. Takie manewry mogą wiązać się ze zmianą prędkości, ale tylko wówczas, gdy zmniejsza ona prawdopodobieństwo kolizji. (Niechęć do zmiany prędkości *Titanica* jest uważana za jedną z przyczyn słynnej katastrofy [291]).

Załóżmy teraz, że prędkość naszego statku może wynosić 3,6, 8,6 lub 13,6 węzła (zwykle określane jako wolno naprzód, pół naprzód, cała naprzód). Ta prosta zmiana powoduje konieczność wprowadzenia dużej liczby ważnych zmian

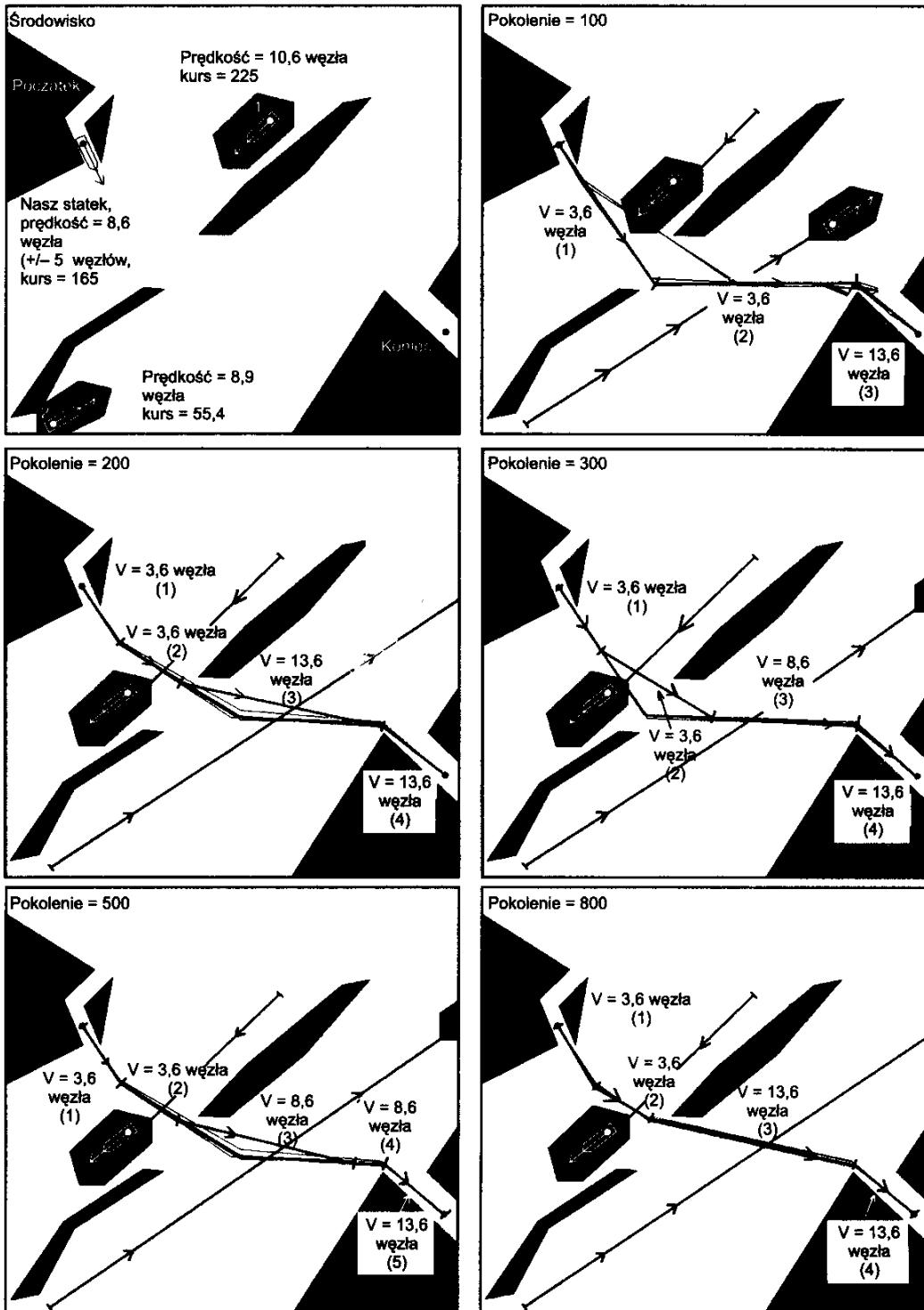


Rysunek 11.14. Ewolucja trasy z uwzgl dniением dw ch poruszaj cych si  cel w przy statycznych ograniczeniach nawigacyjnych. Nasz statek porusza si  z pr dko ci  $\vartheta = 5,6$ w z la

w kodzie systemu ewolucyjnego. Po pierwsze, pr dko ci statku jest reprezentowana przez zmienn  zwi zan  z ka dym odcinkiem trasy. Konieczne jest tak e wprowadzenie operatorów, kt re zmieniaj  warto ci pr dko ci. Je li pr dko ci jest wyra zana za pomoc  ma ego zbioru warto ci, to odpowiednie jest zastosowanie reprezentacji binarnej i prostego przełączania bitów. Je li natomiast pr dko ci mia aby si  zmienia  w sposób ciąg y, to bardziej odpowiednia jest reprezentacja zmiennopozycyjna i r  nicowanie Gaussa lub podobne.

Na rysunku 11.15 przedstawiono wynik eksperymentu, w którym statek móg  rozwija  trzy wymienione wcze sniej pr dko ci. Środowisko jest tak e samo jak w poprzednich doświadczeniach (prezentowanych na rys. 11.13 i 11.14). Zgodnie ze „zdrowym rozs adkiem” pocz atkowe odcinki najlepszej trasy s  promierzane z minimaln  pr dko ci  3,6 w z la, ko cowe za  dwa odcinki z pr dko ci  maksymaln  13,6 w z la.

Te proste przyk ady nie oddaj  pe nej sprawiedliwo ci omawianemu tutaj systemowi. Nasz statek jest „ wiadom” istnienia pewnych (ale nie wszystkich) potencjalnych cel w – tych widocznych na ekranie jego radaru. Co wi cej, statek ten zna tylko bie ac  pr dko ci i kierunek widocznych cel w (a w s wiecie rzeczywistym s  one tylko przybli zone). W zwi zku z tym „krajobraz” mo e



Rysunek 11.15. Ewolucja trasy z uwzględnieniem dwóch poruszających się celów przy statycznych ograniczeniach nawigacyjnych. Nasz statek porusza się ze zmienną prędkością

się zmienić w każdej chwili – każdy obcy statek może zmienić kierunek i/lub prędkość (reagując np. na ruchy innych statków lub naszego statku, albo też jakiegoś niewykrytego wcześniej okrętu), a każdy z celów może pojawiać się lub znikać z radaru naszego statku. W takich sytuacjach zmodyfikowany EP/N musi szybko podać nowe rozwiązanie i, tak jak we wcześniej omawianym problemie TSP, bieżąca populacja w danym momencie jest podstawą znajdowania w locie (lub raczej we flocie) nowych rozwiązań.

11.4. Podsumowanie

Przy rozwiązywaniu problemów ze świata rzeczywistego nie wystarczy znaleźć jednego rozwiązania i poprzestanie na tym. Trzeba w sposób ciągły znajdować nowe rozwiązania, w miarę jak zmienia się sam problem. Cel rozwiązywania problemu może się zmienić, a co za tym idzie powinno się zmienić rozwiązanie. Zmienić się mogą też dostępne zasoby i związane z nimi ograniczenia, a to znowu powinno zmienić rozwiązanie. Co więcej, otrzymywana przez nas informacja na temat tego, jak dobre jest nasze aktualne rozwiązanie, nie musi być w pełni dokładna. W związku z tym stojmy przed zadaniem wielokrotnego ulepszania naszych rozwiązań tak, żeby były dostosowane do ciągu zmieniających się w czasie problemów; co więcej, musimy przy tym uwzględnić nieprzewidziane zdarzenia i posiadanie niepełnej wiedzy. Fakt, że niepewność i niedokładność będzie elementem naszych modeli, odwzorowań i procesów, musi być zaakceptowany i, co więcej, wykorzystany w najlepszy możliwy sposób.

Chociaż nasz cel może nie zmieniać się w czasie, jednak gdy mamy do czynienia z sytuacją konkrowania z innymi, może się zmieniać sposób, w jaki nasz cel i cele naszych konkurentów oddziałują na siebie. Wymaga to znowu ponownego analizowania rozwiązań problemów, z którymi mamy do czynienia. Przyjmijmy, że próbujesz stworzyć dla swojego miasta najlepszy zespół sportowy. Co to znaczy „najlepszy”? Jaki jest Twój cel? Po pierwsze, wygrać najwięcej meczów. Po drugie, przyciągnąć jak najwięcej fanów. A robi się to oczywiście, wygrywając, mając wygodny, dobrze zaprojektowany stadion, angażując radio i telewizję do transmisji Twoich rozgrywek na antenie ogólnonarodowej itd. Po trzecie, minimalizując pensje i inne wydatki. Po czwarte, opracowując najkorzystniejszy system marketingu związany z gadżetami (czapkami, swetrami, kartami kolekcjonerskimi itp.) i prawami do transmisji meczy. Po piąte, zapewniając dobre relacje społeczne: chcesz unikać historii o Twoich graczech popadających w konflikt z prawem. Po szóste, minimalizując stopień rywalizowania na tym samym rynku z innymi drużynami sportowymi, przy czym mogą to być drużyny w innych konkurencjach (np. drużyna koszykarska Chicago Bulls konkujuje z drużyną piłkarską Chicago Bears oraz z drużynami baseballowymi Chicago Cubs i White Sox).

Podchodząc do tego bardziej formalnie, chciałbyś przede wszystkim określić swoje cele tak, żeby były możliwie niezależne: maksymalizacja rentowności, zapewnienie dobrych relacji z klientami, zapewnienie dobrych relacji społecznych i minimalizacja konfliktów z innymi właścicielami drużyn w okolicy. Każdy z tych parametrów możemy podzielić na drobniejsze sprawy. Rentowność wiąże się z przyciąganiem fanów, odpowiednim ustawnianiem cen biletów na podstawie danych demograficznych fanów, marketingiem gadżetów, kosztem pensji członków organizacji itd. Założmy, że możemy podać jawnie poziom realizacji dla każdego z parametrów, a także ich znaczenie.

Żaden z tych parametrów nie uwzględnia relacji, jakie masz ze swoimi konkurentami: innymi właścicielami drużyn sportowych grających w tę samą grę. Jeśli miałbyś zmaksymalizować funkcję opracowaną na podstawie dotychczasowych wysiłków, to uzyskałbyś bez wątpienia kiepski wynik, gdyż w rze-

czystości istnieją inni właściciele drużyn, którzy próbują również osiągnąć sukces, a ich sukces może przyjść tylko w wyniku Twojej porażki!

Przypuśćmy, że reprezentujesz małe miasto bez wielkich stacji telewizyjnych, na przykład San Diego w Kalifornii lub Charlotte w Karolinie Północnej. Jesteś w sytuacji, w której właściciele drużyn w miastach o dużych rynkach, na przykład Nowy Jork, Chicago i Los Angeles, mają więcej pieniędzy, które mogą przeznaczyć na lepszych zawodników, więcej ludzi w okolicy, którymi mogą zapełnić swoje stadiony, i dłużej wpływające pieniądze z praw telewizyjnych, dzięki czemu mogą utrzymywać, a nawet zwiększać ilość pieniędzy, jaką poświęcają co roku na graczy. Dla kontrastu Ty masz tylko tyle pieniędzy, żeby przyciągnąć kilku najlepszych graczy, i musisz szukać i handlować, żeby pozyskać resztę potrzebnych graczy. Wygląda na to, że jest Ci przeznaczona klęska, gdyż wszystkie okoliczności są przeciwko Tobie.

Zamiast godzić się z tą sytuacją, możesz zdecydować się na utworzenie koalicji z innymi właścicielami zespołów o małych rynkach i uzyskać większą siłę rynkową w stosunku do właścicieli o dużych rynkach. Możesz w ten sposób zmusić ich do dzielenia się większą częścią przychodów (w końcu bez tych wszystkich zespołów o małych rynkach nie mieliby z kim grać). Możesz chcieć teraz robić interesy z właścicielami innych zespołów o małych rynkach i bojkotować współpracę z „grubymi rybami”. Możesz stwierdzić, że jesteś mniej zainteresowany pokonywaniem jednych zespołów, a bardziej innych. Możesz zaproponować „czapę płacową”, która nałoży ograniczenie na ilość pieniędzy, które każdy z zespołów może przeznaczyć na graczy. Właściciele dużych rynków będą walczyć z takim ruchem. W zależności od wyniku tych politycznych zmagań dostępne Ci zasoby mogą się zmieniać, wprowadzając zupełnie nowe ograniczenia dotyczące Twojej działalności i wyznaczając całkowicie odmienne rozwiązania Twoich problemów.

Zauważ, że Twoje kontakty nie ograniczają się tylko do kontaktów z właścicielami. W czasie gdy walczysz i współpracujesz z różnymi osobami z tego kręgu, musisz też negocjować ze związkami graczy, które starają się maksymalizować ich przychody, wygody związane z zakwaterowaniem na wyjazdach, emerytury i inne udogodnienia. Musisz jeszcze negocjować z władzami miasta, które mogą chcieć wykorzystywać stadion na potrzeby innych wydarzeń, chyba że Ty sam jesteś właścicielem stadionu.

Za każdym razem przy nawiązywaniu kontaktów musisz brać pod uwagę łączny cel: nie tylko to, co Ty sam starasz się osiągnąć, ale także to, co inni chcą uzyskać, oraz to, czy sam zgadzasz się z nimi, czy też jesteś im przeciwny. Oznacza to, że za każdym razem, gdy rozpoznasz nowego „gracza w tej grze”, musisz ponownie rozważyć przydział swoich zasobów i stwierdzić, czy Twoje poprzednie *rozwiązanie* jest w dalszym ciągu aktualne po dojściu nowego elementu.

Klasyczny przykład sytuacji, w której mamy do czynienia ze zmiennym zestawem przeciwników, pojawia się w słynnej grze zwanej iterowanym dylematem więźnia. Gra modeluje następującą sytuację. Przypuśćmy, że Ty i Twój partner popełniliście przestępstwo. Pojmano Was i zamknięto w oddzielnych celach. Do Twojej celi przychodzi prokurator i mówi, że Wam obu przedsta-

wia taką samą propozycję: jeśli Ty i Twój partner przyznacie się do winy, to prokurator wsadzi Was na cztery lata do więzienia. Jeśli jednak przyznasz się do winy i zgodzisz się złożyć zeznanie przeciwko partnerowi, a ten odmówi pomocy prokuratorowi, to prokurator uwolni Cię, a Twój partner dostanie pięć lat. Jeśli jednak obaj wyprzecie się odpowiedzialności, to prokurator i tak ma na tyle silne dowody, żeby skazać Cię na dwa lata. Zyski (w tym wypadku to raczej kary) zapisane w postaci gry matematycznej wyglądają następująco:

	Zeznawanie	Milczenie
Zeznawanie	(4, 4)	(0, 5)
Milczenie	(5, 0)	(2, 2)

przy czym Ty wybierasz wiersz, a Twój partner kolumnę. Kłopot polega na tym, że przed podjęciem decyzji nie będziesz się mógł skontaktować z partnerem.

Patrząc na tabelkę, oceniasz, że niezależnie od tego, co zrobi partner, lepiej jest przyznać się do winy i „wydać” kolegę. Jeśli będzie on siedział cicho, to Ty możesz też siedzieć cicho i dostaniesz wtedy dwa lata albo możesz go wydać i wydostać się na wolność! Jeśli partner Cię wyda, to możesz siedzieć cicho i trafić za kratki na pięć lat lub wydać go także i trafić do więzienia na cztery lata. Niezależnie od wyboru partnera przyznanie się do winy i pomoc prokuratorowi wydaje się tym, co jest słuszne. Rzecz w tym, że takie samo rozumowanie przeprowadza także Twój partner. Podejmuje on taką samą decyzję jak Ty. W wyniku tego logicznego wyjścia dla Was obu jest zdrada, dzięki czemu obaj dostajecie czteroletnie wyroki, podczas gdy moglibyście zmniejszyć czas spędzony w więzieniu do dwóch lat, gdybyście siedzieli cicho. To prawdziwy dylemat!

Możemy ten problem odwrócić i mówić o nagrodach zamiast kar. Przypuśćmy, że masz dwa wyjścia: *współpracować* lub *zdradzić*. Współpraca oznacza pracę nad zwiększeniem Twojej nagrody, a także nagrody drugiego gracza. Zdrada z kolei implikuje zwiększenie własnej nagrody kosztem przeciwnika. W tej sytuacji zestawienie zysków mogłoby być następujące:

	Współpraca	Zdrada
Współpraca	(3, 3)	(0, 5)
Zdrada	(5, 0)	(1, 1)

I znowu mamy ten sam dylemat. Zdrada wydaje się lepsza od współpracy, ale współpraca opłaca się trzy razy bardziej niż wzajemna zdrada.

Tego rodzaju gra może być wykorzystywana do modelowania wielu sytuacji ze świata rzeczywistego. Na przykład kraje biorące udział w OPEC grają w dylemat więźnia. Każdy z krajów zgadza się na ograniczenie swojego wydobycia ropy, dzięki czemu jej cena rośnie, ale każdy z tych krajów może naruszyć porozumienie i zarobić więcej pieniędzy, nie robiąc tego. Dwie główne różnice między modelem państw OPEC a powyższą sytuacją z więźniami polegają na tym, że w OPEC jest więcej graczy, i co ważniejsze, muszą grać w tę grę nieprzerwanie. Gracze mają tutaj pamięć. Pamiętają, kto zawiódł poprzednim razem, i będą chcieli brać to pod uwagę przy następnym spotkaniu. Podobnie pamiętają również na szczęście, kto współpracował.

Gdy dylemat więźnia jest rozgrywany wielokrotnie, okazuje się, że współpraca staje się racjonalną strategią. Jedna z bardziej niezawodnych strategii w tej grze jest nazywana wet za wet: najpierw współpracujemy, a potem robimy to, co reszta graczy. Jeśli oni oszukują, Ty też oszukujesz. Jeśli współpracują, współpracujesz. W ten sposób nigdy nie odniesiesz korzyści przez więcej niż jedną rundę. Najlepsza strategia zależy od innych graczy, a oni nierzadko w świecie rzeczywistym stale się zmieniają. Nie ma sensu współpraca z partnerem, który zawsze zdradza. Z partnerem, który gra wet za wet, należy współpracować. Partnera, który zawsze współpracuje, należy oszukiwać, gdyż jest naiwniakiem. W świecie rzeczywistym często następuje swego rodzaju selekcja, w której naiwniacy i stale zdradzający są na różne sposoby eliminowani. W wyniku tego musisz na podstawie swojej wiedzy na temat pozostałych graczy w sposób ciągły uaktualniać swoją strategię.

Przedostatni już przykład pochodzi z dziedziny walk zbrojnych. Każdy nowy obiekt musi zostać zidentyfikowany albo jako przyjaciel, albo jako wróg, albo jako element neutralny. Każda potyczka zmienia się zgodnie z funkcją łącznego celu: Twojego celu powiązanego zależnościami z celami innych uczestników. Najlepsza linia postępowania zmienia się zgodnie z funkcją przewidywanych reakcji wroga oraz spodziewanych wyników zdarzeń. Planowanie misji bojowej wymaga stałego dostosowywania się. Wyrażenie „przystosuj się lub gin” nigdzie nie może być bardziej odpowiednie.

W pewnym sensie problemy świata rzeczywistego nie są nigdy rozwiązywane. Każde rozwiązanie prowadzi do następnego problemu. Rzecz w tym, żeby przewidzieć ten problem i zaadaptować swoje rozwiązanie tak, żeby jak najlepiej poradzić sobie z możliwościami, przed jakimi się stanie. Jeśli możesz uwzględnić przewidywane sytuacje w swojej funkcji oceny, to, być może, odkryjesz niezawodne rozwiązanie, które będzie radzić sobie z wyobrażonymi okolicznościami. Jeśli nie możesz przewidzieć końcowych wyników z wystarczającą dokładnością, to prowadzenie ewolucji zróżnicowanej populacji wartościowych rozwiązań może zapewnić skuteczne zabezpieczenie przed dynamicznie zmieniającym się środowiskiem. Może to też złagodzić nieuchronny wpływ szumu, który zmniejsza naszą zdolność do prawidłowej oceny sytuacji i określenia efektywności obranej strategii.

Po wprowadzeniu dylematu więźnia trudno się oprzeć pokusie i nie powiedzieć o innej wersji tego problemu, która jest również znana pod tą samą nazwą. Problem ten jednak nie należy do najłatwiejszych, może więc powinniśmy byli silniej się opierać. W każdym razie opis jego brzmi tak:

W celi znajdują się trzej więźniowie. Wiedzą oni, że dwóch z nich zostanie rano straconych, a jeden zostanie uwolniony, nie wiedzą jednak który. Decyzja została już podjęta i zarządzający więzieniem już ją znają. Jak to czasem bywa, jeden z więźniów jest statystykiem. Oblicza prawdopodobieństwo, że przeżyje i wychodzi mu $1/3$. Wicezorem, który jest ostatnim wieczorem dla dwóch spośród trzech więźniów, przychodzi do celi jeden strażnik i przynosi więźniom posiłek. Nasz bohater, statystyk, mówi do strażnika:

– Popatrz, skoro dwóch z nas zostanie rano straconych, to wiem, że co najmniej jeden z moich towarzyszy zostanie zabity. Zatem nic się nie stanie, jeśli powiesz mi, który z nich umrze; w każdym razie nie pytam się przecież o siebie!

Strażnik myśli przez chwilę i odpowiada:

- W porządku, powiem Ci. Umrze on – i wskazuje na jednego z mężczyzn w celi.
- Wspaniale! – wykrzykuje statystyk. – Moje szanse na przeżycie wzrosły do 1/2!

Tu pojawia się pytanie do czytelnika: czy statystyk popełnił błąd w swoim rozumowaniu?

Wydaje się, że w związku ze zdobytą wiedzą, jego szanse na przeżycie rzeczywiście wzrosły do 1/2, gdyż albo zabity zostanie on, albo drugi więzień. Pomyślmy o tym w ten sposób. Urzędnicy więzieni przeprowadzają eksperyment statystyczny. Daje on trzy wyniki:

	A	B	S
1	E	E	U
2	U	E	E
3	E	U	E

przy czym: S oznacza statystyka, A i B – pozostałych mężczyzn, E – egzekucję, U – uwolnienie. Jeśli założymy tak jak statystyk, że każdy z tych wyników jest jednakowo prawdopodobny, to gdy strażnik wskaże na jednego z pozostałych ludzi, przestrzeń próbek ulega zmniejszeniu. Jeśli strażnik wskaże na A, to wiemy, że możliwe są wyniki 1 lub 3. Jeden z nich oznacza, że statystyk zostanie uwolniony, drugi zaś, że zostanie na nim wykonana egzekucja. Ponieważ wszystkie te możliwości mają równe szanse zajścia, statystyk może myśleć, że jego szanse przeżycia są teraz 1 do 2. To samo rozumowanie mógłby zastosować, gdyby strażnik wybrał B zamiast A.

Możemy jednak o tym problemie myśleć inaczej. Przestrzeń wyników eksperymentu nie jest zdefiniowana w kategoriach trzech powyższych wyników, ale raczej w kategoriach tego, na kogo wskaże strażnik. Ponieważ wiemy, że strażnik nie wskaże na S, są tylko dwie możliwości: wskaże na A lub B. Tutaj jednak wyniki mają inne implikacje. Wiemy, że jeśli A i S umrą, to strażnik wskaże na A. Stanie się to z prawdopodobieństwem 1/3. Podobnie, jeśli B i S umrą, to strażnik wskaże na B i stanie się to z prawdopodobieństwem 1/3. Wreszcie jeśli A i B będą straceni, to możemy założyć, że strażnik z równym prawdopodobieństwem wskaże pierwszego lub drugiego mężczyznę. W związku z tym, że śmierć A i B zachodzi z prawdopodobieństwem 1/3, to szanse na wskazanie A (jak i B) są połową tego, czyli wynoszą 1/6.

Załóżmy teraz, że strażnik wskazał na B. Zmniejszyliśmy teraz przestrzeń próbek do możliwej śmierci B i S oraz A i B. Jednak tu jest pies pogrzebany, wyniki są określone w kategoriach wskazania palcem przez strażnika! Zatem wynikiem jest to, że „B i S umrą i strażnik wskazuje na B”, oraz to, że „A i B

umrą i strażnik wskazuje na B". Prawdopodobieństwo pierwszego zdarzenia było $1/3$, a prawdopodobieństwo drugiego, które zadowala statystyka, to $1/6$. Zatem prawdopodobieństwo zdarzenia, w którym statystyk przeżywa, przy założeniu, że strażnik wskazał na B, wynosi

$$\frac{\frac{1}{6}}{\frac{1}{3} + \frac{1}{6}} = \frac{1}{3}$$

Takie samo rozumowanie sprawdza się, gdy strażnik wskazuje na A, a nie na B.

Które rozwiązanie jest dobre? Może problem ten powinien nazywać się dylematem statystyka! Czy możesz stwierdzić, gdzie leży racja? Spróbuj napisać symulację komputerową, która modeluje tę sytuację i zobacz, co się stanie!

XII. Dzień tygodnia, w którym wypada pierwszy stycznia

Jaki dzień tygodnia wypada częściej pierwszego dnia roku: sobota czy niedziela?

Większość ludzi odpowiedziałaby: żaden. Intuicja bowiem podpowiada nam, że na dłuższą metę częstość wypadania pierwszego dnia stycznia w sobotę bądź w niedzielę jest taka sama. A jednak tak nie jest.

Zanim zagłębimy się w ten problem, przypomnijmy sobie definicję roku przestępnego. Dany rok jest przestępny, jeśli dzieli się przez 4, ale nie dzieli się przez 100 (chyba że dzieli się przez 400). Na przykład: lata 1892 i 1896 były przestępne, rok 1900 – nie, lata 1904, 1908 itd. były przestępne, podobnie jak rok 2000.

Tak więc w ciągu czterystu lat mamy dokładnie 97 lat przestępnych (tzn. wszystkie lata podzielne przez 4, a jest ich 100 – poza tymi, które dzielą się przez 100, ale nie dzielą się przez 400). W ciągu dowolnego okresu czterystu lat mamy dokładnie

$$97 \cdot 366 + 303 \cdot 365 = 146\,097 \text{ dni}$$

Liczba 146 096 jest podzielna przez 7. Oznacza to, że dowolny okres czterystu lat składa się z całkowitej liczby pełnych tygodni. Oznacza to również, że 400 lat stanowi pewien cykl: jeśli pierwszy stycznia 2001 wypada w poniedziałek, to pierwszy stycznia 2401 niewątpliwie także jest poniedziałkiem.

Ponieważ 400 nie jest podzielne przez 7, poszczególne dni tygodnia występują z różną częstością. W trakcie 400 lat niemożliwa jest taka sama liczba poniedziałków, wtorków, śród itd. Wystarczy więc stwierdzić, który z dni tygodnia – sobota czy niedziela – wypada częściej pierwszego stycznia w ciągu dowolnego 400-letniego okresu.

Ponieważ możemy wybrać dowolne czterysta lat, skoncentrujmy się na okresie 2001 – 2400. W tym czasie mamy cztery zwykłe okresy, w których rok przestępny pojawia się dokładnie co 4 lata. Jedynymi wyjątkami są tu lata 2100, 2200 i 2300, które *nie* są przestępne. Zauważ też, że w dowolnym okresie 28 lat w obrębie tych regularnych przedziałów (tzn. w dowolnym okresie 28 lat, który nie obejmuje lat 2100, 2200 i 2300) każdy z siedmiu dni tygodnia wypada czterokrotnie pierwszego stycznia. Oznacza to, że na przykład w okresie od 1 stycznia 2034 do 31 grudnia 2061 pierwszy dzień roku wypadł dokładnie cztery razy w poniedziałek, cztery razy we wtorek, cztery razy w środę itd. Tymi okresami nie musimy się więc zajmować. Każdy z nich zawierał po siedem lat przestępnych, czyli miał

$$7 \cdot 366 + 21 \cdot 365 = 10\ 227 \text{ dni}$$

a liczba ta jest podzielna przez 7.

Rozważmy więc następujący zbiór okresów, obejmujący w sumie 400 lat:

- | | | | |
|-----|-----------|-----|-----------|
| (a) | 2001–2028 | (i) | 2201–2228 |
| (b) | 2029–2056 | (j) | 2229–2256 |
| (c) | 2057–2084 | (k) | 2257–2284 |
| (d) | 2085–2100 | (l) | 2285–2300 |
| (e) | 2101–2128 | (m) | 2301–2328 |
| (f) | 2129–2156 | (n) | 2329–2356 |
| (g) | 2157–2184 | (o) | 2357–2384 |
| (h) | 2185–2200 | (p) | 2385–2400 |

Wszystkie dni tygodnia występują pierwszego dnia roku czterokrotnie w następujących okresach: (a), (b), (c), (e), (f), (g), (i), (j), (k), (m), (n) i (o), a więc obliczeń wymagają tylko pozostałe okresy: (d), (h), (l) i (p). Musimy sprawdzić (np. za pomocą uniwersalnego kalendarza), że:

1 stycznia 2085 wypada w poniedziałek

1 stycznia 2185 wypada w sobotę

1 stycznia 2285 wypada w czwartek

1 stycznia 2385 wypada we wtorek

Mając te dane, z łatwością obliczymy liczbę występowania poszczególnych dni tygodnia w tych czterech okresach:¹

¹Ostatni wiersz tabeli podaje dzień tygodnia wypadający odpowiednio 1 stycznia 2100, 2200, 2300 i 2400 roku.

	20–	21–	22–	23–
–85	pon.	sob.	czw.	wt.
–86	wt.	niedz.	pt.	śr.
–87	śr.	pon.	sob.	czw.
–88	czw.	wt.	niedz.	pt.
–89	sob.	czw.	wt.	niedz.
–90	niedz.	pt.	śr.	pon.
–91	pon.	sob.	czw.	wt.
–92	wt.	niedz.	pt.	śr.
–93	czw.	wt.	niedz.	pt.
–94	pt.	śr.	pon.	sob.
–95	sob.	czw.	wt.	niedz.
–96	niedz.	pt.	śr.	pon.
–97	wt.	niedz.	pt.	śr.
–98	śr.	pon.	sob.	czw.
–99	czw.	wt.	niedz.	pt.
–00	pt.	śr.	pon.	sob.

Obliczenie dla okresów (d), (h), (l) i (p) przedstawia się więc następująco:

poniedziałek	8
wtorek	10
środa	9
czwartek	9
piątek	10
sobota	8
niedziela	10

a całkowita liczba wystąpień poszczególnych dni tygodnia pierwszego dnia roku przez 400 lat wynosi:

poniedziałek	56
wtorek	58
środa	57
czwartek	57
piątek	58
sobota	56
niedziela	58

Tak więc prawdopodobieństwo, że dowolny 1 stycznia wypadnie w niedzielę wynosi 0,145 (tzn. 58/400), podczas gdy prawdopodobieństwo dla soboty wynosi jedynie 0,14 (tzn. 56/400).

Jak możemy skorzystać z tej wiedzy? To jest dopiero rzeczywisty problem!

12. Sieci neuronowe

Gdybym miał rozum.

Słowa Stracha na Wróble z *Czarnoksiężnika z krainy Oz*
Lymana Franka Bauma¹

Po spędzeniu znacznej ilości czasu na próbach rozwiązania jakiegoś problemu, „łamiąc sobie głowę”, nieuchronnie wpadasz na pomysł, żeby zautomatyzować proces myślowy. Marzyłoby się stworzenie symulacji komputerowej, która naśladowałaby funkcjonowanie mózgu tak, żeby mogła rozwiązywać problemy w taki sposób, jak robisz to Ty. Mózg nie działa jednak tak jak szeregowe maszyny cyfrowe. Przetwarzanie biologiczne jest ze swojej natury silnie równoległe, natomiast tradycyjne obliczenia są sekwencyjne. Każdy krok algorytmu jest wykonywany po kolej, aż zostanie osiągnięty warunek zatrzymania. Co więcej, chociaż istnieją pojęciowe podobieństwa między neuronami w ludzkich mózgach a bramkami logicznymi w komputerach, szybkość zmiany stanu neuronów biologicznych jest o wiele mniejsza niż szybkość zmiany stanu bramek logicznych (dla neuronów są to czasy rzędu milisekund, dla bramek – nanosekund). Te i inne różnice w budowie wspierają intuicyjne przekonanie, że te alternatywne urządzenia wejścia-wyjścia powinny być zdolne do efektywnego radzenia sobie z różnymi problemami.

Komputery doskonale się sprawują jako urządzenia do szybkiego określania wyników operacji arytmetycznych. Jeśli chcesz wiedzieć, ile jest $412,14823$ pomnożone przez $519,442$, to zdasz się raczej na ręczny kalkulator niż na chętnego kolegę wyposażonego tylko w papier i ołówek. Dla kontrastu, komputery nie sprawują się najlepiej przy uogólnianiu, czyli obsługując sytuacji wychodzących poza wyznaczony zakres możliwości. Jeśli któryś z naszych przyjaciół zgoli brodę, to nadal możemy go rozpoznać. Z kolei komputer, działający na podstawie ciągu warunków if-then odpowiadających rozpoznawaniu cech charakterystycznych twarzy tej osoby, może mieć z tym dużo większy problem.

¹Przekład Stefania Wortman, Siedmioróg, Wrocław 1998 (przyp. tłum.).

Czy zawsze tak musi być? Czy jest to podstawowe ograniczenie przetwarzania komputerowego? A może jest możliwe opracowanie komputerów, które działają jak biologiczne sieci neuronowe? W każdym razie sieć neuronowa jest urządzeniem wejścia-wyjścia, takim jak automat ze skończoną liczbą stanów lub jakaś inna reprezentacja komputera. Powinno być możliwe stworzenie modeli opisujących, jak sieci neuronowe wykonują działania związane z wejściem-wyjściem i jak te działania są realizowane na komputerze. Powstała tak sztuczna sieć neuronowa (ang. *artificial neural network* – ANN) może wykazywać niektóre zdolności przetwarzania właściwe ludzkim mózgom, oferując jednocześnie szybkość obliczeń właściwą strukturom krzemowym.

12.1. Neurony progowe i liniowe funkcje dyskryminacyjne

Próby opisania sieci neuronowych sięgają wiele lat wstecz, wręcz do czasów sprzed powstania współczesnego komputera cyfrowego. W 1943 roku Warren McCulloch i Walter Pitts [303] podali słynny model aktywności neuronów. Był to prosty, ale ważny początkowy krok prowadzący do uogólnienia opisu działania dotyczącego wejścia-wyjścia pojedynczych neuronów. Ich model był oparty na połączonej z pewnym programem zasadzie wszystko albo nic. Jeśli będzie odpowiednio wiele aktywnych wejść do neuronu (jeśli „poziom aktywności”, określany jako β , będzie odpowiednio wysoki), to neuron odpowie, wysyłając (odpalając, ang. *firing*) sygnał; w przeciwnym razie pozostanie nieaktywny. Zapisując to matematycznie, jeśli $f(\cdot)$ jest funkcją opisującą działanie neuronu, to

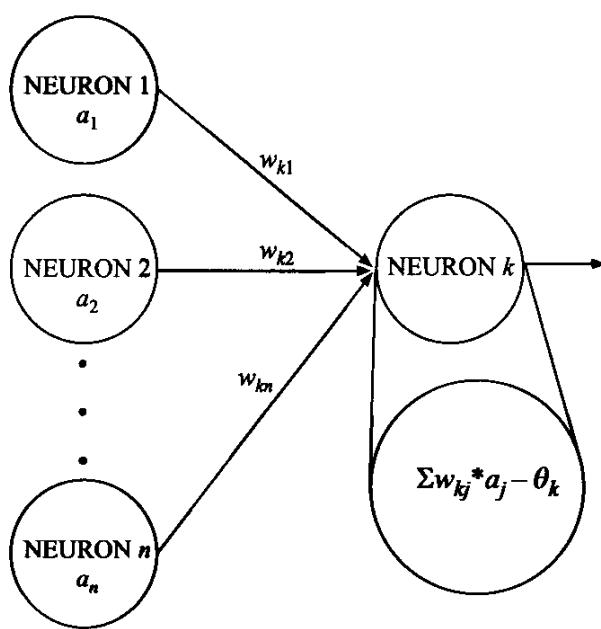
$$f(\beta) = \begin{cases} 1 & \text{jeśli } \beta \geq 0 \\ 0 & \text{jeśli } \beta < 0 \end{cases}$$

Neuron uruchamia się, wykazując aktywność wyjścia 1; w przeciwnym razie pozostaje w spoczynku i aktywność wyjścia jest 0.

Poziom aktywności wejściowej neuronu może być opisany jako suma ważona aktywności wyjściowych przekazanych z innych neuronów (zaproponował to Rosenblatt [396] w modelu zwanym perceptronem). W istocie każdy neuron jest połączony z innymi neuronami za pomocą „synaps”, które mogą wzmacnić lub osłabić przesyłany sygnał. Sytuację tę pokazano na rys. 12.1. Możemy ją opisać za pomocą symboli:

$$\beta_k = \sum_{j=1}^n (w_{kj} a_j) - \theta_k$$

przy czym: β_k oznacza aktywność k -tego neuronu, w_{kj} – opatrzone wagą połączenie od neuronu j do neuronu k , a_j – aktywność wyjściową j -tego neuronu przekazywaną do neuronu k , θ_k – próg dla neuronu k , n – liczbę neuronów komunikujących się z neuronem k . Zauważ, że θ_k można zarówno dodawać, jak i odejmować od sumy, po prostu zmieniając jego znak.



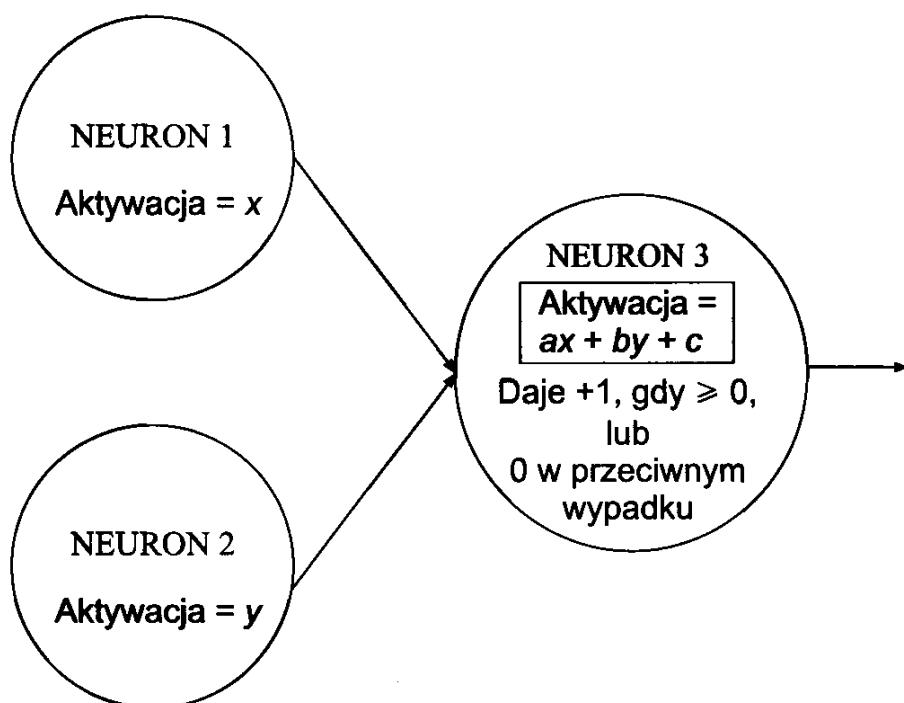
Rysunek 12.1. Prosta sieć neuronowa, w której n neuronów wejściowych łączy się z jednym neuronem wyjściowym k . Między każdym neuronem wejściowym a neuronem wyjściowym jest poprowadzone połączenie opatrzone wagą. Neuron wyjściowy oblicza funkcję $\sum_{j=1}^n w_{kj} \cdot a_j - \theta_k$, a następnie porównuje wynik z progiem (np. równym zero). Sieć uruchamia się, jeśli wartość wyjściowa jest nie mniejsza od ustawionego progu

Łatwo teraz można przewidzieć sytuację, w której setki, jeśli nie miliony lub wręcz miliardy takich neuronów są ze sobą połączone, przetwarzają informacje z otoczenia i wysyłają sygnały, które mogą sterować różnymi aspektami funkcjonowania żywego organizmu. To znaczy łatwo można przewidzieć lub przynajmniej wyobrazić sobie, że potencjalnie leży to w zakresie możliwości neuronów, ale w rzeczywistości określenie własności obliczeniowych takich połączonych neuronów matematycznych wymaga analizy, a nie hipotezy.

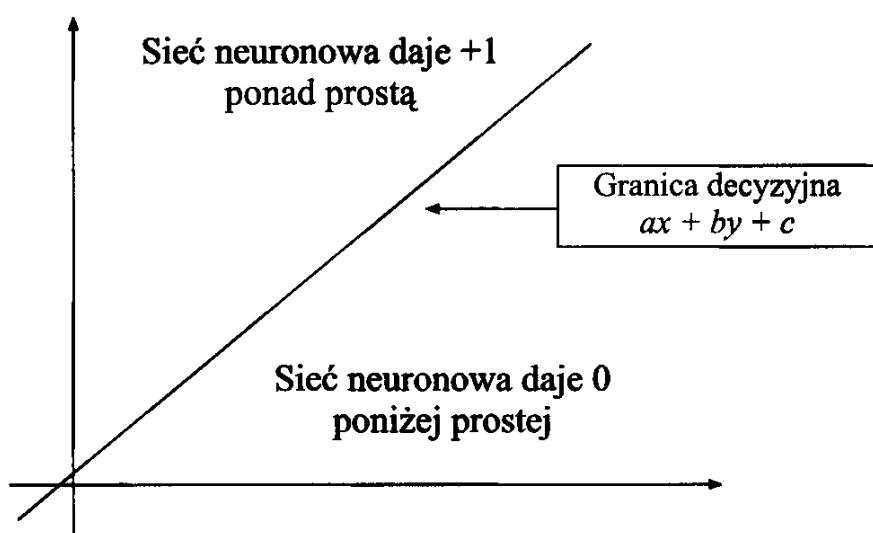
Aby zrozumieć działanie sieci złożonej z takich neuronów, dobrze jest na początku przyjrzeć się działaniu jednego neuronu. Tutaj sprawa jest dość prosta. Neuron to liniowa kombinacja opatrzonych wagami sygnałów wejściowych przesunięta o wartość progu. Jeśli wynikowy poziom aktywności jest większy lub równy零, to neuron się uruchamia. Uprośćmy jeszcze nasze rozważania i przyjrzyjmy się przypadkowi, gdy do interesującego nas neuronu są przyłączone tylko dwa neurony wejściowe (rys. 12.2). Wynik działania pierwszego neuronu wejściowego to x ; wartość ta jest opatrzona wagą a . Wynik działania drugiego neuronu wejściowego to y ; tu mamy wagę b . Wreszcie wartość progu w neuronie wynikowym wynosi c . W ten sposób otrzymujemy:

Jeśli $(ax + by + c \geq 0)$, to (neuron 3 uruchamia się i daje wynik 1).

Szybko przypominamy sobie ze szkoły, że $ax + by + c = 0$ jest równaniem prostej. Zatem to, co w zasadzie robi trzeci neuron, polega na uruchomieniu się, jeśli wartości przychodzące z poprzedzających neuronów powodują, że wynik trafia na jedną ze stron prostej zdefiniowanej przez a , b i c (rys. 12.3). Tak działa



Rysunek 12.2. Sieć neuronowa z dwoma węzłami wejściowymi, które mają odpowiednio poziomy aktywacji x i y . Opatrzone wagami połączenia z węzłem wyjściowym to odpowiednio a i b . Wynik działania sieci jest otrzymywany przez obliczenie $ax + by + c$ i przyównanie tego do zera. Sieć daje wynik +1, gdy $ax + by + c \geq 0$, oraz wynik 0 w przeciwnym wypadku

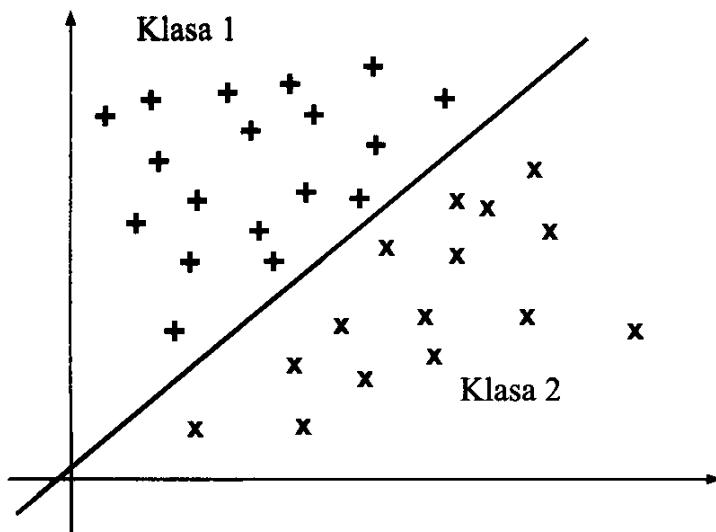


Rysunek 12.3. Prosta sieć neuronowa z rys. 12.2 definiuje prostą, która dzieli płaszczyznę xy . Prosta $ax + by + c$ wyznacza granicę decyzyjną. Jeśli sieci zostanie podany punkt (x, y) ponad tą prostą, to sieć daje wynik +1. W przeciwnym razie daje 0 (mówi się też równoważnie, że nie daje wyniku)

podstawowy system rozpoznawania wzorców: liniowa funkcja dyskryminacyjna (rozróżniająca).

Liniowe funkcje dyskryminacyjne, jak nazwa wskazuje, mogą rozróżnić dwie klasy danych rozgraniczone hiperpłaszczyzną. W dwóch wymiarach hiperpłaszczyzna jest zwykłą prostą, w trzech – standardową płaszczyzną. Funkcje

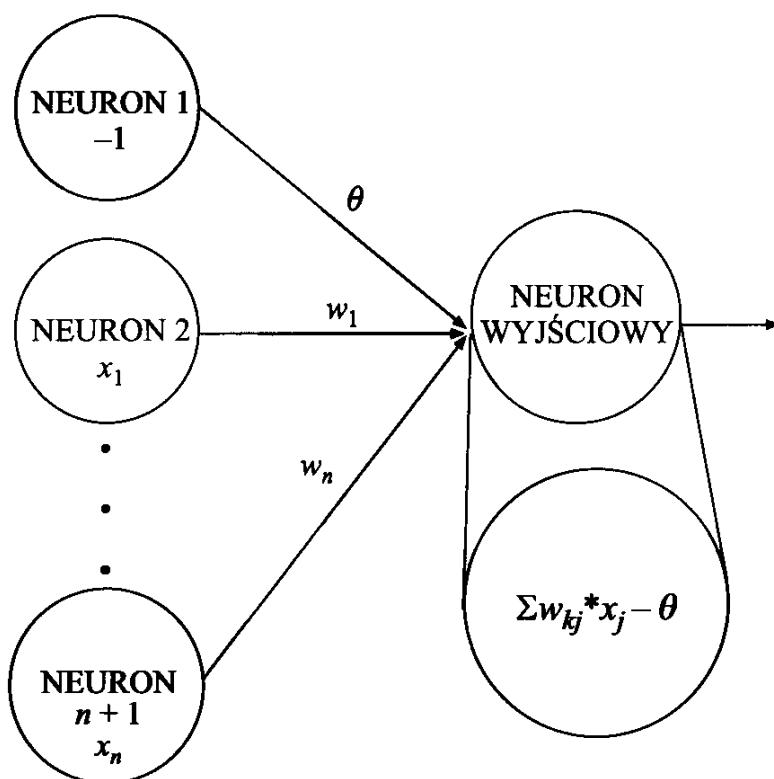
te są często wygodne do rozdzielania klas danych z różnych rozkładów. Rozważmy, na przykład, dane z rys. 12.4. Chcemy znaleźć prostą regułę pozwalającą na rozpoznanie, czy nowy punkt należy do klasy 1, czy do klasy 2. Jeśli dane możemy łatwo rozdzielić prostą, to ta prosta zapewnia regułę umożliwiającą określenie rozróżnienia. Jeśli nowy punkt wypada ponad prostą, to wyrażenie $ax + by + c$ da wartość większą od zera i reguła zostanie spełniona – równoważnie, nasz neuron da w wyniku 1, czyli uruchomi się. Podobnie, odwrotne działanie, przy którym zostanie podane 0, wystąpi, gdy nowy punkt znajdzie się poniżej prostej. W tym momencie pojawia się pytanie, jak określić odpowiednie wartości parametrów a , b i c , żeby w najlepszy sposób oddzielić pewne dane za pomocą liniowej funkcji dyskryminacyjnej.



Rysunek 12.4. Na rysunku *plusy* oznaczają elementy klasy 1, a *krzyżyki* elementy klasy 2. Wszystkie *plusy* można oddzielić od *krzyżyków* za pomocą granicy będącej linią prostą, co oznacza, że sieć neuronowa z rys. 12.2 może prawidłowo poklasyfikować

Żeby uogólnić całą dyskusję, zastanówmy się nad sytuacją z rys. 12.5. Neuron ma tam n wejść, które są reprezentowane jako wektor \mathbf{x} . Jak zwykle w pracach przeglądowych, możemy jeszcze dołożyć tutaj składnik progowy neuronu, przedstawiając go jako jedno z wejść stałe ustawione na -1 (lub równoważnie na $+1$). W ten sposób otrzymujemy $(n+1)$ -wejście. Wagi wejść (wraz z wagą dla progu) są reprezentowane za pomocą wektora $\mathbf{w} = [\theta, w_1, \dots, w_n]$, przy czym θ jest wagą związaną z wartością progową, a pozostałe wagi wskazują współczynniki wzmacniania dla odpowiednich połączeń wejściowych z wartościami w \mathbf{x} .

Najlepszy zbiór wag \mathbf{w} powinien dawać $\mathbf{w}^T \mathbf{x} \geq 0$ dla $\mathbf{x} \in$ klasa 1 oraz $\mathbf{w}^T \mathbf{x} < 0$ dla $\mathbf{x} \in$ klasa 2. Neuron powinien zareagować za każdym razem, gdy otrzyma punkt z klasy 1, i nie reagować, gdy otrzyma punkt z klasy 2. Założymy, że mamy p wzorców wejściowych $\mathbf{x}_1, \dots, \mathbf{x}_p$. Prosty algorytm ([210], str. 109) umożliwiający uzyskanie opisanego tu działania jest następujący:



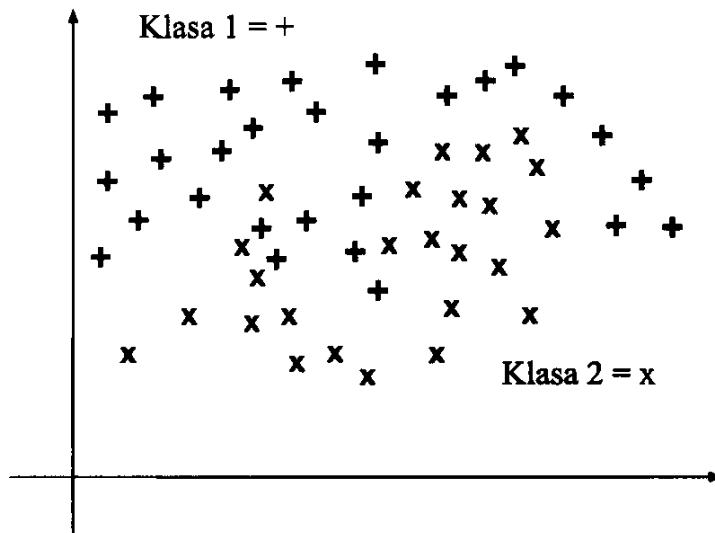
Rysunek 12.5. Inny sposób przedstawiania sieci neuronowej, w którym składnik progowy jest uwzględniony za pomocą węzła wejściowego o wartości aktywacji -1 i wadze połączenia θ

1. Dla każdego po kolei wektora \mathbf{x}_i , jeśli bieżący wektor wag \mathbf{w} powoduje prawidłowe zaklasyfikowanie \mathbf{x}_i , to nie zmieniaj \mathbf{w} .
2. W przeciwnym razie uaktualnij \mathbf{w} , korzystając z reguły:
 - (a) $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{x}_i$, jeśli $\mathbf{w}^T \mathbf{x}_i \geq 0$ oraz $\mathbf{x}_i \in$ klasa 2
 - (b) $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i$, jeśli $\mathbf{w}^T \mathbf{x}_i < 0$ oraz $\mathbf{x}_i \in$ klasa 1

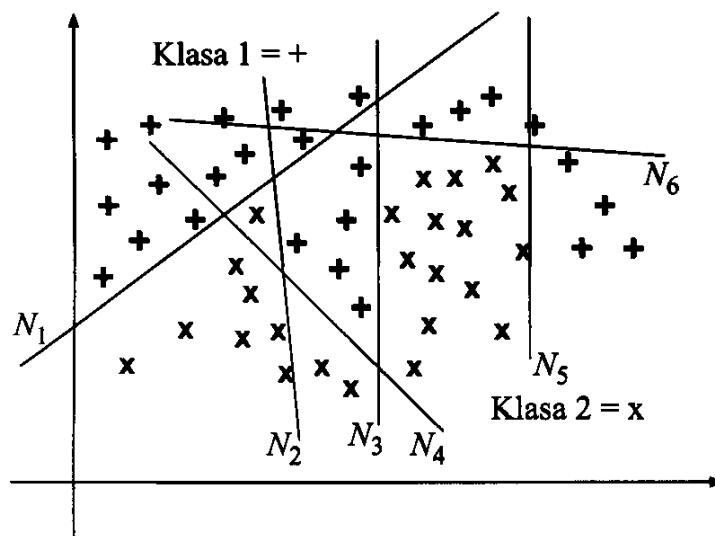
przy czym η jest stałym większym od zera parametrem określającym wielkość kroku.

Można wykazać, że ta prosta reguła – uruchomiona przy warunku początkowym $\mathbf{w} = \mathbf{0}$ – umożliwia rozdzielenie każdych liniowo separowalnych danych po pewnej liczbie s jej zastosowań.

Przypuśćmy, że jesteśmy zainteresowani rozdzieleniem danych należących do dwóch klas, które nie są liniowo separowalne, jak to pokazano na rys. 12.6. Nasz jeden neuron progowy oraz powyższy algorytm poprawiania wag nie są wystarczające. Potrzebujemy większej liczby neuronów, aby móc zorganizować hierarchicznie, tak żeby pewne wcześniejsze neurony mogły określać, czy nowe punkty znajdują się powyżej, czy poniżej odpowiednich prostych, a następnie dalsze neurony mogły określać, czy punkt znajduje się w obszarze ograniczonym przez te proste itd. (zob. rys. 12.7). Problem polega zatem na opracowaniu algorytmu, który umie wyznaczać wektor wag dla takiej znacznie większej sieci neuronowej.



Rysunek 12.6. Ponownie, *plusy* reprezentują elementy klasy 1, a *krzyżyki* elementy klasy 2. W tym wypadku elementy nie są liniowo separowalne. Sieć neuronowa z rys. 12.2 nie może poklasyfikować wszystkich tych punktów bez popełniania błędów



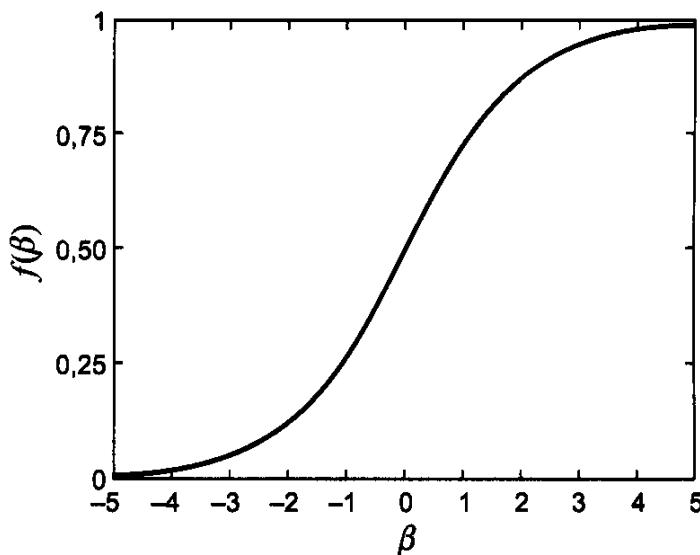
Rysunek 12.7. Chociaż jedna granica decyzyjna w postaci prostej nie może rozdzielić tych danych, można tego dokonać za pomocą wielu prostych. Wszystkie granice są tutaj przedstawione jako N_1, \dots, N_6 , oznaczające proste, które odpowiadają progom uruchamiania hipotetycznych neuronów. Łącząc wyniki działania wielu neuronów, można prawidłowo zidentyfikować wszystkie elementy. Identyfikacja elementów może być przeprowadzana za pomocą następujących reguł: 1) jeśli punkt jest ponad N_1 , to jest w klasie 1; 2) jeśli punkt jest pod N_1 i pod N_2 , to jest w klasie 2; 3) jeśli punkt jest poniżej N_1 i ponad N_2 , i pod N_3 i ponad N_4 , to jest w klasie 1; 4) jeśli punkt jest pod N_1 i pod N_4 , to jest w klasie 2; 5) jeśli punkt jest pod N_1 i ponad N_5 , to jest w klasie 1; 6) jeśli punkt jest ponad N_3 i pod N_5 i pod N_6 , to jest w klasie 2; 7) jeśli punkt jest ponad N_6 , to jest w klasie 1. Zauważ, że każda z tych reguł może być zakodowana za pomocą neuronów przetwarzających wyniki działania N_1, \dots, N_6

12.2. Propagacja wsteczna dla wielowarstwowych perceptronów ze sprzężeniem do przodu

Jedną z możliwości jest wprowadzenie przybliżenia progowej (wszystko albo nic) funkcji wyjściowej każdego neuronu i skorzystanie z metod analizy matematycznej. Przypuśćmy, że zamiast stosowania funkcji z ostrym progiem rozgraniczającym przyjmujemy, że każdy neuron daje stopniowo zmieniający poziom wyjścia o wartościach z przedziału $[0, 1]$. Im większa jest ważona aktywność wejścia neuronu, tym wyjście będzie miało wartość bliższą 1,0. Ukonkretnijmy te rozważania, zakładając, że każdy z neuronów jest sterowany funkcją sigmoidalną (esowatą)

$$f(\beta) = \frac{1}{1 + e^{-\beta}}$$

przy czym β jest iloczynem skalarnym wszystkich wejściowych aktywacji i związanych z nimi wag (bardzo podobnym do wcześniejszego $\mathbf{w}^T \mathbf{x}$). Wówczas sygnał na wyjściu każdego neuronu będzie wyglądać tak jak na rys. 12.8. Wyjście będzie się nasycać na obu skrajnych końcach ważonych wartości wejściowych. Gdy funkcja wyjściowa jest gładka i ciągła, jak to jest przy korzystaniu z funkcji sigmoidalnej takiej jak ta, możliwe jest stosowanie metody „propagacji wstecznej”, służącej do poprawiania wag w celu minimalizacji błędu kwadratowego klasyfikacji sieci neuronowej.



Rysunek 12.8. Funkcja sigmoidalna $1/(1 + e^{-\beta})$ jest często używana jako przybliżenie funkcji progowej dla neuronów w sieci neuronowej. Gdy poziom aktywacji jest silnie dodatni, neuron podaje wartości bliskie 1,0. Gdy poziom aktywacji jest silnie ujemny, neuron podaje wartości bliskie 0,0. Dla aktywacji z zakresu $[-1, 1]$ funkcja sigmoidalna jest bliska funkcji liniowej, ale potem nasyca się w obu kierunkach

Podstawowa koncepcja polega tu na tym, żeby zacząć od wyjścia sieci neuronowej i poprawiać wagi połączone z węzłem wyjściowym w celu zmniejszenia błędu między rzeczywistym a pożdanym działaniem sieci. Stanie się

to bardziej oczywiste, gdy rozważymy n wzorców wejściowych $\mathbf{x}_1, \dots, \mathbf{x}_n$ oraz pożądane klasyfikacje każdego z nich $d_j(1), \dots, d_j(n)$, przy czym $d_j(p)$ odnosi się do pożądanej (ang. *desired*) wartości wyjściowej na j -tym węźle wyjściowym przy podaniu na wejście p -tego wzorca \mathbf{x}_p . (Zauważ, że poniżej używamy i jako zmiennej indeksowej w różnych sytuacjach, należy zatem uważać, żeby odpowiednio interpretować ten symbol w różnych kontekstach). Tak określone „cele” mogą być liczbami rzeczywistymi (np. 0 lub 1, ale nie jesteśmy tutaj ograniczeni do wartości boolowskich). Dla danego zbioru wag \mathbf{w} węzeł wyjściowy o numerze j odpowiada na każdy wzorzec wejściowy zbiorem wyników w postaci liczb rzeczywistych $y_j(1), \dots, y_j(n)$.

Cel jest tutaj zdefiniowany jako takie poprawienie \mathbf{w} , żeby zminimalizować sumę kwadratów błędów związanych z sygnałami wejściowymi sieci neuronowej dla poszczególnych wzorców wejściowych. Wynika z tego, że musimy najpierw obliczyć sumę kwadratów błędów względem wszystkich wyjść j dla każdego wejścia \mathbf{x}_p :

$$E(p) = \frac{1}{2} \sum_j e_j^2(p)$$

gdzie $e_j(p) = d_j(p) - y_j(p)$, a następnie obliczyć średnią tych kwadratów błędów względem wszystkich wzorców $\mathbf{x}_1, \dots, \mathbf{x}_n$:

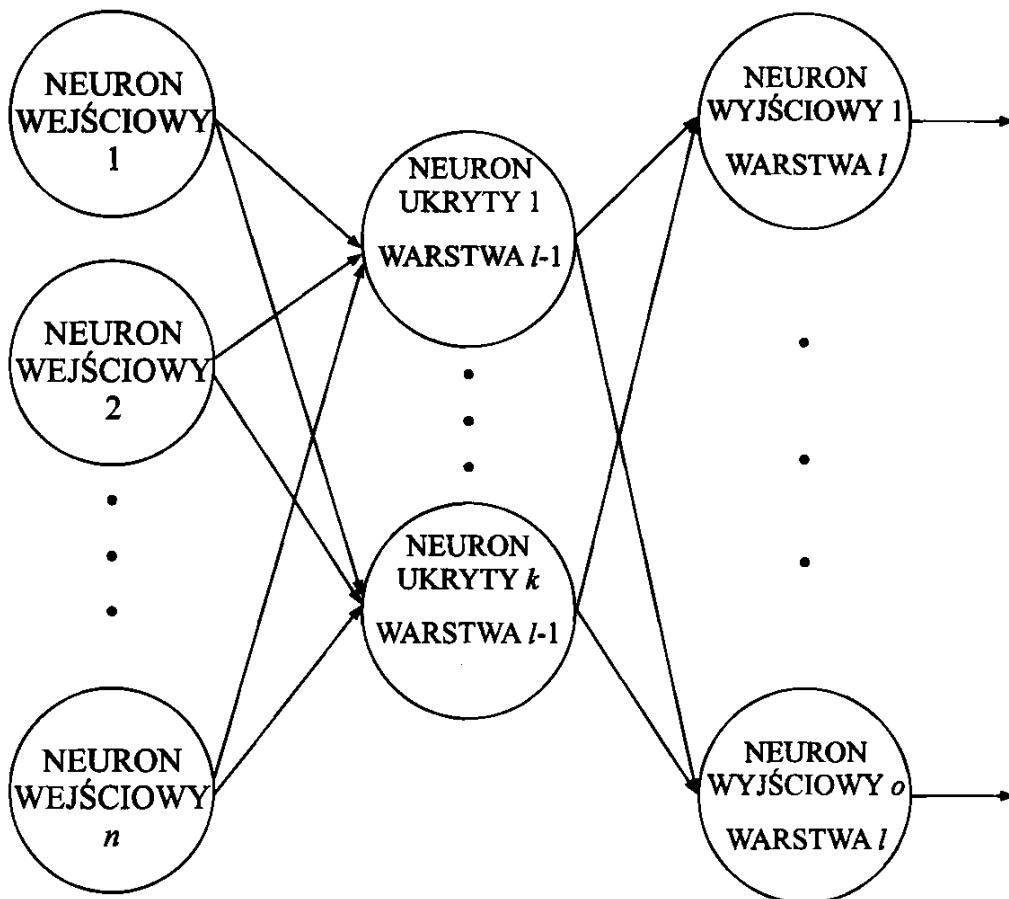
$$E = \frac{1}{n} \sum_{p=1}^n E(p)$$

Przypomnijmy, że każde $y_j(p)$ jest funkcją wektora wejściowego \mathbf{x}_p oraz wektora wag \mathbf{w} :

$$y_j(p) = g(\mathbf{x}_p, \mathbf{w})$$

gdzie g jest pewną funkcją nielinową. Założmy, że sieć neuronowa jest zbudowana z l warstw, przy czym wejścia tworzą warstwę pierwszą, a wyjścia – warstwę l (rys. 12.9). Dzięki takiemu hierarchicznemu rozwiązaniu jest możliwe skupienie uwagi na wagach, które łączą węzły warstwy $l - 1$ z każdym węzłem wyjściowym, i poprawienie ich tak, żeby zredukować błąd na każdym wyjściu sieci, mierzony względem wszystkich wzorców wejściowych.

Jedną z często stosowanych metod poprawiania parametrów (wag), umożliwiającą minimalizację sumy kwadratów błędów modelu (w naszym wypadku modelem jest sieć neuronowa), jest zmienianie ich proporcjonalnie do wielkości błędu pochodzącego ze związanych z nimi łączys. W ten sposób, gdy model działa dobrze i błąd jest mały, parametry są zmieniane nieznacznie. Gdy model działa kiepsko, błąd jest duży i odpowiednie zmiany parametrów będą raczej efektywniejsze, gdy będą się odbywały z dużym krokiem. Inną, pokrewną metodą jest zmienianie parametrów proporcjonalnie do lokalnego gradientu funkcji błędu. Dodatkowo parametry są często zmieniane proporcjonalnie do wartości aktywujących wejścia, tak że większe zmiany są wykonywane, gdy wartości wejściowe są większe, przy czym odpowiednio mniejsze zmiany są wykonywane dla parametrów o mniejszej wartości.



Rysunek 12.9. Wielowarstwowa sieć neuronowa ze sprzężeniem do przodu. Ma ona n neuronów wejściowych połączonych z węzłami „ukrytymi”. Na rysunku pokazano jedną warstwę neuronów ukrytych, nie ma jednak ograniczenia na liczbę takich warstw ukrytych. Końcową warstwą jest warstwa wyjściowa l , która obejmuje o neuronów wyjściowych. W całkowicie połączonej sieci ze sprzężeniem do przodu wszystkie neurony każdej warstwy są połączone ze wszystkimi neuronami warstwy następnej. Zauważ, że sieć neuronowa może mieć więcej niż jedno wyjście (tzn. może być wielowartościowa)

Wracając do węzłów wyjściowych sieci neuronowej, gdybyśmy na podstawie wielkości błędu zaobserwowanego dla p -tego wzorca wejściowego z użyciem powyżej opisanych zasad mieli poprawić i wag, które łączą się z każdym j -tym węzłem, moglibyśmy wykonać operację

$$\Delta w_{ji} = \eta \delta_j(p) y_i(p) \quad (12.1)$$

przy czym: p to numer wzorca wejściowego, j – węzeł wyjściowy, i – indeks, który zmienia się od 1 do liczby węzłów w warstwie $l - 1$, Δw_{ji} – zmiana, jaką trzeba wprowadzić do wagi połączenia między i -tym węzłem w warstwie $l - 1$ a j -tym węzłem wyjściowym, η – stała skalująca, $y_i(p)$ – aktywacja wejściowa wychodząca z i -tego węzła w warstwie $l - 1$, $\delta_j(p)$ – lokalny gradient j -tego węzła wyjściowego dla wzorca wejściowego x_p .

Żeby zaimplementować regułę zmiany wagi, należy obliczyć gradient $\delta_j(p)$. Wymaga to użycia narzędzi analizy matematycznej. Odpowiedni przegląd

sposobów obliczania $\delta_j(p)$ zaprezentowano w [210], strona 144. Główny wynik jest następujący:

$$\delta_j(p) = e_j(p)f'(\beta_j(p))$$

przy czym: $f'(\cdot)$ jest pochodną funkcji przenoszenia, a $\beta_j(p)$ – iloczynem skalarnym i -tej wagi wejściowej oraz wartości aktywacji na węźle wyjściowym przy podaniu na wejście sieci wzorca \mathbf{x}_p . Tu okazuje się, że nasz wybór funkcji sigmoidalnej dla każdego węzła był udany, gdyż pochodna $f(\beta_j)$ jest łatwa do obliczenia za pomocą wzoru $y_j(p)(1 - y_j(p))$. W ten sposób wagi połączeń prowadzących od wszystkich węzłów i w warstwie $l - 1$ do każdego j -tego węzła wyjściowego mogą być poprawione przez dodanie Δw_{ji} do każdej w_{ji} . Określenie Δw_{ji} wymaga wybrania η , użycia $y_i(p)$ oraz iloczynu $y_j(p)(1 - y_j(p))$.

W tym momencie problem przechodzi w problem poprawiania wag, które łączą węzły w warstwie $l - 2$ z węzłami w warstwie $l - 1$. Teraz nie mamy już bezpośredniej informacji dotyczącej „błędu” związanego z każdym węzłem; nie mamy danej pożąданej wartości wyjściowej. Mimo to w dalszym ciągu możemy określić wartość $\delta_j(p)$ dla każdego „węzła ukrytego” jako

$$\delta_j(p) = y_j(p)(1 - y_j(p)) \sum_k \delta_k(p)w_{kj}(p)$$

przy czym k zmienia się od 1 do liczby węzłów wyjściowych (lub gdy mamy do czynienia z wieloma warstwami ukrytymi, do odpowiedniej liczby węzłów w następnej, wyższej warstwie). Składnik, będący sumą to w istocie związany z każdym węzłem w warstwie l gradient ważony. Wielkość uaktualnienia dla wag połączeń z warstwą ukrytą $l - 1$ jest znowu wyznaczana za pomocą (12.1).

Po uaktualnieniu wszystkich wag związanych z węzłami warstwy $l - 1$ możemy postępować dalej wstecz sieci, warstwa po warstwie, aż zostaną uaktualnione wszystkie wagi. Potem będziemy musieli wykonać przetwarzanie dla następnego wzorca wejściowego i ponownie przeliczyć uaktualnienia wag. W ten sposób będziemy powtarzać te czynności do momentu, aż wagi zaczną wykazywać zbieżność. Istnieje wiele wersji tej procedury, ale jej istota jest zawsze taka sama: należy na podstawie gradientu zminimalizować kwadrat błędu uzyskanego jako wynik porównania wartości wyjściowej funkcji nieliniowej (sieci neuronowej) z docelowymi wynikami związanymi z rozważanymi danymi wejściowymi.

Wynik działania tego algorytmu będzie wykazywał zbieżność, ale uzyskane w ten sposób optimum wektora wag \mathbf{w}^* będzie lokalne. To znaczy, dla pewnego otoczenia epsilonowego wokół odkrytego wektora wag \mathbf{w}^* błąd sieci neuronowej będzie minimalny. Nie mamy gwarancji, że nie istnieją inne wektory wag, które dają mniejszy błąd niż uzyskany \mathbf{w}^* , a które są umiejscowione poza jego epsilonowym otoczeniem. Problem ten jest poważny, gdyż funkcja błędu (suma kwadratów różnic między pożądanym a faktycznym wynikiem działania sieci obliczona względem wszystkich wzorców wejściowych) może mieć wiele minimów lokalnych i to dość oddalonych od prawdziwego minimum globalnego (być może osiąganego w wielu punktach). Algorytm propagacji wstecznej

może ustabilizować się na wartości gorszej od optymalnej. Jeśli sieć neuronowa nie daje sprawdzalnego idealnego wyniku (tzn. o zerowym błędzie klasyfikacji), to jedynym sposobem przekonania się, że uzyskany zbiór wag jest odpowiedni, jest wielokrotne uruchomienie opisanej procedury z różnymi, losowo wybranymi wektorami początkowymi wag. Jeśli przy kolejnych próbach procedura osiągnie w zasadzie te same wagi, to umacnia to wiarę, że uzyskane wagi są rzeczywiście optymalne, jednak nigdy nie możemy mieć tutaj całkowitej pewności.

Sieć neuronowa przedstawiona na rys. 12.9 jest nazywana perceptronem wielowarstwowym, przez analogię do jednego perceptronu z funkcją progową opisanego w pracy [396] Rosenblatta. Tę sieć określamy jako sieć „ze sprzężeniem do przodu”, gdyż informacja przepływa w niej tylko w jednym kierunku: od węzłów wejściowych przez węzły ukryte do jednego lub więcej węzłów wyjściowych. Istnieją też inne sieci neuronowe ze sprzężeniem do przodu, które korzystają z innych funkcji przenoszenia. Niektóre z nich wykorzystują zamiast funkcji sigmoidalnej funkcję Gaussa. Inne używają gaussowskich funkcji gęstości, które działają, korzystając z metryki odległości między przychodzącym wektorem aktywującym a określonym wektorem średnich wartości współrzędnych oraz ich odchyleń standardowych (nazywamy je „sieciami o radialnych funkcjach bazowych”). Istnieje wiele innych możliwości, z których każda może pasować do rozwiązywanego przez nas problemu klasyfikacji. Różne rodzaje danych wejściowych i klasyfikacji mogą wymagać różnych funkcji przenoszenia w sieci neuronowej.

Jeden z ważnych wyników uzyskanych w późnych latach osiemdziesiątych XX wieku może oznaczać, że rozróżnianie za pomocą wielowarstwowych sieci neuronowych ze sprzężeniem do przodu może być wykorzystane jako „uniwersalny aproksymator funkcji” [235, 358]. W istocie oznacza to, że – mając dostateczną liczbę węzłów w warstwie ukrytej, odpowiednio poprawiając wektor wag w – sieć neuronowa może obliczyć dowolną funkcję mierzalną ze z góry daną dokładnością. Własność ta występuje w sieciach neuronowych zarówno z funkcjami sigmoidalnymi, jak i Gaussa. W związku z tym zwykle wybieramy właśnie te funkcje, ponieważ dobrze jest, projektując taki system, wiedzieć, że nie ma on ograniczeń co do funkcji wynikowych. Jednak wybór ten często daje fałszywe poczucie bezpieczeństwa, a nie jest podytutowany żadną praktyczną przesłanką, gdyż funkcje te mogą odwzorowywać dowolne funkcje tylko wówczas, gdy liczba węzłów ukrytych dąży do nieskończoności. W praktyce złożoność większości sieci neuronowych jest daleka od tego, co byłoby konieczne, żeby wykonywać dowolne odwzorowywania z dostępną komputerem dokładnością.

Ponadto w wielu zastosowaniach sieci neuronowych ze względu na własność uniwersalności stosujemy tylko jedną warstwę ukrytą. Niejednokrotnie jest tak, że sieć o jednej warstwie ukrytej może wymagać o wiele więcej węzłów, żeby obliczyć to, co sieć wielowarstwowa może uzyskać przy zaledwie kilku węzłach w każdej warstwie. A zatem poleganie na sieciach neuronowych z jedną warstwą ukrytą o funkcji przenoszenia sigmoidalnej lub gaussowskiej jest równie często pomocne przy znajdowaniu odwzorowania, jak i przeszkadza. Wybór odpowiedniej struktury sieci (topologii) pozostaje wciąż raczej sztuką niż nauką.

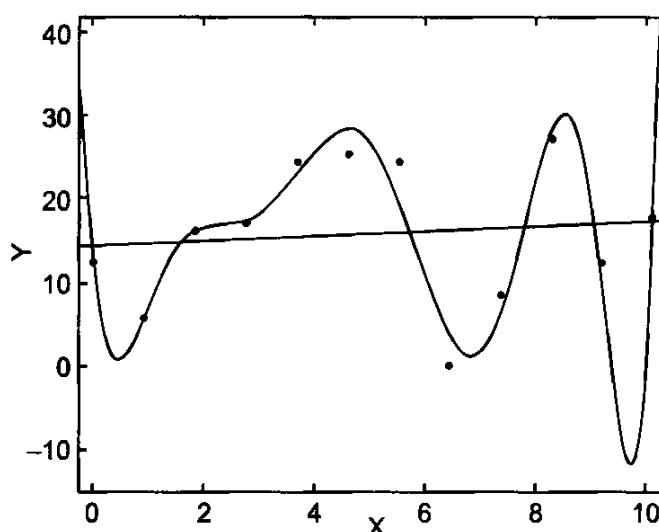
12.3. Szkolenie i testowanie

Załóżmy, że dysponujemy źródłem danych i stoimy przed problemem opracowania klasyfikatora (tzn. procedury lub urządzenia, które będzie używane do stwierdzania, czy wskazane dane należą do jakiejś określonej klasy). Możemy mieć, na przykład, dane na temat cech charakterystycznych 100 mammogramów i zadanie opracowania na ich podstawie klasyfikatora, który potrafi określić, czy mamy do czynienia z nowotworem złośliwym. Założymy dodatkowo, że dla każdego zbioru cech mammogramu znamy poprawną odpowiedź – wiemy, czy pacjent ma raka, czy nie. Założenie to jest dość realistyczne. Możemy mieć do czynienia z przypadkami, gdy pacjenci mieli podejrzane obszary w mammogramach i pobrano tkanki lub zwapnienia [159, 149]. Wreszcie założymy, że dane nie są liniowo separowalne i że podjęliśmy decyzję, żeby opracować sieć neuronową, która będzie służyła jako klasyfikator.

Następne zadanie polega na określeniu rozmiaru zbioru danych treningowych. W naszym wypadku mamy do dyspozycji 100 wzorców wejściowych. Możemy mieć pokusę, żeby użyć wszystkich tych danych, aby wyszkolić perceptron wielowarstwowy, wykorzystując algorytm propagacji wstecznej. Przy takim podejściu napotykamy jednak pewien problem – powstała sieć neuronowa jest „krucha” w tym sensie, że uczymy ją w ten sposób, aby dopasowywała się dokładnie również do szumów występujących w danych. Gdy zostaną zebrane nowe dane i podamy je sieci, którą wyszkoliliśmy za pomocą wszystkich poprzednio dostępnych danych, wtedy sieć często nie działa na takim samym poziomie, jak na to wskazywały wyniki szkolenia. Konieczne jest zostawienie części dostępnych danych, żeby służyły do testowania klasyfikatora neuronowego, który jest trenowany na pozostałym podzbiorze danych. Ilość danych, które należy przeznaczyć na zbiory treningowy i testujący, zależy od konkretnego problemu i wymaga oceny statystycznej. Często dostępne dane są dzielone losowo na równe zbiory – jeden przeznaczony do szkolenia, a drugi do testowania. Nie ma jednak takiej reguły, która działałaby we wszystkich sytuacjach. Co więcej, możesz przewidzieć, że po zakończeniu szkolenia i przetestowaniu opracowanej sieci na nowych danych możesz chcieć zmienić sieć, wyszkolić ją od początku i spróbować jeszcze raz. Niektórzy praktycy zalecają wprowadzenie trzeciego zbioru danych służącego do wykonywania testu końcowego. W ten sposób procedura wielokrotnego powtarzania szkolenia i testowania nie przodzi się w szkolenie na wszystkich dostępnych danych.

Alternatywna procedura, korzystająca z dostępnych danych, jest związana z używaniem grup danych zwanych próbками. W metodzie, zwanej próbkowaniem z odkładaniem jednej grupy na bok, każdy zbiór danych jest po kolei usuwany z procesu szkolenia. Po szkoleniu odłożona próbka jest klasyfikowana najlepszą siecią neuronową i jest zapisywana uzyskana wielkość błędu. Próbka ta jest następnie włączana do zestawu szkoleniowego, a jest usuwana z niego kolejna grupa, po czym cały proces jest powtarzany. W ten sposób każdy punkt danych jest klasyfikowany przez sieć neuronową, która nie wykorzystywała go w procesie szkolenia. Więcej informacji na temat metod statystycznego próbkowania można znaleźć w pracy [110].

Jeśli wiele prób szkolenia sieci neuronowej nie kończy się akceptowalnym poziomem błędu, to korzystne może być dodanie nowych węzłów do struktury sieci. Większa sieć może przybliżać więcej funkcji, ale jednocześnie taka sieć jest bardziej zagrożona zbytnim dopasowaniem do dostępnych danych. Jest to ten sam problem, który występuje, gdy próbujemy dopasować wielomiany do danych. Gdy patrzymy na rys. 12.10, intuicja podsuwa nam, że dane są dobrze reprezentowane przez prostą, ale wielomian dziewiątego stopnia pasuje do dostępnych danych z mniejszym błędem. Problem z modelem dziewiątego stopnia polega na tym, że nie umożliwia on uogólnienia ze względu na dane, które jeszcze się nie pojawiły. Jego błąd w tych niewidocznych punktach będzie prawdopodobnie duży. Wielkie sieci neuronowe sprawiają podobne problemy. Niestety, przy korzystaniu z opartego na gradiencie algorytmu szkolącego, takiego jak algorytm propagacji wstecznej, często zdarza się, że procedura grzeźnie w lokalnie optymalnym zbiorze wag, który nie daje pożądanego poziomu błędu. Czasem jedynym rozwiązaniem tego problemu jest pogodzenie się z nadmierną złożonością sieci neuronowej lub zmiana metody treningu.



Rysunek 12.10. Dopasowywanie prostej dającej minimalny błąd kwadratowy oraz wielomianu dziewiątego stopnia do danych wygenerowanych z prostej przy dodanym szumie Gaussa. Chociaż błąd jest mniejszy przy dopasowywaniu wielomianem dziewiątego stopnia, możliwości uogólnienia w jego wypadku są także mniejsze. Użycie wielomianu девятого stopnia wróży powstawaniu się dużych błędów dla danych między dostępnymi próbami i jeszcze większych błędów poza zakresem dostępnych danych, gdzie funkcja dąży do nieskończoności

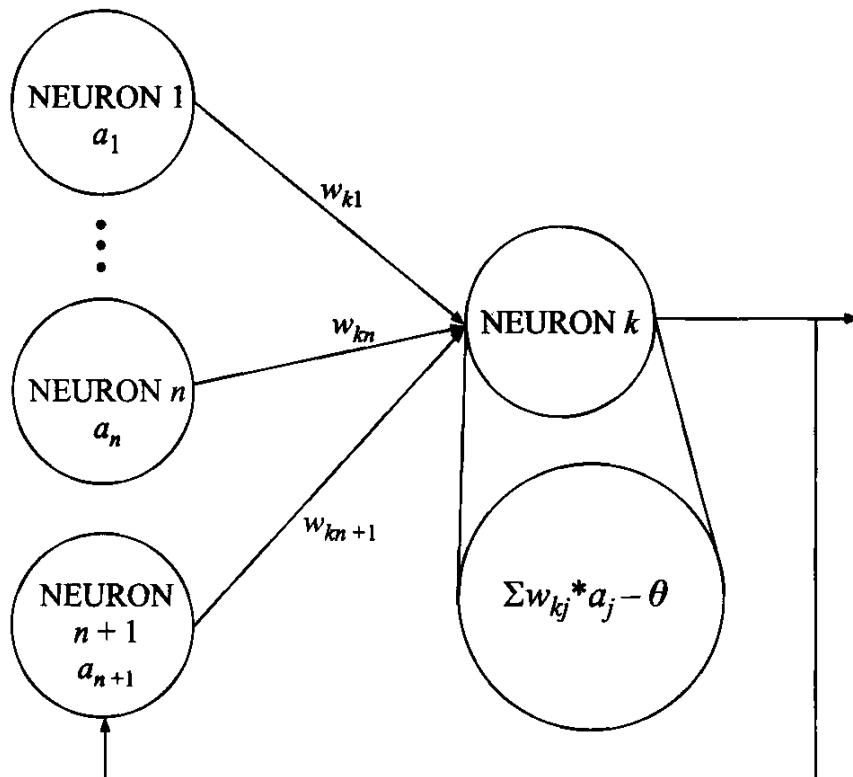
Jedną z takich alternatywnych metod jest symulowane wyżarzanie (zob. rozdział 5). Wyżarzanie to stochastyczny algorytm przeszukiwania i mieliśmy już okazję obserwować, jak możemy tę procedurę zrównoleglić i zmodyfikować do postaci algorytmu ewolucyjnego. W związku z tym nie budzi zdziwienia fakt, że obliczenia ewolucyjne są potencjalnie użyteczne przy opracowywaniu i szkoleniu sieci neuronowych. Omówimy to dokładniej na końcu tego rozdziału.

12.4. Sieci rekurencyjne i architektury rozszerzone

Sieci neuronowe ze sprzężeniem do przodu to w istocie statyczne urządzenia klasyfikujące wzorce. Traktują one każdy wzorzec wejściowy niezależnie od innych wzorców wejściowych i niezależnie od wyników. Czasem ma to sens, ale przypuśćmy, że chcemy otrzymać model dynamiki samochodu na autostradzie. Pozycja samochodu, jego prędkość i przyspieszenie w następnym momencie są funkcją jego bieżącej pozycji, prędkości i przyspieszenia. Oznacza to, że istnieje rekurencyjna zależność między wejściem a wyjściem. Pomożeby tutaj wykorzystanie tej zależności przez umożliwienie sprzężenia zwrotnego między wyjściem sieci neuronowej a jej wejściem. W innych wypadkach może być korzystne użycie pętli sprzężenia zwrotnego od węzłów z dalszych warstw do węzłów warstw wcześniejszych, co jest swego rodzaju realizacją pamięci w sieci neuronowej. Każda sieć neuronowa, która zawiera takie sprzężenie zwrotne, nie jest już określana jako sieć ze sprzężeniem do przodu, ale jako sieć *rekurencyjna*. Zaproponowano wiele rekurencyjnych sieci neuronowych.

12.4.1. Standardowe sieci rekurencyjne

Być może, najprostszym sposobem realizacji rekurencji jest połączenie wyjść sieci neuronowej z jej wejściami. Stąd następny wynik jest funkcją nie tylko kolejnych danych wejściowych, ale także poprzedniego wyniku (rys. 12.11). Takie sieci są czasem używane do uczenia się lub do generowania



Rysunek 12.11. Prosta sieć rekurencyjna. Sygnał wyjściowy (wynik) sieci jest przekazywany jako sygnał wejściowy dla neuronu $n + 1$

ciągu wartości wyjściowych na podstawie jednego wejścia lub ciągu wejść. Jest to istota języka: ciąg symboli wziętych razem przekazuje pomysł. W tym sensie sieć rekurencyjna jest urządzeniem prognozującym. Dla danego wejścia przewiduje ciąg wyjść. Coś, co robimy na co dzień, mówiąc. Fraza „Chłopiec kopnął...” jest najczęściej kończona przez „piłkę”, ale czasem przez „dziewczynkę”. Przewidujemy symbole pojawiające się w następnej kolejności na podstawie już otrzymanych danych oraz prawdopodobnych wyników, które mogą być generowane w każdym kroku. Sieci rekurencyjne mogą dać zwartą reprezentację tego rodzaju ciągów symboli.

12.4.2. Sieć Hopfielda

Zupełnie innym typem sieci rekurencyjnej jest *sieć Hopfielda* [232], która składa się ze zbioru całkowicie połączonych ze sobą neuronów (każdy komunikuje się z każdym). Każdy neuron przybiera wartość ± 1 (określaną jako stan) w zależności od tego, czy suma ważona podłączonych stanów przesunięta o stałą wartość progu jest większa od 0. Można to wyrazić następującym wzorem:

$$y_j = \sum_{i=1}^N w_{ji} s_i - \theta_j$$

przy czym: w_{ji} oznacza wagę połączenia między neuronem i a neuronem j , s_i to stan i -tego neuronu, a θ_j jest przesunięciem j -tego neuronu. Z kolei stan neuronu j jest określony jako

$$s_j = \text{sgn}[y_j]$$

przy czym sgn oznacza *funkcję signum*, która daje wartość +1, jeśli argument jest dodatni, oraz -1, jeśli argument jest ujemny. Gdy y_j jest równe zeru, wynik funkcji może być określony jako dowolna wartość spośród ± 1 .

W sieci tej, zamiast poprawiania wag połączeń w celu uzyskania określonych wyjść dla wszystkich wejść, jest zapamiętywana pewna liczba N -wymiarowych wektorów zawierających symbole ± 1 reprezentowane przez wagi, które łączą neurony i związane z nimi stany. Dla każdego zapamiętanego wzorca \mathbf{v}_i , $i = 1, \dots, p$, wagi sieci są określone wzorem

$$w_{ji} = \begin{cases} \frac{1}{N} \sum_{k=1}^p v_{kj} v_{ki} & \text{jeśli } j \neq i \\ 0 & \text{jeśli } j = i \end{cases}$$

przy czym v_{kj} jest j -tym elementem wzorca \mathbf{v}_k . Tak więc wagi są funkcją ilorazu zewnętrznego zapamiętanych wzorców. Po jednorazowym obliczeniu mają stałe wartości i nie są kolejno poprawiane.

Sieć Hopfielda jest następnie używana do pamiętania zapisanych wzorców jako uogólnień wszystkich możliwych wzorców, które będą podawane sieci. Dla dowolnego wzorca \mathbf{x} , który ma zostać zapamiętany, sieć zaczyna od ustawienia

każdego stanu $s_j = x_j$ dla $j = 1, \dots, N$. Następnie elementy s_j są uaktualniane losowo zgodnie ze wzorem

$$s_i \leftarrow \operatorname{sgn} \left[\sum_{i=1}^N w_{ji} s_j \right]$$

Jeśli żaden stan już nie ulega zmianie, to wynikowy wektor stanów \mathbf{s} jest oznaczany jako *punkt stały* lub jako wynik sieci. Zasadniczo w ten sposób sieć zapamiętuje wzorzec \mathbf{x} .

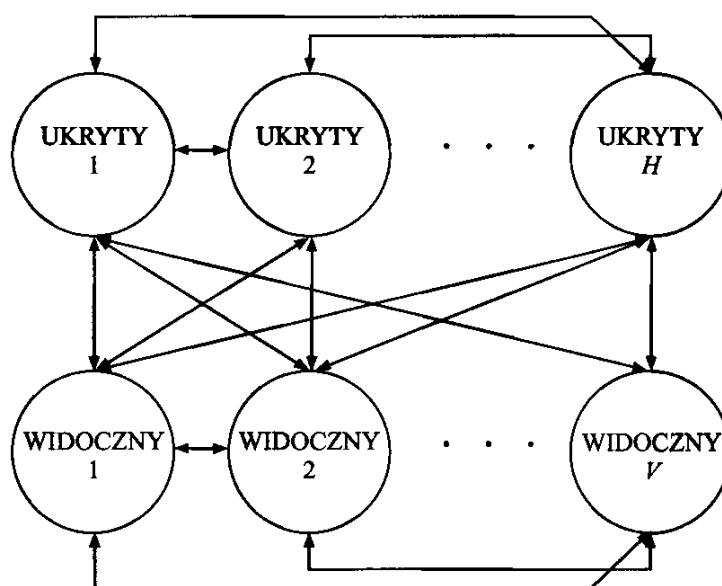
Użyteczność tego podejścia nie polega na możliwości zapamiętania p wzorców $\mathbf{v}_1, \dots, \mathbf{v}_p$, ale raczej na wynikającej z tego zdolności do uogólniania ze względu na nowe wzorce, które są podobne do któregoś z już zapamiętanych. Należy mieć nadzieję, że kiedy sieci zostanie podany wzorzec podobny, powiedzmy, do \mathbf{v}_1 , wtedy sieć będzie przechodzić do tego samego zbioru stanów \mathbf{s} , który był uzyskany, gdy sieci podawano \mathbf{v}_1 , a który jest inny niż zbiory stanów napotykanych przy przetwarzaniu pozostałych wzorców. Niezawodność tego procesu zależy od liczby zapamiętanych wzorców, ich podobieństwa oraz stopnia zasymienia w nowych wzorcach podawanych sieci. Sieci tego typu są często określane jako *pamięci asocjacyjne* (skojarzeniowe), gdyż kojarzą (ang. *associate*) nowe wzorce ze wzorcami już widzianymi.

12.4.3. Maszyna Boltzmanna

Inny pokrewny typ sieci neuronowej jest nazywany *maszyną Boltzmanna* [225]. Jest to zbiór dowolnie połączonych neuronów (niedozwolone jest jedynie połączenie neuronu z samym sobą), z których pewne są oznaczone jako wejściowe, inne jako wyjściowe, a jeszcze inne są traktowane jako węzły ukryte (rys. 12.12). Tak jak w wypadku sieci Hopfielda, jednostki przetwarzające przyjmują wartości binarne (± 1), a wszystkie połączenia są symetryczne. Uaktualnianie stanu węzłów jest wykonywane dla jednego węzła na raz, przy czym każdy z nich jest wybierany losowo. Stan wybranego węzła jest uaktualniany przez przełączanie między ± 1 na podstawie rozkładu Boltzmanna

$$\Pr(s_j \rightarrow -s_j) = \frac{1}{1 + e^{\frac{\Delta E_j}{T}}}$$

przy czym: s_j jest stanem j -tego węzła, ΔE_j – zmianą energii (funkcją oceny), T – temperaturą związaną z systemem. A zatem stany (węzły) sieci są w istocie swej losowe i mogą bez końca przełączać się w zależności od efektu w ΔE_j . Dodatkowo przy zmniejszaniu temperatury T prawdopodobieństwo przełączenia znaku się zmniejsza. Opracowanie reguły spadku gradientu służącej do uaktualniania wag w sieci jest bardziej skomplikowane niż opracowanie propagacji wstecznej. Szczegóły takiej reguły podał Haykin ([210], str. 321–330). Reguła uaktualniania wagi w_{ji} połączenia między węzłami i oraz j okazuje się funkcją korelacji między s_i a s_j .



Rysunek 12.12. Model maszyny Boltzmanna (sieci neuronowej). Każdy neuron jest połączony ze wszystkimi innymi. Niektóre z neuronów są „widoczne” i mogą być oznaczone jako wejściowe lub wyjściowe. Pozostałe neurony są „ukryte”

12.4.4. Sieć wielu współdziałających programów

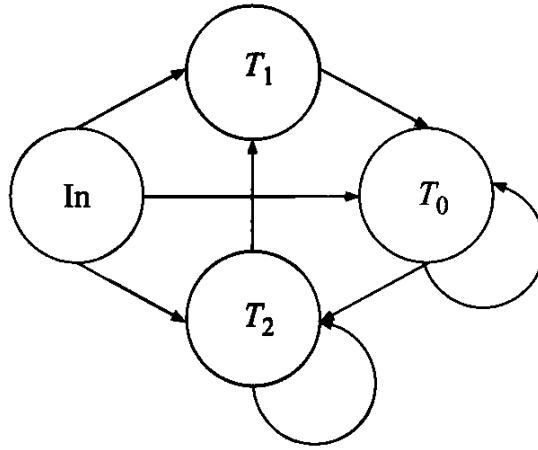
W każdym z wcześniejszej przedstawionych rozwiązań każdy węzeł sieci jest identyczny w tym sensie, że wykonuje tę samą funkcję przenoszenia. Co się jednak stanie, jeśli rozszerzymy ten protokół i dopuścimy, żeby każdy neuron w rzeczywistości mógł realizować dowolną funkcję przenoszenia zapisaną w postaci pewnego wyrażenia symbolicznego? Połączenia między neuronami mogłyby być dowolne i to nawet zwrotne (tzn. łączące neuron z nim samym) przy pewnych neuronach określonych jako wejściowe i pewnych jako wyjściowe. Tego rodzaju architekturę zaproponował Angeline [9], żeby zapewnić generowanie „programów” rekurencyjnych. Na rysunku 12.13 przedstawiono sieć neuronową, w której neurony T_0, T_1 i T_2 mogą być zdefiniowane za pomocą wyrażeń symbolicznych:

$$T_0 = (\text{In} + T_1)T_0$$

$$T_1 = \frac{\text{In}}{T_2 + 0,97831}(T_0 - 0,11345)$$

$$T_2 = T_2 + 0,3745 \times \text{In} \cdot (T_0 + T_1)$$

Warunkiem wstępny przy tworzeniu takiej sieci jest zdefiniowanie wyrażeń, których można użyć. Tutaj wyrażenia takie zawierałyby operatory $\{+, -, \times, /\}$ oraz symbole terminalne $\{\text{In}, \mathbb{R}, T_0, T_1, T_2\}$, przy czym \mathbb{R} oznacza zbiór liczb rzeczywistych. W oczywisty sposób taka sieć „neuronów” jest niezwykle elastyczna, choć odbiega ona od początkowego pomysłu symulowania działania biologicznych neuronów. Teraz możemy mieć neurony realizujące dowolne funkcje! Elastyczność tego rozwiązania stwarza także problemy, które są związane ze szkoleniem takich sieci i wybieraniem ich topologii. Nie istnieją



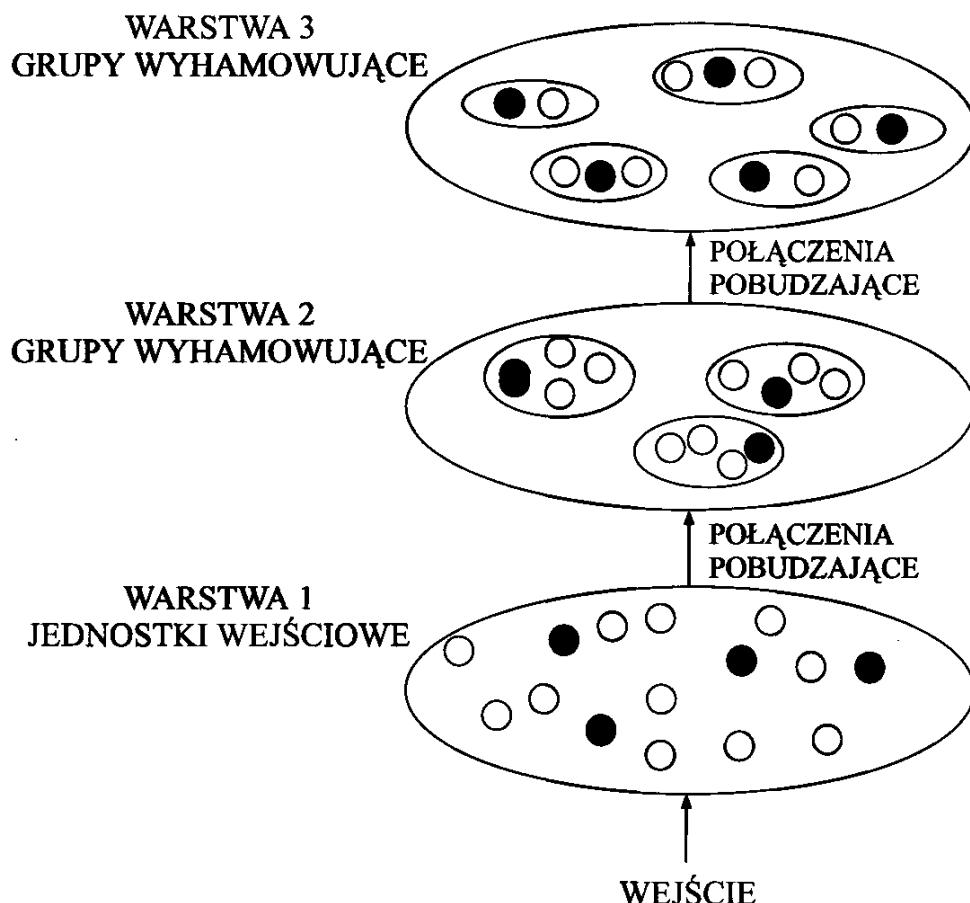
Rysunek 12.13. Model sieci wielu współdziałających programów (ang. *multiple interacting programs* – MIP). Neuron „In” jest neuronem wejściowym, a T_0 , T_1 i T_2 są adaptującymi się programami, które mogą być rekurencyjne

żadne radzące sobie z nimi znane metody gradientowe. Zamiast tego możemy użyć obliczeń ewolucyjnych w celu przystosowania sieci do zadania, które rozwiązujemy. Powrócimy jeszcze do tej możliwości w dalszej części tego rozdziału.

12.5. Grupowanie za pomocą uczenia konkurencyjnego

Cała dotychczasowa dyskusja toczyła się wokół opracowywania urządzenia rozpoznającego wzorce. Założono w niej, że istnieją pewne wzorce, które mają zostać rozpoznane. Zupełnie inna sytuacja pojawia się jednak, gdy otrzymujemy dane i musimy stwierdzić, czy w ogóle można wśród nich wyróżnić jakieś wzorce. Problem ten jest nazywany problemem *wykrywania wzorców*, a nie problemem ich *rozpoznawania*. Takie wykrywanie często jest określane jako *eksploamacja danych* (ang. *data mining*), proces uwidaczniania informacji z dowolnie rozłożonych danych. W takiej sytuacji nie mamy do dyspozycji żadnych z góry danych etykiet klas do nauczenia. Zamiast tego zadanie polega na opracowaniu metody optymalnego grupowania podzbiorów dostępnych danych zgodnie z jakimś przydatnymi regułami. Sieci neuronowe zapewniają możliwość osiągnięcia takiego celu.

Dla uproszczenia założmy, że na dane składają się binarne ciągi wejściowe nad alfabetem $\{0, 1\}$, podobnie jak w wypadku sieci Hopfielda, która działała na wzorcach jako wektorach symboli ze zbioru $\{-1, 1\}$. Celem jest, przez pogrupowanie ciągów wejściowych, odnalezienie w nich pewnych wzorców. Przyjrzyjmy się hierarchicznemu układowi neuronów na rys. 12.14. Każdy neuron wejściowy jest połączony z neuronami na następnym poziomie hierarchii itd. Neurony na wyższych poziomach są połączone w grupy (klastry) i mogą także być połączone przez połączenia wyhamowujące (tzn. wyjście z dowolnego neuronu w grupie usiłuje zmniejszyć wartość wyjścia wszystkich pozostałych neuronów w tej grupie). W danej grupie zadziała tylko jeden neuron – neuron o największej aktywności (założymy, że aktywność jest mierzona w skali $[0, 1]$).



Rysunek 12.14. Model sieci uczenia konkurencyjnego. Sygnał wejściowy pobudza pewne neurony w pierwszej warstwie. Sygnały wyjściowe z tych neuronów są wzmacniane i przekazywane do grup węzłów w drugiej i dalszych warstwach hierarchii, gdzie w każdej z grup jest uruchamiany tylko jeden węzeł. W dalszej kolejności węzły z tej samej grupy mogą być połączone z pozostałymi jej węzłami za pomocą wag hamujących. Po kolejnych podaniach wzorców wejściowych w sieci ustanawiają się związki z każdym wejściem „wzorce uruchamiania neuronów”. Należy mieć nadzieję, że gdy do sieci będą podawane nowe wartości wejściowe, wtedy sieć będzie ponownie wykazywać te same wzorce uruchamiania dla wejść, które będą „bliskie” tym wcześniej zaobserwowanym

Każdemu neuronowi, oprócz neuronów wejściowych, jest przypisana stała waga (1,0), która jest rozłożona między wszystkie jego połączenia wejściowe. To znaczy dla j -tego neuronu

$$\sum_i w_{ji} = 1$$

przy czym w_{ji} jest wagą łączącą neuron i z neuronem j . Początkowo wartości te są wybrane losowo, ale z uwzględnieniem dotychczasowych ograniczeń. Wagi te są aktualniane w wypadku j -tego neuronu tylko wtedy, gdy dla konkretnych danych wejściowych podanych sieci daje on maksymalną wartość wyjścia (sumę ważonych wartości wejść) wśród innych neuronów grupy. Aktualnienie wagi odbywa się zgodnie ze wzorem:

$$\Delta w_{ji} = \begin{cases} 0 & \text{jeśli neuron } j \text{ nie daje maksymalnego wyniku w swojej grupie} \\ g \frac{c_{ik}}{n_k} - gw_{ji} & \text{jeśli neuron } j \text{ daje maksymalny wynik w swojej grupie} \end{cases}$$

przy czym: c_{ik} wynosi 1, gdy i -ty element podanego wzorca S_k jest 1, oraz wynosi 0 w przeciwnym wypadku; n_k jest liczbą aktywnych neuronów w warstwie wejściowej (tzn. liczbą jedynek w S_k); g jest stałą.

Reguła uaktualniania oznacza, że zmiana w wadze w_{ji} wynosi $-gw_{ji}$, jeśli odpowiedni i -ty element wzorca wejściowego to 0. W tym wypadku waga oddaje część swojej wartości w celu ponownego rozdzielenia jej między wagi związane z aktywnymi połączeniami. Dla tych pozostałych aktywnych połączeń zmiana wagi wynosi 0, jeśli $c_{ik}/n_k = w_{ji}$, co oznacza, że waga w_{ji} jest rozkładana równomiernie między aktywne wejścia. W przeciwnym razie wagi, które są większe niż c_{ik}/n_k , zostaną zmniejszone, a wagi mniejsze od tej wartości zostaną powiększone.

W trakcie podawania kolejnych wzorców wejściowych wagi będą dążyły do równomiernego rozkładu między aktywne połączenia, przypomnijmy sobie jednak, że uaktualnianie wag następuje tylko dla neuronów o maksymalnej wartości wyjściowej. Początkowo jest to losowy ciąg wynikający z układu wag na początku, ale w miarę podawania wzorców wejściowych sieć przekształca się w stabilny zbiór neuronów uruchamiany dla każdej „grupy” wzorców wejściowych. Podobne wzorce będą wykazywały tendencję do pobudzania tych samych neuronów. Różniące się wzorce będą miały tendencję do pobudzania różnych neuronów. Gdy zostanie podany nowy wzorzec, konkretny zbiór neuronów, które się uruchomią, wyznaczy grupę dla tego wzorca. W ten sposób sieć wyszukuje strukturę w dostępnych danych i identyfikuje ją za pomocą wzorca neuronów, które zostały uaktywnione po podaniu takiego bodźca. Zauważ jednak, że struktura, która zidentyfikuje sieć, niekoniecznie musi być tą strukturą, której byś się spodziewał. Oznacza to, że gdybyś poprosił ludzi o pogrupowanie liczb $\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$, to mógłbyś otrzymać następujące odpowiedzi:

- $\{1, 2, 3, 4, 5\}; \{6, 7, 8, 9, 10\}$ – „małe” i „duże”,
- $\{1, 2, 3, 5, 7\}; \{4, 8, 9, 10\}$ – „pierwsze” i „złożone”,
- $\{1, 3, 5, 7, 9\}; \{2, 4, 6, 8, 10\}$ – „nieparzyste” i „parzyste”,
- $\{1, 4, 9\}; \{2, 3, 5, 6, 7, 8, 10\}$ – „kwadraty liczb” i „liczby niebędące kwadratami”.

Po zastosowaniu procedury uczenia konkurencyjnego zwykle trudno jest poznać, jakiego rodzaju struktura została wykryta przez sieć. Odkrycie reguły służącej do określania tej struktury jest często nie lada wyzwaniem.

Istnieje wiele innych samoorganizujących się metod grupowania opartych na sieciach neuronowych. Więcej na temat tych innych metod uczenia konkurencyjnego znajdziesz w [210], strony 53, 397–434.

12.6. Wykorzystywanie sieci neuronowych do rozwiązywania TSP

Chociaż na pierwszy rzut oka może się to wydawać nienaturalne, sieci neuronowe możemy także wykorzystać w problemach kombinatorycznych takich jak TSP. Pomyślmy, jak moglibyśmy tego dokonać. Po pierwsze, musimy zdecydować się na rodzaj uczenia: pod nadzorem czy bez nadzoru. Przy uczeniu pod nadzorem konieczne są dane treningowe: pary wejście-wyjście, dla których wiemy, jak powinna wyglądać prawidłowa odpowiedź, i dla których potrafimy obliczyć wielkość błędu popełnionego przez sieć. Jakie powinny być wejścia do sieci uczonej pod nadzorem, gdy jest rozwiązywany problem TSP? Jedyne dane, jakie mamy, to położenie miast, a także metryka odległości (np. euklidesowa). Wydaje się zatem, że wejście powinien stanowić wektor pozycji miast. Jakie powinno być wyjście? Powinno to być jakieś uporządkowanie miast. Dla n miast będziemy potrzebowali n węzłów wyjściowych. Jaki zatem powinien być pożądany wynik? Mogliby to być porządek miast, być może każdy węzeł powinien być wstawiony zgodnie ze swoją siłą wyjściową według kolejności pojawiania się na trasie.

Pojawia się tu jednak problem. Skąd będziemy wiedzieć, jaki ten porządek powinien być? Przecież to jest to, co próbujemy znaleźć! Nie mamy tutaj żadnych danych treningowych. Być może, gdybyśmy dysponowali różnymi już rozwiązaniem TSP dla jakiejś liczby n miast, to moglibyśmy wyszkolić sieć neuronową, żeby przyjmowała na wejściu położenie miast i podawała odpowiedni porządek na wyjściu. Takie rozwiązanie nie będzie jednak działać dla dużych n , gdyż musielibyśmy dysponować wieloma przykładami treningowymi, które są trudne do znalezienia przy dużym n .

Zamiast stosować uczenie pod nadzorem, moglibyśmy wypróbować uczenie bez nadzoru. Ponownie, na wejściu będziemy mieć położenie miast, nie mamy tutaj jednak żadnych konkretnych danych treningowych. Zamiast tego chcemy, żeby sieć sama się zorganizowała w konfigurację, która podaje nam kolejność miast do odwiedzenia. Jednym ze sposobów osiągnięcia tego jest utworzenie sieci o dwóch węzłach wejściowych – po jednym na każdą współrzędną (x, y) pozycji miasta – i połączeniach do n węzłów wyjściowych. Węzły są inicjowane losowymi wagami (w_x, w_y). Pozycje miast są podawane do sieci, która następnie w odpowiedzi poprawia swoje wagi. Przyjmijmy, że indeks podanego do sieci miasta ma wartość m . Znajdujemy węzeł wygrywający, określając, który z węzłów minimalizuje

$$\min_i \{ \| \mathbf{x}_m - \mathbf{w}_i \| \}$$

Norma $\| \cdot \|$ oznacza tutaj odległość euklidesową. Węzeł, który wygrywa konkurencję, powiedzmy k , ma poprawiane wagi tak, żeby znaleźć się bliżej pozycji miasta:

$$\mathbf{w}_k(\text{nowy}) = \mathbf{w}_k(\text{stary}) + \alpha f(G, n)(\mathbf{x}_m - \mathbf{w}_k)$$

przy czym G jest funkcją skalującą, która maleje z upływem czasu, a $f(G, n)$ jest funkcją, która wzmacnia wielkość korekcji, która jest stosowana przy zmienianiu wag k -tego węzła, jak α . Sprawy są jeszcze trochę bardziej złożone, gdyż możemy użyć $f(G, n)$ do poprawienia nie tylko węzła, który wygrał konkurencję, ale także jego sąsiadów, oraz użyć $f(G, n)$ jako funkcji, która kładzie większy nacisk na węzeł wygrywający [61]. Co więcej, możemy opracować $f(G, n)$ w taki sposób, żeby w miarę zmniejszania się G funkcja $f(G, n)$ dawała większą zmianę dla węzła wygrywającego, a mniejszą dla jego sąsiadów. Przy takim uaktualnianiu wag węzeł najbliższy aktualnie podawanemu miastu jest uaktualniany najbardziej, ale jego sąsiedzi są także uaktualniani i przesuwani w kierunku tego samego miasta.

W miarę przechodzenia wszystkich miast na liście będą wygrywały różne węzły. Po wielu podaniach wszystkich miast węzły zorganizują się same w zbiór, w którym miasta położone blisko siebie w przestrzeni będą miały węzły położone blisko siebie w wynikowym pierścieniu. Obchodzenie tego pierścienia węzłów będzie dawało przybliżone rozwiązanie TSP.

Istnieje wiele odmian tej ogólnej procedury, które służą polepszeniu jej efektywności. Jeśli chcesz się dowiedzieć więcej na ich temat, sięgnij do [60, 61]. Podstawowa procedura wydaje się bardzo podobna do reguł uczenia się wykorzystywanych do rozróżniania liniowo separowalnych danych przy zastosowaniu progów. Jest ona także podobna do wcześniejszej opisanej metody uczenia konkurencyjnego. W latach siedemdziesiątych i we wczesnych latach osiemdziesiątych XX wieku sieci neuronowe nie były cenione jako modele systemów inteligentnych. Zmieniło się to, gdy Hopfield i Tank [233] oraz inni badacze pokazali ich oczywistą użyteczność przy rozpoznawaniu wzorców i optymalizacji kombinatorycznej. Dalsza analiza wykazała jednak, że wiele metod sieci neuronowych w zastosowaniu do TSP nie jest konkurencją dla innych heurystyk [180]. Nie każdy problem poddaje się metodom sieci neuronowych, ale czasem opłaca się rozważać alternatywne podejścia do rozwiązywania problemów, które wydają się nie dawać bezpośrednijej korzyści.

12.7. Ewoluujące sieci neuronowe

Szkolenie rekurencyjnej sieci neuronowej jest często trudniejsze niż znajdowanie odpowiednich wag dla wielowarstwowej sieci ze sprzężeniem do przodu. Ale nawet w tym łatwiejszym przypadku, jak się przekonaliśmy, standardowe gradientowe metody szkolenia (takie jak propagacja wsteczna) mogą utknąć w lokalnie optymalnych zbiorach wag, które nie zapewniają najlepszej efektywności, jaką możemy uzyskać z obranej architektury neuronowej. Jest to często pojawiający się problem przy szkoleniu sieci neuronowych, a wynika z tego, że reprezentują one funkcje nieliniowe. Jeśli odwzorowanie parametrów (tutaj są nimi wagi sieci oraz, być może, jej topologia) w wynik jest funkcją nieliniową, a wynik jest oceniany za pomocą jeszcze innej funkcji nieliniowej, to odpowiednie odwzorowanie parametrów w ocenę jest często „upstrzone” lokalnymi optimami. W takich

wypadkach gradientowe i inne klasyczne metody optymalizacji parametrów często nie dają zadowalających wyników.

Przy szkoleniu i projektowaniu sieci neuronowych możemy zdać się na podejście ewolucyjne. Jeśli mamy do czynienia z ustaloną architekturą, to każdy osobnik jest zbiorem wag definiującym działanie wejścia-wyjścia systemu. Wagi są często wyrażane jako wartości rzeczywiste, a więc można je reprezentować w postaci wektora liczb rzeczywistych i poddawać losowemu różnicowaniu oraz selekcji z uwzględnieniem funkcji oceny. Funkcja ta może być choćby wcześniej opisaną funkcją kwadratów błędów, warto jednak zauważyć, że nie musi ona już być funkcją gładką. Wybrałyśmy funkcję kwadratów błędów, gdyż umożliwia stosowanie rachunku różniczkowego i wyprowadzenie opartego na gradiencie równania aktualniania, służącego do szkolenia wag. Rezygnując z metod gradientowych, możemy także zrezygnować z ograniczenia, żeby funkcja oceny była różniczkowalna. Otwiera to wiele dodatkowych możliwości projektowania sieci neuronowych, w których stosuje się funkcje oceny lepiej reprezentujące wartość klasyfikatora w kategoriach rzeczywistych kosztów i zysków.

Jeśli mamy, na przykład, poklasyfikować mammogramy ze względu na to, czy wskazują obecność nowotworu, to mamy dwa sposoby poprawnej diagnozy: 1) poprawnie określić, że jest to nowotwór złośliwy i 2) poprawnie określić, że nie jest to nowotwór złośliwy. Z punktu widzenia operacyjnego chcieelibyśmy ustalić pożądany wynik działania sieci na 1 dla przypadków złośliwych i 0 dla łagodnych. Następnie zmierzylibyśmy błąd kwadratowy wartości docelowej związanej z każdym mammogramem. Robiąc to, nadamy określonym o tym samym stopniu prawidłowości tę samą wartość. Sieć neuronowa, która prawidłowo identyfikuje wynik jako złośliwy (powiedzmy, że daje ona wtedy 1,0), nie daje błędu. Podobnie nie daje błędu sieć, która prawidłowo rozpoznaje przypadek łagodny (powiedzmy, że daje ona wtedy 0,0). Te dwa prawidłowe wyniki mają jednak różną wartość dla pacjenta i lekarza. Zwykle o wiele ważniejsze jest prawidłowe określanie przypadków złośliwych niż łagodnych.

Dodatkowo, błędy nie są tutaj tak samo kosztowne. Istnieją dwa rodzaje błędu: 1) błąd pierwszego rodzaju (typu I), czyli fałszywy alarm, gdy sieć neuronowa wykazuje, że guz jest złośliwy, choć tak nie jest i 2) błąd drugiego rodzaju (typu II), czyli pominięcie, gdy sieć neuronowa rozpoznaje przypadek łagodny, a w rzeczywistości jest on złośliwy. Gdybyśmy użyli zwykłej funkcji błędu kwadratowego, wówczas za błąd przyjęlibyśmy kwadrat różnicy od celu. Oznaczałoby to, że sieć neuronowa, która daje na wyjściu powiedzmy 0,9, gdy celem jest 0,0 (przypadek łagodny), jest tak samo zła jak sieć, która generuje 0,1, gdy celem jest 1,0 (przypadek złośliwy). W praktyce jednak pominięcie jest często o wiele bardziej kosztowne od fałszywego alarmu. Stosowanie ewolucji do znajdowania najlepszego zbioru wag dla sieci neuronowej umożliwia takie dopasowanie funkcji oceny, żeby odpowiadała rzeczywistemu kosztowi i zyskowi z różnych rodzajów poprawnych i niepoprawnych klasyfikacji.

Włożono już wiele wysiłku w metody ewoluowania sieci neuronowych. Najwcześniejsze badania z tej dziedziny pochodzą z lat sześćdziesiątych XX wieku, lecz metody te nie były używane do początku lat osiemdziesiątych [149,

250]. Proces ewolucyjny może poszukiwać nie tylko najlepszego zestawu wag dla ustalonej architektury neuronowej, ale także (w tym samym czasie) najlepszej architektury. Wszystko, co trzeba zapewnić, to kodowanie sposobu, w jaki są reprezentowane połączenia między neuronami. Sama funkcja przenoszenia w każdym węźle też może zostać zakodowana i poddana ewolucji, co pokazuje Angeline we wspomnianym wcześniej systemie wielu współdziałających ze sobą programów, albo jak w pracy [422], gdzie użyto bardziej tradycyjnych funkcji przenoszenia o postaci sigmoidalnej lub Gaussa.

Już tylko na potrzeby ilustracji, rozważmy sieć neuronową o 10 neuronach, z których dwa są oznaczone jako wejściowe (N_1 i N_2). Wyjście będzie się składało z jednego neuronu (N_{10}). Połączenia w tej sieci mogą być reprezentowane w postaci macierzy 10×10 złożonej z jedynek i zer, przy czym 1 oznacza, że jest połączenie między neuronem w i -tym wierszu a neuronem w j -tej kolumnie. Wartość 0 oznacza brak połączenia. Połączenia te możemy różnicować przez losowe zmienianie pozycji w macierzy. Im większa zmiana zostanie wykonana, tym większy jest wpływ na wynikową sieć. Każde połączenie musi mieć też związaną z nim wagę. To z kolei możemy zakodować w postaci macierzy 10×10 złożonej z liczb rzeczywistych. Każdy element macierzy reprezentuje tutaj wagę istniejącego połączenia między i a j . Jeśli połączenia nie ma, to wartość jest pomijana. Wagi mogą być różnicowane przez dodanie szumu Gaussa o zerowej średniej, który wprowadza rozchwanie wartości wszystkich połączeń. Im silniejsze jest to różnicowanie, tym większa jest zmiana w działaniu sieci. Wreszcie możemy także wybrać między funkcjami przenoszenia gaussowskimi a sigmoidalnymi dla każdego węzła, wprowadzając wektor binarny o długości 10. W wektorze tym 1 oznacza, że odpowiedni neuron ma funkcję przenoszenia sigmoidalną; w przeciwnym razie jest to funkcja Gaussa. Bity te możemy zmieniać losowo, tak jak elementy macierzy połączeń.

12.8. Podsumowanie

Budowanie sztucznych mózgów pozostaje w sferze marzeń, jak miało to miejsce od powstania komputerów. Jest to mrzonka i tak będzie przez wiele dekad (lub nawet wieków). W rzeczywistości niektórzy sugerują (np. Walter Freeman), że cyfrowe implementacje sieci neuronowych mogą nie móc emulować biologicznego świata, gdyż prawdziwe mózgi są chaotycznymi urządzeniami analogowymi. Komputery cyfrowe mogą jedynie przybliżać działanie urządzeń analogowych i to zadaną dokładnością. Dokładność ta może nie być wystarczająca do objęcia chaotycznej dynamiki występującej w prawdziwym mózgu. Możliwe jest jednak stworzenie wyidealizowanych modeli pewnych aspektów mózgów biologicznych i zawarcie ich w algorytmach komputerowych, które mogą umożliwić nam rozwiązywanie różnych problemów. Zalety równoległego rozproszonego przetwarzania informacji są różne.

1. Sieci neuronowe dają elastyczne funkcje odwzorowujące, które można dostosować do prawie każdego wymagania ze świata rzeczywistego.

2. Zamiast siłą rzeczy przerwać przetwarzanie, gdy jakiś element sieci neuronowej ulega awarii, można tak zaprojektować jej architekturę, aby była odporna na uszkodzenie. Jest to możliwe, ponieważ przetwarzanie informacji w sieci jest rozproszone między jej połączenia, a nie zgromadzone w jednym miejscu.
3. Istnieją metody szkolenia, które umożliwiają szybkie znajdowanie lokalnie optymalnych parametrów dla wielu architektur neuronowych.
4. Istnieją inne (ewolucyjne) metody trenowania sieci neuronowych, które umożliwiają pokonanie problemu wielu optimów lokalnych oraz dostosowanie topologii i funkcji lokalnego przetwarzania w sieciach neuronowych.
5. Wiele sieci neuronowych można zaimplementować sprzętowo, dzięki czemu można uzyskać bardzo szybkie obliczenia.

Niektóre z tych zalet są także wadami. Na przykład elastyczność oferowana przez sieci neuronowe jest zarówno błogosławieństwem, jak i przekleństwem. Naturalne pragnienie prowadzi do powstawania coraz większych sieci, dzięki czemu sieci te stają się bardziej elastyczne i mogą obliczać bardziej skomplikowane funkcje. Im większa jest jednak sieć, tym bardziej staje się delikatna, gdyż dokładnie dopasowuje się do dostępnych danych i nie udaje jej się wykonywać uogólnienia ze względu na szum czy inne dane. Co więcej, im większa jest sieć i im więcej ma połączeń wewnętrznych, tym trudniej jest wyjaśnić jej działanie. Problem ten może być istotny w medycynie i innych ważnych zastosowaniach, gdzie użytkownik chce znać nie tylko poprawną odpowiedź, ale też rozumowanie, które do niej prowadzi. Wielu użytkowników musi mieć możliwość bronienia się!

Przedstawiony tutaj przegląd metod jest pod pewnymi istotnymi względami niepełny. Wiele powszechnie stosowanych architektur neuronowych pozbawiono, w tym architektury sieci wykorzystujących radialne funkcje bazowe, teorię rezonansu adaptacyjnego, uczenie metodą Hebba, samoorganizujące się odwzorowania (mapy) cech itp. Całą wiedzę na temat tych modeli można ująć jedynie w podręczniku lub poradniku. Dobrymi źródłami informacji dodatkowych są tutaj prace Haykina [210] oraz Fieslera i Beale'a [135].

XIII. Jakiej długości była lina?

Problemy często są wyrażane nieprecyzyjnym językiem i jakiś czas zabiera nam zrozumienie wszystkich koniecznych specyfikacji. Zrozumienie wymogów problemu – niezbędne przy jego modelowaniu – nie zawsze jest proste. Ilustruje to poniższa zagadka.

Przez bloczek była przewieszona lina. Na jednym z jej końców była małpa, a na drugim – ciężarek. Małpa i ciężarek pozostawały w stanie równowagi. Każda stopa liny ważyła ćwierć funta, a małpa i jej matka miały razem cztery lata. Waga małpy i liny wynosiła półtora raza więcej niż wiek matki małpy.

Waga ciężarka przekraczała wagę liny o tyle funtów, ile lat miała małpa w czasie, gdy jej matka była dwa razy starsza od brata małpy w czasie, gdy matka była o połowę młodsza od niego w czasie, gdy będzie on trzykrotnie starszy od matki w czasie, gdy była ona trzy razy starsza niż małpa w poprzednim akapicie.

(Czy teraz rozumiesz, co mieliśmy na myśli, pisząc, że rzeczy nie zawsze są proste?)

Matka małpy była dwa razy starsza od małpy w czasie, gdy matka małpy była o połowę młodsza od małpy w czasie, gdy małpa będzie trzykrotnie starsza od matki w czasie, gdy matka była trzy razy starsza niż małpa w poprzednim akapicie.

Wiek matki małpy przekracza wiek brata małpy o tyle samo, o ile wiek brata małpy przekracza wiek małpy.

Jakiej długości była lina?¹

¹ Należy zaznaczyć, że niewiadome w tym problemie nie muszą być liczbami całkowitymi, jak to było w wypadku pierwszej w tej książce zagadki o wieku trzech synów. Tak więc wiek małpy może, na przykład, wynosić $1\frac{3}{7}$.

Jeśli opis problemu wydał Ci się zbyt zagmatwany, spróbuj przeczytać go jeszcze raz, robiąc przy tym notatki. Do końca książki zostały jeszcze trzy zagadki, więc postaraj się je rozwiązać!

W rzeczywistości problem ten nie jest bardzo trudny do rozwiązania, gdy już się go zrozumie, tzn. gdy wiemy już, jak zamodelować go za pomocą odpowiednich równań. Drugie pod względem trudności jest zrozumienie wszystkich informacji zawartych w opisie problemu. Najtrudniejsze natomiast jest niewątpliwie wykazanie się cierpliwością podczas próby ich zrozumienia. Jest ona wręcz niezbędna do skutecznego rozwiązywania problemów.

Wprowadźmy najpierw kilka zmiennych:

- x – wiek małpy,
- y – wiek brata małpy,
- z – wiek matki małpy,
- W – waga ciężarka,
- r – długość liny.

Teraz możemy przełożyć każde ze zdań zagadki na równania. Będzie łatwiej, jeśli „odczytamy” ją od końca, tzn. rozpoczęmy jej odszyfrowywanie od ostatniego akapitu. Ostatni (czwarty) akapit zagadki mówi nam, że:

$$z - y = y - x \quad (\text{XIII.1})$$

Równanie to jest oczywiste i nie wymaga żadnych komentarzy.

Przyjrzyjmy się teraz trzeciemu akapitowi. „Matka małpy była dwa razy starsza od małpy w czasie, gdy...” możemy zapisać jako:

$$z - A = 2(x - A)$$

przy czym A oznacza jakąś (niekoniecznie całkowitą) liczbę lat, ponieważ zdanie odnosi się do przeszłości. W podobny sposób możemy odczytać pozostałe zdania tego akapitu. Akapit ten mówi nam, że dla pewnych A , B i C prawdziwe są następujące równania:

$$z - A = 2(x - A)$$

$$z - A = \frac{1}{2}(x + B)$$

$$x + B = 3(z - C)$$

$$z - C = 3x$$

Mogimy to sprowadzić do

$$4z = 13x \quad (\text{XIII.2})$$

Drugi akapit jest bardzo zbliżony do trzeciego i może być odczytany w podobny sposób. (Zauważ, że waga liny wynosi $r/4$). Z akapitu tego dowiadujemy się, że dla pewnych A , B i C prawdziwe są następujące równania:

$$W - \frac{1}{4}r = x - A$$

$$z - A = 2(y - B)$$

$$z - B = \frac{1}{2}(y + C)$$

$$y + C = 3(z - D)$$

$$z - D = 3x$$

a zatem

$$W - \frac{1}{4}r = 11x - 2x \quad (\text{XIII.3})$$

Pierwszy akapit dostarcza nam więcej informacji. Po pierwsze, małpa i jej matka mają razem cztery lata:

$$x + z = 4 \quad (\text{XIII.4})$$

Po drugie, waga małpy i liny wynosi półtora raza więcej niż wiek matki małpy:

$$W + \frac{1}{4}r = \frac{3}{2}z \quad (\text{XIII.5})$$

Po trzecie, waga małpy wynosi W , ponieważ małpa i ciężarek pozostawały w stanie równowagi. Z równań (XIII.1), (XIII.2) i (XIII.4) otrzymujemy:

$$x = \frac{16}{17}$$

$$y = 2$$

$$z = \frac{52}{17}$$

Następnie z równań (XIII.3) i (XIII.5) wynika, że:

$$W - \frac{1}{4}r = \frac{72}{17} \quad \text{oraz} \quad W + \frac{1}{4}r = \frac{78}{17}$$

co daje $r = 12/17$. Oznacza to, że długość liny wynosiła $12/17$ stopy.

Bez wątpienia mogą Ci teraz przychodzić do głowy różne pomysły, co można zrobić z autorami tej zagadki za pomocą tej liny! Na szczęście ma ona tylko $8\frac{1}{2}$ cala długości!!

Nawet jeśli problem jest przedstawiony jasno i dokładnie, ze wszystkimi niezbędnymi specyfikacjami, nadal możliwe jest przeoczenie niektórych oczywistych rzeczy i popełnienie głupiego błędu podczas modelowania problemu (zob. rozdział 2). Sytuację taką ilustruje kolejna zagadka.

Na zewnątrz pokoju znajdują się trzy przełączniki sterujące trzema żarówkami znajdującymi się w pokoju. Każda żarówka wisie na długim kablu, trzeba więc uważać, żeby nie uderzyć w nie głową przy wchodzeniu do pokoju! Wiemy, że każdy przełącznik jest połączony tylko do jednej żarówki. Wiemy też, że wszystkie przełączniki są w pozycji „wyłączony” i w pokoju jest ciemno. Teraz zamykamy drzwi i nie widzimy już wnętrza pokoju.

Twoim zadaniem jest ustalenie, który przełącznik jest połączony z którą żarówką, ale przy dość poważnym ograniczeniu: masz możliwość wykonania

tylko jednej próby. Możesz ustawić przełączniki w dowolnej pozycji i wejść do pokoju. Możesz obejrzeć go dokładnie i na tej podstawie możesz rozwiązać zagadkę. Zauważ, że nie wolno Ci manipulować przy przełącznikach po wejściu do pokoju. Możesz z nimi zrobić co chcesz, ale tylko przed otwarciem drzwi do pokoju.

Z początku problem ten może Ci się wydawać „niemożliwy”. Przecież jeśli nie dotkniesz żadnego z przełączników, wejście do pokoju nie będzie miało sensu; wiadomo bowiem, że jest w nim ciemno, nie będzie więc żadnej możliwości stwierdzenia, który przełącznik jest połączony z którą żarówką. Tak więc musisz zrobić coś z przełącznikami. Tylko co?

Jeśli ustawisz tylko jeden z przełączników w pozycji „włączony” i otwryszy drzwi, to oczywiście jedna z żarówek będzie się świecić, nie będziesz jednak mógł stwierdzić, który z dwóch pozostałych przełączników steruje którą z nieświecących żarówek.

Podobnie, jeśli ustawisz dwa z przełączników w pozycji „włączony”, to po wejściu do pokoju nieświecąca żarówka wskaże Ci, który z przełączników jej odpowiada, ale nie będziesz mógł stwierdzić, która z dwóch świecących żarówek odpowiada któremu z ustawionych przez Ciebie przełączników. To dopiero dylemat.

Klucz do rozwiązania tkwi we właściwym modelu problemu. Przedstawione dotychczas rozumowanie jest oparte na modelu wykorzystującym tylko jedną właściwość żarówek, mianowicie to, czy są one włączone, czy nie. Żarówki mają jednak również inne właściwości. Jakie? A co z temperaturą? Opis problemu nie ograniczał naszej uwagi do świecenia żarówek. W rzeczywistości to tylko część Twojego rozumienia tego problemu.

Jeśli do naszego modelułączymy temperaturę żarówek, rozwiązanie staje się naprawdę proste. Jedyne, co musisz zrobić, to ustawić dwa przełączniki w pozycji „włączony”, odczekać kilka minut, a następnie wyłączyć jeden z nich. Gdy wejdziesz do pokoju, jedna z żarówek będzie świecić, a dwie pozostałe nie. Jedna z nieświecących żarówek będzie jednak gorąca, a to z pewnością umożliwi Ci bezbłędnie odpowiedzieć na postawione w zagadce pytanie. Uważaj tylko na palce i nie poparz się!

13. Systemy rozmyte

Czasem sobie myślę, że najpewniejszym znakiem tego,
że gdzieś we wszechświecie istnieje intelligentne życie,
jest to, że *nikt* nie próbował się z nami skontaktować.

Bill Watterson: *Calvin and Hobbes*

Przed nastaniem ery komputerów obliczenia wykonywano za pomocą suwaków logarytmicznych. Z konieczności odpowiedź prawie zawsze nie była precyzyjna i brzmiała „wystarczająco dobrze”, gdyż wzrok i sprawność rąk nie były wystarczające, żeby za pomocą suwaka uzyskać dowolną liczbę cyfr znaczących. Pomnożenie, powiedzmy, π przez e wymagało ustawienia podziałki suwaka na około 3,14, a następnie odczytania odpowiedzi w okolicach 2,72 na górnej skali. Przy odrobinie wprawy można było dawać odpowiedzi z dokładnością występującą w pierwotnym zadaniu (tutaj były to dwa miejsca po przecinku). Nikt nie mógł spodziewać się dokładniejszego dostrojenia. Być może, stąd wzięło się powiedzenie „wystarczająco dokładnie jak na państwowe roboty”. W tym świetle dzisiejsza precyzja nowoczesnych komputerów cyfrowych jest wielkim skarbem.

Gdy ludzie komunikują się ze sobą, rzadko robią to z możliwie największą dokładnością. Nie jest to po prostu praktyczne. Na przykład, grając na giełdzie, słyszy się powiedzenie: *kupuj tanio, sprzedawaj drogo*. To dobra rada, ale jak tanio jest „tanio” i jak drogo jest „drogo”? Jeśli odpowiedź brzmi, powiedzmy, że cena 10 dolarów za akcje to cena niska, to czy 10,01 dolara to nadal niska cena? A co jeśli chodzi o 10,02 czy 11, czy 20 dolarów? Spodziewanie się, że ktoś będzie zachowywał się tak, jakby istniała jakaś wyraźna linia oddzielająca ceny niskie od wysokich, jest nierozsądne. Ludzie rozumują i działają na pojęciach opartych na własnym rozumieniu stopnia, do jakiego te pojęcia reprezentują określone warunki. Takie słowa, jak nisko, wysoko, blisko, bardzo stary, czerwony, wcześnie itp. mają swoje ogólne znaczenie rozumiane przez wszystkich. Znaczenia te są nieprecyzyjne, lecz mimo to użyteczne. Próba nadania precyzyjnego znaczenia każdemu terminowi nie jest jedynie niepraktyczna, jest ona naprawdę niemożliwa do zrealizowania, gdyż każde pojęcie oznacza dla każdego człowieka trochę co innego. Określenie „w średnim wieku” ma różne znaczenie

i obejmuje ludzi, którzy mają poniżej 30 lat, ale też takich po pięćdziesiątce. Skuteczna komunikacja nie tylko zyskuje na nieprecyzyjnych opisach, ale wręcz wymaga ich istnienia.

Naturalne jest rozważenie możliwości opracowywania algorytmów, które traktowałyby pojęcia w taki sam sposób jak ludzie, w istocie wykonując obliczenia na słowach, a nie na liczbach. Ale wszystkie obliczenia sprowadzają się w końcu do operacji na liczbach. Musimy zatem poszukiwać matematycznego opisu obejmującego stosowanie przez nas pojęć, które nie są czarno-białymi wyrażeniami boolowskimi, ale mają niewyraźny, niejednoznaczny, *rozmyty* charakter.

13.1. Zbiory rozmyte

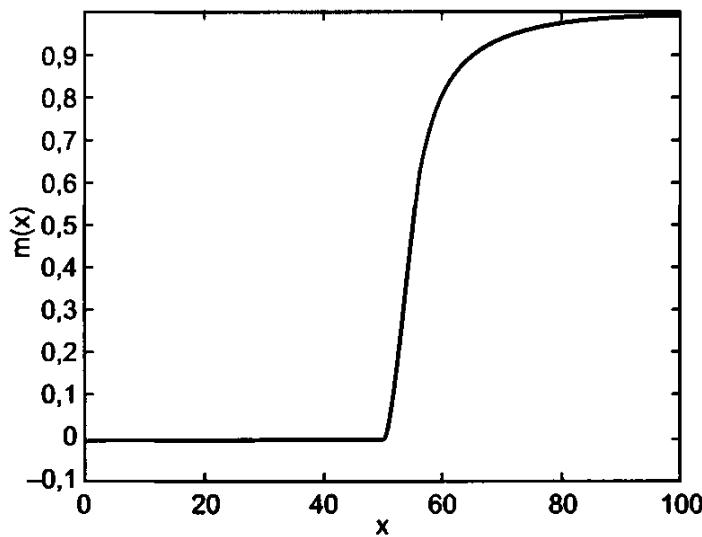
Pierwsze podstawy działań na rozmytych opisach świata rzeczywistego podał Lotfi Zadeh w 1965 roku [505]. Jego ujęcie to w istocie proste rozszerzenie klasycznej teorii zbiorów. W klasycznym zbiorze elementy albo należą, albo nie należą do tego zbioru. Zbiór parzystych liczb naturalnych mniejszych od 10 jest ściśle określonym zestawem czterech elementów $\{2, 4, 6, 8\}$. Alternatywnie zbiór ten mógłby być opisany jako funkcja charakterystyczna $m_A(x)$ zdefiniowana nad jakąś „przestrzenią rozważyń”, w której przybiera wartość 1, gdy argument x jest elementem zbioru A , i wartość 0 w przeciwnym razie. Zbiór rozmyty daje możliwość, żeby $m_A(x)$ przyjmowała wartości inne niż $\{0, 1\}$, powiedzmy z zakresu $[0, \alpha]$, przy czym każda wartość oznacza stopień przynależności x do A . Dla zachowania spójności możliwy zakres wartości jest często skalowany do zakresu liczb rzeczywistych między 0 a 1 włącznie. Wartość $m_A(x) = 1$ oznacza, że element x na pewno należy do A . Wartość $m_A(x) = 0$ oznacza, że element x na pewno nie należy do A . Wartości pośrednie oznaczają pośrednie stopnie przynależności do tego zbioru. To właśnie ten pośredni zakres reprezentuje rozszerzenie zbiorów rozmytych poza klasyczne zbiory „ostre” (nierożmyte).

Często wygodnie jest opisywać funkcje charakterystyczne, nazywane w przypadku zbiorów rozmytych *funkcjami przynależności*, jako odwzorowania ciągłe, a nie jako pary $(x, m_A(x))$. Na przykład funkcja przynależności opisująca zbiór rozmyty określający wiek ludzi uznanych za „starych” może być następująca:

$$\begin{aligned} m_A(x) &= 0 && \text{jeśli } x \leq 50 \\ m_A(x) &= \left(1 + \frac{25}{(x-50)^2}\right)^{-1} && \text{jeśli } x > 50 \end{aligned}$$

(rys. 13.1) zgodnie z tym, co zaproponował Zadeh [506]. Zauważ, że nie jest to jedyna możliwa funkcja przynależności opisująca wiek „stary”. W rzeczywistości nie ma dobrej odpowiedzi na pytanie, jaką funkcję tutaj dobrąć. Wybór jest tutaj subiektywny: każdy może zaproponować funkcję przynależności, która odpowiada jego własnemu punktowi widzenia. Cecha ta jest zaletą systemów rozmytych, gdyż umożliwiają one łatwiejsze odzwierciedlenie indywidualnego nastawienia do danego problemu i dzięki temu można je dopasować do

punktu widzenia każdego człowieka. Stwarza ona jednak także wyzwanie dla tych, którzy chcą wykorzystywać takie funkcje przynależności do obliczeń.



Rysunek 13.1. Jedna z możliwych funkcji przynależności zbioru rozmytego określającego wiek ludzi uznanych za „starych”. Funkcja ma wartość 0, gdy wiek człowieka wynosi mniej niż 50 lat, po tym progu gwałtownie wzrasta do 1

13.2. Zbiory rozmyte i miary probabilistyczne

Ponieważ wartości funkcji przynależności są zwykle znormalizowane do przedziału $[0, 1]$, który stanowi też zakres wartości dla prawdopodobieństw, często pojawia się okazja do popełnienia pomyłki dotyczącej związku między zbiorami rozmytymi a miarami probabilistycznymi. W pierwszej chwili można uznać, że pojęcia te dotyczą tego samego, że miary rozmyte to po prostu jeszcze jedno określenie prawdopodobieństw. Takie podejście nie jest jednak poprawne: rozmytość i prawdopodobieństwo mają dwie istotnie różne interpretacje.

Doskonały przykład podali Bezdek i Pal [44]. Przypuśćmy, że byłeś spragniony i ktoś zaproponował Ci wybór między dwiema butelkami z płynami. Powiedziano Ci, że dla pierwszej butelki funkcja przynależności zbioru rozmytego wszystkich płynów pitnych wynosi 0,91. Ponadto powiedziano Ci, że prawdopodobieństwo, iż druga butelka zawiera nadający się do picia płyn, wynosi też 0,91. Z której butelki byś się napił? Już fakt, że musisz podjąć decyzję, od razu świadczy o tym, iż istnieje jakaś różnica w interpretacji; w przeciwnym wypadku nie trzeba byłoby podejmować żadnej decyzji. Prześledźmy ten przykład głębiej, żeby lepiej zrozumieć wszystkie konsekwencje.

Płyn, którego stopień przynależności do zbioru płynów nadających się do picia wynosi 0,91, może niezbyt dobrze smakować, ale przy jakiejkolwiek rozsądnej interpretacji nie będziemy się spodziewać, że będzie on trujący. Z kolei płyn, o którym wiemy, że z prawdopodobieństwem 0,91 jest zdatny do picia, jest także z prawdopodobieństwem 0,09 (będącym dopełnieniem tego pierwszego) niezdany do picia, gdyż być może jest zmieszany z arszenikiem. Przewidujemy,

że pijąc z pierwszej butelki, nie otrujemy się. Nie możemy tego samego powiedzieć w przypadku drugiej butelki. Co więcej, po sprawdzeniu zawartości obu butelek, na przykład przez namówienie odważnego kolegi na napicie się z obu, interpretacja probabilistyczna dla drugiej butelki musi ulec rozstrzygnięciu – albo płyn w butelce nadaje się do picia, albo nie. Prawdopodobieństwo, że nadaje się do picia, staje się równe 1 albo 0. Rozmyty opis płynu w pierwszej butelce może po wykonaniu próby pozostać taki sam. Widać zatem wyraźnie, że opisy probabilistyczne nie są tym samym co opisy rozmyte.

13.3. Operacje na zbiorach rozmytych

Podobnie jak w przypadku klasycznych zbiorów, musimy być w stanie działać na zbiorach rozmytych i umieć opisywać ich przecięcie, sumę, dopełnienia itp. Jest to szczególnie ważne, gdyż chcielibyśmy móc użyć złożonych opisów lingwistycznych w sposób matematyczny. Na przykład kwiat, który jest żółty i pachnący, należy do zbioru żółtych przedmiotów i do zbioru pachnących przedmiotów. „Żółty pachnący” kwiat powinien zatem należeć do zbioru żółtych pachnących przedmiotów, który jest przecięciem dwóch wspomnianych zbiorów. Istnieje kilka alternatywnych sposobów definiowania operacji na zbiorach rozmytych, odzwierciedlających te własności. Dla jasności skupimy się tutaj na sformułowaniu podanym w [505].

- Zbiór rozmyty jest *pusty*, wtedy i tylko wtedy, gdy jego funkcja przynależności jest równa zeru dla wszystkich elementów przestrzeni rozważyń X (dla wszystkich możliwych argumentów tej funkcji).
- Dwa zbiory rozmyte A i B są *równe*, $A = B$, wtedy i tylko wtedy, gdy $m_A(x) = m_B(x)$ dla wszystkich $x \in X$. Dla wygody można to zapisać jako $m_A = m_B$.
- *Dopełnienie* zbioru rozmytego A jest oznaczane A' i jest zdefiniowane jako $m_{A'} = 1 - m_A$.
- Zbiór rozmyty A jest *zawarty w* lub jest *podzbiorem* B wtedy i tylko wtedy, gdy $m_A \leq m_B$.
- *Suma* dwóch zbiorów rozmytych A i B jest zbiorem rozmytym $C = A \cup B$ o funkcji przynależności $m_C(x) = \max\{m_A(x), m_B(x)\}$ dla $x \in X$.
- *Przecięcie (iloczyn)* dwóch zbiorów rozmytych A i B jest zbiorem rozmytym $C = A \cap B$ o funkcji przynależności $m_C(x) = \min\{m_A(x), m_B(x)\}$ dla $x \in X$.

Wracając do naszego przykładu z żółtym pachnącym kwiatem, przyopuszcmy, że dla tego kwiatu funkcje przynależności dają $m_{\text{Żółte}}(x) = 0,8$ i $m_{\text{Pachnace}} = 0,9$. Wtedy wartość funkcji przynależności dla zbioru rozmytego żółtych pachnących kwiatów byłaby $\min\{0,8, 0,9\} = 0,8$. Z kolei wartość funkcji przynależności dla zbioru rozmytego żółtych lub pachnących kwiatów byłaby $\max\{0,8, 0,9\} = 0,9$.

Dla zbioru rozmytego kwiatów, które nie są żółte, otrzymalibyśmy $1 - 0,8 = 0,2$. Tak wyglądają podstawowe mechanizmy łączenia opisów rozmytych.

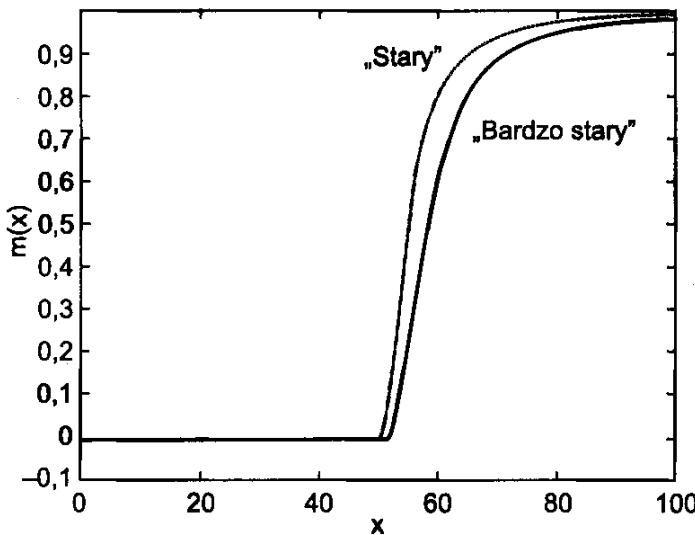
Innym często spotykanym określeniem rozmytym jest słowo „bardzo”, które działa jako wzmacnienie warunku (np. bardzo stary, bardzo cichy). Zastosowanie określenia „bardzo” do opisu rozmytego intuicyjnie powinno dawać odpowiedni efekt wzmacniający dla funkcji przynależności. W wielu wypadkach możemy to wprowadzić w sposób jawnny. Rozważmy, na przykład, ponownie funkcję przynależności dla zbioru rozmytego A , określającego wiek ludzi uznanych za starych:

$$\begin{aligned} m_A(x) &= 0 && \text{jeśli } x \leq 50 \\ m_A(x) &= \left(1 + \frac{25}{(x-50)^2}\right)^{-1} && \text{jeśli } x > 50 \end{aligned}$$

Jeśli powiemy, że A^* jest zbiorem rozmytym elementów, które są „bardzo A ”, to taki nowy zbiór możemy zdefiniować jako

$$m_{A^*}(x) = (m_A(x))^2 \quad \text{dla wszystkich } x \in X$$

(rys. 13.2). Wynikiem podniesienia do kwadratu funkcji przynależności jest tutaj intensyfikacja stopnia przynależności. Otrzymanie podobnego stopnia przynależności do A^* co dla A wymaga użycia bardziej skrajnej wartości x .



Rysunek 13.2. Zastosowanie terminu stopniującego „bardzo” jako modyfikatora wcześniejszej podanej funkcji przynależności zbioru rozmytego określającego wiek ludzi uznanych za „starych”. Słowo „bardzo” zostało nadane takie znaczenie, które odpowiada podniesieniu do kwadratu funkcji przynależności podstawowego zbioru rozmytego. Wartości funkcji przynależności dla ludzi w wieku powyżej 50 lat w zbiorze rozmytym dotyczącym ludzi uznanych za „bardzo starych” jest mniejsza niż odpowiednia wartość w zbiorze ludzi „starych”. Intuicyjnie, jeśli ktoś ma wiek o pewnej dodatniej wartości funkcji przynależności w zbiorze obejmującym ludzi „starych”, to inna osoba musi być o wiele starsza, żeby jej wiek miał ten sam stopień przynależności w zbiorze ludzi „bardzo starych”

Inne powszechnie stosowane terminy lingwistyczne także można opisać matematycznie za pomocą zbiorów rozmytych.

- Jeśli zbiór A jest bardzo niedokładny, można to opisać jako $A = \text{bardzo (nie dokładny)}$, co oznacza wzmacnianie dopełnienia zbioru rozmytego „dokładny”. W związku z tym funkcję przynależności zbioru A możemy tu zdefiniować jako $m_A = (1 - m_{\text{Dokładny}})^2$. Zauważ, że nie jest to ten sam zbiór rozmyty co zbiór $A = \text{nie bardzo dokładny}$, który jest dopełnieniem zbioru „bardzo dokładny”, dla którego $m_A = 1 - m_{\text{Dokładny}}^2$.
- Zadeh w [506] wprowadził dodatkowe pojęcia „plus” i „minus”, które służą za łagodniejsze wersje „bardzo”:

$$\begin{aligned}m_{A+}(x) &= m_A(x)^{1,25} \\m_{A-}(x) &= m_A(x)^{0,75}\end{aligned}$$

Zdefiniował on też pojęcie „wysoce” jako z grubsza równoważne „minus bardzo bardzo” oraz „plus plus bardzo”.

Takie modyfikatory są nazywane *modyfikatorami stopniującymi*. Używa się ich powszechnie w codziennym języku i można je systematycznie stosować do modyfikowania zbiorów rozmytych w sposób odpowiadający ich znaczeniu.

13.4. Relacje rozmyte

Chociaż powyżej wspomniane opisy rozmyte mogą zapewnić określenie szerskiego zakresu użytkowych terminów oraz związanych z nimi wyrażeń matematycznych opisujących takie wartości lingwistyczne, nie umożliwiają jednak wprowadzenia naturalnego rozszerzenia do wyrażeń warunkowych. To znaczy, przypuśćmy, że chcielibyśmy zająć się przypadkiem, w którym

Jeśli A , to B

Jeśli, na przykład, droga jest oblodzona, to prowadzenie samochodu jest niebezpieczne, czyli ujmując to bardziej zwięzle, jeśli *oblodzone*, to *niebezpieczne*. Określenia oblodzone i niebezpieczne oznaczają zbiory rozmyte, a warunek *jeśli-to* oznacza relację między tymi zbiorami. Przestrzeń rozważań dla A i B nie musi być taka sama, a zatem nowa przestrzeń rozważań, która opisuje relację między A i B , jest iloczynem kartezjańskim $A \times B$ wszystkich par (a, b) takich, że $a \in A$ oraz $b \in B$. Bardziej konkretnie, jeśli R jest relacją rozmytą między zbiorem rozmytym A a zbiorem rozmytym B , to R jest podzbiorem rozmytym w iloczynie kartezjańskim $A \times B$, przy czym każdej parze (a, b) jest przypisana pewna wartość funkcji przynależności $m_R(a, b)$.

Przypuśćmy, że $A = \text{małe liczby nad przestrzenią rozważań } U = \{1, 2\}$, a $B = \text{duże liczby nad przestrzenią rozważań } V = \{1, 2, 3\}$. Wartości funkcji przynależności dla elementów w A są zadane za pomocą par uporządkowanych $(a, m_A(a))$ jako $(1, 1)$ i $(2, 0,6)$ oraz podobnie dla B : $(1, 0,2)$, $(2, 0,5)$, $(3, 1)$. Teraz $R = A \times B$ jest określona w ten sposób, że przynależność równa

się $\min\{m_A(a), m_B(b)\}$ dla każdej pary (a, b) . Można to dogodnie przedstawić w postaci tablicy (macierzy):

	1	2	3
1	0,2	0,5	1,0
2	0,2	0,5	0,6

Zatem R jest podzbiorem rozmytym w $U \times V$, który opisuje przynależność każdego z jego elementów składowych (u, v) za pomocą relacji między zbiorami rozmytymi A i B .

Relacje rozmyte możemy także łączyć. Jeśli R jest relacją rozmytą między X a Y , a S jest relacją rozmytą między Y a Z , to złożenie relacji rozmytych R i S jest zdefiniowane na parach $(x, z) \in X \times Z$ jako

$$R \circ S = \max_y [\min\{m_R(x, y), m_S(y, z)\}]$$

Jeśli dziedziny zbiorów X , Y i Z są skończone, to $R \circ S$ nazywamy złożeniem typu *max-min* (maksyminowym) tablic relacji R i S . W złożeniu typu *max-min* operacje dodawania i mnożenia są zastąpione przez max i min. Po odrzuceniu algebraicznego opisu możemy teraz posługiwać się złożonymi rozmytymi wyrażeniami warunkowymi.

Przyjmijmy, że warunek „jeśli A , to B ” jest przypadkiem szczególnym warunku „jeśli A , to B , w przeciwnym razie C ”, przy czym B i C są zdefiniowane nad tą samą przestrzenią rozważyń V , a A jest zdefiniowane nad przestrzenią rozważyń U . Zdanie „jeśli A , to B , w przeciwnym razie C ” definiujemy jako

$$(A \times B) \cup (\sim A \times C)$$

a zatem prostsze „jeśli A , to B ” możemy uważać za nieokreślony zbiór rozmyty C . Jeśli zinterpretujemy to w ten sposób, że $(\sim A \times C)$ może być w zasadzie dowolnym podzbiorem rozmytym w $(U \times V)$, to możemy zdefiniować „jeśli A , to B ” jako $(A \times B)$, gdyż możemy wybrać $(\sim A \times C)$ tak, że

$$(A \times B) \cup (\sim A \times C) = (A \times B)$$

Jest to tak zwana *metoda przybliżonego wnioskowania Mamdaniego*. Zauważ, że nie jest to jedyna możliwa do przyjęcia definicja, gdyż moglibyśmy dokonać innych wyborów, żeby uzyskać taki sam wynik oddziaływanego wyrazu $(\sim A \times C)$.

Przyjrzymy się jednej z możliwości. W tym celu przypuśćmy, że przyjęliśmy A i B jak wyżej. Wiemy już zatem, że

	1	2	3
1	0,2	0,5	1,0
2	0,2	0,5	0,6

Ale co z $(\sim A \times X)$? Zbiór rozmyty $\sim A$ to $\{(1, 0), (2, 0,4)\}$, a zatem jeśli C jest dowolnym podzbiorem rozmytym w V , powiedzmy $\{(1, \alpha), (2, \beta), (3, \gamma)\}$, to

	1	2	3
1	0	0	0
2	$\min\{0,4, \alpha\}$	$\min\{0,4, \beta\}$	$\min\{0,4, \gamma\}$

Obliczenie $(A \times B) \cup (\sim A \times C)$ daje

	1	2	3
1	0,2	0,5	1,0
2	$\max[0,2, \min\{0,4, \alpha\}]$	0,5	0,6

Możemy stwierdzić bez żadnej niejednoznaczności, że $(2, 2)$ i $(2, 3)$ mają odpowiednio stopnie przynależności 0,5 i 0,6, gdyż $\min\{0,4, \beta\}$ i $\min\{0,4, \gamma\}$ nie będą nigdy większe od 0,5 i 0,6. Dla $(2, 1)$ nie wiemy jednak, czy 0,2 będzie większe od minimum z 0,4 i α . To jest ciekawy przypadek obliczenia rozmytego, w którym odpowiedź na pytanie, czym jest „jeśli A , to B ”, zależy w części od zbioru rozmytego C związanego z zależnością warunkową od dopełnienia A . Zauważ, że gdyby stopień przynależności dla $(2, 1)$ w $(A \times B)$ wynosił 0,4 lub więcej, to stopień przynależności dla $(2, 1)$ w $(A \times B) \cup (\sim A \times C)$ też byłby bez niejednoznaczności równy 0,4. Zgodnie z metodą Mamdaniego określania relacji *jeśli-to* skutek oddziaływanie $(\sim A \times C)$ nie ma żadnego znaczenia.

Ważne rozszerzenie zasady związanej z relacjami rozmytymi pojawia się, gdy dla konkretnego podzbioru rozmytego A chcemy określić wynikowy podzbiór rozmyty B , mając $R = A \times B$. Tutaj $B = A \circ R$ jest złożeniem typu max-min A i R . Na przykład, jeśli $A = \{(1, 1,0), (2, 0,6)\}$ oraz

$$R = \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0,2 & 0,5 & 1,0 \\ 2 & 0,2 & 0,5 & 0,6 \end{array}$$

jak wyżej, to $B = \{(1, 0,2), (2, 0,5), (3, 1,0)\}$.

Żeby zilustrować złożone rozmyte wyrażenie warunkowe, rozważymy następujący przykład z [506]. Niech przestrzenią rozważań będzie zbiór $U = \{1, 2, 3, 4, 5\}$ ze zbiorami rozmytymi

$$A = \text{„małe”} \equiv \{(1, 1,0), (2, 0,8), (3, 0,6), (4, 0,4), (5, 0,2)\}$$

$$B = \text{„duże”} \equiv \{(1, 0,2), (2, 0,4), (3, 0,6), (4, 0,8), (5, 1,0)\}$$

$$C = \text{„nie bardzo duże”} \equiv \{(1, 0,96), (2, 0,84), (3, 0,64), (4, 0,36), (5, 0)\}$$

Zauważ, że A i B są zbiorami rozmytymi zdefiniowanymi, natomiast C jest zbiorem rozmytym obliczonym, wyznaczonym jako dopełnienie zbioru rozmytego „bardzo duże”, dla którego wartości funkcji przynależności są określone jako

$$m_{\text{bardzo } B}(x) = m_B^2(x)$$

Przypuśćmy, że chcemy znaleźć relację R , która opisuje rozmyte zdanie warunkowe: „Jeśli x jest małe, to y jest duże, w przeciwnym razie y jest nie bardzo duże”. Innymi słowy: „Jeśli A , to B , w przeciwnym razie C ”.

Z powyższych opisów wynika, że R jest określone jako $(A \times B) \cup (\sim A \times C)$. Najpierw musimy rozwiązać oddzielnie każdy składnik tego złożonego wyrażenia. $(A \times B)$ jest określone macierzą

	1	2	3	4	5
1	0,2	0,4	0,6	0,8	1,0
2	0,2	0,4	0,6	0,8	0,8
3	0,2	0,4	0,6	0,6	0,6
4	0,2	0,4	0,4	0,4	0,4
5	0,2	0,2	0,2	0,2	0,2

$(\sim A \times C)$ jest określone macierzą

	1	2	3	4	5
1	0	0	0	0	0
2	0,2	0,2	0,2	0,2	0
3	0,4	0,4	0,4	0,36	0
4	0,6	0,6	0,6	0,36	0
5	0,8	0,8	0,64	0,36	0

Sumę tych dwóch zbiorów uzyskujemy, biorąc wynik max z wartości odpowiednich elementów macierzy, zatem

	1	2	3	4	5
R =	1	0,2	0,4	0,6	0,8
	2	0,2	0,4	0,6	0,8
	3	0,4	0,4	0,6	0,6
	4	0,6	0,6	0,6	0,4
	5	0,8	0,8	0,64	0,36

Przypuśćmy, że teraz pytamy się, jaka będzie funkcja przynależności dla Y , jeśli X to *bardzo małe*? (Przypomnij sobie, że „bardzo małe” jest określone przez wzięcie kwadratu wartości funkcji przynależności powyżej podanego zbioru rozmytego „małe”). Możemy to wyznaczyć, znajdując złożenie typu max-min X i R :

$$X \circ R = [1,0 \ 0,64 \ 0,36 \ 0,16 \ 0,04] \circ R = [0,36 \ 0,4 \ 0,6 \ 0,8 \ 1,0]$$

A zatem, jeśli X jest *bardzo małe*, to zgodnie z regułą „Jeśli X jest małe, to Y jest duże, w przeciwnym razie Y jest nie bardzo duże”, funkcja przynależności dla wynikowego Y jest odrobinę bliższa *duże* niż *nie bardzo duże*. Przykład ten prowadzi naturalnie do rozszerzenia, w którym reguły rozmyte są stosowane do sytuacji często pojawiających się w systemach sterowania, gdzie do rozpoznawania warunków i wskazywania odpowiednich sposobów postępowania stosuje się wiele reguł rozmytych.

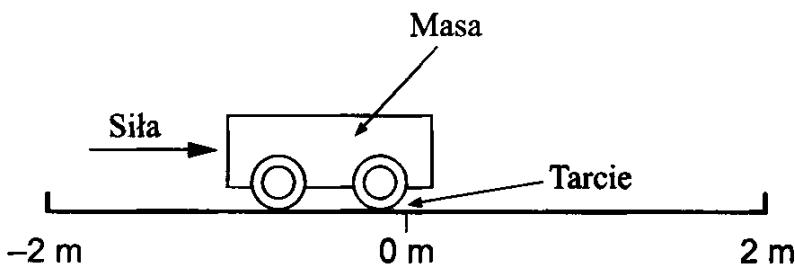
13.5. Projektowanie regulatora rozmytego

Połączenie wielu reguł rozmytych opisanych w poprzednim podrozdziale umożliwia projektowanie skomplikowanych algorytmów, które radzą sobie z rzeczywistymi problemami. Jeden z takich problemów pojawia się przy sterowaniu procesami. W takich problemach jest wymagane zarezerwowanie odpowiedniej

ilości zasobów, aby było możliwe uzyskanie określonego działania systemu. Oto kilka przykładów.

- Utrzymanie samolotu w prostym i poziomym locie.
- Utrzymanie temperatury w pomieszczeniu na poziomie 20° Celsjusza.
- Utrzymanie pracy silnika samochodu na poziomie 800 obrotów na minutę na jałowym biegu oraz przy włączonej lub wyłączonej klimatyzacji.
- Koordynowanie ruchu wind w budynku tak, żeby na wszystkie żądania odpowiadały jak najszybciej.

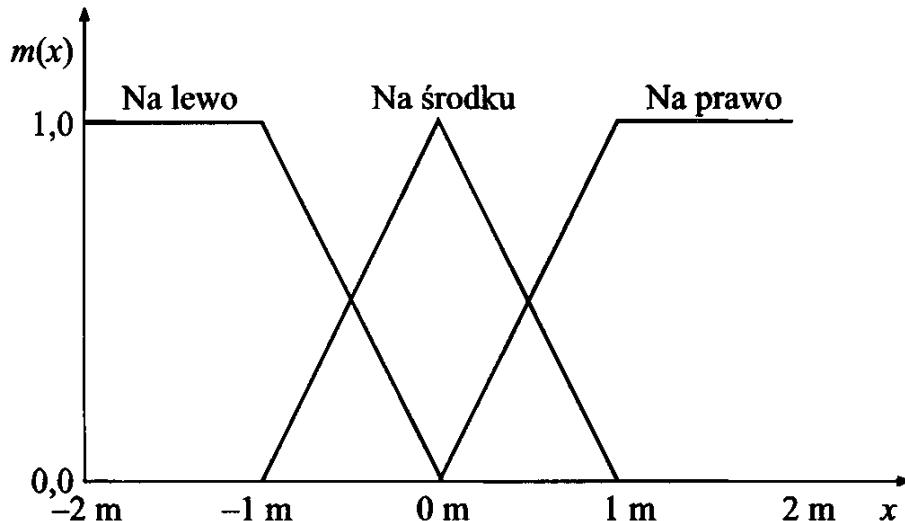
Każdy z tych systemów – samolot, pomieszczenie, samochód, windy – możemy opisać za pomocą równań różniczkowych, automatów ze skończoną liczbą stanów lub innych konstrukcji matematycznych. Następnie możemy użyć zbiorów reguł rozmytych do opisania stanu systemu oraz najlepszego sposobu postępowania, żeby uzyskać pożądane wyniki.



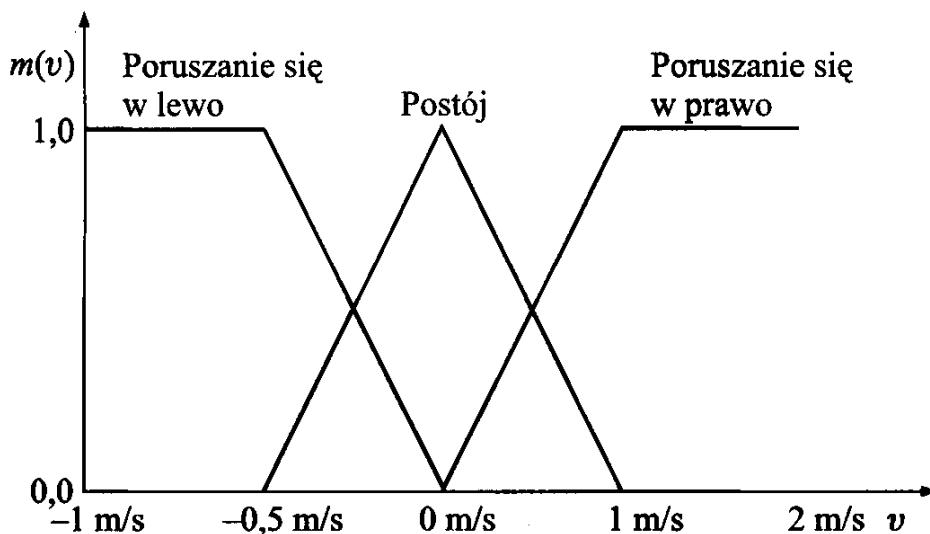
Rysunek 13.3. Wózek na czterometrowej platformie. Celem jest umieszczenie wózka z powrotem na środku platformy, robiąc to z zerową prędkością. Dostępne operacje sterujące to pchanie i ciągnięcie wózka, który ma swoją masę i współczynnik tarcia

Rozważmy następujący typowy przykład. Mamy wózek na czterometrowej platformie (rys. 13.3). Wózek ma pewną masę i współczynnik tarcia. Celem układu sterującego jest pchanie lub ciągnięcie wózka, żeby powrócił na środek platformy, być może, z dodatkowym celem, żeby nastąpiło to w minimalnym czasie. Interesują nas tu zmienne: x opisująca pozycję wózka, v opisującą prędkość wózka i u opisującą siłę nim sterującą. Możemy opisać pozycję wózka, korzystając ze zbiorów rozmytych *na lewo*, *na środku* lub *na prawo*. Na rysunku 13.4 przedstawiono możliwe funkcje przynależności dla tych terminów lingwistycznych, przy użyciu trójkątnych i trapezoidalnych krzywych. Podobnie, prędkość wózka może być opisana jako *poruszanie się w lewo*, *postój*, *poruszanie się w prawo*. Na rysunku 13.5 przedstawiono odpowiednie funkcje przynależności zbiorów rozmytych. Wreszcie siła, jaką stosujemy, może być opisana jako *ciągnięcie*, *brak siły* oraz *pchanie*, przy czym *ciągnięcie* oznacza stosowanie siły w celu przesuwania wózka w lewo i podobnie *pchanie* oznacza stosowanie siły w celu przesuwania wózka w prawo. Funkcje przynależności dla siły przykładowej do wózka pokazano na rys. 13.6. W końcu na potrzeby tego przykładu założymy, że jedyne trzy możliwości stosowania siły opisują następujące rozmyte wartości: *pchanie* lub *ciągnięcie z siłą jednego niutona*, lub *niestosowanie siły w ogóle*. Przy obliczaniu rozmytych działań sterujących nie będziemy tutaj

rozoważali wartości pośrednich, ale będziemy w dalszym ciągu mogli pchać lub ciągnąć wózek z dowolną siłą między plus a minus jednym niutonem.



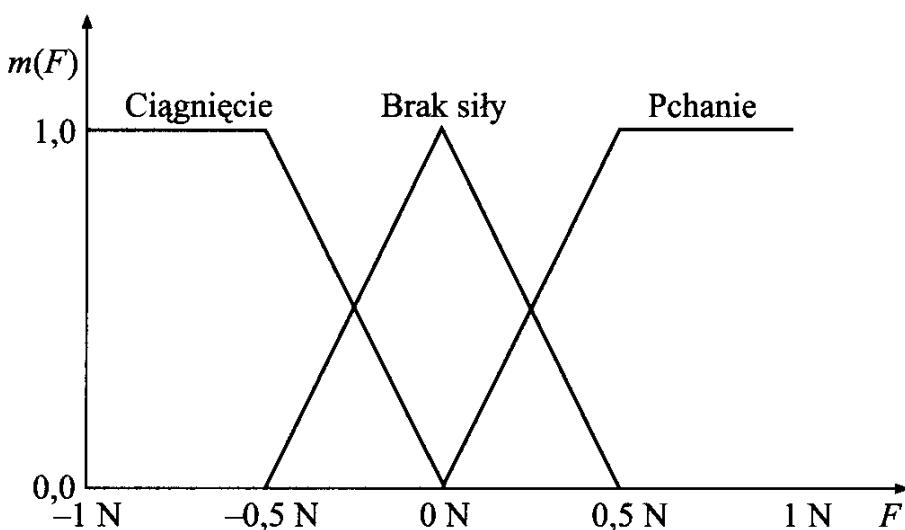
Rysunek 13.4. Potencjalny zbiór funkcji przynależności opisujących pozycję wózka jako „na lewo”, „na środku”, „na prawo”. Zauważ, że z rysunku nakładających się trapezoidalnych i trójkątnych funkcji wynika, że wózek może jednocześnie należeć do dwóch spośród trzech zbiorów



Rysunek 13.5. Potencjalny zbiór funkcji przynależności opisujących prędkość wózka jako „poruszanie się w lewo”, „postój” i „poruszanie się w prawo”

Mając dane te funkcje przynależności, musimy następnie opracować sensowne reguły ich stosowania. Na przykład, jeśli wózek jest w pobliżu pozycji *na środku* i ma *postój*, to siła, jaką powinniśmy zastosować, to *brak siły*. To pierwsza reguła. A oto kilka innych reguł, które możemy tu rozważyć:

- Jeśli *na lewo*, to *pchanie*.
- Jeśli *na prawo*, to *ciągnięcie*.
- Jeśli *na środku*, to *brak siły*.
- Jeśli *poruszanie się w lewo*, to *pchanie*.
- Jeśli *postój*, to *brak siły*.



Rysunek 13.6. Potencjalny zbiór funkcji przynależności opisujących siłę przyłożoną do wózka jako „ciągnięcie”, „brak siły” i „pchanie”

- Jeśli poruszanie się w prawo, to ciągnięcie.
- Jeśli na lewo i poruszanie się w lewo, to pchanie.
- Jeśli na prawo i poruszanie się w prawo, to ciągnięcie.

Trudniej jest określić, co robić w przypadku, gdy wózek jest *na lewo*, ale wykonyuje *poruszanie się na prawo*. Czy powinniśmy pchać wózek? To może przypieszyć ruch wózka w kierunku środka, ale ponieważ wózek już porusza się w prawo, takie dodatkowe popchnięcie może spowodować przesunięcie wózka za środek na prawą stronę platformy. Na potrzeby tego przykładu ograniczmy nasze rozważania do powyższych dziewięciu reguł (ośmiu opatrzonych kropkami i jednej wymienionej na początku akapitu).

Dla dowolnego stanu wejściowego zdefiniowanego za pomocą wartości pozycji i prędkości wózka możemy dla każdej reguły podać rozmyty wynik. Pierwszy krok polega na określeniu wartości funkcji przynależności dla poprzednika każdej reguły. Poprzedniki w naszym przykładzie to albo pojedyncze zbiory rozmyte, albo złożone zbiory rozmyte, będące koniunkcją dwóch zbiorów rozmytych. Jeśli reguła ma tylko jeden zbiór rozmyty w poprzedniku, to wszystko, co musimy zrobić, to wyznaczyć wartość funkcji przynależności tego zbioru rozmytego dla bieżącego stanu systemu. Jeśli reguła ma więcej niż jeden zbiór rozmyty w poprzedniku, to wartość funkcji przynależności jest określona jako minimum z wartości funkcji przynależności dla obu zbiorów składowych.

Przypuśćmy, że wózek znajduje się na pozycji $x = -0,5$ metrów i stoi, $v = 0$. W związku z tym dla Reguły 1, „Jeśli na środku i ma postój, to brak siły”, funkcja przynależności dla $x = -0,5$ w zbiorze rozmytym *na środku* wynosi $m_{\text{na środku}}(x = -0,5) = 0,5$, a funkcja przynależności dla $v = 0$ w zbiorze rozmytym *postój* wynosi $m_{\text{postój}}(v = 0) = 1$. Stąd wartość funkcji przynależności poprzednika w tym wypadku wynosi 0,5. Dla Reguły 2, „Jeśli na lewo, to

pchanie", $m_{\text{na lewo}}(x = -0,5) = 0,5$, a zatem tyle wynosi wartość dla poprzednika. Wartości poprzedników dla pozostałych reguł to:

- Reguła 3: „Jeśli na prawo, to ciągnięcie” – $m_{\text{na prawo}}(x = -0,5) = 0$
- Reguła 4: „Jeśli na środku, to brak siły” – $m_{\text{na środku}}(x = -0,5) = 0,5$
- Reguła 5: „Jeśli poruszanie się w lewo, to pchanie” –
 $m_{\text{poruszanie się w lewo}}(v = 0) = 0$
- Reguła 6: „Jeśli postój, to brak siły” – $m_{\text{postój}}(v = 0) = 1$
- Reguła 7: „Jeśli poruszanie się w prawo, to ciągnięcie” –
 $m_{\text{poruszanie się w prawo}}(v = 0) = 0$
- Reguła 8: „Jeśli na lewo i poruszanie się w lewo, to pchanie” –
 $\min\{m_{\text{na lewo}}(x = -0,5), m_{\text{poruszanie się w lewo}}(v = 0)\} = 0$
- Reguła 9: „Jeśli na prawo i poruszanie się w prawo, to ciągnięcie” –
 $\min\{m_{\text{na prawo}}(x = -0,5), m_{\text{poruszanie się w prawo}}(v = 0)\} = 0$

Następnie, mając dane wszystkie wartości funkcji przynależności dla poprzedników, należy je powiązać z wartościami funkcji przynależności dla następców (tzn. zbiorów rozmytych określających akcję, jaka ma być wykonana w wyniku zastosowania reguły). Powiązanie to znowu odbywa się na zasadzie obliczania, reguła po regule, minimum między funkcją przynależności związaną z każdą możliwą akcją a wartością funkcji przynależności poprzednika. W rezultacie otrzymujemy kolekcję zbiorów rozmytych opisujących wynik na podstawie każdej reguły – obliczone w ten sposób wartości podano w tab. 13.1.

Trzeci krok polega na zebraniu wszystkich wyników z poszczególnych reguł. Uzyskujemy to przez wzięcie dla każdego stanu (tutaj $-1, 0$ lub 1) maksymalnej wartości funkcji przynależności następcnika uwzględniającej wszystkie wyniki reguł. Daje to $(-1, 0), (0, 1), (1, 0,5)$. Jest to zbiór rozmyty wszystkich możliwych akcji (działan) ciągnięcia, nieprzykładania siły i pchania wózka przy danych warunkach wejściowych oraz danych regułach.

Krok końcowy polega na „defuzyfikacji” uzyskanego zbioru rozmytego, żeby uzyskać jedną (nieroźmytą) wartość wynikową. Istnieje wiele sposobów na osiągnięcie tego. Mizumoto [325] naszkicował 11 różnych procedur defuzyfikacji, które zaproponowano w literaturze. Najczęściej jest stosowana metoda „środka obszaru”, w której korzysta się ze stosunku

$$\frac{\sum_{i=1}^n F(y_i)y_i}{\sum_{i=1}^n F(y_i)}$$

gdzie: y_i jest i -tym możliwym elementem w zbiorze wynikowym Y (tutaj $-1, 0$ lub 1), przy czym takich elementów jest n , a $F(y_i)$ jest wartością funkcji przynależności związaną z każdą z tych wartości. Mamy na przykład

$$\frac{(0)(-1) + (1)(0) + (0,5)(1)}{0 + 1 + 0,5} = \frac{0,5}{1,5} = \frac{1}{3}$$

Tabela 13.1. W rezultacie otrzymujemy kolekcję zbiorów rozmytych opisujących wynik na podstawie każdej reguły. Ograniczamy się tutaj do rozważania tylko trzech możliwych wartości stosowanej siły: -1 , 0 lub 1 niuton. Następniki wskazują stopień przynależności każdej z opcji do odpowiedniego zbioru rozmytego

	<i>Wartość funkcji przynależności poprzednika</i>	<i>Następnik</i>
Reguła 1:	0,5	<i>brak siły</i> $\{(-1, 0), (0, 1), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 0,5), (1, 0)\}$	
Reguła 2:	0,5	<i>pchanie</i> $\{(-1, 0), (0, 0), (1, 1)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0,5)\}$	
Reguła 3:	0	<i>ciągnięcie</i> $\{(-1, 1), (0, 0), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0)\}$	
Reguła 4:	0,5	<i>brak siły</i> $\{(-1, 0), (0, 1), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 0,5), (1, 0)\}$	
Reguła 5:	0	<i>pchanie</i> $\{(-1, 0), (0, 0), (1, 1)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0)\}$	
Reguła 6:	1	<i>brak siły</i> $\{(-1, 0), (0, 1), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 1), (1, 0)\}$	
Reguła 7:	0	<i>ciągnięcie</i> $\{(-1, 1), (0, 0), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0)\}$	
Reguła 8:	0	<i>pchanie</i> $\{(-1, 0), (0, 0), (1, 1)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0)\}$	
Reguła 9:	0	<i>ciągnięcie</i> $\{(-1, 1), (0, 0), (1, 0)\}$
Wynik:	$\{(-1, 0), (0, 0), (1, 0)\}$	

Oznacza to, że wynik reguł sterowania rozmytego sugeruje, żeby użyć siły $1/3$ niutona do popchania wózka. Jeśli weźmiemy pod uwagę pozycję wózka, to działanie to wydaje się sensowne.

Projektowanie regulatora rozmytego wymaga wielokrotnego poprawiania reguł oraz funkcji przynależności zbiorów w nich używanych. Jest bardzo mało prawdopodobne, że uda Ci się opracować idealny regulator rozmyty dla złożonego, niestabilnego systemu bez kilku przebiegów stosowania metody prób i błędów, ale jest całkiem prawdopodobne, że będziesz mógł opracować bardziej zwarty regulator za pomocą reguł rozmytych, niż gdybyś zastosował reguły ostre (nierożmyte). Reguły rozmyte często mogą lepiej obsługiwać sytuacje, gdzie w grę wchodzi szum lub nieprzewidziane okoliczności.

13.6. Grupowanie rozmyte

Widzieliśmy już, jak możemy stosować sieci neuronowe do szukania struktur w danych i w ten sposób grupowania danych, gdy nie dysponujemy żadnymi etykietami klas. Zbiory rozmyte też umożliwiają grupowanie danych. Istnieją dwie główne metody wykonywania grupowania rozmytego (jak wynika z pracy Klira i Yuana [261]). Pierwsza z nich to metoda rozmytych c -średnich, w której liczba grup c jest określana przed grupowaniem. Druga to hierarchiczne grupowanie oparte na rozmytej relacji równoważności, przy czym liczba grup jest określana przez algorytm grupowania, a nie przez użytkownika. Metoda ta wymaga znajomości trochę większego zakresu matematyki rozmytej niż w wypadku c -średnich. W związku z tym, dla jasności prezentacji, skupimy się tutaj na metodzie c -średnich i odeślemy do pracy [261], gdzie znajduje się więcej informacji na temat użycia rozmytych relacji równoważności do grupowania (zob. też [43]).

Bezdek [42] zaproponował następującą metodę grupowania rozmytego. Przypuśćmy, że podano nam zaobserwowane wartości $\mathbf{x}_1, \dots, \mathbf{x}_n$, przy czym każde \mathbf{x}_k jest p -wymiarowym wektorem liczb rzeczywistych dla wszystkich $k \in \{1, \dots, n\}$. Celem jest znalezienie kolekcji c zbiorów rozmytych A_i dla $i = 1, \dots, c$, które stanowią „pseudopodział” danych. Należy określić funkcje przynależności dla każdego z tych zbiorów rozmytych tak, żeby dane były silnie powiązane ze sobą wewnątrz każdej grupy, a słabo między różnymi grupami. Każda grupa jest częściowo zdefiniowana przez swój centroid \mathbf{v}_i , który możemy obliczyć w zależności od wartości funkcji przynależności zbiorów rozmytych dla wszystkich danych dostępnych dla tej grupy:

$$\mathbf{v}_i = \frac{\sum_{k=1}^n [A_i(\mathbf{x}_k)]^m \mathbf{x}_k}{\sum_{k=1}^n [A_i(\mathbf{x}_k)]^m} \quad (13.1)$$

dla wszystkich $i = 1, \dots, c$. Parametr $m > 1$ steruje wpływem funkcji przynależności posiadanych danych¹. Podany tutaj stosunek powinien wyglądać znanomo, gdyż ma związek z podaną wcześniej procedurą defuzyfikacji. W istocie jest to średnia ważona danych w każdej z grup, przy czym wagi odpowiadają wartościom funkcji przynależności.

Dla danej dowolnej początkowej kolekcji zbiorów rozmytych $\mathbf{A} = \{A_1, \dots, A_c\}$ możemy obliczyć związane z nią centroidy i określić efektywność podziału na grupy za pomocą funkcji oceny

$$J_m(\mathbf{A}) = \sum_{k=1}^n \sum_{i=1}^c [A_i(\mathbf{x}_k)]^m \|\mathbf{x}_k - \mathbf{v}_i\|^2$$

przy czym: $\|\cdot\|$ jest normą iloczynu skalarnego w \mathbb{R}^p , a $\|\mathbf{x}_k - \mathbf{v}_i\|^2$ jest odległością między \mathbf{x}_k a \mathbf{v}_i . Celem jest tutaj minimalizacja $J_m(\mathbf{A})$ dla pewnego zbioru grup \mathbf{A} i wybranej wartości m . Wybór m jest jednak subiektywny i w wielu zastosowaniach używamy $m = 2$. Oznacza to, że należy tutaj wielokrotnie

¹Uwaga: zmieniliśmy notację, żeby pasowała do opisu Klira i Yuana w [261].

ulepszać ciąg zbiorów grup rozmytych $\mathbf{A}(1), \mathbf{A}(2), \dots$ aż zostanie znaleziony zbiór $\mathbf{A}(t)$ taki, że nie jest możliwe dalsze ulepszenie $J_m(\mathbf{A})$.

Następujący algorytm [42] jest zbieżny do lokalnego minimum lub do punktu siodłowego $J_m(\mathbf{A})$, nie ma jednak gwarancji, że będzie zbieżny do globalnego minimum.

1. Dla danej liczby grup c oraz wybranej wartości m wybierz $\mathbf{A}(0)(t = 0)$.
2. Określ c środków grup, $\mathbf{v}_1(t), \dots, \mathbf{v}_c(t)$, za pomocą wzoru (13.1).
3. Uaktualnij grupy rozmyte $\mathbf{A}(t + 1)$, korzystając z następującej procedury. Dla każdego \mathbf{x}_k :
 - (a) jeśli $\|\mathbf{x}_k - \mathbf{v}_i(t)\|^2 > 0$ dla $i = 1, \dots, c$, to uaktualnij funkcję przynależności dla \mathbf{x}_k w A_i , w $t + 1$:

$$A_i(t + 1)(\mathbf{x}_k) = \left(\sum_{j=1}^c \left(\frac{\|\mathbf{x}_k - \mathbf{v}_i(t)\|^2}{\|\mathbf{x}_k - \mathbf{v}_j(t)\|^2} \right)^{\frac{1}{m-1}} \right)^{-1}$$

-
-
-
- (b) jeśli $\|\mathbf{x}_k - \mathbf{v}_i(t)\|^2 = 0$ dla pewnego $i \in I \subseteq \{1, \dots, c\}$, to dla każdego $i \in I$ ustaw $A_i(t + 1)(\mathbf{x}_k)$ na liczbę z przedziału $[0, 1]$, taką że:

$$\sum_{i \in I} A_i(t + 1)(\mathbf{x}_k) = 1 \text{ oraz}$$

ustaw $A_i(t + 1)(\mathbf{x}_k) = 0$ dla pozostałych $i \notin I$

-
-
-
4. Jeśli $|\mathbf{A}(t + 1) - \mathbf{A}(t)| < \epsilon$, przy czym ϵ jest małą dodatnią stałą, to zatrzymaj się; w przeciwnym razie $t \leftarrow t + 1$ i przejdź do kroku 2.

Klir i Yuan [261] zasugerowali, żeby mierzyć różnicę między $\mathbf{A}(t + 1)$ a $\mathbf{A}(t)$ w kroku 4 za pomocą formuły

$$\max_{i \in \{1, \dots, c\}, k \in \{1, \dots, n\}} |A_i(t + 1)(\mathbf{x}_k) - A_i(t)(\mathbf{x}_k)|$$

Oto przykład, który zilustruje działanie tej procedury. Mamy dziesięć dwuwymiarowych punktów danych¹. Pierwszych pięć punktów zostało obranych zgodnie z funkcją gęstości Gaussa o średniej w punkcie $(1, 1)$ i o odchyleniu standardowym 1,0 dla każdego z wymiarów. Pozostałych pięć punktów zostało obranych zgodnie z funkcją gęstości Gaussa o średniej w punkcie $(3, 3)$ i odchyleniu standardowym 1,0. Wygenerowane dane wejściowe są następujące:

¹Dziękujemy Kumarowi Chellapilla za pomoc polegającą na zaprogramowaniu na potrzeby tego przykładu algorytmu rozmytych c -średnich.

- 1: (1,7229, 0,2363)
- 2: (1,0394, 3,1764)
- 3: (2,5413, 1,4316)
- 4: (0,7011, 0,5562)
- 5: (0,0337, 1,0300)
- 6: (2,6843, 2,7687)
- 7: (3,9778, 2,8863)
- 8: (3,0183, 3,1279)
- 9: (3,8180, 2,2006)
- 10: (3,7023, 2,7614)

Wybrałyśmy liczbę grup $c = 2$ (tak, żeby zgadzało się to ze sposobem generowania danych) i przyjęłyśmy $m = 2$. Losowo wybrałyśmy początkowe wartości funkcji przynależności dla punktów w grupach jako:

Grupa	Dane									
	1	2	3	4	5	6	7	8	9	10
$A_1(x)$	0,4717	0,4949	0,3763	0,6991	0,4651	0,4179	0,5109	0,4664	0,4448	0,6386
$A_2(x)$	0,5283	0,5041	0,6237	0,3009	0,5349	0,5821	0,4891	0,5336	0,5552	0,3614

Po takim inicjowaniu wygenerowane w końcu grupy były umieszczone w
 $(0,2748, 0,8913)$ oraz $(3,2105, 2,6175)$

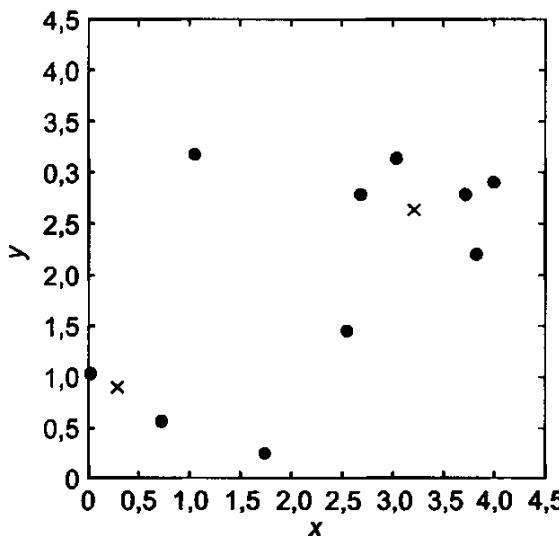
przy $J_m(\mathbf{A}) = 9,1122$. Zauważ, że środki grup nie wypadają dokładnie w punktach średnich naszych dwóch rozkładów Gaussa, ale też nie powinniśmy się tego spodziewać, gdyż punkty te wynikają z rzeczywistego rozłożenia naszych punktów, a nie z rozkładów, które te punkty wygenerowały. Końcowe rozmyte funkcje przynależności dla tych punktów to:

Grupa	Dane									
	1	2	3	4	5	6	7	8	9	10
$A_1(x)$	0,7573	0,4639	0,2546	0,9484	0,9913	0,0311	0,0360	0,0232	0,0366	0,0169
$A_2(x)$	0,2427	0,5361	0,7454	0,0516	0,0087	0,9689	0,9640	0,9768	0,9634	0,9831

Na rysunku 13.7 przedstawiono rozkład próbek oraz centroidy rozmytych grup.

13.7. Rozmyte sieci neuronowe

Jeśli jesteś przewidujący, to zapewne już zastanawiałeś się nad możliwościami połączenia sieci neuronowych z systemami rozmytymi. W poprzednim rozdziale omawialiśmy potencjalne możliwości sieci neuronowych, jeśli chodzi o wykonywanie grupowania i klasyfikacji przy użyciu wartości ostrych (nieroźmytych). Całkiem sensowne jest myślenie o zastąpieniu tych wartości liczbami rozmytymi. W rozmytej sieci neuronowej każdy węzeł wykonuje rozmyte operacje



Rysunek 13.7. Wynik grupowania metodą rozmytych c -średnich dla 10 punktów wygenerowanych z dwóch rozkładów Gaussa. Punkty te przedstawiono jako małe czarne kółka. Końcowe położenie grup przedstawiono za pomocą dwóch znaków \times . Wartości funkcji przynależności dla każdej z grup we wszystkich punktach podane są w tekście

arytmetyczne i zamiast generowania jednej wartości generuje zbiór rozmytych funkcji przynależności dla różnych możliwych wartości. Na wejścia rozmytej sieci neuronowej są podawane liczby rozmyte, wagi są także liczbami rozmytymi, a wykonywane w każdym węźle ocenianie, które odbywało się przy użyciu iloczynu skalarnego przy zwykłych liczbach, jest tutaj dokonywane przy pomocy rozmytego dodawania i mnożenia. Aby lepiej wytłumaczyć, o co tutaj chodzi, musimy zacząć od definicji liczb rozmytych i opisać, jak wykonuje aię na nich operacje arytmetyczne.

Liczba rozmyta to zbiór rozmyty A na dziedzinie R , który ma co najmniej trzy następujące własności:

1. Maksymalna wartość funkcji przynależności dowolnego elementu zbiuru A wynosi 1,0 (tzn. A jest *normalnym* zbiorem rozmytym).
2. Nośnik zbioru rozmytego A musi być ograniczony.
3. αA dla każdego $\alpha \in (0, 1]$ musi być przedziałem domkniętym.

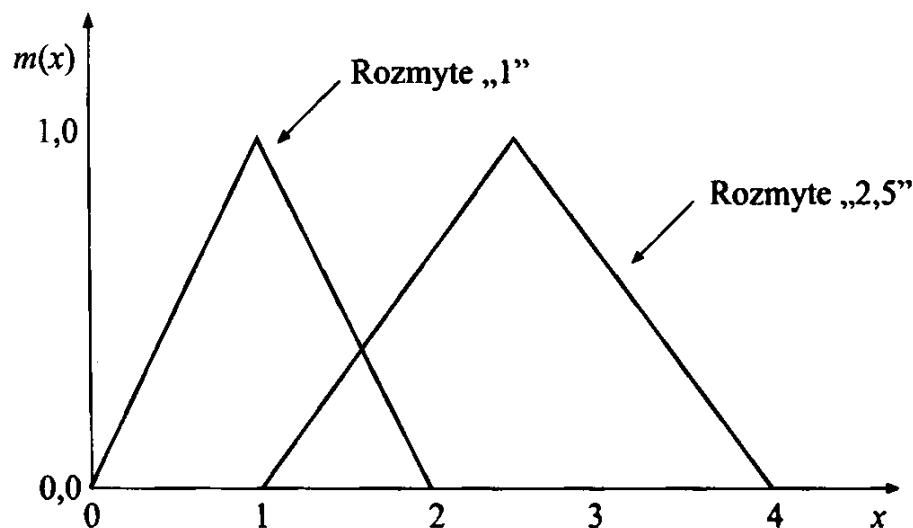
Wprowadziłmy tutaj nowe pojęcia i notacje, które musimy wyjaśnić, zanim będziemy mogli iść dalej:

- Nośnik zbioru rozmytego A w przestrzeni rozważań U to zbiór nieroźmyty zawierający wszystkie te elementy U , których stopień przynależności do A jest niezerowy.
- Notacja αA oznacza wszystkie wartości $\{u | A(u) \geq \alpha\}$. Zbiór ten nazywany jest α -cięciem (α -przekrojem) zbioru rozmytego A .

Z pomocą tych wstępnych definicji możemy teraz udowodnić, że zbiór rozmyty A jest „liczbą rozmytą” wtedy i tylko wtedy, gdy istnieje przedział domknięty $[a, b] \neq \emptyset$, taki że

$$A(x) = \begin{cases} l(x) & \text{dla } x \in (-\infty, a) \\ 1 & \text{dla } x \in [a, b] \\ r(x) & \text{dla } x \in (b + \infty) \end{cases}$$

przy czym: l jest funkcją rosnącą z $(-\infty, a)$ do $[0, 1]$, a r jest funkcją malejącą w przedziale $(b, +\infty)$. Funkcje l i r muszą jeszcze odpowiednio być prawostronnie i lewostronne ciągłe. Typowe przykłady liczb rozmytych przedstawiono na rys. 13.8.



Rysunek 13.8. Dwie liczby rozmyte: 1 i 2,5. Liczby rozmyte mają pewne matematyczne własności jak zbiory rozmyte. Są one opisane w tekście

Po zdefiniowaniu liczb rozmytych możemy przejść do definicji arytmetyki rozmytej, w tym do definicji operacji dodawania, odejmowania, mnożenia i dzielenia. Dla każdego $\alpha \in (0, 1]$ każde α -cięcie liczby rozmytej to przedział domknięty liczb rzeczywistych i każda liczba rozmyta może być jednoznacznie reprezentowana przez swoje α -cięcia. A zatem wygodnie jest opisywać rozmyte operacje arytmetyczne w kategoriach operacji na związkanych z tymi α -cięciami przedziałach domkniętych.

W szczególności wspomniane wcześniej cztery operacje arytmetyczne są zdefiniowane następująco:

$$[a, b] + [d, e] = [a + d, b + e]$$

$$[a, b] - [d, e] = [a - e, b - d]$$

$$[a, b] \cdot [d, e] = [\min(ad, ae, bd, be), \max(ad, ae, bd, be)]$$

$$[a, b]/[d, e] = [a, b] \cdot [1/e, 1/d] \text{ przy } 0 \neq [d, e]$$

Ponieważ liczby rozmyte można przedstawić za pomocą ich α -cięć, powyższe operacje możemy zastosować do α -cięć liczb rozmytych, a następnie wygenerować odpowiednie wynikowe liczby rozmyte.

Przypuśćmy, na przykład, że mamy dwie liczby rozmyte:

$$A(x) = \begin{cases} 0 & \text{dla } x \leq 0 \text{ oraz } x > 2 \\ x & \text{dla } 0 < x \leq 1 \\ -x + 2 & \text{dla } 1 < x < 2 \end{cases}$$

$$B(x) = \begin{cases} 0 & \text{dla } x \leq 3 \text{ oraz } x > 5 \\ x - 3 & \text{dla } 3 < x \leq 4 \\ -x + 5 & \text{dla } 4 < x < 5 \end{cases}$$

$A(x)$ jest rozmyta „1”, zaś $B(x)$ jest rozmyta „4”. Żeby określić α -cięcia dla $A(x)$ i $B(x)$, ustalamy wzory, w których $A(x)$ i $B(x)$ mają nośnik równy α i rozwiążemy je względem x . W związku z tym dla $A(x)$ uzyskujemy ${}^\alpha A = [\alpha, 2 - \alpha]$ oraz dla $B(x)$ – ${}^\alpha B = [\alpha + 3, 5 - \alpha]$. Przyjmujemy, że α -cięcie zbioru $(A * B)$, przy czym $*$ jest jednym z wymienionych wcześniej operatorów arytmetycznych, jest zdefiniowane jako

$${}^\alpha(A * B) = {}^\alpha A * {}^\alpha B \text{ dla } \alpha \in (0, 1]$$

(jeśli $*$ jest równe $/$, to jest wymagane dodatkowe ograniczenie $0 \notin {}^\alpha B$ dla $\alpha \in (0, 1]$). Gdybyśmy chcieli dodać $A + B$, wówczas ${}^\alpha(A + B) = [2\alpha + 3, 7 - 2\alpha]$. Podobnie

$$\begin{aligned} {}^\alpha(A - B) &= [2\alpha - 5, -1 - 2\alpha] \\ {}^\alpha(A \cdot B) &= [\alpha^2 + 3\alpha, \alpha^2 - 7\alpha + 10] \\ {}^\alpha(A/B) &= [\alpha/(5 - \alpha), (2 - \alpha)/(\alpha + 3)] \end{aligned}$$

Skupiąc naszą uwagę na przykładzie z dodawaniem, dla $(A + B)$ wynikiem będzie

$$(A + B)(x) = \begin{cases} 0 & \text{dla } x \leq 3 \text{ oraz } x > 7 \\ \frac{1}{2}(x - 3) & \text{dla } 3 < x \leq 5 \\ \frac{1}{2}(7 - x) & \text{dla } 5 < x < 7 \end{cases}$$

co jest opisem liczby rozmytej „5”.

Tego rodzaju operacje arytmetyki rozmytej są wykorzystywane do obliczania wyniku na wyjściu każdego neuronu rozmytego w rozmytej sieci neuronowej. Bardziej konkretnie, wynik k -tego węzła jest określony jako

$$Y_k = f \left(\sum_{j=0}^n W_j X_{kj} \right)$$

przy czym: $f(\cdot)$ jest nieliniową funkcją przenoszenia (zwykle sigmoidalną), W_j jest wejściową wagą rozmytą dla j -tego węzła, a X_{kj} jest j -tą rozmytą aktywacją wchodzączą do k -tego węzła. Tak więc suma $\sum_{j=0}^n W_j X_{kj}$ jest rozmytą sumą arytmetyczną ciągu rozmytych iloczynów arytmetycznych. Dalsze

przekształcenie tej sumy za pomocą funkcji $f(\cdot)$ uzyskujemy za pomocą „zasady rozszerzania” dla zbiorów rozmytych (algorytmiczne szczegóły znajdują się w [261]). Chociaż przykład rozmytej sieci neuronowej nie jest tematem tej książki, powinieneś w tym momencie móc wyobrazić sobie, jak taka konstrukcja powinna być przeprowadzona, i uzyskać intuicję na temat rodzajów operacji wykonywanych przez taką sieć. Bardziej szczegółowe przedstawienie różnych interpretacji sieci neuronowych znajdziesz w [43].

13.8. Podejście rozmyte do TSP

Podejście rozmyte możemy również zastosować do problemu komiwojażera (lub innych problemów optymalizacyjnych). Moglibyśmy opisać długość dowolnej danej ścieżki między dwoma miastami za pomocą funkcji przynależności zbiorów rozmytych. Jeszcze przed określeniem położenia każdego miasta moglibyśmy utworzyć jakieś funkcje przynależności, które reprezentowałyby możliwe długości ścieżek. Najkrótsza długość jest dowolnie bliska zera, co ma miejsce, gdy dwa miasta są umieszczone w zasadzie w tym samym punkcie. Najdłuższa możliwa długość to odległość między wierzchołkami leżącymi po przeciwnych stronach przekątnej prostokąta, w którym muszą być umieszczone wszystkie miasta. Tę maksymalną odległość oznaczmy $MaxL$. Wiedząc teraz, że zakres możliwych długości ścieżek między dowolnymi dwoma miastami może zmieniać się od zera do $MaxL$, możemy podzielić ten zakres na cztery równe części, korzystając z wartości $\frac{1}{4}MaxL$, $\frac{1}{2}MaxL$ i $\frac{3}{4}MaxL$, a następnie utworzyć następujące funkcje przynależności opisujące długość odcinka ścieżki w kategoriach języka naturalnego.

- Wartość funkcji przynależności zbioru *krótkich ścieżek* wynosi 1,0 dla ścieżek o długości między zero a $\frac{1}{4}MaxL$. Dla długości powyżej $\frac{1}{4}MaxL$ funkcja przynależności tego zbioru zmniejsza się liniowo do zera przy długości $\frac{1}{2}MaxL$ i pozostaje zerowa dla wszystkich dłuższych odcinków.
- Wartość funkcji przynależności zbioru *średniokrótkich ścieżek* wynosi 1,0 dla ścieżek o długości między $\frac{1}{4}MaxL$ a $\frac{1}{2}MaxL$. Dla długości od zera do $\frac{1}{4}MaxL$ funkcja przynależności zwiększa się liniowo od zera do 1,0. Dla ścieżek o długości od $\frac{1}{2}MaxL$ do $\frac{3}{4}MaxL$ zmniejsza się liniowo od 1,0 do zera i pozostaje zerowa dla wszystkich dłuższych ścieżek.
- Wartość funkcji przynależności zbioru *średnidługi ścieżek* wynosi 1,0 dla ścieżek o długości między $\frac{1}{2}MaxL$ a $\frac{3}{4}MaxL$. Funkcja przynależności jest zerowa dla wszystkich długości mniejszych od $\frac{1}{4}MaxL$ i rośnie liniowo od zera do 1,0 dla długości od $\frac{1}{4}MaxL$ do $\frac{1}{2}MaxL$. Funkcja ta maleje liniowo od 1,0 do zera dla długości od $\frac{3}{4}MaxL$ do $MaxL$.
- Wartość funkcji przynależności zbioru *długi ścieżek* wynosi 1,0 dla ścieżek o długości między $\frac{3}{4}MaxL$ a $MaxL$. Funkcja przynależności jest zerowa dla wszystkich długości mniejszych od $\frac{1}{2}MaxL$, a jej wartość rośnie liniowo od zera do 1,0 dla długości od $\frac{1}{2}MaxL$ do $\frac{3}{4}MaxL$.

Na podstawie tych rozmytych opisów moglibyśmy utworzyć funkcję oceny faworyzującą te pełne trasy, które dają większą ogólną wartość funkcji przynależności w zbiorze krótkich ścieżek, a mniejszą dla pozostałych zbiorów.

Moglibyśmy, na przykład, przyjąć za funkcję oceny

$$f(x) = \sum_{j=1}^n \sum_{i=1}^4 \alpha_i M_{ij}$$

przy czym: i jest indeksem i -tej funkcji przynależności, α_i – współczynnikiem skalującym dla i -tej funkcji przynależności, M_{ij} – wartością i -tej funkcji przynależności dla j -tego odcinka ścieżki (tzn. M_{11} jest wartością funkcji przynależności zbioru krótkich ścieżek dla pierwszego odcinka danej trasy). Każda ścieżka ma n odcinków i każdy odcinek będzie miał przypisaną wartość funkcji przynależności dla każdej opisanej przez nas klasy długości ścieżek. Naszym celem jest minimalizacja wartości $f(x)$ na podstawie uprzedniego wyboru współczynników skalujących α_i . Wydaje się bardziej sensowne, żeby przypisywać coraz większe wagi funkcjom przynależności związanym z większymi długościami ścieżek, gdyż to są rzeczy, które chcemy minimalizować.

Czy takie podejście jest dobre w przypadku problemu komiwojażera? Być może. Jest jednak wysoce prawdopodobne, że będzie niewystarczające, aby osiągnąć rozwiązania bliskie optymalnych, gdyż funkcja przynależności dla krótkich ścieżek nie jest wystarczająco precyzyjnie wyskalowana. Prawdopodobnie nie wystarczy mieć po prostu krótkie ścieżki. Potrzebne może się okazać zdefiniowanie większej liczby funkcji przynależności, które opisywałyby *bardzo krótkie ścieżki, niezwykle krótkie ścieżki* itd. W końcu dojdziemy do używania odległości między poszczególnymi parami miast, a wtedy zysk obliczeniowy z rozmytego podejścia pozostaje niejasny. Mimo to możliwe jest myślenie o rozwiązywaniu rozmytego problemu komiwojażera.

13.9. Ewoluujące systemy rozmyte

W każdym systemie rozmytym trzeba podjąć wiele decyzji projektowych. Ile funkcji przynależności zbiorów rozmytych należy użyć? Jaki powinny mieć kształt? Trójkątny, trapezoidalny, gaussowski czy może jeszcze jakiś inny? Jaki rodzaj skalowania względnego powinien być wprowadzony? W wypadku problemu sterowania rozmytego, jakie działania powinny być wykonywane na podstawie rozmytych opisów stanu bieżącego? Jeśli wózek jest *na lewo*, to czy powinniśmy go *pchnąć*, czy *pchnąć mocno*, czy może *pchnąć lekko*? Każdy z tych wyborów musi zostać dokonany, zanim cały system rozmyty można w ogóle oceniać. Proces dostosowywania systemu rozmytego, żeby dał pożądaną odpowiedź, bardzo przypomina dostosowywanie topologii sieci neuronowej do momentu, aż zostaje osiągnięty pożądany wynik. Często sprowadza się to do żmudnego stosowania metody prób i błędów.

Pokazaliśmy już, jak tego rodzaju metodę prób i błędów możemy zautomatyzować za pomocą obliczeń ewolucyjnych. Jako konkretny przykład możemy rozważyć zadanie dopasowania opisanego wcześniej systemu rozmytego sterującego umieszczaniem wózka na środku platformy. Jak wyglądały nasze wybory projektowe podczas opracowywania tego systemu? Musieliśmy najpierw zdecydować, które parametry wózka są istotne. Skupiliśmy się w naszym przypadku na pozycji i prędkości, ale moglibyśmy też dołączyć przyspieszenie wózka, co spowodowałoby konieczność dodania zestawu deskryptorów rozmytych dla tego elementu opisu. Dla każdego użytego parametru musieliśmy określić, ile rozmytych opisów należałoby zastosować i jaki kształt te opisy przyjmowałyby.

Jeśli ograniczymy, dla uproszczenia prezentacji, naszą uwagę do trójkątów i trapezów, to każdy zbiór rozmyty jest określony przez rozmieszczenie wierzchołków jego trójkąta lub trapezu. Na rysunku 13.4 rozmyta pozycja wózka *na środku* ma środek w punkcie zero i rozciąga się ze swoim nośnikiem od minus jednego do jednego metra. Być może, byłoby możliwe, aby ustalić, że środek tego zbioru rozmytego powinien być w zerze, gdyż żaden inny wybór nie jest racjonalny, ale nie ma sposobu, żeby a priori wiedzieć, że funkcja przynależności zbioru *na środku* powinna mieć nośnik sięgający od minus jednego do plus jednego metra. Być może, lepszy regulator dałoby się uzyskać, gdybyśmy przyjęli $\pm 0,8945$ metra albo wręcz jakiś zakres, który nie jest symetryczny względem punktu środkowego! Wartości te moglibyśmy optymalizować za pomocą algorytmu ewolucyjnego, który generowałby i testował wiele regulatorów na podstawie rozmytych opisów, z których korzystają. Podobnie możemy też przeprowadzić ewolucję każdej z rozmytych reguł sterujących, która polegałaby na dostosowaniu warunków, żeby każde „jeśli” zostało spełnione, oraz czynności wykonywanych w wyniku zastosowania reguły.

13.10. Podsumowanie

Przy stosowaniu komputerów do rozwiązywania problemów świata rzeczywistego pojawia się silne dążenie do jak największej precyzji. Wydaje się bowiem, że wielką zaletą komputera jest zdolność do dokładnego obliczania trudnych, przy zastosowaniu innych środków, wyrażeń matematycznych. Jego możliwości w tym zakresie znacznie przekraczają możliwości ludzkiego umysłu oraz zmysłów, którymi się człowiek posługuje. Komputery i ludzie są jednak stworzeni (rozwijają się) do działania w zupełnie odmiennych środowiskach. Środowisko komputera to świat binarny rzeczy czarnych i białych. Środowisko człowieka jest zaszumione, chaotyczne i zmienne w czasie. Co więcej, taki świat odbieramy za pomocą niedoskonałych zmysłów, które dają jedynie rozmyty obraz tego, co nas otacza. Z konieczności opracowaliśmy na drodze ewolucji język słów, aby opanować naszą niepełną, niedoskonałą wiedzę o świecie rzeczywistym. Stworzyliśmy też zupełnie inny język matematyki, służący do opisywania wyidealizowanego świata, który działa precyzyjnie, ale w próżni, gdzie mnóstwo fizycznych obiektów koncentruje się w jednym punkcie, a żadne linie nie mają szerokości.

Mimo tej pewnej nieprecyzyjności naszego języka naturalnego możemy go ująć w kategorie matematyczne i w istocie dokonywać obliczeń, korzystając ze słów. Zalety wynikające z możliwości opisywania różnych klas sytuacji za pomocą niejednoznacznych terminów często przenoszą się na budowę praktycznych regulatorów i algorytmów rozpoznawania wzorców. W tym rozdziale poznaliśmy pewne proste przykłady tego rodzaju oraz przyjrzaliśmy się podstawom rozmytych operacji arytmetycznych, a także opisaliśmy relacje między dwoma bardziej rozmytymi zbiorami. Więcej na temat potencjalnych możliwości logiki rozmytej znajdziesz w [44, 503, 261, 403].

XIV. Wszystko od czegoś zależy

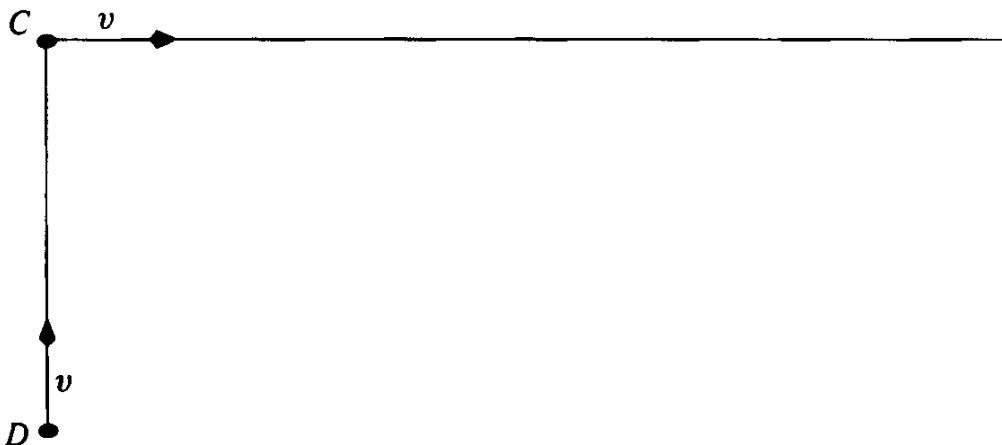
Jest taka stara historia o chłopcu, który zobaczył burzę z piorunami i zapytał tatę, co ją spowodowało. Ojciec spojrzał na syna i powiedział: „Motyl po drugiej stronie ziemi”. To zaintrygowało chłopca, a zatem ojciec ciągnął dalej: „Motyl machnął skrzydłem, poruszyło się więc powietrze, oddalając się małym podmuchem od motyla. Ten podmuch popchnął inne prądy powietrzne, które były coraz większe i większe, przechodząc przez całą planetę, aż dotarły tutaj w postaci burzy”. Ten tak zwany „efekt motyla” pojawia się w wielu podobnych historiach; mają one jeden wspólny element – fakt, że jeden ruch może spowodować ciąg innych ruchów o rosnącym rozmiarze i niespodziewanych konsekwencjach oraz że wszystko na świecie od czegoś zależy.

Być może, pamiętasz problem z podrozdziału 1.3 dotyczący poszukiwania najlepszego miejsca na sklep. Możesz zgromadzić informacje demograficzne na temat Twoich klientów, przyjąć pewne założenia co do tego, jak daleko gotowi są jechać, żeby dotrzeć do Twojego sklepu, i wykorzystać to do lokalizacji najlepszego punktu. Jeśli tak zrobisz, to na pewno wybierzesz kiepskie miejsce, gdyż w tej analizie nie wziąłeś pod uwagę pewnego istotnego elementu – Twojej konkurencji! Oni też tak jak Ty szukają miejsca na swój sklep. Jeśli wezmą pod uwagę to, gdzie Ty prawdopodobnie umieścisz swój sklep, a Ty tego nie zrobisz w stosunku do ich sklepów, to przegrasz.

Wiele rzeczywistych problemów ma podobny charakter. Rozwiążanie ich wymaga wzięcia pod uwagę innych części problemu, które się zmieniają w tym samym czasie, gdy zmieniasz się i Ty. Mogą to być konkurenci, którzy chcą Cię pokonać, i przyjaciele, którzy chcą Ci pomóc, lub też gapiowie, którzy po prostu wchodzą Ci w drogę. Zawsze jednak musisz brać pod uwagę, jak ze sobą współpracują te elementy problemu. Oto kilka przykładów na rozgrzewkę.

Zacznijmy od prostego przypadku psa goniącego kota. Spójrz na rys. XIV.1. Widać tam punkty, w których początkowo znajdują się pies D

i kot C . Pies widzi kota i zaczyna szczekać. Kot słyszy psa i zaczyna biec z lewa na prawo. Przypuśćmy, że pies i kot biegną z taką samą prędkością v i że pies goni kota, biegając wprost na niego, nie tracąc go jako celu. Tu pojawia się pierwsze pytanie. Tym razem łatwe – czy pies złapie kota? W porządku, to łatwe, odpowiedź brzmi nie, gdyż biegną z tą samą prędkością, a pies musi przebyć większy dystans niż kot, jeśli jego wróg ma być złapany. Pojawia się zatem następne pytanie, tym razem trochę bardziej skomplikowane. Po odpowiednio długim czasie ścieżka psa stanie się prawie identyczna ze ścieżką kota. Jeśli pies znajdzie się bezpośrednio za kotem, to w jakiej odległości będzie on od kota, przy założeniu, że początkowo odległość DC wynosiła 20 metrów?



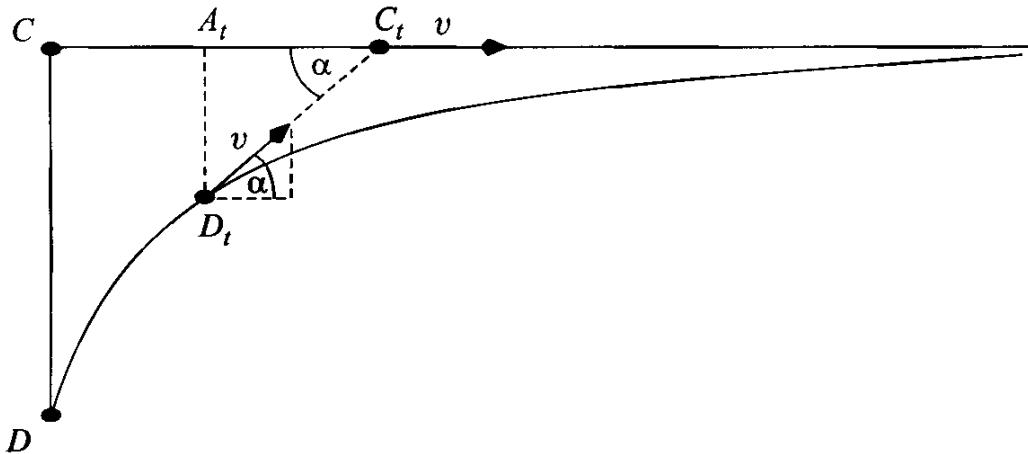
Rysunek XIV.1. Pozycje na początku wyścigu

Jeśli nie możesz od razu wpaść na sposób prowadzący do rozwiązania, to, być może, wystarczy, żebyś podał górne ograniczenie na wartość rozwiązania. Jak myślisz, czy szukana odległość może być większa niż 20 metrów? Oczywiście, że nie, gdyż taką odległość mamy na początku i gdyby pies pobiegł do początkowej pozycji kota C , to byłby za nim 20 metrów. Zamiast tego pies zbliża się asymptotycznie do kota jak termicznie sterowany pocisk, który leci z taką samą prędkością jak jego cel. Wiemy zatem, że odpowiedź musi brzmieć: odległość jest mniejsza niż 20. O ile mniejsza?

Pomoże nam tutaj odrobiną trygonometrii. Niech α oznacza chwilowy kąt między odcinkiem $C_t D_t$ a odcinkiem CC_t , gdzie C_t oznacza bieżącą pozycję kota, a D_t bieżącą pozycję psa, przy czym oba biegną z prędkością v . Kąt ten pokazano na rys. XIV.2.

Prędkość, z jaką pies zbliża się do kota, zależy od dwóch czynników: 1) prędkości v , z jaką biegną oba zwierzęta, i 2) czynnika $v \cos \alpha$, który jest zorientowany w kierunku, w którym ucieka kot. Punkt A_t na rysunku pokazuje mniej więcej rzutowania pozycji psa na drogę kota. Możesz sobie teraz wyobrazić, że zadanie polega na tym, żeby określić, jak szybko A_t zbliża się do C_t , w miarę wzrostu t . Kot biegnie z prędkością v , ale rzutowanie pozycji psa A_t na drogę kota następuje z prędkością $v \cos \alpha$. Różnica między tymi prędkościami wynosi $v(1 - \cos \alpha)$ i to właśnie jest prędkość, z którą pies zbliża się do kota.

Miedzy długością $D_t C_t$ a długością $A_t C_t$ istnieje ścisły związek. W miarę jak maleje $D_t C_t$, z taką samą szybkością rośnie $A_t C_t$. Zatem suma odległości

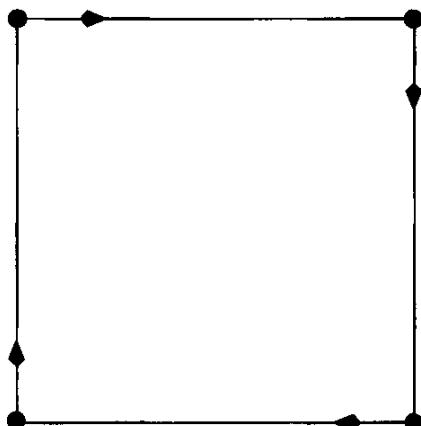


Rysunek XIV.2. Gonitwa

D_tC_t i A_tC_t jest zawsze stała i w tym wypadku wynosi 20 metrów, czyli tyle, ile w momencie pierwszego szczeknięcia psa. W miarę jak rośnie t , punkt D_t zbliża się do A_t , a więc odległości D_tC_t i A_tC_t w końcu staną się równe. To daje odpowiedź na nasze pytanie: ponieważ suma D_tC_t i A_tC_t jest zawsze równa 20 metrów, a w końcu wielkości te się zrównują, to końcowa odległość między psem a kotem będzie 10 metrów! Uff!!!

Przenieśmy się teraz na równinę Masai Mara w Kenii, gdzie cztery lwy wytropiły antylopę gnu. Lwy znalazły się w mało prawdopodobnej sytuacji, gdyż znajdują się na rogach kwadratu, w którego środku stoi antylopa. Żaden z lwów nie ma przewagi nad pozostałymi, idą więc jeden za drugim po spirali, zbliżając się do swojej ofiary. Podobnie jak w zadaniu z kotem, każdy z nich kieruje się za swym najbliższym sąsiadem, co pokazano na rys. XIV.3.

Każdy lew biegnie bezpośrednio za swym sąsiadem. Drapieżnik z lewego górnego rogu biegnie za lwem z prawego górnego rogu, drapieżnik z prawego górnego rogu biegnie za lwem z prawego dolnego rogu itd. Na początku lwy są oddalone o 1000 metrów i biegą z prędkością 20 metrów na sekundę. Twoje zadanie polega na określaniu, ile czasu zajmie lwom dotarcie do punktu spotkania. Aha, a czy w ogóle ich tory się kiedyś przetną? Jeśli tak, to gdzie? I jeszcze, jak długie będą pokonane przez nie trasy? Ten problem składa się z wielu elementów.



Rysunek XIV.3. Cztery lwy na równinie Afryki

Problem czterech zbliżających się do siebie lwów natychmiast wydaje się podobny do problemu z psem goniącym kota, z tą różnicą, że zamiast dwóch mamy tutaj cztery poruszające się obiekty. W istocie rozwiązanie jest też trochę prostsze. Każdy lew porusza się pod kątem prostym do swojego sąsiada, gdyż wszystkie one są ze sobą powiązane, a zatem każdy goniący lew zbliża się do swojego sąsiada z prędkością 20 metrów na sekundę. Wniosek z tego jest taki, że lwy spotkają się po 50 sekundach, gdyż na początku są od siebie w odległości 1000 metrów. Ponieważ wszystkie lwy spotkają się po 50 sekundach, każdy z nich pokona po drodze 1000 metrów. Ponieważ lwy są zawsze pod kątami prostymi, zawsze są umiejscowione w wierzchołkach kwadratu, który obraca się. Powierzchnia tego kwadratu zmniejsza się, gdyż jego boki zmniejszają się z szybkością 20 metrów na sekundę. Po 50 sekundach powierzchnia kwadratu będzie równa zeru i wszystkie lwy będą stały jeden na drugim (i jeszcze na antylopie!) w środku początkowego kwadratu. W trakcie biegu zwierzęta zakreślą nieprzecinające się ze sobą spirale logarytmiczne. Gdyby te spirale się przecinały, oznaczałoby to, że inny lew był w punkcie przecięcia wcześniej niż obecnie tam znajdujący się drapieżnik. To zaś jest niemożliwe, gdyż cztery odległości lwów od środka są zawsze sobie równe i stale maleją w miarę upływu czasu. Zagadka ta jest trochę łatwiejsza niż zagadka z psem i kotem, gdyż czwarty lew jest „powiązany” z pierwszym.

A teraz będzie zagadka, w której nie będzie kłów ani pazurów, ale za to uczestnicy ze sobą współpracują. Opisano ją w [346]; my jednak uaktualniliśmy ją, żeby pasowała do dwudziestego pierwszego wieku.

Czterech ludzi w telewizyjnym programie typu reality show musi odbyć wyprawę w głęb pustyni Sahara. Każda osoba może zabrać ze sobą wodę na 10 dni i może każdego dnia przebyć 24 kilometry. (Możemy zatem przyjąć, że każda osoba zaczyna z 240 jednostkami wody, przy czym 1 jednostka to ilość wody zużywana przez uczestnika na przebycie 1 kilometra). Ich wspólny cel polega na tym, żeby co najmniej jeden członek zespołu dotarł tak daleko w głęb pustyni, jak tylko się da – i wrócił! Zespół zebrał informacje na temat pustyni i wie, że jest niezamieszkała; można więc bezpiecznie zostawić po drodze wodę na drogę powrotną. Reguły programu wymuszają, że żaden z członków zespołu nie może wrócić do cywilizacji, żeby uzupełnić zapasy wody. Reguły programu wymuszają też to, żeby każdy pozostał przy życiu. Prawnicy producentów programu nie chcą żadnych procesów, a reklamujący program nie chcieliby, żeby ktokolwiek wybrał się na pustynię, zostawił całą swoją wodę dla towarzyszy, a następnie czołgał się, umierając z pragnienia! Jak daleko zatem zespół może dotrzeć?

Jeśli przez cały czas będą razem, to uda im się pięciodniowa wycieczka na 120 kilometrów w głęb pustyni i będą mieć dość wody na powrót. Jeśli uczestnicy będą współpracować, mogą dojść o wiele dalej. Przypuśćmy, że przyjmiemy model problemu pokazany na rys. XIV.4. Grupa zaczyna w punkcie S . Wszyscy czterej członkowie dochodzą do punktu A , gdzie jeden z nich wraca do S z ilością wody wystarczającą do powrotu, zostawiając wodę w punkcie A na potrzeby powracających kolegów. Pozostali członkowie grupy idą do B i również jeden z nich wraca do S . Dalej przechodzą do C , gdzie stosują tę samą

strategię, i wreszcie ostatnia osoba wędruje do D i wraca. Jest to dobry schemat rozwiązyania tego problemu. Pojawia się teraz pytanie: jak daleko od S znajduje się D ?



Rysunek XIV.4. Wycieczka w głąb pustyni

Każdy członek zespołu musi przejść odcinek SA dwa razy – po razie w każdym kierunku. Przez SA podróż odbywa się zatem osiem razy. Podobnie odcinek AB trzeba pokonać sześć razy, BC cztery razy, a CD dwa. Przyjmijmy, że osoba, która pokonuje odcinek SA , zużywa a jednostek wody. Zatem w A musi zostać $3a$ jednostek wody na potrzeby drogi powrotnej AS dla trzech podróżników, którzy przechodzą przez A . Stosując podobną notację, mamy, że w B musi zostać $2b$ jednostek, a w C c jednostek.

Celem jest zmaksymalizowanie odległości SD . Każdy podróżujący, gdy zaczyna, ma 240 jednostek wody. Dotarcie do punktu A powoduje zużycie a jednostek wody, każdy zatem z podróżników w punkcie A ma $(240 - a)$ jednostek wody. Pierwsza osoba wracająca do punktu początkowego będzie potrzebowała dodatkowo a jednostek, żeby wrócić, i będzie potrzebowała $3a$ jednostek, żeby zapewnić pozostałym uczestnikom programu picie wody na ostatnim odcinku drogi powrotnej. Oznacza to, że pierwsza osoba będzie zaczynała z 240 jednostkami, zużyje a jednostek, żeby dostać się do A , zostawi tam $3a$ jednostek dla pozostałych podróżujących i zużyje a jednostek na drogę powrotną. Na końcu zostanie więc jej $(240 - 5a)$ jednostek wody. Każdy z pozostałych podróżujących zużyje a jednostek, żeby się dostać do A . Pierwszy gracz może więc uzupełnić zasoby swoich towarzyszy, dając każdemu a jednostek, a w optymalnej sytuacji pierwszy podróżny nie powinien już mieć wody po powrocie z A , zatem $(240 - 8a) = 0$, czyli $a = 30$.

Po obsłużeniu pierwszego odcinka podróży w głąb Sahary pozostałe etapy analizujemy, stosując analogiczne podejście, co prowadzi do równań $(240 - 6b) = 0$, czyli $b = 40$, $(240 - 4c) = 0$, czyli $c = 60$, i podobnie $(240 - 2d) = 0$, czyli $d = 120$. Ponieważ każda osoba zużywa 1 jednostkę wody na kilometr, pozycje A , B , C i D są w odległości odpowiednio 30, 70, 130 i 250 kilometrów od S . Odpowiedź brzmi zatem 250 kilometrów.

Przy okazji, odległość, którą osiągnęli, posuwając się w głąb Sahary, możemy matematycznie wyrazić jako

$$120 \left(1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \right) = 250$$

co jest wygodne przy uogólnianiu zadania na sytuację, gdy mamy n uczestników telewizyjnego reality show. W tej sytuacji maksymalna odległość, na jaką się mogą udać, to

$$120 \sum_{i=1}^n \frac{1}{i}$$

Wartość ta może być dowolnie duża, gdyż ten ciąg jest rozbieżny do nieskończoności.

Wspaniałą zagadkę o nadzwyczajnej „współpracy i konkurowaniu” można znaleźć w [456]. Zagadka ma dwie wersje: krótką i rozszerzoną. Zaczniemy od krótkiej.

Dziesięciu piratów napadło na statek i odkryło na nim 100 sztuk złota. Zdobytych muszą podzielić między siebie. Chcą przy tym być w porządku i przestrzegać prawa morza – łupy mają iść do najsilniejszego. W związku z tym urządzili zawody w siłowaniu się na rękę, żeby określić, jak silny jest każdy z piratów, i ustawić ich od najsilniejszego do najsłabszego. Żadni dwaj piraci nie są tak samo silni, nie ma więc wątpliwości co do tego, że porządek jest jeden, o czym piraci dobrze wiedzą. Możemy w tej krótkiej wersji problemu oznaczyć piratów od najslackszego do najsilniejszego P_1, P_2 i tak dalej do P_{10} . Piraci wierzą również w demokrację (któroś powiedział, że wśród złodziei nie istnieje honor?!), a zatem zgadzają się, żeby najsilniejszy pirat zaproponował sposób podziału, a następnie wszyscy, łącznie z proponującym, głosują za projektem. Jeśli 50 lub więcej procent piratów zagłosuje za, to propozycja jest uważana za przyjętą i jest realizowana. W przeciwnym razie proponujący jest wyrzucany za burtę do pełnej rekinów wody i procedura rozpoczyna się od nowa, przy czym teraz proponuje drugi pod względem siły pirat.

Wszyscy piraci lubią złoto, ale bardziej nienawidzą pływania z rekami niż kochają złoto. Każdy z nich woli raczej zostać na pokładzie bez złota, niż zostać wyrzucony za burtę i odpierać ataki drapieżnych ryb. Wszyscy piraci myślą racjonalnie i wiedzą, że jeśli zniszczą którykolwiek sztukę złota, na przykład próbując podzielić ją na mniejsze kawałki, to kruszec straci prawie całą swoją wartość. Wreszcie piraci nie mogą się zgodzić na wspólne posiadanie sztuk złota, gdyż żaden z nich nie wierzy, że jego towarzysze będą respektowali związane z tym porozumienie.

Wreszcie czas na pytanie: Co powinien zaproponować najsilniejszy pirat, żeby dostać jak najwięcej złota?

Czasami najlepszy sposób analizy tego rodzaju strategii polega na pracy od końca, gdyż na tym etapie może być oczywiste, jaka strategia jest dobra, a jaka zła. Możemy następnie zdobyta tak wiedzę przenieść do sytuacji o krok wcześniejszej, a następnie kontynuować to przenoszenie dalej, gdyż każda z tych strategicznych decyzji jest skoncentrowana na tym samym pytaniu: Co zrobi następny pirat, jeśli postąpię w dany sposób? Dla dowolnych n piratów odpowiedź na to pytanie jest oparta na decyzji dla $n - 1$ piratów.

Zaczniemy zatem od końca, gdzie mamy tylko dwóch piratów P_1 i P_2 . W tej sytuacji strategia najsilniejszego pirata P_2 jest oczywista: powinien zaproponować 100 kawałków złota dla siebie i nic dla P_1 . Jego głos będzie stanowić 50% ogółu głosów, co wystarczy do przyjęcia jego propozycji, i dzięki temu stanie się on jedynym bogatym piratem!

Rozważmy teraz sytuację z trzema piratami. Zauważmy, że pirat P_1 wie (i P_3 wie, że P_1 wie!), iż jeśli propozycja P_3 zostanie odrzucona, to procedura znajdzie się w sytuacji z dwoma piratami, w której P_1 nie otrzymuje nic.

P_1 zagłosuje zatem za każdą propozycją P_3 , w której coś mu przypadnie. Wiedząc to, otrzymujemy, że optymalna strategia P_3 polega na użyciu minimalnej ilości złota dla przekupienia P_1 i zabezpieczenia jego głosu. P_3 powinien zatem zaproponować 99 sztuk złota dla siebie, 0 dla P_2 i 1 sztukę dla P_1 .

Strategia P_4 w sytuacji z czterema piratami jest podobna. Ponieważ potrzebuje on 50% głosów, musi skaptować jednego dodatkowego pirata. Tutaj również powinien użyć minimalnej ilości złota do zabezpieczenia swojego głosu, a zatem jego propozycja wygląda tak: 99 sztuk złota dla siebie, 0 dla P_3 , 1 sztuka dla P_2 i 0 dla P_1 . Oczywiście P_2 będzie szczęśliwy, głosując na tę propozycję, gdyż po wyrzuceniu P_4 za burtę procedura znajdzie się w sytuacji z trzema piratami, w której P_2 nic nie dostaje.

Strategia P_5 w sytuacji z pięcioma piratami jest odrobinę inną. Najsilniejszy pirat potrzebuje od swoich towarzyszy dwóch dodatkowych głosów. W związku z tym proponuje 98 sztuk złota dla siebie, 0 dla P_4 , 1 sztukę dla P_3 , 0 dla P_2 i 1 sztukę dla P_1 . Głosy P_3 i P_1 są tutaj oczywiście zabezpieczone dlatego, że przy scenariuszu z czterema piratami osoby te nie dostałyby nic.

Łatwo możemy teraz opisać propozycję dla P_6 w sytuacji z sześcioma piratami, dla P_7 z siedmioma piratami itd. W szczególności propozycja dla P_{10} wygląda tak: 96 sztuk złota dla siebie, po jednej sztuce dla P_8 , P_6 , P_4 i P_2 , dla pozostałych zaś nic. Tak wygląda rozwiązanie zagadki w krótkiej wersji. Dobrze jest być najsilniejszym piratem, przynajmniej gdy piratów jest mało, a złota dużo.

Przejdzmy teraz do rozszerzonej wersji zagadki. Zostawmy wszystkie dotychczasowe założenia, ale zwiększymy liczbę piratów do 500. Pojawia się tutaj ten sam wzór, ale zadanie ma tym razem haczyk, gdyż to rozumowanie działa przy liczbie do 200 piratów. Pirat P_{200} zaproponuje 1 sztukę dla siebie, 1 dla każdego pirata o parzystym numerze i nic dla pozostałych. I właśnie w tym miejscu zaczyna się cała zabawa w tej wersji zadania.

Pirat P_{201} może użyć poprzedniej strategii, ale konieczność przekupywania towarzyszy spowoduje, że dla siebie nic nie zaproponuje. Proponuje zatem po 1 sztuce złota dla każdego pirata o nieparzystym numerze od P_{199} do P_1 . W ten sposób nie zostaje mu nic, ale przynajmniej unika zjedzenia żywcem przez rekiny.

Pirat P_{202} również nic nie dostaje. Musi użyć wszystkich 100 sztuk złota do przekupienia 100 piratów i pozostańia na pokładzie. Wybór tych piratów tym razem nie jest jednoznaczny, gdyż mamy tu 101 towarzyszy, którzy są gotowi przyjąć złoto (piratów, którzy nic nie dostają w scenariuszu z 201 piratami). Mamy zatem 101 sposobów na rozdzielenie sztuk złota.

A co ze scenariuszem z 203 piratami? Pirat musi zdobyć, łącznie ze swoim, 102 głosy za swoją propozycję, ale nie ma wystarczająco dużo sztuk złota, żeby przekupić 101 spośród swoich kolegów. Zatem P_{203} wyleci za burtę niezależnie od tego, co zaproponuje. Biedaczek.

Jest to jednak ważne dla P_{204} , gdyż wie on, że P_{203} będzie głosował za każdym rozwiązaniem, żeby ocalić swe życie! P_{204} może zatem liczyć na P_{203} niezależnie od tego, co zaproponuje. Ułatwia to zadanie, gdyż może on liczyć na P_{203} , siebie i 100 przekupionych towarzyszy, może zatem zabezpieczyć

sobie 102 głosy. Znowu, przekupywani złotem powinni być ci piraci, którzy nie dostaną nic w propozycji *P202*.

Pirat *P205* w scenariuszu z 205 piratami stoi w obliczu zadania niemożliwego do wykonania. Nie może liczyć na pomoc *P203* ani *P204* – każdy z nich będzie głosował przeciwko niemu, chcąc ocalić siebie. *P205* zostanie wyrzucony za burtę niezależnie od tego, co zaproponuje. Morał jest taki: nie należy być najsilniejszym w grupie 205 demokratycznych piratów.

Taki sam los czeka *P206* – może być pewny głosu *P205*, ale to wszystko, na co może liczyć, zatem pójdzie za burtę.

Podobnie *P207* stoi przed mokrym końcem swego istnienia, gdyż potrzebuje 104 głosów: swojego, 100 od przekupionych piratów oraz 3 głosów dodatkowych. Może dostać głosy od *P205* i *P206*, ale to za mało... trafia więc za burtę.

Los pirata *P208* jest inny, gdyż potrzebuje również 104 głosów, ale *P205*, *P206* i *P207* będą za nim głosować, żeby ocalić swe życie! Głosy te, jego własny głos i 100 głosów przekupionych piratów sprawią, że jego propozycja zostanie przyjęta i *P208* przeżyje. Odbiorcami jego sztuk złota muszą być ci piraci, którzy nie dostaliby nic przy propozycji *P204*, czyli piraci o numerach parzystych od *P2* do *P200* i dalej *P201*, *P203* i *P204*.

Teraz widzimy już strukturę, która ciągnie się w nieskończoność. Piraci, którzy mogą zaproponować akceptowalne podziały (chociaż nie dostają złota, przynajmniej zostają na pokładzie), są rozdzielani coraz dłuższymi ciągami piratów, którzy są wyrzucani za burtę niezależnie od tego, co zaproponują. To wyrzucanie zaś zapewnia ich głos silniejszemu piratowi. W związku z tym piraci, którzy mogą zaproponować akceptowalny podział, to *P201*, *P202*, *P204*, *P208*, *P216*, *P232*, *P264*, *P328*, *P426* itd., to znaczy piraci, których numer jest równy 200 plus potęga 2.

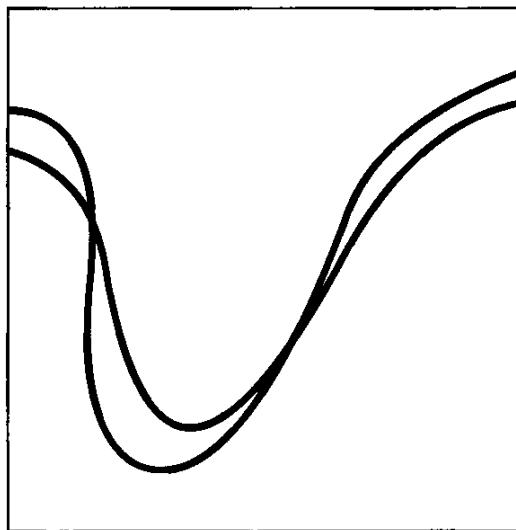
Oprócz tego łatwo jest określić, którzy piraci są szczęściarzami otrzymującymi złoto. Jak widzieliśmy wcześniej, rozwiązanie nie jest tutaj jednoznaczne; jedna z możliwości polega na tym, żeby *P201* dał złoto piratom o nieparzystych numerach od *P1* do *P199*, *P202* dał złoto piratom o parzystych numerach od *P2* do *P200*, *P204* dał złoto piratom o nieparzystych numerach, *P208* – o parzystych numerach i tak dalej na przemian, wybierając nieparzyste i parzyste.

Jak zatem widzimy na podstawie tej zagadki, bycie najsilniejszym i dysponowanie możliwością przedstawienia pierwszej propozycji nie zawsze jest korzystne, chyba że liczba piratów jest dość mała. Czasami dobrze jest być dużą rybą w małym stawie.

Na koniec przedstawimy wielką zagadkę wziętą z opowieści o Sherlocku Holmesie autorstwa Sir Arthur Conan Doyle'a [107]. Zagadka ta (która możemy znaleźć też w [265]) stawia rozwiązującego przed niewiarygodnie zagmatwaną siatką wzajemnych powiązań. Oto ona.

Sherlock Holmes został poproszony o przeprowadzenie dochodzenia w pewnej wielce tajemniczej sprawie zniknięcia dziesięcioletniego Lorda Saltire, jedynego syna i dziedzica diuka Holdernesse. Lord Saltire zniknął pewnej nocy z Priory School. Dr Huxtable, założyciel i dyrektor szkoły, poprosił

Holmesa o pomoc. W trakcie swojego śledztwa Holmes odkrył ciało zamordowanego nauczyciela języka niemieckiego, pana Heideggera. Zauważył on też odcinek błotnistej nawierzchni ze świeżymi śladami przedniego i tylnego koła roweru, wyglądającymi jak na rys. XIV.5. Holmes wiedział, że najważniejsze jest tu odgadnięcie kierunku jazdy roweru. Czy ślady te prowadziły do szkoły, czy też w kierunku przeciwnym? Holmesowi znalezienie odpowiedzi zajęło ułamek sekundy.



Rysunek XIV.5. Ślady roweru w blocie

Pojawia się tu kłopot. W wersji Conan Doyle'a [107] wyjaśnienie, które Holmes podał Watsonowi, nie jest żadnym wyjaśnieniem! Holmes „wydedukował” kierunek jazdy rowerzysty na podstawie stopnia zagłębienia śladów i stwierdził, że ciężar kierującego sprawił, iż ślad tylnego koła był głębszy. To jednak nie implikuje kierunku jazdy rowerzysty! Czy potrafisz pokazać, że jesteś lepszy od Sherlocka? Czy rowerzysta jechał z lewa na prawo, czy z prawa na lewo? Sprawiedliwości musi stać się zadość! Masz okazję się wykazać!

14. Systemy koewolucyjne

- Cóż, w *naszym kraju* – powiedziała wciąż jeszcze zdyszana Alicja – zwykle jest się w innym miejscu... jeżeli biegło się tak szybko i tak długo, jak my biegłyśmy.
- Musi to być powolny kraj! – powiedziała Królowa. Bo tu, jak widzisz, trzeba biec tak szybko, jak się *potrafi*, żeby zostać w tym samym miejscu. Jeżeli chce się znaleźć w innym miejscu, trzeba biec co najmniej dwa razy szybciej!

Lewis Carroll: *O tym, co Alicja odkryła po drugiej stronie lustra*¹

Rozwiązywanie problemu zaczyna się zwykle od ujęcia go w zrozumiały sposób za pomocą wymiernych kryteriów. Opracowuje się funkcję oceny, która precyzuje związki między poszczególnymi parametrami zadania i określa, jaka metoda lub zestaw metod zostanie użyty do znalezienia ustawień parametrów sterujących optymalizacją funkcji. Proces ten poznaliśmy na przykładzie problemu komiwojażera, spełnialności formuł boolowskich oraz ogólnego problemu programowania nieliniowego. W każdym wypadku określiliśmy funkcję oceny, biorąc pod uwagę odpowiednio odległość, prawdziwość i fałszywość podformuł czy matematyczne związki między liczbami, a także, być może, jakieś dodatkowe ograniczenia dotyczące występujących w problemie zmiennych. Dogłębne zrozumienie funkcji oceny umożliwia opracowanie poprawnej kombinacji reprezentacji, operatorów poszukiwania oraz kryteriów selekcji, która pomoże w znalezieniu ekstremów funkcji oceny i dobrego rozwiązania lub zbioru rozwiązań problemu.

Czasem jednak nie mamy pojęcia, jak określić dobrą funkcję oceny. Do problemów tego rodzaju należy odkrywanie optymalnych strategii w grach. W tego typu sytuacjach kłopot polega na tym, że nie ma dobrego pomysłu na to, jak ocenić jakość działania, poza prostym stwierdzeniem, że się wygrało, przegrało lub zremisowało. Przypuśćmy, że przyjaciel rzucił Ci wyzwanie, żebyś zagrał w grze, w którą nigdy wcześniej nie grałeś. Być może, gra jest na tyle nowa, że przyjaciel też jeszcze w nią nie grał. Co mógłbyś zrobić? Przeczytałbyś reguły, żeby zrozumieć, jakie ruchy możesz wykonywać, a jakich nie, oraz sprawdziłbyś, jaki jest cel: kryterium wygranej lub przegranej. Poza tym jesteś pozostawiony sam sobie i musisz od zera opracować swoją strategię i taktyki.

¹Przekład Maciej Słomczyński, wydawnictwo Zielona Sowa, Kraków 2002 (przyp. tłum.).

możliwe ruchy i odpowiedzi przeciwnika oraz własne kryteria decyzyjne umożliwiające preferowanie jednych ruchów kosztem innych. Jest to prawdopodobnie najtrudniejszy problem do rozwiązania – bez żadnych wskazówek, na których można by się oprzeć, żadnych pomysłów na temat tego, co robić. Musisz po prostu uczyć się metodą prób i błędów.

Sytuacja ta bardzo przypomina to, przed czym stają żywe stworzenia w naturalnym środowisku. Zwierzęta (a także wiele roślin) są idealnymi „rozwiązywaczami” zadań, przez cały czas stojąc wobec istotnego problemu, jak uniknąć stania się czyimś obiadem. Wiele stosowanych przez zwierzęta defensywnych i ofensywnych strategii przeżycia jest genetycznie na stałe wpisanych w ich zachowania instynktowne. Jaki jest jednak początek tych zachowań? Możemy prześledzić podobieństwa w strategiach przeżycia u wielu gatunków. Wiele gatunków wykorzystuje, na przykład, barwy ochronne, dzięki którym mogą wtopić się w tło w swoim otoczeniu jak patyczak i kameleon, a jednak ich strategia jest taka sama: pozostać niezauważonym. Inne zwierzęta nauczyły się, że „ich bezpieczeństwo zależy od ich stanu liczebnego”; należą do nich ryby żyjące w ławicach i zwierzęta stadne, na przykład antylopy. Co więcej, zwierzęta stadne różnych gatunków nauczyły się szukać wznieśień i ustawiać w pierścień, patrząc na zewnątrz, dzięki czemu mogą dość szybko dostrzec atakującego drapieżnika. Te skomplikowane zachowania zostały opanowane w wyniku prób i błędów wielu pokoleń – nie ulega wątpliwości, że życie i śmierć odegrały w tym procesie niepośrednią rolę.

Tak właśnie wygląda proces koewolucji. Udział w nim bierze nie tylko jeden osobnik i nawet nie jeden gatunek stawiający czoła środowisku, ale raczej wiele osobników przeciwko innym osobnikom. Każdy z nich walczy o zasoby dostępne w środowisku, które samo w sobie jest groźne i nie dba o to, które osobniki wygrają lub przegrają w walce o byt. Konkurujące ze sobą osobniki przy poszukiwaniu lepszych strategii przeżycia, które dadzą im przewagę nad przeciwnikami, używają losowego różnicowania oraz selekcji. Monarcha wędrowny¹ nauczył się, jak być trującym i zostawiać zły posmak w ustach swoich wrogów. Z kolei inny podobny motyl z gatunku *Limenitis archippus* wykształcił się tak, że nie musi być trujący – samo podobieństwo wyglądu do monarchi wystarczyło, żeby ogłupić ptaki, a było łatwiejsze do osiągnięcia przez ewolucję. Antylopy nauczyły się tworzyć pierścień, żeby szybciej dostrzec drapieżniki. Drapieżniki z kolei, na przykład lwy, nauczyły się polować w grupach oraz wykorzystywać wysokie, jasne trawy afrykańskiej sawanny do maskowania się, kiedy tropią swe ofiary. Każde usprawnienie po jednej stronie prowadzi do usprawnienia po drugiej, co jest swego rodzaju „wyścigiem zbrojeń” mającym na celu przetrwanie. Jedne osobniki starają się odpowiedzieć na wyzwanie rzucane przez innych, co stawia nowe wymagania tym drugim itd.

Bardzo uważnie wybraliśmy język, którego tutaj użyliśmy do opisu uczenia się związanego z takim procesem koewolucyjnym. Antylopy nie zbierały się, żeby przedyskutować nowe koncepcje na temat „Sposobów przeżycia w erze

¹ Gatunek amerykańskich barwnych motyli (przyp. tłum.).

lwów”, ale doszły do strategii gromadzenia się w stada i ustawiania się w pierścień obronny na wzniesieniach. Niemniej jednak tych strategii nie sposób nie zauważać i w oczywisty sposób wynikają one z jakiegoś procesu uczenia się. Tam, gdzie wyuczone zachowania są instynktowne, wyodrębnili się w wyniku losowego różnicowania i selekcji w ramach gatunku. W takich wypadkach cały genom interesującego nas gatunku jest elementem procesu uczenia się, osobniki zaś reprezentują potencjalne genetyczne wariacje na ogólny temat zachowania. Nikt nie mówił patyczakowi, żeby wyglądał jak otaczające go rośliny. Nikt nie powiedział motylowi monarsze, żeby był trujący. A jednak te i wiele innych, nawet bardziej zadziwiających wynalazków zostało wprowadzonych w naturalnym procesie koewolucyjnym, w którym nie występuje żadna jawnie podana funkcja oceny inna niż po prostu kryterium zachowania życia lub poddania się śmierci – przeważnie śmierci.

Tego rodzaju proces może wydawać się bardzo rozrzutny: „uczenie się przez śmierć”, jak go określił Marvin Minsky, zwracając się we wczesnych latach sześćdziesiątych XX wieku do jednego z pionierów obliczeń ewolucyjnych Hansa Bremermannna. Gdy jednak brakuje lepszego rozwiązania, ewolucja naturalna podpowiada metodę rozwiązywania problemów, w której dostępne mogą być tylko podstawowe informacje, na przykład dopuszczalne rodzaje ruchów oraz zasady gry. Jak pokażemy dalej w tym rozdziale, koncepcja koewolucji może też być wykorzystana, gdy mamy pewne jawne informacje zawarte na przykład w postaci funkcji oceny. Lecz by wprowadzić tę koncepcję, zaczniemy od zera.

14.1. Gry

Powyżej wspomnialiśmy po raz pierwszy o możliwości grania w pewną grę. Zanim przejdziemy do szczegółów, przyjrzymy się podstawowym elementom gier. Gry charakteryzują się regułami, które opisują dopuszczalne ruchy, które każdy gracz może wykonać. Ruchy te określają zachowanie każdego gracza, sposób, w jaki każdy z nich przydziela swoje zasoby. Bardziej formalnie – zachowanie gracza to ciąg par bodziec-reakcja, w którym zasoby są przydzielane w odpowiedzi na konkretne sytuacje. Gdy gracz wykonuje ruch, otrzymuje jakąś wypłatę, a gracz zwykle robi wszystko, by ta wypłata po pewnym czasie była jak największa. W niektórych grach, jak na przykład warcaby lub szachy, owa zapłata przychodzi na końcu gry. Łatwo jednak sobie wyobrazić zastępczą wypłatę, która jest powiązana z szansami gracza na wygranie w danym momencie i jest oparta na historii wykonanych w grze ruchów.

Niektóre gry są oparte na konkurencji, inne zaś na współpracy, a jeszcze inne mają mieszany charakter. W tym rozdziale zobaczymy różne ich przykłady. Wypłaty określają postać gry. Jeśli system wypłat wskazuje, że gdy jeden z graczy wygrywa i uzyskuje wypłatę, wówczas drugi przegrywa i jego zysk jest zmniejszany, to gra ma zdecydowanie charakter konkurencyjny. Pewne gry są skonstruowane tak, że dwie lub więcej stron może uzyskać wypłatę jednocześnie. Zastanów się nad traktatami podpiswanymi przez kraje. Każdy z sygna-

tariuszy traktatu może patrzeć na „grę”, w której uczestniczy, jak na wspólną inwestycję z rezultatem korzystnym dla wszystkich, przy założeniach, że każdy ma zamiar honorować traktat!

Ponad 70 lat temu von Neumann i Morgenstern [481] opracowali podstawy klasy gier, w której dwaj gracze biorą udział w ciągu ruchów, obaj znają wszystkie możliwe posunięcia, które każdy z nich może wykonać, i jeszcze znają dokładnie wypłaty, które każdy z nich uzyska. Zastanowiwszy się chwilę nad rzeczywistymi grami – piłką nożną, przejęciem firmy, małżeństwem – natychmiast zauważysz, jak bardzo uproszczone są te reguły, a jednak teoria gier von Neumanna i Morgensterna przyjęła się szeroko w ekonomii i innych dziedzinach.

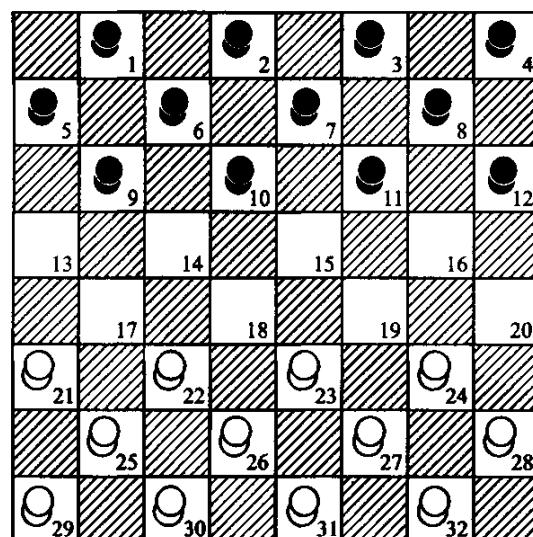
W grach z sumą zerową, w których jakikolwiek zysk jednego z graczy jest realizowany kosztem innego z nich (tak jak przy transakcjach w handlu bydłem), podstawowa strategia polega na minimalizacji maksymalnej szkody, którą przeciwnik może nam wyrządzić swoim ruchem. Strategia taka jest nazywana strategią minimaksową. Gdy ją stosujemy, sprawdzamy wszystkie możliwe odpowiedzi na nasze ruchy, a następnie wybieramy taki z nich, przy którym reakcja przeciwnika wyrządza nam najmniejszą możliwą szkodę. Jeśli w grze z sumą zerową obaj gracze stosują strategię minimaksową, to wypłata, którą każdy z nich otrzymuje, jest określana jako wartość rozgrywki lub też wartość gry. Jeśli wartość jest dodatnia, to jeden z graczy może mieć zawsze przewagę nad drugim, którego odpowiednia wypłata będzie ujemna.

Minimalizacja maksymalnej szkody, którą może wyrządzić przeciwnik, brzmi atrakcyjnie nawet w grach o sumie niezerowej. Taki sposób postępowania jest jednak zachowawczy. Przypuśćmy, że jesteś w środku gry i Ty oraz Twój przeciwnik macie dwa ruchy A i B. Jeśli obaj zagracie A, to obaj zarobicie 1 dolara. Jeśli Ty zagrzasz A, Twój przeciwnik zaś B, to zarobisz 2 dolary. Jeśli zagrassz B, a Twój przeciwnik zagra A, to zarobisz 100 dolarów, ale jeśli obaj zagracie B, to nie dostaniesz nic. Gracz minimaksowy po przeanalizowaniu sytuacji powie: „Jeśli zagram A, to wiem, że dostanę co najmniej 1 dolara. Jeśli zagram B, to mogę nie dostać nic”. To prawda, ale różnica między jednym dolarem a niczym jest tak mała, że trudno uznać za sensowne poprzestanie na wypłacie 1-dolarowej i zrezygnowanie z szansy zagrania B, w nadziei, że przeciwnik popełni błąd i zagra A, a Ty otrzymasz dużą wypłatę 100 dolarów. Strategia minimaksowa polega na unikaniu ryzyka, a w tym wypadku wydaje się działać niezgodnie ze zdrowym rozsądkiem.

Wracając do wyzwania, które chcemy podjąć w tym rozdziale, przypuśćmy, że zamierzasz zagrać w grę. Nie wiesz o niej prawie nic – znasz jedynie jej zasady i sposób rozłożenia zasobów – może tu chodzić na przykład o ułożenie pionków na planszy. W jaki sposób mógłbyś znaleźć strategię wygrywającą, która byłaby minimaksową lub jakąś inną strategią, mimo iż nie znasz ani funkcji wypłaty, ani swojego przeciwnika? W pierwszej chwili sytuacja może wydawać się zbyt trudna, gdyż nikt nie wejdzie w taką grę dla zabawy (jej opis brzmi bardziej jak opis kary). Ale zauważmy, że taka sytuacja przypomina to, co natura narzucała żywym organizmom przez miliardy lat. Jej odkrycia są genialne, a zatem takie mogą też być wytwory symulowanej ewolucji.

14.2. Uczymy się grać w warcaby

Przypuśćmy, że miałeś nauczyć się gry w warcaby, ale musiałeś w gruncie rzeczy zacząć naukę od podstawowych zasad. Podano Ci dodatkowe reguły gry, które zaraz opiszemy. Widzisz pionki na szachownicy i wiesz, ile jest Twoich oraz ile jest przeciwnika. Oprócz tego jesteś pozostawiony sam sobie. Uczyńmy tę grę jeszcze trudniejszą, przyjmując, że po skończeniu gry nie wiesz, czy wygrałeś, przegrałeś, czy też może był remis. Dopiero po zagraniu pewnej losowej liczby partii jest Ci oznajmiane, ile zarobiłeś punktów, co określa Twoją ogólną jakość działania. A więc nie tylko nie wiesz, które z wykonanych ruchów w którejkolwiek z gier były dobre, a które złe, ale nawet nie wiesz, które gry wygrałeś, które przegrałeś, a w których zremisowałeś. Prawdziwe wyzwanie! W rzeczywistości trochę prostsze wyzwanie postawił już Allen Newell, ojciec sztucznej inteligencji, we wczesnych latach sześćdziesiątych XX wieku, gdy stwierdził, że w wyniku rozgrywki w warcaby (lub w szachy) nie ma wystarczającej ilości informacji, aby maszyna mogła nauczyć się grania w tę grę (zob. [324]). Nasze wyzwanie powoduje podniesienie poprzeczki jeszcze wyżej, gdyż nie daje nam informacji zwrotnej na temat tego, które gry zostały wygrane, przegrane, a w których nastąpił remis. Tego rodzaju wyzwanie możemy podjąć z użyciem koewolucji.



Rysunek 14.1. Układ początkowy przy grze w warcaby. Obaj gracze mają po 12 pionków. Pionki mogą być w różnych kolorach, na przykład w czerwonym i białym lub czarnym i białym. Grający białymi robi ruch jako drugi

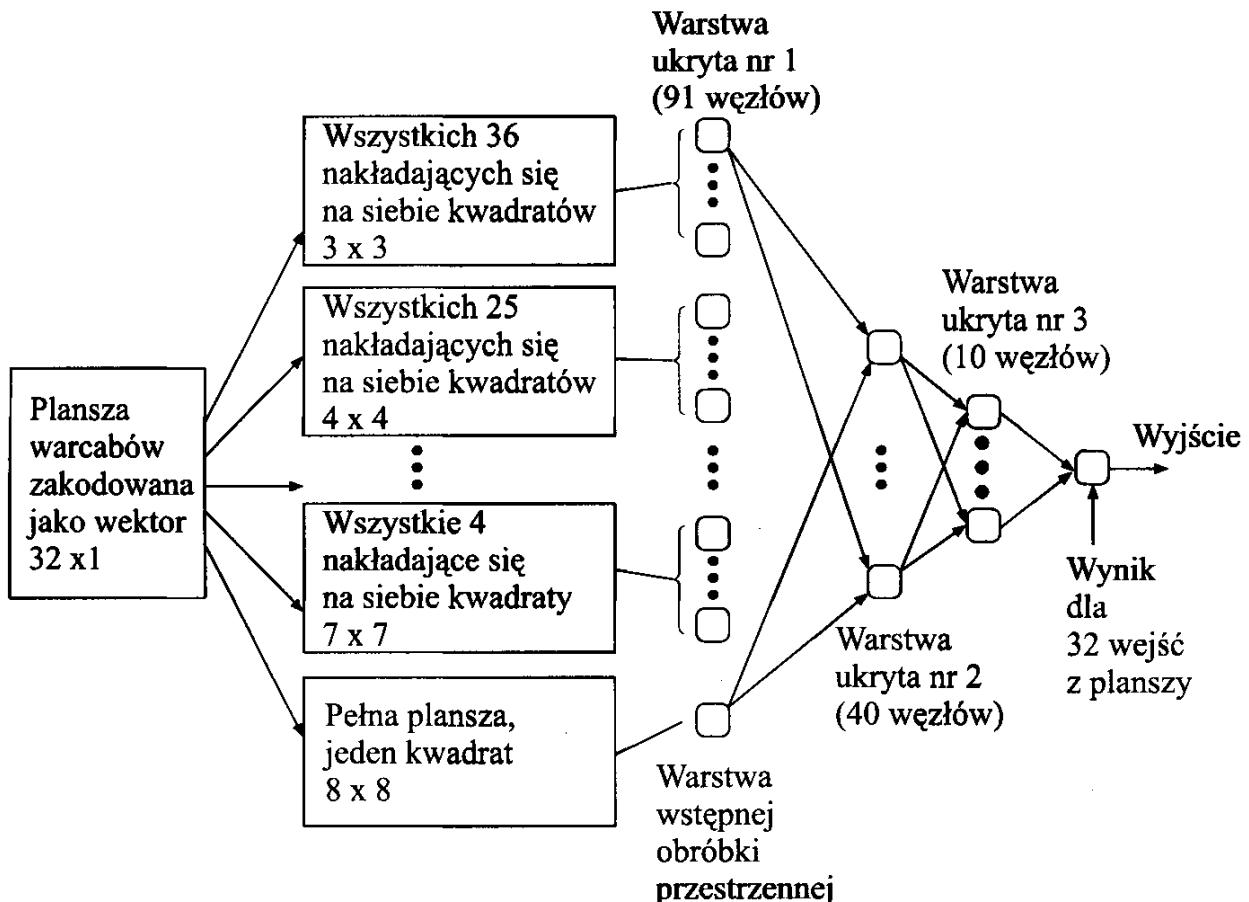
Przyjrzyjmy się najpierw regułom. W warcaby, zgodnie z tradycją, gra się na szachownicy 8×8 , mającej pola pomalowane naprzemiennie na czerwono i czarne (rys. 14.1). W grze biorą udział dwaj gracze; o pierwszym mówimy, że gra „czerwonymi”, a o drugim, że „białymi”. Każda ze stron ma 12 pionków, które początkowo są ułożone naprzemiennie na 12 polach tego samego koloru, przy czym prawe skrajne pole w rzędzie najbliższym gracza jest puste. Gra zaczyna się ruchem czerwonych, po czym gracze wykonują ruchy na przemian. Pionki mogą się poruszać po jednym polu na ukos, chyba że pionek znajduje

się tuż obok pionka przeciwnika, za którym jest wolne miejsce. Wtedy pionek może zbić pionek przeciwnika, przeskakując nad nim i jednocześnie usuwając go z planszy. Jeśli takie zbiście spowoduje, że pionek znajdzie się w pozycji do następnego bicia, bicie to musi być też wykonane i tak dalej do momentu, gdy poruszany pionek nie może wykonać już żadnego więcej skoku. Jeśli jest możliwe wykonanie bicia, to musi ono zostać wykonane przed wykonaniem ruchu bez bicia. Jeśli jednak jest możliwe wiele ruchów bijących, to gracz może wybrać, który z nich wykonać, i to nawet wtedy, jeśli różne bicia prowadzą do usunięcia różnej liczby pionków. Gdy pionek dojdzie do ostatniego rzędu na planszy, staje się damką i odtąd może poruszać się po jednym polu w dowolnym kierunku – do przodu i do tyłu¹. Gra kończy się, gdy któryś z graczy nie może wykonać żadnego ruchu, co zwykle następuje, gdy zostaje on pozbawiony wszystkich pionków. Ale może też skończyć się, gdy wszystkie pionki są zablokowane. W takiej sytuacji gracz, który nie może wykonać ruchu, przegrywa na korzyść swojego przeciwnika. Gra może się też skończyć, gdy któraś ze stron zaproponuje remis, a druga strona zaakceptuje propozycję.

Pod koniec lat dziewięćdziesiątych i na początku pierwszej dekady XXI wieku Chellapilla i Fogel [150] zaimplementowali system koewolucyjny, który nauczył się grać w warcaby na poziomie porównywalnym z grą ludzi będących ekspertami od tej gry. System ten działał w następujący sposób. Każda plansza warcabów była reprezentowana za pomocą wektora o długości 32, którego elementy odpowiadały polom dostępnym na planszy. Elementy te mogły przyjmować wartości ze zbioru $\{-K, -1, 0, +1, K\}$, przy czym: K było podlegającą ewolucji rzeczywistą wartością odpowiadającą damce, 1 odpowiadało zwykłemu pionkowi, 0 odpowiadało pustemu polu. Znak przy wartości wskazywał, czy pionek należał do gracza (plus), czy do jego przeciwnika (minus). Ruch gracza był określany na podstawie oceny zakładanej wartości potencjalnych przyszłych pozycji. Funkcja oceny była zorganizowana w postaci sieci neuronowej ze sprzężeniem do przodu, mającej warstwę wejściową, wiele warstw ukrytych i jeden węzeł wyjściowy (rys. 14.2). Na wierzchołek wyjściowy było nałożone takie ograniczenie, że mógł on dawać tylko wartości z przedziału $[-1, 1]$. Im większa była wartość, tym bardziej pozycja „podobała się” sieci neuronowej, im mniejsza, tym bardziej „się nie podobała”. Do wyboru najlepszego ruchu był wykorzystywany algorytm minimaksowy z oceną wykonywaną za pomocą sieci neuronowych.

Zastosowany system koewolucyjny zaczynał od populacji 15 sieci neuronowych, z których każda miała losowo określone wagę połączeń oraz wartość K należące do przedziału $[1, 3]$. W wyniku tego sieci neuronowe początkowo dokonywały losowych ocen, które prowadziły do losowych decyzji i to przeważnie kiepskich losowych decyzji; niektóre z nich były gorsze od innych. Każda z 15 sieci-rodziców dawała potomstwo przez wykonanie mutacji każdej z wag i war-

¹Istnieją inne warianty tej gry, w których damka może poruszać się o więcej niż jedno pole. Oficjalne reguły podawane przez European Draughts Association oraz przez United States Checkers Federation nakładają ograniczenie, w którym damka porusza się tylko o jedno pole na raz.



Rysunek 14.2. Architektura sieci neuronowej użytej przez Chellapilla i Fogla.

Warstwa wejściowa odpowiada pozycjom na planszy. Pierwsza warstwa ukryta składała się z neuronów odpowiadających kwadratowym obszarom na planszy. Pozostałe warstwy ukryte realizowały nielinowe funkcje prowadzące do końcowego wyniku. Gdy wynik był bliski +1, oznaczało to, że sieć neuronowa wskazywała na to pole. Natomiast pozycji, które dawały w wyniku wartości bliskie -1 należało unikać.

tości K , a następnie 30 sieci neuronowych grało ze sobą w warcaby tak, że każda z nich grała czerwonymi w 5 starciach. Grającym sieciom przyznawano punkty za wygraną (+1), za przegraną (-2) i za remis (0). Następnie procedura selekcji zatrzymywała 15 najlepiej ocenionych sieci neuronowych jako rodziców następnego pokolenia. Ten koewolucyjny proces grania był powtarzany przez setki pokoleń. Zauważ, że sieci neuronowe nie otrzymywały informacji zwrotnej związanej z wynikami poszczególnych gier ani też zewnętrznych ocen jakości ich ruchów. Jedyną informacją zwrotną był pełny wynik uzyskiwany po ciągu gier.

Najlepsza uzyskana tak sieć neuronowa (w 840. pokoleniu) została ręcznie przetestowana przez ludzi grających w warcaby przez Internet w darmowym serwisie warcabowym (www.zone.com) i nosiła nazwę „Blondie24” (nazwa miała przyciągnąć graczy). Po rozegraniu 165 partii Blondie24 znalazła się w czołówce 500 graczy przy 120 000 graczech zarejestrowanych w serwisie. Szczegóły związane z tym badaniem można znaleźć w [67, 68, 69, 150].

W grze w warcaby jest możliwych 10^{20} różnych pozycji – jest to znacznie więcej niż możemy obliczyć. W chwili pisania tej książki nie było wiadomo, czy warcaby są grą rozwiązywalną, co znaczy tyle, że nikt nie wiedział na pewno,

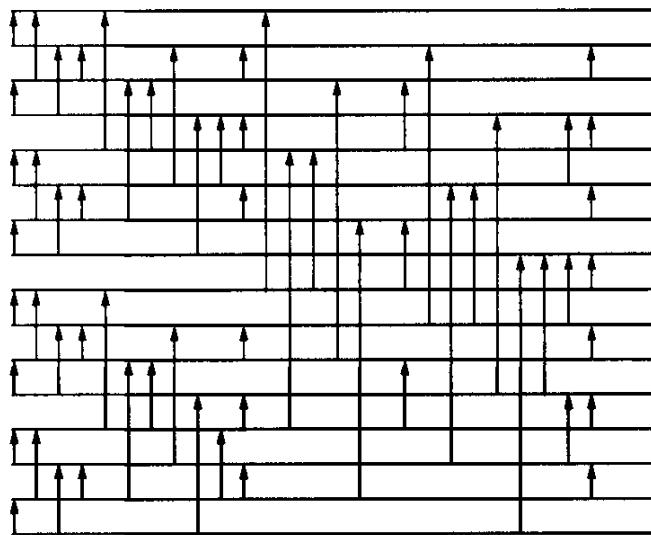
czy gra może być wygrana przez czerwone lub białe, albo czy gra zawsze powinna zakończyć się remisem. Jeśli jednak gra o 10^{20} możliwościach nie wystarcza Ci, to może wystarczą szachy z liczbą 10^{64} . Fogel i Hays [158] połączyli sieci neuronowe z koewolucją i stworzyli program grający w szachy na poziomie mistrzowskim, znowu nie zapewniając symulowanym graczom żadnej informacji zwrotnej na temat tego, które gry były wygrane lub przegrane. System koewolucyjny, po prostu grając sam ze sobą, mógł poprawić parametry strategii stosowanych przez różne sieci neuronowe i doprowadzić gry do bardzo wysokiego poziomu. Koewolucja może być uniwersalną metodą optymalizacji rozwiązań w skomplikowanych grach oraz sensownym wyborem przy badaniu użytecznych strategii, gdy mamy bardzo małą wiedzę na temat dziedziny, w której się poruszamy¹.

14.3. Optymalizacja z użyciem konkurujących populacji

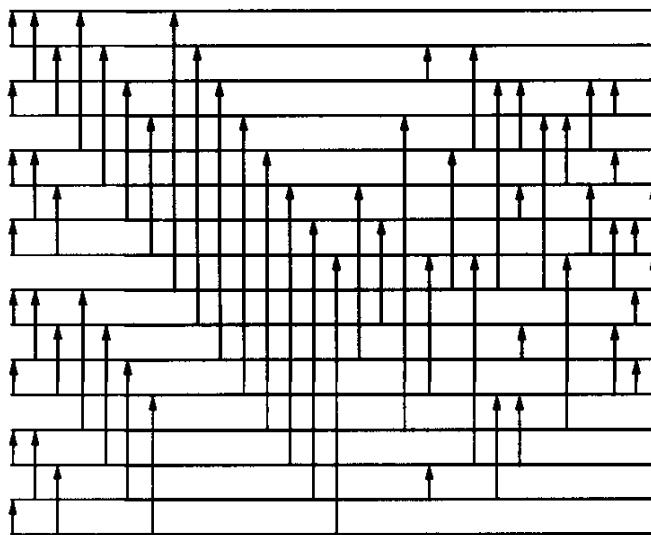
Eksperyment z Blondie24 był oparty na zasadzie koewolucji w ramach jednej populacji. W naturze jednak o przeżycie walczą nie tylko osobniki jednego gatunku, ale osobniki różnych gatunków. Czasami te zależności są określane jako relacje drapieżnik–ofiara, kiedy indziej jako gospodarz–pasożyt. W każdym razie koewolucję możemy zaimplementować jako symulację wykorzystującą wiele populacji grających po przeciwnych stronach i używanych jako zdobycz. Przyjrzymy się kilku przykładom.

W pracy [218] Hillis przedstawił obecnie słynny już przykład wykorzystujący symulację tak zwanych gospodarzy i pasożytów do poprawienia ewolucyjnej optymalizacji. Badał on problem ewolucji sieci sortującej, która jest niczym więcej jak ciągiem operacji potrzebnych do posortowania zbioru liczb o ustalonej długości. Sieć sortującą możemy przedstawić graficznie jak na rys. 14.3, gdzie uwidoczniono zaproponowaną po raz pierwszy w [34] i [138] sieć sortującą 16 elementów. Rozmieszczenie strzałek określa kolejność operacji. Każda liczba na liście do posortowania jest początkowo przypisana do jednej z poziomych linii. Między liniami połączonymi strzałkami są dokonywane kolejno porównania, a cały proces przebiega od lewa do prawa. Dwa elementy są zamieniane miejscami na liście, jeśli element przy grocie strzałki jest mniejszy od elementu przy jej drugim końcu. Po wykonaniu wszystkich porównań uwidocznionych na diagramie sortowania ostateczna lista jest uporządkowana malejąco. Zamieszczona sieć wymaga 63 porównań. Sieć z rys. 14.4, wymyślona przez Greena (w [264]), wymaga tylko 60 porównań i z tego, co wiemy, wciąż jest najlepszą znaną siecią sortującą 16 elementów.

¹Jeśli taka ludzka wiedza jest dostępna, a cel polega na opracowaniu systemu o lepszej jakości, to ta wiedza powinna zostać użyta i połączona z dodatkową wiedzą zgromadzoną przez komputer za pomocą takich metod jak koewolucja. Celem opisanych tutaj eksperymentów było pokazanie, jak dobre wyniki można uzyskać samymi metodami koewolucyjnymi, nawet gdy takiej wiedzy nie ma.



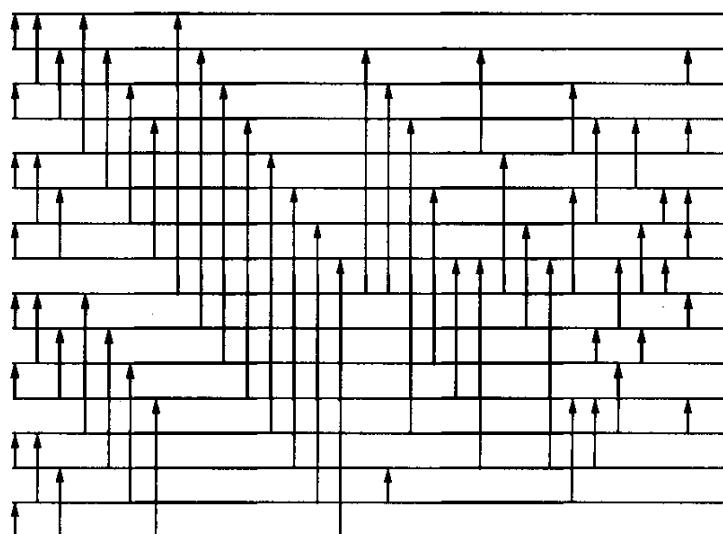
Rysunek 14.3. Sieć sortująca porządkującą 16 elementów za pomocą 63 porównań, zaczerpnięta z [218]



Rysunek 14.4. Bardziej oszczędna sieć sortująca, wymagająca tylko 60 porównań, zaczerpnięta z [218]

Hillis przeprowadził ewolucyjne poszukiwanie za pomocą struktury danych reprezentującej sieci sortujące. Celem było znalezienie minimalnej sieci sortującej 16 elementów. Sieci były oceniane na podstawie procentu poprawnie posortowanych zadań wejściowych (ciągów nieposortowanych liczb). W zaproponowanej reprezentacji oraz sposobie implementacji było wiele specyficznych niuansów. Były związane między innymi z operatorami różnicowania, selekcją i narzuconą przestrzenną bliskością rozwiązań oraz innymi aspektami algorytmu. Przechodząc do wyniku końcowego, po 5000 pokoleń ewolucji populacji o rozmiarze 65 536 osobników najlepsza sieć sortująca wymagała 65 porównań. Wybrana przez Hillsa reprezentacja wymagała co najmniej 60 porównań, nie istniała więc możliwość znalezienia znaczaco lepszego rozwiązania (o liczbie porównań mniejszej niż 60), ale 65 posłużyło jako punkt odniesienia przy korzystaniu z podejścia koewolucyjnego opartego na gospodarzach i pasozytach.

Hills zauważył, że wiele z testów sortujących użytych w początkowej ewolucji było zbyt łatwych (wiele z nich było już na początku prawie posortowanych) i prowadziło do marnowania czasu obliczeniowego. Opracował metodę, w której dwie populacje ewoluowały, konkurując ze sobą w celu przeciwdziałania tym ograniczeniom. Pierwsza populacja składała się z sieci sortujących, druga zaś ze zbiorów zadań testowych. Sieci sortujące były oceniane na podstawie przykładów testowych, które były dostępne w symulowanym lokalnym otoczeniu każdej sieci (była określona metryka odległości między przykładami a sieciami). Zestawy przykładów testowych (składające się z 10–20 elementów) były oceniane ze względu na to, jak dobrze odnajdywały wady sieci sortujących. Do każdej populacji stosowano operatory różnicowania i selekcji, w wyniku czego jednocześnie podlegały optymalizacji sieci neuronowe i przykłady testowe o coraz większych wymaganiach. Hills podał, że rozwiązanie koewolucyjne nie miało tendencji do zatrzymywania się w lokalnych optimach nawet po przejściu przez dziesiątki tysięcy pokoleń (faktyczna liczba nie została podana) i że jego procedura wygenerowała sieć sortującą o 61 porównaniach; sieć tę przedstawiono na rys. 14.5. Być może, możliwe jest znalezienie lepszej sieci.



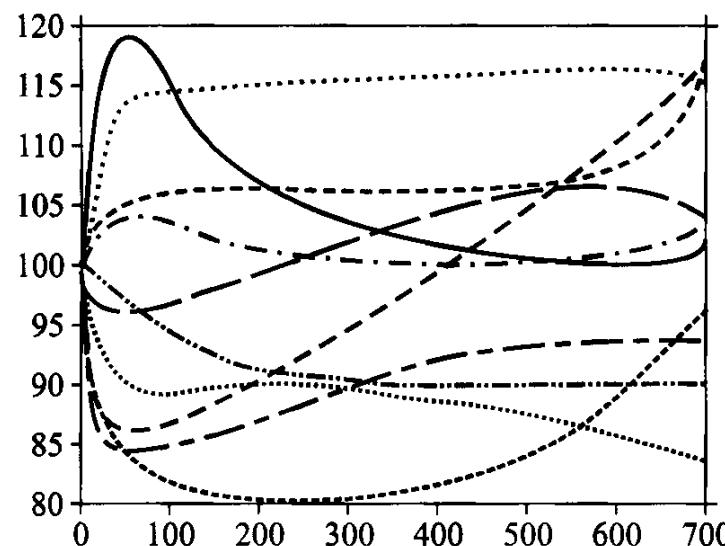
Rysunek 14.5. Sieć sortująca z 61 porównaniami, zaczerpnięta z [218]

Sieci sortujące to zaledwie początek, jeśli myślisz się o wykorzystaniu koewolucji w optymalizacji. Sebald i Schlenzig [421], na przykład, badali problem opracowania sterowanych dozowników leków dla pacjentów poddawanych operacji. Używali przy tym skomplikowanych modeli zachowań pacjentów w odpowiedzi na leki mające wpływ na ciśnienie krwi. Cel polegał na znalezieniu klasy strategii sterujących, które można byłoby stosować do wszystkich pacjentów i które umożliwiłyby utrzymywanie ciśnienia krwi pacjenta na zadanym poziomie przez określony czas. Na podstawie danych ze świata rzeczywistego, rozważań fizycznych i wcześniejszych wyników naukowych przyjęli oni, że odpowiedzi pacjenta można opisać, stosując modele

$$X[t+1] = \alpha X[t] + \beta u[t-d] + D_0 + D_1 t + D_2 t^2 + D_3 t^3$$

przy czym: $X[t]$ jest średnim ciśnieniem tętniczym (ang. *medium arterial pressure – MAP*) w milimetrach słupka rtęci (mmHg) w chwili t , d – opóźnieniem

związanym z transportem leku, u – szybkością jego wchłaniania. Parametry od D_0 do D_3 modelują wpływ czynników zewnętrznych, na przykład podawanie dodatkowych leków. Na pierwszy rzut oka odpowiedź pacjenta na lek jest modelowana za pomocą prostej funkcji liniowej, biorącej pod uwagę dotychczasowe MAP i ilość podanego leku. Utrudnienie pojawia się jednak wówczas, gdy stwierdzimy, że opóźnienie związane z transportem d jest sterownikowi nieznane. Wartość ta to czas między pierwszym wstrzyknięciem leku dożylnie, a jego zadziałaniem na pacjenta. Jeśli opóźnienie jest duże, to działanie sterownika nie będzie widoczne przez odpowiednio długi czas i może prowadzić do sytuacji, gdy sterownik będzie zbyt czuły. Powiązmy to z efektem zakłócającym, wprowadzonym przez nieznane równanie trzeciego stopnia opisywane przez parametry D_0-D_3 , a zadanie sterowania stanie się bardzo trudne. Na rysunku 14.6 przedstawiono wykresy zmian ciśnienia krwi różnych pacjentów bez podłączonego układu sterowania przy rozmaitych wartościach zewnętrznych parametrów. Wykresy te przedstawiają ciśnienie krwi pacjenta w sytuacji, gdy lek nie był podawany. Próbka ta obrazuje szeroki zakres możliwych zachowań, ale żadna linia nie pokazuje dodatkowego efektu związanego z opóźnieniem transportu.



Rysunek 14.6. Wykresy zmian ciśnienia krwi u kilku pacjentów bez podłączonego układu sterującego. Wygląd ich wynika z wartości różnych zewnętrznych parametrów. Ciśnienie krwi jest podane w mmHg jako funkcja numeru kroku czasu o długości 2 sekund [421]

Sebald i Schlenzig użyli koewolucji do optymalizacji sterownika CMAC, żeby działał dla tych pacjentów. Szczegóły związane ze sterownikami CMAC nie są omawiane w tej książce; mówiąc jednak ogólnie, są to adaptacyjne sterowniki wykorzystujące uczenie się ze wzmacnieniem, których parametry początkowe są używane do adaptacji ich zachowania na podstawie zaobserwowanych błędów w ich decyzjach. Sebald i Schlenzig skonstruowali scenariusz z dwiema populacjami, w którym ewolucji podlegała populacja pacjentów, przy czym poszukiwanie szło w kierunku określania przypadków trudnych dla sterowników CMAC, a populacja sterowników ewoluowała równocześnie w kie-

runku znalezienia parametrów o stabilnym charakterze, umożliwiających leczenie jak największej liczby pacjentów. Po zaledwie 50 pokoleniach ewolucji najlepsze sterowniki CMAC potrafiły sterować zachowaniem prawie wszystkich potencjalnych pacjentów z dokładnością do 5 mmHg wokół docelowej wartości.

Inni badacze prowadzili prace nad grami typu pogoń-ucieczka, w których myśliwy próbuje złapać uciekającego, znowu wykorzystując koewolucję. W pracy [387] Reynolds zaimplementował symulację dziecięcej gry podobnej do „berka”, w której jeden pojazd goni drugi do chwili, aż zbliży się na określona odległość do pojazdu gonionego, i w której goniący staje się gonionym i symulacja trwa dalej. Goniącemu zapewniono pewną przewagę w ten sposób, że mógł poruszać się szybciej. Koewolucja wykonywała optymalizację sterowników każdego z pojazdów, odpowiednio rozgrywając grę po każdej stronie. Podobna próba była przeprowadzona przez Cliffa i Millera w połowie lat dziewięćdziesiątych XX wieku, a niektóre graficzne przykłady z ich pracy można znaleźć na stronie <http://www.cogs.susx.ac.uk/users/davec/pe.html>. Dysponują oni filmami, które obrazują początkowe losowe zachowanie goniącego i uciekającego oraz wyuczone w wyniku koewolucji zachowanie po różnych liczbach pokoleń. Podczas gdy uciekający z dziewięćsetnego pokolenia z trudem unika goniącego z tej samej generacji, uciekający z pokolenia dwusetnego nie jest w ogóle w stanie nic zrobić. Koewolucja nie musi oczywiście ograniczać się do symulacji. Floreano i Nolfi [136] prowadzili eksperymenty koewolucyjne z gonieniem i uciekaniem dla prawdziwych robotów typu Khepera.

Co więcej, jak wspomnialiśmy już w punkcie 9.1.12, interesujące podejście do problemów związanych ze spełnianiem ograniczeń przedstawił Paredis [350]. Metoda przez niego użyta była oparta na modelu koewolucyjnym, w którym populacja potencjalnych rozwiązań ewoluowała wspólnie z populacją ograniczeń. Lepsze rozwiązania spełniały więcej ograniczeń, lepsze zaś ograniczenia były naruszane przez więcej rozwiązań. W tym podejściu nie rozróżniano rozwiązań dopuszczalnych i niedopuszczalnych. Podejście to testowano dla problemu N królowych i zostało porównane z podejściami opartymi na pojedynczej populacji [351, 350].

Przedstawiliśmy już wcześniej (w rozdziale 9) algorytm koewolucyjny (GENOCOP III), w którym rozwijały się obok siebie dwie populacje (w pełni dopuszczalnych punktów odniesienia oraz niekoniecznie dopuszczalnych punktów realizujących poszukiwanie). W systemie tym ocena punktów realizujących poszukiwanie zależy od bieżącej populacji punktów odniesienia. W tym samym rozdziale przedstawiliśmy podejście Le Riche'a i in. [280], w którym dwie populacje osobników współpracowały ze sobą i zbliżały się do optymalnego rozwiązania dopuszczalnego z dwóch stron (tzn. ze strony obszaru dopuszczalnych i niedopuszczalnych rozwiązań).

To tylko niektóre z przykładów wskazujących ogólną użyteczność podejścia koewolucyjnego przy rozwiązywaniu problemów optymalizacyjnych. Często przy odrobinie zastanowienia możemy przełożyć zwykły problem optymalizacyjny na problem koewolucyjny, a to może nierzadko przynieść jakąś korzyść.

14.4. Jeszcze jeden przykład gry

Oto jeszcze jeden przykład gry, tym razem z zakresu zastosowań wojskowych, w której komputer za pomocą koewolucji potrafił nauczyć się, jak grać. Ostatnio Johnson i in. [246] zbudowali system koewolucyjny, który nauczył się grać w wersję gry „TEMPO Military Planning Game”, polegającej na planowaniu wojskowym. Ta dwustronna gra jest używana na kursach przygotowawczych Defence Resources Management Institute do wprowadzania podstaw nowoczesnego zarządzania obroną. Zespoły graczy konkurują w budowaniu swoich sił, dzieląc ograniczony budżet w trakcie trwania kolejnych okresów budżetowych („lat”) na różne kategorie, takie jak „nabór” i „działanie” jednostek „ofensywnych” i „defensywnych”. Reguły nie są tutaj bardziej skomplikowane niż w przypadku gry w Monopol. Gracze łatwo się jednak przekonują, że widoczna na pierwszy rzut oka prostota reguł jest myląca – reguły te prowadzą do trudnych problemów decyzyjnych. Nie jest znany żaden algorytm optymalnej gry.

Melich i Johnson z U.S. Naval Postgraduate School badali planowanie struktury armii i poszukiwali sposobów na danie odpowiedzi na pytanie: Jakie należy wykonać inwestycje i kiedy, żeby osiągnąć zdolność wygrywania z nieznanym przeciwnikiem? Będąc pod wrażeniem powodzenia metod obliczeń ewolucyjnych, widocznego choćby na przykładzie programu do grania w warcaby [150], wpadli na pomysł, żeby poddać ewolucji grających w grę TEMPO i dzięki temu uzyskać wnioski na temat zachowań związanych z planowaniem i opracowywaniem budżetu. W 2000 roku Johnson zastosował podejście ewolucyjne do znacznie uproszczonej wersji gry TEMPO.

Na rzeczywisty, pełny zestaw kategorii inwestowania, które można wykonać w grze TEMPO, składają się: 1) obsługa istniejących sił, 2) werbunek dodatkowych lub zmodyfikowanych sił, 3) badania naukowe i rozwój (ang. *research and development* – „R&D”), 4) działania wywiadowcze oraz 5) działania kontrwywiadowcze. Istnieją tu cztery rodzaje sił: dwie ofensywne („siły ofensywne A i B”) oraz dwie defensywne („siły defensywne A i B”). Z każdym rodzajem są związane różne systemy uzbrojenia o rozmaitych kosztach nabycia i działania (mierzonych w „dolarach”) oraz wartości efektywności (mierzonych w „utilach”). „R&D” oznacza bieżącą inwestycję, otwierającą możliwość nabycia w którymś z przyszłych lat nowego systemu broni o prawdopodobnie lepszym stosunku ceny do siły niż w przypadku obecnie dostępnych. Inwestycje w wywiad umożliwiają uzyskanie informacji o możliwościach jednostek przeciwnika i jego inwestycjach. Łożenie na kontrwywiad osłabia znaczenie informacji uzyskiwanych przez wywiad przeciwnika.

W początkowych eksperymentach Johnsona był dostępny tylko jeden ofensywny i jeden defensywny system uzbrojenia; R&D, wywiad i kontrwywiad były pominięte. Eksperymenty przeprowadzono z użyciem systemu ewolucyjnego opisanego w języku programowania Lisp. Osobniki (algorytmy gry) były reprezentowane jako programy w prostym, podobnym do Lispu, języku i zapisane za pomocą zmiennych opisujących aktualny stan oraz operacje przydzielające w przyszłym budżecie fundusze na określone kategorie inwestowania.

Zmienne opisujące stan określają pełną kwotę dostępną w budżecie, prawdopodobieństwo wojny oraz aktualnie posiadane zapasy, aktualne ograniczenia pozyskiwania jednostek i broni, ceny oraz koszty działania jednostek ofensywnych i defensywnych. Do kategorii uwzględnianych w budżecie należały pozyskiwanie i działanie jednostek ofensywnych i defensywnych.

Algorytmy inwestowania były oceniane ze względu na dopasowanie w ten sposób, że wystawiano je w różnych grach przeciwko wybranym innym algorytmom z tej samej populacji i zapisywano wygrane oraz przegrane w sposób podobny do eksperymentów z [150]. Zastanawiano się, czy coś interesującego da się uzyskać z takiego prostego układu? I rzeczywiście dało się uzyskać. Zaczynając od początkowego pokolenia, składającego się z zupełnie losowych programów, ewolucja doprowadziła do powstania algorytmu, który przypisywał pieniądze zgodnie z podstawowymi sensownymi regułami opisanymi przez Johnsona jako „głupie, ale nie szalone” – nie próbował uzyskiwać jednostek ponad odpowiednie ograniczenia ich pozyskiwania ani uruchamiać jednostki w liczbach przekraczających to, co było dostępne; algorytm ten sprawdzał również, czy początkowe przypisanie do operacji jednostek ofensywnych nie wyczerpało dostępnych zasobów przed wykonaniem dalszych przydziałów.

Zachęceni tymi wynikami Johnson i in. w [246] opracowali program koewolucyjny, który prowadził ewolucję algorytmów inwestowania dla wersji gry TEMPO o znacznie większej liczbie cech pochodzących z pełnej gry: z wieloma typami broni, wywiadem i kontrwywiadem, ale bez R&D. W systemie koewolucyjnym dwie strony były modelowane w odrębnych populacjach. Dostępne były moduły sterujące działaniem systemu i określające zakres eksperymentów. Przygotowano zaopatrzenie zgodnie ze zmieniającymi się od gry do gry, podczas obliczania funkcji przystosowania, parametrami broni (tzn. kosztami, użytecznością, pierwszym momentem dostępności), tak by ewoluujący gracze nie „nauczyli” się, że broń, której oczekują, jest dostępna¹.

Ten system koewolucyjny obejmował populacje, środowiska oraz parametry ewolucji dla każdego z dwóch graczy X i Y. Dwie zastosowane populacje były początkowo złożone z losowo wygenerowanych osobników. Każdy osobnik był reprezentowany przez zapisany w postaci listy zbiór reguł logiki rozmytej o zmiennej liczbie reguł; wielkość ta wyznaczała fenotyp. Dwa zbiory środowisk były odczytywane na początku z dwóch plików wejściowych. Każde środowisko utrzymywało listę broni i ich parametrów, konfigurację wywiadu, kontrwywiadu, prawdopodobieństwo wojny i budżet. W czasie gry osobnik utrzymywał te dane przez środowisko, oceniał broń i odpowiednio dzielił kolejne budżety. Decyzje (ile kupić, co uruchomić) były przekazywane z powrotem do środowiska. Dwa zbiory parametrów ewolucji były także czytane z dwóch plików. Pliki te sterowały zachowaniem zastosowanego systemu koewolucyjnego. Dla każdego

¹ Można tutaj wprowadzać nowe klasy broni („ofensywne i defensywne C, D itd.”) oraz (inaczej niż w „oficjalnym” TEMPO) umożliwiać modelowanie asymetrycznych sytuacji, w których dwie strony mają różne budżety i dysponują bronią o różnych parametrach.

pokolenia każdy osobnik z pierwszej populacji grał pewną liczbę partii przeciwko losowo wybranym osobnikom z drugiej populacji. Podobnie każdy osobnik z drugiej populacji grał pewną liczbę partii przeciwko losowo wybranym osobnikom z pierwszej. Następnie każdemu osobnikowi przypisywano przystosowanie na podstawie wszystkich gier, które przeprowadził. Osobniki w populacji były sortowane ze względu na te oceny, a pewien procent najlepszych osobników był zachowywany. Osobniki te podlegały też mutacji i krzyżowaniu. Najlepszy osobnik z każdej populacji był dodatkowo rejestrowany (tzn. zachowywany) i grał przeciwko ustalonemu „zachłannemu ekspertowi”, który działał jako punkt odniesienia.

Zestaw reguł logiki rozmytej, który reprezentował osobnika, składał się ze zmiennej liczby reguł postaci „if-then”. Część „if” składała się ze zmiennej liczby wyrażeń odnoszących się do prawdopodobieństwa wojny, budżetu, parametrów broni, wyników wywiadu itp. Określały one stopień, w jakim dana reguła mogła być stosowana. Część „then” przypisywała każdemu parametrowi, do którego się stosowała (systemowi broni, rodzajowi wywiadu lub kontrwywiadu), wartość jego „sensowności”. Reguły były stosowane do każdej możliwej kategorii inwestycji; w ten sposób otrzymywano jej sumaryczną „sensowność”. Rozkład budżetu był obliczany przez liniowe przeskalowanie wartości „sensowności”, po którym następowała normalizacja.

W systemie koewolucyjnym ewolucji podlegała pełna struktura oraz wartości reguł, tzn. liczba reguł, liczba i typ wyrażeń w warunkach „if” oraz wartości „sensowności” w częściach „then”. W zestawie reguł zastosowano logikę rozmytą, ujmującą w zwarty sposób istotne nieliniowości i przedstawiającą je w postaci łatwo zrozumiałej dla ludzi. Oto przykład:

```
if [PWar IS Very Low – Low][CATEGORY IS DEFENSIVE]
    [SUBTYPE IS 1 OR 2][Inventory IS Low]
    [MaxAcquisitionUnits IS Low – Medium]
    [AcquisitionCost IS Very Low]
    [UtilsPerAcquisitionCost IS Very Low – Low]
then [Evaluation IS Low]
```

Wyrażenia w części „if” odnosiły się do 7 z 16 zmiennych, których można używać przy tworzeniu reguły dotyczącej broni: PWar jest prawdopodobieństwem wojny; CATEGORY ma wartość albo „OFFENSIVE”, albo „DEFENSIVE”; „TYPE” (nie występuje w przykładzie) to albo A, albo B, analogicznie do określeń „siły ofensywne A” lub „siły defensywne A”; „SUBTYPE” to liczba umożliwiająca rozróżnianie różnych broni tej samej kategorii i typu, jak w opisach „siły ofensywne A 1” lub „siły ofensywne A 2”. „Inventory” oznacza liczbę jednostek broni w rezerwie – jest to największa liczba jednostek, jakimi można operować (po wydaniu pieniędzy na koszty operacji); „MaxAcquisition-Units” to największa liczba jednostek danej broni, jaką można nabyć w jednym roku; „AcquisitionCost” to koszt pozyskania w dolarach na jednostkę; „Utils-PerAcquisitionCost” to stosunek wartości jednostki w utilach do kosztu jej pozyskania.

Opis postaci „AcquisitionCost IS Very Low” oznacza stopień przynależności kosztu pozyskania do pewnego zbioru rozmytego reprezentowanego wewnętrznie przez funkcję przynależności o kształcie dzwonu (gaussowską) o danym środku c oraz „szerokości” (odchyleniu standardowym) σ . Program używał wewnętrznie rzeczywistych wartości liczbowych dla c i σ , na których działały operatory mutacji i krzyżowania. Człowiekowi przedstawiano wyrażenia postaci „Very Low”, gdyż takie opisy są prawdopodobnie bardziej strawne niż pary liczb, takie jak $c = 48,7682$, $\sigma = 17,1056$. Zakres sensownych kosztów pozyskania podzielono na pięć zakresów od „Very Low” do „Very High”. W naszym przykładzie przedział $c \pm \sigma$ znajdował się wewnątrz zakresu „Very Low” dla kosztów pozyskania. Opis „Evaluation IS Low” w części „then” reguły odnosił się do wartości „sensowności”. Skoro reguła była stosowana do broni, oznaczało to, że *nie* było powodu, żeby ją kupować.

Obok systemu koewolucyjnego był dostępny system, w którym gracz-człowiek mógł rozgrywać partie z jednym z zapamiętanych graczy-automatów. Gracz-automat zarządał swoim budżetem zgodnie ze swoim zestawem reguł, a człowiek współpracował z systemem za pomocą interfejsu w postaci arkusza kalkulacyjnego. Jeden z uzyskanych w wyniku ewolucji graczy wygrał z Johnsonem dwa razy, co wywołało zarówno złość, jak i radość autora – bycie pokonanym przez własne potomstwo wywołuje mieszane emocje.

Ten system grający był stosowany na zajęciach ekonomii w U.S. Naval Postgraduate School. Wielu studentów musiało trzy lub cztery razy stanąć w szranki z programem uzyskanym w wyniku ewolucji, zanim uzyskali wynik, który chcieli poddać pod ocenę. Profesor prowadzący zajęcia (pominiemy jego nazwisko) po długich i ściśle zaplanowanych wysiłkach pokonał maszynę za pierwszym razem, ale tylko o włos. Podczas gry przypisywał ruchom maszyny wszystkie najwymyślniejsze rodzaje pobudek. Był zawiedziony, gdy dowiedział się po fakcie, że walczył z dokładnie trzema regułami w powyżej przedstawionej postaci. Krótko mówiąc, gracze uzyskani w wyniku ewolucji wykazywali jakość działania porównywalną z ludzką!

14.5. Sztuczne życie

Koewolucję możemy zastosować nie tylko do rozwiązywania problemów optymalizacyjnych, ale też do badania modeli zjawisk naturalnych, w tym pewnych aspektów życia. Pomysł badania „życia takiego, jakie mogłyby być, a nie jakieś jest” wywołał duże zainteresowanie w późnych latach osiemdziesiątych i wczesnych dziewięćdziesiątych XX wieku. Badania te określano mianem badania „sztucznego życia”. Ale tak jak ze wszystkim, co dotyczy algorytmów ewolucyjnych, korzeniami sięgają one wiele dekad wstecz, nawet do najwcześniejzych wysiłków Nilsa Barricellego z lat pięćdziesiątych XX wieku. W 1954 roku Barricelli opublikował pracę [32], w której opisał eksperyment z liczbami przemieszczającymi się w ramach pewnej siatki. Liczby te reprezentowały wędrujące organizmy. Jeśli różne organizmy wchodziły na tę samą pozycję w siatce, to stosowano do nich pewne reguły określające, czy liczby te powinny ulec

mutacji, reprodukcji lub przemieszczaniu. Wraz z upływem czasu ten prosty układ umożliwiał powstanie w siatce interesujących układów liczbowych i stał się dowodem na istnienie czegoś, co później opisano jako *własności wyłaniające się* (ang. *emergent properties*) – nieprzewidywalnych zachowań wyłaniających się jako wynik współdziałania reguł lokalnych.

W 1969 roku Conrad przedstawił bardziej rozwinięty model symulujący ekosystem [76, 77]. Model ten był oparty na pojęciu *świata*, który był zorganizowany jako jednowymiarowy, cykliczny ciąg *miejsc*. Między te miejsca były rozdzielone zasoby zwane *żetonami*, a 200–400 organizmów było inicjowanych, aby „żyły” w tym świecie w określonych miejscach. Kodowanie organizmów było dość zawikłane i w istocie obejmowało program komputerowy wykonujący ciąg instrukcji określających zachowanie każdego organizmu. Organizmy wykorzystywały obecne w środowisku *żetony* jako źródło energii do poprawienia ich wewnętrznego stanu, naprawiania się oraz przechodzenia z miejsca na miejsce. Był to, być może, pierwszy program ewolucyjny z zależnościami hierarchicznymi między symulowanymi zachowaniami organizmów oraz związanym z nimi programem genetycznym bez podanego jawnie kryterium oceny jakości rozwiązań. Zamiast tego odpowiedniość różnych zachowań wynikała ze wzajemnego oddziaływania organizmów w poszczególnych miejscach modelowanego świata.

Wiele lat później Ray stworzył symulację noszącą nazwę *Tierra* [382], w której programy współdziałały ze sobą w dwuwymiarowej siatce. Programy te, napisane w asemblerze, posiadały funkcję replikacji samych siebie, przy czym krótsze programy replikowały się szybciej niż długie. W ten sposób pojawił się wewnętrzny nacisk selekcyjny na krótsze programy, które konkurowały o miejsce przy zapełnianiu siatki. Ray zauważył jednak, że oprócz tej cechy związanej z selekcją pojawiły się nowe programy, które służyły jako pasożyty swoich gospodarzy. Pierwotnie programy napisane przez Raya były rozłożone na trzy moduły: jeden określał długość programu, drugi monitorował pętlę reprodukcyjną, a ostatni faktycznie wykonywał operację kopiowania programu do nowej pozycji w siatce. Programy-pasożyty powstałe w wyniku ewolucji włączały pierwsze dwa moduły i odrzucały trzeci. Zamiast tego wykorzystywały konstrukcję wskaźnikową, dzięki której zapożyczyczały procedury ze starszych programów-gospodarzy umożliwiające im kopowanie się. W ten sposób tak długo, jak długo w siatce istniały programy-gospodarze, te o wiele krótsze programiki mogły wygrywać konkurencję z nimi dzięki swojej większej szybkości replikacji. Te i inne cechy eksperymentu *Tierra* były ponownie własnościami, które wyłoniły się w nieprzewidziany sposób z symulacji.

Jeśli, jak wiele innych osób, uważasz tego rodzaju eksperymenty za fascynujące, powinieneś zatrzymać się nad życiem, jakie może być, jest zajmujące, ale książka ta jest poświęcona rozwiązywaniu problemów, powróćmy zatem do gier i przyjrzyjmy się kolejnemu, związanemu ze sztucznym życiem ujęciu, w którym pojawia się prawdziwy problem do rozwiązania.

14.6. Gry ze współpracą i konkurowaniem

Przypuśćmy, że grasz w grę, a Twoje zadanie polega na maksymalizacji wypłaty. Jak wspomnieliśmy wcześniej, nie wszystkie gry polegają na konkurowaniu. W niektórych sytuacjach możesz uzyskać większą wypłatę, gdy współpracujesz z innym graczem, a nie próbujesz z nim konkurować. W innych wypadkach sytuacja może być bardziej skomplikowana. Na przykład jedną z prostych gier o tego rodzaju skomplikowanej dynamice jest już poznany przez nas w rozdziale 11 iterowany dylemat więźnia (ang. *iterated prisoner's dilemma* – IPD). Przypomnijmy, że w podstawowej wersji IPD występują dwaj gracze, z których każdy ma do wyboru albo współpracować, albo zdradzić. Współpraca oznacza dążenie do zwiększenia zysku przez obu graczy. Zdrada oznacza próbę zwiększenia własnego zysku kosztem drugiego gracza. Gra ta ma „niezerową sumę”, co oznacza, że zysk jednego gracza niekoniecznie oznacza odebranie go drugiemu, oraz jest grą „bez współpracy”, co oznacza, że gracze nie komunikują się zawsze przy określaniu swoich ruchów.

Jak już widzieliśmy, wypłaty możemy zapisać w macierzy 2×2 (tab. 14.1). Z dylematem więźnia mamy do czynienia, gdy na wypłaty R , S , T i P są nałożone następujące ograniczenia:

$$(1) \quad R > \frac{S + T}{2}$$

$$(2) \quad T > R > P > S$$

Pierwsze ograniczenie zapewnia, że nieoczywista staje się potrzeba współpracy oraz że wypłaty z ciągu ruchów polegających na współpracy są większe od wypłat z ciągu ruchów na przemian współpraca-zdrada i zdrada-współpraca (co może oznaczać bardziej skomplikowaną formę współpracy [5]). Drugie ograniczenie zapewnia, że zdrada jest dominującym działaniem, a także zapewnia, że wypłaty wynikające ze wzajemnej współpracy są większe niż ze wzajemnego zdradzania.

Tabela 14.1. Ogólna postać macierzy wypłat dla dylematu więźnia. Pozycje postaci (A, B) reprezentują wypłatę odpowiednio dla A i B. Litery R , S , T i P oznaczają zmienne określające wypłatę przy różnych kombinacjach ruchów. R to „nagroda” (ang. *reward*), którą obaj gracze otrzymują, współpracując, T to „pokusa” (ang. *temptation*) związana z wypłatą w sytuacji, gdy pierwszy gracz zdradzi, a drugi będzie współpracował. S to wypłata związana z byciem „naiwniakiem” (ang. *sucker*), gdy pierwszy gracz współpracuje, a drugi zdradza. P to „kara” (ang. *penalty*), która jest nakładana na obu graczy, gdy obaj zdradzą

		Gracz B	
		Współpraca	Zdrada
Gracz A	Współpraca	(R, R)	(S, T)
	Zdrada	(T, S)	(P, P)

Jeśli zamierzasz w tej grze wykonać tylko jeden ruch, to zdrada jest jedyną sensowną rzeczą, jaką możesz zrobić. Jeśli jednak gra jest powtarzana i gracze wykonują ciąg ruchów, to zdradzanie nie jest takie atrakcyjne, jak można się przekonać, wykonując koewolucyjną symulację.

W 1987 roku Axelrod [15] przeprowadził symulację, w której strategie w IPD były reprezentowane jako tabele oparte na trzech poprzednich ruchach wykonanych w grze z użyciem macierzy wypłat przedstawionej w tab. 14.2. Strategia w grze była ciągiem złożonym z 64 wystąpień wszystkich kombinacji poprzednich posunięć wykonanych przez obu graczy w trzech poprzednich rundach, jak również przypadków, gdy gracze właśnie zaczynali i nie mieli jeszcze dostępnej historii trzech ostatnich ruchów. Na przykład jednym z elementów strategii gracza mogłyby być polecenie [JA(W,W,W) i PRZECIWNIK(W,W,W) oznacza W], przy czym JA odnosi się do trzech ruchów wykonanych przez tę strategię, PRZECIWNIK oznacza ruchy drugiego gracza, W oznacza współpracę.

Tabela 14.2. Konkretna macierz potencjalnych wypłat w dylemacie więźnia

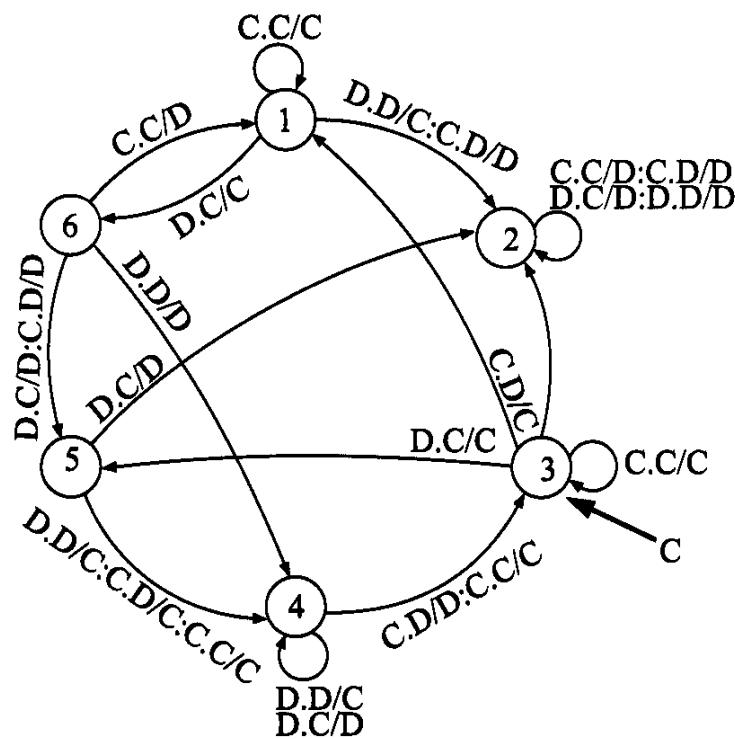
		Gracz B	
		Współpraca	Zdrada
Gracz A	Współpraca	(3, 3)	(0, 5)
	Zdrada	(5, 0)	(1, 1)

Axelrod ułożył symulację ewolucji, w której strategie konkurowały ze sobą w systemie karuzelowym (każdy grał z każdym możliwym przeciwnikiem) przez 151 ruchów przy każdej potyczce. Strategie zarabiały punkty za każde spotkanie, a po przejściu przez wszystkie możliwe połączenia używano selekcji proporcjonalnej, dzięki której dalej przechodziły te, które uzyskały lepsze wyniki. Nowe warianty strategii były tworzone za pomocą mutacji (zmieniającej wynik związany z określoną kombinacją ruchów) oraz krzyżowania jednopunktowego. Po wykonaniu 10 prób Axelrod dokonał dwóch interesujących obserwacji. Pierwsza z nich dotyczyła tego, że we wczesnych pokoleniach średni wynik u rodziców, którzy przeżyli, był mniejszy, co oznaczało, że większość tych, którzy wygrywali, zdradzała. W dalszym ciągu jednak średni wynik rodziców, którzy przeżywali, zwiększył się i zaczął zbliżać się do 3, co oznaczało, że populacja nauczyła się, że należy współpracować. Nikt jednak nie wskazywał rozwiązaniom, że mają to robić – własność ta wynikła z samej symulacji i pokazała, że wzajemna współpraca może się pojawić między graczami nawet wówczas, gdy zdradzanie jest racjonalnym wyborem przy „jednoetapowym” dylemacie więźnia.

Druga obserwacja dotyczyła postaci strategii, które podlegały ewolucji. Wiele z nich przypominało taktykę zwaną „wet za wet”, która jest bardzo prosta, ale jednocześnie bardzo efektywna. Przy pierwszym ruchu gra się na współpracę, a przy następnych naśladuje się to, co robił partner przy poprzednim ruchu. Jeśli przeciwnik stale współpracuje, to współpracuje także gracz

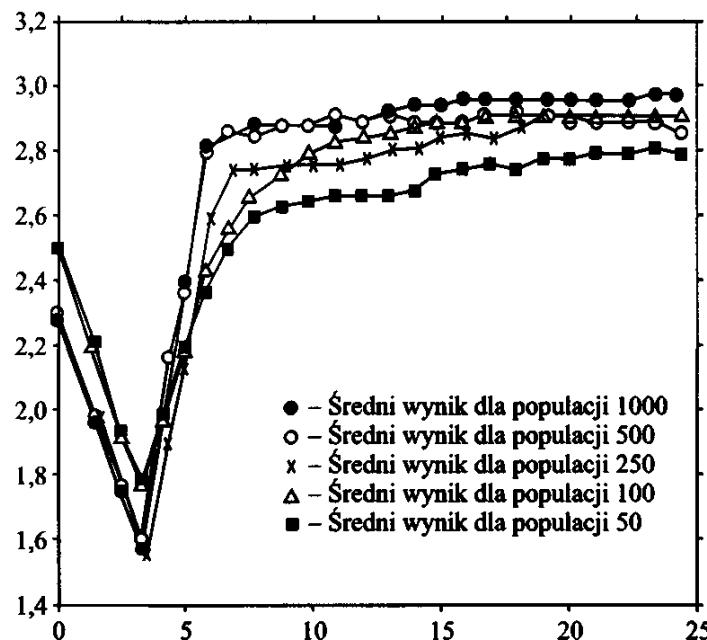
stosujący „wet za wet”. Jeśli przeciwnik wykorzystuje współpracę przy pierwszym ruchu i zdradza, to gracz stosujący „wet za wet” weźmie odwet i zdradzi za następnym razem i będzie tak robił do momentu, aż przeciwnik znów zacznie współpracę. W ten sposób żadna inna taktyka nie może zdobyć zbyt dużej przewagi nad „wet za wet”. Wcześniej Axelrod przeprowadził dwa turnieje gier stosujących IPD z udziałem strategii opracowanych przez badających teorię gier naukowców z całego świata. Za każdym razem jednak wygrała strategia „wet za wet”. W związku z tym było interesujące, a nawet dodające otuchy, zaobserwowanie, że taka strategia wypłynęła z jego prostej symulacji.

Ciekawe jest też określenie wpływu innych reprezentacji na strategie. W 1993 roku, na przykład, Fogel badał IPD, zastępując tabele w rozwiązaniu Axelroda automatami ze skończoną liczbą stanów [146, 148]. Na rysunku 14.7 przedstawiono przykład automatu ze skończoną liczbą stanów wykorzystywanego jako strategia w IPD. Zastosowano taką samą procedurę prowadzenia gry między strategiami, nowe strategie były jednak tworzone w wyniku mutacji rodziców, którzy przeżyli, przez losowe dodawanie lub usuwanie stanów, zmianie przejść do następnego stanu, symboli wyjściowych, stanu początkowego lub ruchu początkowego. Na rysunku 14.8 przedstawiono średnie wyniki dla populacji o rozmiarach od 50 do 1000 rodziców. Wyniki są w istocie takie same i bardzo zbliżone do tego, co zaobserwował Axelrod: po początkowej tendencji do zdradzania się nawzajem następował wzrost wzajemnej współpracy. Wynik okazał się odporny na tego rodzaju zmianę w reprezentacji.



Rysunek 14.7. Automat ze skończoną liczbą stanów do gry w iterowany dylemat więźnia [146]. C oznacza współpracę (ang. *cooperate*), D oznacza zdradę (ang. *defect*), pogrubiona strzałka pokazuje stan początkowy

Może to dawać nadzieję, że nawet gdy ludzie mają podjąć decyzję, na których obie strony mogą skorzystać, w końcu zaczynają współpracować. Na

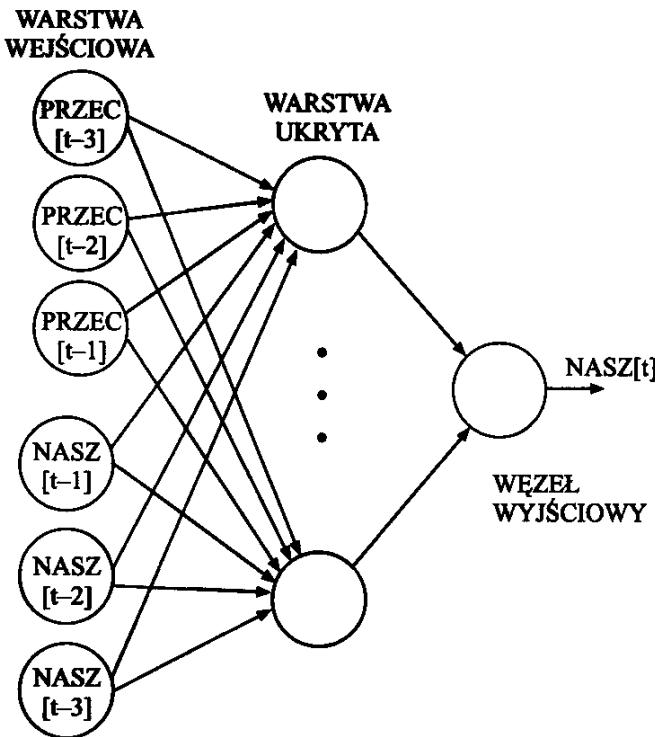


Rysunek 14.8. Wyniki Fogla dla ewolucji automatów ze skończoną liczbą stanów grających w iterowany dylemat więźnia [146]. Wykres przedstawia średnią wyników wszystkich rodziców jako funkcję numeru pokolenia

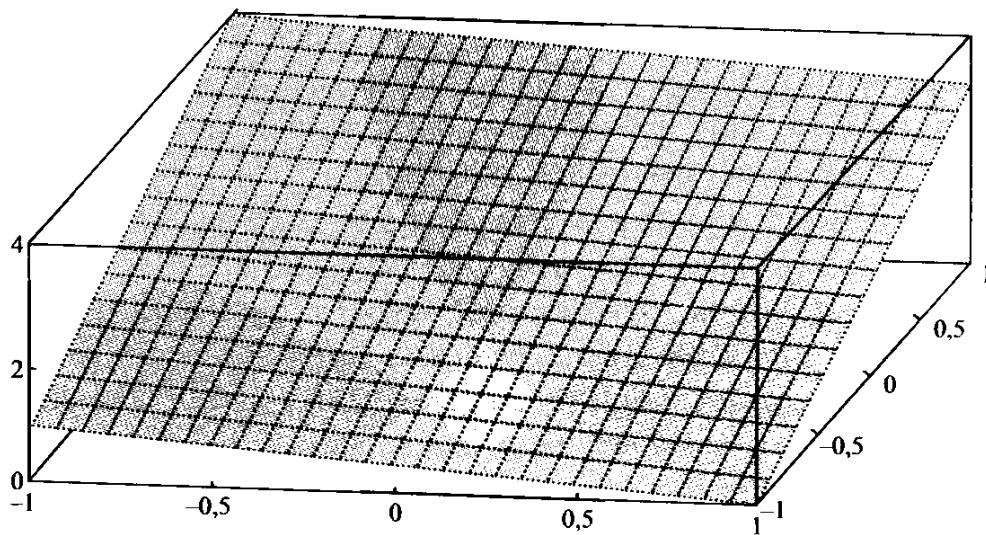
podstawie własnych doświadczeń możesz podchodzić do tego sceptycznie. Zawsze pojawia się wiele sytuacji, w których ludzie są przyłapywani na „zdradzie” popełnionej w taki lub inny sposób skuszeni zyskiem, podczas gdy współpracując, mogliby osiągnąć zadowalający przychód.

Jedno z możliwych wyjaśnień uzyskanego wyniku jest takie, że w standartowym IPD są możliwe tylko dwa ruchy: współpraca i zdrada. W rzeczywistym świecie pojawiają się jednak różne stopnie współpracy i zdrady. Zakres możliwych ruchów jest ciągły, a nie dwubiegowy. Harrald i Fogel [208] przebadali wersję IPD, w której gracze mogli wykonywać ruchy zgodnie ze skalą liczbową od -1 do 1, przy czym -1 oznaczała pełną zdradę, 1 pełną współpracę. Wartości te podawano na wejścia sztucznych sieci neuronowych wykorzystywanych do reprezentacji strategii. Każda z tych sieci miała sześć węzłów wejściowych – po jednym dla każdego z trzech poprzednich ruchów obu graczy (rys. 14.9) oraz ukrytą warstwę neuronów, która miała zmienny rozmiar. Takie sieci neuronowe generowały liczby między -1 a 1 z wypłatami rosnącymi zgodnie z pokazanym na rys. 14.10 liniowym przybliżeniem funkcji wypłat z tab. 14.2. Sieci neuronowe, które przeżywały w każdym pokoleniu dawały potomstwo w wyniku losowej mutacji Gaussa, stosowanej do każdej wartości określającej wagę i zniekształcenie (w tej chwili powinieneś być już dość dobrze zaznajomiony z tego rodzaju operacjami) i każda gra była znowu grana przez 151 ruchów.

Wyniki w tym nowym systemie zupełnie nie przypominały tego, co było obserwowane uprzednio. Kilka typowych przykładów uwidoczniono na rys. 14.11. Niektóre z nich przypominają wykresy, jakie widzieliśmy na rynku akcji po marcu 2000 roku (strome spadki). Zauważ, jak bardzo wykresy te różnią się od wykresów z rys. 14.8. Wszystkie odpowiadają wynikom uzyskanym przy 20 ukrytych węzłach w sieciach neuronowych, a zatem strategie były



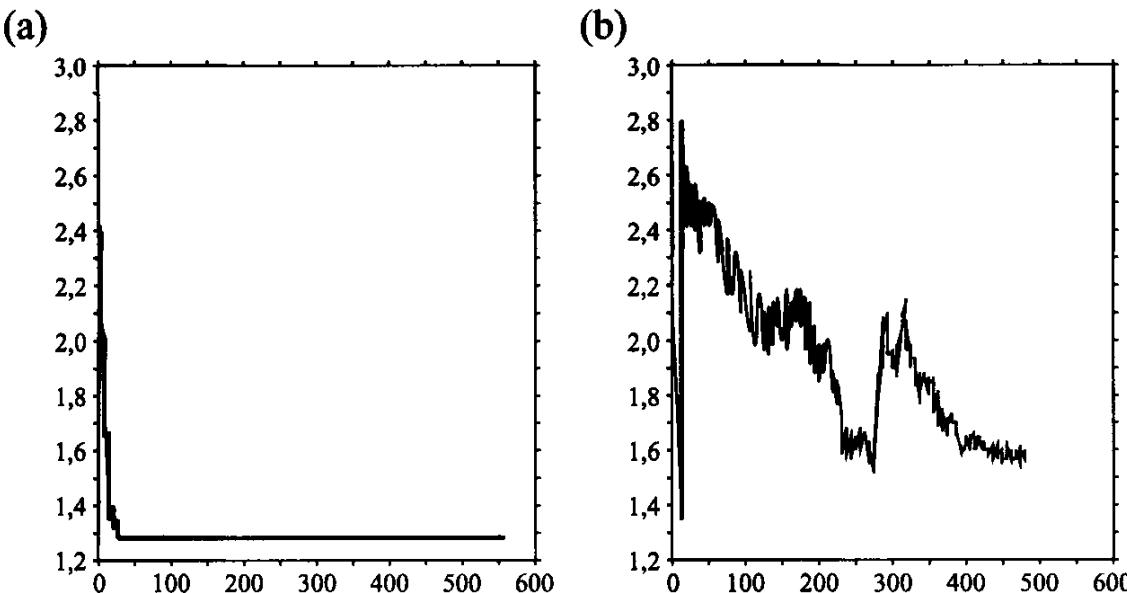
Rysunek 14.9. Sieć neuronowa użyta do gry w ciągłą wersję iterowanego dylematu więźnia



Rysunek 14.10. Przybliżenie za pomocą płaszczyzny funkcji zysku użytej przez Axelroda dla dylematu więźnia [208]; oś pionowa określa wypłatę dla ruchów wykonanych zgodnie z osią poziomą

potencjalnie bardzo skomplikowane. Znacząca większość prób dawała wyniki wskazujące tendencję zmniejszania wypłaty, a nie zwiększania jej. Tu pojawia się pytanie: co powoduje taką zmianę?

Darwen i Yao [82] przebadali odmianę IPD, w której zastosowali osiem możliwości wyboru zamiast dwóch (tzn. zdrada lub współpraca). Ich wyniki były bardzo podobne do podanych w [208] i autorzy doszli do tych samych wniosków. W pierwszym z nich stwierdzili, że w miarę jak rośnie liczba możliwości wyboru w grze, zmniejsza się przeglądana część całej macierzy gry. Przy tylko dwóch możliwościach wyboru ewoluujące strategie przeszukują w zasa-



Rysunek 14.11. Typowe przykłady średniego przychodu u przeżywających rodziców przy zastosowaniu sieci neuronowych w iterowanym dylemacie więzienia [208]. (a) Typowy wynik, gdzie wszystkie podlegające ewolucji sieci neuronowe szybko wpadły we wzajemne zdradzanie. (b) Inny typowy wynik, gdzie początkowa współpraca zamieniła się we wzajemne zdradzanie się. Oba wykresy przedstawiają średni wynik rodziców w zależności od numeru pokolenia

dzie cały zbiór czterech możliwych wyników. W badanym przez nich przypadku z ośmioma opcjami wiele z 64 możliwości nie było branych pod uwagę przy przeszukiwaniu. Darwen i Yao pokazali, że każdy podzbiór 2×2 macierzy wypłat 8×8 sam stanowił dylemat więźnia, a zatem było możliwe, że ewoluujące rozwiązania stabilizowały się na ciągu ruchów, który był czymś w rodzaju lokalnego optimum dla większej macierzy 8×8 . W ich drugim wniosku, który prawdopodobnie lepiej możemy określić jako obserwację, zauważyl, że gdy IPD miał więcej możliwości wyboru, strategie ewoluowały ku dwóm typom, które od siebie zależały, umożliwiając sobie nawzajem uzyskiwanie dużych wypłat, podczas gdy gra przeciwko członkom własnego typu nie dawała tak dobrych rezultatów. Taka wzajemna zależność była zadziwiająco odporna na napływ innych strategii, które potencjalnie mogły powodować zwiększenie zysku w populacji.

Obserwacje Darwena i Yao są bardzo interesujące, ale w dalszym ciągu nie wyjaśniają konsekwentnego zmniejszania się wypłat widocznego przy zastosowaniu ciągłego zakresu możliwych opcji. Mimo to tych kilka przykładów ilustruje pojawiające się o wiele większe bogactwo możliwości związane ze stosowaniem koewolucji do badania powstawania strategii w bardziej skomplikowanych grach. Co roku są publikowane setki prac na temat dylematu więzienia, a w bardzo wielu historycznych publikacjach korzystano na różne sposoby z algorytmów ewolucyjnych. W tych i w wielu innych badaniach wskazano na potencjał związany ze stosowaniem symulacji koewolucyjnych do prostych i złożonych gier bez współpracy i o niezerowej sumie.

14.7. Modelowanie inteligentnych agentów

Każdy gracz w IPD jest „agentem”, którego celem jest zarezerwowanie zasobów, żeby osiągnąć swoje cele w bieżącym stanie środowiska. Co jednak będzie, gdy na agentów tych spojrzelibyśmy w taki sposób, jakby mieli oni własne cele oraz własny mechanizm ewolucyjny poprawiania swego zachowania przy dążeniu do tego celu? Zamiast jednej lub dwóch populacji koewoluujących agentów mielibyśmy ich setki. Tego rodzaju podejście zostało przyjęte w pracy [154] w ramach ujęcia zwanego *problemem El Farol*.

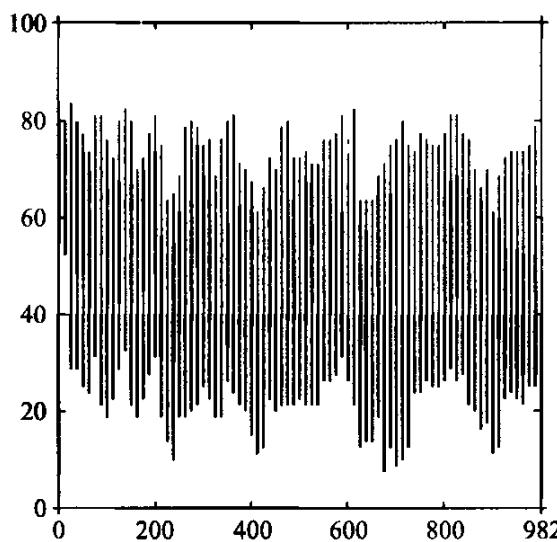
Problem ten wprowadził Arthur [13] na podstawie anegdoty związanej z barem El Farol w Santa Fe w Nowym Meksyku, oferującym w czwartkowe wieczory irlandzką muzykę. Przypuśćmy, że mamy 100 miłośników irlandzkiej muzyki i że każdy z nich niezależnie w losowy sposób decyduje się, czy pewnego czwartkowego wieczoru przyjść do El Farol. Przypuśćmy dalej, że każdy z uczestników bawi się dobrze, jeśli w barze znajduje się mniej niż 60 osób. W przeciwnym wypadku jest zbyt tłoczno i nikt się nie bawi. W tym momencie każdy z miłośników muzyki staje wobec problemu przewidzenia, ilu ludzi znajdzie się w barze. Wypad do baru ma sens tylko wówczas, gdy zgodnie z przewidywaniem znajdzie się tam mniej niż 60 ludzi.

W tym momencie sprawa staje się trochę zawiła. Jeśli większość ludzi przewiduje, że bar będzie pusty, to przyjdą i zamiast pustki będzie tłum, powodujący kiepską zabawę dla wszystkich. Jeśli zaś większość przewidzi, że bar będzie pełny, to pozostałą w domach, ale ominie ich doskonały wieczór z irlandzką muzyką. Bar, aby jakoś zaradzić problemowi, ogłasza co tydzień informacje o frekwencji, żeby można je było uwzględnić w modelach przewidywania. Zadanie dla każdego z agentów polega zatem na wykorzystaniu tych danych i przewidzeniu frekwencji w następnym tygodniu i to w sytuacji, gdy pozostałych 99 miłośników muzyki robi to samo.

Pomysł Arthura polegał na tym, żeby zamodelować każdą osobę jako agenta obliczeniowego, który ma do dyspozycji pewną liczbę różnych modeli przewidywania – takich jak przewidywanie tej samej frekwencji w następnym tygodniu lub używanie średniej z ostatnich czterech tygodni. Wśród tych dostępnych możliwości, które może mieć każdy agent, niektóre modele będą zachowywały się lepiej niż inne. Każdy agent, żeby przewidzieć sytuację, mógł użyć najlepszego według swoich doświadczeń modelu, a następnie albo udać się do baru, albo pozostać w domu. Modele w trakcie trwania procesu nie ulegały zmianie u żadnego z agentów, ale to, który model jest najlepszy, mogło zmienić się w każdej chwili ze względu na fluktuacje cotygodniowej frekwencji, wynikające ze stosowania przez innych agentów ich własnych modeli. Co ciekawe, w wyniku symulacji Arthur stwierdził, że protokół ten spowodował, że do baru wybierało się średnio 60 agentów, przy czym rzeczywista liczba dość stabilnie fluktuowała wokół tej średniej.

Być może, bardziej interesujące jest jednak to, co się dzieje, gdy każdy agent może w trakcie każdego tygodnia symulacji prowadzić ewolucję własnych modeli, co opisano w [154]. Każdy agent mógł korzystać z 10 modeli-rodziców w postaci autoagresywnych średnich ruchomych oraz 10 modeli-potomków oraz

używać 10 pokoleń różnicowania i selekcji w każdym tygodniu w celu optymalizacji modeli przed obliczeniem samego przewidywanego wyniku frekwencji w barze i decyzją zostać, czy pójść do baru. Tym razem zamiast stabilnej fluktuacji frekwencji równej około 60 osób, liczba gości zmieniała się w sposób nieprzewidywalny, co pokazano na rys. 14.12. Wynik Arthura z ogólnie stabilną liczbą uczestników imprezy przy ustalonym zestawie modeli zmienił się w silnie zmienne zjawisko, gdy agenci mogli prowadzić adaptacyjną ewolucję swoich modeli. Co więcej, w pracy [154] pokazano, że frekwencja jest tutaj dobrze modelowana przez łańcuchy Markowa, w których stan stabilny nie jest oczekiwany. Dodanie do systemu ewolucji, a koewolucji w szczególności, może prowadzić do wielu niezamierzonych konsekwencji.



Rysunek 14.12. Typowy przykład rozkładu frekwencji (frekwencja jako funkcja od liczby tygodni) w barze, gdy każdy z agentów może prowadzić ewolucję swoich własnych modeli przewidywania przyszłej frekwencji [154]

Szeroko rozumiana dziedzina opracowywania modeli inteligentnych agentów ma potencjalny wpływ na projektowanie środowiska biznesowego, a także systemów edukacyjnych. W pracy [47] Blair przedstawił analizę możliwości wynikających ze wzajemnego oceniania studentów i nauczycieli. W systemie tym nauczyciele byli oceniani przez studentów, co powodowało, że mieli tendencję do zaniżania standardów. Działo się to na takiej zasadzie, że studenci dawali wyższe oceny, a nauczyciele oceniali ich jako „lepszych” (słowo *lepszych* jest w cudzysłowie, ponieważ studenci ci byli lepsi nie w sensie nabytej wiedzy lub umiejętności jej stosowania, ale w sensie ocen na papierze). Blair wykazał, że przy pewnych prostych założeniach związanych z kosztem przygotowania wykładów oraz indywidualnymi i społecznymi zyskami związanymi z uczeniem można wyciągnąć smutny wniosek, że najlepsza rzecz, jaką może zrobić nauczyciel, to rutynowo obniżać standardy, kupując sobie w ten sposób lepsze oceny studentów za cenę ich złego przygotowania, co z dużym prawdopodobieństwem prowadzi do ponurej spirali obniżania jakości po obu stronach. Modele koewolucyjne wypłat studentów i nauczycieli uczestniczących w tego rodzaju systemach wzajemnego oceniania mogą wskazywać, które z nich mogą prowadzić do

katastrofalnych skutków i które do polepszania całej współpracy między studentami a nauczycielami.

14.8. Zagadnienia związane z koewolucją

Koewolucja może być bardzo cennym narzędziem optymalizacji. Jak dla każdego narzędzia ważne jest, żeby rozumieć, kiedy go używać, kiedy nie używać i na jakie przeszkody można natrafić przy jego stosowaniu. Jedna z możliwych przeszkód przy pomyślnym stosowaniu koewolucji jest związana ze stagnacją rozwiązań na wartości o jakości mniejszej od oczekiwanej. Stać się tak może z wielu różnych powodów. Niektóre z nich są związane z omówionym wcześniej zagadnieniem przedwczesnej zbieżności w optymalizacji funkcji (zob. punkt 11.2.1). Na przykład zbyt silne zdanie się na rekombinację lub inny operator różnicowania ograniczony stopniem zróżnicowania w aktualnej populacji może skutkować zawężonym poszukiwaniem w ramach dostępnej dziedziny problemu. W koewolucji z konkurencją, a zwłaszcza przy układzie z dwiema populacjami – takim jak proces z drapieżnikami i ofiarami – może to prowadzić do stagnacji. Nacisk na usprawnienie populacji pojawia się, gdy konkurencyjna populacja jest sprawniejsza. Jeśli jedna z populacji popada w stagnację, to samo dzieje się z naciskiem selekcyjnym dla drugiej, a co za tym idzie postęp w przeciwej populacji zwalnia lub wręcz się zatrzymuje. W rezultacie nacisk zwrotny na pierwszą populację też się zmniejsza, w wyniku czego kończy się wyścig zbrojeń i przekształca się w coś przypominającego obojętność, w której żadna ze stron nie dba za bardzo o pokonanie przeciwnika.

W minionej dekadzie pojawiła się hipoteza, że głównym czynnikiem sukcesu metody koewolucyjnej jest zastosowanie środowiska stochastycznego [48]. Jednym z przykładów, który był co najmniej umiarkowanym sukcesem, była próba uzyskania w wyniku ewolucji strategii dla tryktraka [356, 355]. Tryktrak to bardzo interesująca gra, a wynika to częściowo z tego, że dostępne graczom opcje są określone rzutami kostką, a niekiedy słabsi gracze mogą uniknąć porażki z lepszymi po prostu w wyniku pomyślnego ciągu rezultatów rzutu kostką. Stochastyczny charakter gry oraz jej niestabilność co do możliwości przewidzenia zwycięzcy stanowią nieodłączny element gry. W związku z tym nie jest tutaj możliwe, żeby strategie stabilizowały się na określonych deterministycznych ciągach ruchów, które na ogół nie muszą być bardzo efektywne, ale wystarczają przy istniejącym poziomie konkurencji. Świeższe rezultaty dotyczące warcabów i szachów [67, 68, 69, 150, 158, 258] wskazują, że pomyślne zastosowanie ewolucji nie wymaga użycia gry stochastycznej nawet wówczas, gdy gra jest skomplikowana i wymaga stosowania wielu różnych stanów. Można jednak pokazać, że środowiska stochastyczne pomagają przy postępie koewolucji, przy czym nie są one konieczne do jej powodzenia – podobne zjawisko zachodzi także dla środowisk ergodycznych (czyli takich, gdzie każdy możliwy stan może być osiągnięty z każdego – zob. [48]).

Istnieją jeszcze inne powody stagnacji w koewolucji. Patrząc na rzecz probabilistycznie, algorytm ewolucyjny znajduje najłatwiejsze działające strategie.

A dokładniej, jeśli istnieje prosta strategia, którą można szybko odkryć, i bardziej skomplikowana, złożona strategia, która jest trudniejsza do stworzenia, ale równie dobra lub, być może, nawet efektywniejsza, to ewolucja najprawdopodobniej najpierw znajdzie i wykorzysta tę prostszą strategię. Z punktu widzenia koewolucji z konkurencją stwarza to kolejną trudność, gdyż populacje pełne prostych strategii mogą nie być w stanie za pomocą losowego różnicowania błyskawicznie przenieść się do obszarów bardziej złożonych, ale lepszych rozwiązań.

Przypuśćmy, że w IPD jedną ze stosowanych strategii jest strategia „wet za wet”. Przekonaliśmy się, że jest to bardzo prosta strategia, którą możemy ująć w dwóch spartańskich zasadach. Oznacza to, że jest bardzo mało prawdopodobne, żeby udało się, niezależnie od użytej reprezentacji, za pomocą typowych operatorów różnicowania zmienić tę strategię w bardzo skomplikowaną, a zarazem efektywną strategię, która korzysta z, powiedzmy, pamięci ostatnich 12 ruchów. Najbardziej prawdopodobna droga wiodąca do skomplikowanej strategii będzie prowadziła przez ciąg kroków pośrednich, z których każdy reprezentuje rozwiązanie o rosnącej złożoności. Z rosnącą złożonością często jednak łączy się mniejsze prawdopodobieństwo odkrycia czegoś efektywnego lub bardziej poprawnie – czegoś bardziej efektywnego od prostych rozwiązań-rodziców, z których to powstaje.

Sytuacja ta przypomina nieco „paragraf 22” – można łatwo znaleźć proste, ale być może mało efektywne rozwiązania, lecz potem okazuje się, że te rozwiązania nie dają podstawy do przejścia do innych bardziej złożonych i bardziej efektywnych rozwiązań. Nie należy przy tym liczyć na szanse utworzenia złożonych, bardzo efektywnych rozwiązań z niczego, wpadamy więc w pułapkę, gdyż gdziekolwiek nie zaczniemy, mamy duże szanse na zatrzymanie się na rozwiązaniach o mniejszej złożoności i o, być może, gorszej efektywności. Może się też okazać, że proste rozwiązania wyczerpią wszystkie dostępne struktury danych reprezentujące rozwiązania [451] (np. w prostej sieci neuronowej raczej wszystkie wartości wag i progów będą miały wpływ na wynik niż znajdą się połączenia zbyteczne). W takiej sytuacji trudne może się okazać pomyślne różnicowanie dobrej strategii, gdyż oznacza zmianę parametrów, które aktywnie biorą udział w generowaniu zachowań decydujących o jej wartości.

Opracowano wiele koncepcji, które miały zapobiec tego rodzaju sytuacjom. Jedna z nich jest związana z premiami zawartymi w indywidualnych funkcjach wypłaty odpowiadającymi za złożoność strategii. Nie korzystamy tutaj z zasady oszczędności, która faworyzuje najprostsze możliwe rozwiązania, ale dajemy pierwszeństwo bardziej skomplikowanym strategiom w nadziei, że ta dodatkowa komplikacja stanie się potencjalnym źródłem różnorodności, które można by wykorzystać. Alternatywnie, proces koewolucyjny możemy zacząć od prostych rozwiązań i, korzystając z losowego różnicowania, zwiększać długość kodu rozwiązania, na przykład przez dodanie do zestawu uzupełniających reguł lub opatrzonych wagami połączeń do sieci neuronowej, lub stanów do automatu ze skończoną liczbą stanów. Przy takim postępowaniu podstawa kodowania pozostaje taka sama, dzięki czemu rozwiązania zapamiętują swoje wcześniejsze zachowania, które już wykazały się efektywnością, ale jest dodawany także

nowy podlegający różnicowaniu materiał genetyczny, który nie wpada w konflikt z tym, co już zostało osiągnięte. W istocie proces ten może dojść do rozwiązań, które z natury nie zapominają tego, czego się nauczyły w poprzednich pokoleniach poddawanych różnicowaniu i selekcji.

Kolejna koncepcja umożliwiająca unikanie ewolucyjnej stagnacji jest prosta: zapisujemy najlepsze rozwiązania w każdym pokoleniu lub w pewnych odstępach, a następnie korzystamy z nich jak z „alei gwiazd” [398]. Powstałe w wyniku ewolucji nowe rozwiązania mogą następnie być konfrontowane nie tylko z innymi ewoluującymi rozwiązaniami, ale także z najlepszymi reprezentantami poprzednich pokoleń umieszczonymi w „alei gwiazd” (w tak zwanym „turnieju mistrzów” [136]). Taki mechanizm wymusza na rozwiązańach, żeby pamiętały zachowania, które były efektywne we wcześniejszych zmaganiach. W przeciwnym razie byłoby możliwe, żeby zachowania takie były zapominane, a najlepszy reprezentant z wczesnego pokolenia mógłby pokonać strategię z pokolenia późniejszego nawet przy użyciu wydawałoby się prostego triku taktycznego, który został już wcześniej pokonany przez pośrednie populacje [355, 136]. Koncepcję tę możemy rozszerzyć na system z jeszcze ostrzejszymi kryteriami, zwany „turniejem rozwiązań dominujących” [451], gdzie najlepsze rozwiązania są sprawdzane przy każdym pokoleniu, a rozwiązanie jest określone jako dominujące, jeśli może pokonać wszystkie dominujące rozwiązania z poprzedniego pokolenia. Odnajdowanie nowych rozwiązań dominujących wskazuje na prawdziwy postęp ewolucji systemu koewolucyjnego.

Jeszcze inny pomysł na wymuszenie postępu w koewolucji polega na zastosowaniu skończonej długości życia rodziców, którzy przeżywają w każdym pokoleniu. Zabezpiecza to ewolucję przed sytuacją, w której rodzic może ulokować się w części przestrzeni rozwiązań, w której tworzy potomstwo kiepskiej jakości, na czym bezpośrednio korzysta, pokonując je. Zjawisko takie zostało odnotowane w [146, 148]. Zapewniając sobie, że rodzice przynajmniej raz na jakiś czas wymierają, chwilowa utrata, być może, bardzo dobrych rozwiązań może być zrekompensowana dalszym postępem powstały na bazie potomstwa jego sukcesorów, które bez takiego mechanizmu by się nie pojawiły. Tego rodzaju sytuacje zależą od problemu, a zatem przy ich stosowaniu konieczne jest stwierdzenie eksperymentalne lub za pomocą analizy, czy mają sens.

Jedna z wertych wspomnienia procedur poprawiania własności optymalizacyjnych symulacji koewolucyjnych, w których bierze udział wiele populacji konkurencyjnych ze sobą rozwiązań (np. w układzie gospodarz-pasożyt), polega na uważnym doborze przeciwników, z którymi się konkuuruje [397]. Powróćmy w naszych rozważaniach do przykładu z koewoluującymi modelami pacjentów na stole operacyjnym oraz modelami strategii sterowania dozowaniem leków regulujących ciśnienie krwi. Strategie sterowania były tam oceniane na podstawie tego, jak dobrze utrzymywały ciśnienie krwi na zadanym poziomie docelowym dla każdego testowanego pacjenta. Dla kontrastu pacjenci byli oceniani na podstawie tego, jak szybko doprowadzali do awarii sterownika, gdyż w tym wypadku byliśmy zainteresowani w znajdowaniu pacjentów stawiających największy

sze wymagania. Przypuśćmy, że w populacji istnieje wiele strategii sterowania, z których każda może sprostać znacznej większości pacjentów. Chociaż z punktu widzenia zbliżania się do ulepszonych rozwiązań brzmi to dobrze, to taka sytuacja implikuje też, że z pacjentami jest związana niewielka ilość informacji, którą możemy użytecznie zastosować. Jeśli wszystkie sterowniki mogą być użyte dla wszystkich pacjentów, to nie ma nacisku selekcyjnego, który mógłby odróżnić jeden sterownik od drugiego. Podobny argument może dotyczyć pacjentów, którzy powodują awarie wielu sterowników.

Jeden z pomysłów na zwiększenie szans dalszego koewolucyjnego postępu w tego rodzaju sytuacjach polega na zmniejszaniu liczby członków każdej populacji w zależności od liczby innych jej członków, którzy mogą pokonać konkretnego osobnika w przeciwej populacji. Pomoże tutaj przykład. Przypuśćmy, że dla konkretnego pacjenta w populacji prawidłowo zachowuje się 97 ze 100 sterowników w populacji konkurującej. Zamiast przypisać każdemu z tych 97 sterowników sukces o wartości jeden punkt, możemy każdemu z nich przypisać $1/97$ punktu. Na ogół, jeśli mamy N osobników, którzy mogą prawidłowo potraktować przeciwnika (cokolwiek oznacza prawidłowo), to wynik przypisany każdemu z nich wynosi $1/N$. W ten sposób, jeśli osoby umieją prawidłowo potraktować w walce jakiegoś przeciwnika, na ogół dla innych trudnego, to te osoby otrzymują więcej za swoje wysiłki. Podobnie przeciwnicy, których łatwo pokonać, uzyskują tylko niewielki ułamek punktu.

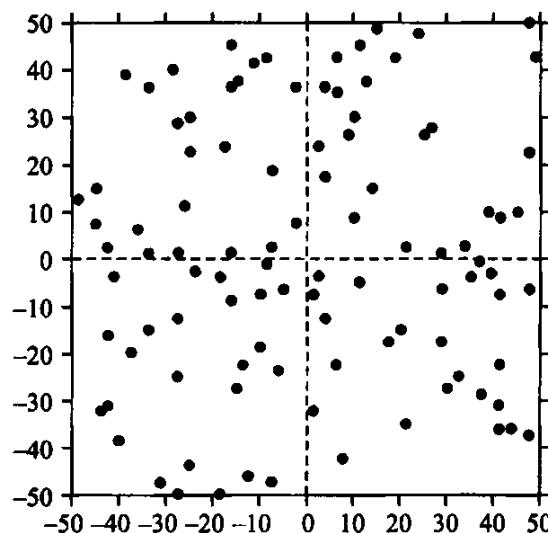
Pomysł ten przypomina trochę zakłady wzajemne (ang. *pari-mutuel*) na wyścigach konnych. Wypłaty zależą od ilości pieniędzy postawionej na każdego konia. Jeśli koń jest oczywistym czempionem, to szansa z nim związana jest mniejsza i to nie dlatego, że ktoś myśli, że prawdziwe szanse wygrania przez tego konia są duże, ale dlatego, że dużo ludzi na niego postawiło. Gdy ten koń wygra, wypłata będzie mała. Jeśli każdy może ocenić, który koń wygra, to prawie nie opłaca się na niego stawiać. Dla kontrastu powód, dla którego konie o małej szansie na zwycięstwo dają duże wygrane, to mała liczba dolarów na nie postawionych. Jeśli możemy wybrać konia, który nie jest faworytem, i wygrać, to sytuacja ta przypomina nieco sytuację sterownika, który prawidłowo odmierza lek pacjentowi, podczas gdy reszcie sterowników się to nie udaje. Taki sterownik otrzymuje właśnie dodatkowe punkty.

Analogiczny pomysł polega na skupieniu się na wybieraniu takich przeciwników, którzy mogą sprawiać największy kłopot dla jak największej liczby osobników. Takie rozwiązanie może oszczędzić czas przy przeprowadzaniu procesu konkurencji, zmniejszając liczbę spotkań między takimi rozwiązaniami, że o jednym z nich wiadomo, iż jest słabe w porównaniu z wszystkimi uczestnikami. Jednym ze sposobów realizacji tego pomysłu jest losowe wybieranie przeciwników proporcjonalnie do ich przystosowania; wartość ta jest obliczana za pomocą przed chwilą opisanej metody. Bez wątpienia Ty sam możesz także opracować jakieś inne metody lub rozszerzyć je, żeby obejmowały inne aspekty rozwiązywania problemów z użyciem współpracy, takie jak nauczenie poszczególnych neuronów, żeby współpracowały przy problemach rozpoznania wzorców [290, 288, 289].

14.9. Przykład koewolucji ze współpracą w zastosowaniu do TSP

Przy jeszcze innym podejściu do koewolucji dąży się do opracowania rozwiązań, które współpracują przy rozwiązywaniu różnych części problemu. Tego rodzaju koewolucja jest określana jako koewolucja ze współpracą. Metoda ta korzeniami sięga co najmniej pracy [360]. Istnieją też nowsze przykłady jej stosowania [126, 127, 361]. Główny pomysł polega na rozłożeniu dużego problemu na mniejsze części i prowadzeniu ewolucji dla tych mniejszych kawałków, przy czym sztuczka polega tu na tym, że ocena, jaką otrzymuje rozwiązanie, zależy od innych rozwiązań, z którymi jest ono łączone w celu uzyskania rozwiązania całego problemu. Nie próbuje się przy tym bezpośrednio rozdzielać wartości żadnego konkretnego rozwiązania między częścią problemu. Zamiast tego każde rozwiązanie otrzymuje wypłatę na podstawie tego, jak dobrze radzi sobie z problemem cały zespół, do którego należy.

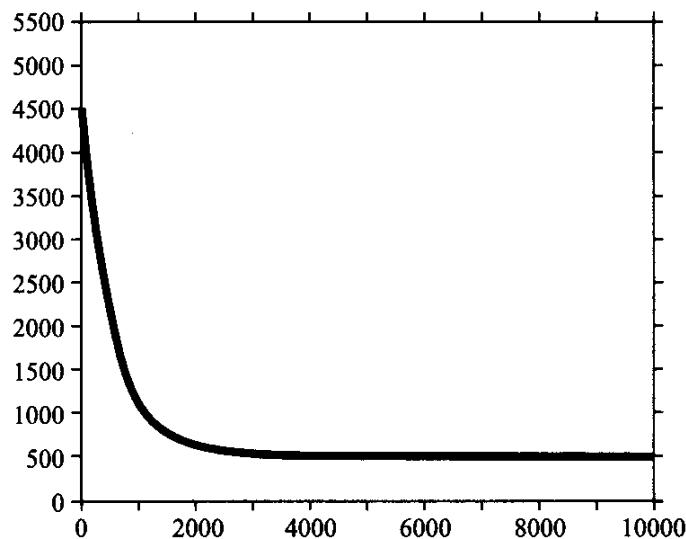
Pokażemy teraz przykład takiego podejścia do problemu komiwojażera. Przypuśćmy, że w naszym zadaniu występuje 100 miast. Na potrzeby tego przykładu rozmieszczać równomiernie losowo po 25 miast w każdej ćwiartce dostępnego kwadratu powierzchni. Rozkład miast przedstawiono na rys. 14.13.



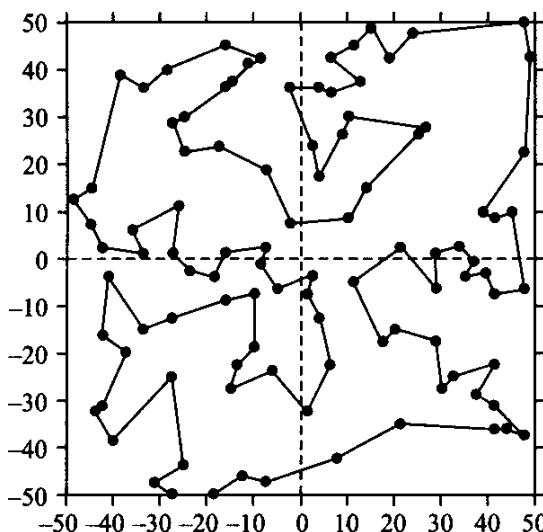
Rysunek 14.13. Rozkład 100 miast w naszym przykładzie TSP. Każda ćwiartka ma 25 miast

Moglibyśmy zająć się problemem znajdowania trasy o minimalnej długości dla tych miast za pomocą prostego podejścia ewolacyjnego. Jako punkt odniesienia przy mierzeniu efektywności dla tego przykładu zastosowaliśmy reprezentację w postaci permutacji przy 50 zainicjowanych losowo rodzicach. Każdy z rodziców generował potomków za pomocą operatora inwersji, który wybierał losowo dwa punkty w permutacji i odwracał kolejność miast między tymi punktami. 100 rozwiązań w populacji w każdym pokoleniu było ocenianych ze względu na ich długość. Pięćdziesiąt rozwiązań o najmniejszym wyniku (najlepszych) było zachowywanych jako rodzice do następnego pokolenia; cały proces był powtarzany przez 10 000 pokoleń. Na rysunku 14.14 pokazano szybkość optymalizacji najlepszego rozwiązania w populacji jako funkcję liczby pokoleń,

a na rys. 14.15 – najlepsze rozwiązanie uzyskane w wyniku ewolucji. Całkowita długość trasy dla tego rozwiązania wynosi 833,813 jednostek.



Rysunek 14.14. Szybkość optymalizacji najlepszego rozwiązania TSP przy traktowaniu tego problemu jako problem holistyczny. Wykres przedstawia koszt (długość trasy) jako funkcję numeru pokolenia



Rysunek 14.15. Najlepsze rozwiązanie uzyskane w wyniku ewolucji trwającej 10 000 pokoleń

Teraz dla odmiany zobaczymy, jak możemy rozwiązać ten problem za pomocą koewolucji ze współpracą. Podzielmy problem na cztery podproblemy, z których każdy ma własną populację i każdy jest związany z oddzielną ćwiartką obszaru zadania. Mamy jedną populację dla pierwszej ćwiartki, jedną dla drugiej itd. Każda populacja ma 50 rodziców i będzie miała 50 potomków – po jednym na rodzica. W dalszym ciągu korzystamy z reprezentacji w postaci permutacji, teraz jednak każda permutacja jest listą miast z określonej ćwiartki odpowiadającej konkretnej populacji. Każdy członek każdej populacji jest tworzony losowo jako bezładna lista numerów od 1 do 25 odpowiadających miastom z tej ćwiartki. Trasy są tworzone przez wybieranie po jednej permutacji z każdej

z czerech populacji i łączenie ich w pełną trasę złożoną ze wszystkich miast. Początkowo dla każdej z populacji tworzymy 100 rozwiązań, a następnie wszystkie je oceniamy i zachowujemy najlepsze 50 z nich.

Ocenę każdego rozwiązania w każdej populacji obliczamy w ten sposób, że przechodzimy przez listę wszystkich rozwiązań (wyznaczoną za pomocą „indeksu listy”) i dla każdej permutacji łączymy ją z rozwiązaniami losowo wybranymi z pozostałych populacji. W ten sposób bierzemy pod uwagę po jednym reprezentancie z każdej populacji, ale w tym momencie oceniamy tylko permutację wybraną za pomocą bieżącego indeksu listy. Permutację tę będziemy nazywali *permutacją indeksowaną*. Wynik jest oparty na całkowitej długości całej trasy zdefiniowanej za pomocą czterech permutacji. Aby zakończyć budowanie trasy, musimy połączyć ze sobą omawiane cztery permutacje.

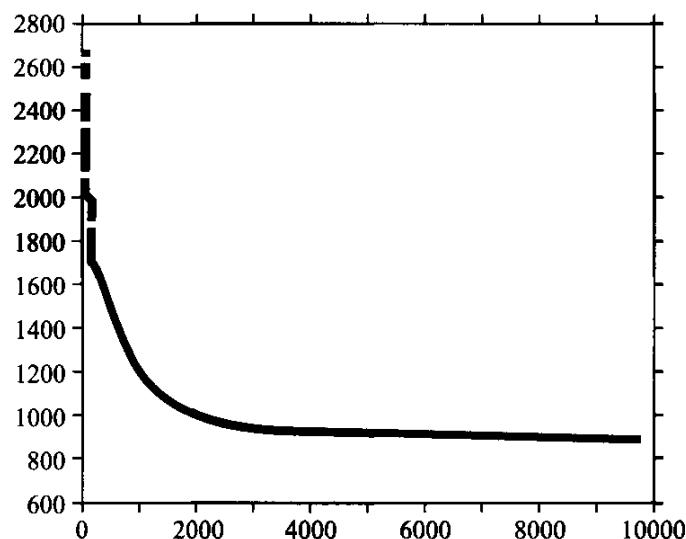
Przypuśćmy, że stosujemy regułę łączenia ćwiartki 1 z ćwiartką 2, 2 z 3, 3 z 4 i 4 z 1. Podejście to może nie jest najlepsze, ale wystarcza na potrzeby tego przykładu. Żeby ukonkretnić przykład, przyjmiemy przy inicjowaniu, że będziemy zaczynać od pierwszej ćwiartki i znajdziemy miasto o najmniejszej wartości współrzędnej y , w rzeczywistości jest to miasto najbliższe drugiej ćwiartce (poźniej w trakcie ewolucji połączenia między ćwiartkami będą się zmieniać). Możemy przenieść to na ostatnią pozycję w permutacji z pierwszej populacji, reprezentującej ostatnie miasto w pierwszej ćwiartce, zanim przejdziemy do drugiej ćwiartki. Możemy teraz przejść do drugiej ćwiartki i znaleźć miasto o największej wartości współrzędnej y oraz przenieść je na początek permutacji wybranej z drugiej populacji, żeby mogło być pierwszym przystankiem w drugiej ćwiartce. Możemy tę procedurę kontynuować przez wszystkie ćwiartki, aż zostanie zdefiniowany początek i koniec każdej permutacji. Cała trasa jest tworzona w ten sposób, że przechodzimy przez pierwszą permutację od początku do końca, następnie łączymy ją z początkiem drugiej permutacji i przechodzimy od początku do końca i tak dalej, aż trasa powróci do początku pierwszej permutacji. Całkowita długość trasy jest przypisywana permutacji indeksowanej. Proces ten jest powtarzany dla każdej następnej permutacji, aż zostanie ocenionych wszystkich 400 permutacji (po 100 w każdej populacji).

Gdy zostaną ocenione wszystkie permutacje, uruchamiamy procedurę selekcji i usuwamy z każdej populacji 50 najgorzej ocenionych permutacji, a następnie stosujemy operatory różnicowania, żeby utworzyć po jednym nowym rozwiązaniu z każdego z pozostałych rodziców. Możemy tutaj użyć tego samego operatora inwersji co wcześniej. Możemy jednak też połączyć go z innym operatorem, który da procedurze koewolucji ze współpracą szansę nauczenia się, które miasta powinny być punktami połączenia między ćwiartkami, nie nakładając ograniczenia, żeby te połączenia były tworzone za pomocą podejścia zachłannego, jak powyżej.

W przedstawionym tu przykładzie każdy rodzic zawsze używa operatora inwersji, ale oprócz tego z prawdopodobieństwem 0,5 stosuje operator przesunięcia, który może przesunąć permutację o maksymalnie 13 kroków do przodu lub do tyłu (liczba ta to w zasadzie połowa 25 – liczby miast w permutacji). Żeby zapewnić bliskość między rodzicami a potomstwem, liczba kroków prze-

sunięcia w każdym kierunku jest wykorzystywana do liniowego stopniowego zmniejszania prawdopodobieństwa. To znaczy, że prawdopodobieństwo wybrania przesunięcia o 1 jednostkę wynosi $13/91$, o 2 jednostki $12/91$ itd., aż do prawdopodobieństwa przesunięcia o 13 jednostek, które wynosi $1/91$. Operator przesunięcia zmieni pierwsze i ostatnie miasto w permutacji, pozostawiając resztę ścieżki bez zmian. Pierwsze miasto w permutacji jest wyznaczane jako pierwsze miasto, na które się trafia, przechodząc z innej ćwiartki, ostatnie zaś miasto to miasto wyjścia z ćwiartki, łączące ją z następną ćwiartką. W ten sposób proces koewolucji ze współpracą może nauczyć się, których miast należy używać jako punktów wejścia i wyjścia, a nie polegać na zasadzie wykorzystywania miasta leżącego najbliżej sąsiedniej ćwiartki. Zauważ, że nie twierdzimy tutaj, że ten schemat jest optymalny. Korzystamy z niego tylko do przedstawienia potencjału tkwiącego w opisywanej metodzie.

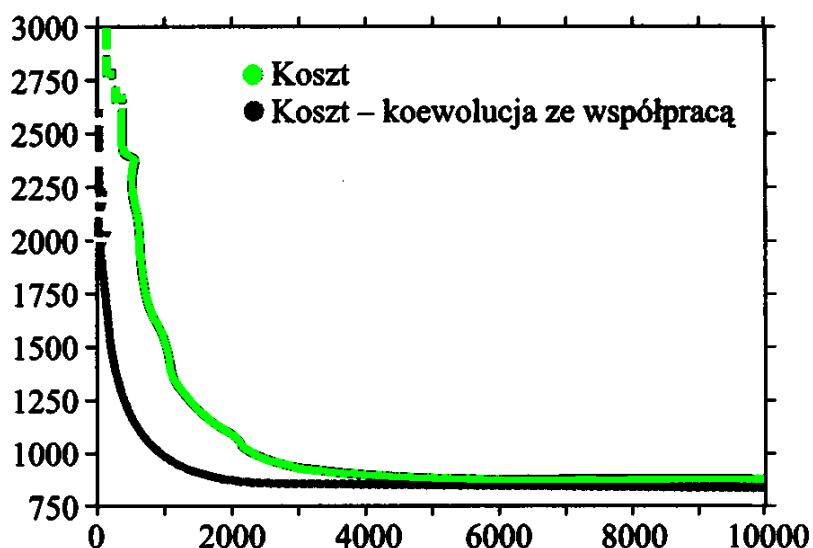
Po utworzeniu wszystkich permutacji potomnych w każdej z populacji, należy ponownie zastosować ocenę, żeby określić wartość każdego z rodziców i potomków. Zamiast jednak łączenia rozwiązań z populacji losowo, możemy tutaj dobierać rozwiązania, faworyzując te, które miały lepsze rezultaty w poprzednim pokoleniu. Przy pierwszym pokoleniu nie mamy takiej informacji, ale już przy drugim taka informacja już jest. Jedna z możliwych dróg jej wykorzystania polega na łączeniu w parę permutacji indeksowanej z permutacjami z pozostałych ćwiartek, które uzyskały najlepsze wyniki, i taki właśnie zabieg będziemy wykonywali w naszym przykładzie. Każda permutacja indeksowana otrzymuje ocenę na podstawie całkowitej długości trasy otrzymanej przez połączenie jej z najlepszymi reprezentantami pozostałych populacji. Po obliczeniu wszystkich ocen proces selekcji i różnicowania trwa do momentu przejścia przez 10 000 pokoleń.



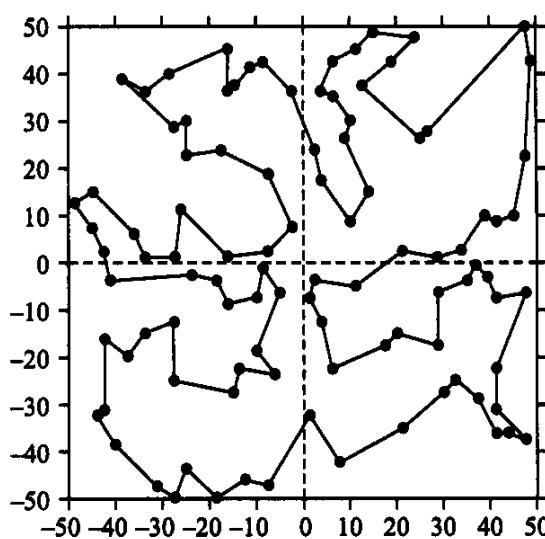
Rysunek 14.16. Szybkość optymalizacji długości najlepszej trasy w eksperymencie z koewolucją ze współpracą. Na wykresie przedstawiono koszt (długość trasy) jako funkcję numeru pokolenia

Na rysunku 14.16 przedstawiono szybkość optymalizacji długości najlepszej trasy w eksperymencie z koewolucją ze współpracą. Na rysunku 14.17 dla

ułatwienia porównania przedstawiono ten sam wykres wraz z wykresem dla standardowego podejścia ewolucyjnego. Zauważ, że przy koewolucji ze współpracą wynik szybciej ulega poprawie i wykorzystuje przewagę związaną ze sposobem łączenia sąsiednich ćwiartek. Na rysunku 14.18 przedstawiono najlepszą uzyskaną w wyniku ewolucji, trasę o długości 814,565, która jest około 2,3% krótsza niż trasa uzyskana w wyniku standardowej ewolucji. Zauważ, że metoda koewolucyjna nauczyła się przechodzić między ćwiartkami za pomocą miast, które nie są miastami najbliższymi sąsiednim ćwiartkom. Mogła wykorzystać początkowe podpowiedzi co do tego, jakich miast użyć, a następnie poprawić związane z nimi pomysły.



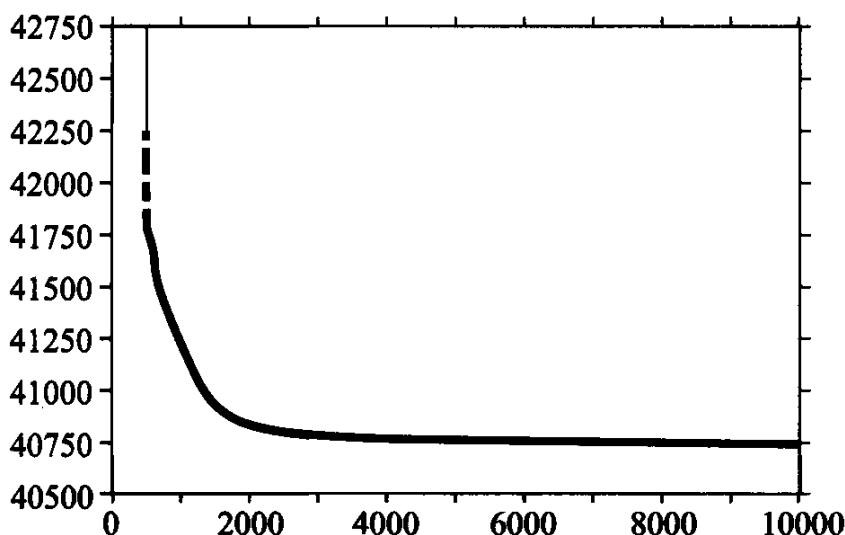
Rysunek 14.17. Porównanie szybkości optymalizacji dla koewolucji ze współpracą oraz dla standardowego podejścia. Na wykresie przedstawiono koszt (długość) trasy jako funkcję numeru pokolenia



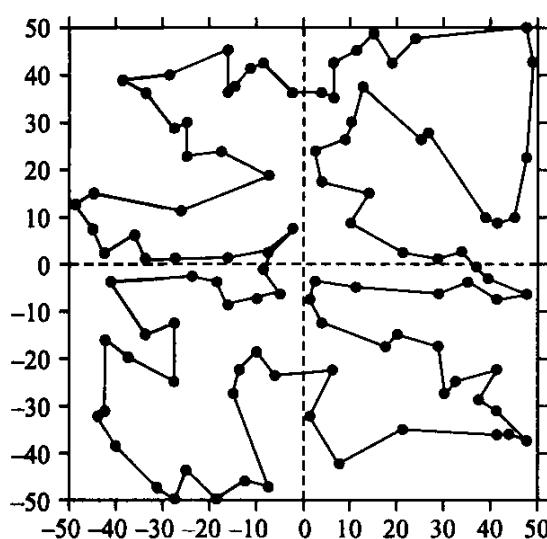
Rysunek 14.18. Najlepsza trasa uzyskana w wyniku koewolucji ze współpracą

Fakt, że koewolucja ze współpracą wygenerowała rozwiązanie lepsze niż rozwiązanie otrzymane standardową metodą, jest bardzo zachęcający. Wynik ten jest częściowo rezultatem ograniczenia możliwości przecinania granic ćwiart-

tek. W standardowej metodzie było możliwe swobodne przekraczanie tych granic, jeśli tylko taki był wynik ewolucji. Dla porównania uruchomiliśmy standar-dową metodę jeszcze raz, stosując karę 10 000 jednostek za każde przekroczenie granicy ćwiartek. Wymusiło to na algorytmie ewolucyjnym przekraczanie ich minimalną liczbę razy (cztery). Na rysunku 14.19 pokazano szybkość optymalizacji, przy czym wykres pokazuje tylko zakres długości tras, gdy ewolucja doszła do momentu, gdy rozwiązania miały tylko cztery przejścia między ćwiartkami. Na rysunku 14.20 przedstawiono najlepsze rozwiązanie po doprowadzeniu ewolucji do 10 000 pokoleń. Jego długość wynosi 834,035 jednostek, co jest prawie taką samą wartością jak trasa uzyskana pierwotnie za pomocą standardowej metody.



Rysunek 14.19. Szybkość optymalizacji przy wprowadzonym do standardowego podejścia wymuszeniu, żeby trasa przechodziła granice ćwiartek minimalną liczbę razy. Na wykresie przedstawiono koszt (długość trasy plus kara 10 000 za przekroczenie granicy) jako funkcję numeru pokolenia



Rysunek 14.20. Najlepsze rozwiązanie uzyskane w wyniku standardowej ewolucji z wymuszeniem, żeby trasa przekraczała granice ćwiartek minimalną liczbę razy

Chociaż rezultaty te uzyskano w wyniku pojedynczych prób użycia tych metod i w porównaniach nie brano pod uwagę wymagań obliczeniowych ani narzutu związanego z programowaniem, ilustrują one potencjał związany z zastosowaniem metody ewolucyjnej do optymalizacji fragmentów większego problemu przez łączenie rozwiązań podproblemów i ocenianie ich na podstawie tego, jak dobrze współpracują przy rozwiązywaniu pełnego problemu. Nie wszystkie metody koewolucyjne muszą mieć charakter konkurencyjny, żeby dawać pozytywne rezultaty.

14.10. Podsumowanie

Perspektywa prowadzenia koewolucji rozwiązań na zasadzie czynienia z nich przeciwników w jakieś grze lub na zasadzie poszukiwania rozwiązań, które dobrze współpracują przy rozwiązywaniu jakiegoś problemu, otwiera możliwość rozwiązywania problemów, dla których nie dysponujemy ludzką wiedzą. Tego rodzaju metody stoją u podstaw tworzenia maszyn, które będą zdolne do rozwiązywania problemów bez udziału człowieka. Metod tych użyto już do naużenia komputerów grania w gry strategiczne na poziomie porównywalnym z ludźmi-ekspertami. Co więcej, metody koewolucyjne możemy zwykle połączyć z innymi metodami sztucznej inteligencji lub z ludzką wiedzą albo wprowadzając heurystyki sterujące poszukiwaniem ewolucyjnym, albo dobierając początkowe populacje.

Koewolucję możemy też wykorzystywać do zmniejszenia ilości obliczeń potrzebnych do znalezienia rozwiązań. Przypuśćmy na przykład, że chcesz znaleźć sterownik dla niestabilnego systemu. Zamiast dostosowywać parametry jednego sterownika na podstawie, być może, ogólnego zachowania sterownika przez jakiś czas, możesz ustawić schemat koewolucyjny, w którym dwa sterowniki współzawodniczą w utrzymywaniu systemu w stanie stabilnym i pierwszy z nich, który zawiedzie, jest eliminowany. W tym momencie przestaje być konieczne testowanie zwycięzcy. Taki zabieg może oszczędzić istotną liczbę cykli obliczeniowych. I chociaż prawdą jest, że koewolucja może czasem podczas dochodzenia do rozwiązania w wyniku stosowanego w niej procesu „uczenia się przez śmierć” prowadzić do „marnowania” cykli procesora, można ją stosować w sytuacjach, gdy nie jest dostępna żadna inna tradycyjna metoda lub metoda ewolucyjna.

Do chwili obecnej koewolucja była jedną z bardziej obiecujących, a przy tym niewystarczająco stosowanych metod z zakresu obliczeń ewolucyjnych lub szerzej z zakresu uczenia się maszyn. Od najwcześniejszych dni istnienia komputerów ludzie opracowywali wizje maszyn, które mogły same nauczyć się, jak rozwiązywać problemy. Na smutną ironię zakrawa fakt, że większość wysiłków przy badaniach sztucznej inteligencji w ciągu ostatnich pięciu dekad miało niewiele wspólnego z uczącymi się maszynami, a raczej z realizowaniem za pomocą oprogramowania tego, co już znamy jako fakty, lub metody, wnioskowania informacji. Jeśli inteligencję pojmować jako zdolność adaptowania zachowania do osiągania celów w różnych środowiskach [148], to koewolucja zapewnia

bezpośrednie mechanizmy generowania intelligentnych maszyn. Wraz z rosnącą łatwością budowania efektywnych klastrów komputerów osobistych, które działają w sposób równoległy, czyli architektur, które znakomicie pasują do prowadzenia koewolucji na wielu populacjach, możemy oczekiwąć znaczącego postępu w uczeniu się maszyn i rozwiązywaniu problemów wynikających z zastosowania koewolucji. Przykłady zamieszczone w tym rozdziale pozwalają zrozumieć przyszłość.

XV. Kto jest wyższy?

Jeśli kiedyś byłeś w barze odwiedzanym przez sportowców, to prawdopodobnie słyszałeś tam ludzi porównujących zawodników z różnych czasów. Ktoś mógłby, na przykład, powiedzieć: „Babe Ruth był lepszym uderzającym w baseballu niż Barry Bonds”. Mógłbyś też usłyszeć, jak ktoś mówi: „lato 1969 było najlepszym latem w moim życiu” albo: „studenci na moich zajęciach w 2003 roku byli inteligentniejsi niż ci w 2002 roku”.

Zwykle intuicyjnie wiemy, o co chodzi mówiącemu, po bliższym przyjrzeniu się możemy jednak stwierdzić, że nasza intuicja słabnie pod wpływem pojawiających się wątpliwości co do tego, jak dokładnie powinniśmy interpretować tego rodzaju stwierdzenia. Przypuśćmy, na przykład, że Twój najlepszy przyjaciel powiedział, że zrobił lepszy interes, kupując Ferrari, niż gdyby kupił Porsche. O co mu chodzi? Czy to znaczy, że mniej za niego zapłacił? Być może. Czy to znaczy, że zaoferowano mu lepsze warunki finansowania? Być może. A co z lepszymi osiągami, przyspieszeniem czy zwrotnością? Może chodzi również i o te rzeczy.

Znajdowanie kompromisu w sytuacji istnienia wielu kryteriów to bardzo trudny element zagadnień związanych z podejmowaniem decyzji i rozwiązywaniem problemów. Zdanie to ilustruje poniższa historyjka (zaadaptowana na podstawie [455]).

Dwie grupy studentów uczęszczają do małego uniwersytetu należącego do Ivy League. Studenci z pierwszej grupy, oznaczanej przez nas tutaj jako grupa A, przechwalają się, że są wyżsi niż studenci z drugiej grupy, oznaczanej przez nas jako grupa B. Jednocześnie studenci z grupy B cieszą się opinią lepszych logików niż studenci z grupy A.

Któregoś popołudnia jeden ze studentów z grupy A, przechodząc w centrum kampusu obok kilku ławek, zauważył kogoś z grupy B i podszedł do niego, mówiąc: „Jesteśmy od was wyżsi!”. Student z grupy B, który stwierdził, że

musi wyciągnąć szyję, patrząc w niebo, żeby zobaczyć swojego przyjaciela z grupy A, odpowiedział: „Co masz na myśli, mówiąc, że jesteście wyżsi niż my? Usiądź, proszę”. Zaskoczony propozycją swego drobnego kolegi student z grupy A usiadł.

„Popatrz, założmy, że studenta z grupy A oznaczymy symbolem a , zaś studenta z grupy B symbolem b . Nadążasz?” – zapytał niski student, siedząc obok swego wyższego kolegi, który przytaknął. „W porządku, zatem mówisz: ‘Jesteśmy wyżsi od was!’ Ale co to, duży człowieku, znaczy? Czy znaczy to, że...”. Niski student wciągnął powietrze i poprawił swoje druciane okulary. „.... że:

1. Każdy a jest wyższy od każdego b ?
2. Najwyższy a jest wyższy od najwyższego b ?
3. Każdy a jest wyższy od pewnego b ?
4. Każdy b jest niższy od pewnego a ?
5. Każdy a ma odpowiadającego sobie b (każdy z nich innego), od którego jest wyższy?
6. Każdy b ma odpowiadającego sobie a (każdy z nich innego), który jest od niego wyższy?
7. Najniższy b jest niższy od najniższego a ?
8. Najniższy a jest wyższy od większej liczby członków grupy B niż najwyższy b jest wyższy od członków grupy A?
9. Suma wzrostu wszystkich a jest większa niż suma wzrostu wszystkich b ?
10. Średni wzrost wszystkich a jest większy od średniego wzrostu wszystkich b ?
11. Jest więcej a , którzy są wyżsi od jakiegoś b , niż b , którzy są wyżsi od jakiegoś a ?
12. Jest więcej a o wzroście większym niż średni wzrost wszystkich b niż b o wzroście większym od średniego wzrostu wszystkich a ?
13. Czy też mediana wzrostu wszystkich a jest większa niż mediana wzrostu wszystkich b ?

„Hm...” – zadumał się wysoki student z grupy A. „Nie jestem pewien. Muszę to przemyśleć.”

„Zrób to. Mam teraz zajęcia z logiki, ale wrócę tu za godzinę. Powiem ci coś. Wiesz, że my, studenci z grupy B, myślimy, że jesteśmy lepszymi logikami niż wy w grupie A. Ale, jak będziesz umiał mi powiedzieć, które z moich 13 pytań są niezależne od pozostałych, a które są zależne, to może zmienię opinię”. Zostawiając w ten sposób swego kolegę, student z grupy B wziął swoją książkę do logiki i poszedł na wykład.

Umiesz pomóc studentowi z grupy A? Czy możesz znaleźć wszystkie pary takie, że odpowiedź „tak” na pierwsze pytanie implikuje odpowiedź „tak” na

drugie? Czy są tutaj jakiekolwiek pytania, które są równoważne, czyli takie, gdzie odpowiedź na obydwa musi być taka sama? Czy są jakieś pary, które są zależne, ale nie są równoważne?

W tym momencie, żeby nie zdradzać od razu odpowiedzi, zajmiemy się wielkim twierdzeniem Fermata, które wspominamy na ostatniej stronie ostatniego rozdziału (rozdziału 17). Twierdzenie to mówi, że dla liczb naturalnych x , y , z i n , jeśli $n \geq 3$, równanie $x^n + y^n = z^n$ nie zachodzi. Mamy tutaj małą rzeczkę do przemyślenia. Przypuśćmy, że przyjmiemy założenie, iż $n \geq z$. W pierwszej chwili wydaje się ono niewinne, gdyż interesują nas głównie sytuacje, gdy n jest duże. Zabawne jest jednak to, że jak się zrobi takie założenie, to dowód twierdzenia Fermata jest bardzo łatwy.

Przypuśćmy, że istnieją liczby naturalne x , y , z oraz n takie, że $n \geq z$ oraz $x^n + y^n = z^n$. Jasne jest, że $x < z$, $y < z$ i $x \neq y$. Dodatkowo, ze względu na symetrię, możemy założyć, że $x < y$. Mamy zatem:

$$z^n - y^n = (z - y)(z^{n-1} + yz^{n-2} + \dots + y^{n-1}) \geq 1 \cdot nx^{n-1} > x^n$$

co jest sprzeczne z założeniem! Jak widać, małe, niewinne założenie zmienia istotnie problem.

Przy okazji, gdybyś znalazł krótki, elegancki dowód tego twierdzenia, gdy $n < z$, to natychmiast daj nam znać!

Co się jednak dzieje z naszym przyjacielem, którego zostawiliśmy na ławce w kampusie? „Widzę, że wciąż tu jesteś” – powiedział niski student z grupy B , wracając ze swoich zajęć z logiki. „Masz odpowiedź?” Duży tylko pokwał głową. „Być może, jednak my w grupie B jesteśmy lepszymi logikami” – powiedział niższy student, sięgając do kieszeni i wręczając swemu koleżance kawałek papieru. „W porządku, oto odpowiedzi. Możesz je sam sprawdzić. Upewnij się, że nie popełniłem żadnego błędu!” Na papierze było napisane. „Oznaczmy punkty zadania liczbami, które przy nich stoją w wyliczeniu, i niech wyrażenie $p \rightarrow q$ oznacza, że odpowiedź *tak* na pytanie p implikuje odpowiedź *tak* na pytanie q .

Twoja praca domowa polega na sprawdzeniu, że:

$$1 \rightarrow 2, 1 \rightarrow 3, 1 \rightarrow 4, 1 \rightarrow 7, 1 \rightarrow 8, 1 \rightarrow 10, 1 \rightarrow 11, 1 \rightarrow 12, 1 \rightarrow 13$$

$$2 \rightarrow 4$$

$$3 \rightarrow 7$$

$$4 \rightarrow 2$$

$$5 \rightarrow 7$$

$$6 \rightarrow 2, 6 \rightarrow 4, 6 \rightarrow 9$$

$$7 \rightarrow 3$$

$$8 \rightarrow 3, 8 \rightarrow 7$$

Wynika stąd, że odpowiedzi na pytania 2 i 4 są takie same oraz że identycznie sprawia się ma dla 3 i 7”.

Zatem, czy niski chłopak popełnił jakiś błąd?

15. Wielokryterialne podejmowanie decyzji

Nie ma nic z pożądliwości w cieszeniu się dobrym obiadem, nie w większym stopniu występuje ona przy delektowaniu się koncertem. Myślę jednak, że jest coś pożądliwego w jednoczesnym smakowaniu i obiadu, i koncertu.

Gilbert K. Chesterton: *Generally speaking*

Problemy ze świata rzeczywistego prowadzą zwykle do rozważań w wielu kierunkach. Bardzo rzadko zdarza się, że mamy do czynienia tylko z jednym parametrem. Pomyśl na przykład o kupowaniu samochodu. Być może, jesteś osobą, która po prostu musi mieć Porsche 911 Carrera w „sportowym żółtym kolorze”, a cena nie gra roli. Jeśli jesteś tego rodzaju osobą, to Twój problem wyboru samochodu jest w zasadzie dwuwartościowym problemem decyzyjnym: albo masz samochód, albo go nie masz! Być może, mógłbyś się nieco poświęcić i, powiedzmy, choć lubisz żółty kolor, zadowoli Cię czerwony lub czarny, ale zdecydowanie nie chciałbyś niebieskiego ani białego. W tym momencie optymalizacja odbywa się w jednym wymiarze, którym jest kolor. Jeśli jednak jesteś taki, jak większość ludzi, to perspektywa zakupu samochodu powoduje konieczność przemyślenia wielu rzeczy.

Większość ludzi (ale nie wszyscy!) przy zakupie samochodu zaczyna od zastanawiania się nad ceną. Czy możesz sobie na niego pozwolić? Oznacza to, że cena jest istotnym czynnikiem przy zakupie. Następnie ważny jest wygląd. Czy samochód Ci się podoba? Dalej jest rozważana funkcjonalność. Czy będzie mógł przewieźć wszystko, czego potrzebujesz? Dzieci? Deskę surfingową? Są jeszcze inne kwestie do rozważenia, takie jak samochodowy sprzęt audio, dane o bezpieczeństwie w razie wypadku, jakość przyspieszenia, prestiż związany z posiadaniem itd. Płacąc swoimi pieniędzmi za samochód, kupujesz wszystkie te parametry.

Kupowanie samochodu jest proste w porównaniu z prowadzeniem dużej firmy. Jeśli jesteś udziałowcem takiej firmy, to chcesz, aby jej cena giełdowa rosła, byś mógł zdobyć więcej cennych aktywów. Jeśli dbasz o środowisko, możesz chcieć wiedzieć, czy firma podchodzi do procesów produkcyjnych ostrożnie i zapewnia procesy recyklingu jej produktów, i unika zatruwania powietrza oraz wody. Jeśli zaś kierujesz firmą, to musisz brać pod uwagę o wiele więcej czynników [217].

Oczywiście maksymalizacja zysku jest dla Ciebie istotna, ale w jakim czasie? Krótki okres oznacza tutaj zwykle najbliższe trzy miesiące – do ukażania się następnego raportu kwartalnego przeznaczonego dla udziałowców. W średniej perspektywie czasowej to zapewne rok, gdyż udziałowcy i analitycy korporacyjni zwykle dokonują porównań w cyklach rocznych. Długi okres to prawdopodobnie okres od dwóch do pięciu lat. Szczerze mówiąc, większość ludzi nie patrzy tak daleko w przyszłość przy prowadzeniu firmy. Zwróć jednak uwagę, że nie wystarczy tutaj powiedzieć, że chcesz zmaksymalizować zyski. Musisz powiedzieć, kiedy chcesz te zyski osiągnąć, gdyż czynności, które wykonyujesz, żeby zmaksymalizować krótkoterminowe zyski, mogą mieć niepożądany wpływ na zyski długoterminowe i na odwrót. Musisz wziąć też pod uwagę stabilność tych przychodów. Czy przeszkadza Ci to, że w czasie są one raz większe, raz mniejsze, czy też fluktuacje są dopuszczalne, dopóki tendencja jest rosnąca.

Przychody to miła rzecz, ale nie wszystko przekłada się wprost na wynik końcowy. A co z udziałem w rynku? Możesz chcieć uzyskać pozycję jak Microsoft, który dominuje na rynku aplikacji biurowych. Żeby zwiększyć udział w rynku, możesz musieć poświęcić zyski. A co z dywersyfikacją? Chcesz być znany z jednego produktu, czy z wielu? Bycie znanim jako producent wielu produktów daje bezpieczeństwo, ale wprowadzenie na rynek wielu produktów wymaga czasu i pieniędzy. Co jeśli chodzi o morale robotników? Prawdopodobnie zależy Ci na szczęśliwych robotnikach. Co należy zrobić, żeby to zapewnić? Niektórzy ludzie są szczęśliwi, gdy pracują nad interesującymi problemami, będącymi prawdziwymi wyzwaniami. Dla zatrudniającego to doskonała sytuacja. Inni zadowalają się, gdy mają opiekę zdrowotną, wakacje, świadczenia chorobowe, urlop macierzyński, refundację edukacji, ubezpieczenie na życie, nowy sprzęt, samochód w lizingu, członkostwo w klubach sportowych oraz przestrzenne biuro z pięknym widokiem. Dla zatrudniającego sytuacja taka nie jest tu tak komfortowa.

A co jeśli chodzi o nadzór nad firmą? Jeśli jesteś jedynym właścicielem, to jesteś dyktatorem. To, co powiesz, jest wykonywane, dopóki nie jest sprzeczne z prawem. Jeśli własność dzielisz z innymi, to, dopóki masz więcej niż 50% udziałów, w dalszym ciągu to, co powiesz, jest wykonywane. Jeśli jednak masz ich mniej niż połowę, to Twoje słowo może nie mieć w ogóle znaczenia! Być może, będziesz chciał poświęcić część zysków, żeby zachować kontrolę nad swoim biznesem, szczególnie jeśli jest to biznes rodzinny, który był przekazywany z pokolenia na pokolenie.

Efektywne rozwiązywanie problemów wymaga rozważania kompromisów nierozerwalnie związanych z przydzielaniem zasobów służących do osiągnięcia celów. Dążenie do wielu celów może wymagać metod, które odbiegają od standardowych metod optymalizacji z jednym celem. Jak wspomnialiśmy powyżej, jeśli mamy choćby dwa cele do optymalizacji, to może być możliwe istnienie jednego rozwiązania, które jest najlepsze ze względu na pierwszy z nich, ale nie ze względu na drugi, oraz innego rozwiązania, które jest najlepsze ze względu na drugi, ale nie ze względu na pierwszy. Problemy z wieloma celami stawiają

zadanie zdefiniowania jakości rozwiązania w kategoriach wielu parametrów, być może będących w konflikcie¹.

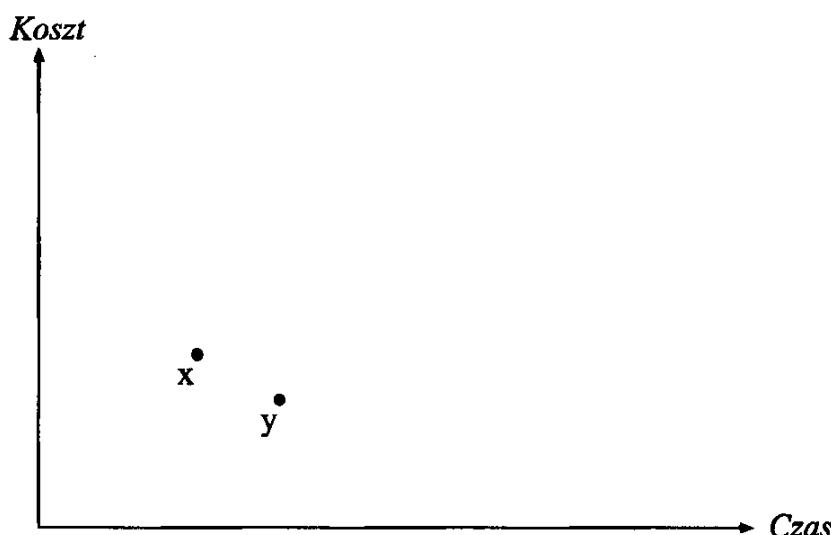
Za każdym razem, gdy stoisz wobec wielu celów będących w konflikcie, masz do czynienia z tak zwanym wielokryterialnym podejmowaniem decyzji. W takich wypadkach możesz poszukiwać sposobu na połączenie w całość tego, co wnosi każdy cel, i tak stworzyć łączny, wspólny cel. Następnie możesz znaleźć optymalne rozwiązanie dla tego ogólnego celu. Możesz też poszukać optymalnych rozwiązań w innym sensie, w którym cele są traktowane w pewien sposób niezależnie, a rozwiązanie jest optymalne, jeśli nie ma lepszego rozwiązania, które jest lepsze ze względu na wszystkie parametry. Przyjrzyjmy się temu pomysłowi bliżej i omówmy prosty przykład.

Przypuśćmy, że ktoś postawił Cię w roli osoby odpowiedzialnej za amerykańską firmę produkującą zabawki. Chcesz jednocześnie zminimalizować koszt produkcji zabawek oraz czas ich wykonywania. Te dwa cele są w konflikcie. Mógłbyś obniżyć koszt, przenosząc dużą część swojej produkcji do jakiegoś kraju w Azji Południowo-Wschodniej, gdzie cena robocizny jest niska. Taki ruch zwiększa jednak czas produkcji, gdyż dochodzą do niego dodatkowe tygodnie przewożenia towarów w jedną i drugą stronę przez ocean. Zamiast tego mógłbyś składać wszystko w Stanach i oszczędzać w ten sposób czas, ale wtedy musiałbyś płacić więcej za pracę oraz pokrywać dodatkowe koszty związane z podatkiem od wynagrodzeń, ubezpieczeniem robotników itd. Kwestię tę przedstawiono na rys. 15.1. Mamy dwa różne rozwiązania **x** i **y** takie, że pierwsze jest lepsze, jeśli chodzi o *czas*, drugie zaś jest lepsze, jeśli chodzi o *koszt*. Żadne z nich nie jest lepsze od drugiego, jeśli chodzi o oba cele.

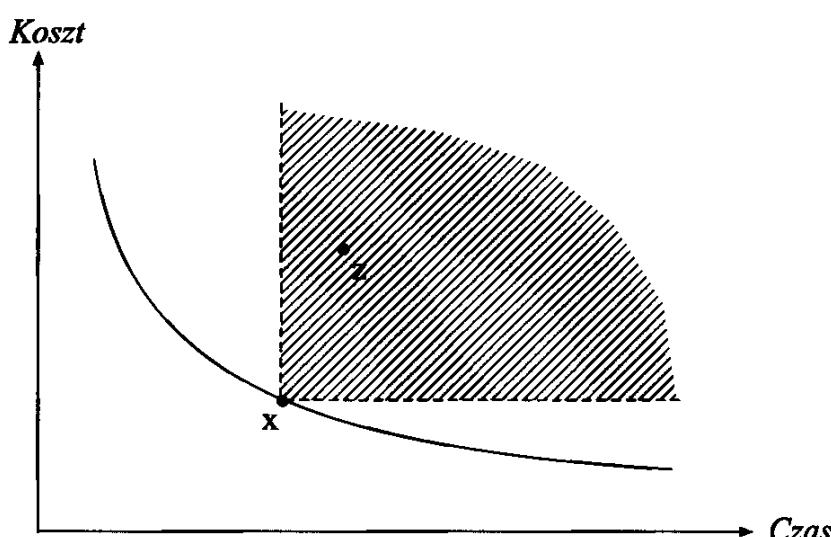
Zastanówmy się, korzystając z tego przykładu, nad pojęciem optymalności. Widzimy, że rozwiązania **x** i **y** nie są bezpośrednio porównywalne. Bez stworzenia jakiegoś sposobu tłumaczenia *kosztu* na *czas* niemożliwe jest stwierdzenie, które z tych rozwiązań jest lepsze. Może jednak któreś z rozwiązań **x** i **y** jest w jakimś sensie „optymalne”?

Intuicyjnie jest jasne, że konkretne rozwiązanie **x** „dominuje” na pewnym obszarze przestrzeni poszukiwań, to znaczy, dowolne rozwiązanie **z** z tego obszaru jest gorsze od **x**, jeśli chodzi o oba cele (rys. 15.2). Oczywiście takie **z** nie może być „optymalne” w żadnym rozsądny sensie. Z drugiej strony, jeśli dla rozwiązania **x** stwierdzimy, że nie ma innego rozwiązania, które jest lepsze pod

¹ Jeden z autorów (Z.M.) miał kilka lat temu wypadek podczas jazdy swoim kabrioletem Porsche 911. Wypadek ten był pod wieloma względami „optymalny”: 1) optymalny był przedmiot, w który nastąpiło uderzenie – było to drzewo, dzięki temu nikt nie odniósł obrażeń, mimo iż na drodze były inne samochody, a kilka metrów za drzewem stała ściana domu; drzewo to stało naprawdę w dobrym miejscu; 2) prędkość przy uderzeniu była optymalna – firma ubezpieczeniowa przez cztery tygodnie podejmowała decyzję o kasacji samochodu, gdyż decyzja ta była na granicy – gdyby prędkość samochodu była odrobinę mniejsza, samochód zostałby naprawiony, a kto by chciał jeździć wyklepanym 911, skoro można mieć nowy? Gdyby zaś prędkość była trochę większa, cała sprawa mogłaby się skończyć mnóstwem pomyślnie dla kierowcy. I wreszcie 3) moment, gdy wypadek miał miejsce, był optymalny – w Karolinie Północnej był właśnie listopad, a kto chciałby jeździć kabrioletem w zimie? Oczywiście kryteria tej optymalizacji nie stały ze sobą w konflikcie.



Rysunek 15.1. Dwa „optymalne” rozwiązania: x i y



Rysunek 15.2. Na podstawie rozwiązania x jest wyznaczany (zacieniony) obszar zdominowany. Każde rozwiązanie w tym obszarze, na przykład z , jest rozwiązaniem zdominowanym

względem wszystkich celów naszego zadania, to rozwiązanie x jest w pewnym sensie „optymalne”.

Powyższe rozwiązania możemy sformalizować. Problem optymalizacji wielokryterialnej definiujemy następująco:

$$\begin{aligned} & \text{zminimalizuj } f_i(\mathbf{x}), \quad i = 1, \dots, m \\ & \text{przy zakresach } l(p) \leq x_p \leq u(p), \quad p = 1, \dots, n \\ & \quad g_k(\mathbf{x}) \leq 0, \quad k = 1, \dots, l \\ & \quad h_j(\mathbf{x}) = 0, \quad j = 1, \dots, q \end{aligned}$$

Zakładamy tutaj, że mamy do czynienia z problemem minimalizacji. Mamy te wzory przystosować do sytuacji z maksymalizacją, mnożąc funkcje celu przez -1 . Wektor rozwiązań \mathbf{x} ma n zmiennych decyzyjnych, przy czym mamy do czynienia z m funkcjami celu f_1 do f_m . Zatem przestrzeń decyzyjna jest n -wymiarowa, a przestrzeń celów – m -wymiarowa. W dalszej części tego

rozdziału warto pamiętać o tym rozróżnieniu. Wynika to stąd, że mogą istnieć interesujące (nieliniowe) odwzorowania między tymi przestrzeniami. Jeśli, na przykład, będziemy rozważali pewne aspekty metody optymalizacji wielokryterialnej (np. różnorodność), to ważne będzie, żeby umieć rozróżnić, czy mówimy o różnorodności punktów w przestrzeni decyzyjnej, czy w przestrzeni celów.

Załóżmy teraz, że mamy zestaw potencjalnych rozwiązań problemu optymalizacji wielokryterialnej. Jak wynika z naszych rozważań, może być wygodne podzielenie rozwiązań na *zdominowane* i *niezdominowane*. Rozwiązanie \mathbf{z} jest zdominowane, jeśli istnieje dopuszczalne rozwiązanie \mathbf{x} , które jest: 1) co najmniej tak dobre jak \mathbf{z} ze względu na wszystkie wymiary, tzn. dla każdego celu f_i ($i = 1, \dots, m$)

$$f_i(\mathbf{x}) \leq f_i(\mathbf{z}) \text{ dla wszystkich } 1 \leq i \leq m$$

oraz 2) ścisłe lepsze od \mathbf{z} co najmniej ze względu na jeden cel i , tzn.

$$f_i(\mathbf{x}) < f_i(\mathbf{z}) \text{ dla pewnego } 1 \leq i \leq m$$

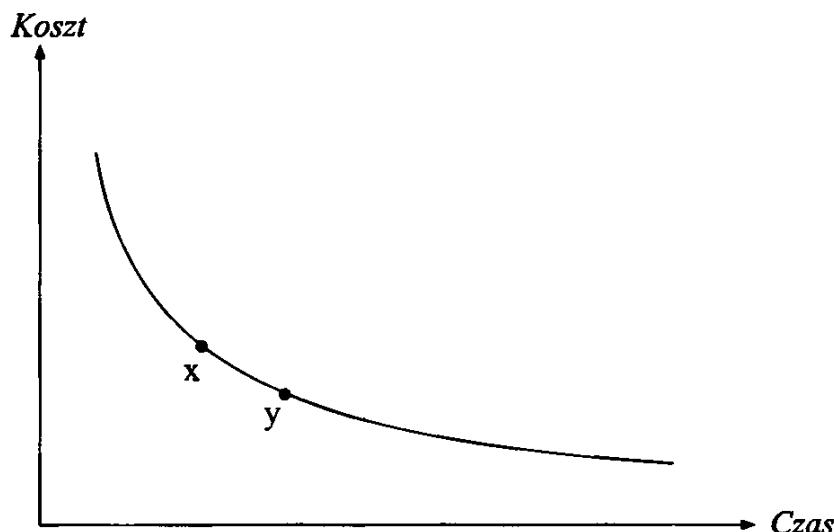
Każde rozwiązanie, które nie jest zdominowane przez żadne inne rozwiązanie dopuszczalne, nazywamy rozwiązaniem *niezdominowanym*. Przypomnijmy, że na rys. 15.2 pokazano sytuację, w której \mathbf{z} jest zdominowane.

Możemy teraz omówić kwestię optymalności. Zbiór niezdominowanych rozwiązań z całej dopuszczalnej przestrzeni poszukiwań nazywamy *zbiorem optymalnym w sensie Pareto*. Nazwa ta pochodzi od włoskiego ekonomisty Vilfreda Pareto (1848–1923). W związku z tym, że rozwiązania z tego zbioru nie są zdominowane przez żadne inne rozwiązanie dopuszczalne, są one w tym sensie optymalne dla problemu optymalizacji wielokryterialnej¹. Zbiór optymalny w sensie Pareto musi być zbiorem niezdominowanym, odwrotne stwierdzenie nie musi być jednak prawdą. Zbiór niezdominowany może zawierać pewne optymalne w sensie Pareto rozwiązania i pewne rozwiązania w tym sensie nieoptymalne. Zadanie dowolnego algorytmu znajdującego zbiór optymalny w sensie Pareto polega na daniu w wyniku zbioru niezdominowanych rozwiązań (istnieją efektywne procedury określające brak zdominowania dla dowolnego zbioru rozwiązań [98]), który przybliża zbiór optymalny w sensie Pareto tak, jak to tylko jest możliwe. Wszystkie rozwiązania optymalne w sensie Pareto mogą być interesujące, a w idealnej sytuacji system powinien podawać zbiór wszystkich optymalnych w sensie Pareto punktów.

Rozwinięciem naszego wcześniejszego przykładu jest rys. 15.3 przedstawiający kompromis między *kosztem* a *czasem*. Jak poprzednio szukamy sposobu na jednoczesną minimalizację dwóch celów: *kosztu* (f_1) i *czasu* (f_2). Ciągła linia na rysunku reprezentuje granicę (front) wartości optymalnych w sensie Pareto,

¹Przy dogłębnej analizie optymalizacji wielokryterialnej warto jeszcze zdefiniować lokalny zbiór optymalny w sensie Pareto, w którym dla każdego \mathbf{x} z tego zbioru nie ma zdominowanego rozwiązania w otoczeniu \mathbf{x} (a nie w całej przestrzeni poszukiwań), oraz pojęcie *silnej* dominacji, przy której rozwiązanie jest ścisłe lepsze ze względu na wszystkie cele. Analiza taka nie jest jednak tematem tego rozdziału.

która ukazuje kompromis między tymi dwoma celami. Dla każdego rozwiązania na tej granicy poprawa kosztu spowoduje zwiększenie czasu i na odwrót. Rozwiązania x i y na tej linii reprezentują dwa z wielu niezdominowanych (optymalnych w sensie Pareto) rozwiązań: x lepiej optymalizuje czas, y zaś efektywniej minimalizuje koszt.



Rysunek 15.3. Kompromis między kosztem a czasem

Teraz należy podjąć decyzję. Wiemy, że istnieje zbiór niezdominowanych rozwiązań (pełny zbiór optymalny w sensie Pareto). Każde z rozwiązań w tym zbiorze jest potencjalnie interesujące. Problem polega jednak na tym, że w większości wypadków możemy zrealizować tylko jedno rozwiązanie. Wróćmy myślami do przykładu z firmą produkującą zabawki. Możesz mieć do dyspozycji cztery potencjalne rozwiązania, które są optymalne w sensie Pareto: 1) wysłać 100% materiałów za ocean, 2) wysłać 77% materiałów za ocean, 3) wysłać 22% materiałów za ocean i 4) zostawić wszystko w Stanach Zjednoczonych. Oczywiście nie można wysłać wszystkiego za wielką wodę i równocześnie zatrzymać w Stanach. W istocie możesz wybrać tylko jedno z tych rozwiązań. Możesz powiedzieć, że każde z tych rozwiązań jest w jakimś sensie optymalne, ale musisz dysponować jakimś sposobem na skoncentrowanie się na jednym z nich.

Znalezienie tego sposobu wymaga spojrzenia poza ujęcie problemu w kategoriach zdominowanych i niezdominowanych rozwiązań w celu dołączenia dodatkowych informacji wysokiego poziomu. Na przykład do każdego z celów możemy przypisać istotność względną każdego z celów, być może, przypisując im jakieś liczby lub rozmyte opisy lingwistyczne. Jeden z celów może być „bardzo ważny”, a inny „nie bardzo ważny” lub też jeden można ocenić na 9 w 10-cio stopniowej skali, inny zaś na 2, przy czym wyższa ocena oznacza większą istotność. Alternatywnie moglibyśmy ułożyć cele w hierarchię ważności (istotności), a następnie wybierać rozwiązania, korzystając z tego uporządkowania. Moglibyśmy też wybrać najważniejszy cel, a następnie przekształcić pozostałych $m - 1$ celów w ograniczenia, wykorzystując je jako progi, które muszą pozostać przekroczone. W każdym z tych przypadków informacja wyższego poziomu jest wykorzystywana do przekształcenia problemu wielokryterialnego w pro-

blem z jednym celem, który polega na znalezieniu minimum (lub maksimum) funkcji oceny. W podrozdziale 15.1 rozważamy metody radzenia sobie z problemami wielokryterialnego podejmowania decyzji za pomocą określania istotności względnej i zależności między celami przez łączenie wyników w jedną wspólną funkcję oceny. Inne podejście polega na odsunięciu momentu, w którym jest wykorzystywana informacja wyższego poziomu, do momentu po określeniu zbioru optymalnego w sensie Pareto. W metodzie tej należy najpierw znaleźć zbiór optymalny w sensie Pareto; omawiamy ją w podrozdziale 15.2.

15.1. Redukowanie do problemów z jedną liczbą

Istnieją klasyczne sposoby przekształcania problemów optymalizacji wielokryterialnej w zadania z jednym celem. Bez wątpienia, najbardziej znanym z nich jest metoda prostej sumy ważonej, w której dla każdego celu jest określona funkcja oceny f_i , a potem każdej z nich jest przypisywana wartość liczbową. Następnie funkcje te są łączone we wspólną funkcję oceny F :

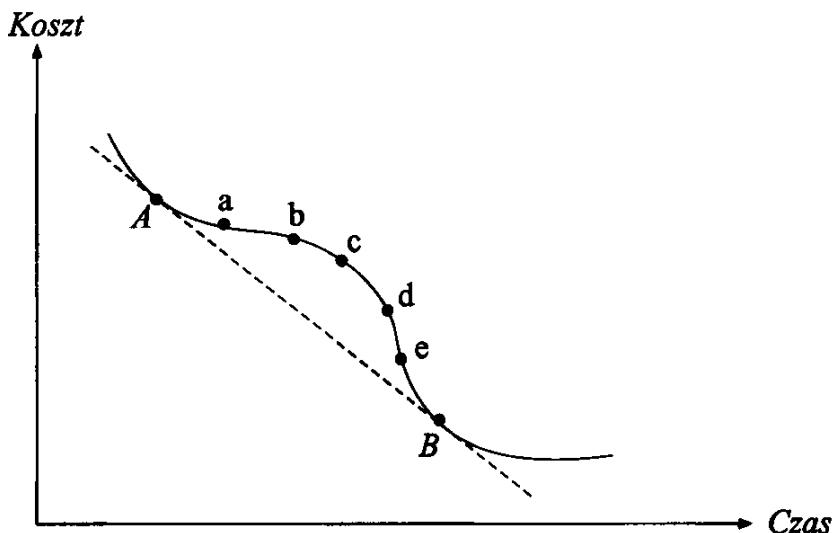
$$F(\mathbf{x}) = \sum_{i=1}^m w_i f_i(\mathbf{x})$$

Zwykle wagi $w_i \in [0..1]$ i $\sum_{i=1}^m w_i = 1$. W takim liniowym przykładzie dowolne wartości wag (zakładamy, że są dodatnie) mogą zostać znormalizowane bez zmieniania rozwiązania \mathbf{x} , które minimalizuje lub maksymalizuje $F(\mathbf{x})$.

Skąd jednak pochodzą te wagi? To pytanie jest zasadnicze! Odpowiedź jest taka, że wagi są określonym przez rozwiązującego zadanie subiektywnym odzwierciedleniem istotności względnej celów. Jeśli zapytałbyś się różnych ludzi, jakie wartości wag powinny zostać użyte, to otrzymałbyś różne odpowiedzi. I to właśnie tak powinno wyglądać, gdyż różne rzeczy mają różne znaczenie dla różnych ludzi. Nie wolno jednak tracić z oczu użytych wartości liczbowych. Niektóre z celów mogą być mierzone w dolarach, inne w sekundach, a jeszcze inne w metrach lub innych bezpośrednio nieporównywalnych jednostkach. Później poznamy metodę dostarczającą strukturę potrzebną przy zajmowaniu się takimi nieporównywalnymi rzeczami, a na razie wystarczy nam wiedza, że zwykle trzeba poprawić funkcje oceny f_i , żeby pasowały do siebie przy zastosowanych wagach.

Atrakcyjność metody agregacji wielu celów, zwanej metodą sumy liniowej (znanej też jako metoda średniej arytmetycznej), wynika głównie z jej prostoty. Atrakcyjność ta jest także wynikiem tego, że wkład każdej funkcji f_i jest liniowy. Zwiększenie wartości jednej funkcji f_i prowadzi do proporcjonalnego zwiększenia ogólnej oceny. Co więcej, łącząc te rozważania z optymalnością w sensie Pareto, optymalne rozwiązanie dopuszczalne dla F jest również optymalne w sensie Pareto, jeśli wszystkie wagi są dodatnie. I dalej, jeśli \mathbf{x} jest rozwiązaniem optymalnym w sensie Pareto dla *wypukłego* problemu optymalizacji wielokryterialnej, to istnieje niezerowy dodatni wektor wag \mathbf{w} taki, że \mathbf{x} jest rozwiązaniem F [321].

Jeśli jesteś jednak zainteresowany rozwiązaniami optymalnymi w sensie Pareto, to musisz się pogodzić z pewnymi niedogodnościami z nimi związanymi. Dobranie wag tak, żeby uzyskać rozwiązanie optymalne w sensie Pareto w pożądanym obszarze przestrzeni celów, może być trudne. Dla konkretnego wektora wag może istnieć kilka rozwiązań minimalnych. Każde z nich może reprezentować inne rozwiązanie minimalne w sensie Pareto, a niektóre z nich mogą być zdominowane przez inne (co oznacza marnowanie wysiłków poszukiwawczych). Także dla przestrzeni celów, które nie są wypukłe (rys. 15.4), mogą być odcinki punktów optymalnych w sensie Pareto, które nie mogą być wygenerowane jako liniowe sumy ważone.



Rysunek 15.4. Niektóre z rozwiązań optymalnych w sensie Pareto (**a, b, c, d i e**) są „schowane” w niewypukłej przestrzeni celów

Inna metoda – metoda funkcji odległości – łączy wiele funkcji oceny w jedną na podstawie wektora poziomu popytu \mathbf{y} :

$$F(\mathbf{x}) = \left(\sum_{i=1}^m |f_i(\mathbf{x}) - y_i| \right)^{\frac{1}{r}}$$

przy czym (zwykle) $r = 2$ (metryka euklidesowa). Czasami w metodzie tej dodajemy także wagę:

$$F(\mathbf{x}) = \left(\sum_{i=1}^m w_i |f_i(\mathbf{x}) - y_i| \right)^{\frac{1}{r}}$$

i wtedy metodę tę nazywamy metodą metryczną z wagami lub podejściem z wektorem celu [170]. Gdy zastosowana wartość r jest duża, problem przekształca się w problem minimalizacji największego odchylenia $|f_i(\mathbf{x}) - y_i|$ i jest nazywany problemem Czebyszewa z wagami. (Minimalizacja maksimum spośród n celów jest też czasami określana jako podejście „minimaksowe” [170]). W tym ujęciu rozwiązanie optymalne jest rozwiązaniem, które minimalizuje odległość (lub funkcję zastępczą) między $F(\mathbf{x})$ a \mathbf{y} .

Jeszcze jedno podejście – zwane metodą ϵ -ograniczeń – jest oparte na pomyśle zachowania tylko jednego z celów (być może tego najważniejszego), powiedzmy f_r , i na przekształceniu reszty celów w ograniczenia

$$f_i(\mathbf{x}) \leq \epsilon_i \text{ dla } 1 \leq i \leq m \text{ oraz } i \neq r$$

Parametry ϵ_i reprezentują górne ograniczenia wartości f_i ; intuicyjnie reprezentują one wymagania wobec każdego z celów. Główna zaleta tego podejścia polega na tym, że można go zastosować do problemów zarówno z wypukłymi, jak i wklęsłymi przestrzeniami. Co więcej, rozwiązywanie w metodzie ϵ -ograniczeń jest optymalne w sensie Pareto dla dowolnego danego wektora górnych ograniczeń. Główna wada polega na złożoności określania wartości liczb ϵ_i ¹.

Przy rozwiązywaniu wielokryterialnych problemów optymalizacyjnych możemy użyć metody programowania celu [393]. Podobnie jak w niektórych wcześniejszej omówionych metodach, szukamy tutaj rozwiązań, które mierzą w z góry określone cele (mogą one być rozmaitego rodzaju: nierówności, równości lub zakresy) dla pewnych funkcji oceny. Jeśli żadne rozwiązanie nie osiąga celu dla wszystkich funkcji oceny, to problem przechodzi w problem znajdowania rozwiązań, które minimalizują odchylenie od celów²:

$$\text{zminimalizuj } \sum_{i=1}^m \alpha_i p_i$$

przy czym dla każdego i parametr α_i jest współczynnikiem wagowym dla (dodatniego) odchylenia p_i od i -tego celu przy

$$f_i(\mathbf{x}) - p_i \leq t_i$$

przy czym t_i reprezentują cele.

Szczerze mówiąc, wszystkie powyższe metody są zbyt uproszczone i nie-zbyt dobrze pasują do wielokryterialnego podejmowania decyzji w świecie rzeczywistym. Każda z nich wymaga znaczającej modyfikacji w celu dostosowania jej do danej sytuacji, która nie różni się od uwagi odnoszącej się do dopasowywania metod liniowych do problemów nieliniowych.

Ludzie badali wielokryterialne podejmowanie decyzji przez wiele dekad – co najmniej od czasu pamiętnej pracy Keeneya i Raiffy [255] albo nawet Neumanna i Morgensterna [481]. Keeney i Raiffa [255] zaproponowali pomysł polegający na nałożeniu struktury na problem decyzji wielokryterialnego podejmowania i wprowadzeniu spojrzenia na niego od góry do dołu tak, że najpierw widoczne są cele na najwyższym poziomie, a potem kolejno na niższych poziomach, dzięki czemu tworzy się struktura hierarchiczna. Fogel w [162] opisał podobne podejście określone jako podejście „Valuated State Space” (wartościowanej przestrzeni stanów), które może dać bardzo elastyczną metodę rozwią-

¹Z tą metodą jest luźno związane tak zwane podejście „leksykograficzne”, w którym każdy cel ma określony priorytet, a przy obliczaniu najpierw jest używany pierwszy cel z więzami wyznaczonymi przez drugi z kolei i tak dalej [170].

²Dla celów typu „mniejsze lub równe”.

zywania problemów wielokryterialnego podejmowania decyzji bez odwoływanie się do rozważyń związanych z optymalnością w sensie Pareto.

15.1.1. Valuated State Space

Podejście Valuated State Space zaczyna się od pytania ludzi z autorytetem w danej dziedzinie, jakie są ważne parametry zadania. Na przykład na początku tego rozdziału omawialiśmy podejmowanie decyzji dotyczących kupowania samochodu, na którą miały wpływ takie parametry, jak cena, wygląd, wygoda, użyteczność i bezpieczeństwo. Ograniczmy się do tego zestawu. Zgodnie z wynikami Millera [322] dla dowolnego problemu ze świata rzeczywistego mamy do czynienia z 7 ± 2 różnymi parametrami, nad którymi warto się zastanawiać, a przynajmniej ludzie w ten sposób postrzegają problemy. W naszym przypadku mamy pięć parametrów i stanowią one pierwszy poziom Valuated State Space.

Zanim rozważymy jakkolwiek rodzaj istotności względnej parametrów, każdy z nich musi zostać określony za pomocą podparametrów i, być może, podpodparametrów i tak dalej aż do momentu, gdy będzie można zmierzyć konkretny stopień osiągnięcia. Jeśli chodzi, na przykład, o koszt, to jakie są elementy kosztu? Mamy koszt nalepki, ale to jedynie ozdoba okna, możemy więc go pominąć. Mamy też wynegocjowaną cenę – to już jest prawdziwy koszt. Co jeszcze? Jak wygląda finansowanie? Większość ludzi nie kupuje samochodu za gotówkę, ale bierze kredyt na określony czas. Koszt obejmuje zatem miesięczne opłaty i okres dokonywania tych opłat. Co jeszcze? Jak się ma sprawa ubezpieczenia? Właściciel Porsche 911 Carrera może wydawać na to dość dużą sumę. Dalej idą opłaty licencyjne dla rządu i, być może, jeszcze podatki od luksusu, które możemy ująć łącznie w grupę zatytułowaną „opłaty dodatkowe”. Mamy jeszcze do czynienia z utratą wartości opisywaną za pomocą wielkości kwoty, którą mógłbyś otrzymać za samochód, gdybyś go sprzedał. Parametr ten należy mieć na względzie nawet przed kupnem samochodu. Liczą się też inne rzeczy, na przykład koszty paliwa i serwisowania i, być może, nawet opłaty za parking i inne koszty, ale na potrzeby naszego przykładu pominiemy je tutaj.

Czy możemy zmierzyć każdy z tych podparametrów? Zobaczmy. Wynegocjowana cena jest mierzalna. Sposób finansowania nie jest mierzalny, ale możemy sprawić, że będzie, dzieląc go dalej na podpodparametry jak wielkość raty i czas, na jaki jest kredyt. Cena ubezpieczenia jest mierzalna. Tak zwane opłaty dodatkowe są mierzalne. Utrata wartości nie jest mierzalna bezpośrednio, ale możemy ją bezpośrednio oszacować. Patrząc na to hierarchicznie, mamy:

1.0 Koszt

- 1.1 Wynegocjowana cena
- 1.2 Finansowanie
 - 1.2.1 Wielkość raty
 - 1.2.2 Okres kredytu
- 1.3 Ubezpieczenie
- 1.4 Opłaty dodatkowe
- 1.5 Obniżenie wartości

Ponieważ każdy z parametrów najniższego poziomu jest mierzalny, następny krok polega na zdefiniowaniu stopnia osiągnięcia dla każdego parametru, który ma dla Ciebie znaczenie. Jeśli jesteś matematykiem, to możesz mieć ochotę na wprowadzenie continuum stopni osiągnięcia. Na przykład, jeśli chodzi o cenę, to może ona przybierać wartości od „za darmo” do jakiejś większej wartości, powiedzmy, 80 000 \$. Wartość znajdowania się gdzieś w tym zakresie nie skaluje się jednak liniowo do wielkości ceny. Z punktu widzenia psychologicznego ludzie mają tendencję do ustanawiania przedziałów klasowych – czasami określanych jako „progi bólu” – stanowią one dyskretne segmenty określające różne stany w Valuated State Space. Na przykład, jeśli chodzi o wynegocjowaną cenę, możemy sobie wyobrazić, że czyjeś przedziały klasowe to „mniej niż 5000 \$”, „między 5000 \$ a 10 000 \$”, „między 10 000 \$ a 20 000 \$”, „między 20 000 \$ a 30 000 \$”, „między 30 000 \$ a 40 000 \$”, „między 40 000 \$ a 80 000 \$” oraz „więcej niż 80 000 \$”. Przedziały te oddają pojęcia ważne dla kupującego i fakt, że 35 000 \$ jest o 4000 \$ mniej niż 31 000 \$, nie robi dużej różnicy kupującemu. Możesz uważać, że to nieprawdopodobne, ale tak właśnie działa ludzka psychika. Po zdefiniowaniu przedziałów klasowych jest możliwe przypisanie każdemu z przedziałów wartości stopni osiągnięcia, zwykle w dziesięciostopniowej skali, przy czym 10 oznacza najlepszą ocenę, a 0 najgorszą. W naszym przypadku możemy przyjąć, że stopnie osiągnięcia są wyznaczone następująco:

- 10 „mniej niż 5000 \$”,
- 9 „między 5000 \$ a 10 000 \$”,
- 7 „między 10 000 \$ a 20 000 \$”,
- 5 „między 20 000 \$ a 30 000 \$”,
- 3 „między 30 000 \$ a 40 000 \$”,
- 2 „między 40 000 \$ a 80 000 \$”,
- 0 „więcej niż 80 000 \$”.

Przypisanie zero punktów ostatniej kategorii odpowiadałoby preferencjom osoby, która nigdy w życiu nie mogłaby sobie pozwolić na samochód za 80 000 \$.

W tej metodzie przypisywania wartości przedziałom klasowym nie pojawia się problem łączenia pewnych celów, z których jedne dobrze byłoby minimałizować, inne zaś dobrze byłoby maksymalizować. Zawsze tak możemy ułożyć przedziały, aby przedziałowi klasowemu najbliższemu optimum przypisać najwyższą ocenę. Nie ma tutaj też problemu ze skalowaniem funkcji oceny lub normalizacją funkcji, które dotyczą różnych rodzajów jednostek, takich jak odległość i czas. Pozostaje przypisać parametrom wagi istotności, które odzwierciedlają subiektywne, związane z celami odczucia kupującego.

Kompletna Valuated State Space z uwzględnieniem kosztu dla osoby, która kupuje samochód, może wyglądać tak:

- | | | |
|----|-----|----------------------------|
| 10 | 1.0 | Koszt |
| 10 | 1.1 | Wynegocjowana cena |
| | 10 | mniej niż 5000 \$ |
| | 9 | między 5000 \$ a 10 000 \$ |

- 7 między 10 000 \$ a 20 000 \$
- 5 między 20 000 \$ a 30 000 \$
- 3 między 30 000 \$ a 40 000 \$
- 2 między 40 000 \$ a 80 000 \$
- 0 więcej niż 80 000 \$

4 1.2 Finansowanie

- 8 1.2.1 Wielkość raty

 - 10 0%
 - 9 między 0 a 1%
 - 7 między 1 a 3%
 - 4 między 3 a 5%
 - 1 więcej niż 5%

- 7 1.2.2 Okres kredytu

 - 10 bez okresu
 - 9 mniej niż 1 rok
 - 8 1, 2 lub 3 lata
 - 7 4 lub 5 lat
 - 4 więcej niż 5 lat

3 1.3 Ubezpieczenie

- 10 poniżej 100 \$ na rok
- 9 między 100 \$ a 300 \$ na rok
- 6 między 300 \$ a 500 \$ na rok
- 4 między 500 \$ a 1000 \$ na rok
- 1 więcej niż 1000 \$ na rok

1 1.4 Opłaty dodatkowe

- 3 1.4.1 Opłaty licencyjne

 - 10 mniej niż 1% wynegocjowanej ceny
 - 8 między 1 a 3% wynegocjowanej ceny
 - 4 więcej niż 3% wynegocjowanej ceny

- 10 1.4.2 Podatki od luksusu

 - 10 bez podatku
 - 1 samochód kwalifikuje się do opodatkowania

8 1.5 Obniżenie wartości (przewidywana zachowana cena przy sprzedaży)

- 10 więcej niż 90% wynegocjowanej ceny
- 8 między 70 a 90% wynegocjowanej ceny
- 6 między 50 a 70% wynegocjowanej ceny
- 4 między 30 a 50% wynegocjowanej ceny
- 2 między 10 a 30% wynegocjowanej ceny
- 0 mniej niż 10% wynegocjowanej ceny

Aby skonstruować pełną Valuated State Space, należałyby jeszcze opisać w podobny sposób wszystkie pozostałe istotne parametry zakupu (wygląd, wygody, użyteczność, bezpieczeństwo).

Po ustaleniu wszystkich stopni osiągnięcia, przedziałów klasowych i wag istotności względnej, następnym krokiem jest stwierdzenie, czy któryś z parametrów jest *krytyczny*. Taki parametr ma specjalne znaczenie: mówi się, że parametr jest krytyczny, kiedy nieuzyskanie żadnego stopnia osiągnięcia w przypadku takiego parametru neguje sens wkładu wszystkich pozostałych parametrów. Konieczny, ale niewystarczający warunek oznaczający, że parametr jest krytyczny, polega na tym, iż wartość stopnia osiągnięcia dla najgorszego przedziału klasowego musi być równa zeru. Oczywistym przykładem parametru krytycznego w przypadku samochodu mogłyby być wynegocjowana cena. Jeśli nie możesz zapłacić sumy żądanej, to nie ma znaczenia, jak dobry jest samochód, jaki jest jego kolor, ani też, ile desek surfingowych można nim przewieźć. Takiego samochodu po prostu nie kupujesz!

Jeśli mamy tylko jeden parametr krytyczny, to wystarczy, iż zrozumie my, że jeśli zajdzie najgorsza sytuacja, to wartość całej oceny będzie równa zeru niezależnie od tego, jakie wielkości odpowiadają pozostałym przedziałom. Przypadkowi, gdy mamy do czynienia z więcej niż jednym parametrem krytycznym, przyjrzymy się za chwilę dokładniej.

Zadanie podjęcia optymalnej decyzji ze względu na wszystkie cele przy sformułowaniu problemu w postaci Valuated State Space polega na określeniu, jaka decyzja – jaki przedział zasobów – doprowadzi nas do najlepszego zestawu stanów (przedziałów klasowych) w Valuated State Space. Określenie tego wymaga wyznaczenia funkcji normalizującej, która będzie wiązała, poczawszy od najniższych poziomów, wpływ każdego z mierzalnych parametrów. Taką funkcją normalizującą mogłyby być zwykła średnia arytmetyczna lub, w przypadku istnienia parametrów krytycznych, średnia geometryczna (funkcja ważonego iloczynu). Przeliczmy jeden przykład ze względu na wspomniany powyżej koszt.

Przypuśćmy, że udałeś się do sprzedawcy samochodów i znalazłeś interesującego Cię nowego czterodrzwiowego sedana. Parametry tego samochodu mogą być takie, jak zaznaczone za pomocą znaków „X”:

10	1.0	Koszt
10	1.1	Wynegocjowana cena
	10	mniej niż 5000 \$
	9	między 5000 \$ a 10 000 \$
X	7	między 10 000 \$ a 20 000 \$
	5	między 20 000 \$ a 30 000 \$
	3	między 30 000 \$ a 40 000 \$
	2	między 40 000 \$ a 80 000 \$
	0	więcej niż 80 000 \$
4	1.2	Finansowanie
8	1.2.1	Wielkość raty
10		0%

	X	9	między 0 a 1%
		7	między 1 a 3%
		4	między 3 a 5%
		1	więcej niż 5%
		7	1.2.2 Okres kredytu
		10	bez okresu
		9	mniej niż 1 rok
	X	8	1, 2 lub 3 lata
		7	4 lub 5 lat
		4	więcej niż 5 lat
3	1.3		Ubezpieczenie
		10	poniżej 100 \$ na rok
		9	między 100 \$ a 300 \$ na rok
	X	6	między 300 \$ a 500 \$ na rok
		4	między 500 \$ a 1000 \$ na rok
		1	więcej niż 1000 \$ na rok
1	1.4		Opłaty dodatkowe
		3	1.4.1 Opłaty licencyjne
		10	mniej niż 1% wynegocjowanej ceny
	X	8	między 1 a 3% wynegocjowanej ceny
		4	więcej niż 3% wynegocjowanej ceny
		10	1.4.2 Podatki od luksusu
	X	10	bez podatku
		1	samochód kwalifikuje się do opodatkowania
8	1.5		Obniżenie wartości (przewidywana zachowana cena przy sprzedaży)
		10	więcej niż 90% wynegocjowanej ceny
		8	między 70 a 90% wynegocjowanej ceny
	X	6	między 50 a 70% wynegocjowanej ceny
		4	między 30 a 50% wynegocjowanej ceny
		2	między 10 a 30% wynegocjowanej ceny
		0	mniej niż 10% wynegocjowanej ceny

Jeśli żaden z tych parametrów nie jest krytyczny, to możemy użyć średniej arytmetycznej. Obliczenia muszą być wykonywane w kierunku od najniższego poziomu do poziomów wyższych. Najpierw koncentrujemy się na 1.2 i widzimy, że punkt ten składa się z poziomów 1.2.1 i 1.2.2, które mają odpowiednio stopnie osiągnięcia 9 i 8 oraz wagę istotności względnej 8 i 7. W związku z tym ich wkład w 1.2 jest określony dla każdego parametru stosunkiem uzyskanego stopnia osiągnięcia do maksymalnego możliwego stopnia osiągnięcia z uwzględnieniem wag ich istotności względnej. W tym przypadku obliczenia wyglądają tak:

$$\frac{9}{10} \cdot \frac{8}{15} + \frac{8}{10} \cdot \frac{7}{15} = 0,480 + 0,373 = 0,853$$

co jest jeszcze przeskalowywane do skali 0–10, w wyniku czego otrzymujemy 8,53. Zauważ, że 15 to suma 8 i 7 – odpowiednich wag istotności względnej. Zatem wkład punktu 1.2 wynosi 8,53. Podobnie obliczamy wkład 1.4 na podstawie 1.4.1 i 1.4.2:

$$\frac{8}{10} \cdot \frac{3}{13} + \frac{10}{10} \cdot \frac{10}{13} = 0,185 + 0,769 = 0,954$$

co po przeskalowaniu daje 9,54. Liczby te mogą być połączone na następnym poziomie hierarchii:

$$\frac{7}{10} \cdot \frac{10}{26} + \frac{8,53}{10} \cdot \frac{4}{26} + \frac{6}{10} \cdot \frac{3}{26} + \frac{9,54}{10} \cdot \frac{1}{26} + \frac{6}{10} \cdot \frac{8}{26} = 0,691$$

czyli po przeskalowaniu 6,91. Zauważ, że tyle wynosi wkład parametru o nazwie „koszt”, a pełna wartość związana z wyborem samochodu byłaby obliczana przez połączenie wkładów wszystkich, również pozostałych, parametrów. Dwa różne samochody możemy zatem porównywać, jeśli chodzi o to, jak dobrze spełniają warunki całego procesu wielokryterialnego podejmowania decyzji, zestawiając ich końcowe oceny.

Gdyby wszystkie parametry były krytyczne na pierwszym poziomie Valuated State Space¹, to zamiast stosować ważoną średnią arytmetyczną moglibyśmy użyć ważonej średniej geometrycznej. Przy takim podejściu obliczamy iloczyn wszystkich wkładów podniesionych do potęg będących wagami istotności względnej:

$$\prod_{i=1}^m \text{Wkład}_i^{w_i}$$

przy czym mamy do czynienia z m parametrami; w_i to wagi istotności względnej przeskalowane do wartości między zerem a jedynką i sumujące się do 1,0. W ten sposób, jeśli wkład, jaki wnosi parametr, wynosi zero, to iloczyn sprawia, że cały wynik się zeruje. Przy zastosowaniu średniej geometrycznej analizowany przez nas sedan uzyskałby:

$$(0,7)^{\frac{10}{26}} \times (0,853)^{\frac{4}{26}} \times (0,6)^{\frac{3}{26}} \times (0,954)^{\frac{1}{26}} \times (0,6)^{\frac{8}{26}} = 0,684$$

Valuated State Space zapewnia wygodną strukturę służącą do oceny parametrów w kategoriach wygodnych dla ludzi, pozwala im na stosowanie subiektywnych wag istotności względnej oraz zapewnia mechanizm umożliwiający uwzględnianie stopni krytyczności parametrów. Metodę tę można rozszerzyć tak, żeby korzystała z czysto lingwistycznych opisów przedziałów klasowych oraz z arytmetyki rozmytej przy obliczaniu oceny. Co więcej, można ją rozszerzyć tak, żeby opisywała zależności między dwoma lub więcej graczami w grze, w której całkowity poziom powodzenia zależy od powodzenia poszczególnych

¹Zauważ, że parametr krytyczny niekoniecznie musi mieć krytyczne podparametry. Krytyczność to własność parametru na określonym poziomie hierarchii. Zauważ też, że nasz przykład ma charakter czysto edukacyjny, gdyż zakładamy krytyczność parametrów, które niekoniecznie mają szansę dawać zerową zapłatę, co jest warunkiem koniecznym.

graczy. Valuated State Space zapewnia uporządkowanie rankingowe wszystkich możliwych wyników i szybkie porównywanie dwóch lub więcej potencjalnych decyzji w celu określenia, która z nich jest lepsza.

15.1.2. Metody rozmyte wielokryterialnego podejmowania decyzji

W rozdziale 13 dowiedzieliśmy się, że na precyzyjne, czyli *ostre* (nierożmyte), opisy parametrów możemy patrzeć jak na skrajną postać opisu rozmytego. Logikę rozmytą możemy stosować do rozwiązywania problemów wielokryterialnego podejmowania decyzji na wiele sposobów. Jeśli powiesz, że istotność względna określonego parametru wynosi 9 w skali 0–10, to możesz też określić jego istotność jako „bardzo ważny” i możesz próbować stworzyć funkcję przynależności, która opisuje zbiór rzeczy bardzo ważnych, a także zbiór rzeczy mniej ważnych. Co więcej, możemy pomyśleć w takich kategoriach o Valuated State Space i przedziałach klasowych dla któregoś parametru. Przedziały klasowe nie muszą być rozdzielane ostrymi granicami. Mogą one mieć charakter rozmyty. Jeśli kupujesz samochód i bierzesz pod uwagę cenę, to zamiast stosować dyskretne przedziały, powiedzmy, mniej niż 10 000 \$, między 10 000 \$ a 30 000 \$ i więcej niż 30 000 \$, mógłbyś określić przedziały „tanie”, „w średniej cenie”, „drogo”. Każda konkretna cena może mieć dodatnią wartość funkcji przynależności w więcej niż jednej klasie. Samochód o cenie 27 500 \$ może mieć wartość funkcji przynależności 0,8 w zbiorze samochodów „w średniej cenie” i 0,3 w zbiorze samochodów „drogich”. Możemy następnie zastosować omawiane w rozdziale 13 metody usuwania rozmycia, żeby uzyskać jeden globalny opis.

Logikę rozmytą możemy też zastosować do opisu łączenia różnych parametrów. Najbardziej typową funkcją łączącą jest zwykła średnia arytmetyczna. Widzieliśmy też jednak, że gdy zajmujemy się parametrami, które są *krytyczne*, odpowiednia może być średnia geometryczna. Inną funkcję łączającą możemy wyprowadzić z metody rozmytej zwanej *uporządkowanymi średnimi ważonymi* (ang. *ordered weighted averages* – OWA), zaproponowanej przez Yagera [502] i wzbudzającej duże zainteresowanie (zob. [177, 214, 139] i wiele innych prac). Operator OWA jest przekształceniem $F : R^n \rightarrow R$, z którym jest związany wektor o długości n , zwykle oznaczany \mathbf{w} , przy czym każde w_i dla $i = 1, \dots, n$ jest z zakresu $[0, 1]$, suma zaś wszystkich w_i wynosi 1, oraz

$$F(a_1, \dots, a_n) = \sum_{j=1}^n w_j b_j$$

przy czym b_j jest j -tym największym elementem \mathbf{a} .

Operator OWA do żadnej z pozycji w \mathbf{a} nie stosuje żadnej ustalonej wagi, dopóki nie zostanie ustalona kolejność elementów w \mathbf{a} . Na przykład, jeśli $\mathbf{w} = (0,6, 0,2, 0,8, 0,5)^T$ oraz $\mathbf{a} = (0,9, 0,2, 0,8, 0,5)$, gdzie \mathbf{a} może oznaczać wkład pochodzący z czterech różnych parametrów, to

$$F(0,9, 0,2, 0,8, 0,5) = 0,6 \times 0,9 + 0,2 \times 0,8 + 0,1 \times 0,5 + 0,1 \times 0,2 = 0,77$$

Zwróć uwagę, że zmieniliśmy kolejność elementów a . Operator OWA przypisuje największą wagę do największego elementu a , drugą co do wielkości wagę do drugiego co do wielkości elementu itd. Jeśli a jest uporządkowane od największych elementów do najmniejszych, to łatwo możemy zauważyc, że operator OWA możemy tak ustawić, żeby dawał rozmyte operatory maksimum, minimum oraz prostej sumy. Robi się to, przyjmując w odpowiednio równe $(1, 0, \dots, 0)^T$, $(0, 0, \dots, 1)^T$ lub $(1/n, \dots, 1/n)^T$. Fuller w pracy [177] przedstawia interesujący przykład wykorzystania wywiadów z ekspertami do określenia rozmytego stopnia istotności względnej parametrów i opracowania rozmytych kwantyfikatorów uwzględniających opinie ekspertów. W pracach [71, 140, 293, 405] można znaleźć przeglądy zastosowań logiki rozmytej do problemów wielokryterialnego podejmowania decyzji. Pojęcia logiki rozmytej możemy też stosować do wnioskowania opartego na przypadkach [392]. Dopasowujemy tutaj znanne przypadki, stosując rozmyte opisy, takie jak „trocę podobny”, „zdecydowanie podobny” itp. Następnie opisujemy użyteczność poszczególnych przypadków za pomocą terminów takich jak „dość użyteczny” lub „bardzo użyteczny” na podstawie ich dominacji ze względu na jedno lub więcej kryteriów. San Pedro i Burstein [406] pokazały zastosowanie rozmytego wnioskowania opartego na przypadkach dla wielokryterialnego podejmowania decyzji przy przewidywaniu pogody.

15.1.3. Podsumowanie

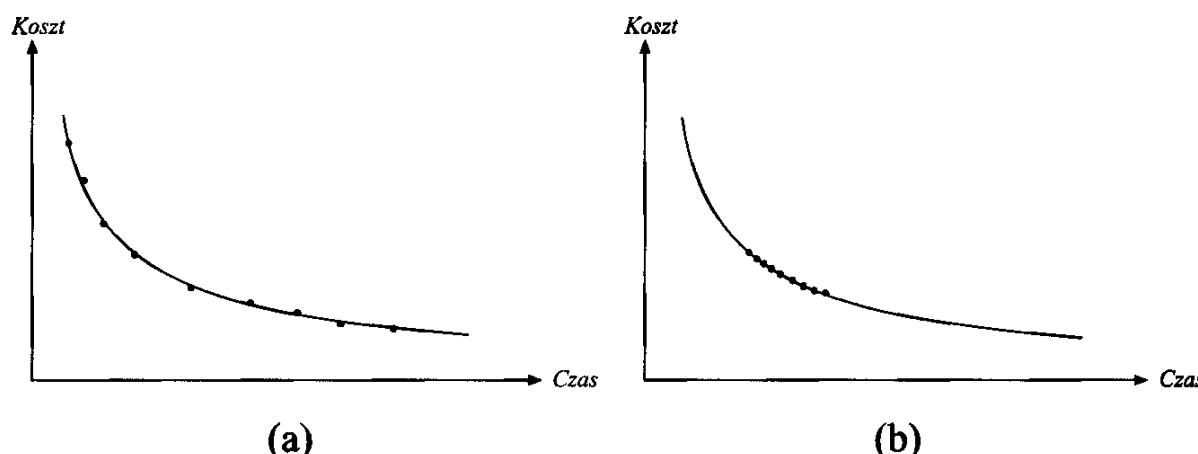
Poznaliśmy wiele zalet i wad łączenia wielu celów w jedną wartość skalarną. Do zalet należy łatwość porównywania, systematyczna strukturalizacja celów i możliwość oddawania w Valuated State Space indywidualnych, subiektywnych opinii niezależnie od istotności względnej, stopnia krytyczności i przedziałów klasowych, które stwarzają różnicę ocenającemu. Niektóre z tych zalet stwarzają pewne nowe wyzwania. Na przykład, gdy w podejmowanie decyzji są uwikłane dwie osoby lub więcej, wówczas mogą się nie zgadzać co do istotności względnej parametrów. Z kolei, jeśli jest pożądane określenie, które rozwiązania są optymalne w sensie Pareto, to wiele metod łączących oceny w jedną wartość nie będzie mogło podać odpowiedzi. Może być za to sensowne określenie zestawu możliwych rozwiązań na podstawie zbioru optymalnego w sensie Pareto, a następnie przyjrzenie im się bliżej w celu określenia, które z nich zrealizować. Przesuwa to naszą uwagę z problemu znajdowania ekstremów funkcji łączących parametry na problem znajdowania kolekcji różnych rozwiązań, które znajdują się na granicy rozwiązań optymalnych w sensie Pareto.

15.2. Podejście ewolucyjne do wielokryterialnego podejmowania decyzji

Gdy myślisz o znajdowaniu zbioru punktów na granicy obszarów, powinieneś natychmiast rozważyć możliwość użycia algorytmów ewolucyjnych. Takie podejście jest naturalne, gdyż algorytmy ewolucyjne z natury swej przetwarzają

zbiory punktów. Sztuka polega na tym, żeby opracować algorytm ewolucyjny, który rozłoży rozwiązań-kandydatów wzduż granicy Pareto, a nie będzie szukał jednego „najlepszego” rozwiązania. Na tym etapie podejmowania decyzji wszystkie rozwiązania optymalne w sensie Pareto są jednakowo ważne, gdyż zakładamy, że nie ma żadnych dodatkowych informacji wysokiego poziomu, które umożliwiłyby określenie kompromisów związanych z poszczególnymi celami. Podstawowym punktem docelowym w tym podejściu jest znalezienie rozwiązań, które są tak blisko granicy Pareto, jak to tylko jest możliwe, przy jednoczesnym zachowaniu możliwie dużej różnorodności.

Koncepcję tę ilustruje rys. 15.5. Mamy dwa zbiory rozwiązań (a) i (b) na granicy Pareto, ale mają one różną różnorodność. Wolimy korzystać z rozwiązania (a) niż z (b), gdyż pierwsze rozwiązanie lepiej „rozpoznaje” granicę Pareto. Ważne jest jednak, żeby pamiętać, że zbiór (a) optymalnych rozwiązań na granicy Pareto z rys. 15.5 jest różnorodny, ale tylko w przestrzeni celów. Często różnorodność w przestrzeni celów implikuje różnorodność w przestrzeni decyzyjnej, lecz dla złożonych i w wysokim stopniu nieliniowych problemów tak nie musi być. Może się zatem zdarzyć, że różnorodność wektorów \mathbf{x} złożonych ze zmiennych decyzyjnych znaczaco różni się od różnorodności odpowiednich rozwiązań w przestrzeni celów.



Rysunek 15.5. Dwa zbiory rozwiązań na granicy Pareto. Różnorodność na rys. (a) jest „większa” niż różnorodność na rys. (b)

Algorytmy ewolucyjne uzyskały status *metody* poszukiwania rozwiązań na granicy Pareto przy trudnych problemach wielokryterialnej optymalizacji. Oczywiście możesz używać algorytmów ewolucyjnych również do wszystkich zadań optymalizacyjnych z poprzedniego podrozdziału. Lecz w przypadku przeszukiwania populacji różnorodnych punktów o pożądanych własnościach (niezdominowane rozwiązania zróżnicowane), algorytmy ewolucyjne są naturalnym wyborem. Algorytmy ewolucyjne możemy opracować tak, żeby zamiast w wielu przebiegach zaczynających od różnych pozycji początkowych poszukiwać różnorodnych punktów na granicy, wykonać to zadanie w jednym przebiegu. Przyjrzyjmy się trochę historii stosowania obliczeń ewolucyjnych do optymalizacji w sensie Pareto.

15.2.1. Krótka historia

Precyzyjna kategoryzacja wszystkich różnych ewolucyjnych podejść do wielokryterialnej optymalizacji jest bardzo trudna. W ciągu ostatnich 20 lat pojawiło się wiele różnych rozwiązań. Szeroko rzecz ujmując, możemy jednak podzielić stosowanie algorytmów ewolucyjnych na dwie klasy: algorytmy ewolucyjne mogą być stosowane zarówno w przypadku, w którym problem wielokryterialny jest przekształcany w sformułowanie z jednym kryterium, jak i w przypadku, w którym wyszukiwanie jest przeprowadzane wprost z uwzględnieniem optymalności w sensie Pareto, bez stosowania żadnej formy *łączenia* ocen wynikających z różnych celów.

Przy zastosowaniu łączenia parametry zadania możemy poddawać systematycznemu różnicowaniu, żeby określić zbiór rozwiązań optymalnych w sensie Pareto. Jako alternatywę możemy wykorzystać algorytmy ewolucyjne, które można zastosować do bezpośredniego poszukiwania rozwiązań optymalnych w sensie Pareto przy użyciu jednej populacji i przy jednym przebiegu algorytmu. Te dwa różne podejścia możemy określić odpowiednio jako „bez metody Pareto” i „na podstawie metody Pareto”. Optymalizacja „bez metod Pareto” jest też określana jako *metoda selekcji z kryterium*, gdyż algorytm podczas selekcji przełącza się między różnymi celami. W metodzie tej wartości oceny rozwiązania dla poszczególnych celów (tzn. $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_m(\mathbf{x})$) są wykorzystywane do określenia, których rodziców należy wybrać i jak zastępować rodziców potomkami. Optymalizacja „na podstawie metody Pareto” jest nazywana też *selekcją Pareto*. W tym podejściu osobnikom w ewoluującej populacji jest przypisywana waga na podstawie ich dominacji nad innymi rozwiązaniami oraz ze względu na optymalność w sensie Pareto.

W 1984 roku Schaffer w pracach [410, 411] przedstawił pierwszą próbę prowadzenia ewolucji rozwiązań przy wielu kryteriach bez łączenia celów w jedną funkcję oceny. Podejście to nosiło nazwę VEGA, co było skrótem od „vector evaluated genetic algorithm” (algorytm genetyczny z oceną wektorową). Główny pomysł zastosowany w VEGA polegał na tym, że dla problemu z m celami w każdym pokoleniu wybierało się $1/m$ rodziców za pomocą każdej z m funkcji celu. Celem tego rozwiązania było zapewnienie, żeby osobników bardzo dobrych ze względu na jedno z kryteriów (tzw. *specjalistów*) było trudno usunąć w wyniku selekcji, przy czym osobniki o przeciętnej jakości ze względu na wiele kryteriów mają wiele szans na przeżycie, gdyż są uwzględniane przy ocenie każdego kryterium. Podejście to było oparte na nadziei, że pojawi się rozwiązanie o cechach „cudownego dziecka”, dobre ze względu na wszystkie cele, i że przy okazji nie straci się po drodze specjalistów.

W pracy [410] Schaffer badał problem ewolucji systemu klasyfikującego dla zadania rozróżnienia wzorca wieloklasowego¹. Wcześniejsze wysiłki z za-

¹ Był to prosty problem polegający na redukcji sygnałów EMG (zapisu czynności elektrycznej mięśni) do 12-bitowych wzorców. Interpretacja była taka, że każdy z sześciu mięśni ludzkiej nogi mógł być albo aktywny, albo nieaktywny (6 bitów) w trakcie wykonywania kroku lub kiwnięcia na boki (razy dwa) w trakcie trwania cyklu chodu. Wśród

stosowaniem tradycyjnego algorytmu ewolucyjnego dały przypadki, w których pewne dobre reguły rozróżniające dla każdej klasy rodziców były generowane w różnych osobnikach, nawet we wczesnych pokoleniach. Niestety, reguły te nigdy nie mogły być pomyślnie połączone przez ewolucję tak, żeby powstało „cudowne dziecko”. Dlaczego? Ponieważ reguła selekcji faworyzowała klasifikatory oparte na sumarycznej liczbie poprawnie sklasyfikowanych przypadków. Jeśli jeden osobnik odziedziczył użyteczne reguły, powiedzmy dla klas 1 i 2, to mógł zastąpić innego osobnika, który miał dobre reguły powiedzmy dla klasy 3, ale złe dla 1 i 2. Pierwszy z nich mógł się wydawać dwa razy lepszy od drugiego, a populacja odrzucała specjalistę, zanim pojawiła się korzystna rekombinacja. Mechanizm selekcji VEGA został zaprojektowany z myślą o chronieniu specjalistów przy zapewnianiu większej liczby potomków dla osobników, które wykazywały się dobrymi właściwościami ze względu na wiele celów.

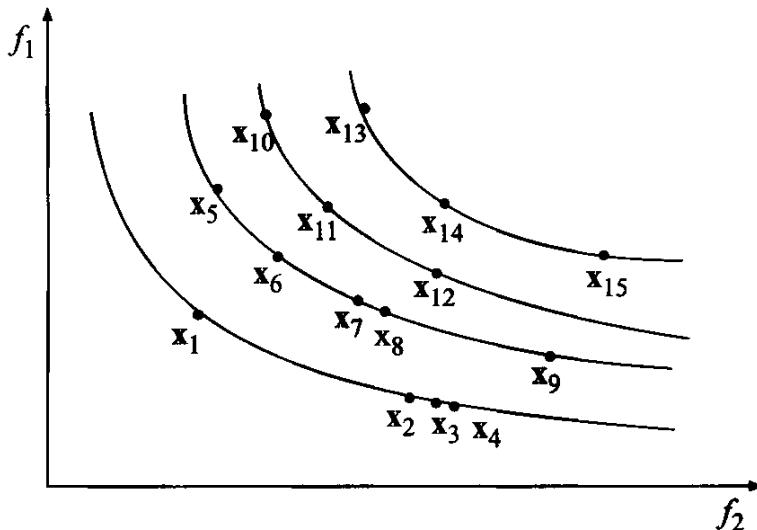
Schaffer stwierdził, że jego schemat selekcji stworzył szansę optymalizacji wielokryterialnej przy użyciu algorytmów ewolucyjnych. Podał pewne proste problemy wielokryterialne (dwa z nich są przedstawione w punkcie 15.2.2), które posłużyły mu do badania dynamiki systemu VEGA oraz do określenia jego użyteczności. Wysiłki te doprowadziły do wyznaczenia pewnych heurystyk prowadzących do rozwiązania. W jednej z nich, na przykład, rozmiar populacji zwiększał się wraz z liczbą celów. Pomysł polegał tu na tym, że wystarczająca eksploracja celów musi być wspierana przez większą różnorodność, która wynika z większego rozmiaru populacji. W innych eksperymentach, za pomocą których badano preferencje selekcji dla rozwiązań niezdominowanych, wykazano, że prowadzi to do szybkiej zbieżności do osobników o nieoptimalnej wartości. Zaobserwowano też, że VEGA ma problemy związane z *tworzeniem gatunków*; przy tym zjawisku osobniki w populacji dążą do doskonalenia się ze względu na różne kryteria, co może blokować szanse na znalezienie lepszego rozwiązania. Jeden z możliwych sposobów radzenia sobie z takim tworzeniem gatunków polega na krzyżowaniu osobników z różnych gatunków zamiast rekombinacji w sposób losowy.

Aby być wiernym historii, powinniśmy wspomnieć też pracę Fourmana [173], który w tym samym czasie co Schaffer badał podobny pomysł. Fourman najpierw próbował korzystać z selekcji leksykalnej za pomocą listy priorytetowej kryteriów, dawało to jednak kiepskie wyniki dla jego problemu projektowania obwodów. W końcu wpadł na pomysł, żeby w momencie konieczności przeprowadzenia selekcji, losowo wybierać kryterium. Schemat ten działał o wiele lepiej niż lista priorytetowa kryteriów.

Kilka lat później Goldberg w [187] zaproponował algorytm ewolucyjny, który oceniał rozwiązania na podstawie dominacji, a nie wartości związanych z poszczególnymi celami. Zaproponowana metoda działała w sposób iteracyjny, najpierw określając wszystkie rozwiązania, które nie były zdominowane. Rozwiązaniom tym przypisywano następnie ten sam wynik i usuwano z dalszych

tych klas znajdowały się klasy „normalne” oraz cztery klasy nienormalnego chodu. Schaffer wybrał ten problem, gdyż 1) był mały i 2) dane wejściowe były bitowe (naturalne dla systemów klasyfikujących).

rozoważań. Dalej proces znowu określał, które z pozostałych rozwiązań są niezdominowane. Tym rozwiązaniom przypisywano znowu ten sam wynik, z tym że gorszy niż wynik przypisany rozwiązaniom usuniętym w poprzedniej iteracji. Następnie te rozwiązania również były usuwane. Ten proces był wykonywany do momentu, gdy każdemu z rozwiązań została przypisana ocena, po czym na podstawie uzyskanej oceny wykonywano selekcję. Proces ten możemy wyobrazić sobie jak obieranie cebuli: każda warstwa odpowiada bieżącemu niezdominowanemu zbiorowi rozwiązań. Pomysł ten przedstawiono na rys. 15.6.



Rysunek 15.6. Kilka rozwiązań przypisanych do różnych niezdominowanych linii granicznych

Goldberg [187] wskazał na znaczenie zachowania różnorodności rozwiązań albo w przestrzeni rozwiązań, albo w przestrzeni celów. Zaproponował też metodę współdzielenia wartości (punkt 11.2.1), zapobiegającą zbieżności algorytmu ewolucyjnego do jednego rozwiązania na granicy Pareto i utrzymującą różnorodne rozwiązania na granicy rozwiązań niezdominowanych. Na przykład na rys. 15.6 rozwiązania x_1 , x_2 , x_3 i x_4 na pierwszej niezdominowanej granicy nie są rozłożone jednostajnie: rozwiązanie x_1 jest odizolowane, rozwiązania x_2 , x_3 i x_4 są zgrupowane. W związku z tym rola mechanizmów współdzielenia przystosowania polegałaby na kładzeniu większego nacisku na rozwiązanie x_1 i na zabezpieczaniu przed jego zniknięciem w dalszych pokoleniach.

Te dwa odmienne podejścia wzbudziły zainteresowanie społeczności zajmującej się obliczeniami ewolucyjnymi. Począwszy od wczesnych lat dziewięćdziesiątych XX wieku, wielu naukowców zaczęło informować o nowych podejściach, pierwszych wynikach i nowych zadaniach testowych. Pojawiało się też coraz więcej prac na temat zastosowań metod ewolucyjnych do rozwiązywania pochodzących ze świata rzeczywistego problemów optymalizacji wielokryterialnej. Nie było to zaskakujące, gdyż metody obliczeń ewolucyjnych są odpowiednio nie tylko do przeszukiwania nadzwyczajnie dużych przestrzeni poszukiwań (jak wynika z wcześniejszych rozdziałów tej książki), ale mogą też wykorzystywać podobieństwa między rozwiązaniami oraz przybliżać przy jednym uruchomieniu całą granicę rozwiązań optymalnych w sensie Pareto.

Przed pojawiением się tego wzrostu zainteresowania najwcześniejsze wyniki związane z optymalizacją wielokryterialną z użyciem algorytmów ewolucyjnych skupiały się na selekcji opartej na łączeniu funkcji celów. Rozwiązania tego rodzaju pojawiły się już w 1957 roku w pracy Boxa [53], w której zaproponował on podejście do swojej metody ewolucyjnej oparte na ϵ -ograniczeniach (przy czym nazwa ta została wprowadzona formalnie wiele lat później). Box rozważał sposoby określania, który z wielu celów obrać jako główne kryterium, a które uwzględniać jako ograniczenia. We wczesnych latach sześćdziesiątych XX wieku L. Fogel [161, 167] badał problem wielokryterialny, w którym ewolucji były poddawane automaty ze skończoną liczbą stanów używane do przewidywania ciągów symboli. Problem wielokryterialny polegał na znalezieniu automatów o lepszych zdolnościach przewidywania przy jednoczesnej minimalizacji ich złożoności. Łączna funkcja oceny obejmowała liniową średnią ważoną ze: 1) skumulowanej wypłaty wynikającej z przewidzianych symboli w porównaniu z symbolami, które się rzeczywiście pojawiły i 2) liczby stanów w automacie ze skończoną liczbą stanów.

W 1990 roku Fogel [143] rozszerzył te wczesne wyniki, stosując je do problemu identyfikacji systemu. Zadanie polegało na znalezieniu optymalnych modeli średnich kroczących autoregresji (tak z uwzględnieniem, jak i bez zewnętrznych danych; modele te oznaczono odpowiednio jako ARMAX i ARMA) na podstawie zaobserwowanych danych. Za pomocą kryteriów z zakresu teorii informacji Fogel [143] opracował modele, które były oceniane ze względu na ich dopasowanie do zaobserwowanych danych oraz ich złożoność mierzoną liczbą stopni swobody. Zastosowano dwa kryteria informacyjne: kryterium informacyjne Akaike (ang. *Akaike's information criterion – AIC*) oraz zasadę minimalnej długości opisu (ang. *minimal description length – MDL*). Są to powszechnie stosowane zasady przy budowaniu modeli, znajdujące kompromis między jakością dostosowania a złożonością modelu, aczkolwiek korzystają z różnych matematycznych formuł do mierzenia złożoności. Fogel rozszerzył te wyniki w 1991 roku [144] tak, żeby obejmowały ewolucję sieci neuronowych przy selekcji opartej na łączeniu funkcji celów do minimalizacji zarówno błędu modelu, jak i złożoności.

Na początku lat dziesiątych XX wieku pojawiło się wiele zastosowań ewolucyjnej optymalizacji wielokryterialnej opartej na łączeniu funkcji celów. Na przykład Syswerda i Palmucci [462] zastosowali ją do oceny planów pracy, Jacob i in. [236] użyli jej do planowania zadań. Wienke i in. [497] wykorzystali podejście ewolucyjne do problemu optymalizacji natężenia linii emisji atomowej pierwiastków śladowych, stosując metodę programowania celu (zwawaną też podejściem z doborem sposobu łączenia celów). Ranjithan i in. [377] użyli metody opartej na ϵ -ograniczeniach do rozwiązywania problemu zapobiegania zatruciowi wód gruntowych. Kursawe [276] zbudował ogólny wielokryterialny system strategii ewolucji oparty na porządku leksykograficznym, w którym przy każdym kroku był losowo wybierany zgodnie z pewnym wektorem prawdopodobieństw jeden z celów. Było to podobne do wspomnianych wcześniej eksperymentów z selekcją kryterium Schaffera i Fourmana. Hajela i Lin [206] zoptymalizowali dziesięcioprętową płaską kratownicę, w której waga i przesu-

nięcie miały być minimalizowane za pomocą podejścia podobnego do VEGA. Wagi każdego z celów były dołączone do użytej struktury danych, a różnorodność była utrzymywana przez współdzielenie przystosowania oraz ograniczenia w tworzeniu par. W przeciwnieństwie do metody selekcji opartej na łączaniu funkcji celów, Hilliard i in. [216] użyli metody rankingowej opartej na optymalności w sensie Pareto do rozwiązywania problemu planowania z dwoma celami uwzględniającego czas i opóźnienie, a Liepins i in. [283] stwierdzili, że dla kilku problemów pokrycia zbiorami podejście oparte na metodzie Pareto dawało lepsze wyniki niż podejście bez korzystania z niej.

Wszystkie opisane w poprzednim akapicie prace zostały opublikowane w 1991 lub 1992 roku. Po tym okresie więcej wysiłku skierowano na podejścia z wykorzystaniem optymalności w sensie Pareto. Na przykład w 1993 roku przedstawiono trzy niezależne badania; wszystkie oparte na podejściu w sensie Pareto.

Srinivas i Deb [448] zaproponowali metodę zwaną genetycznym algorymem sortowania bez zdominowania (ang. *nondominated sorting genetic algorithm* – NSGA). W populacji jest określany ranking osobników ze względu na brak dominacji, a rozwiązaniom w najlepszym zbiorze – tym na pierwszej linii granicznej – jest przypisywane przystosowanie równe liczbie osobników w populacji. Funkcje współdzielenia są wykorzystywane do utrzymywania różnorodności populacji, przy czym każda linia graniczna ma własną funkcję współdzielenia. Dla każdego rozwiązania \mathbf{x} na granicy są obliczane znormalizowane odległości euklidesowe od pozostałych rozwiązań \mathbf{y} z tej samej linii granicznej. Wartość przystosowania rozwiązania \mathbf{x} jest także modyfikowana przez tak zwaną licznosć niszy $nc_{\mathbf{x}}$ dla \mathbf{x} :

$$nc_{\mathbf{x}} = \sum_{\mathbf{y}} sh(d(\mathbf{x}, \mathbf{y}))$$

(definicję funkcji współdzielenia sh podano w punkcie 11.2.1). Pierwotne przystosowanie osobnika jest dzielone przez licznosć niszy.

Przypisanie pierwotnych wartości przystosowania (tzn. przystosowania przed modyfikacją) dla rozwiązań z drugiej linii granicznej jest wykonywane dopiero po wykonaniu powyższych modyfikacji wartości dla rozwiązań z pierwszej linii. Te pierwotne wartości przystosowania dla osobników z drugiej linii są mniejsze od najmniejszej zmodyfikowanej wartości dla osobników z pierwszej linii. Zauważ jeszcze, że odległość współdzielenia jest obliczana w przestrzeni decyzyjnej (inaczej niż w pozostałych metodach, gdzie funkcję współdzielenia liczy się w przestrzeni celów).

Najwyraźniej, ponieważ rozwiązania na pierwszej niezdominowanej linii granicznej mają najwyższe wartości przystosowania, szanse na ich wybranie (użyto stochastycznej selekcji proporcjonalnie do pozostałe części) do reprodukcji są większe w stosunku do innych rozwiązań. Dzięki temu system może przeszukiwać niezdominowane obszary przestrzeni poszukiwań i zbiegać w ich kierunku. Pełne omówienie tego systemu wraz z początkowymi wynikami eksperymentalnymi podano w [448].

W pracy [234] Horn i Nafpliotis przedstawili odmianę algorytmu genetycznego, zwaną niszowym genetycznym algorytmem Pareto (ang. *niched Pareto genetic algorithm* – NPGA), który wykorzystuje binarną selekcję turniejową opartą na dominacji Pareto. W trakcie takiej selekcji turniejowej dwaj losowo wybrani rodzice są porównywani z wybraną podpopulacją o rozmiarze s , tzn. każdy z dwóch wybranych rodziców jest porównywany z każdym z s rozwiązań z tej podpopulacji. W wyniku albo jeden z rodziców w oczywisty sposób wygrywa (tzn. dominuje nad wszystkimi s rozwiązaniami w wybranej podpopulacji, przy czym dla drugiego tak nie jest) lub zachodzi remis (tzn. obaj rodzice dominują nad wszystkimi s rozwiązaniami lub są zdominowani przez co najmniej jedno z s rozwiązań w podpopulacji). W tym drugim wypadku obaj rodzice są umieszczani w populacji potomków (która jest zapełniana w trakcie trwania tego procesu) i są obliczane ich liczności niszy: rodzic o mniejszej liczności niszy wygrywa¹. Ta sama procedura jest wykonywana przy wyborze drugiego rodzica, po czym dwaj wybrani rodzice podlegają reprodukcji i tworzą potomstwo. Na tym etapie populacja potomków jest niepusta i od tego momentu do rozstrzygania remisów jest wykorzystywana liczność niszy.

Podobnie jak w przypadku NSGA, są faworyzowane rozwiązania niezdominowane, gdyż dominacja jest sprawdzana w stosunku do s wybranych rozwiązań. Co więcej, wspierana jest różnorodność, gdyż są preferowani rodzice z mniej zatłoczonych obszarów. Pełne omówienie tego systemu wraz z początkowymi wynikami eksperymentalnymi podano w [234].

Fonseca i Fleming w [168] wprowadzili wielokryterialny algorytm genetyczny (ang. *multiobjective genetic algorithm* – MOGA). Było to prawdopodobnie (o ile wiemy) pierwsze podejście, w którym położono nacisk na niezdominowane rozwiązania oraz na procedury promujące różnorodność. Przystosowanie było przypisywane rozwiązaniom w następujący sposób.

Najpierw dla każdego rozwiązania \mathbf{x} algorytm przypisywał ocenę równą jeden plus liczba rozwiązań w populacji, które dominują nad \mathbf{x} . Każde niezdominowane rozwiązanie miało przypisany wynik 1, gdyż w populacji nie było rozwiązań dominujących nad nim. Zauważ, że w każdej populacji jest co najmniej jedno rozwiązanie ocenione na 1. Co więcej, maksymalna wartość dowolnej oceny jest równa najwyższej rozmiarowi populacji *pop_size*. Następnie algorytm przypisuje wyjaśnione poniżej nieprzetworzone wartości przystosowania. Wykorzystuje on przy tym odwzorowanie oparte na wspomnianej powyżej funkcji oceny. Rozwiązania ocenione na 1 otrzymują największe nieprzetworzone wartości przystosowania: jeśli mamy k_1 rozwiązań o ocenie 1, to otrzymują one (w dowolnej kolejności) nieprzetworzone wartości przystosowania *pop_size*, $pop_size - 1, \dots, pop_size - k_1 + 1$. Następnie rozwiązania o ocenie 2 otrzymują nieprzetworzone wartości przystosowania z następnej grupy dużych liczb: $pop_size - k_1, pop_size - k_1 - 1, \dots, pop_size - k_1 - k_2 + 1$ (jeśli nie ma rozwiązań o ocenie 2, to algorytm wybiera grupę rozwiązań o najmniejszej ocenie). Procedura ta jest kontynuowana dopóty, dopóki wszystkim osobnikom nie zostanie

¹Na początku tego procesu, gdy populacja potomków jest pusta, jeden z rodziców jest wybierany losowo.

przypisana nieprzetworzona wartość przystosowania. W następnym kroku algorytm zmienia nieprzetworzoną wartość przystosowania na średnią wartość przystosowania dla wszystkich osobników: każdemu osobnikowi o takiej samej ocenie jest przypisywana ta sama wartość średnia przystosowania, która jest średnią wszystkich nieprzetworzonych wartości przystosowania osobników o takiej samej ocenie. Na przykład wszystkim osobnikom o ocenie 1 zostanie przypisana średnia wartość przystosowania

$$\frac{1}{k_1} \sum_{i=1}^{k_1} (pop_size - i + 1)$$

Następnie osobnikom jest przypisywana wspólna wartość przystosowania uzyskana przez podzielenie wartości średniej przystosowania osobnika przez liczbę jego niszy. Wreszcie te wspólne wartości przystosowania są skalowane tak, żeby średnia wartość przystosowania wszystkich osobników o takiej samej pierwotnej ocenie była taka jak poprzednio – tuż po uśrednieniu ich nieprzetworzonych wartości przystosowania.

Podobnie jak w dwóch innych opisanych wyżej podejściach, faworyzowane są rozwiązania niezdominowane, gdyż bezpośrednio jest sprawdzana dominacja wśród wszystkich rozwiązań w populacji i odpowiednio jest przypisywany wynik. Co więcej, jest w nich też promowana różnorodność, gdyż rodzice z mniej zatłoczonych obszarów są częściej bardziej pożądani, ponieważ wartość przystosowania jest modyfikowana z uwzględnieniem liczności niszy. Pełne omówienie tego systemu wraz z początkowymi wynikami eksperymentalnymi podano w [168].

Badania te zostały rozwinięte w setkach prac związanych z różnymi aspektami optymalizacji wielokryterialnej. Szczegółowy ich przegląd znajduje się w [74].

W podrozdziale 7.4 wskazaliśmy, że selekcja *elitarna* (w której najlepsze rozwiązania w populacji mają gwarancję przetrwania do następnego pokolenia) może być zastosowana w algorytmach ewolucyjnych. W ogromnej większości ostatnio prowadzonych badań w zakresie ewolucyjnej optymalizacji wielokryterialnej jest wykorzystywana jakaś forma selekcji elitarnej.

Przy optymalizacji jednokryterialnej najlepszego osobnika, po jego odkryciu, łatwo jest zidentyfikować. Jak jednak przekonaliśmy się wcześniej w tym rozdziale, w optymalizacji wielokryterialnej korzystającej z podejścia Pareto mamy raczej do czynienia ze zbiorem najlepszych (tj. niezdominowanych) rozwiązań, a nie z jednym najlepszym rozwiązaniem. Bezpośrednia implementacja selekcji elitarnej w wyłącznie wielokryterialnym środowisku wymuszałaby przeprowadzanie przynajmniej wszystkich niezdominowanych rozwiązań w populacji do następnego pokolenia. Użycie selekcji elitarnej musi być wykonane ostrożnie, gdyż po kilku pokoleniach, szczególnie gdy mamy do czynienia z wieloma celami, większość osobników w populacji będzie niezdominowana. Oznaczałoby to, że wszystkie kwalifikowałyby się jako rozwiązania elitarne. Zastosowanie selekcji elitarnej do nich wszystkich zagro-

ziłoby różnorodności populacji, co jest szczególnie groźne przy zastosowaniu operatora różnicowania zmniejszającego różnorodność (np. krzyżowania). W tym wypadku ważne jest, żeby selekcja elitarna była dodatkowo kontrolowana.

W połowie lat dziewięćdziesiątych XX wieku pojawiły się prace wprowadzające elitarne algorytmy ewolucyjne do problemów wielokryterialnych, w których naukowcy zajmowali się problemem sterowania selekcją elitarną. I tak na przykład Rudolph [399] zaproponował złożony mechanizm identyfikacji najbardziej obiecujących osobników dla następnego pokolenia: w populacji potomków były wyszukiwane niezdominowane rozwiązania i porównywane ze wszystkimi osobnikami z populacji rodziców (populacje te były badane oddzielnie). Jeśli rozwiązanie-potomek dominuje nad jakimś (co najmniej jednym) rozwiązaniem z populacji rodziców, to: a) potomek jest umieszczany w zbiorze P_1 i b) zdominowani rodzice są usuwani z populacji. Jeśli rozwiązanie-potomek nie dominiuje nad żadnym z rodziców i dodatkowo nie jest zdominowane przez żadnego z nich, to jest umieszczane w zbiorze P_2 . Intuicyjnie zbiory P_1 i P_2 to początkowe zbiory obiecujących osobników dla następnego pokolenia, aczkolwiek osobniki w P_1 są bardziej obiecujące niż osobniki z P_2 . W końcu następna generacja jest tworzona przez wybieranie osobników w pewnym uprzywilejowanym porządku. Podobne podejście było badane przez Deba i in. [99] z tą różnicą, że populacje rodziców i potomków były łączone przed określeniem rozwiązań niezdominowanych. Osyczka i Kundu [341] eksperymentowali z podejściem, w którym były utrzymywane dwie populacje: standardowa i elitarna, w której znajdowały się dotychczas znalezione niezdominowane rozwiązania. Specjalny sposób określania przystosowania w tym algorytmie kładzie nacisk na rozwiązania bliskie granicy Pareto, a także utrzymuje różnorodność wśród niezdominowanych rozwiązań. Podobnie Zitzler i Thiele [509] zaproponowali algorytm, który utrzymywał zewnętrzną populację dotychczas znalezionych niezdominowanych rozwiązań. Ten elitarny zbiór brał również udział w różnicowaniu, co miało kierować zwykłą populację w stronę obiecujących obszarów przestrzeni poszukiwań.

Pojawiło się wiele dobrych przeglądów algorytmów ewolucyjnych w zastosowaniu do optymalizacji wielokryterialnej. Fonseca i Fleming w [169] przedstawili przegląd metod łączących wiele kryteriów w jedną funkcję oceny, dającą w wyniku jedną wartość, a także przegląd metod, opartych na optymalności w sensie Pareto i dających zbiór wartości. Zitzler i in. w [507] podali empiryczne porównanie ośmiu algorytmów ewolucyjnych do optymalizacji wielokryterialnej w zastosowaniu do sześciu zadań testowych. Veldhuizen [479] porównał cztery algorytmy na siedmiu zadaniach testowych, a Knowles i Corne [262] porównali 13 algorytmów na sześciu zadaniach testowych. Badania te wykazały, że selekcja elitarna jest bardzo cenny przy implementowaniu wysokiej jakości algorytmów ewolucyjnych do optymalizacji wielokryterialnej. Coello w [74] podał obszerny przegląd metod ewolucyjnej optymalizacji wielokryterialnej. W niedawno wydanych książkach [75, 98, 340] oraz we wspomnianych artykułach przeglądowych znajduje się doskonała prezentacja klasycznych i ewolucyjnych metod optymalizacji wielokryterialnej.

15.2.2. Algorytm ewolucyjny dla wielokryterialnego NLP

Prawdopodobnie najgruntowniej przebadanym problemem optymalizacji wielokryterialnej jest problem zaproponowany przez Schaffera w jego wczesnej pracy [410]. Jest to problem testowy z jedną zmienną i dwoma celami, f_1 oraz f_2 , określony następująco:

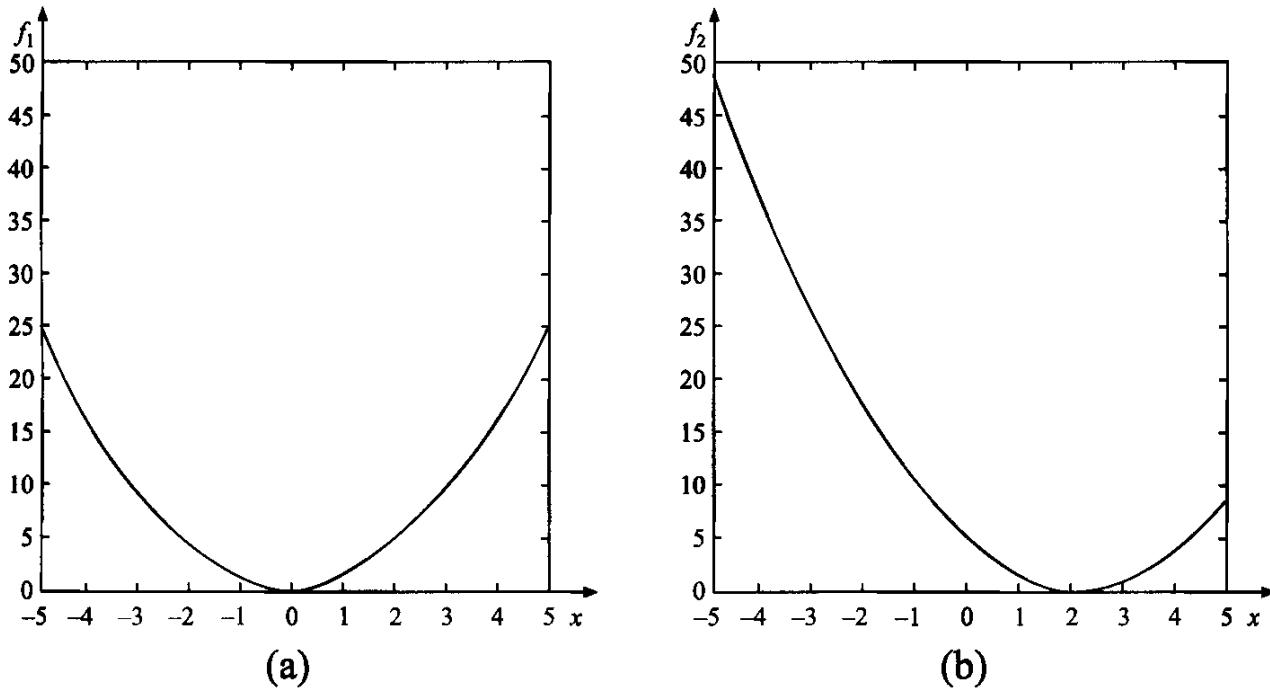
$$\begin{aligned} & \text{zminimalizuj } f_i(x), \quad i = 1, 2 \\ & \text{przy } -A \leq x \leq A \end{aligned}$$

przy czym:

$$\begin{aligned} f_1(x) &= x^2 \\ f_2(x) &= (x - 2)^2 \end{aligned} \tag{15.1}$$

Zauważ, że $\mathbf{x} = (x_1)$, gdyż w zadaniu mamy do czynienia tylko z jedną zmienną. Żeby uprościć notację, opuścimy indeks, tzn. zamiast pisać x_1 , będziemy pisali po prostu x .

Na rysunku 15.7 przedstawiono wykresy obu celów, f_1 i f_2 , jako funkcje zmiennej decyzyjnej x . Jasne jest, że problem ten ma rozwiązania optymalne w sensie Pareto dla $x \in [0, 2]$: między -5 a 0 obie funkcje maleją, między 0 a 2 funkcja f_1 rośnie, funkcja f_2 maleje, między 2 a 5 obie funkcje rosną.

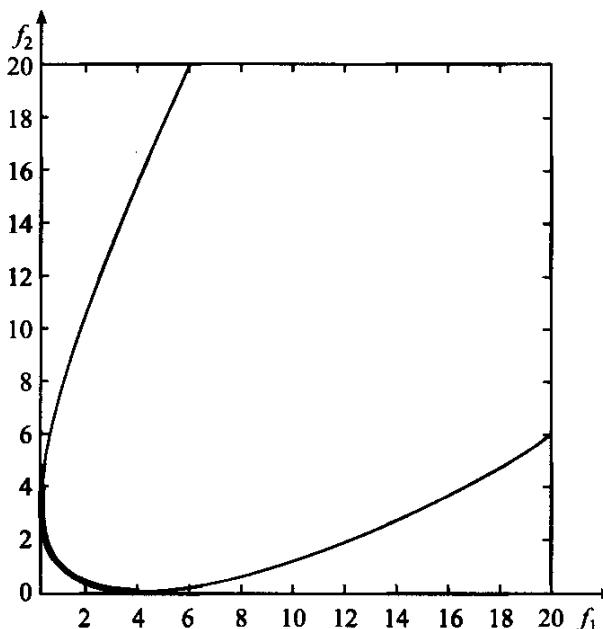


Rysunek 15.7. Zmienna decyzyjna i dwa cele: (a) f_1 oraz (b) f_2 dla $A = 5$

Możliwe jest narysowanie wartości f_2 w zależności od f_1

$$f_2 = (\sqrt{f_1} - 2)^2 \tag{15.2}$$

przez proste zastąpienie x we wzorach (15.1). Na rysunku 15.8 przedstawiono zależność między f_1 a f_2 , a także granicę Pareto w przestrzeni celów.



Rysunek 15.8. Granica rozwiązań optymalnych w sensie Pareto (linia pogrubiona) w przestrzeni celów

Zwróć uwagę, że w tym przykładzie przestrzeń decyzyjna jest jednowymiarowym przedziałem $[-5, 5]$. Każda wartość z przedziału $[0, 2]$ jest rozwiązaniem optymalnym w sensie Pareto. Przestrzeń celów jest dwuwymiarowa: granica rozwiązań optymalnych w sensie Pareto jest określona wzorem (15.2), przy czym $f_1 \in [0, 4]$. Co odpowiada zakresowi $x \in [0, 2]$.

W celu zilustrowania niektórych omówionych wcześniej zagadnień użyjemy teraz algorytmu MOGA Fonesci i Fleminga [168]. Zostało tutaj zastosowane kodowanie binarne, krzyżowanie dwupunktowe oraz selekcja proporcjonalna. W naszym przykładzie używamy częstości krzyżowania 0,8, częstości mutacji $1/24$ (24 to długość struktury danych) oraz maksymalnej liczby pokoleń równej 80.

Wykres wyników uzyskanych przez MOGA zestawiony z prawdziwą granicą Pareto przedstawiono na rys. 15.9.

Niżej podajemy jeszcze dwa dodatkowe przykłady. Pierwszy z nich również zaproponował Schaffer [411]. Ma on także tylko jedną zmienną, tym razem jednak linia rozwiązań optymalnych w sensie Pareto jest niespójna.

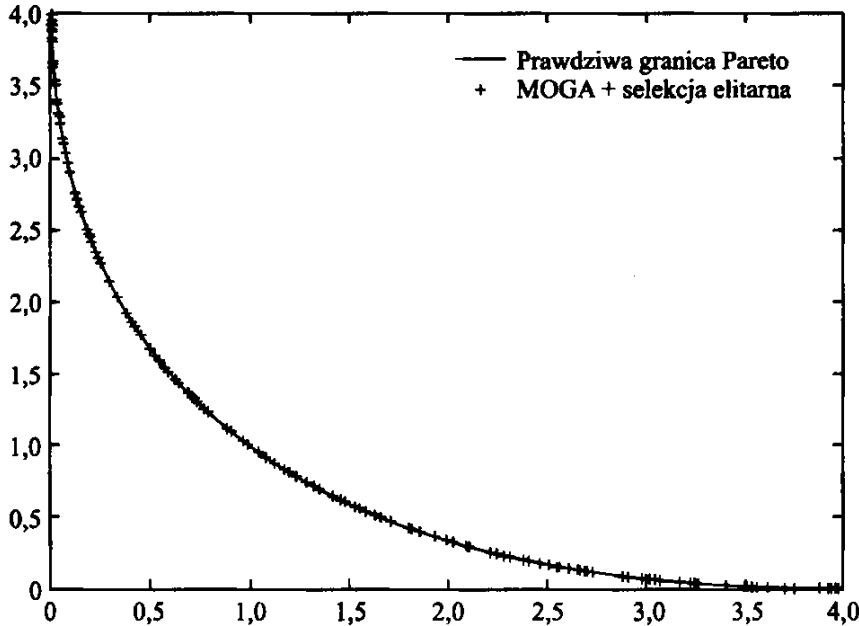
zminimalizuj $f_i(x)$, $i = 1, 2$

przy $-5 \leq x \leq 5$

przy czym:

$$f_1(x) = \begin{cases} -x & \text{jeśli } x \leq 1 \\ -2 + x & \text{jeśli } 1 < x \leq 3 \\ 4 - x & \text{jeśli } 3 < x \leq 4 \\ -4 + x & \text{jeśli } 4 < x \end{cases}$$

$$f_2(x) = (x - 5)^2$$



Rysunek 15.9. Wykres przedstawiający granicę rozwiązań optymalnych w sensie Pareto wygenerowaną przez MOGA (zaznaczona krzyzykami) oraz faktyczną granicę (zaznaczona linią ciągłą) rozwiązań optymalnych w sensie Pareto dla problemu Schaffera

Drugi przykład zaproponował Valenzuela-Rendó [475]. Tym razem mamy dwie zmienne. Prawdziwa granica Pareto jest spójna i wypukła. Zadanie jest sformułowane następująco:

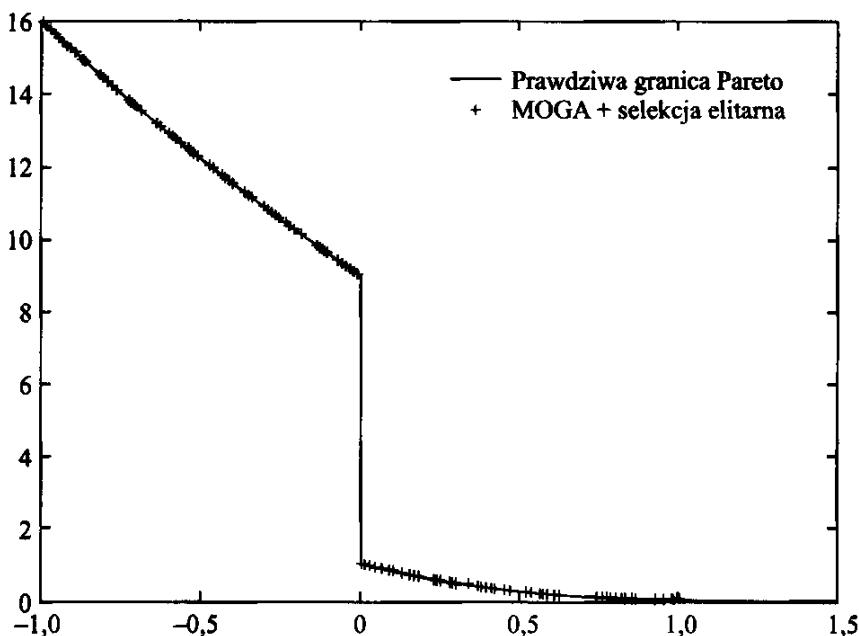
$$\begin{aligned} & \text{zminimalizuj } f_i(x_1, x_2), \quad i = 1, 2 \\ & \text{przy } -3 \leq x_i \leq 3, \quad i = 1, 2 \end{aligned}$$

przy czym:

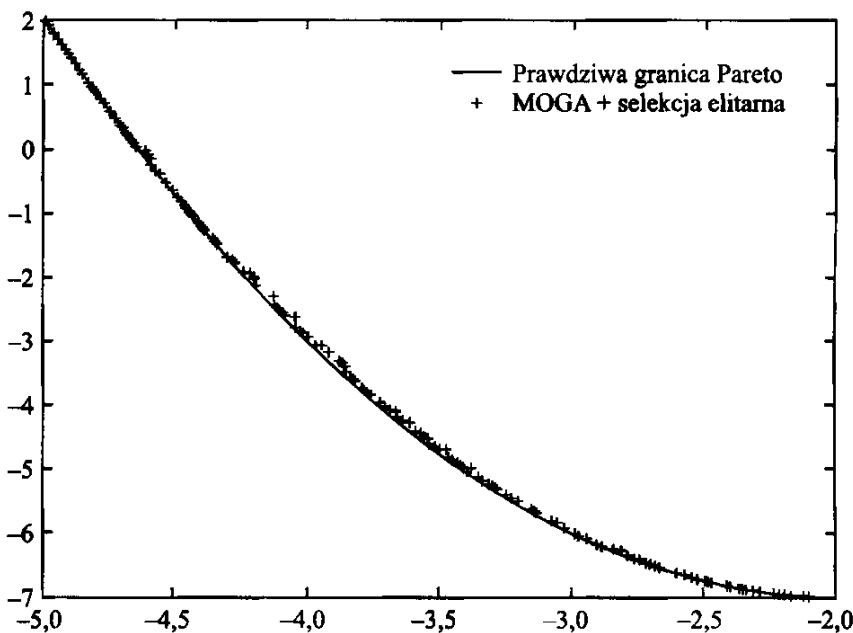
$$\begin{aligned} f_1(x_1, x_2) &= x_1 + x_2 + 1 \\ f_2(x_1, x_2) &= x_1^2 + 2x_2 - 1 \end{aligned}$$

W obydwu przypadkach w MOGA użyto tych samych parametrów co w pierwszym zadaniu. Wykresy wyników uzyskanych przez MOGA porównane z prawdziwymi granicami Pareto dla drugiego zadania Schaffera oraz zadania Valenzueli-Rendóna są pokazane odpowiednio na rys. 15.10 i 15.11.

Te trzy zadania testowe, a także wiele innych, zostały omówione w pracy Coello i in. [75]. Wyniki pokazane na rys. 15.9, 15.10 i 15.11 świadczą o tym, że dla tych trzech zadań testowych MOGA zadziałała „dość dobrze”. Jednakże przy ocenie algorytmów rozwiązujących problemy optymalizacji wielokryterialnej lub przy porównywaniu takich algorytmów takie wizualne sprawdzenie lub ogólne obserwacje w stylu „dość dobrze” nie są w pełni satysfakcjonujące. Należały tu wprowadzić bardziej formalne metody, w których uwzględniano by metryki jakości, bliskość rozwiązań znajdujących się na granicy rozwiązań optymalnych w sensie Pareto oraz różnorodność rozwiązań niezdominowanych. Pełne omówienie wielu takich metryk znajduje się w rozdziale 4 w [75] oraz w podrozdziale 8.2 w [98].



Rysunek 15.10. Wykres przedstawiający granicę rozwiązań optymalnych w sensie Pareto wygenerowany przez MOGA (zaznaczona krzyżykami) oraz faktyczną granicę (zaznaczona linią ciągłą) rozwiązań optymalnych w sensie Pareto dla drugiego problemu Schaffera. Zwróć uwagę, że pionowa linia nie jest częścią faktycznej granicy rozwiązań optymalnych w sensie Pareto i została zamieszczona tylko dla ilustracji



Rysunek 15.11. Wykres przedstawiający granicę rozwiązań optymalnych w sensie Pareto wygenerowany przez MOGA (zaznaczona krzyżykami) oraz faktyczną granicę (zaznaczona linią ciągłą) rozwiązań optymalnych w sensie Pareto dla problemu Valenzueli-Rendóna

15.3. Podsumowanie

Kiedy stykamy się z problemem ze świata rzeczywistego, wtedy prawie zawsze mamy do czynienia z wieloma kwestiami. Metody przedstawione w tym rozdziale mogą pomóc Ci w radzeniu sobie z nimi przy użyciu co najmniej dwóch

podejście. Przy pierwszym z nich wkład związany z różnymi celami jest łączony w jedną funkcję oceny. Przy drugim parametry są traktowane bardziej niezależnie, w wyniku czego otrzymujemy zbiór rozwiązań, które są potencjalnie optymalne w tym sensie, że nie są zdominowane. Koniec końców i tak prawie zawsze będziesz musiał wybrać do wdrożenia jedno rozwiązanie, sposób dojścia do niego może się jednak znacznie różnić w zależności od zastosowanej konkretnej metody. W zależności od metody może się też różnić uzyskane rozwiązanie. Ani podejście z jedną łączoną oceną nie jest istotnie lepsze, ani podejście z użyciem optymalności w sensie Pareto. W żargonie tego rozdziału – żadne z nich nie „dominuje” nad drugim.

Przy stosowaniu podejścia Pareto do problemów z ograniczeniami, zadanie nie polega tylko na znalezieniu granicy rozwiązań optymalnych w sensie Pareto, ale te rozwiązania optymalne w sensie Pareto powinny być też *dopuszczalne*, tzn. powinny spełniać podane ograniczenia. W rozdziale 9 przedstawiliśmy wiele metod radzenia sobie z ograniczeniami. Metody te możemy zaadaptować do problemów optymalizacji wielokryterialnej. Niektóre z tych metod są proste – należy odrzucać rozwiązania niedopuszczalne, stosować funkcje kary, przy porównywaniu dwóch rozwiązań niedopuszczalnych oceniać „bliskość” do granicy obszaru rozwiązań dopuszcjalnych. Inne zaś możemy rozszerzyć, dołączając dodatkowe elementy, na przykład używanie liczności niszy do porównywania rozwiązań niedopuszczalnych. Ray i in. [381], pracując nad bardziej rozbudowanym przykładem, użyli trzech różnych metod oceniania rozwiązań opartych na pojęciu dominacji. W tych trzech metodach brano pod uwagę m wartości funkcji oceny (wyłącznie), $l + q$ wartości opisujących naruszanie ograniczeń (problem miał l ograniczeń w postaci nierówności i q ograniczeń równościowych) oraz $m + l + q$ wartości funkcji celu połączonej z opisem naruszeń ograniczeń. Następnie algorytm wybierał rozwiązania z tych trzech zbiorów i wstawiał je do populacji następnego pokolenia, stosując pewien szczególny zestaw reguł preferencji. Obsługa ograniczeń w środowisku wielokryterialnym jest bardzo ważnym i aktualnym polem badań naukowych.

Istnieje wiele innych otwartych interesujących problemów związanych z wielokryterialnymi problemami optymalizacyjnymi. Należą do nich opracowanie sposobu organizacji macierzy jakości wyników oraz zadań testowych, opracowanie metody wizualizacji, umożliwiającej przedstawianie rozwiązań niezdominowanych oraz dokonanie porównania wielu algorytmów wielokryterialnej optymalizacji. Interesujące problemy badawcze pojawiają się też, jeśli chodzi o szczegóły rozwiązań programistycznych, na przykład sterowanie selekcją elitarną, sterowanie różnorodnością w przestrzeni decyzyjnej oraz przestrzeni celów, a także obsługa ograniczeń. Najlepsze istniejące obecnie pozycje zawierające możliwie najszersze omówienie ewolucyjnych metod wielokryterialnej optymalizacji to [75, 98]. Można też znaleźć istotne wyniki w sprawozdaniach z ostatnich konferencji [508, 373, 278, 374, 62, 171].

Jeśli masz do czynienia z problemem o wielu celach, to niezależnie od tego, czy obierzesz metodę łączenia celów, czy metodę rozwiązań optymalnych w sensie Pareto, masz przed sobą zadanie wymagające myślenia. Upewnij się, że stosując podejście z łączeniem celów, poświęcileś wystarczająco dużo uwagi

istotności względnej parametrów oraz stopniowi ich krytyczności i bądź skrupulatny w zapewnianiu różnorodności osobników w populacji, jeśli chcesz znaleźć cały zbiór rozwiązań wzduż granicy Pareto, zanim skupisz się na jednym lub większej liczbie kandydatów. Być może, najważniejsze jest, żeby mieć pewność, że umie się uzasadnić wybrane rozwiązanie. Większość osób niemających styczności z matematyką, ekonomią i inżynierią nigdy nie słyszała o Pareto, ale często, widząc rozwiązanie wiedzą, że jest ono dobre. Mamy nadzieję, że Ty też to wiesz. Umiejętność wyjaśnienia rozwiązania klientowi może często decydować o różnicy między pomyślnym jego użyciem a „ciekawym wynikiem”.

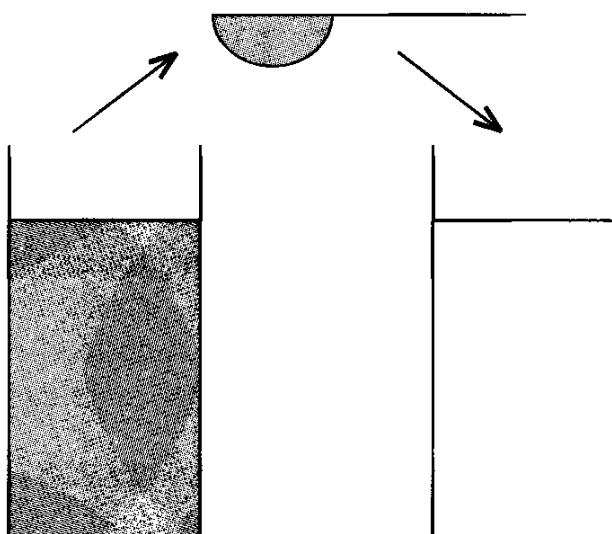
XVI. Czy lubisz proste rozwiązania?

Wszyscy mamy skłonność do komplikowania spraw, zwłaszcza że, będąc głównie naukowcami i inżynierami, operujemy równaniami matematycznymi. Kusi nas opisywanie świata językiem matematycznym, nawet gdy nie ma takiej potrzeby. Często przekonasz się, że największą przeszkodą w rozwiązywaniu problemów jest nadmierne komplikowanie rozwiązania. Wyświetl sobie przysługę i *pomyśl*, zanim zapiszesz jakieś równanie. Zilustrujmy to zagadnienie kilkoma przykładami.

Masz przed sobą szklankę wody i szklankę soku. Ich objętości są jednakowe. Bierzesz łyżeczkę, zanurzasz ją w szklance z sokiem, nabierasz pełną łyżeczkę soku i przelewasz go do szklanki z wodą (rys. XVI.1). Po krótkim mieszaniu nabierasz pełną łyżeczkę z tej szklanki i przelewasz do szklanki z sokiem. Pytanie brzmi: czego jest więcej – czy soku w wodzie, czy wody w soku?

Możesz zacząć rozwiązywać ten problem, obliczając wymagane objętości płynów. Możesz przyjąć, że objętość wody lub soku w szklance wynosi V , a objętość łyżeczki oznaczyć x . Następnie mógłbyś obliczyć stosunek wody do soku po dodaniu łyżeczki soku do wody. Potem mógłbyś obliczyć ilość wody przelanej do szklanki z sokiem itd. Metoda ta ma jednak kilka wad. Po pierwsze, wymaga obliczeń. Po drugie, jest oparta na założeniu istnienia mieszaniny doskonałej (tzn. założeniu, że procentowy udział soku w wodzie jest taki sam w łyżeczce i w szklance), które nie musi być prawdziwe. Mogliśmy przecież zanurzyć łyżeczkę płytko pod powierzchnię, gdzie udział soku w wodzie jest z reguły mniejszy niż bliżej dna szklanki.

Problem możemy jednak rozwiązać bez rozważania proporcji wody i soku. Pomyśl o szklance wody. Wlewamy do niej łyżeczkę soku. Następnie odejmujemy z niej trochę wody i trochę soku. Ile płynu pozostało w szklance? Dokładnie tyle samo, ile było na początku. Nie więcej i nie mniej. To samo dzieje



Rysunek XVI.1. Szklanka soku i szklanka wody. Pełna łyżeczka soku wędruje do szklanki z wodą

się w przypadku szklanki z sokiem. Ujęliśmy z niej łyżeczkę soku i zastąpiliśmy ją łyżeczką mieszaniny wody z sokiem. Po wykonaniu tych operacji płynu jest w niej jednak tyle samo.

Tak więc, bez wyliczania żadnych proporcji

$$W + j_1 - j_2 - w_1 = W \text{ oraz } J - j_1 + j_2 + w_1 = J$$

przy czym: W i J oznaczają, odpowiednio, objętości wody i soku w szklankach (i są sobie równe), j_1 oznacza ilość soku w łyżeczce podczas pierwszego przelewania (rys. XVI.1), a j_2 i w_1 oznaczają, odpowiednio, ilości soku i wody w łyżeczce podczas drugiego przelewania. Zastanów się nad tym. Wielkość $j_1 - j_2$ to ilość soku w wodzie, a w_1 – ilość wody w soku. Z każdego z podanych wyżej równań wynika, że $j_1 - j_2$ musi być równe w_1 , tak więc ilość wody w soku jest dokładnie taka sama jak ilość soku w wodzie!

Możemy nawet rozważyć rozwiązanie tego problemu bez użycia równań. Musimy tylko uświadomić sobie, że podczas drugiego przelewania ilość wody w łyżeczce musi być dokładnie taka sama jak pozostawiona przed chwilą w szklance wody ilość soku, ponieważ nalaliśmy do tej szklanki łyżeczkę soku (rys. XVI.2). Co więcej, możemy przelać na przemian dowolną liczbę łyżeczek ze szklanki do szklanki. Dopóki objętości płynów w obu szklankach będą takie same, dopóty ilość wody w soku będzie zawsze dokładnie taka sama jak ilość soku w wodzie.

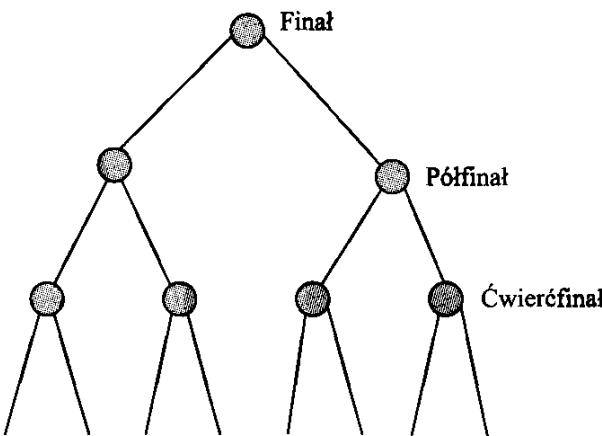


Rysunek XVI.2. łyżeczka podczas drugiego przelewania z odrobiną soku

Następny przykład jest również bardzo pouczający. Rozgrywany jest turniej tenisowy, w którym bierze udział 937 graczy. Zawodnik, który wygra mecz, przechodzi dalej, przegrany zaś odpada z gry. Ile meczów trzeba rozegrać, żeby zakończyć turniej?

Tym razem także możemy dość łatwo obliczyć rozwiązanie. Możemy wyobrazić sobie ten turniej jako drzewo binarne, którego każdy węzeł oznacza mecz. Jedna rozgrywka jest w korzeniu (mecz finałowy), dwie na kolejnym poziomie (półfinały), cztery o poziom niżej (ćwierćfinały) itd. (rys. XVI.3). Ponieważ zawodnik może być wylosowany tylko w pierwszej kolejce, 512 graczy musi przejść do drugiej kolejki. W pierwszej kolejce mamy więc $937 - 512 = 425$ meczów, w których bierze udział 850 graczy. Zwycięzcy tych meczów (425 zawodników) wraz z wylosowanymi szczęśliwcami ($937 - 850 = 87$) tworzą grupę 512 graczy, którzy przechodzą do drugiej kolejki. Ponieważ $512 = 2^9$, pozostałe obliczenia są już bardzo proste. Całkowita liczba meczów podczas turnieju wynosi

$$425 + 256 + 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 936$$



Rysunek XVI.3. Turniej tenisowy

Moglibyśmy jednak w o wiele krótszym czasie uzyskać ten sam wynik, nie robiąc żadnych obliczeń. Wszystko, co nam potrzebne, to wiedza, że każdy mecz w turnieju eliminuje jednego gracza. W turnieju z n graczami, żeby wyłonić zwycięzcę, musimy wyeliminować $n - 1$ spośród nich, a więc musimy rozegrać $n - 1$ meczów. Zatem przy 937 gracach potrzebujemy 936 meczów. *Game, set, and match!*

Trzeci przykład dotyczy zegara i jego wskazówek. O północy wskazówka minutowa zegara pokrywa się ze wskazówką godzinową. O której godzinie wskazówki znów nałożą się na siebie?

Łatwo jest to sobie wyobrazić. W miarę jak wskazówka minutowa przesuwa się wokół tarczy zegara, wskazówka godzinowa też się porusza, ale o wiele wolniej. O godzinie pierwszej wskazówka minutowa jest na 12, a wskazówka godzinowa jest na 1. Pięć minut później wskazówka minutowa, wskazując na godzinę 1, prawie przykrywa wskazówkę godzinową, ale wskazówka godzinowa w ciągu tych pięciu minut też się trochę przesunęła, przykrywanie nie jest więc dokładne.

Przypuśćmy, że prędkość wskazówki godzinowej wynosi v cm/h. Wówczas prędkość wskazówki minutowej wynosi $12v$ cm/h, a odległość pokonana przez nią w trakcie jednego obrotu (tzn. jednej godziny) wynosi $12v$ cm. Po jednej godzinie, czyli o godzinie 1, (szybsza) wskazówka minutowa znajduje się pięć

minut za (wolniejszą) wskazówką godzinową (odległość ta wynosi v cm). Po czasie t wskazówka minutowa dogania wskazówkę godzinową:

$$t \cdot 12v = v + t \cdot v$$

gdziś odległość pokonana przez wskazówkę minutową (iloczyn czasu t i prędkości $12v$) jest o v cm większa od odległości pokonanej przez wskazówkę godzinową (jest ona, znowu, iloczynem czasu t i prędkości v). Powyższe równanie daje $t = 1/11$, a zatem wskazówki nachodzą na siebie co $1\frac{1}{11}$ godziny.

Ponownie, do tej odpowiedzi moglibyśmy dojść znacznie szybciej, gdybyśmy przypomnieli sobie, że wskazówki nachodzą na siebie 11 razy w ciągu 12 godzin. Ponieważ nachodzą na siebie w stałych odstępach czasu, musi to następować co $1\frac{1}{11}$ godziny.

Obserwacja ta umożliwia nam rozwiązywanie następującego aktualnienia. Wskazówki sekundowa, minutowa i godzinowa pokrywają się o północy. O której godzinie wskazówki te nałożą się na siebie następny raz?

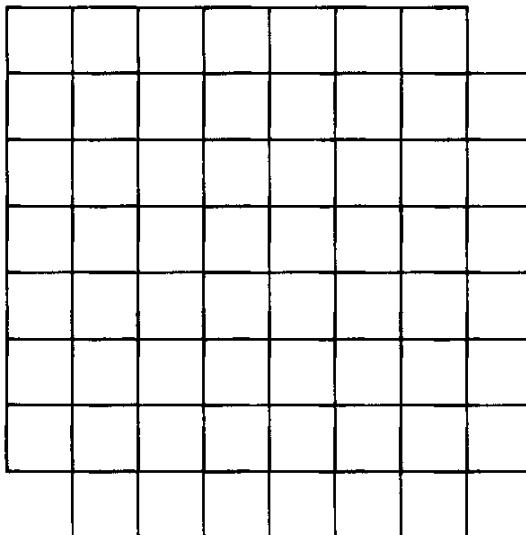
Oto klasyczny przykład zadania, które można rozwiązać albo długą metodą, albo krótką. Mamy dwa miasta A i B , które są od siebie odległe o 400 km. Z miasta A wyrusza pociąg i jedzie w kierunku miasta B ze stałą prędkością 40 km/h. Jednocześnie z miasta B wyrusza drugi pociąg, który jedzie w kierunku miasta A ze stałą prędkością 60 km/h. W tym samym czasie pracowita pszczoła, która odpoczywała sobie na przodzie pierwszego pociągu, zaczyna bardzo ciekawą podróż. Ze stałą prędkością 75 km/h leci w kierunku drugiego pociągu i gdy tylko do niego dociera, zmienia kierunek lotu na przeciwny. Pszczoła robi tak za każdym razem, gdy dotrze do któregoś z pociągów. Jaka odległość pokona pszczoła do chwili, gdy oba pociągi się spotkają?

W pierwszej chwili wygląda na to, że zanosi się na niezłe rachunki! Możemy określić długość każdego odcinka, jaki przebyła pszczoła, gdzie odcinek odpowiada odległości pokonanej przy locie w jednym kierunku, a następnie obliczyć nieskończoną sumę długości tych zmniejszających się odcinków. Wiemy jednak, że istnieje lepszy sposób obliczenia tego!

Pociągi spotkają się po czterech godzinach, gdyż oba pokonują $40 + 60 = 100$ kilometrów co godzinę, a odległość między miastami wynosi 400 km. Z kolei pszczoła leci przez te cztery godziny bez przerwy ze stałą prędkością 75 km/h, zatem całkowita odległość pokonana przez pszczołę wynosi 300 km; c.b.d.o.

A oto jeszcze jedna zagadka dla Ciebie. Mamy zwykłą szachownicę z usuniętymi dwoma narożnymi polami po przekątnej (rys. XVI.4). Mamy też 31 kostek domina. Każda prostokątna kostka domina pokrywa dwa pola szachownicy. Zadanie polega na takim rozmieszczeniu kostek, poziomo lub pionowo, żeby pokryły całą szachownicę.

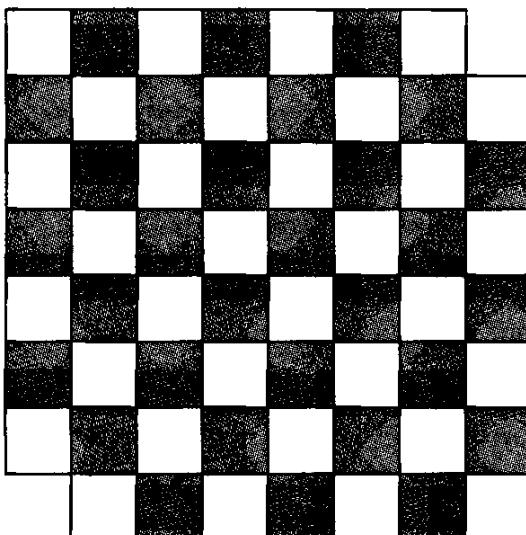
Można godzinami dopasowywać kostki domina do szachownicy, ale prosta obserwacja umożliwia bardzo szybkie rozwiązywanie tej zagadki. Zauważ, że jeśli pokolorujemy pola szachownicy na czarno i biało zgodnie z tradycyjnym



Rysunek XVI.4. Szachownica z usuniętymi dwoma narożnymi polami oraz jedna kostka domina

wzorem, to otrzymamy 30 pól w jednym kolorze i 32 w drugim. Pola w przeciwnieństwach rogach muszą być tego samego koloru (rys. XVI.5).

Ponieważ każda kostka domina zawsze pokrywa jedno pole białe i jedno czarne, niezależnie od ich rozmieszczenia na planszy, niemożliwe jest pokrycie jej takimi kostkami!

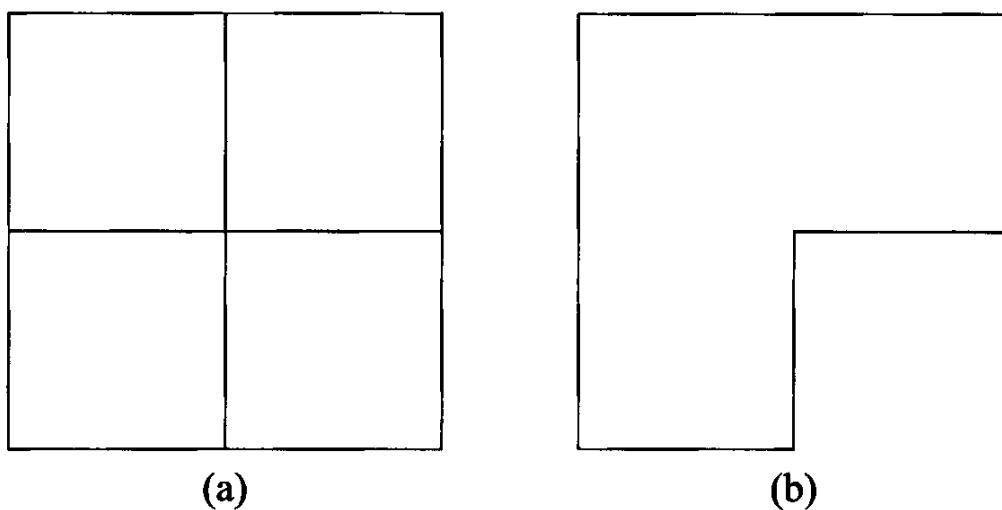


Rysunek XVI.5. Szachownica z usuniętymi dwoma narożnymi polami oraz jedna kostka domina. Dodano białe i czarne kolory pól

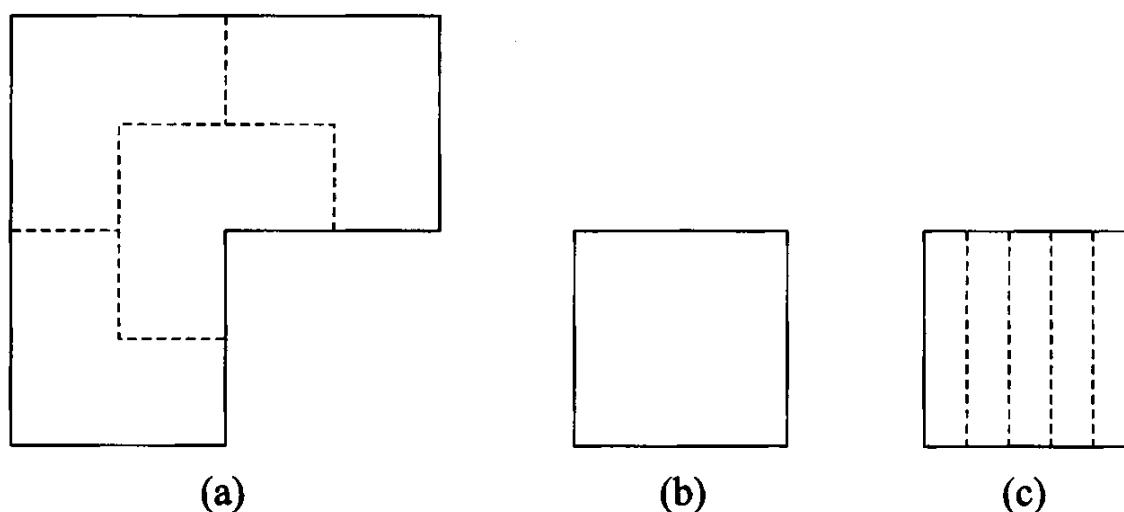
Zadanie to występuje też w wersji trójwymiarowej. Chodzi tym razem o wypełnienie cegielkami pudełka o wymiarach $6 \times 6 \times 6$. Każda z cegiełek ma wymiary $1 \times 2 \times 4$. Czy potrafiłbyś to zrobić?

Ostatnia zagadka jest najbardziej zadziwiająca z dotychczas przedstawionych. (Wypróbuj ją na przyjacieliu!) Mamy kwadrat podzielony na cztery jednakowe mniejsze kwadraty (rys. XVI.6a).

Jeden z czterech mniejszych kwadratów zostaje usunięty. Twoje zadanie polega na podzieleniu pozostały części kwadratu (rys. XVI.6b) na cztery jednakowe części. Możesz używać ciągłych linii dowolnego kształtu. Zabiera



Rysunek XVI.6. Kwadrat podzielony na cztery jednakowe kwadraty (a); jeden z czterech mniejszych kwadratów jest usunięty (b)



Rysunek XVI.7. Wymagany podział kwadratu z usuniętą ćwiartką (a); usunięta ćwiartka (b); wymagany podział w drugiej części zagadki (c)

to trochę czasu, ale przedzej czy później większość ludzi znajduje rozwiązanie (rys. XVI.7a).

Jest to jednak dopiero pierwsza część całej zagadki. Nastawia ona umysł rozwiązującego na określony sposób myślenia. Druga (i w rzeczywistości główna) część zagadki polega na podzieleniu pozostałego kwadratu (rys. XVI.7b) na pięć jednakowych kawałków. Chociaż ta część zagadki jest *niesłychanie* prosta (zob. rys. XVI.7c), to znalezienie jej rozwiązania wymaga od niektórych ludzi włożenia w to znacznego wysiłku¹.

To samo możemy obserwować na szerszą skalę. Nasze doświadczenia z przeszłości nie zawsze muszą wskazywać dobry kierunek! Ważne jest, żeby zachować pewną elastyczność, umożliwiającą rozwiązywanie nowych zadań nowymi sposobami! Zdolność ta świadczy o prawdziwej inteligencji.

¹Pewien profesor matematyki, któremu daliśmy to zadanie do rozwiązania, „udowodnił” nawet, że takiego podziału nie można wykonać!

16. Systemy hybrydowe

„Nadszedł już czas – powiedział Mors –
Pograżyć się w rozmowie
O tym, czym but jest, łódź i lak, Kapusta i Królowie,
I czemu w morzu woda wrze,
A wieprz ma skrzydła sowie.”

Lewis Carroll: *O tym, co Alicja odkryła po drugiej stronie lustra*¹

Wiemy, że nie istnieje jeden algorytm, który byłby najlepszy do rozwiązywania wszystkich problemów. Do algorytmu musimy w jakiś sensowny sposób włączyć wiedzę na temat problemu, którym się zajmujemy. W przeciwnym razie algorytm ten nie będzie wiele lepszy od poszukiwania losowego. Jednym ze sposobów radzenia sobie z tym zagadnieniem jest hybrydyzacja algorytmu ewolucyjnego z bardziej standardowymi procedurami, takimi jak wspinanie się czy metody zachłanne. Poszczególne rozwiązania można ulepszać, stosując metody lokalne, a następnie przywracać je do współzawodniczenia z innymi członkami populacji. Początkowa populacja może być wyodrębniana za pomocą rozwiązań znalezionych metodami klasycznymi. Istnieje wiele sposobów tworzenia metod hybrydowych, które łączą algorytmy ewolucyjne z innymi procedurami.

W rzeczywistości istnieje tak wiele sposobów hybrydyzacji, że częstym zjawiskiem jest przeładowywanie systemów hybrydowych zbyt dużą liczbą składników. Niektóre nowoczesne systemy hybrydowe zawierają składniki rozmyto-neuronowo-ewolucyjne i pewne inne ściśle związane z problemem heurystyki. Na wielu konferencjach na temat inteligencji obliczeniowej często słyszy się żart polegający na pytaniu, kiedy ukaże się pierwsza praca pod tytułem „samoadaptacyjne zachłanne neuronowo-rozmyto-ewolucyjno-wyżarzeniowe podejście do ulepszonych zasad pamiętania w poszukiwaniu z tabu na drzewie definiującym zbiór reguł generujących systemy eksperckie w eksploracji danych”. Możemy oczywiście wymyślić dowolną liczbę innych możliwości. Sztuka polega tutaj jednak na tym, aby jak najskuteczniej określić synergię dwóch różnych podejść i zadbać o to, żeby to ogólne było jak najbardziej oszczędne.

¹Przekład Maciej Słomczyński, wydawnictwo Zielona Sowa, Kraków 2002 (przyp. tłum.).

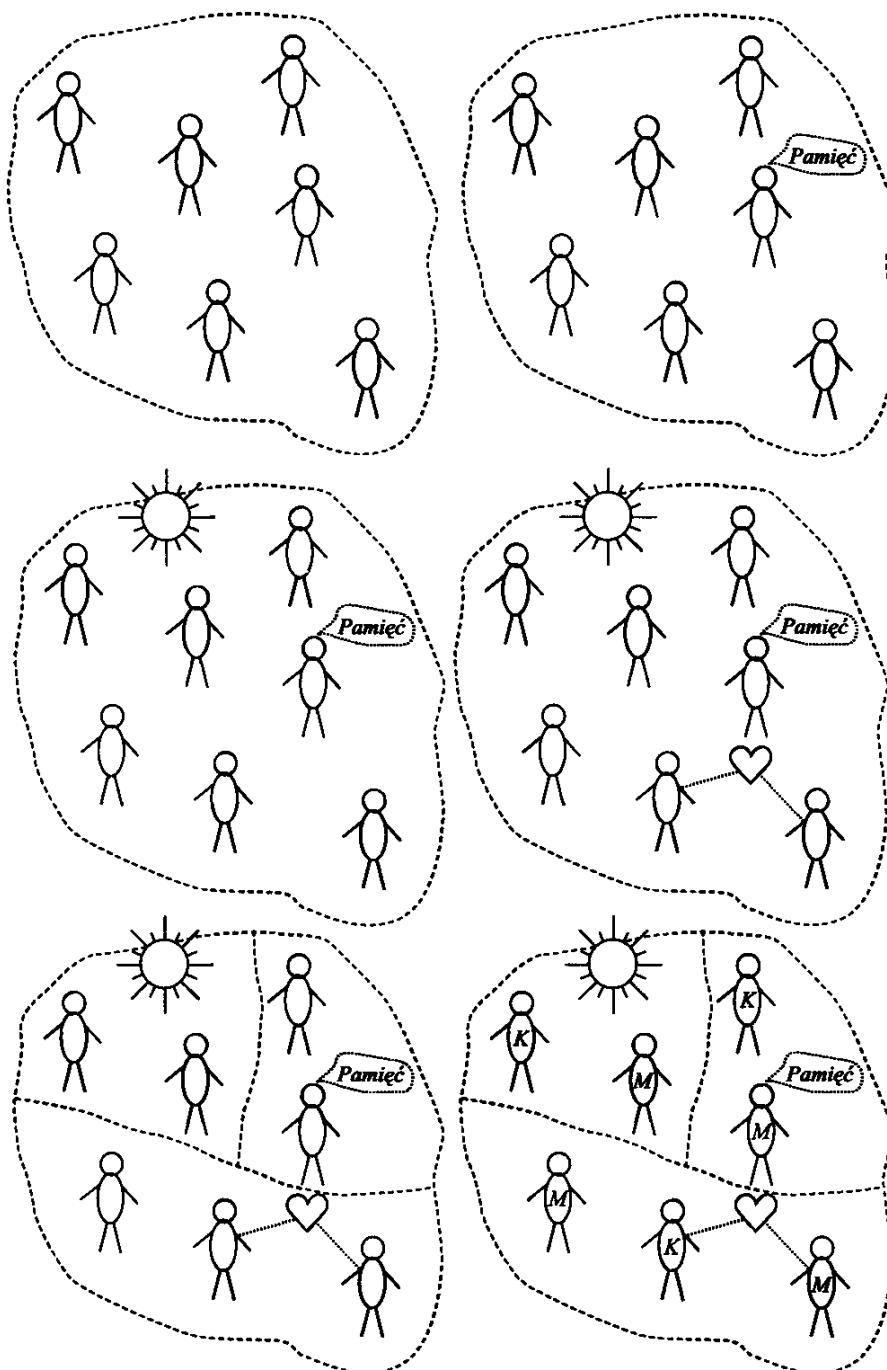
Algorytmy ewolucyjne są niezwykle elastyczne i można je rozszerzać, włączając do nich różnorodne pomysły i alternatywne podejścia. Rozważaliśmy już, na przykład, możliwość włączenia poszukiwania lokalnego do algorytmu ewolucyjnego w przypadku TSP (rozdział 8). Przedstawiliśmy także różne metody radzenia sobie z ograniczeniami, do których należało lamarkowskie uczenie się oraz efekt Baldwina (rozdział 7). Omówiliśmy też niektóre sposoby dołączania w algorytmach ewolucyjnych parametrów sterujących do osobników, tak żeby osobniki te mogły się same dostosowywać do rozwiązywanego problemu (rozdział 10). Algorytmy ewolucyjne możemy też rozszerzać, dokładając do osobników lub całych populacji pamięć, żeby lepiej obsługiwały przypadek środowisk ze zmieniającym się czasem (rozdział 11). Ta garść przykładów to zaledwie początek.

Przyjrzyjmy się zestawowi sześciu obrazków (rys. 16.1). Zaczniemy od obrazka w lewym górnym rogu, gdzie przedstawiono populację osobników, z jaką zwykle mamy do czynienia w algorytmie ewolucyjnym. Przechodząc do kolejnego obrazka w prawym górnym rogu, rozszerzamy każdego osobnika, dodając mu strukturę pamięciową (jawnie lub niejawnie). Jest to krok prowadzący do poszukiwania z tabu. Następnie na obrazku w środku po lewej możemy wprowadzić pojęcie temperatury, przy którym populacja jest stopniowo schładzana, jak w wypadku symulowanego wyżarzania. Możemy teraz przyjąć, że zakres operatora różnicowania zależy od temperatury otoczenia osobnika, przy czym wyższe temperatury oznaczają większe kroki różnicowania.

Teraz jesteśmy gotowi na dalsze rozszerzenia. Do systemu możemy wprowadzić pojęcie *atrakcyjności*, które wpływa na sposób, w jaki osobniki kojarzą się w pary i generują rozwiązania potomne (obrazek w środku po prawej). Możemy teraz nawet mówić o uwodzeniu [394]! Możemy też rozważać różne sposoby doboru osobników w pary (np. „twój rozum, moja piękność” [222]), ile osobników powinno brać udział w tworzeniu potomstwa (np. rozważane w [117] „orgie”) lub ilu potomków powinno być tworzonych [132]. Następnie możemy podzielić populację na kilka podpopulacji (obrazek na dole po lewej) i uruchomić algorytm ewolucyjny w trybie rozproszonym. Wreszcie możemy osobnikom przypisać płeć (M = mężczyźni, K = kobiety na obrazku na dole po prawej) oraz relacje rodzinne, a nawet reguły zabraniające łączenia się w pary osobnikom z tej samej rodziny [129]. Możemy nawet rozważyć możliwość występowania w populacji osobników, które nie śpią lub śpią [128]!

Jak można się było spodziewać, wiele z tych rozszerzeń daje początek innym możliwościom. Możemy rozbić populację na wiele podpopulacji, co prowadzi do modeli *wyspowych*, *migracyjnych* lub *gruboziarnistych* [300]. Moglibyśmy dla każdej podpopulacji używać innego algorytmu ewolucyjnego z różnicowaniem i selekcją ograniczonymi do podpopulacji. Od czasu do czasu jeden lub więcej osobników dokonuje migracji z jednej podpopulacji do drugiej. W ten sposób możemy dzielić informacje między podpopulacjami. Taka architektura nasuwa wiele interesujących pytań.

- Ile podpopulacji powinniśmy rozważać i jak duże powinny one być?
- Czy wszystkie podpopulacje powinny mieć ten sam rozmiar?



Rysunek 16.1. Stopniowe rozszerzanie algorytmu ewolucyjnego. Zaczynamy od „zwykłego” algorytmu ewolucyjnego i kolejno dodajemy pamięć, temperaturę, kojarzenie par i atrakcyjność, podpopulacje i płeć

- Czy w każdej podpopulacji powinny być używane te same parametry algorytmu ewolucyjnego, czy też może należy każdy algorytm ewolucyjny jakoś dostosowywać do każdej podpopulacji?
- Jakich rozwiązań topologicznych powinniśmy użyć, żeby połączyć te podpopulacje? Czy osobnik z dowolnej podpopulacji może bezpośrednio migrować do dowolnej innej?
- Jakie są mechanizmy migracji:
 - Jak często osobniki migrują? Czy powinno to zachodzić okresowo, czy też powinno być uruchamiane przez jakieś zdarzenie, na przykład zmniejszona różnorodność populacji?

- Jakie osobniki powinny migrować? Najlepsze? Najgorsze? Typowe? Losowo wybrane?
- Jakie są zasady migracji? Jakie są konsekwencje tego, że osobnik x przechodzi z podpopulacji S_j do podpopulacji S_k ? Czy kopia x pozostaje w S_j ? Czy osobnik ten wyrzuca jakiegoś osobnika z S_k ?

Możemy też pomyśleć o zorganizowaniu populacji w ten sposób, że osobnikom są przypisane konkretne położenia geograficzne (model ten nazywa się modelem *dyfuzjnym*, z *sąsiedztwem* lub też modelem *drobnoziarnistym*, zob. rys. 16.2).

•	•	•	•	•	•
•	•	•	•	•	•
•	•	•	•	•	•
•	•	•	•	•	•
•	•	•	•	•	•
•	•	•	•	•	•

Rysunek 16.2. Topologia siatki w klasycznym modelu z dyfuzją

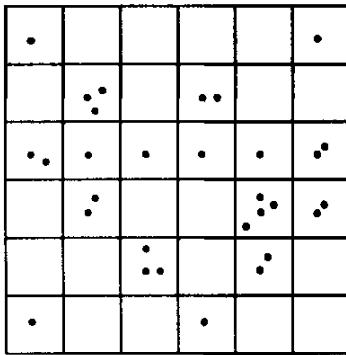
W tego rodzaju modelu w jednej populacji są utrzymywane zachodzące na siebie sąsiedztwa. Każdy osobnik ma lokalne sąsiedztwo, które jest stosunkowo małe, ograniczone i określone za pomocą topologii sąsiedztwa. Selekcja, różnicowanie i zastępowanie są ograniczone do tego sąsiedztwa. Zachodzenie na siebie sąsiedztw wpływa na rozprowadzanie (dyfuzję) informacji po całej populacji.

Za oboma tymi podejściami kryje się obserwacja, że z punktu widzenia genetyki populacji różnicowanie i selekcja zwykle nie zachodzą globalnie w populacji. Zamiast tego są one zwykle ograniczone do *demów*: lokalnych podzbiorów krzyżujących się ze sobą osobników. Przy zastosowaniu modeli z zestrukturalizowaną populacją cechy genetyczne osobników mogą rozprzestrzeniać się w populacji w sposób podobny do procesu dyfuzji [195].

Jednym z najważniejszych wyborów, przed jakimi stoją projektanci algorytmów ewolucyjnych, jest wybór rozmiaru populacji. W wielu zastosowaniach ta decyzja może być krytyczna. Jeśli populacja jest zbyt mała, to algorytm ewolucyjny może być zbyt szybko zbieżny do rozwiązania. Jeśli jest zbyt duża, to algorytm może niepotrzebnie marnować zasoby obliczeniowe. Rozmiar populacji ma wpływ na dwa ważne czynniki związane z poszukiwaniem ewolucyjnym: różnorodność populacji i nacisk selekcyjny. Zauważ jednak, że zarówno w modelu dyfuzjnym, jak i wyspowym przyjęte jest, że populacja ma ustalony rozmiar. W modelu dyfuzjnym rozmiar populacji jest stały. W modelu wyspowym typowo każda z N podpopulacji ma ustaloną liczbę $\mu = M/N$ osobników. Oczywiście każda podpopulacja mogłaby mieć inny rozmiar μ_i , przy czym $\sum_{i=1}^N \mu_i = M$. Migracja może zwiększać i zmniejszać rozmiary podpopulacji,

utrzymując wartość M na stałym poziomie, ale wartości μ_i są w większości implementacji stałe.

Ostatnio w pracy [274] zaproponowano nowy model zwany modelem *patchworkowym*, który łączy w sobie niektóre własności modelu wyspowego i modelu dyfuzyjnego. Każdy osobnik jest zamodelowany jako agent mobilny, który żyje w wirtualnej niszy ekologicznej i współdziała ze swoim środowiskiem za pomocą swoich czujników i mechanizmów motorycznych. Jego decyzje są oparte na informacji lokalnej zebranej za pomocą czujników, a jego akcje (działania) mają wpływ tylko na jego lokalne środowisko. Osobnikom są też przypisywane inne własności, takie jak maksymalna długość życia, zdolność do rozmnażania się, śmiertelność zależna od ich przystosowania oraz umiejętności uwzględniania preferencji przy podejmowaniu decyzji. W modelu patchworkowym zastosowano strukturę przestrzenną, w której agenci poruszają się i współpracują na obszarze dwuwymiarowej siatki. Każda komórka siatki (zwana „łata”) ma lokalne własności przestrzenne, takie jak maksymalna liczba osobników, które może w sobie pomieścić (rys. 16.3).



Rysunek 16.3. Model patchworkowy

Agenci mobilni są rozszerzani za pomocą zbioru adaptacyjnych reguł behawioralnych. Reguły te opisują wykonywane przez osobniki akcje, związane na przykład z chęcią reprodukcji lub migracji. Każdy osobnik proponuje akcję, którą chce wykonać. Akcja ta może zajść lub nie. Jednoczesne zdarzenia są obsługiwane w dwukrokowym procesie. Najpierw agenci podejmują decyzje i planują kolejność akcji, a następnie symulacja wykonuje zaplanowane akcje i rozwiązuje pojawiające się konflikty. Na przykład, gdy agenci występujący w tej samej komórce siatki zdecydują się na połączenie w parę, ich zamiar jest najpierw planowany, ale nie jest wykonany natychmiast. W tej sytuacji propozycja reprodukcji może skończyć się niepowodzeniem ze względu na brak partnera lub ze względu na brak miejsca na potomka. Nawet jeśli potomek powstanie, to może się okazać, że nie jest wystarczająco konkurencyjny, żeby mógł znaleźć się w populacji. Podobnie, propozycja migracji może skończyć się niepowodzeniem ze względu na propozycje innych osobników i/lub ze względu na zbytnie zatłoczenie. Zauważ, że model patchworkowy umożliwia całemu systemowi osobników dostosowanie się do zmian przez modyfikację odpowiednich rozmiarów lokalnych populacji, reguł decyzyjnych, protokołów migracji itp., przy czym jednocześnie jest rozwiązywany sam problem. Na różnych etapach mogą być potrzebne różne reguły.

Pomysł modelowania ewolucji agentów autonomicznych według zasad ekologii został wprowadzony w dziedzinie obliczeń ewolucyjnych zwanej *sztucznym życiem*. Faktycznie w niektórych z najwcześniejszych prób związanych z symulowaniem ewolucji było wykorzystywane właśnie to podejście [32, 77, 14]. Jednym z powszechnie stosowanych programów sztucznego życia jest program zwany SWARM [323], na który składa się zestaw narzędzi w języku Objective C, służących do modelowania sztucznej ekologii. Ostatnio w ekologii zyskały na znaczeniu tak zwane modele oparte na osobnikach [92, 248], ale w bardzo niewielu podejściach rzeczywiście przedstawia się osobniki jako autonomicznych agentów i naśladuje procesy ewolucyjne [275, 106].

Innym interesującym osiągnięciem w dziedzinie obliczeń ewolucyjnych jest pojawienie się tak zwanych *algorytmów kulturowych* [388]. Procedury te obsługują dwa tryby dziedziczenia: jeden z nich jest stosowany na poziomie mikroewolucyjnym wyrażanym za pomocą cech genetycznych lub behawioralnych, drugi zaś na poziomie makroewolucyjnym wyrażanym za pomocą tak zwanych *przekonań* (ang. *beliefs*) populacji. Te dwa tryby współdziałają ze sobą poprzez kanał komunikacyjny, który umożliwia osobnikom zmianę struktury przekonań populacji, a także wpływa na zachowanie osobnika w zależności od przekonań populacji. A zatem algorytmy kulturowe mają trzy podstawowe składniki: strukturę przekonań, strukturę populacji oraz kanał komunikacyjny [390].

Podstawową strukturę algorytmu kulturowego przedstawiono na rys. 16.4.

```

procedure algorytm kulturowy
begin
     $t \leftarrow 0$ 
    inicjuj populację  $P(t)$ 
    inicjuj sieć przekonań  $B(t)$ 
    inicjuj kanał komunikacyjny
    oceń  $P(t)$ 
    while (not warunek zakończenia) do
        begin
            skomunikuj ze sobą  $P(t)$  i  $B(t)$ 
            skoryguj  $B(t)$ 
            skomunikuj ze sobą  $B(t)$  i  $P(t)$ 
            zmień dopasowanie  $P(t)$ 
             $t \leftarrow t + 1$ 
            wybierz  $P(t)$  z  $P(t - 1)$ 
            przeprowadź ewolucję  $P(t)$ 
            oceń  $P(t)$ 
        end
    end

```

Rysunek 16.4. Struktura algorytmu kulturowego

Algorytmy kulturowe wydają się odpowiednie do korygowania „ewolucji ewolucji”, gdzie osobniki w populacji uczą się podczas trwania ich życia. Rozwiążanie to jest uogólnieniem poglądu, że ewolucja leży u podłożu zarówno uczenia się filogenetycznego gatunków, jak i uczenia się społecznego w społeczeństwach ludzkich [14].

Inna ostatnio zaproponowana metoda to *optymalizacja kolonii mrówek* (ang. *ant colony optimization* – ACO) [103, 104], stanowiąca system wieloagentowy, w którym niskopoziomowe współdziałanie sztucznych (tzn. symulowanych) mrówek daje w wyniku skomplikowane zachowanie większej kolonii mrówek. Inspiracją dla tych algorytmów były kolonie prawdziwych mrówek [103], które znaczą ziemię substancją chemiczną zwaną feromonem. Substancja ta wpływa na zachowanie poszczególnych mrówek. Im więcej feromonu jest na konkretnej ścieżce, tym większe jest prawdopodobieństwo, że mrówka wybierze tę ścieżkę. Sztuczne mrówki w algorytmach ACO zachowują się podobnie.

Podstawowe koncepcje wprowadzonego przez Dorigo i in. [105] algorytmu ACO zilustrujemy na przykładzie TSP. W problemie tym minimalizujemy

$$COST(i_1, \dots, i_n) = \sum_{j=1}^{n-1} d(C_{i_j}, C_{i_{j+1}}) + d(C_{i_n}, C_{i_1}) \quad (16.1)$$

przy czym $d(C_x, C_y)$ oznacza odległość między miastami x i y .

Niech $b_i(t)$ ($i = 1, \dots, n$) będzie liczbą mrówek w mieście i w chwili t i niech $m = \sum_{i=1}^n b_i(t)$ będzie całkowitą liczbą mrówek. Niech $\tau_{ij}(t+n)$ będzie intensywnością śladu feromonowego na ścieżce łączącej (i, j) w chwili $t+n$:

$$\tau_{ij}(t+n) = (1 - \rho)\tau_{ij}(t) + \Delta\tau_{ij}(t, t+n) \quad (16.2)$$

przy czym $0 < \rho \leq 1$ jest współczynnikiem reprezentującym odparowywanie feromonu. Określamy jeszcze $\Delta\tau_{ij}(t, t+n) = \sum_{k=1}^m \Delta\tau_{ij}^k(t, t+n)$, przy czym $\Delta\tau_{ij}^k(t, t+n)$ jest ilością substancji (feromonu z prawdziwej mrówkiej) na jednostkę długości ścieżki zostawianej przez k -tą mrówkę w chwili $t+n$ na trasie (i, j) ; wyraża się ją wzorem

$$\Delta\tau_{ij}^k(t+n) = \begin{cases} \frac{Q}{L_k} & \text{jeśli } k\text{-ta mrówka używa krawędzi } (i, j) \text{ na} \\ & \text{swojej trasie} \\ 0 & \text{w przeciwnym wypadku} \end{cases}$$

przy czym: Q jest stałą, a L_k jest długością trasy znalezionej przez k -tą mrówkę. Dla każdej krawędzi intensywność ścieżki w chwili 0, $\tau_{ij}(0)$, jest ustalana na bardzo małą wartość.

Podczas budowania trasy prawdopodobieństwo, że mrówka k z miasta i odwiedzi miasto j , wynosi

$$P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}t]^\alpha [\eta_{ij}]^\beta}{\sum_{h \in \text{dozwolone}_k(t)} [\tau_{ih}t]^\alpha [\eta_{ih}]^\beta} & \text{jeśli } j \in \text{dozwolone}_k(t) \\ 0 & \text{w przeciwnym wypadku} \end{cases} \quad (16.3)$$

przy czym: $dozwolone_k(t)$ to zbiór miast, które nie zostały odwiedzone przez mrówkę k do chwili t , a η_{ij} reprezentuje lokalną heurystykę. Dla TSP $\eta_{ij} = 1/d(C_i, C_j)$ i jest nazywana „widocznością”.

Parametry α i β sterują istotnością względną śladów feromonowych i widoczności. Prawdopodobieństwo przejścia jest więc kompromisem między widocznością, zgodnie z którą bliższe miasta powinny być wybierane z większym prawdopodobieństwem, a intensywnością śladu, zgodnie z którą występowanie dużego ruchu na połączeniu (i | j) świadczy o tym, że korzystanie z tego połączenia jest opłacalne.

Z każdą mrówką jest związana struktura danych zwana *listą tabu*, która zabezpiecza przed odwiedzeniem miasta więcej niż raz. Na takiej liście $tabu_k(t)$ jest utrzymywany zbiór miast odwiedzonych przez mrówkę k do chwili t . W związku z tym zbiór $dozwolone_k(t)$ możemy zdefiniować jako $dozwolone_k(t) = \{j | j \notin tabu_k(t)\}$. Po zakończeniu trasy lista $tabu_k(t)$ (przy czym $k = 1, \dots, m$) jest opróżniana i każda mrówka może w następnym cyklu obrać inną trasę.

Korzystając z powyższych definicji, możemy podać ogólny szkic algorytmu ACO (rys. 16.5). Takie algorytmy mrówkowe zastosowano do wielu rodzajów problemów, od TSP do wyznaczania tras w sieciach telekomunikacyjnych [103].

```

procedure system mrówkowy
inicjuj
for t = 1 to liczba cykli do
begin
    for k = 1 to m do
    begin
        repeat
            wybierz dla mrówki k następne miasto
            z prawdopodobieństwem  $P_{ij}^k$  dla j-tego miasta
            opisanym przez wzór (16.3)
        until mrówka k nie zakończyła trasy
        oblicz długość  $L_k$  trasy wygenerowanej przez mrówkę k
    end
    zapamiętaj najlepsze dotychczas znalezione rozwiązanie
    uaktualnij intensywność śladów  $\tau_{ij}$  na wszystkich ścieżkach
    zgodnie ze wzorem (16.2)
end

```

Rysunek 16.5. Procedura systemu mrówkowego

Istnieją też inne pokrewne algorytmy, które modelują zjawiska występujące w rojach i które są wykorzystywane w optymalizacji. Metoda *roju cząsteczek* (ang. *particle swarm*) [259] działa na populacji osobników, względem których jest stosowane różnicowanie, ale bez selekcji. Każdy osobnik ma pozycję i przed-

kość, które są uaktualniane zgodnie z zależnością między parametrami osobnika a najlepszym dotychczas znalezionym położeniem osobnika w populacji. W ten sposób poszukiwanie jest przesunięte w kierunku lepszych obszarów przestrzeni przeszukiwania, czego rezultatem jest swego rodzaju „gromadzenie się” blisko najlepszych rozwiązań. Inną trochę podobną procedurą jest *ewolucja różnicowa* (ang. *differential evolution*) [264]. W jej najbardziej podstawowym sformułowaniu różnicowanie jest wykonywane w ten sposób, że bierze się różnicę wektorową między dwoma losowo wybranymi rozwiązaniami, a następnie dodaje się ją do jakiegoś innego losowo wybranego osobnika, uprzednio ją skalując. Wraz z upływem czasu populacja przesuwa się w kierunku lepszych obszarów przestrzeni przeszukiwania.

Ogólnie rzecz biorąc, dziedzina systemów opartych na agentach staje się nowym paradygmatem obliczeniowym, a jej szybki rozwój może mieć ważne i zasadnicze konsekwencje. Podstawowa ogólna koncepcja polega tu na tym, że agenci o różnych możliwościach mogą na różne sposoby współdziałać ze sobą, żeby rozwiązać jakiś problem. Agent może być uznawany za potencjalne rozwiązanie problemu, jak w algorytmach ewolucyjnych, lub też za jednostkę podejmującą decyzje, jak w systemie mrówkowym. Zachowanie każdego agenta jest oparte na regułach określających jego decyzje. Wynikiem współdziałania agentów jest *emergentne zachowanie* całego systemu, które może być trudne do przewidzenia ze względu na złożoność wszystkich interakcji. Nawet prosty zbiór reguł dla każdego agenta może dać w wyniku złożony system adaptacyjny.

Müller w pracy [331] napisał:

Termin *agent autonomiczny* wydaje się magicznym słowem lat dwudziestolecia XX wieku. Koncepcja autonomicznych programów komputerowych, które są zdolne do elastycznego działania w złożonych i zmieniających się środowiskach wieloagentowych, zmieniła sposób, w jaki sztuczna inteligencja definiuje zakres swych działań, i właśnie znajduje swoją drogę do praktyki przemysłowej inżynierii oprogramowania. Technologia oparta na agentach jest używana do modelowania złożonych, dynamicznych i otwartych aplikacji, na przykład w planowaniu produkcji, sterowaniu ruchem, zarządzaniu przepływem czynności oraz w coraz większym stopniu w Internecie.

Z powodu tej różnorodności nie ma ogólnej zgodności co do tego, czym jest agent lub jakie główne cechy powinien mieć. Zaproponowano wiele różnych architektur. Możemy pogrupować je w trzy kategorie [331].

- *Agenci reaktywni* (ang. *reactive agents*) są zbudowani na podstawie paradygmatu opartego na zachowaniu. Decyzje podejmowane przez takich agentów zapadają zazwyczaj w czasie działania systemu i są oparte na ograniczonej ilości informacji (często uzyskanej z czynników). Część badań w dziedzinie sztucznego życia jest prowadzona w tym kierunku [277].
- *Agenci wnioskujący* (ang. *deliberative agents*) są związani z tradycyjnymi metodami sztucznej inteligencji. Każdy agent ma: 1) wewnętrzną

(symboliczną) reprezentację środowiska i 2) mechanizmy wnioskowania logicznego umożliwiające podejmowanie decyzji. Przykładami tej kategorii są klasyczne systemy planowania.

- *Agenci komunikujący się* (ang. *interacting agents*) potrafią koordynować swoje działania z działaniami pewnych innych agentów, a robią to, komunikując się i negocjując. Tacy agenci zwykle utrzymują o sobie nawzajem pewną wiedzę, a także mogą wnioskować na temat innych agentów. Kierunek ten bada rozproszona sztuczna inteligencja.

Algorytmy ewolucyjne można stosować do projektowania systemów opartych na agentach, a nawet do ich optymalizacji za pomocą różnicowania i selekcji wykonywanych względem zachowania agentów w symulacji. W pewnym sensie algorytmy ewolucyjne reprezentują jedne z najbardziej ogólnych procedur opartych na agentach.

Widzieliśmy już inną możliwość, kiedy algorytmy ewolucyjne są używane w kontekście systemów koewolucyjnych, gdzie rozwiązania konkurują i/lub współpracują ze sobą, żeby zapewnić sobie przetrwanie. Istnieje wiele sposobów stosowania koewolucji – przypomnijmy sobie kilka z nich.

Moglibyśmy, na przykład, rozważyć scenariusz z wieloma populacjami i przebiegającymi w nich różnymi procesami ewolucyjnymi. Te procesy ewolucyjne mogą ze sobą współdziałać, a funkcja oceny dla każdej z populacji może zależeć od innych osobników obecnych w populacji lub też od stanu procesu ewolucji w innych populacjach.

Nawet w ramach jednej populacji istnieje wiele sposobów wykorzystania konkurencyjnych ze sobą rozwiązań. Rozważając iterowany dylemat więźnia (rozdział 11), przekonaliśmy się, że rozwiązania w jednej populacji mogą działać jako strategie, przy czym przystosowanie każdej strategii jest funkcją zależną od innych strategii w populacji. Proces ten można uogólnić na dwie lub więcej populacji, jak w podanych przez Hillisa przykładach z sieciami sortującymi lub jak w przypadku regulatorów ciśnienia krwi opisanych przez Sebalda i Schlenziga. W pracy [350] Paredis użył podobnego pomysłu w kontekście problemów związanych ze spełnianiem ograniczeń. W swojej metodzie zastosował model koewolucyjny, w którym populacja potencjalnych rozwiązań koewoluowała z populacją ograniczeń. Bardziej przystosowane rozwiązania spełniały więcej ograniczeń, natomiast bardziej przystosowane ograniczenia były narušane przez więcej rozwiązań. Osobniki z populacji rozwiązań mogły pochodzić z całej przestrzeni przeszukiwania, a ponadto nie rozróżniano osobników dopuszczalnych i niedopuszczalnych.

Systemy koewolucyjne mogą być uważane za użyteczne narzędzia do modelowania aplikacji biznesowych. Pewien system koewolucyjny zastosowano, na przykład, do modelowania strategii dwóch konkurencyjnych firm przewozowych obsługujących te same trasy, przy czym jedna firma była kolejowa, a druga autobusowa [333]. Rentowność jednej firmy zależała od bieżącej strategii (wielkości taboru i cen) drugiej z nich. W badaniu tym zajmowano się rozwojem w czasie wzajemnych zależności między różnymi strategiami. Podejście to może zostać uogólnione do scenariusza, w którym konkurowuje ze

sobą m firm. Każda z nich jest reprezentowana przez agenta, który posługuje się aktualnie najlepszą strategią, wynikającą z procesu ewolucyjnego w jednej z m populacji. Ciekawe jest obserwowanie organizacji współpracy i konkurencji między różnymi firmami. Czy możliwe jest stwierdzenie współpracy (grupowania) słabszych firm w celu zapewnienia lepszej konkurencyjności z większymi? W jakiej sytuacji (np. przy jakich zasobach) jedna firma eliminuje inną z konkurencji? Mamy tutaj wiele możliwości i możemy wykonać wiele pasjonujących eksperymentów!

Zauważ, że podejście to przeddefiniuje pojęcie agenta. Możemy teraz przyjąć, że każdy agent jest procesem ewolucyjnym, a interakcje między agentami są modelowane za pomocą koewolucji. Zauważ też, że agent reprezentuje strategię firmy i może uwzględniać wiele zmiennych decyzyjnych (np. ceny, inwestycje). Ponieważ wartości tych zmiennych podlegają ewolucji, agent może być uważany za agenta reaktywnego, gdyż jego zachowanie zależy od bieżącego stanu środowiska zdefiniowanego przez strategie innych agentów. Jednocześnie tacy agenci ewolucyjni mogą być uważani za agentów wnioskujących, gdyż mogą oni być rozszerzani o mechanizm wnioskowania logicznego umożliwiający podejmowanie decyzji (mechanizmy te można zakodować jako dodatkowe chromosomy w osobnikach i również poddawać ewolucji). Wreszcie agenci ci zaliczają się także do kategorii agentów komunikujących się, gdyż mogą się komunikować, a nawet negocjować pewne działania.

Systemy koewolucyjne mogą mieć też duże znaczenie przy podchodzeniu do problemów dużej skali [360], gdzie problem jest rozkładany na mniejsze podproblemy, jak widzieliśmy to przy omawianiu koewolucji ze współpracą. Do każdego z podproblemów jest stosowany oddzielny proces ewolucyjny, przy czym procesy te są ze sobą połączone. Jak poprzednio, ocena osobników w jednej populacji zależy od rozwoju pozostałych populacji¹.

16.1. Podsumowanie

Algorytmy ewolucyjne są szczególnie elastycznym narzędziem do rozwiązywania problemów. Nie tylko możemy połączyć je w rozwiązania hybrydowe z bardziej tradycyjnymi algorytmami, takimi jak algorytmy zachłanne (np. przy inicjowaniu populacji), ale także możemy rozszerzyć je tak, żeby obejmowały wiele różnych mechanizmów spotykanych w naturze. Możemy tutaj dołożyć stosowanie wielu populacji, pamięci, uczenia się osobników, uczenia się społecznego i dużo więcej innych koncepcji. Nie ma tu naprawdę żadnych ograniczeń dotyczących tego, co możemy zrobić, tyle tylko, że oczywiście, ogranicza nas własna wyobraźnia.

¹ Wiele tematów w obliczeniach ewolucyjnych, które teraz przyciągają znaczną uwagę, tak naprawdę po raz pierwszy zaproponowano już dekady temu. Koewolucja nie należy tutaj do wyjątków. Barricelli [33], Reed i in. [384], Fogel i Burgin [165] i wielu innych używało w latach sześćdziesiątych XX wieku koewolucji w celu rozwijania strategii w grach i symulowanych walkach.

Z tą elastycznością jest związana pewna cena. Czy pamiętasz twierdzenie o braku darmowych obiadów [499]? Nie ma żadnego algorytmu, który byłby najlepszy do rozwiązywania wszystkich problemów. Bez względu na to, jakie rozszerzenie zrobimy lub na ile lepiej nasz algorytm modeluje jakieś naturalne układy, zawsze będą występowały problemy, dla których będzie on szczególnie dobry, i problemy, dla których będzie on szczególnie kiepski. Niestety, istnieje bardzo mało opracowań teoretycznych pomagających podejmować tutaj odpowiednie decyzje dotyczące tego, które rozszerzenia dołożyć i kiedy. Każde z opisanych przez nas rozszerzeń daje atrakcyjne możliwości. Każde także obliguje korzystającego z niego do używania go z namysłem – w przeciwnym razie można znaleźć się na prostej drodze do uzyskania pierwszego „opartego na agentach systemu wielopopulacyjnej koewolucyjnej kolonii mrówek opartej na metodach roju cząsteczek i ewolucji różnicowej”. Po oddzieleniu systemów hybrydowych od problemów, dla których je opracowano, stają się one interesujące same w sobie. Należy skupić się na rozwiązywaniu problemów zamiast na budowaniu złożonych i bezużytecznych symulacji.

17. Podsumowanie

Czy jeżeli ludożerca je widelcem i nożem – to postęp?

Stanisław J. Lec: *Mysli nieuczescane*¹

W książce przedstawiliśmy obszerny materiał, a ponieważ zbliżamy się do jej końca, skorzystajmy z okazji, żeby powtórzyć to, co omówiliśmy. Podstawowe pytanie, jakie należy sobie zadać, mając do rozwiązania jakiś problem, brzmi: „jaki jest mój cel?”. Jeśli nie masz dobrze określonego i przemyślanego celu, to nadzieja, że go osiągniesz, jest nikła, a nawet gdy go osiągniesz, nie będziesz wiedział, że to zrobiłeś! Ilekroć rozwiążujesz problem, musisz widzieć swój cel. Zbyt łatwo jest po prostu maszerować w poszukiwaniu rozwiązania, od razu na początku zapomniawszy o tym, co się próbuje rozwiązać. Równie łatwo jest skierować uwagę na coś innego, na jakieś uboczne zagadnienie, które może mieć porównywalne znaczenie, ale też może być bezużyteczne.

W pochodzący z początku lat osiemdziesiątych XX wieku filmie telewizyjnym *Magnum*, którego akcja rozgrywa się na Hawajach, a tytułowy bohater to prywatny detektyw grany przez Toma Sellecka, pojawia się doskonała kwestia. W pewnym momencie detektyw musi otworzyć wytrychem zamek, żeby dostać się do głównego budynku w posiadłości, w której przebywał. (*Magnum* był odesłany do budynku dla gości). Posiadłość była strzeżona przez dwa dobermany. Gdy *Magnum* zaczął otwierać zamek, usłyszał nadbiegające psy. Powiedział wówczas do siebie: „Otwieraj zamek. Nie patrz na psy. Otwieraj zamek. Nie patrz na psy”. Oczywiście potem *Magnum* spojrzał na psy. Należy stworzyć sobie najlepsze warunki do rozwiązania problemu i skupić się na nim, mając na myśli konkretny cel.

Po określeniu celu musisz zająć się sposobem, w jaki chcesz modelować swój problem. Jest dość mało prawdopodobne, że będziesz mógł zająć się problemem bezpośrednio. Większość problemów nie ma charakteru matematycznego ani obliczeniowego. Musimy użyć naszej wiedzy, żeby przeformułować lub

¹ Wydawnictwo Literackie, Kraków, wydanie IV, 1972 (przyp. tłum.).

zamodelować problem tak, aby dał się wyrazić w postaci matematycznego lub obliczeniowego schematu. To właśnie w tym miejscu często stoimy wobec trudnych decyzji, które dotyczą tego, co zamieścić w modelu, a co z niego usunąć. Jeśli zdamy się na tradycyjne metody rozwiązywania problemów – programowanie liniowe, regresję liniową itd. – to możemy chcieć z modelu usunąć bardzo wiele. Możemy przyjąć funkcję kosztu w postaci błędu kwadratowego nawet wówczas, gdy nie mamy na myśli tej funkcji. Możemy przybliżać nieliniowe ograniczenia liniowymi tylko po to, żeby uzyskać jakąś odpowiedź. Tego rodzaju drogi na skróty nie zawsze są bez sensu. Czasami użycie ich jest korzystne, ale musisz umieć uzasadnić swoje decyzje.

Po określeniu modelu musisz zdecydować, jaką metodą obliczysz lub znajdziesz rozwiązanie. Jeśli przestrzeń możliwych rozwiązań jest mała, to nie ma potrzeby szukania rozwiązania. Wystarczy wszystko wyliczyć i będziesz miał pewność, że znalazłeś najlepsze rozwiązanie. Większość rzeczywistych problemów nie jest jednak tak dogodna. Są z nimi związane duże przestrzenie przeszukiwania, dla których nie istnieją analityczne metody określania optymalnego rozwiązania, a co więcej sam problem może się zmieniać w czasie, może obejmować szum, który nie jest ani stacjonarny, ani gaussowski, może też zmieniać się w zależności od innych, konkurujących ludzi rozwiązujących problemy, którzy chcą, żebyś odniósł porażkę. Prawdziwy świat nie zawsze jest przyjazny. Musisz rozważyć, jak każda z tych możliwości wpływa na rozwiązania, które oferujesz. Czy są one stabilne, czy też może będą działać tylko w ograniczonym zakresie i warunkach? Czy możesz je adaptować w locie w miarę pojawiania się nowych sytuacji? Czy umiesz przewidzieć, co w następnym kroku zrobią Twoi przeciwnicy? Po opanowaniu materiału tej książki możesz z całą pewnością odpowiedzieć twierdząco na wszystkie te pytania.

Rozwiązywanie problemów jest sztuką. Jak wszystkie przedsięwzięcia artystyczne wymaga ćwiczenia.

- „Przepraszam, jak się mogę dostać do Carnegie Hall?”
- „Ćwicząc, ćwicząc, ćwicząc.”

Jednym ze sposobów, aby stać się naprawdę dobrym w rozwiązywaniu problemów, jest ciągłe zadawanie sobie zagadek – tak jak robiliśmy to tutaj. Nie muszą być one bardzo skomplikowane ani też przebiegłe, ale powinny stanowić pouczającą lekcję. Skoro już przeczytałeś całą książkę, wróć do każdej zagadki poprzedzającej rozdział tej książki i przeanalizuj jeszcze raz związany z nim problem. Czy umiesz znaleźć nowe powiązania między materiałem w następującym po zagadce rozdziale a samą zagadką? Czy na podstawie tych zagadek możesz określić inne główne kwestie w rozwiązywaniu problemów? Zachęcamy Cię do ciągłego poszukiwania nowych książek z zagadkami i łamigłówkami, żeby utrzymywać swoje umiejętności rozwiązywania problemów w gotowości. Jeśli lekcja związana z jakimś problemem nie jest oczywista, spróbuj się zaśtanowić, czy nie umiesz sam stworzyć takiej lekcji. Jeszcze lepiej, gdybyś zaczął tworzyć własne zagadki i zadawał je przyjaciołom, pozwól im też zadawać zagadki!

Większość znanych nam matematyków wie, jak udowadniać fakty. Wynika to z tego, że ćwiczą się w udowadnianiu. Zdobywają doświadczenie, jak udowadniać. Kierują się intuicją co do tego, jakie rzeczy są udowadnialne i jakie rzeczy należy udowodnić na początku. Nie istnieje ogólny algorytm rozwiązywania problemów. We wczesnych latach informatyki próbowano go wymyślić. Jego możliwości były jednak przeklamowane. Być może, któregoś dnia będziemy mieli komputery, które będą automatycznie rozwiązywać za nas problemy. Nasze problemy będziemy opisywać komputerom ustnie i zostawiać im resztę pracy. Jest jeszcze jednak daleka droga do zbudowania Komandora Data z serialu *Star Trek – Nowe pokolenie*. Do tego czasu nasze problemy musimy rozwiązywać sami.

Oto dziesięć heurystyk przydatnych przy rozwiązywaniu problemów. Chcielibyśmy, żebyś pamiętał o nich, rozwiązuając rzeczywiste problemy.

1. *Każdy problem wart rozwiązań jest wart przemyślenia.* Nie śpiesz się z podaniem odpowiedzi. Przemyśl ją. Pomyśl o różnych sposobach operowania informacją, którą masz pod ręką. Wcześniej przedstawiliśmy zagadkę zatytułowaną *Jakie są ceny w sklepach 7–11?* Żeby rozwiązać tę zagadkę, musisz w swoich rozważaniach przejść od liczb wymiernych (dolarów i centów) do liczb całkowitych. Zadanie to wymagało rozłożenia na czynniki pierwsze dużej liczby (711 000 000), a następnie rozdzielenia możliwości na kilka przypadków. Przypadki te były uporządkowane według największego czynnika pierwszego, którym było 79. Podejście to wymagało wykorzystania przeszukiwania wyczerpującego, ale wiele gałęzi drzewa poszukiwań można było dość wcześnie odciąć. Nierazko pierwsze sformułowanie problemu prowadzi do rozwiązania, ale też i do frustracji. Poświęć tyle czasu, ile trzeba na wymyślenie i opracowanie sensownego podejścia do zadania.
2. *Skoncentruj się na elementach zasadniczych i nie przejmuj się szumem.* Często wydaje się, że mamy do czynienia z potwornym problemem z tak dużą ilością szczegółów, że nie wiemy, gdzie zacząć. W takiej sytuacji bardzo ważne jest, żeby mieć jasno określony cel! Gdy wiesz, co chcesz osiągnąć, masz o wiele większe szanse na odfiltrowanie „szumu” w problemie i skupienie się na rzeczach, które są naprawdę istotne. Czy pamiętasz zagadkę *Jaka jest najkrótsza droga?* Miałeś tam znaleźć najkrótszą ścieżkę między dwoma miastami rozdzielonymi rzeką. Wydawało się, że najważniejsze jest ustawnienie mostu przez rzekę, ale w końcu okazało się, że najlepszy sposób rozwiązywania problemu polega na tym, żeby w ogóle zapomnieć o moście i rzece! Ta rzeka to był po prostu taki szum.
3. *Czasami znalezienie rozwiązania może być naprawdę proste. Nie rób z tego rzeczy trudniejszej niż jest w rzeczywistości.* W wypadku niektórych problemów możesz się czuć trochę głupio, nawet gdy już znalazłeś dobrą odpowiedź, ponieważ ktoś inny może pokazać Ci, jak rozwiązać

je w szybszy, łatwiejszy sposób. Jako inżynierowie i naukowcy jesteśmy tak wyćwiczeni w myśleniu w kategoriach symboli i algorytmów, że możemy łatwo zapomnieć o zdrowym rozsądku! Wszystkie zadania z rozdziału zatytułowanego *Czy lubisz proste rozwiązań?* były tego rodzaju (np. szklanka wody i szklanka soku, liczba meczów w turnieju tenisowym, nakładające się wskazówki zegara, pszczoła podróżująca między dwoma poruszającymi się pociągami). Czy próbowałeś „obliczyć” poprawne odpowiedzi do tych zadań? Zadaj je swoim przyjaciołom i zobacz, co zrobią. Jeśli zaczną obliczanie, to nie przerywaj im. Powstrzymaj pokusę, żeby ich zatrzymać. Gdy już skończą, pomóż im odrobiną zdrowego rozsądku. Bądź jednak łagodny.

4. *Strzeż się oczywistych rozwiązań. Mogą być błędne.* Czasami odpowiedzi są tak jasne, że po prostu muszą być poprawne. Jest tak dopóki ktoś nie pokaże Ci, że sprawy nie są tak oczywiste, jak na początku myślałeś. Podaliśmy kilka przykładów nieintuicyjnych zagadek w rozdziale zatytułowanym *Jak dobrą masz intuicję?* Obliczenie średniej prędkości samochodu, który pokonał pewną odległość, wydaje się dość proste, przekonaliśmy się jednak, że nie musi się to zgadzać z naszym pierwszym wyobrażeniem. Często te „proste” zadania są włączone do jakiegoś większego systemu, który jest odpowiedzialny za inne zagadnienie. Gdy większy system zawiedzie, gotowi jesteśmy przejrzeć wszystko tylko nie to, co trzeba.
5. *Nie daj się zwieść wcześniejszemu doświadczeniu.* Doświadczenie jest zazwyczaj atutem, ale musisz używać go ostrożnie. Nie ulegaj chęci rozwiązywania wszystkich swoich problemów „starą znaną” metodą. Nie pozwól, żebyś zabierał się do rozwiązywania zadań, korzystając zawsze z tych samych metod. Twoje ostatnio zdobyte doświadczenie może być mylące przy następnym problemie. Czy pamiętasz jeszcze przypadek, gdy musiałeś podzielić figurę na cztery jednakowe części, a następnie miałeś podzielić kwadrat na pięć jednakowych części? Bądź świadomy swoich uprzedzeń i przesądów, staraj się je rozpoznać i nie pozwól im wygrać z Tobą.
6. *Bierz się do rozwiązywania. Nie mów: „Nie wiem jak”.* Większość ludzi nie planuje przegranej, oni po prostu przegrywają, nie planując. A jeśli nie masz żadnego planu, to świat zaczyna przemykać obok i nie możesz niczego rozwiązać. Nawet zły plan jest zwykle lepszy niż brak planu. Zasadę tę zilustrowaliśmy dwoma zagadkami w rozdziale 1, w którym mieliśmy udowodnić pewną własność wielokąta oraz określić liczbę uściśków ręki, jakie wykonała pani Smith. Oba zadania okazały się łatwe, gdy zaczęliśmy je rozwiązywać, żadne z nich jednak nie dawało wielu punktów zaczepienia, żeby wykonać pierwszy krok. Najważniejsze było tutaj wymyślenie, jaki on powinien być! Należy czegoś spróbować. Jeśli to nie działa, to nie ma wstydu. Należy spróbować czegoś innego. I tak do skutku. Wytrwałość to metoda.

Doskonałym przykładem zastosowania tej zasady są algorytmy ewolucyjne. Zaczynając nawet od zupełnie losowych rozwiązań dla wydawałoby się trudnych problemów i stosując procesy losowego różnicowania i selekcji, możemy szybko dojść do użytecznych wyników. Od czegoś jednak trzeba *zacząć!* Nie można zacząć od pustej populacji.

Inny istotny aspekt procesu rozwiązywania to możliwość „pobawienia” się zadaniem. Zrozumienie jego istoty. Być może, rozwiązanie da się zgadnąć – nawet jeśli okaże się złe, to może prowadzić do interesującego odkrycia. Rozważaliśmy, na przykład, zagadkę z czterema liczbami, gdzie iloczyn pierwszych dwóch był równy podwojonej sumie dwóch pozostałych i na odwrót. Bawiąc się tym zadaniem, stwierdziliśmy, że niemożliwe jest, żeby wszystkie te liczby były duże (*Jakie to liczby?*). Następny krok był na poziomie elementarnego ćwiczenia i polegał na udowodnieniu, że najmniejsza z czterech liczb nie może być większa niż cztery. To sprowadziło problem do czterech stosunkowo łatwych przypadków.

Pamiętaj zawsze o tym powiedzeniu: „Jeśli nie poruszasz się do przodu, to się cofasz”. Zatem naprzód!

7. *Nie ograniczaj się do przestrzeni poszukiwań „zdefiniowanej” przez problem. Rozszerzaj swoje horyzonty.* W naszym pędzie do poruszania się naprzód możemy coś pominąć lub zgubić okazję do obejścia trudnych aspektów problemu. Zadanie może być bardzo jasno sformułowane i możemy myśleć, że rozumiemy je jednoznacznie, a mimo to możemy utknąć. To dobry moment, żeby wprowadzić dodatkowe zmienne, zastanowić się nad tym, co by się stało, gdybyś mógł dostać się do jakiegoś innego punktu na ścieżce do rozwiązania; może mógłbyś już go rozwiązać? Może mógłbyś zbudować most, którym mógłbyś się dostać do tego punktu pośredniego.

Przypomnij sobie zadanie z kulą na stole bilardowym (rozdział 2). Jeśli jesteś zmuszony do ograniczenia myślenia do ram stołu, to kula bilardowa odbija się nie tylko na stole, ale też po całej Twojej głowie! Gdy rozszerzysz swoje horyzonty lub nawet odwróciś perspektywę, pozwalając, żeby stół poruszał się za kulą podążającą po linii prostej, to odpowiedź stanie się jasna. W zarządzaniu nazywamy to „myśleniem nieszablonowym”. Czy pamiętasz jeszcze zadanie, w którym miałeś zbudować cztery trójkąty z sześciu zapałek (znów w rozdziale 2)? Nawet jedna z wcześniej podanych zagadek z trójkątem i punktem wewnętrznym może być łatwo rozwiązana, jeśli, jak zaraz się przekonasz, poszerzysz trochę swoje horyzonty.

8. *Ograniczenia mogą pomagać.* Wydawałoby się, że ograniczenia służą do tego, żeby grać nam na nerwach – gdybyśmy tylko mogli się ich pozbyć! W istocie ograniczenia nie zawsze muszą być interpretowane jako dodatkowe elementy, które tylko zwiększą złożoność problemu. W końcu

wynika z nich podzbiór dopuszczalnych rozwiązań w przestrzeni poszukiwań, a zatem możemy nasze poszukiwanie skoncentrować na tym obszarze. Uwagę tę dobrze ilustrowała zagadka, w której mieliśmy znaleźć sześciocyfrową liczbę o pewnych interesujących własnościach (tzn. ograniczeniach; *Czy potrafisz dostosować się do problemu?*). W rozdziale 9, który poprzedzał tę zagadkę, omawialiśmy niektóre z tego rodzaju metod (np. operatory zależne od problemów i przeszukiwanie granicy między dopuszczalną a niedopuszczalną przestrzenią poszukiwań), które zyskują na sytuacji, gdy w problemie występują ograniczenia. Jeśli masz do czynienia z problemem o dużej ilości ograniczeń, to myśl pozytywnie. Co masz do stracenia? Myślenie negatywne rzadko prowadzi do rozwiązania problemu.

9. *Nie mieć satysfakcji ze znalezienia jakiegoś rozwiązania.* Znalezienie jakiegoś rozwiązania nie musi oznaczać końca procesu rozwiązywania. Czasami to dopiero początek! Czy to rozwiązanie jest jedyne? Czy są jakieś inne możliwości? Czy pamiętasz przypadek pana Brunatnego i kwestię określenia koloru niedźwiedzia, którego zobaczył? Jeśli doszedłeś do tego, jak wyglądają wszystkie możliwe układy geograficzne, które rozwiązują zagadkę, to zdobyłeś zdecydowanie lepsze zrozumienie zagadki niż ktoś, kto znalazł tylko jedno rozwiązanie i zgadł kolor niedźwiedzia, choćby uzyskana odpowiedź była prawidłowa.
10. *Bądź cierpliwy. Bądź wytrwały.* Każdy, kto rozwiązuje zadania, czasem nie radzi sobie. W istocie częściej to zachodzi, niż nie zachodzi. Musisz mieć cierpliwość, żeby zrozumieć problem, przemyśleć go, sformułować model, wypróbować metody korzystania z niego i jeszcze dojść do tego, że się myliłeś. Czy pamiętasz zadanie dotyczące długości linii, która utrzymywała małpę i ciężarek w równowadze? Przebrnięcie przez ten problem wymagało prawdziwej cierpliwości. Jeśli masz poddać się frustracji, to przejdź do punktu 6 i spróbuj jeszcze raz. Jeśli wydaje Ci się, że Twoje osiągnięcia przy rozwiązywaniu problemów są niezbyt dobre, to weź pod uwagę, że w wysokiej klasy lidze baseballowej, gdybyś trafił raz na trzy razy, to byłbyś graczem w czołówce. Jeśli to Ci nie pomaga, to przyjrzyj się karierze tego człowieka.
 - (a) W roku 1832 stracił pracę.
 - (b) W roku 1832 przegrał w wyborach do parlamentu stanowego.
 - (c) W roku 1833 nie powiodło się jego przedsięwzięcie.
 - (d) W roku 1834 został wybrany w wyborach do parlamentu stanowego.
 - (e) W roku 1835 umarła jego ukochana.
 - (f) W roku 1836 miał załamanie nerwowe.
 - (g) W roku 1838 przegrał, ubiegając się o stanowisko spikera.
 - (h) W roku 1843 przegrał nominację do Kongresu.

- (i) W roku 1846 został wybrany do Kongresu.
- (j) W roku 1848 przegrał ponowną nominację do Kongresu.
- (k) W roku 1854 został pokonany w wyborach do Senatu USA.
- (l) W roku 1856 został pokonany przy nominacji na wiceprezydenta.
- (m) W roku 1858 ponownie został pokonany w wyborach do Senatu USA.
- (n) W roku 1860 został wybrany na prezydenta USA.

Nawet gdyby Abraham Lincoln nic nie osiągnął, można o nim powiedzieć, że był wytrwały! Nie poddawaj się!

Jeśli będziesz pamiętać o tych dziesięciu „przykazaniach”, to Twoja sprawność rozwiązywania problemów stanie się potężna, nie zapominaj jednak o niebezpieczeństwstwie robienia łatwych uogólnień. Czasami udaje się nam dostrzec wzór, który prowadzi do poprawnego rozwiązywania. Rozważmy na przykład następujący scenariusz. Mamy do czynienia z 2^n tenisistami; dwóch z nich jest bliźniakami. Zasady turnieju są takie jak zwykle. W pierwszej rundzie jest 2^{n-1} meczów. Zwycięzcy przechodzą do następnej rundy, na którą składa się 2^{n-2} meczów itd. Przy założeniu, że każdy gracz ma 50% szans na wygranie gry, jakie jest prawdopodobieństwo, że bliźniacy zagrają przeciwko sobie w tym turnieju?

Oznaczmy przez p prawdopodobieństwo, że bliźniacy spotkają się w turnieju. Oczywiście p jest funkcją n :

- Jeśli $n = 1$ (tzn. jeśli są dwaj gracze w turnieju), to $p = 1$, gdyż bliźniacy na pewno się spotkają.
- Jeśli $n = 2$ (tzn. jeśli są czterej gracze w turnieju), to $p = 1/2$. Czy wiesz dlaczego? Jest tak, ponieważ szansa, że bliźniacy zagrają razem w pierwszej rundzie, wynosi $1/3$, a szansa, że będą grali z innymi przeciwnikami, wynosi $2/3$. W drugim przypadku mają szanse $1/4$, że spotkają się w drugiej rundzie, gdyż obaj muszą wygrać swoje mecze, a szansa takiej wygranej dla każdego z nich wynosi $1/2$. Zatem

$$p = \frac{1}{2}, \quad \text{gdyż} \quad \frac{1}{3} \cdot 1 + \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{2}$$

- Rozsądek zatem wskazuje, że $p(n) = 1/2^{n-1}$.

I rzeczywiście tak jest, co możemy udowodnić przez indukcję.

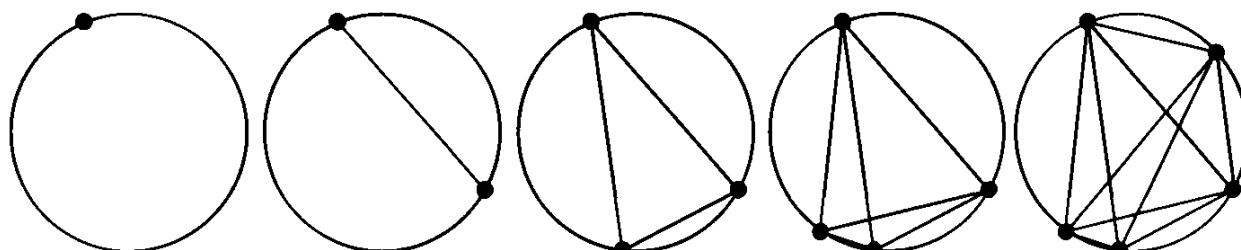
- Szanse, że bliźniacy są w dwóch przeciwnych połowaach drzewa turnieju wynoszą odpowiednio $p_1 = (2^{n-1} - 1)/(2^n - 1)$ oraz $p_2 = (2^{n-1})/(2^n - 1)$.
- Szanse, że się spotkają w pierwszym przypadku (ta sama połowa) wynoszą $m_1 = 1/2^{n-2}$ (z założenia indukcyjnego).

- Szanse, że się spotkają w drugim przypadku (przeciwne połowy) wynoszą $m_2 = 1/2^{n-1} \cdot 1/2^{n-1}$, gdyż mogą się spotkać tylko w końcowym meczu. Obaj muszą zatem wygrać wszystkich $n - 1$ meczów.
- Prawdopodobieństwo, że się spotkają wynosi:

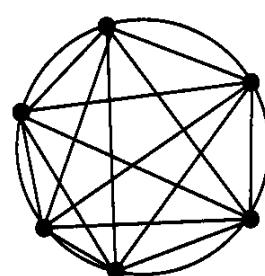
$$p = p_1 \cdot m_1 + p_2 \cdot m_2 = \frac{2^{n-1} - 1}{2^n - 1} \frac{1}{2^{n-2}} + \frac{2^{n-1}}{2^n - 1} \frac{1}{2^{n-1}} \cdot \frac{1}{2^{n-1}} = \frac{1}{2^{n-1}}$$

Hipoteza, że $p(n)$ jest równe $1/2^{n-1}$, okazała się prawdziwa. Musimy jednak uważać przy formułowaniu takich hipotez, o czym się przekonamy, analizując poniższy przykład.

Na rysunku 17.1 przedstawiono pięć początkowych przypadków (dla $n = 1, 2, 3, 4$ i 5) rozmieszczenia n punktów na okręgu, przy czym każdy punkt jest połączony z pozostałymi za pomocą cięciwy. Na ile części jest podzielone każde koło? Wydaje się, że dodanie 6. punktu da 32 fragmenty. W rzeczywistości jednak otrzymujemy 31 części (zob. rys. 17.2), gdyż wzór na ich liczbę ma postać $(n^4 - 6n^3 + 23n^2 - 18n + 24)/24$.



Rysunek 17.1. Pięć początkowych przypadków dzielenia koła cięciwami



Rysunek 17.2. Dzielenie koła dla $n = 6$

Rysunek 17.1 wskazuje, że liczba części rośnie zgodnie ze wzorem 2^{n-1} , gdyż

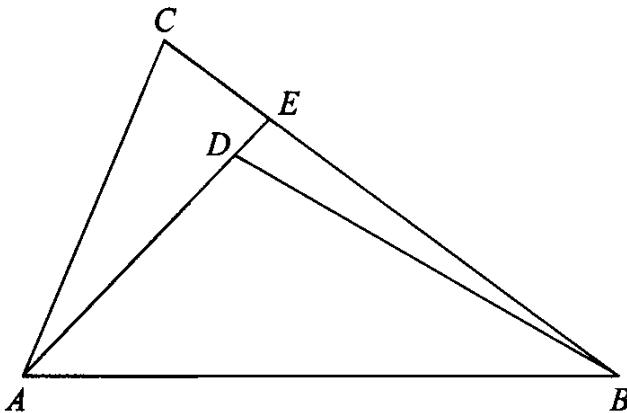
n	Liczba części
1	1
2	2
3	4
4	8
5	16

Musisz uważać, żeby nie wyciągać łatwych wniosków, które „muszą” być prawdziwe. Ponieważ jesteśmy ludźmi, dobrze nam wychodzi zauważanie wzorów. Jesteśmy w tym tak dobrzy, że możemy je wymyślić tam, gdzie ich nie ma! Zawsze staraj się polegać przede wszystkim na solidnych dowodach, a nie na przeczuciach.

Wróćmy teraz do dwóch zagadek przedstawionych we wstępie. Pierwsza z nich polegała na udowodnieniu, że

$$AD + DB \leq AC + CB$$

w trójkącie ABC , przy czym D jest dowolnym punktem jego wnętrza (rys. 17.3).



Rysunek 17.3. Trójkąt ABC z wewnętrznym punktem D oraz przedłużenie odcinka AD

Rozszerzmy odcinek AD w kierunku boku trójkąta i oznaczmy punkt przecięcia tego odcinka z bokiem BC przez E . Ponieważ suma dwóch boków trójkąta jest dłuższa od trzeciego boku, mamy

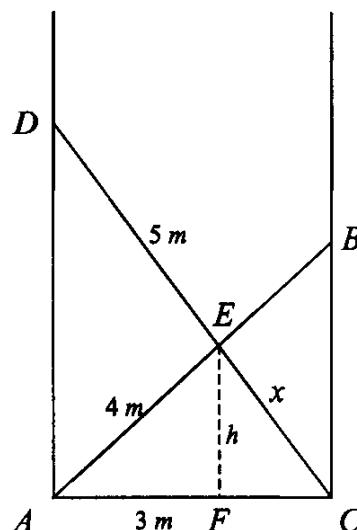
$$\begin{aligned} AD + DB &\leq AD + (DE + EB) = AE + EB \leq (AC + CE) + EB = \\ &= AC + CB \end{aligned}$$

co kończy nasz dowód. Mógłbyś też skonstruować odcinek, który przechodzi przez D i jest równoległy do AB , a następnie przejść przez bardzo podobny ciąg nierówności.

Druga zagadka była o studni o średnicy trzech metrów, do której wrzuciliśmy dwa patyki o długości odpowiednio czterech i pięciu metrów, które wylądowały na dnie w szczególnym układzie. Zadanie polegało na znalezieniu odległości h od dna studni do punktu przecięcia patyków. Wprowadziłmy następujące oznaczenia. Końce krótszego patyka to A i B , końce dłuższego to C i D (punkty A i D dotykają dna studni). Punkt przecięcia to punkt E . Wysokość h punktu przecięcia nad dnem studni to długość odcinka EF . Wielkość x oznacza długość odcinka CE (rys. 17.4).

Wiemy, że $AB = 4$ oraz $CD = 5$. Łatwo można obliczyć, że $AD = 4$ i $BC = \sqrt{7}$ (z twierdzenia Pitagorasa). Trójkąty CEF i CDA są podobne, zatem

$$\frac{EF}{DA} = \frac{EC}{CD} \Rightarrow \frac{h}{4} = \frac{x}{5}$$



Rysunek 17.4. Dwa płytki w studni

zatem $h = 4x/5$. Trójkąty EBC i EAD są również podobne, a więc

$$\frac{EC}{BC} = \frac{DE}{DA} \Rightarrow \frac{x}{\sqrt{7}} = \frac{5-x}{4}$$

To ostatnie równanie daje $x = 5\sqrt{7}/(4 + \sqrt{7})$, zatem

$$h = \frac{4x}{5} = \frac{4\sqrt{7}}{4 + \sqrt{7}}$$

Trudno jest docenić rozwiązania tych dwóch zagadek, jeśli się nie spędziło trochę czasu na ich rozwiązywaniu. Rzeczywiście wiele kwestii rozważanych wcześniej w tym rozdziale możemy zilustrować za pomocą przeprowadzonych tu rozumowań.

Poznaliśmy tutaj wiele różnych algorytmów rozwiązywania problemów. Bez wątpienia spędziliśmy większość czasu na rozważaniu możliwości stosowania algorytmów ewolucyjnych przy rozwiązywaniu zadań. Stało się tak, gdyż algorytmy te mają wiele zalet.

- Algorytmy ewolucyjne są w swej istocie bardzo proste.* Podstawowy przebieg procedury jest prosty. Populacja rywalizujących rozwiązań w czasie kilku pokoleń podlega losowemu różnicowaniu i selekcji ze względu na pewną funkcję oceny. W porównaniu z wieloma metodami opartymi na rachunku różniczkowym i całkowym algorytmy ewolucyjne są łatwe do implementacji i zrozumienia.
- Algorytmy ewolucyjne mają szerokie możliwości stosowania.* Algorytmy ewolucyjne można zastosować do każdego problemu, który omówiliśmy w tej książce. Bezspornie istnieje wiele lepszych sposobów rozwiązywania niektórych z tych problemów i dla tych problemów nie zalecamy stosowania algorytmów ewolucyjnych. Gdy dysponujesz metodą, która jest idealna dla Twojego zadania, użyj jej! Lecz bardzo często możesz dojść do wniosku, że świat rzeczywisty stwarza problemy, które nie pasują do klasycznych algorytmów i będziesz zmuszony albo pójść na

kompromis z założeniami dotyczącymi zadania, żeby zastosować jedną z tych procedur z półki, albo przyjąć inne podejście. Podejście ewolucyjne jest elastyczne. Możesz je dostosować do wielu różnych sytuacji, a co więcej, możesz go użyć do tego, żeby dostosowywało się samo w miarę poznawania problemu.

3. *Algorytmy ewolucyjne dają możliwość tworzenia przydatnych metod hybrydowych.* Większość klasycznych algorytmów możemy użyć albo odrzucić. Z kolei algorytmy ewolucyjne dają możliwość włączenia do procedury wiedzy związanej z dziedziną ich zastosowania. Możesz je też połączyć zarówno z metodami deterministycznymi, jak i lokalnymi. Możesz opracować operatory różnicowania, które korzystają ze specyficznych informacji związanych z problemem, jak mieliśmy okazję to zobaczyć w przypadku operatora krzyżowania geometrycznego w rozdziale 9. Możesz nawet połączyć obliczenia ewolucyjne z sieciami neuronowymi i systemami rozmytymi. We wszystkich zastosowaniach, które opracowaliśmy dla problemów ze świata rzeczywistego, a jest to łącznie ponad 30 lat doświadczeń, nigdy nie używaliśmy algorytmu ewolucyjnego bez dołączenia jakiegoś innego aspektu związanego z problemem.
4. *Algorytmy ewolucyjne są ze swojej natury równolegle.* W miarę jak komputery stają się tańsze i szybsze, perspektywa stosowania komputerów do przetwarzania równoległego staje się coraz sensowniejsza. Niedługo większość komputerów biurkowych będzie maszynami wieloprocesorowymi. Algorytmy ewolucyjne często zyskują na tego rodzaju architekturze, ponieważ wiele z rozwiązań umożliwia ocenianie i różnicowanie rozwiązań w trybie równoległym. Co więcej, jak przekonaliśmy się w rozdziale 16, istnieją algorytmy ewolucyjne, które pracują na podpopulacjach, można więc sobie wyobrazić, że każda z nich jest obsługiwana przez oddzielny procesor. Możemy w ten sposób znacznie skrócić czas wykonania algorytmu ewolucyjnego.
5. *Algorytmy ewolucyjne są odporne na zmiany dynamiczne.* W rozdziale 11 zobaczyliśmy, że możemy zyskać na wiedzy, jaką zebrał algorytm w trakcie przetwarzania, wykorzystując ją do rozwiązywania problemu w czasie, gdy się zmienia. Czasami problem może zmieniać się tak bardzo, że nasz ostatni zbiór najlepszych rozwiązań może być bezwartościowy. Rozpoczęcie od tych właśnie rozwiązań nie kosztuje wiele, gdyż jako alternatywę mamy rozpoczęcie od zera. Świat rzeczywisty zawsze stawia przed nami problemy, które się zmieniają. Nigdy nie wystarczy rozwiązanie statycznego problemu. W czasie gdy Ty podziwiasz swoje rozwiązanie, ktoś inny już rozwiązuje następny problem. Algorytmy ewolucyjne dają zawarty w populacji magazyn wiedzy, który możesz wykorzystać.
6. *Algorytmy ewolucyjne dają możliwość samoadaptacji.* Elastyczność algorytmów ewolucyjnych objawia się w wielu parametrach sterujących,

które masz do dyspozycji. Poznaliśmy sposób, żeby zamiast próbować dostrajać te parametry ręcznie, pozwolić, aby algorytm ewolucyjny przeprowadzał na bieżąco, w miarę zapoznawania się z problemem, ewolucję ich wartości. Tego rodzaju samoadaptacyjne metody są swojego rodzaju formą uczenia się przez wzmacnienie, w którym strategie rozwiązania problemu, które zadziałyły w przeszłości, są nagradzane. W miarę jak zmieniają się warunki, algorytm ewolucyjny może uaktualnić swoje strategie poszukiwania i w dalszym ciągu poszukiwać nowych sposobów rozwiązania problemu.

7. *Algorytmy ewolucyjne mogą nauczyć się rozwiązywania problemów, dla których nie ma znanych rozwiązań.* Prawdopodobnie największa zaleta algorytmów ewolucyjnych wynika z ich zdolności do rozwiązywania problemów, w których rozwiązywaniu ludzie nie są biegli. Chociaż ludzka wiedza powinna być wykorzystywana za każdym razem, gdy tylko jest dostępna, często jest zdecydowanie niewystarczająca do opracowania automatycznej procedury rozwiązującej problem. Ekspertki nie zawsze się ze sobą zgadzają, mogą podawać niespójne informacje, mogą nie mieć odpowiednich kwalifikacji lub mogą się po prostu mylić. Jeśli wiedza jest rzeczywiście dostępna, to można ją uwzględnić jako wskazówkę do rozmieszczenia wśród populacji i włączyć do rywalizacji z innymi pomysłami. Na koniec zaś selekcja naturalna może określić, co działa, a co nie działa.

Algorytmy ewolucyjne to nie panaceum, ale często mogą wygenerować w rozsądny czasie praktyczne rozwiązania trudnych problemów. Z solidnymi podstawami wszystkich metod rozwiązywania problemów opisanyimi w tej książce oraz zdając sobie sprawę z ich zalet i ograniczeń jesteś doskonale wyposażony i gotowy do działania.

Zakończymy, opisując zachwycający problem, w którym będziemy mieli do czynienia z przygotowywaniem trzech toastów na maśle. (Nie myślałeś chyba, że pójdzie Ci tak łatwo?!) Opis problemu jest następujący.

- Mamy stary toster z dwoma drzwiczkami na zawiasach po każdej stronie. Można do niego włożyć jednocześnie dwa testy, a on opieka na raz jedną stronę kromki.
- Wymagane czynności pochłoną:
 - 30 sekund – upieczenie jednej kromki (po jednej stronie);
 - 3 sekundy – włożenie kromki do tostera;
 - 3 sekundy – wyjącie kromki z tostera;
 - 3 sekundy – obrócenie kromki bez wyjmowania jej z tostera;
 - 12 sekund – posmarowanie masłem jednej strony testa.

- Dodatkowo czynności włożenia kromki, odwrócenia jej, wyjęcia lub posmarowania masłem wymagają użycia obu rąk, nie mogą być więc wykonywane jednocześnie.
- Każda kromka jest smarowana tylko po jednej stronie; masło możemy jednak rozsmarować dopiero po podgrzaniu tej strony tostu.

Wszystkie te specyfikacje, jak w przypadku rzeczywistych problemów, są bardzo ważne. Zgodnie z powyższymi wymaganiami możliwe jest: 1) włożenie podgrzanej i posmarowanej kromki chleba z powrotem do tostera lub 2) podgrzewanie jednej strony tostu przez, powiedzmy, 15 sekund, wyjęcie go na chwilę, a następnie po upływie jakiegoś czasu kontynuowanie podgrzewania przez pozostałe 15 sekund. Musimy założyć, że na początku trzy kromki chleba do opiekania nie znajdują się w tosterze, a opiekanie zakończymy, gdy te trzy podgrzane kromki znajdują się poza tosterem.

Zadanie polega, jak zapewne już zgadłeś, na opracowaniu planu czynności, który umożliwi opieczenie i posmarowanie masłem trzech kromek chleba w najkrótszym czasie! Proste rozwiązanie daje czas 120 sekund, ale da się to zrobić dużo szybciej! Ale jak znajdziesz ten plan? Użyjesz do tego algorytmu zuchłanego? Poszukiwania wyczerpującego? Algorytmu ewolucyjnego? Jakiegoś rozwiązania hybrydowego?

Korzystając z naszego zdrowego rozsądku (!), znaleźliśmy rozwiązanie, które daje 114 sekund. Co ciekawe, słynny matematyk Martin Gardner twierdzi na stronie 17 książki [178]:

„... czas ten można zmniejszyć do 111 sekund...”.

Próbowaliśmy znaleźć rozwiązanie o czasie 111 sekund, ale nie udało się nam to. Wydaje nam się, że twierdzenie Gardnera jest podobne do słynnego zdania Pierre'a Fermata, który (około 1637 roku) napisał na marginesie *Arytmetyki*:

Nie można zapisać sześcianu liczby jako sumy dwóch sześcianów ani czwartej potęgi liczby jako sumy dwóch czwartych potęg, ani ogólnie żadnej liczby będącej potągą większą niż druga jako sumy dwóch takich samych potęg.

Innymi słowy, równanie $x^n + y^n = z^n$ nie ma rozwiązania całkowitoliczbowego dla n większych od 2. Uwaga ta była zwieńczona słynnym zdaniem:

Mam naprawdę wspaniały dowód tego twierdzenia, ale ten margines jest zbyt wąski, żeby go tu zapisać.

Ponieważ twierdzenie to (znane jako wielkie twierdzenie Fermata) ostatnio udowodnił – po ponad 350 latach! – Andrew Wiles [498, 465], mamy nadzieję na odkrycie rozwiązania problemu tostowania, które daje 111 sekund. Mamy też nadzieję, że nie zajmie to kilku wieków¹!

¹To tylko żart: w późniejszych wydaniach [178] liczba 111 została zamieniona na 114, gdyż w pierwotnej wersji liczono czas wymagany do skończenia podgrzewania, a trzy sekundy należy dodać, żeby wyjąć kromkę z tostera. Co więcej, Walter Kosters z Leiden University w Holandii przysłał nam elegancki dowód, że niemożliwe jest skonstruowanie rozwiązania krótszego niż 114 sekund. W przypisie tym mamy jednak za mało miejsca, żeby się zmieścił...

A. Rachunek prawdopodobieństwa i statystyka

Bóg nie gra w kości.

Albert Einstein

Pojęcia rachunku prawdopodobieństwa i statystyki umożliwiają modelowanie wielu zjawisk naturalnych, w których występuje losowość. Wiele wczesnych odkryć w tych dziedzinach wyrosło z potrzeby zrozumienia działania różnych gier, w których jakąś rolę odgrywa szansa. Rzeczywiście, mało ludzi zwraca większą uwagę na rachunek prawdopodobieństwa niż właściciele kasyn w Las Vegas. Podstawy rachunku prawdopodobieństwa i statystyki pojawiają się jednak w prawie wszystkich przedsięwzięciach związanych z rozwiązywaniem problemów, a więc ważne jest, żeby rozumieć najważniejsze pojęcia tych dziedzin.

A.1. Podstawowe pojęcia rachunku prawdopodobieństwa

Rachunek prawdopodobieństwa jest działem matematyki, który zajmuje się niepewnością nieodłącznie związaną ze zjawiskami świata rzeczywistego. Stałym elementem rozważań na temat prawdopodobieństwa jest pojęcie *eksperymentu*. *Przestrzeń próbkowa* S określa zbiór wszystkich możliwych wyników eksperymentu. Następnie zbiorom wyników są przypisywane prawdopodobieństwa. Zbiory te są nazywane *zdarzeniami*. Rozważmy, na przykład, rzuty kostką do gry. Możliwych jest sześć wyników, które określają przestrzeń próbłową dla tego eksperymentu: $S = \{1, 2, 3, 4, 5, 6\}$. Możemy tutaj zdefiniować takie zdarzenia, jak $A = \{\text{Wyrzucono liczbę parzystą}\}$ lub $B = \{\text{Wyrzucono liczbę mniejszą od czterech}\}$. Z formalnego punktu widzenia prawdopodobieństwa są przypisywane tylko do zdarzeń, a nie do wyników. Opis prawdopodobieństwa wyrzucenia 1 uzyskujemy przez wskazanie zbioru $A = \{1\}$ i obliczenie prawdopodobieństwa zdarzenia A , które oznaczamy $P(A)$ lub $\Pr(A)$. Zauważmy jeszcze, że przestrzeń próbkowa S jest też zdarzeniem – zawiera ono wszystkie możliwe wyniki.

Podobnie, zbiór pusty \emptyset jest również zdarzeniem – zdarzeniem, które nie zawiera żadnego wyniku eksperymentu.

Użyteczne jest pierwotne zdefiniowanie prawdopodobieństwa za pomocą trzech aksjomatów:

1. $P(A) \geq 0$.
2. $P(S) = 1$.
3. (a) Jeśli A_1, A_2, \dots, A_n jest skończonym ciągiem parami rozłącznych zdarzeń, to prawdopodobieństwo $P(A_1 \cup A_2 \cup \dots \cup A_n) = \sum_{i=1}^n P(A_i)$.
- (b) Jeśli A_1, A_2, \dots jest nieskończonym ciągiem parami rozłącznych zdarzeń, to prawdopodobieństwo $P(A_1 \cup A_2 \cup \dots) = \sum_{i=1}^{\infty} P(A_i)$.

Te elementarne aksjomaty stanowią podstawę zajmowania się prawdopodobieństwem w sposób intuicyjny. Prawdopodobieństwo uzyskania pewnego (tj. dowolnego) wyniku eksperymentu wynosi 1,0. Jest to maksymalne prawdopodobieństwo i odpowiada zdarzeniu, które zawiera wszystkie możliwe wyniki. Prawdopodobieństwo każdego innego zdarzenia A różnego od S musi być równe lub większe od zera. Oznacza to, że prawdopodobieństwa mieszczą się zawsze w przedziale $[0, 1]$. Wreszcie, jeśli dwa lub więcej zdarzeń jest parami rozłącznych, to prawdopodobieństwo sumy tych zdarzeń jest równe sumie prawdopodobieństw poszczególnych zdarzeń niezależnie od tego, czy chodzi o skończony czy nieskończony ciąg takich rozłącznych (wzajemnie się wykluczających) zdarzeń.

Te aksjomaty prowadzą bezpośrednio do pewnej podstawowej obserwacji. Niech A' oznacza dopełnienie zdarzenia A ; to znaczy zbiór zawierający wszystkie wyniki z S , które nie znajdują się w A . Wówczas z definicji wynika, że A i A' są rozłączne, a ich sumą jest S . Drugi i trzeci aksjomat dają: $P(A) + P(A') = P(S) = 1$. Z czego uzyskujemy, że prawdopodobieństwo dopełnienia zdarzenia A wynosi $1 - P(A)$.

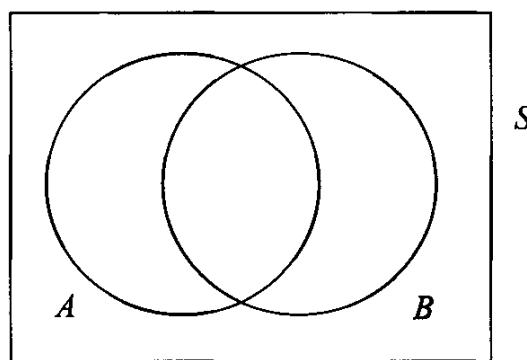
Często chcemy określić prawdopodobieństwo zajścia zdarzenia A , gdy wiemy, że zaszło już pewne inne zdarzenie B . Tego rodzaju prawdopodobieństwo nazywamy *prawdopodobieństwem warunkowym* i zapisujemy jako $P(A|B)$, przy czym zapis ten czytamy jako „prawdopodobieństwo A pod warunkiem B ”. Taką sytuację przedstawiono na rys. A.1, stosując tzw. *diagram Venna*. Wielkości $P(A|B)$ nie można wyprowadzić bezpośrednio z aksjomatów prawdopodobieństwa, musi zatem pojawić się jej definicja:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad \text{dla } P(B) > 0$$

Takie ujęcie jest sensowne, gdyż żeby zdarzenie $A|B$ mogło zajść, musi zajść B , co następuje z prawdopodobieństwem $P(B)$, a wynik musi należeć do iloczynu zdarzeń A i B , co zachodzi z prawdopodobieństwem $P(A \cap B)$.

Dwa zdarzenia A i B są określane jako *niezależne* wtedy i tylko wtedy, gdy

$$P(A \cap B) = P(A)P(B)$$



Rysunek A.1. Diagram Venna przedstawiający przestrzeń próbłową S oraz dwa zdarzenia A i B mające częściowo wspólne wyniki

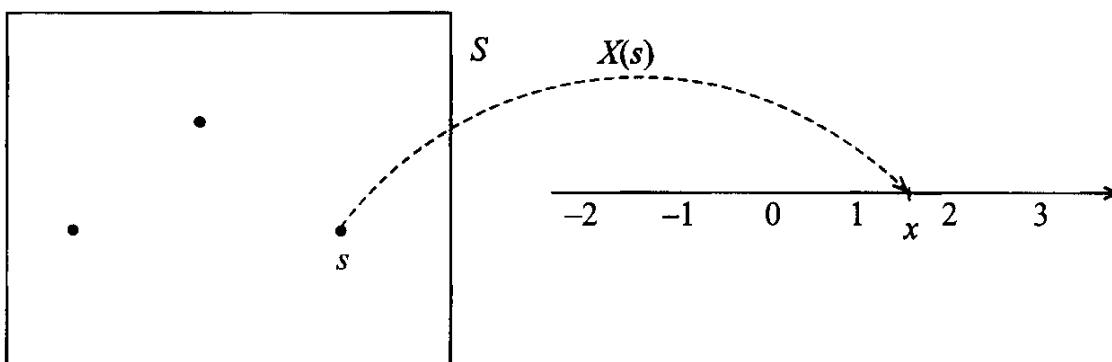
Tu napotykamy pierwszy paradoks, gdyż wydaje się, że niezależne są takie rzeczy, które nie mają wspólnych elementów. Takie podejście nie ma jednak nic wspólnego z niezależnością w świecie prawdopodobieństwa. Niezależność w sensie matematycznym oznacza, że fakt zajścia zdarzenia B nie daje żadnej informacji, która miałaby wpływ na zmianę prawdopodobieństwa zajścia zdarzenia A . Zobaczmy to, podstawiając iloczyn $P(A)P(B)$ w miejsce $P(A \cap B)$ w definicji $P(A|B)$. Prawdopodobieństwo B występujące w liczniku i mianowniku możemy skrócić, zatem w wyniku otrzymujemy, że gdy A i B są niezależne, prawdopodobieństwo warunkowe $P(A|B) = P(A)$. Gdy A i B są rozłączne, prawdopodobieństwo iloczynu (koniunkcji) zdarzeń A i B jest zerowe. Oznacza to, że gdy A i B są rozłączne, zdarzenia A i B nie są niezależne z wyjątkiem trywialnej sytuacji, gdy $P(A) = 0$.

A.2. Zmienne losowe

Mając podstawowy opis prawdopodobieństwa zdarzeń pochodzących z przestrzeni próbowej, można zdefiniować funkcje, które odwzorowują wyniki z przestrzeni próbowej na liczby rzeczywiste. Takie funkcje nazywamy *zmiennymi losowymi* (rys. A.2). Zmienne losowe są często oznaczane zapisanymi kursywą wielkimi literami, na przykład X czy Y , przy czym wartości przez nie przyjmowane dla konkretnego $s \in S$ są zapisywane za pomocą analogicznych małych liter zapisanych kursywą (np. x lub y). W zapisie formalnym zmienna losowa to funkcja $X : S \rightarrow \mathbb{R}$, dla której $X(s) = x$.

A.2.1. Dyskretne zmienne losowe

W przypadku zmiennych losowych możemy zadawać pytania o prawdopodobieństwo przyjęcia przez nie określonych wartości w wyniku przeprowadzenia eksperymentu. Przypuśćmy, na przykład, że naszym eksperymentem jest rzut monetą. Mamy tutaj dwa wyniki: orła i reszkę. Przestrzeń próbowa to $S = \{\text{orzeł, reszka}\}$. Założymy, że moneta jest symetryczna, czyli że prawdopodobieństwo uzyskania orła jest takie samo jak reszki. Ponieważ zdarzenia te są rozłączne i stanowią podział S , każde z nich ma prawdopodobieństwo 0,5.



Rysunek A.2. Zmienna losowa X jest funkcją, która odwzorowuje wyniki z S na liczby rzeczywiste

Formalnie musielibyśmy zdefiniować zdarzenia $O = \{\text{orzeł}\}$ i $R = \{\text{reszka}\}$ i obliczyć $P(O) = P(R) = 0,5$. Zdefiniujmy teraz zmienną losową X , która odwzorowuje wynik „orzeł” na liczbę rzeczywistą 1,0, a wynik „reszka” na liczbę rzeczywistą 0,0. Możemy zapytać: „Jakie jest prawdopodobieństwo, że X przyjmie wartość 1,0?”. Jest to równoważne z pytaniem: „Jakie jest prawdopodobieństwo zdarzenia A , które zawiera wszystkie wyniki s z S takie, że $X(s) = 1$?”. W naszym przypadku jest tylko jeden wynik w S , dla którego X daje 1,0 – orzeł. Stąd prawdopodobieństwo, że X przyjmie wartość 1 wynosi $P(O)$, czyli 0,5.

Formalnie prawdopodobieństwo, że zmienna losowa X przyjmuje wartość x jest zapisywane jako $P(X(s) = x)$, często jednak wygodniej jest to zanotować jako $P(X = x)$. Możemy jeszcze bardziej uprościć notację i pisać po prostu $p(x)$. (Zwróćmy uwagę na użycie małego p w tym skrótnym zapisie – taki zapis jest powszechny). Łatwo jest zapomnieć przy pytaniu o, powiedzmy, wartość $p(1)$, że stoi za tym cała struktura eksperymentu, obejmująca wyniki i zdarzenia, aksjomaty rachunku prawdopodobieństwa i zmienną losową. Należy pamiętać, że zmienna losowa X jest funkcją, a nie liczbą, a więc zawsze, gdy pytamy o prawdopodobieństwo w związku ze zmienną losową, chodzi nam o wynik działania funkcji na rezultacie pochodzący z przestrzeni próbkowej eksperymentu.

W powyższym przykładzie zmienna losowa X może przyjmować dwie wartości 0,0 i 1,0. Taka zmienna losowa pojawia się w wielu różnych rozważaniach. Wynika to z tego, że często interesuje nas podział wyników eksperymentu na dwie grupy (np. zdaliśmy lub oblaliśmy egzamin, trafiliśmy do celu lub spudłowaliśmy, wygraliśmy lub przegraliśmy grę). Zmienną losową, która odwzorowuje każdy wynik z S na 0 lub 1, nazywamy zmienną losową *Bernoulliego*. Jest ona przykładem *dyskretnej* zmiennej losowej – wartości przez nią przyjmowane nie pokrywają ciągłego przedziału. Przykłady *ciągłych* zmiennych losowych pojawią się jeszcze w dalszej części tego dodatku, teraz będziemy kontynuować omawianie zmiennych dyskretnych.

Przy rozważaniu zmiennych losowych Bernoulliego możemy użyć symbolu p na oznaczenie prawdopodobieństwa $P(X = 1)$. Wówczas drugie prawdopodobieństwo $P(X = 0)$ wynosi $1 - p$. Zmienna losowa Bernoulliego jest zatem jednoznacznie określona przez jeden parametr p oznaczający prawdopodobieństwo.

stwo *sukcesu*, niezależnie od tego, jak ten sukces jest określony w kontekście eksperymentu.

W sposób naturalny pojawia się potrzeba rozważania eksperymentów opisywanych przez ciąg zmiennych losowych Bernoulliego. Wykonajmy, na przykład, n strzałów do celu. Przy każdym strzale prawdopodobieństwo trafienia (sukcesu) wynosi p . Jeśli założymy, że każdy strzał jest niezależny od pozostałych (tzn. wynik żadnego strzału nie ma wpływu na prawdopodobieństwo sukcesu przy następnych), to eksperiment ten jest opisywany przez zmienną losową *dwumianową*, która może przyjmować wartości całkowite od 0 do n . Oznacza to, że dla zmiennej losowej dwumianowej możemy pytać o prawdopodobieństwo $P(X = 5)$ w doświadczeniu z 10 strzałami z prawdopodobieństwem trafienia do celu (sukcesu) równym 0,4, przy założeniu, że każdy strzał jest niezależny od pozostałych. Określenie tego prawdopodobieństwa wymaga rozważenia wszystkich wyników eksperymentu, w których było dokładnie 5 sukcesów. Taki eksperiment to ciąg 10 strzałów do celu. Jego wyniki możemy sobie wyobrażać jako ciągi SSSPSPPSPP, przy czym S oznacza sukces, a P – porażkę. Musimy określić prawdopodobieństwo zdarzenia, które obejmuje wszystkie wyniki z dokładnie 5 sukcesami.

Na podstawie założenia, że każdy strzał jest niezależny, wiemy, iż prawdopodobieństwo dla, powiedzmy, SSSPSPPSPP to iloczyn $p(S)$, $p(S)$, $p(S)$, $p(P)$, $p(S)$, $p(P)$, $p(S)$, $p(P)$ i $p(P)$. A ponieważ $p(S)$ jest dla każdego strzału stałe równe p (tak założyliśmy), więc opisany iloczyn upraszcza się do $p^5(1-p)^5$. Ale jest to tylko jeden z możliwych sposobów uzyskania 5 trafień (sukcesów) w 10 strzałach. Istnieją inne ciągi, które również opisują 5 trafień: SSSSSPPPPP, SSSSPPPPPS itd. Skoro nie interesuje nas kolejność trafień, a jedynie ich liczba, mamy do czynienia z *kombinacjami*, a nie *permutacjami*. Liczba kombinacji 5 trafień w 10 strzałach wynosi $\frac{10!}{(5!)(5!)} = 252$. Stąd prawdopodobieństwo uzyskania 5 trafień przy 10 strzałach przy danym prawdopodobieństwie sukcesu $p = 0,4$ wynosi

$$P(X = 5) = 252(0,4)^5(0,6)^5 = 0,200658$$

Powtórzmy – zmienna losowa dwumianowa opisuje sytuację, gdy liczba prób n jest ustalona przed rozpoczęciem eksperymentu, a prawdopodobieństwo sukcesu jest stałe i wynosi p przy każdej próbie. Co więcej, każda próba jest niezależna od pozostałych, a my chcemy określić prawdopodobieństwo uzyskania pewnej liczby sukcesów (zwykle oznaczanej k) we wspomnianych n próbach. Jeśli liczba prób nie jest z góry ustalona lub jeśli prawdopodobieństwo sukcesu nie jest stałe, lub jeśli próby nie są niezależne, to nie mamy do czynienia ze zmienną losową dwumianową.

Zmienne losowe są używane do modelowania procesów ze świata rzeczywistego. Jak w wypadku wszystkich modeli, wierność opisu za pomocą zmiennej losowej zależy od tego, na ile dobrze są spełnione jej założenia. Zmienna losowa Bernoulliego z $p = 0,5$ to dobry model dla typowego rzutu monetą. Zmienna losowa dwumianowa dobrze modeluje sytuację, gdy chcemy określić prawdopodobieństwo sytuacji, że 3 osoby wykażą reakcję obronną na szczepionkę, która została podana 1000 ludziom, przy założeniu, że szansa pojawienia się takiej

reakcji wynosi p i jest stała dla każdej osoby. Czy jednak jest to dobry model przy pytaniu o prawdopodobieństwo, że student zda, powiedzmy, dwa lub trzy z czterech swoich sprawdzianów w czasie semestru? Moglibyśmy przyjąć, że $n = 4$, a następnie próbować obliczyć $P(X = 2 \text{ lub } 3)$. Kłopot jednak w tym, że nie wiemy, jakie jest prawdopodobieństwo zdania jednego testu przez studenta i, co więcej, nie wiemy, czy to prawdopodobieństwo jest dla każdego testu takie samo. Każdy test może mieć przecież inny stopień trudności, a i student będzie się uczył w trakcie zajęć. Stopień odpowiedniości między układem w świecie rzeczywistym a zmienną losową dwumianową może być lub może nie być wystarczający. Kwestia ta wymaga subiektywnej oceny. Im większe jest nasze doświadczenie w modelowaniu rzeczywistych procesów, tym lepsza jest ocena stosowności tej metody.

Przypuśćmy, że chcemy opracować model opisujący liczbę meteorów widzianych na niebie w ciągu godziny. Od razu wiadomo, że odpowiednia zmienna nie będzie dwumianowa – nie mamy tu określonej liczby prób. Wynikiem eksperymentu może być dowolna liczba całkowita od zera do, teoretycznie, nieskończoności. Przy tego typu zjawiskach jedną z możliwych do wybrania zmiennych losowych jest zmienna losowa *Poissona*. Opisuje ona sytuację, w której zdarzenia zachodzą z pewną intensywnością, oznaczaną λ , a liczba zdarzeń $x \in \{0, \dots, \infty\}$. Zmienna losowa *Poissona* opisuje prawdopodobieństwo zaobserwowania x zajść jakiegoś procesu przy założonym parametrze intensywności λ jako

$$P(X = x) = \frac{e^{-\lambda} \lambda^x}{x!}$$

Podobnie jak zmienne losowe Bernoulliego i dwumianowa, zmienna Poissona jest dyskretna – przyjmuje tylko nieujemne wartości całkowite.

Warto teraz wprowadzić dodatkową własność zmiennych losowych. Z akcjomatów prawdopodobieństwa wynika, że prawdopodobieństwo, iż zmienna losowa X przyjmie określoną wartość, musi wynosić co najmniej 0,0 i nie może być większe niż 1,0. Tak więc prawdopodobieństwo, że zmienna losowa Poissona przyjmie dowolną wartość z zakresu od zera do nieskończoności, wynosi 1,0. Innymi słowy, $P(X = 0) + P(X = 1) + \dots = 1,0$. Sprawdźmy, że tak jest w rzeczywistości.

Rozwiniecie e^λ w nieskończony szereg Maclaurina daje

$$e^\lambda = 1 + \lambda + \frac{\lambda^2}{2!} + \frac{\lambda^3}{3!} + \dots = \sum_{x=0}^{\infty} \frac{\lambda^x}{x!}$$

a zatem

$$\sum_{x=0}^{\infty} \frac{e^{-\lambda} \lambda^x}{x!} = e^{-\lambda} e^\lambda = 1$$

Jest wiele innych dyskretnych zmiennych losowych, które należy znać, nie są one jednak tematem tego dodatku. Pełniejsze ich omówienie znajduje się w [101, 100].

A.2.2. Ciągłe zmienne losowe

Ciągłe zmienne losowe, w odróżnieniu od dyskretnych zmiennych losowych, mogą przyjmować dowolną wartość z ciągłego przedziału, choćby ten przedział obejmował wszystkie liczby rzeczywiste. Musimy w tym momencie natychmiast przejść z myślenia o oddzielnych dyskretnych masach prawdopodobieństwa na myślenie o *gęstości*. Jeśli zmienna losowa jest ciągła, to prawdopodobieństwo, że przyjmie ona dowolną konkretną wartość, wynosi dokładnie zero. O dodatkowych prawdopodobieństwach możemy mówić tylko wówczas, gdy interesują nas zakresy wartości. A zatem prawdopodobieństwo, że ciągła zmienna losowa X przyjmie wartość z przedziału $[a, b]$, jest określone jako

$$P(a \leq X \leq b) = \int_a^b f(x)dx$$

Funkcja $f(x)$ w tym wzorze jest *funkcją gęstości prawdopodobieństwa* zmiennej losowej X . Analogiczne pojęcie dla dyskretnych zmiennych losowych to *funkcja masy prawdopodobieństwa* – w wypadku takich zmiennych gęstość prawdopodobieństwa jest skoncentrowana w mierzalne masy w pojedynczych punktach. Ponieważ prawdopodobieństwo uzyskania dowolnego wyniku z przestrzeni próbkowej S wynosi 1,0, więc prawdopodobieństwo, że ciągła zmienna losowa przyjmie wartość między $-\infty$ a ∞ , musi być 1,0. Dodatkowo wymagamy, żeby funkcja gęstości prawdopodobieństwa była zawsze nieujemna.

Być może, najprostszym przykładem ciągłej zmiennej losowej jest zmienna losowa *jednostajna* o funkcji gęstości prawdopodobieństwa

$$f(x) = \frac{1}{b-a}, \quad a \leq x \leq b$$

Potrzebne jest tu wyjaśnienie; czasami ta funkcja jest definiowana z użyciem dodatkowego warunku, w którym określamy, że $f(x) = 0$ poza przedziałem $[a, b]$. Na ogół przyjmujemy, że jeśli jakaś wartość funkcji gęstości nie jest określona, to wynosi zero. W tym przypadku łatwo jest sprawdzić, że $f(x)$ jest poprawną funkcją gęstości prawdopodobieństwa: 1) jest zawsze nieujemna i 2) jej całka od $-\infty$ do ∞ wynosi 1. Zmienna losowa jednostajna jest podstawą wielu innych zmiennych losowych – wykonując szereg operacji, można jednostajną zmienną losową przekształcić w inne zmienne losowe, użyteczne w różnych kontekstach.

Ciągłą zmienną losową o najszerzym zakresie zastosowania jest zmienna losowa Gaussa (znana także jako zmienna *normalna*) o funkcji gęstości prawdopodobieństwa

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

z x przebiegającym wszystkie liczby rzeczywiste. Rozkład ten nazywamy „normalnym”, gdyż odpowiada temu, co normalnie się pojawia przy mierzeniu różnych wielkości naturalnych: wysokości człowieka, długości łodygi różnych kwiatów, odległości między oczami muszki owocowej. Fakt ten ma swoje teoretyczne uzasadnienie w postaci *centralnego twierdzenia granicznego*, które omówimy

w dalszej części tego dodatku. Zmienna losowa normalna może także służyć do przybliżania innych zmiennych losowych – nawet, jak niedługo zobaczymy, gdy są one dyskretne. Zwróćmy uwagę, że zmienna losowa normalna jest określona za pomocą dwóch parametrów: μ i σ . Oba mają swoje fizyczne interpretacje: μ oznacza położenie osi symetrii zmiennej (a także *wartość oczekiwana* – zobacz dalej), σ jest miarą stopnia rozrzutu (rozproszenia) funkcji (a także *odchyleniem standardowym* – zobacz dalej).

Istnieje wiele innych przydatnych ciągłych zmiennych losowych. Zmienna losowa *wykładnicza*

$$f(x) = \lambda e^{-\lambda x} \quad x > 0$$

jest często używana do modelowania czasu życia elementów składowych lub sprzętu (np. żarówek). Zwróćmy uwagę, że zmienna ta jest niezerowa tylko dla dodatnich x oraz że λ musi być dodatnia. Zmienna losowa *chi-kwadrat*

$$f(x) = \frac{1}{\sqrt{2\pi}} x^{-\frac{1}{2}} e^{-\frac{x}{2}} \quad x > 0$$

opisuje, co wyniknie, kiedy weźmiesz kwadrat zmiennej losowej normalnej o wartości średniej zero i wariancji jeden. Jest ona czasami używana przy modelowaniu błędów, o których zakłada się, że mają rozkład Gaussa, a nas interesuje opis w postaci kwadratu błędu. Zmienna losowa chi-kwadrat w rzeczywistości może być używana do opisu sumy ciągu zmiennych losowych Gaussa. W tym wypadku zmienna ta jest parametryzowana symbolem ν , oznaczającym tak zwane *stopnie swobody* tej zmiennej (zob. [101], gdzie znajdziesz więcej o parametryzowanej zmiennej losowej chi-kwadrat). Zarówno zmienna wykładnicza, jak i zmienna chi-kwadrat są specjalnymi przypadkami tak zwanych zmiennych losowych *Gamma* (nazwa wynika z użycia funkcji Gamma w ich definicji); dokładniejszy opis tych zmiennych nie jest tematem tego dodatku.

Kolejną ważną ciągłą zmienną losową jest zmienna losowa *Cauchy'ego*, która ma funkcję gęstości prawdopodobieństwa bardzo podobną do zmiennej Gaussa, z tą jednak różnicą, że ma o wiele grubsze ogony (części na wykresie funkcji skrajnie po prawej i po lewej stronie):

$$f(x) = \frac{a}{\pi(x^2 + a^2)} \quad a > 0 \quad -\infty < x < \infty$$

Zmienna losowa Cauchy'ego dla $a = 1$ to specjalny przypadek tak zwanego rozkładu t , który będzie używany później w tym dodatku przy omawianiu testowania hipotez statystycznych.

A.3. Statystyki opisowe zmiennych losowych

Bardzo często przydaje się scharakteryzowanie zmiennych losowych za pomocą ich własności statystycznych. Jaka jest, na przykład, najbardziej prawdopodobna wartość pewnej zmiennej losowej? Wielkość ta jest nazywana *modą* zmiennej. W przypadku ciągłej zmiennej losowej wartość ta odpowiada maksimum funkcji gęstości prawdopodobieństwa. W pewnych sytuacjach zmienna losowa może

mieć więcej niż jedną modę, tzn. mieć więcej niż jedną wartość pojawiającą się z maksymalnym prawdopodobieństwem lub też maksymalizującą funkcję gęstości prawdopodobieństwa. W rzeczywistości może być nawet nieskończonie wiele takich wartości, jak w przypadku zmiennej losowej jednostajnej.

Inną przydatną statystyką opisową jest *medianą*, wskazującą punkt środkowy, dla którego 50% masy prawdopodobieństwa lub jego gęstości znajduje się poniżej (i powyżej) tej wartości. W kategoriach matematycznych mediana jest wartością x , dla której $P(X \leq x) = P(X \geq x) = 0.5$. Dla niektórych dyskretnych zmiennych losowych może istnieć więcej niż jedna taka wartość. Na przykład taka sytuacja zachodzi dla zmiennej losowej Bernoulliego o parametrze $p = 0.5$. Każda liczba większa od zera i mniejsza od jedynki spełnia podany warunek. Jedna z konwencji przyjmuje, że w takiej sytuacji za medianę przyjmuje się punkt środkowy powstałego zakresu wartości. Takie rozwiązanie nie jest jednak pozbawione wad.

Najbardziej powszechnie używaną statystyką opisową charakteryzującą zmienną losową jest jej wartość *średnia*, inaczej zwana *wartością oczekiwana*. Średnia to w istocie przeciętna wartość ważona, liczona z uwzględnieniem wartości prawdopodobieństwa przypisanego do poszczególnych wartości. Dla dyskretnych zmiennych losowych wartość oczekiwana jest określona jako

$$E(X) = \sum_{x \in D} xP(X = x)$$

przy czym D jest dziedziną X . Konkretnie, dla zmiennej losowej Bernoulliego

$$E(X) = 0 \cdot P(X = 0) + 1 \cdot P(X = 1) = 0 \cdot (1 - p) + 1 \cdot p = p$$

Z pomocą podobnych obliczeń można sprawdzić, że wartość oczekiwana zmiennej losowej dwumianowej wynosi np , a wartość oczekiwana zmiennej losowej Poissona wynosi λ .

Ciągłym odpowiednikiem dyskretnej wartości oczekiwanej jest całka

$$E(X) = \int_{-\infty}^{\infty} xf(x)dx$$

Dla zmiennej losowej jednostajnej, której gęstość przyjmuje dodatnie wartości z odcinka $[a, b]$, otrzymujemy

$$\begin{aligned} E(X) &= \int_{-\infty}^{\infty} x \frac{1}{b-a} dx = \int_a^b x \frac{1}{b-a} dx = \frac{1}{b-a} \frac{x^2}{2} \Big|_a^b = \frac{1}{b-a} \frac{b^2 - a^2}{2} = \\ &= \frac{b+a}{2} \end{aligned}$$

co zgadza się z naszą intuicją wynikającą z rysunku. Wartość oczekiwana zmiennej losowej Gaussa wynosi μ , wartość oczekiwana zmiennej losowej wykładniczej wynosi λ^{-1} , a wartość oczekiwana zmiennej losowej chi-kwadrat to ν – liczba stopni swobody tej zmiennej. Zmienna losowa Cauchy'ego jest patologicznym

przypadkiem. Chociaż funkcja gęstości prawdopodobieństwa jest tutaj symetryczna, wartość oczekiwana nie istnieje. Jest tak, gdyż całka

$$\int_{-\infty}^{\infty} x \left(\frac{a}{\pi(x^2 + a^2)} \right) dx$$

jest rozbieżna dla $a > 0$. Warto zapamiętać, że nie każda zmienna losowa ma dobrze określoną wartość oczekiwana oraz że nie wystarczy „rzucić okiem” na funkcję gęstości prawdopodobieństwa, żeby określić wartość średnią związaną z nią zmiennej losowej.

Moda, mediana i wartość średnia są wszystkimi miarami *tendencji centralnej*, czyli statystykami dającymi informacje na temat tego, czego należy się spodziewać, próbując zmienną losową (co, jak pamiętamy, oznacza, że został określony eksperyment, pojawił się wynik, a jego wartość została przekształcona za pomocą zmiennej losowej na wartość rzeczywistą). Te miary tendencji centralnej, choć bardzo przydatne, nie opisują niepewności nieodłącznie związanej z przewidywaniami. Oprócz tej miary jest ona często przydatna do opisania, w jakim stopniu masa lub gęstość prawdopodobieństwa jest rozproszona w całym zakresie możliwych wartości. Na przykład zmienna jednostajna na odcinku $[-a, a]$ oraz zmienna Gaussa dla $\mu = 0$ i $\sigma = a$ mają tę samą wartość oczekiwana, jednak prawdopodobieństwo uzyskania wartości w odległości nie większej niż a jednostek od średniej (zera) jest w każdym z tych przypadków inne (tzn. wynosi 1,0 dla zmiennej jednostajnej i około 0,68 dla zmiennej Gaussa).

Jednym ze sposobów opisu zmienności zmiennej losowej jest podanie jej *wariancji*, która dla dyskretnych zmiennych losowych jest zdefiniowana jako

$$V(X) = \sum_{x \in D} (x - E(X))^2 P(X = x)$$

a dla ciągłych zmiennych losowych

$$V(X) = \int_{-\infty}^{\infty} (x - E(X))^2 f(x) dx$$

W istocie jest to uśredniony kwadrat odległości wartości zmiennej w danym punkcie od wartości oczekiwanej tej zmiennej. Uśrednienie jest przy tym brane z uwzględnieniem wagi prawdopodobieństwa danego punktu (lub w przypadku ciągłym z uwzględnieniem prawdopodobieństwa nieskończonym małego odcinka dx). Możemy też, zamiast pracować z jednostkami błędu podniesionymi do kwadratu, wziąć pierwiastek kwadratowy wariancji i uzyskać wielkość zwaną *odchyleniem standardowym*.

Wariancja zmiennej losowej Bernoulliego wynosi

$$\begin{aligned} V(X) &= (0 - p)^2(1 - p) + (1 - p)^2p = p^2(1 - p) + (1 - p)^2p = \\ &= (1 - p)p(p + 1 - p) = (1 - p)p \end{aligned}$$

Wartość $(1 - p)$ jest często oznaczana symbolem q , zatem wariancję zmiennej losowej Bernoulliego można zapisać jako pq . Wariancje pozostałych omawianych wcześniej zmiennych losowych wyglądają tak:

dwumianowej: npq

Poissona: λ

jednostajnej: $(b - a)^2/12$

Gaussa: σ^2

wykładniczej: λ^{-2}

chi-kwadrat: 2ν

Cauchy'ego: nieokreślona

Ponieważ zmienna losowa Gaussa jest podstawowa w rozważaniach związanych z rachunkiem prawdopodobieństwa i statystyką (jak zaraz pokażemy), często na oznaczenie średniej i wariancji zmiennej losowej używa się, odpowiednio, symboli μ i σ^2 .

Wariancję możemy łatwiej obliczyć, stosując wzór

$$V(X) = E(X^2) - E(X)^2$$

przy czym wartość oczekiwana w sytuacji, gdy jakaś funkcja, powiedzmy g , została zastosowana do zmiennej X , jest obliczana zgodnie ze wzorem

$$E(g(X)) = \int_{-\infty}^{\infty} g(x)f(x)dx$$

W ten sposób uzyskujemy

$$E(X^2) = \int_{-\infty}^{\infty} x^2 f(x)dx$$

Mogliśmy brać pod uwagę wartość oczekiwana X^2 , możemy też rozważać wartości oczekiwane X^3 , X^4 itd. Liczby te są nazywane *momentami* zmiennej losowej – $E(X^n)$ jest momentem zmiennej X rzędu n . Możemy też rozważać *moment centralny* rzędu n – $E((X - E(X))^n)$. Istnieją interpretacje geometryczne momentów centralnych trzeciego i czwartego rzędu (nazywane, odpowiednio, *skośnością* i *kurtozą*). Wskazują one, na ile funkcja gęstości prawdopodobieństwa jest przekrzywiona w lewą lub prawą stronę od średniej, oraz jak bardzo ta funkcja jest spiczasta lub spłaszczona. Obszerniejszy opis momentów, ich interpretacji oraz funkcji generujących momentów można znaleźć w [100].

A.4. Twierdzenia graniczne i nierówności rachunku prawdopodobieństwa

Jednym z najważniejszych twierdzeń rachunku prawdopodobieństwa jest *centralne twierdzenie graniczne*. Istnieje kilka wersji tego twierdzenia; w najbardziej podstawowej wersji twierdzenie to brzmi:

Twierdzenie: Jeśli X_1, X_2, \dots, X_n są niezależnymi zmiennymi losowymi o identycznych rozkładach oraz mają skończone wartości oczekiwane i wariancje, to znormalizowana suma $(S_n - n\mu)/(\sigma\sqrt{n})$, gdzie $S_n = X_1 + X_2 + \dots + X_n$, przy $n \rightarrow \infty$, jest zbieżna do zmiennej losowej Gaussa o wartości oczekiwanej zero i wariancji jeden.

Bardziej opisowo – pod warunkiem że dodamy do siebie wystarczająco dużo niezależnych zmiennych losowych o identycznym rozkładzie i mających skończone średnie i wariancje, ich suma będzie dążyć do zmiennej losowej Gaussa.

To jest naprawdę fascynujący i potężny wynik. Jeśli weźmiemy sumę wystarczająco dużej liczby, powiedzmy, zmiennych losowych Poissona, z których każda ma parametr intensywności równy λ , to suma ta będzie miała funkcję (masy) prawdopodobieństwa, którą można dobrze przybliżyć funkcją odpowiadającą zmiennej losowej Gaussa o średniej $n\lambda$ i wariancji $n\lambda$. Suma wystarczająco dużej liczby zmiennych losowych dwumianowych jest w przybliżeniu gaussowska ze średnią np i wariancją npq . Zauważmy, że w obu tych wypadkach wartość sumy poissonowskich lub dwumianowych zmiennych losowych nie może być ujemna i może przyjmować tylko wartości całkowite. Mimo tego podobieństwo do zmiennej losowej Gaussa może być bardzo duże.

Podobnie, suma zmiennych losowych wykładniczych, zmiennych losowych chi-kwadrat lub nawet zmiennych losowych jednostajnych w granicy zbliża się do rozkładu Gaussa. Suma zmiennych losowych Cauchy'ego nie ma tej własności – nie zbliża się do rozkładu Gaussa, gdyż zmienna ta nie ma skończonej średniej ani wariancji. Dowód wspomnianego twierdzenia opisano w [100].

Inne podobne twierdzenie nosi nazwę *prawa wielkich liczb*. Jeśli X_1, \dots, X_n są niezależnymi zmiennymi losowymi, z których każda ma skończoną średnią i wariancję, to suma $S_n = X_1 + \dots + X_n$ spełnia zależność

$$\lim_{n \rightarrow \infty} P\left(\left|\frac{S_n}{n} - E(X)\right| \geq \epsilon\right) = 0$$

dla $\epsilon > 0$. Podchodząc do tego twierdzenia od innej strony, zastanówmy się najpierw nad zmienną losową Y równą zmiennej losowej S_n podzielonej przez n . Dojdziemy po chwili do tego, że prawdopodobieństwo, iż wielkość różnicy między Y a jej wartością oczekwaną będzie większa niż dowolna ustalona wartość dodatnia, w granicy wynosi zero. Im więcej dodajemy zmiennych losowych, tym mniejsza jest wariancja znormalizowanej ich sumy. Istnieją dwie wersje tego twierdzenia: *słaba* i *mocna*, które mówią o różnych typach zbieżności (zbieżności w sensie prawdopodobieństwa i z prawdopodobieństwem 1). Więcej o rodzajach zbieżności stochastycznej można dowiedzieć się z [345].

Centralne twierdzenie graniczne daje narzędzie służące do przybliżania rozkładu sumy odpowiedniej liczby zmiennych losowych w sytuacji, gdy niewiele wiadomo o nich samych. Oprócz tego istnieją twierdzenia, które przy równej słabych lub nawet przy braku założeń na temat rozkładu podają ograniczenia na prawdopodobieństwo, że zmienna losowa odbiega od swojej wartości śred-

niej. Jednym z ważnych wyników, który nie nakłada żadnych ograniczeń oprócz wymagania skończości średniej i wariancji, jest *nierówność Czebyszewa*:

$$P(|X - E(X)| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}$$

lub po przyjęciu $\epsilon = k\sigma$

$$P(|X - E(X)| \geq k\sigma) \leq \frac{1}{k^2}$$

Wyrażając to słowami, prawdopodobieństwo, że próbka ze zmiennej losowej będzie odległa od swojej wartości średniej o co najmniej k odchyłeń standardowych, jest zawsze mniejsze lub równe $1/k^2$. Jeśli coś wiemy na temat rozkładu zmiennej losowej, którą się zajmujemy, ograniczenie to może być brane na wyróst. Na przykład dla zmiennej losowej Gaussa prawdopodobieństwo uzyskania wartości, która jest odległa od średniej o więcej niż dwukrotna wartość odchylenia standardowego, wynosi około 0,05. Nierówność Czebyszewa zapewnia, że to prawdopodobieństwo jest mniejsze niż 0,25. Gdy na temat rozkładu wiemy niewiele, nierówność Czebyszewa może dać bardzo użyteczne informacje.

A.5. Dodawanie zmiennych losowych

Centralne twierdzenie graniczne i prawo wielkich liczb dają wskazówki co do granic sum zmiennych losowych, często jednak ważne jest, żeby wiedzieć, jakie są rozkłady sum mniejszego zbioru zmiennych losowych. Należy sobie zdawać sprawę z tego, że gęstość zmiennej losowej $Y = X_1 + X_2$ nie jest sumą funkcji gęstości prawdopodobieństwa zmiennych X_1 i X_2 , a raczej sumą wartości uzyskanych przez próbkowanie za pomocą X_1 i X_2 . Gdy X_1 i X_2 są niezależne, tzn. gdy ich łączna funkcja gęstości spełnia $f(x_1, x_2) = f_{X_1}(x_1)f_{X_2}(x_2)$, wówczas funkcja $g(y)$ gęstości zmiennej losowej Y jest opisana za pomocą następującej całki splotu:

$$g(y) = \int_{-\infty}^{\infty} f_{X_1}(y - z)f_{X_2}(z)dz \quad -\infty < y < \infty$$

Dla wielu często spotykanych rodzajów zmiennych losowych warto znać podane poniżej zależności.

- Suma dwóch niezależnych zmiennych losowych Gaussa o średnich μ_1 i μ_2 oraz wariancjach σ_1^2 i σ_2^2 jest także zmienną losową Gaussa i ma średnią $\mu_1 + \mu_2$ oraz wariancję $\sigma_1^2 + \sigma_2^2$.
- Suma dwóch niezależnych zmiennych losowych Poissona o parametrach intensywności λ_1 i λ_2 jest zmienną losową Poissona o intensywności $\lambda_1 + \lambda_2$.
- Suma dwóch niezależnych zmiennych losowych wykładniczych o parametrach λ_1 i λ_2 jest zmienną losową wykładniczą o parametrze $\lambda_1\lambda_2/(\lambda_1 + \lambda_2)$.

- Suma dwóch niezależnych zmiennych losowych chi-kwadrat o stopniach swobody ν_1 i ν_2 jest zmienną losową chi-kwadrat o stopniach swobody $\nu_1 + \nu_2$.

Nie należy jednak dać się zwieść wrażeniu, że po dodaniu dwóch zmiennych losowych o tym samym rozkładzie zawsze w wyniku otrzymujemy zmienną losową o tym rozkładzie. Jeśli bowiem, na przykład, dodamy dwie niezależne zmienne losowe o rozkładzie jednostajnym i mające dodatnią gęstość w przedziale $[0, 1]$, to wynikiem będzie zmienna losowa o trójkątnej funkcji gęstości prawdopodobieństwa rozciągającej się w przedziale $[0, 2]$.

A.6. Generowanie liczb losowych przez komputer

Wiele z zaprezentowanych w tej książce metod rozwiązywania problemów wymaga generowania liczb losowych zgodnie z określonym, podanym w opisie rozkładem. Ważne jest, żeby pamiętać, iż prawie wszystkie procedury generujące liczby losowe w rzeczywistości podają ciąg liczb *pseudolosowych*. Nie są one losowe w ścisłe matematycznym sensie, gdyż cały ciąg liczb jest tworzony w wyniku działania deterministycznej funkcji na wartości początkowej ziarna. Możemy jednak utworzyć ciągi liczb, których charakter wydaje się bardzo zbliżony do losowego.

Wiele środowisk programistycznych oferuje generatorów liczb losowych w postaci wbudowanych funkcji. Większość z nich ma funkcję, która daje w wyniku z jednostajnym rozkładem liczby z przedziału $[0, 1]$. Niektóre ze środowisk mają także funkcje dające liczby z rozkładem Gaussa przy danej średniej i wariancji. Warto wiedzieć, że nie każdy generator liczb losowych jest tak samo dobry. Najbardziej podstawową miarą jakości generatora jest długość cyklu: liczba wygenerowanych wartości, zanim generator zamknie cykl i zacznie powtarzać już podany ciąg. Im większa długość cyklu, tym lepiej. Jeśli nie jesteśmy pewni własności generatora liczb losowych, którego używamy, to należy wtedy rozważyć korzystanie z takiego, który jest dobrze udokumentowany. Wiele generatorów liczb losowych jest szczegółowo opisanych w książkach dotyczących programowania (np. [363]).

Jeśli możemy generować liczby losowe o rozkładzie jednostajnym zawarte w przedziale $[0, 1]$, to mamy podstawę do generowania wielu innych zmiennych losowych. Jeśli, na przykład, chcielibyśmy wygenerować zmienną losową Bernoulliego o parametrze p , to wywoływalibyśmy generator jednostajny dla $[0, 1]$ i sprawdzalibyśmy, czy wynik jest mniejszy niż p . Jeźliby tak było, uzyskalibyśmy w wyniku 1,0, w przeciwnym razie uzyskalibyśmy 0,0. Zmienne losowe dwumianowe możemy uzyskać, kolejno odwołując się do zmiennych o rozkładzie Bernoulliego.

Zmienne losowe jednostajne mogą być także wykorzystywane do tworzenia innych ważnych zmiennych losowych, takich jak zmienne losowe Gaussa czy wykładnicze. Stosując wzory

$$\sqrt{-2 \log U_1} \sin(2\pi U_2) \quad \text{oraz} \quad -\sqrt{-2 \log U_1} \cos(2\pi U_2)$$

możemy, używając dwóch niezależnych generatorów dających wartości U_1 i U_2 o rozkładzie jednostajnym, uzyskać dwa generatory liczb o rozkładzie Gaussa. Generator liczb o rozkładzie wykładniczym z parametrem λ możemy uzyskać, stosując przekształcenie $-(\log U_1)/\lambda$.

Na bazie zmiennej Gaussa możemy uzyskać zmienną chi-kwadrat. Jeśli liczbę uzyskaną ze zmiennej o rozkładzie Gaussa podniemy do kwadratu, to uzyskamy zmienną chi-kwadrat (o jednym stopniu swobody). Przypomnijmy, że suma dwóch niezależnych zmiennych losowych o rozkładzie chi-kwadrat jest zmienną o rozkładzie chi-kwadrat o liczbie stopni swobody równej sumie liczb stopni swobody zmiennych będących składnikami. W ten sposób możemy uzyskać dowolną zmienną losową chi-kwadrat, dodając do siebie wystarczająco dużo podniesionych do kwadratu parami niezależnych zmiennych losowych Gaussa.

Można także uzyskać zmienną losową Cauchy'ego. Wystarczy wziąć stosunek dwóch niezależnych zmiennych losowych o rozkładzie Gaussa. Innym sposobem jest zastosowanie wzoru

$$\operatorname{tg}[\pi(U_1 - 0,5)]$$

który zależy tylko od jednej zmiennej losowej o rozkładzie jednostajnym.

Pozostała nam jeszcze do rozważenia zmienna losowa Poissona. Wygenerowanie jej jest jednak odrobinę trudniejsze. Zalecana w [133] procedura polega na tym, że przyjmujemy dowolną wartość progową N określającą maksymalną liczbę, jaką będziemy mogli zaobserwować, a następnie określamy prawdopodobieństwa zajścia $0, 1, 2, \dots, N$ obserwowanych zdarzeń (robimy to, stosując tabele statystyczne lub obliczając je wprost z funkcji rozkładu masy prawdopodobieństwa). Następnie pobieramy liczbę z generatora o rozkładzie jednostajnym i określamy pierwszą liczbę z $0, 1, \dots, N$, dla której zapamiętane prawdopodobieństwo jest większe od liczby z generatora. W mało prawdopodobnej sytuacji, w której wszystkie $N + 1$ liczb przejrzano bez uzyskania łącznego prawdopodobieństwa większego od uzyskanego z rozkładu jednostajnego, przyjmujemy, że wynik jest N .

A.7. Szacowanie

Jedno z podstawowych zadań w statystyce polega na dogłębny badaniu parametrów populacji przez pobieranie próbek i szacowanie¹. Przypuśćmy, na przykład, że chcemy określić, jaki jest średni wzrost mężczyzn w Szwecji. Liczba mężczyzn, do których trzeba byłoby dotrzeć i których trzeba byłoby zmierzyć, jest jednak zbyt duża. W dodatku nie moglibyśmy być pewni, że sprawdziliśmy wszystkich (stanowi to problem przy każdym spisie ludności). Zamiast tego przypuśćmy, że możemy przyjąć, iż wartości wzrostu mężczyzn są rozłożone zgodnie z rozkładem Gaussa o pewnej średniej μ i wariancji σ^2 , przy czym

¹ Profesor Will Gersch z University of Hawaii w Manoa powiedział, że „celem badań statystycznych jest zrozumienie czegoś, a nie uzyskanie danych”.

parametry te nie są znane. W skrócie zapisujemy to jako $N(\mu, \sigma)$, przy czym N to skrót od „normalny”¹. Naszym zadaniem jest teraz oszacowanie tych parametrów na podstawie próbki złożonej z n osób. Jeśli wzrost każdej osoby jest próbką ze zmiennej losowej X o rozkładzie $N(\mu, \sigma)$, to możemy oszacować μ , biorąc średnią z próbki: $\bar{x} = (x_1 + \dots + x_n)/n^2$. Im większy jest rozmiar n próbki, tym na mocy prawa wielkich liczb większe jest prawdopodobieństwo, że nasze oszacowanie jest prawidłowe (za chwilę opowiemy jeszcze o mierzeniu zaufania do oszacowania).

W przypadku naszej próbki zmienna losowa \bar{X} jest sumą zmiennych losowych X_1, \dots, X_n podzieloną przez n . Zmienna \bar{X} ma oczywiście własną funkcję gęstości prawdopodobieństwa. W tym wypadku jest ona zmienną losową Gaussa, gdyż suma zmiennych losowych Gaussa jest zmienną losową Gaussa. Miałaby ona te same własności, gdyby n było wystarczająco duże i każde X_i dla $i = 1, \dots, n$ miało skończoną średnią i wariancję – w tym wypadku wynikałoby to z centralnego twierdzenia granicznego. Ponieważ \bar{X} jest zmienną losową, interesujące jest określenie jej wartości oczekiwanej, jeśli taka istnieje. Mamy tutaj

$$E(\bar{X}) = E\left(\frac{X_1 + \dots + X_n}{n}\right) = \frac{1}{n}(E(X_1) + \dots + E(X_n)) = \frac{1}{n}n\mu = \mu$$

Tak więc wartość oczekiwana średniej z próbki jest taka sama jak wartość średnia w całej populacji. Zauważmy, że nie korzystaliśmy tutaj z faktu, że populacja ma rozkład Gaussa. Własność ta zachodzi zatem niezależnie od rozkładu populacji (przy założeniu, że średnie są skończone). Jeśli estymator, taki jak tutaj \bar{X} , ma tę własność, że jego wartość oczekiwana jest równa parametrowi, który on szacuje, to jest nazywany *estymatorem nieobciążonym*.

Jeśli chodzi o szacowanie wariancji populacji, to wariancja próbki

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

jest estymatorem nieobciążonym wariancji σ^2 . Zwróćmy uwagę, że w mianowniku jest $n - 1$, a nie n ; umieszczenie n spowodowałoby zwiększenie oszacowania wariancji (powstałby *estymator obciążony*).

Inna metoda szacowania parametrów polega na poszukiwaniu wartości parametrów, które maksymalizują prawdopodobieństwo pojawienia się odpowiednich danych. Przy danym zbiorze obserwowanych wartości x_1, \dots, x_n i przyjętej jako punkt odniesienia pewnej zmiennej losowej, jeśli próbki są niezależne i mają identyczne rozkłady, to *funkcja wiarygodności* jest określona jako

$$L(\mathbf{x}, \theta) = f(x_1, \theta)f(x_2, \theta)\dots f(x_n, \theta)$$

¹ Możemy to też zapisać jako $N(\mu, \sigma^2)$. Należy w związku z tym zawsze jawnie wskazywać, czy drugi parametr to odchylenie standardowe czy wariancja.

² Pamiętajmy, że \bar{x} jest konkretną rzeczywiście uzyskaną próbką ze zmiennej losowej \bar{X} .

przy czym θ oznacza parametr (lub parametry), który kalibruje funkcję gęstości prawdopodobieństwa $f(x)$ (np. w przypadku zmiennej losowej wykładniczej jest to λ). Pomyśl polega na znalezieniu wartości θ , która maksymalizuje $L(\mathbf{x}, \theta)$. Typowa procedura stosowana dla wielu zmiennych losowych korzystających z funkcji wykładniczych (np. zmiennych losowych Gaussa i wykładniczych) polega na zastosowaniu logarytmu do funkcji wiarygodności, a następnie wzięciu pochodnej ze względu na parametr θ . Tak uzyskany wzór przyrównujemy do zera i rozwiązujemy ze względu na θ . W przypadku wielu parametrów szacowanie maksymalnej wiarygodności dla każdego z nich uzyskujemy przez obliczenie kolejnych pochodnych cząstkowych funkcji wiarygodności (lub jej logarytmu) ze względu na te parametry (parametrami tymi mogą być na przykład μ i σ w wypadku zmiennej losowej Gaussa).

Korzystanie z jednego oszacowania parametru (uzyskanego za pomocą zasad maksymalizacji wiarygodności lub innymi środkami) nie daje miary pewności co do dokładności oszacowania. Uzyskanie tego wymaga brania pod uwagę wariancji oszacowania. Jeśli możemy założyć, że populacja zachowuje się zgodnie z rozkładem Gaussa, to rozkład \bar{X} jest także gaussowski i ma średnią $E(X)$ oraz wariancję $V(X)/n$. Uzyskujemy

$$\frac{\bar{X} - E(X)}{\frac{\sigma}{\sqrt{n}}} \sim N(0, 1)$$

przy czym: $E(X)$ oznacza średnią populacji, σ – jej odchylenie standardowe, n – rozmiar próbki. We wzorze tym „~” czytamy jako „ma rozkład jak”, a $N(0, 1)$ identyfikuje standardową zmienną losową Gaussa. Rozważmy prawdopodobieństwo

$$P\left(\bar{X} - \frac{1,96\sigma}{\sqrt{n}} < E(X) < \bar{X} + \frac{1,96\sigma}{\sqrt{n}}\right)$$

Określa ono szansę, że \bar{X} da wartość, która znajduje się maksymalnie w odległości 1,96 standardowego odchylenia od swojej średniej. (Zauważmy, że σ/\sqrt{n} jest standardowym odchyleniem \bar{X}). Ponieważ \bar{X} jest zmienną losową Gaussa, prawdopodobieństwo to wynosi około 0,95. Przedział

$$\left[\bar{X} - \frac{1,96\sigma}{\sqrt{n}}, \bar{X} + \frac{1,96\sigma}{\sqrt{n}}\right]$$

nazywamy *przedziałem 95-procentowej ufności*. Stopień ufności związany z przedziałem może być zwiększyły lub zmniejszony przez zwiększenie lub zmniejszenie współczynnika skalującego 1,96. Interpretacja tego przedziału wygląda tak, że przy powtarzaniu wartości \bar{X} w granicy 95% uzyskanych przedziałów będzie obejmować wartość $E(X)$. Nie należy natomiast interpretować tego przedziału w ten sposób, że prawdopodobieństwo, iż $E(X)$ znajdzie się w jego granicach, wynosi 0,95. Subtelna różnica polega tutaj na tym, że to ostatnie stwierdzenie zakłada, że $E(X)$ jest zmienną losową. Tak jednak nie jest – wartość ta to parametr populacji. Zmienny jest natomiast sam przedział.

Jeśli początkowa populacja nie ma charakteru gaussowskiego, to z centralnego twierdzenia granicznego wiemy, że dla wystarczająco dużego n rozkład \bar{X}

będzie w przybliżeniu gaussowski i w związku z tym możliwe jest użycie tej samej formuły na określenie przedziału 95-procentowej ufności. Istnieje jednak pewien problem, który być może zauważysz, gdyż do tych obliczeń jest potrzebna wartość σ (lub równoważnie $V(X)$), a to jest kolejny parametr populacji, który prawie zawsze jest nieznany. Oznacza to, że musimy oszacować σ za pomocą odchylenia standardowego s naszej próbki. Stwarza to jednak potencjalnie problem, gdyż samo s jest zmienną losową, a rozkład

$$\frac{\bar{X} - E(X)}{\frac{s}{\sqrt{n}}}$$

nie jest rozkładem Gaussa. Na szczęście rozkład ten jest znany i opisujemy go za pomocą tak zwanego rozkładu t , który ma jeden parametr zwany *liczbą stopni swobody* równy $n - 1$. Rozkład t z jednym stopniem swobody jest rozkładem Cauchy'ego; przy nieskończonej liczbie stopni swobody rozkład ten staje się równoważny z rozkładem Gaussa. Kształt rozkładu t stopniowo przechodzi od tej pierwszej postaci do drugiej. Oznacza to, że aby zachować przedział wokół parametru na określonym poziomie ufności, musimy zmieniać współczynnik skalowania. W miarę zwiększania n współczynnik ten zbliża się do 1,96 (wartości używanej dla przypadku gaussowskiego). Dla małych n współczynnik skalowania może być dość duży. Na przykład dla $n = 2$ współczynnik skalowania dla poziomu ufności 95% dla $E(X)$ wynosi 12,706.

Im więcej próbek jest pobieranych, tym węższy jest przedział ufności. Dla bardzo dużego n możemy użyć wartości 1,96 i przyjmować, że uzyskane za pomocą s oszacowanie dla σ jest bardzo dobre. Przy małym n należy: 1) założyć, że populacja ma rozkład normalny i 2) skorzystać z rozkładu t do obliczenia odpowiedniego współczynnika skalowania. Dla rozkładu t , podobnie jak dla rozkładu Gaussa, możemy często w książkach o statystyce znaleźć tablice opisujące jego zachowanie (zob. [101]).

Jeśli dane nie są zgodne z rozkładem normalnym i n jest małe, to użycie przybliżenia gaussowskiego lub rozkładu t oznacza niespełnienie wymaganych założeń. Na ile to niespełnienie jest istotne, zależy od konkretnego problemu. W każdym wypadku wymaga to oceny, której jakość zależy od doświadczenia oceniającego.

Możliwe jest także uzyskanie przedziałów ufności dla wariancji populacji (a także dla innych parametrów). Więcej na ten temat można znaleźć w pozycjach [101, 100].

A.8. Statystyczne testowanie hipotez

Rachunek prawdopodobieństwa i statystyka udostępniają narzędzia do weryfikowania rozmaitych poglądów na temat świata rzeczywistego. W szczególności często zdarza się konieczność określenia, która z dwóch wykluczających się hipotez rzeczywiście zachodzi. Zwykle hipotezy te to $H_0 : \mu = \mu_0$ i $H_a : \mu \neq \mu_0$, przy czym: μ_0 jest pewną określoną wartością, μ – wartością średnią jakiejś wielkości w pewnej populacji lub procesie. H_0 jest określana jako *hipoteza zerowa*,

H_a jako *hipoteza alternatywna*. Zadanie polega na stwierdzeniu, czy przy dostępnych danych istnieją wystarczające przesłanki do odrzucenia H_0 . Możemy tutaj popełnić dwa rodzaje błędów. Pierwszy z nich, zwany błędem *pierwszego rodzaju* (typu I), pojawia się, kiedy odrzucamy H_0 w sytuacji, gdy w rzeczywistości hipoteza ta jest prawdziwa. Sytuację tę określamy też jako *fałszywy alarm*. Drugi rodzaj błędu, zwany błędem *drugiego rodzaju* (typu II), pojawia się, gdy nie odrzucimy H_0 , chociaż hipoteza będzie fałszywa. Sytuację tę określamy też jako *wynik fałszywie ujemny*. Na ogólnie w przypadku każdego testu statystycznego istnieje niezerowe prawdopodobieństwo i dla błędów pierwszego rodzaju, i dla błędów drugiego rodzaju. W praktyce przed zebraniem danych ustalamy prawdopodobieństwo błędu pierwszego rodzaju. Tradycyjnie przyjmujemy, że prawdopodobieństwo to (zwane *poziomem istotności* i zazwyczaj oznaczone α) wynosi 0,05. Prawidłowy dobór poziomu istotności zależy od zastosowania i kosztu popełnienia błędu pierwszego rodzaju. Po określaniu tego poziomu zwykle jedyne sposoby wpływu na prawdopodobieństwo błędu drugiego rodzaju sprowadzają się do wyboru odpowiedniego testu statystycznego i do określenia rozmiaru próby statystycznej.

Statystyczne testowanie hipotez zaczynamy na ogół z przekonaniem, że hipoteza zerowa jest prawdziwa. Na tej podstawie określamy rozkład statystyczny badanej statystyki, która szacuje interesujący nas parametr występujący w hipotezie zerowej. W przypadku zajmowania się wartością średnią w populacji lub w procesie używamy średniej z próbki \bar{x} jako estymatora μ . Jak już wiemy, \bar{X} jest nieobciążonym estymatorem μ i jeśli populacja początkowa ma rozkład Gaussa, to również taki rozkład ma \bar{X} . W związku z tym, jeśli przyjmiemy, że pewna populacja ma rozkład Gaussa o parametrach μ i σ , a interesuje nas sprawdzenie, czy μ jest równe jakiejś określonej wartości, to przy sprawdzaniu tego faktu odwołujemy się do rozkładu \bar{X} . Moglibyśmy, na przykład, chcieć znaleźć wystarczające dowody, żeby odrzucić pogląd, że poziom zanieczyszczeń w rzece jest większy niż powiedzmy 100 mg/litr przy założeniu, że rozkład zanieczyszczeń jest gaussowski. Zaczynamy od hipotezy zerowej $H_0 : \mu = 100$ przeciwko hipotezie alternatywnej $H_a : \mu \neq 100$. Jeśli znamy wartość σ , to przy założeniu hipotezy zerowej otrzymujemy, że rozkład \bar{X} to $N(100, \sigma/\sqrt{n})$. Powiedzmy, że pobieramy z rzeki n próbek wody. Możemy się zapytać, dla jakich wartości \bar{x} będziemy skłonni odrzucić hipotezę zerową? Odpowiedź jest następująca: będziemy chcieli to zrobić, gdy uzyskana wartość \bar{x} będzie dużo większa lub dużo mniejsza od 100. Jedynym sposobem zaś na określenie, co oznacza *dużo* w tym wypadku, jest wyrażenie tego za pomocą odchylenia standardowego \bar{x} : σ/\sqrt{n} .

Ponieważ przy naszej hipotezie zerowej $\bar{X} \sim N(100, \sigma/\sqrt{n})$, prawdopodobieństwo, iż \bar{x} znajdzie się w zakresie $100 \pm 1,96\sigma/\sqrt{n}$, wynosi 0,95. Możemy zatem przyjąć, że jeśli \bar{x} wypadnie poza ten zakres, to będziemy mogli powiedzieć, iż istnieje *statystycznie istotna* przesłanka do odrzucenia H_0 . Prawdziwość H_0 jest możliwa z prawdopodobieństwem 0,05. Podchodząc do tego trochę inaczej, możemy obliczyć wartość $(\bar{x} - \mu)/(\sigma/\sqrt{n})$ i odrzucić H_0 , o ile uzyskany wynik będzie większy od 1,96 lub mniejszy niż -1,96. Obszar wartości, dla których odrzucamy H_0 , nazywamy *obszarem krytycznym*. Jeśli nasza statystyka testo-

wa nie pojawi się w obrębie obszaru krytycznego, to *nie udaje się odrzucić H_0* . Warto nadmienić, że nigdy nie możemy *zaakceptować H_0* . Dla dowolnego rozmiaru próbki n istnieje nieskończenie wiele hipotez, dla których nie uda się obalić H_0 (np. gdyby \bar{x} wynosiło 100,4 przy $\sigma = 10$ oraz $n = 50$, nie udałoby się odrzucić $H_0 : \mu = 100$, ale także nie udałoby się odrzucić $H_0 : \mu = 100,1$). Żadna ilość dowodów nie może nas przekonać, że H_0 jest prawdziwe. Jeśli gdziekolwiek pojawia się rozumowanie, które *akceptuje hipotezę zerową*, to możemy bezpiecznie przyjąć, że autor nie rozumie, jak działa statystyczne testowanie hipotez, lub też, że niestarannie stosuje skróty myślowe.

Jeśli nieznane jest σ , podobnie jak było przy określaniu poziomu ufności, możemy za przybliżenie σ przyjąć odchylenie standardowe próbki s . Jeśli n jest wystarczająco duże, możemy znowu użyć gaussowskiego przybliżenia dla \bar{X} . Jeśli n jest małe, musimy przyjąć, że populacja, którą badamy, ma charakter gaussowski i przy modelowaniu \bar{X} skorzystać z rozkładu t o $n - 1$ stopniach swobody. Więcej na temat statystycznego testowania hipotez można znaleźć w [101, 100].

Jeśli nie możemy przyjąć założeń co do rozkładu badanej populacji, to musimy posłużyć się *nieparametrycznymi* metodami testowania hipotez. Opis tych metod nie jest tematem tego dodatku. Dokładny opis związanych z nimi metod znajduje się w [433].

A.9. Regresja liniowa

Pospolity, powracający problem w inżynierii polega na modelowaniu wartości zmiennej zależnej y w odniesieniu do m niezależnych zmiennych x_1, \dots, x_m . Często jako pierwszy jest stosowany model liniowy tej zależności. Bez utraty ogólności możemy rozważyć model

$$y = a + bx$$

Dla danego zbioru n obserwacji, y_i, x_i przy $i = 1, \dots, n$, zadanie polega na znalezieniu najlepszych wartości a i b . Zeby uczynić znaczenie słowa „najlepszych” precyzyjnym, przyjmiemy, że chodzi o dobranie takich a i b , aby wartość

$$\sum_{i=1}^n [y_i - (a + bx_i)]^2 \tag{A.1}$$

była minimalna. Ujmując to słowami, mamy wybrać takie a i b , żeby suma podniesionych do kwadratu różnic między wartością przewidzianą przez model a faktyczną wartością y_i była jak najmniejsza. Obliczenie pochodnych cząstkowych z (A.1) względem a i b oraz przymiarkanie ich do zera daje układ dwóch równań liniowych o dwóch niewiadomych:

$$\sum_{i=1}^n y_i = an + b \sum_{i=1}^n x_i$$

$$\sum_{i=1}^n x_i y_i = a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2$$

Uzyskujemy następujące rozwiązania dla a i b :

$$a = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n \sum x^2 - (\sum x)^2}, \quad b = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

Zauważmy, że aby uprościć wzory, opuściliśmy wszystkie indeksy. Odpowiednie obliczenia mogą polegać na tym, że najpierw znajdujemy b , a następnie otrzymujemy a ze wzoru

$$a = \bar{y} - b\bar{x}$$

Podobne podejście możemy zastosować, gdy mamy do czynienia z modelem liniowym o wielu zmiennych.

Jakość modelu liniowego możemy zmierzyć za pomocą zmienności zmiennej opisywanej przez model. Tak zwana *zmienność całkowita* zmiennej zależnej y to

$$\sum_{i=1}^n (y_i - \bar{y})^2$$

może zostać podzielona na dwie części:

$$\sum_{i=1}^n (y_i - \bar{y})^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \sum_{i=1}^n (\hat{y}_i - \bar{y})^2$$

przy czym \hat{y}_i jest oszacowaniem y_i . Pierwszą część nazywamy *zmiennością nie wyjaśnioną*, drugą zaś część *zmiennością wyjaśnioną*. Stosunek zmienności wyjaśnionej do zmienności całkowitej

$$\frac{\sum_{i=1}^n (\hat{y}_i - \bar{y})^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

daje procent zmienności danych, który jest wyjaśniony przez model. Wielkość ta jest oznaczana przez R^2 .

Związana z tym wielkość statystyczna r , będąca pierwiastkiem kwadratowym z R^2 , opatrzonym znakiem, mierzy stopień korelacji liniowej między zmienną zależną a oszacowaniem wynikającym z modelu. Wartość r równa ± 1 oznacza dokładną korelację liniową (przy czym korelacja ta jest albo dodatnia, albo ujemna). Wartość r możemy bezpośrednio obliczyć ze wzoru

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2} \sqrt{\sum (y - \bar{y})^2}}$$

przy czym sumy są wzięte względem wszystkich n próbek i ponownie pominięto indeksy przy zmiennych x i y .

Często wygodnie jest przyjąć model o postaci

$$y = a + bx + \epsilon$$

przy czym ϵ reprezentuje zewnętrzne czynniki zaszumiające. Jeśli czynniki te mają rozkład Gaussa o zerowej średniej i stałej wariancji, to możliwe staje się używanie statystycznego testowania hipotez do sprawdzenia, czy istnieją statystycznie znaczące przesłanki do odrzucenia przekonania, że ten współczynnik dla badanej niezależnej zmiennej losowej jest zerowy. Procedura ta, podobnie jak wiele innych szczegółów dotyczących projektowania liniowych i nieliniowych modeli danych, jest opisana w [101, 100].

A.10. Podsumowanie

Ten dodatek zawiera jedynie pobiczny przegląd ważnych pojęć z rachunku prawdopodobieństwa i statystyki, które zwykle poznajemy na pierwszych latach studiów na jednym z pierwszych wykładów z tych przedmiotów. Pozostałe tematy obejmują: 1) dystrybuantę, która jest całką funkcji gęstości (masy) prawdopodobieństwa, 2) testy hipotez statystycznych oparte na analizie wartości wariancji i wiarygodności, 3) bayesowskie szacowanie parametrów, 4) próbkowanie w badaniach reprezentacyjnych, 5) statystyczną kontrolę jakości, 6) projektowanie czynnikowe oraz 7) statystykę repróbkowania. Lista ta nie jest pełna. Zachęcamy do głębszego poznania struktury, teorii i zastosowania tych i innych tematów z rachunku prawdopodobieństwa i statystyki. Tym bardziej, że mogą one stanowić bezcenną podstawę niezbędną do zrozumienia sposobu działania i warunków stosowania wielu algorytmów statystycznych opisanych w głównej części książki.

B. Zadania i projekty

Między nauczaniem a badaniami naukowymi zachodzi taka relacja jak między spowiedzią a grzechem: jeśli się nie zgrzeszyło, to nie ma się nic do powiedzenia!

Anonim

Najlepszym sposobem, żeby nauczyć się rozwiązywać problemy, jest ich rozwiązywanie. Musisz eksperymentować z różnymi pomysłami, stosować je do zadań, które są Ci stawiane, i oceniać uzyskane wyniki. W dodatku tym podajemy pewne propozycje, które powinieneś rozważyć. Pierwszy ich zestaw jest przeznaczony dla tych, którzy świeżo wkroczyli w świat rozwiązywania problemów za pomocą komputerów, ale nawet jeśli jesteś bardziej zaawansowany, powinieneś mimo to zastanowić się nad odpowiedziami na zadawane tu pytania. Drugi zestaw propozycji obejmuje tematy projektów, jakie mógłbyś zrealizować. Mamy nadzieję, że jeśli bierzesz udział w formalnych zajęciach z heurystyki na poziomie uniwersyteckim, to są one zorganizowane tak, że uwzględniają projekty. Przy takim podejściu masz możliwość uczenia się na podstawie uzyskanych doświadczeń oraz podzielenia się nimi ze swoimi kolegami, a nawet ze swoim wykładowcą!

Napisanie własnego oprogramowania od zera to zawsze świetny sposób na nauczenie się algorytmu, istnieje jednak wiele książek, które dają kod źródłowy tradycyjnych algorytmów optymalizujących (np. [363]). W Internecie jest także dostępnych wiele źródeł darmowego oprogramowania realizującego algorytmy ewolucyjne. Na naszych stronach osobistych umieściliśmy trochę takiego oprogramowania, które można sobieściągnąć. Znajdziesz je, wpisując: <http://www.cs.adelaide.edu.au/~zbyszek/> oraz <http://www.naturalselection.com/people/dbf.html>. Jeśli adresy te zmieniłyby się lub z jakiegoś powodu przestały być dostępne, to możesz zawsze spróbować skontaktować się z każdym z nas e-mailem lub za pośrednictwem wydawcy. Poza tym łatwo nas znaleźć w sieci WWW.

Jeśli nie korzystałeś nigdy z metod rozwiązywania problemów i ten tekst jest Twoim wprowadzeniem w ten świat, to możesz zacząć od podanych tu prostych zadań.

1. Twój pierwszy krok przy rozwiązywaniu problemu powinien polegać na określeniu, co chcesz osiągnąć. Co więcej, musisz też ustalić, czego powinieneś unikać! Twój cel, gdy zostanie prawidłowo sformułowany, może być ujęty ilościowo, dzięki czemu możliwe stanie się określenie względnej wartości wszystkich możliwych rozwiązań Twojego zadania. Postawimy Cię teraz w kilku różnych sytuacjach. Twoje zadanie będzie polegać na znalezieniu odpowiednich funkcji oceny dla każdego z podanych przypadków.
 - (a) Zaczniemy od czegoś prostego. Masz znaleźć liczbę rzeczywistą x , która minimalizuje $f(x) = x^2$.
 - (b) Masz znaleźć ciąg zer i jedynek, dla którego suma wartości jego elementów jest maksymalna.
 - (c) Teraz coś bardziej skomplikowanego. Oto problem klasyfikacji wzorców. Lekarz ocenił cechy mammogramów, które badał, oraz określił, czy pacjent miał złośliwego raka. Masz do dyspozycji 100 wyników, z których każdy ma 10 cech wejściowych i jedną klasyfikację wyjściową.
 - (d) Rolnik chce znać najlepszy model do przewidywania ilości zbranej pszenicy w zależności od liczby słonecznych dni, opadów oraz ilości wysypywanych miesięcznie nawozów.
 - (e) Masz za zadanie sterować wózkiem z przymocowanym na jednym z końców kijem. Wózek nie może przekroczyć granic dostępnego toru, kij nie może przechylić się bardziej niż o wyznaczony kąt. W każdym momencie możesz określić położenie i prędkość wózka oraz kija.
 - (f) A teraz trudne zadanie. Opracuj odpowiednią funkcję oceny, która opisze Twoje własne cele. (Wskazówka: Przemyśl istotne parametry oraz ich wagę w zależności od ich istotności względnej, a następnie odpowiednio połącz stopień osiągnięcia w związku z każdym z parametrów).
2. Oto kilka podstawowych rzeczy, które należy rozważyć przy rozwiązywaniu problemów. Lista ta jest jedynie przeglądem podstawowych przedstawionych w tej książce zagadnień:
 - (a) Co to jest zmienna losowa?
 - (b) Które spośród następujących algorytmów są deterministyczne:
 - i. programowanie liniowe,
 - ii. symulowane wyżarzanie,
 - iii. przeszukiwanie z tabu,
 - iv. programowanie dynamiczne,
 - v. algorytmy ewolucyjne.

- (c) Co sprawia, że symulowane wyżarzanie jest algorytmem randomizowanym?
 - (d) Jakie aspekty algorytmów ewolucyjnych są losowe?
 - (e) Jeśli do rozwiązywania problemu korzystasz z algorytmu stochastycznego, to czy za każdym jego uruchomieniem otrzymujesz ten sam wynik? Jak to wygląda, gdy korzystasz z algorytmu deterministycznego?
 - (f) Czy umiałbyś wymyślić sytuacje, w których przy kolejnych próbach lepiej nie jest generować tego samego rozwiązania dla tego samego problemu?
3. Przy modelowaniu problemu musisz uwzględnić wszystkich „graczy”, którzy mają na Ciebie wpływ i na których Ty masz wpływ. Przemyśl każdą z podanych tu gier i wymień wszystkich biorących w niej udział graczy:
- (a) Zajmujesz się planowaniem produkcji butów w fabryce.
 - (b) Zajmujesz się prowadzeniem drużyny baseballowej.
 - (c) Musisz rano dostać się do pracy.
 - (d) Jesteś pisarzem i chcesz napisać bestseller.
 - (e) Jesteś studentem z dyplomem i chcesz zostać profesorem na uniwersytecie.
 - (f) Jesteś studentem z dyplomem i chcesz uniknąć zostania profesorem na uniwersytecie.
 - (g) Jesteś strażakiem, który przyjechał do płonącego budynku.

Wróć teraz do każdej z tych sytuacji i określ jakiś sensowny cel, który można tam osiągnąć, a następnie zobacz, czy możesz wyrazić ten cel w kategoriach ilościowych. Jak wyglądają „rozwiązań”? Na przykład, jeśli zajmujemy się planowaniem produkcji butów, to jedną z rzeczy, które trzeba rozważyć, jest dochodowość fabryki określona jako różnica między przychodami a kosztami. Jakie rzeczy powodują powstawanie przychodów? Jakie kosztów?

B.1. Kilka praktycznych problemów

Skoro już wypróbowałeś swoje możliwości przy modelowaniu pewnych sytuacji, możesz się zabrać do rozwiązywania pewnych praktycznych problemów, które możesz znaleźć w świecie rzeczywistym. Poniższa lista zadań to spis dobrych projektów, które pomogą Ci poznać podstawy rozwiązywania problemów i konkretnych algorytmów znajdowania rozwiązań.

1. *Skalowalność algorytmów rozwiązywających TSP*. Jak dobrze różne algorytmy rozwiązuje problem komiwojażera się skalują? W książce tej poznaliśmy wiele podejść do TSP. Niektóre z nich były oparte na metodach deterministycznych, na przykład algorytm zachłanny lub algorytm 2-opt, inne zaś,

jak symulowane wyżarzanie lub algorytmy ewolucyjne, miały charakter stochastyczny. Ciekawe byłoby stwierdzenie, jak te różne podejścia skalują się wraz ze wzrostem liczby miast w zadaniu. Spróbuj wykonać następujące doświadczenie. Dla danego algorytmu rozłoż losowo n miast wewnątrz jednostkowego kwadratu. W rozdziale 8 znajdują się wzory, których możesz użyć do oceny oczekiwanej długości najlepszej trasy. Wybierz próg, na przykład 10% powyżej oczekiwanej najlepszej długości, i wypróbuj swój algorytm na 100 różnych losowo otrzymanych zadaniach TSP z n miastami. Jak wiele prób może znaleźć rozwiązanie, które osiąga próg? Co się dzieje z liczbą sukcesów, gdy rośnie n ? Czy umiesz znaleźć zależność funkcyjną między częstością sukcesów a liczbą miast? Jeśli używasz metod stochastycznych, to możesz próbować odpowiedzieć jeszcze na następujące pytania:

- Ile średnio tras musisz ocenić, żeby znaleźć taką, która osiąga próg?
- Ile jest prób pomyślnie osiągających próg, jeśli liczba ocenianych tras jest ustalona?
- Jaka jest zależność między liczbą sukcesów a rozmiarem populacji w algorytmie ewolucyjnym traktowaną jako funkcja liczby pokoleń?

2. *Klasyfikacja wzorców.* Internet daje możliwość łatwego uzyskiwania zebranych przez innych ludzi zbiorów danych, które były wykorzystywane do testowania różnych algorytmów dla rozmaitych problemów. Jeden z takich problemów dotyczy klasyfikacji komórek ze względu na to, czy oznaczają one raka piersi. Jeśli odwiedzisz stronę:

<http://www.ics.uci.edu/~mlearn/MLRepository.html>

to możesz ściągnąć zestaw danych o nazwie „breast-cancer-wisconsin”. Twoje zadanie polega na opracowaniu procedury, która umie poprawnie sklasyfikować jak najwięcej przypadków. Możesz w tym celu użyć sieci neuronowej lub liniowej funkcji dyskryminacji. Pamiętaj, żeby rozdzielić dane na zbiory do trenowania i testowania. Użyj zbioru do testowania do końcowej oceny swojej metody. Jeśli znasz inne miejsca, w których są przechowywane podobne dane, dla dowolnego zastosowania, wypróbuj swoją metodę budowania odpowiedniego modelu dla tych danych. Po sprawdzeniu jej na kilku przykładach możesz wrócić do początku i dodać do danych wejściowych trochę szumu. Co się dzieje, gdy zwiększasz zmienność szumu? Czy efektywność rośnie, czy słabnie?

3. *Optymalizacja z ograniczeniami.* W rozdziale 9 podaliśmy kilka funkcji z ograniczeniami oraz różne podejścia dostosowane charakterem do warunków, które każda z nich stawia. Czy potrafisz określić, ile zyskujesz na tych specjalnie opracowanych operatorach? Przypuśćmy, że masz wypróbować algorytm ewolucyjny, który używa kodowania binarnego do rozwiązywania problemu optymalizacyjnego z ograniczeniami o wartościach rzeczywistych i który korzystał, powiedzmy, z jednopunktowego operatora krzyżowania oraz mutacji bitowej jako operatora różnicowania wraz z selekcją proporcjonalną lub rankingową. Jak

to podejście, w którym nie skorzystano z żadnych informacji o problemie, wypada w porównaniu z wynikami podanymi w rozdziale 9? Pamiętaj, że będziesz musiał w każdym przypadku wykonać wiele prób po to, żeby określić średnią efektywność. Co się stanie, gdy zmienisz reprezentację na reprezentację w postaci wektorów o wartościach ciągłych (zmiennopozycyjnych), a następnie zmienisz także operatory różnicowania na, powiedzmy, krzyżowanie arytmetyczne i mutację Gaussa? Jak wygląda porównanie tych struktur? Co się stanie, gdy w ogóle usuniesz mutację lub krzyżowanie? Pamiętaj, żeby nie przypisywać sukcesu lub porażki metodzie przez dołączenie lub usunięcie jakiegokolwiek konkretnego aspektu. Ważne jest połączenie wszystkich aspektów.

4. *Zbadaj metody adaptacyjne i samoadaptujące się.* W rozdziale 10 dowiedzieliśmy się, że algorytmy ewolucyjne mogą obejmować środki do dostosowywania swoich parametrów służących do poszukiwania ewolucyjnego. Czy umiałbyś wymyślić nowe metody sterowania na bieżąco parametrami ewolucji? Spróbuj tego. Przyjmij, że rodzic generuje potomka, który jest lepszy niż dotychczasowe rozwiązania. Może sensowne jest dalsze generowanie potomków w tym samym kierunku? Jak mógłbyś zaimplementować taki operator różnicowania dla reprezentacji ciągłej? Może sensowne byłoby też położenie mniejszego nacisku na poruszanie się w przeciwnym kierunku? Czy mógłbyś opracować przykłady, przy których ta heurystyka się nie sprawdza?

Widzieliśmy wiele podejść do samoadaptacji, w których z każdym osobnikiem są związane jego własne parametry sterujące strategią przeszukiwania przestrzeni rozwiązań. Czy sądzisz, że byłaby jakaś korzyść z takiego podejścia, w którym osobniki mogą korzystać z parametrów strategii innych rozwiązań w populacji? Może w całej populacji jest zawarta użyteczna informacja, której nie można znaleźć w żadnym z pojedynczych rozwiązań. Czy umiałbyś wymyślić samoadaptujący się operator, który wykorzystywałby taką „wiedzę” z całej populacji do prowadzenia poszukiwania lepszych rozwiązań? Spróbuj sformułować go w kontekście optymalizacji ciągłej lub może dla TSP, ale praca z dyskretnymi strukturami danych może stanowić tutaj pewne wyzwanie.

5. *Przy modelowaniu połącz różne rodzaje operatorów różnicowania.* Jeśli masz jakieś doświadczenie z modelowaniem systemów, to wiesz, że jeden z podstawowych sposobów podejścia do tego zagadnienia polega na założeniu, na podstawie jakichś danych, postaci modelu, a następnie zastosowaniu algorytmu gradientowego do optymalizacji jego parametrów. Postępowanie takie ma wady dwojakiego rodzaju: 1) musisz najpierw wybrać model i 2) stosując algorytm gradientowy możesz utknąć w lokalnym optimum. Jednak w przypadku algorytmów poszukiwania stochastycznego możesz opracować takie operatory różnicowania, które nie działają na informacji gradientowej, wobec tego możesz jednocześnie poszukiwać struktury modelu i jego parametrów. Pomyśl, jak zastosowałbyś takie rozwiązanie w kontekście ewoluujących sieci neuronowych lub systemów rozmytych. Będziesz musiał tutaj poddawać różnicowaniu nie tylko wagę sieci czy lokalizacje funkcji przynależności, ale także liczbę węzłów oraz funkcji, odpowiednio. Możesz spróbować uruchomić swój algorytm dla jakichś

danych (rozważ np. wyżej wspomniane zadanie klasyfikacji wzorców) i porównać swoje wyniki z uzyskanymi standardową metodą ręcznego dostrajania za-kończonego optymalizacją gradientową.

6. Połącz operatory poszukiwania lokalnego z poszukiwaniem ewolucyjnym. Zamiast porównywać dwa algorytmy, na przykład algorytm zachłanny z algorytmem ewolucyjnym, sprawdź, czy możesz znaleźć sposoby na poprawienie obydwu przez zastosowanie pewnych aspektów jednego w drugim. Wypróbuj ten pomysł na TSP albo na SAT. Pomyśl, jaką metodę ewolucyjną zastosowałeśbyś do nich. Jaką metodę klasyczną? Jakie są zalety i wady każdej z nich? Czy są takie rzeczy, w których podejście ewolucyjne jest dobre, a w których podejście klasyczne zawodzi i na odwrót? Czy widzisz sposoby wprowadzenia do Twojego algorytmu innych aspektów, takich jak temperatura lub pamięć z symulowanego wyżarzania lub poszukiwania z tabu, które mogłyby tutaj pomóc? Wykonaj ciąg prób na wybranym przez siebie zadaniu i określ ulepszenie, które możesz uzyskać, stosując metody hybrydowe, w porównaniu z metodami jednorodnymi.

7. Jak ważne jest inicjowanie? Wiele z rozważanych przez nas algorytmów wymaga podania na wstępie punktu początkowego lub całego ich zbioru. Czy są jakieś zasady dobierania tych punktów? Powiedzmy, że masz rozwiązać TSP. Czy nie byłoby lepiej zacząć od rozwiązania lub populacji rozwiązań, które zostały uzyskane za pomocą jakiegoś algorytmu optymalizacji lokalnej, na przykład poszukiwania zachłannego lub 2-opt? Co się dzieje, gdy masz rozwiązać NLP? Czy powinieneś najpierw wykonać jakąś lokalną optymalizację, a następnie zacząć poszukiwanie stochastyczne za pomocą wyżarzania lub algorytmów ewolucyjnych? Czy nie daje Ci to krótkoterminowych zysków, które na dłuższą metę okazują się być stratą, czyli czy nie jest tak, że algorytm w początkowym stadium zachowuje się lepiej, ale potem nie znajduje lepszych rozwiązań?

8. Zbadaj odporność różnych rozwiązań w środowisku zmieniającym się w czasie i w środowisku z szumem. W rozdziale 11 podaliśmy kilka sposobów radzenia sobie ze środowiskami zmiennymi w czasie. Spróbuj zaimplementować dla jakichś problemów NLP metodę niszowania lub współdzielenia wartości. Zaczni od zadań bez ograniczeń, ale z wieloma optimami lokalnymi. Czy na podstawie funkcji potrafisz opracować heurystyki ustalania parametrów skalujących? Jak sytuacja się zmieni, gdy będziesz miał do czynienia z problemem z ograniczeniami? Co się stanie, gdy obszar rozwiązań dopuszczalnych będzie niespójny i będzie się składał z kilku części oddzielonych rozwiązaniami niedopuszczalnymi?

Kolejny projekt może polegać na porównaniu efektywności różnych podejść do zaszumionych środowisk zmieniających się w czasie. Czy jakieś algorytmy są bardziej wrażliwe niż inne? Jak wygląda porównanie wyżarzania z poszukiwaniem z tabu w przypadku TSP, w którym pozycje miast mają zaszumiony charakter (tzn. miasto znajduje się w punkcie (50, 50), z tym że przy odwołaniu do tych wartości są podawane dwie losowe wartości o średniej 50)?

Czy algorytmy ewolucyjne są tu pod jakimś względem lepsze od wyżarzania bądź poszukiwania z tabu? Czy umiesz wymyślić sposób połączenia wszystkich trzech metod, który dałby lepsze wyniki?

9. Prognozowanie finansowe. Zarabianie pieniędzy cieszy się zawsze nie-słabnącym zainteresowaniem, w szczególności ciekawe jest przewidywanie przyszłych wartości akcji i towarów. Dzięki Internetowi możesz teraz łatwo zdobyć historie cen akcji wielu firm. Twoje zadanie polega na wyszukaniu wystarczającej ilości danych o różnych firmach z rozmaitych sektorów rynku (np. z bankowości, twórców układów scalonych, sklepów z artykułami gospodarstwa domowego, Internetu czy z przemysłu samochodowego). Po zdobyciu i przejrzeniu tych danych, spróbuj opracować modele, które pomogą Ci przewidzieć przyszłe ceny akcji z dokładnością umożliwiającą osiągnięcie zysku. Wymaga to nie tylko zdefiniowania modeli, z których będziesz korzystać (np. funkcji liniowych zależnych od poprzednich wartości, sieci neuronowych, systemów rozmytych), ale też funkcji wypłaty dla uzyskanych różnych wyników. Co więcej, musisz przemyśleć, jak będziesz oceniał swoje modele. Nie zapomnij, w celu określenia wiarygodności swoich modeli, przetestować je na najnowszych danych. Upewnij się też, że rozmiar Twojej próbki danych jest odpowiedni. I pamiętaj, że cały ten wysiłek jest tylko „dla rozrywki”!

Oprócz powyższych projektów (i jeśli chcesz, możesz wymyślać również własne), w miarę jak będziesz się coraz więcej dowiadywał o algorytmach ewolucyjnych, możesz też próbować wykonać kilka ciekawych eksperymentów. Sprawdź, czy Twoja intuicja zgadza się z rzeczywistością. Pamiętaj, uczysz się najwięcej, gdy Twoje hipotezy okazują się fałszywe!

10. Porównywanie operatorów różnicowania. Co sądzisz o istotności względnej różnych operatorów różnicowania? Zależy ona od problemu i, co więcej, zależy też od stadium ewolucji. Różne operatory mogą zmieniać swoją istotność względową. Przypuśćmy, że do celów testowych będziemy wykorzystywali tutaj TSP. Wypróbuj różne algorytmy ewolucyjne, zmieniającczęstość stosowania różnych operatorów różnicowania. Jak wskazaliśmy w rozdziale 8, możemy tutaj skorzystać z wielu możliwości. Następnie przejdź do problemów NLP, powiedzmy, nawet do czegoś tak prostego jak liniowe układy równań. Czy wartość operatorów różnicowania korzystających z jednego rodzica wydaje się w tym kontekście większa czy też mniejsza niż w TSP? Co się dzieje na różnych etapach ewolucji? Czy operatory korzystające z dwóch rodziców lepiej sprawdzają się na początku ewolucji? Jak opracowałbyś metodę umożliwiającą algorytmowi ewolucyjnemu ustalanie częstotliwości stosowania różnych operatorów w zależności od tego, jakie dają one wyniki?

11. Krzyżowanie z bezgłowym kurczakiem. Podstawowy pomysł związany z większością operatorów korzystających z dwóch rodziców polega na tym, żeby wziąć fragment jednego rozwiązania i połączyć go z fragmentem innego. Co jednak się stanie, gdy spróbujesz wykonać krzyżowanie jakiegoś rodzica z Twojej populacji z jakimś zupełnie losowym rozwiązaniem? Protokół ten

testuje hipotezę, że krzyżowanie rzeczywiście działa na „cegiełkach” pochodzących z dobrych rozwiązań i nie jest tylko swego rodzaju „makromutacją”. Jeśli możesz wykonać krzyżowanie istniejących rozwiązań z losowymi rozwiązaniami i uzyskać odpowiedzi lepsze (lub co najmniej tak dobre), niż gdy wykonujesz krzyżowanie rozwiązań w ramach tej samej populacji, to prawdopodobnie nie zastosowałeś reprezentacji, która wskazuje na użyteczne cegiełki rozwiązań. Może to sugerować konieczność stworzenia dla Twojego zadania innych operatorów różnicowania.

Opisany operator został nazwany przez Terry'ego Jonesa w 1995 roku operatorem krzyżowania z „bezgłowym kurczakiem”. Autor ten powiedział, że takie krzyżowanie z losowymi danymi można porównać do tworzenia potomków przy pomocy włóczącego się po okolicy mutanta w postaci kurczaka bez głowy. To w dalszym ciągu kurczak, ale czegoś mu brakuje istotnego. Stwierdzono, że rekombinacja z kurczakiem bez głowy działa lepiej niż zamiana gałęzi w drzewach analizy syntaktycznej, które w wielu przykładach kodują programy. Umiałbyś znaleźć inne zadania, gdzie tak jest?

12. Znaczenie osobników i pamięć. W rozdziale 16 omawialiśmy różne metody hybrydowe rozszerzania algorytmów ewolucyjnych. Jedną z rzeczy, które przy okazji możemy w związku z tym podejściem wypróbować, jest wprowadzenie znaczników (lub zbiorów znaczników) dla wszystkich osobników tak, żeby mogły one rozpoznawać inne osobniki w populacji. Możesz zatem opracować dodatkowe reguły, które w zależności od układu znaczników opisują, które rozwiązania z którymi wchodzą w rekombinację i oczywiście także, które rozwiązania odmówią rekombinacji z innymi. Znaczniki mogą być uaktualniane za pomocą samoadaptacji. W ten sposób osobniki mogą utworzyć swego rodzaju pamięć tego, które rozwiązania należy brać pod uwagę, a których unikać. Czy takie podejście daje jakieś obliczeniowe zyski? Wypróbuj to na zadaniach NLP i TSP. Jaki jest wymagany narzut obliczeniowy?

Powyzsza lista to nic więcej jak tylko zbiór propozycji. Dziedzina obliczeń ewolucyjnych, a także dziedziny związane z innymi omawianymi tu metodami rozwiązywania problemów, to wciąż dziedzina, w której możemy znaleźć więcej pytań niż odpowiedzi. Wasze projekty mogą różnić się pod względem złożoności i być dostosowane do Twojego profilu programistycznego oraz dostępnego Ci czasu. Niektóre z nich są dość proste, inne wymagają podejścia zespołowego.

Oto jeszcze garść pomysłów, z których mógłbyś skorzystać.

- Porównaj efektywność pracy kilku algorytmów (np. wspinania się, stochastycznego wspinania się, symulowanego wyżarzania, algorytmów ewolucyjnych) na kilku funkcjach testowych.
- Porównaj różne metody selekcji (tzn. selekcję proporcjonalną, rankinguową, turniejową) w algorytmach ewolucyjnych zastosowanych do różnych problemów.
- Zaimportuj różne metody obsługi ograniczeń dla problemów NLP z ograniczeniami (dekodery, algorytmy naprawiające, funkcje kary itp.).

- Przetestuj samoadaptujące metody poprawiania operatorów różnicowania w algorytmach ewolucyjnych użytych do rozwiązywania zadań NLP. Sprawdź, czy potrafisz przenieść niektóre z tych pomysłów do wyżarzania lub poszukiwania z tabu.
- Zaimplementuj reprezentację diploidalną w NLP, TSP lub w SAT. Sprawdź, jakie różne metody określania relacji dominacji możesz wymyślić.
- Zmodyfikuj istniejący algorytm ewolucyjny, dodając do niego ewolucję lamarkowską lub efekt Baldwina. Przeprowadź próby z różnymi problemami i określ, kiedy takie dodatki przynoszą korzyść.
- Zbadaj możliwości rozwiązywania problemów NLP z ograniczeniami lub problemów komiwojażera z wieloma komiwojażerami za pomocą koewolucji, w których rozwiązania współzawodniczą ze sobą wprost, ale nie dysponujemy zewnętrzna funkcją oceny. Czy możesz wymyślić sposób implementacji tego podejścia?
- Weź aplikację, nad którą ostatnio pracowałeś na innych zajęciach lub w pracy, i spróbuj połączyć ją z aplikacją wykorzystującą symulowane wyżarzanie, poszukiwanie z tabu lub z algorytmami ewolucyjnymi. Przetestuj wyniki i zobacz, jakiego rodzaju ulepszenia możesz uzyskać. Jeśli stwierdzisz, że podejście to jest rzeczywiście interesujące i chcesz się tym podzielić z innymi, to opisz je w fachowym artykule i wyślij na konferencję lub do czasopisma!

Nade wszystko ważne jest, żeby Twoje projekty były wyzwaniem, żeby wciągały i żeby dawały dużo zabawy!

B.2. Ogłaszañie wyników eksperymentów obliczeniowych z zastosowaniem metod heurystycznych

Wiele metod jest ocenianych przez wykonywanie eksperymentów na wielu zadaniach testowych. Często dość trudno jest uogólnić takie wyniki eksperymentalne, żeby można było głosić jakieś „ogólne”, dotyczące danej metody stwierdzenie. Możemy jednak w ten sposób zademonstrować użyteczność nowej metody na wielu starannie dobranych zadaniach, porównując ją z innymi uznanymi metodami. Zgodnie z [31] nowa metoda heurystyczna może mieć następujące cechy:

- daje wysokiej jakości rozwiązania szybciej niż inne podejścia;
- znajduje rozwiązania wyższej jakości niż inne podejścia;
- jest mniej czuła niż inne podejścia na zmiany w charakterze problemu, jakości danych lub stopniu dostosowania parametrów;
- jest łatwiejsza do implementacji;
- ma szerszy zakres zastosowania.

Co więcej [31]:

Raporty naukowe o heurystykach są cenne, jeśli są *odkrywcze* – dają wyobrażenie o ogólnej budowie heurystyk lub struktury problemu, wnikając w zasady decydujące o jakości algorytmu oraz wyjaśniając jego zachowanie, i są *teoretyczne* – zawierając teoretyczne spostrzeżenia dotyczące na przykład ograniczeń jakości rozwiązania.

Barr i in. [31] przedstawiają użyteczny przegląd tego, jak projektować i podawać do wiadomości wyniki eksperymentów obliczeniowych z użyciem metod heurystycznych. Na przykład przy przygotowywaniu i podawaniu do wiadomości wyników eksperymentów może być pożądane wykonanie następujących pięciu kroków (wymienionych w [31]):

- zdefiniowanie celu eksperymentu;
- wybranie miary jakości i czynników, które chce się badać;
- zaprojektowanie i wykonanie eksperymentu;
- przeanalizowanie danych i wyciągnięcie wniosków;
- podanie do wiadomości wyników eksperymentu.

Ważne jest, żeby poświęcić uwagę wszystkim tym punktom. Cel eksperymentów może na przykład ulegać zmianie [31]:

Eksperymenty obliczeniowe z algorytmami są zwykle wykonywane:
a) aby porównać działanie różnych algorytmów dla tej samej klasy problemów lub b) żeby scharakteryzować albo opisać efektywność algorytmu w oderwaniu od innych metod. Chociaż cele te są ze sobą w pewnej zależności, badający musi określić, co konkretnie ma być uzyskane w wyniku testów (tzn. na jakie pytania szukamy odpowiedzi, jakie hipotezy mają zostać sprawdzone).

Możemy też przyjąć, że efektywność działania algorytmu jest mierzona jakością najlepszego znalezionego rozwiązania, czasem jego odkrycia, czasem potrzebnym na dotarcie do „akceptowalnego” rozwiązania lub odpornością metody – że wymienimy kilka możliwości. W większości przypadków istotne jest porównanie nowej metody z ustalonimi już metodami rozwiązywania danej klasy problemów. Ważne jest, żeby pamiętać o przeanalizowaniu podstawowych czynników (takich jak wpływ rozmiaru problemu na jakość rozwiązania oraz na koszt obliczenia). Końcowy raport powinien zawierać także wystarczającą ilość informacji, żeby umożliwić czytelnikowi odtworzenie wyników, przynajmniej jeśli ma on dostęp do podobnego sprzętu.

Na stronach WWW jest wiele bibliotek ze standardowymi zadaniami testowymi. Należy z nich często korzystać.

Bibliografia

1. Aarts, E. and J. Korst (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley, Chichester, UK.
2. Aarts, E.H.L. and J.K. Lenstra, eds. (1995). *Local Search in Combinatorial Optimization*. John Wiley, Chichester, UK.
3. Adami, C. (1997). *Introduction to Artificial Life*. Telos Press, New York, NY.
4. Akl, S.G. (1989). *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, Englewood Cliffs, NJ.
5. Angeline, P.J. (1994). An Alternative Interpretation of the Iterated Prisoner's Dilemma and the Evolution of Non-mutual Cooperation. *Artificial Life IV*. R.A. Brooks and P. Maes, eds., MIT Press, Cambridge, MA, pp. 353-358.
6. Angeline, P.J. (1995). Adaptive and Self-Adaptive Evolutionary Computation. In *Computational Intelligence: A Dynamic System Perspective*. M. Palaniswami, Y. Attikiouzel. R.J. Marks, D. Fogel, and T. Fukuda, eds., IEEE Press, Piscataway, NJ, pp. 152-161.
7. Angeline, P.J. (1996). The Effects of Noise on Self-Adaptive Evolutionary Optimization. In [163], pp. 433-439.
8. Angeline, P.J. (1997). Tracking Extrema in Dynamic Environments. In [10], pp. 335-345.
9. Angeline, P.J. (1998). Evolving Predictors for Chaotic Time Series. In *Applications and Science of Computational Intelligence*, SPIE Vol. 3390, S.K. Rogers, D.B. Fogel. J.C. Bezdek, and B. Bossachi, eds., SPIE, Bellingham, WA, pp. 170-180.
10. Angeline, P.J., R.G. Reynolds, J.R. McDonnell, and R. Eberhart, eds. (1997). *Evolutionary Programming VI*. Lecture Notes in Computer Science, Vol. 1213, Springer, Berlin.

11. Applegate, D., R.E. Bixby, V. Chvatal, and W. Cook (1995). Finding Cuts in the TSP: A Preliminary Report. Report 95-05, DIMACS, Rutgers University, NJ.
12. Arabas, J., Z. Michalewicz, and J. Mulawka (1994). GAVaPS – A Genetic Algorithm with Varying Population Size. In [365], pp. 73-78.
13. Arthur, W.B. (1994). Inductive Reasoning and Bounded Rationality. *Amer. Econ. Assn. Papers Proc.*, Vol. 84, pp. 406-411.
14. Atmar, J.W. (1976). Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form. PhD Dissertation, New Mexico State University, Las Cruces. NM.
15. Axelrod, R. (1987). Evolution of Strategies in the Iterated Prisoner's Dilemma. In *Genetic Algorithms and Simulated Annealing*. L. Davis, ed., Pitman, London, pp. 32-41.
16. Bäck, T. (1992). The Interaction of Mutation Rate, Selection, and Self-Adaptation within a Genetic Algorithm. In [299], pp. 85-94.
17. Bäck, T. (1992). Self-Adaption in Genetic Algorithms. *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life*, F.J. Varela and P. Bourgine, eds., MIT Press, Cambridge, MA, pp. 263-271.
18. Bäck, T. (1993). Evolutionary Programming and Evolution Strategies: Similarities and Differences. In [153], pp. 11-22.
19. Bäck, T. (1993). Optimal Mutation Rates in Genetic Search. In [172], pp. 2-8.
20. Bäck, T. (1994). Selective Pressure in Evolutionary Algorithms: A Characterization of Selection Mechanisms. In [365], pp. 57-62.
21. Bäck, T. (1995). *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, NY.
22. Bäck, T. (1997). Mutation Parameters. In [26], page E1.2.1:7.
23. Bäck, T., ed. (1997). *Proceedings of the Seventh International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.
24. Bäck, T. (1998). On the Behavior of Evolutionary Algorithms in Dynamic Environments. In [369], pp. 446-451.
25. Bäck, T., A.E. Eiben, and M.E. Vink (1998). A Superior Evolutionary Algorithm for 3-SAT. In [359], pp. 125-136.
26. Bäck, T., D.B. Fogel, and Z. Michalewicz, eds. (1997). *Handbook of Evolutionary Computation*. Oxford University Press, New York, and Institute of Physics, London, UK.
27. Bäck, T. and M. Schütz (1996). Intelligent Mutation Rate Control in Canonical Genetic Algorithms. Technical Report, Center for Applied Systems Analysis, Dortmund, Germany.
28. Bagley, J.D. (1967). The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms. PhD Dissertation, University of Michigan, Ann Arbor, MI. *Dissertation Abstracts International*, 28(12), 5106B. (University Microfilms No. 68-7556).

29. Bak, P. (1996). *How Nature Works*, Oxford University Press, London, UK.
30. Banzhaf, W., P. Nordin, R.E. Keller, and F.D. Francone (1997). *Genetic Programming – An Introduction*. Morgan Kaufmann, San Mateo, CA.
31. Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart (1995). Designing and Reporting on Computational Experiments with Heuristic Methods. In *Proceedings of the International Conference on Metaheuristics for Optimization*, Kluwer Publishing, Norwell, MA, pp. 1-17.
32. Barricelli, N.A. (1954). Esempi Numerici de Processi di Evoluzione. *Methodos*, pp. 45-68.
33. Barricelli, N.A. (1963). Numerical Testing of Evolution Theories. Part II: Preliminary Tests of Performance, Symbiogenesis and Terrestrial Life. *Acta Biotheoretica*, Vol. 16, No. 3-4, pp. 99-126.
34. Batcher, K.E. (1964). A new internal sorting method. Goodyear Aerospace Report GER-11759.
35. Bazaraa, M.S., M.D. Sherali, and C.M. Shetty (1993). *Nonlinear Programming: Theory and Algorithms*. 2nd edition, John Wiley, New York, NY.
36. Bean, J.C. and A.B. Hadj-Alouane (1992). A Dual Genetic Algorithm for Bounded Integer Programs. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53.
37. Beasley, D., D.R. Bull, and R.R. Martin (1993). An Overview of Genetic Algorithms: Part 2, Research Topics. *University Computing*, Vol. 15, No. 4, pp. 170-181.
38. Belegundu, A.D., D.V. Murthy, R.R. Salagame, and E.W. Constans (1994). Multiobjective Optimization of Laminated Ceramic Composites Using Genetic Algorithms. *Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Panama City, Florida, pp. 1015-1022.
39. Belew, R.K. and L.B. Booker, eds. (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA.
40. Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ.
41. Bertsekas, D.P. (1987). *Dynamic Programming. Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ.
42. Bezdek, J.C. (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum Press, NY.
43. Bezdek, J.C., J. Keller, R. Krishnapuran, and N. Pal (1999). *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. Kluwer Academic Publishers, Norwell, MA.
44. Bezdek, J.C. and S.K. Pal, eds. (1992). *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*. IEEE Press, Piscataway, NJ.
45. Bilchev, G. and I. Parmee (1995). Ant Colony Search vs. Genetic Algorithms. Technical Report, Plymouth Engineering Design Centre, University of Plymouth, UK.

46. Beyer, H.-G. (1995). Toward a Theory of Evolution Strategies: On the Benefits of Sex – the (μ, λ) Theory. *Evolutionary Computation*, Vol. 3, No. 1, pp. 81-111.
47. Blair, A.D. (1999). Co-evolutionary Learning – Lessons for Human Education? In *Proceedings of the Fourth Conference of the Australasian Cognitive Science Society*, Newcastle, Australia.
48. Blair, A.D. and J.B. Pollack (1997). What Makes a Good Co-evolutionary Learning Environment? *Australian Journal of Intelligent Information Processing Systems*, Vol. 4, pp. 166-175.
49. Bland, R.G. and D.F. Shallcross (1989). Large Traveling Salesman Problems Arising from Experiments in X-Ray Crystallography: A Preliminary Report on Computation. *Operations Research Letters*, Vol. 8, pp. 125-128.
50. Bledsoe, W.W. (1961). The Use of Biological Concepts in the Analytical Study of Systems. Technical Report, Panoramic Research, Inc., Palo Alto, CA, November.
51. Bonomi, E. and J.-L. Lutton (1984). The n -city Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm. *SIAM Review*, Vol. 26, pp. 551-568.
52. Bowen, J. and G. Dozier (1995). Solving Constraint Satisfaction Problems Using a Genetic/Systematic Search Hybrid that Realizes When to Quit. In [130], pp. 122-129.
53. Box, G.E.P. (1957). Evolutionary Operation: Method for Increasing Industrial Productivity. *Applied Statistics*, Vol. 6, No. 2, pp. 81-101.
54. Box, G.E.P. and P.V. Voule (1955). The Exploration and Exploitation of Response Surfaces: An Example of the Link between the Fitted Surface and the Basic Mechanism of the System. *Biometric*, Vol. 11, pp. 287-323.
55. Branke, J. (1999). Enhanced Evolutionary Algorithms for Changing Optimization Problems. In [370], pp. 1877-1884.
56. Braun, H. (1990). On Solving Traveling Salesman Problems by Genetic Algorithms. In [420], pp. 129-133.
57. Bremermann, H.J. (1958). The Evolution of Intelligence. The Nervous System as a Model of Its Environment. Technical Report No. 1, Contract No. 477(17), Dept. Mathematics, University of Washington, Seattle, July.
58. Bremermann, H.J., M. Rogson, and S. Salaff (1966). Global properties of evolution processes. In *Natural Automata and Useful Simulations*, H.H. Pattee, E.A. Edlsack, L. Fein, and A.B. Callahan, eds., Spartan Books, Washington DC, pp. 3-41.
59. Brindle, A. (1981). Genetic Algorithms for Function Optimization. PhD Dissertation, University of Alberta, Edmonton, Canada.
60. Budinich, M. (1996). A Self-Organizing Neural Network for the Traveling Salesman Problem that is Competitive with Simulated Annealing. *Neural Computing*, Vol. 8, pp. 416-424.
61. Burke, L.I. (1994). Neural Methods for the Traveling Salesman Problem: Insights from Operations Research. *Neural Networks*, Vol. 7, No. 4, pp. 681-690.

62. Cantu-Paz, E. et al. (2003). *Proceedings of the Genetic and Evolutionary Conference 2003*, Lecture Notes in Computer Science, Vol. 2723 and Vol. 2724, Springer, Berlin.
63. Cartwright, H.M. and G.F. Mott (1991). Looking Around: Using Clues from the Data Space to Guide Genetic Algorithm Searches. In [39], pp. 108-114.
64. Chakraborty, U., K. Deb, and M. Chakraborty (1996). Analysis of Selection Algorithms: A Markov Chain Approach. *Evolutionary Computation*, Vol. 4, No. 2, pp. 132-167.
65. Chellapilla, K. (1998). Combining Mutation Operators in Evolutionary Programming. *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 3, pp. 91-96.
66. Chellapilla, K. and D.B. Fogel (1997). Exploring Self-Adaptive Methods to Improve the Efficiency of Generating Approximate Solutions to Traveling Salesman Problems Using Evolutionary Programming. In [10], pp. 361-371.
67. Chellapilla, K. and D.B. Fogel (1999). Evolution, neural networks, games, and intelligence. In *Proceedings of the IEEE*, Vol. 87, No. 9, pp. 1471-1496.
68. Chellapilla, K. and D.B. Fogel (1999). Evolving Neural Networks to Play Checkers without Expert Knowledge. *IEEE Transactions on Neural Networks*, Vol. 10, No. 6, pp. 1382-1391.
69. Chellapilla, K. and D.B. Fogel (2001). Evolving an Expert Checkers Playing Program without Using Human Expertise. *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4, pp. 422-428.
70. Chellapilla, K., D.B. Fogel, and S.S. Rao (1997). Gaining Insight into Evolutionary Programming Through Landscape Visualization: An Investigation into IIR Filtering. In [10], pp. 407-417.
71. Chen, S.J. and C.L. Hwang (1993). Fuzzy Multiple Attribute Decision-Making: Methods and Applications. *Lecture Notes in Economics and Mathematical Systems*, Vol. 375, Springer, Heidelberg.
72. Christofides, N. (1976). Worst-Case Analysis of a New Heuristic for the Traveling Salesman Problem. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA.
73. Clarke, G. and J.W. Wright (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *Operations Research*, Vol. 12, pp. 568-581.
74. Coello Coello, C.A. (1999). A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal*, Vol. 1, No. 3, pp. 269-308.
75. Coello Coello, C.A., Van Veldhuizen, D.A., and Lamont, G.B. (2002). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic, New York, NY.
76. Conrad, M. (1969). Computer Experiments on the Evolution of Coadaptation in a Primitive Ecosystem. PhD dissertation, Stanford University, Stanford, CA.

77. Conrad, M. and H.H. Pattee (1970). Evolution Experiments with an Artificial Ecosystem. *Journal of Theoretical Biology*, Vol. 28, pp. 393-409.
78. Cook, W.J., W.H. Cunningham, W.R. Pulleyblank, and A. Schrijver (1998). *Combinatorial Optimization*. John Wiley, Chichester, UK.
79. Costa, L. and Oliveira, P., An Adaptive Sharing Elitist Evolution Strategy for Multiobjective Optimization. *Evolutionary Computation*, Vol. 11, No. 4, pp. 417-438.
80. Coveyou R.R. and J.G. Sullivan (1961). Permutation (Algorithm 71). *Communications of the ACM*, Vol. 4, No. 11, p. 497.
81. Craighurst, R. and W. Martin (1995). Enhancing GA Performance through Crossover Prohibitions Based on Ancestry. In [130], pp. 130-135.
82. Darwen, P.J. and X. Yao (2001). Why More Choices Cause Less Cooperation in Iterated Prisoner's Dilemma. In *Proceedings of the 2001 Congress on Evolutionary Computation*, IEEE, Piscataway, NJ, pp. 987-994.
83. Dasgupta, D. and Z. Michalewicz, eds. (1997). *Evolutionary Algorithms in Engineering Applications*. Springer, New York, NY.
84. Davidor, Y., H.-P. Schwefel, and R. Männer, eds. (1994). *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science, Vol. 866, Springer, Berlin.
85. Davis, L. (1985). Job Shop Scheduling with Genetic Algorithms. In [199], pp. 136-140.
86. Davis, L. (1995). Applying Adaptive Algorithms to Epistatic Domains. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 162-164.
87. Davis, L., ed. (1987). *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, San Mateo, CA.
88. Davis, L. (1989). Adapting Operator Probabilities in Genetic Algorithms. In [199], pp. 61-69.
89. Davis, L. (1991). Bit-Climbing, Representational Bias, and Test Suite Design. In [39], pp. 18-23.
90. Davis, L. (1998). Private communication.
91. Davis, L., ed. (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, NY.
92. DeAngelis, D.L. and L. Godbout (1991). An Individual-Based Approach to Predicting Density-Dependent Dynamics in Smallmouth Bass Populations. *Ecological Modelling*, Vol. 57, pp. 91-115.
93. De Jong, K.A. (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral Dissertation, University of Michigan, Ann Arbor, MI. *Dissertation Abstract International*, 36(10), 5140B. (University Microfilms No 76-9381).
94. De Jong, K.A. (2002). *Evolutionary Computation*. MIT Press, Cambridge, MA.

95. De Jong K.A. and W.M. Spears (1989). Using Genetic Algorithms to Solve NP-Complete Problems. In [409], pp. 124-132.
96. de la Maza, M. and B. Tidor (1993). An Analysis of Selection Procedures with Particular Attention Payed to Boltzmann Selection. In [172], pp. 124-131.
97. Deb, K. (2000). An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, pp. 311-338.
98. Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley, New York, NY.
99. Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2001). A Fast Elitist Non-dominated Sorting Genetic Algorithm: NSGA-II. In [414], pp. 849-858.
100. DeGroot, M.H. (1975). *Probability and Statistics*. Addison-Wesley, Reading, MA.
101. Devore, J.L. (1995). *Probability and Statistics for Engineering and the Sciences*. 4th edition, Duxbury Press, Belmont, CA.
102. Dhar, V. and N. Ranganathan (1990). Integer Programming vs. Expert Systems: An Experimental Comparison. *Communications of ACM*, Vol. 33, No. 3, pp. 323-336.
103. Dorigo M. and G. Di Caro (1999). The Ant Colony Optimization Meta-Heuristic. *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover, eds., McGraw-Hill, New York, NY. (Also available as: Tech. Rep. IRIDIA/99-1, Universite Libre de Bruxelles, Belgium).
104. Dorigo M. and L.M. Gambardella (1997). Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53-66.
105. Dorigo M., V. Maniezzo, and A. Colorni (1991). Positive Feedback as a Search Strategy. Tech. Rep. No. 91-016, Politecnico di Milano, Italy.
106. Downing, K. (1997). EUZONE: Simulating the Evolution of Aquatic Ecosystems. *Artificial Life*, Vol. 3, pp. 307-333.
107. Doyle, A.C. (1905). *The Adventure of the Priory School*. In *The Complete Sherlock Holmes*, Vol. II, Barns & Noble Classics, New York, NY, 2003.
108. Dozier, G., J. Bowen, and D. Bahler (1994). Solving Small and Large Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm. In [365], pp. 306-311.
109. Dozier, G., J. Bowen, and D. Bahler (1994). Solving Randomly Generated Constraint Satisfaction Problems Using a Micro-Evolutionary Hybrid that Evolves a Population of Hill-Climbers. In [366], pp. 614-619.
110. Efron, B. (1982). *The Jackknife, the Bootstrap, and other Resampling Plans*, SIAM, Philadelphia, PA.
111. Eiben, A.E. (1999). Multi-Parent Recombination. In *Handbook of Evolutionary Computation*, 2nd edition, T. Bäck, D.B. Fogel, and Z. Michalewicz, eds., Institute of Physics, Philadelphia, PA, pp. 289-290.

112. Eiben, A.E. and T. Bäck (1997). Empirical Investigation of Multiparent Recombination Operators in Evolution Strategies. *Evolutionary Computation*, Vol. 5, No. 3, pp. 347-365.
113. Eiben, A.E., T. Bäck, M. Schoenauer, and H.-P. Schwefel, eds. (1998). *Proceedings of the 5th Parallel Problem Solving from Nature Conference*. Lecture Notes in Computer Science, Vol. 1498, Springer, Berlin.
114. Eiben, A.E., R. Hinterding, and Z. Michalewicz (1999). Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, pp. 124-141.
115. Eiben, A.E., C.H.M. van Kemenade, and J.N. Kok (1995). Orgy in the Computer: Multi-Parent Reproduction in Genetic Algorithms. In *Third European Conference on Artificial Life*, Springer, Berlin, pp. 934-945.
116. Eiben, A.E., P.-E. Raue, and Zs. Ruttkay (1994). Solving Constraint Satisfaction Problems Using Genetic Algorithms. In [365], pp. 542-547.
117. Eiben, A.E., P.-E. Raue, and Zs. Ruttkay (1994). Genetic Algorithms with Multi-Parent Recombination. In [84], pp. 78-87.
118. Eiben, A.E. and Zs. Ruttkay (1996). Self-Adaptivity for Constraint Satisfaction: Learning Penalty Functions. In [367], pp. 258-261.
119. Eiben, A.E. and C.A. Schippers (1996). Multi-Parent's Niche: N-ary Crossovers on NK-Landscapes. In [480], pp. 319-328.
120. Eiben, A.E. and J.E. Smith (2003). *Introduction to Evolutionary Computing*, Springer, Berlin.
121. Eiben, A.E., I.G. Sprinkhuizen-Kuyper, and B.A. Thijssen (1998). Competing Crossovers in an Adaptive GA Framework. In [369], pp. 787-792.
122. Eiben, A.E. and J.K. van der Hauw (1997). Adaptive Penalties for Evolutionary Graph-Coloring. *Artificial Evolution '97*, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, eds., Lecture Notes in Computer Science, Vol. 1363, Springer, Berlin, pp. 95-106.
123. Eiben, A.E. and J.K. van der Hauw (1997). Solving 3-SAT with Adaptive Genetic Algorithms. In [368], pp. 81-86.
124. Eiben, A.E., J.K. van der Hauw, and J.I. van Hemert (1998). Graph Coloring with Adaptive Evolutionary Algorithms. *Journal of Heuristics*, Vol. 4, pp. 25-46.
125. English, T.M. (1996). Evaluation of Evolutionary and Genetic Optimizers: No Free Lunch. In [163], pp. 163-169.
126. Eriksson, R. (1996). Applying Cooperative Coevolution to Inventory Control Parameter Optimization. Master's thesis, University of Sk'ovde, Sweden.
127. Eriksson, R. and B. Olsson (1997). Cooperative Coevolution in Inventory Control Optimization. In *Proceedings of Third Int. Conf. Artif. Neural Networks and Genetic Algorithms*, G.D. Smith, N.C. Steele, and R.F. Albrecht, eds., Springer, Berlin.

128. Escazut, C. and Ph. Collard (1997). Genetic Algorithms at the Edge of a Dream. In *Artificial Evolution '97*, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, eds., Lecture Notes in Computer Science, Vol. 1363, Springer, Berlin, pp. 69-80.
129. Eshelman, L. (1990). The CHC Adaptive Search Algorithm: How to Have Safe Search when Engaging in Nontraditional Genetic Recombination. In [380], pp. 265-283.
130. Eshelman, L.J., ed. (1995). *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
131. Eshelman, L.J. and J.D. Schaffer (1993). Crossover's Niche. In [172], pp. 9-14.
132. Esquivel S., A. Leiva, and R. Gallard (1997). Multiple Crossover per Couple in Genetic Algorithms. In [368], pp. 103-106.
133. Evans, M., N. Hastings, and B. Peacock (1993). *Statistical Distributions*. 2nd edition, John Wiley, New York, NY.
134. Falkenauer, E. (1994). A New Representation and Operators for GAs Applied to Grouping Problems. *Evolutionary Computation*, Vol. 2, No. 2, pp. 123-144.
135. Fiesler, E. and R. Beale, eds. (1997). *Handbook of Neural Computation*, IOP Press, Philadelphia, PA.
136. Floreano, D. and S. Nolfi (1997). God Save the Red Queen! Competition in Coevolutionary Robotics. In *Genetic Programming 1997*, J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, eds., Morgan Kaufmann, San Mateo, CA, pp. 398-406.
137. Floudas, C.A. and Pardalos, P.M. (1992). *Recent Advances in Global Optimization*. Princeton Series in Computer Science, Princeton University Press, Princeton, NJ.
138. Floyd, R.W. and D.E. Knuth (1967). Improved Constructions for the Bose-Nelson Sorting Problem. *Notices American Mathematical Sonety*, Vol. 14, p. 283.
139. Fodor, J.C., J.-L. Marichal, and M. Roubens (1995). Characterization of the Ordered Weighted Averaging Operators. *IEEE Transactions on Fuzzy Systems*, Vol. 3, pp. 236-240.
140. Fodor, J.C. and M. Roubens (1994). *Fuzzy Preference Modelling and Multi-criteria Decision Support*. Kluwer, Dordrecht.
141. Fogarty, T. (1989). Varying the Probability of Mutation in the Genetic Algorithm. In [409], pp. 104-109.
142. Fogel, D.B. (1988). An Evolutionary Approach to the Traveling Salesman Problem. *Biological Cybernetics*, Vol. 60, pp. 139-144.
143. Fogel, D.B. (1990). System Identification through Simulated Evolution. MSc. Thesis, UC San Diego, CA.
144. Fogel, D.B. (1991). An Information Criterion for Optimal Neural Network Selection. *IEEE Transactions on Neural Networks*, Vol. 2, No. 5, pp. 490-497.

145. Fogel, D.B. (1992). Evolving Artificial Intelligence. PhD Dissertation, University of California at San Diego, La Jolla, CA.
146. Fogel, D.B. (1993). Evolving Behaviors in the Iterated Prisoner's Dilemma. *Evolutionary Computation*, Vol. 1, No. 1, pp. 77-97.
147. Fogel, D.B. (1993). Applying Evolutionary Programming to Selected Traveling Salesman Problems. *Cybernetics and Systems*, Vol. 24, No. 1, pp. 27-36.
148. Fogel, D.B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine, Intelligence*. IEEE Press, Piscataway, NJ.
149. Fogel, D.B., ed. (1998). *Evolutionary Computation: The Fossil Record*, IEEE Press, Piscataway, NJ.
150. Fogel, D.B. (2002). *Blondie 24: Playing at the Edge of AI*. Morgan Kaufmann, San Francisco, CA.
151. Fogel, D.B. and J.W. Atmar (1990). Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems. *Biological Cybernetics*, Vol. 63, No. 2, pp. 111-114.
152. Fogel, D.B. and W. Atmar, eds. (1992). *Proceedings of the 1st Annual Conference on Evolutionary Programming*, Evolutionary Programming Society, La Jolla, CA.
153. Fogel, D.B. and W. Atmar, eds. (1993). *Proceedings of the 2nd Annual Conference on Evolutionary Programming*. Evolutionary Programming Society, La Jolla, CA.
154. Fogel, D.B., K. Chellapilla, and P.J. Angeline (1999). Inductive Reasoning and Bounded Rationality Reconsidered. *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 2, pp. 142-146.
155. Fogel, D.B., L.J. Fogel, and J.W. Atmar (1991). Meta-Evolutionary Programming. In *Proceeding of the 25th Asilomar Conference on Signals, Systems and Computers*, R.R. Chen, ed., Maple Press, San Jose, CA, pp. 540-545.
156. Fogel, D.B. and A. Ghozeil (1997). A Note on Representations and Variation Operators. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 2, pp. 159-161.
157. Fogel, D.B. and A. Ghozeil (1996). Using Fitness Distributions to Design More Efficient Evolutionary Computations. In [367], pp. 11-19.
158. Fogel, D.B. and T.J. Hays (2003). New Results in Evolving Strategies in Chess. In *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation VI*, Vol. 5200, B. Bosacchi, D.B. Fogel, and J.C. Bezdek (chairs), SPIE, Bellingham, WA, pp. 56-63.
159. Fogel, D.B., E.C. Wasson, and E.M. Boughton (1995). Evolving Neural Networks for Detecting Breast Cancer. *Cancer Letters*, Vol. 96, pp. 49-53.
160. Fogel, D.B., E.C. Wasson, E.M. Boughton, and V.W. Porto (1998). Evolving Artificial Neural Networks for Screening Features from Mammograms. *Artificial Intelligence in Medicine*, Vol. 14, pp. 317-326.
161. Fogel, L.J. (1964). On the Organization of Intellect. PhD Dissertation, University of California at Los Angeles, Los Angeles, CA.

162. Fogel, L.J. (1995). The Valuated State Space Approach and Evolutionary Computation for Problem Solving. In *Computational Intelligence: A Dynamic Systems Perspective*, M. Palaniswami, Y. Attikiouzel, R.J. Marks, D. Fogel, and T. Fukuda, eds., IEEE Press, NY, pp. 129-136.
163. Fogel, L.J., P.J. Angeline, and T. Bäck, eds. (1996). *Proceedings of the 5th Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA.
164. Fogel, L.J., P.J. Angeline, and D.B. Fogel (1995). An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines. In [304], pp. 355-365.
165. Fogel, L.J. and G.H. Burgin (1969). Competitive Goal-Seeking Through Evolutionary Programming. Final Report, Contract AF 19(628)-5927, AF Cambridge Research Lab.
166. Fogel, L.J., D.B. Fogel, and W. Atmar (1993). Evolutionary Programming for ASAT Battle Management. In *Proceedings of the 27th Asilomar Conference on Signals, Systems, and Computers*, A. Singh, ed., IEEE Computer Society Press, Los Alamitos, CA, pp. 617-621.
167. Fogel, L.J., A.J. Owens, and M.J. Walsh (1966). *Artificial Intelligence Through Simulated Evolution*. John Wiley, New York, NY.
168. Fonseca, C.M. and P.J. Fleming (1993). Genetic Algorithm for Multiobjective Optimization: Formulation, Discussion, and Generalization. In [172], pp. 416-423.
169. Fonseca, C.M. and P.J. Fleming (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, Vol. 3, No. 1, 1995, pp. 165-180.
170. Fonseca, C.M. and P.J. Fleming (1997). Multiobjective Optimization. In *Handbook of Evolutionary Computation*, T. Bäck, D.B. Fogel, and Z. Michalewicz, eds., Oxford/IOP, NY, pp. C4.5:1-9.
171. Fonseca, C.M., P.J. Fleming, E. Zitzler, K. Deb, and L. Thiele, eds. (2003). *Evolutionary Multi-Criterion Optimization. Second International Conference EMO 2003*. Lecture Notes in Computer Science, Vol. 2632, Springer, Berlin.
172. Forrest, S., ed. (1993). *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.
173. Fourman, M. (1985). Compaction of Symbolic Layout Using Genetic Algorithms. In [199], pp. 141-153.
174. Fox, B.R. and M.B. McMahon (1990). Genetic Operators for Sequencing Problems. In [380], pp. 284-300.
175. Fox, M.S. (1987). *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann, San Mateo, CA.
176. Freville, A. and G. Plateau (1993). Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems. *European Journal of Operational Research*, Vol. 24, pp. 206-215.

177. Fuller, R. (1996). OWA Operators in Decision Making. In *Exploring the Limits of Support Systems*, C. Carlsson, ed., TUCS General Publications, No. 3, Turku Centre for Computer Science, Abo, pp. 85-104.
178. Gardner, M. (1978). *Aha! Insight*. Scientific American, Inc./W.H. Freeman, New York, NY.
179. Garey, M. and D. Johnson (1979). *Computers and Intractability*. W.H. Freeman, San Francisco, CA.
180. Gee, A.H. and R.W. Prager (1995). Limitations of Neural Networks for Solving Traveling Salesman Problems. *IEEE Transactions on Neural Networks*, Vol. 6, No. 1, pp. 280-282.
181. Gehlhaar, D.K., G.M. Verkhivker, P.A. Rejto, C.J. Sherman, D.B. Fogel, L.J. Fogel, and S.T. Freer (1995). Molecular Recognition of the Inhibitor AG-1343 by HIV-1 Protease: Conformationally Flexible Docking by Evolutionary Programming. *Chemistry and Biology*, Vol. 2, No. 5, pp. 317-324.
182. Gent, I.P. and T. Walsh (1992). The Enigma of SAT Hill-Climbing Procedures. Technical Report 605, Department of Computer Science, University of Edinburgh.
183. Ghozeil, A. and D.B. Fogel (1996). A Preliminary Investigation into Directed Mutations in Evolutionary Algorithms. In [480], pp. 329-335.
184. Glover, F. (1995). Tabu Search Fundamentals and Uses. Graduate School of Business, University of Colorado, CO.
185. Glover, F. and G. Kochenberger (1995). Critical Event Tabu Search for Multidimensional Knapsack Problems. In *Proceedings of the International Conference on Metaheuristics for Optimization*, Kluwer Publishing, Norwell, MA, pp. 113-133.
186. Gkwer, F. and M. Laguna (1997). *Tabu Search*. Kluwer, London.
187. Goldberg, D.E. (2003). *Algorytmy genetyczne i ich zastosowania*. Wyd. 3. Wydawnictwa Naukowo-Techniczne, Warszawa.
188. Goldberg, D.E., K. Deb, and J.H. Clark (1992). Accounting for Noise in the Sizing of Populations. In [490], pp. 127-140.
189. Goldberg, D.E., K. Deb, and J.H. Clark (1992). Genetic Algorithms, Noise, and the Sizing of Populations. *Complex Systems*, Vol. 6, pp. 333-362.
190. Goldberg, D.E., K. Deb, and B. Korb (1991). Don't Worry, be Messy. In [39], pp. 24-30.
191. Goldberg, D.E., K. Deb, and D. Thierens (1992). Toward a Better Understanding of Mixing in Genetic Algorithms. In [39], pp. 190-195.
192. Goldberg, D.E. and R. Lingle (1985). Alleles, Loci, and the TSP. In [199], pp. 154-159.
193. Goldberg, D.E. and R.E. Smith (1987). Nonstationary Function Optimization Using Genetic Algorithms with Dominance and Diploidy. In [200], pp. 59-68.
194. Goldberg, D.E. et al. (1999). *Proceedings of the Genetic and Evolutionary Conference 1999*, Morgan Kaufmann, San Mateo, CA.

195. Gorges-Schleuter, M. (1991). ASPARAGOS: An Asynchronous Parallel Genetic Optimization Strategy. In [39], pp. 422-427.
196. Greene, F. (1994). A Method for Utilizing Diploid and Dominance in Genetic Search. In [365], pp. 439-444.
197. Greene, F. (1997). Performance of Diploid Dominance with Genetically Synthesized Signal Processing Networks. In [23], pp. 615-622.
198. Grefenstette, J.J. (1984). GENESIS: A System for Using Genetic Search Procedures. *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, Rochester, MI, pp. 161-165.
199. Grefenstette, J.J., ed. (1985). *Proceedings of the First International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Hillsdale, NJ.
200. Grefenstette, J.J., ed. (1987). *Proceedings of the 2nd International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Hillsdale, NJ.
201. Grefenstette, J.J. (1986). Optimization of Control Parameters for Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 1, pp. 122-128.
202. Grefenstette, J.J. (1987). Incorporating Problem Specific Knowledge into Genetic Algorithms. In [87], pp. 42-60.
203. Grefenstette, J.J., R. Gopal, B. Rosmaita, and D. Van Gucht (1985). Genetic Algorithms for the TSP. In [199], pp. 160-168.
204. Gregory, J. (1995). Nonlinear Programming FAQ. Usenet sci.answers, 1995. Available at <ftp://rtfm.mit.edu/pub/usenet/sci.answers/nonlinear-programming-faq>.
205. Hadj-Alouane, A.B. and J.C. Bean (1992). A Genetic Algorithm for the Multiple-Choice Integer Program. Department of Industrial and Operations Engineering, University of Michigan, TR 92-50.
206. Hajela, P. and C.-Y.Lin (1992). Genetic Search Strategies in Multicriterion Optimal Design. *Structural Optimization*, Vol. 4, pp. 99-107.
207. Harik, G., E. Cantu-Paz, D.E. Goldberg, and B.L. Miller (1997). The Gambler's Ruin Problem, Genetic Algorithms, and the Sizing of Populations. In [368], pp. 7-12.
208. Harrald, P.G. and D.B. Fogel (1996). Evolving Continuous Behaviors in the Iterated Prisoner's Dilemma. *BioSystems*. Vol. 37, pp. 135-145.
209. Hart, W.E. and R.K. Belew (1991). Optimizing an Arbitrary Function is Hard for Genetic Algorithms. In [39], pp. 190-195.
210. Haykin, S. (1994). *Neural Networks: A Comprehensive Foundation*, IEEE Press, Piscataway, NJ.
211. Held, M. and Karp, R.M. (1970). The Traveling Salesman Problem and Minimum Spanning Trees. *Operations Research*, Vol. 18, pp. 1138-1162.
212. Held, M. and Karp, R.M. (1971). The Traveling Salesman Problem and Minimum Spanning Trees: Part II. *Mathematical Programming*, Vol. 1, pp. 6-25.
213. Herdy, M. (1990). Application of the Evolution Strategy to Discrete Optimization Problems. In [420], pp. 188-192.

214. Herrera, F., E. Herrera-Viedma, and J.L. Verdegay (1996). Direct Approach Processes in Group Decision Making Using Linguistic OWA Operators. *Fuzzy Sets and Systems*, Vol. 79, pp. 175-190.
215. Hesser, J. and R. Männer (1991). Towards an Optimal Mutation Probability for Genetic Algorithms. In [420], pp. 23-32.
216. Hilliard, M.R., G.E. Liepins, M. Palmer, and G. Rangarajen (1989). The Computer as a Partner in Algorithmic Design: Automated Discovery of Parameters for a Multiobjective Scheduling Heuristic. In *Impacts of Recent Computer Advances on Operations Research*, R. Sharda, B.L. Golden, E. Wasil, O. Balci, and W. Stewart, eds., North-Holland, New York, NY.
217. Hillier, F.S. and G.J. Lieberman (1980). *Introduction to Operations Research*. 3rd edition, Holden-Day, Inc., Oakland, CA.
218. Hillis, W.D. (1992). Co-evolving Parasites Improve Simulated Evolution as an Optimization Procedure. In *Artificial Life II*, C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, eds., Addison-Wesley, Reading, MA, pp. 313-324.
219. Himmelblau, D. (1992). *Applied Nonlinear Programming*. McGraw-Hill, NY.
220. Hinterding, R. (1995). Gaussian Mutation and Self-Adaption in Numeric Genetic Algorithms. In [366], pp. 384-389.
221. Hinterding, R. (1997). Self-Adaptation Using Multi-Chromosomes. In [368], pp. 87-91.
222. Hinterding, R. and Z. Michalewicz (1998). Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In [369], pp. 810-815.
223. Hinterding, R., Z. Michalewicz, and A.E. Eiben (1997). Adaptation in Evolutionary Computation: A Survey. In [368], pp. 65-69.
224. Hinterding, R., Z. Michalewicz, and T.C. Peachey (1996). Self-Adaptive Genetic Algorithm for Numeric Functions. In [480], pp. 420-429.
225. Hinton, G.E. and T. Sejnowski (1983). Optimal Perceptual Inference. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE, Washington D.C., pp. 448-453.
226. Hochbaum, D.S. (1997). *Approximation Algorithms for NP-hard Problems*, PWS Publishing, Boston, MA.
227. Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes*. Lecture Notes in Economics and Mathematical Systems, Vol. 187, Springer, Berlin.
228. Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.
229. Hollstein, R.B. (1971). Artificial Genetic Adaption in Computer Control Systems. PhD Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI.
230. Homaifar, A. and S. Guan (1991). A New Approach on the Traveling Salesman Problem by Genetic Algorithms. Technical Report, North Carolina A & T State University.

231. Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained Optimization via Genetic Algorithms. *Simulation*, Vol. 62, pp. 242-254.
232. Hopfield, J.J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, Vol. 79, pp. 2554-2558.
233. Hopfield, J.J. and D.W. Tank (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, Vol. 52, pp. 141-152.
234. Horn, J. and N. Nafpliotis (1993). Multiobjective Optimization Using the Niched Pareto Genetic Algorithm. Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL, Technical Report IlliGAL 93005.
235. Kornik, K., M. Stinchcombe, and H. White (1990). Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks. *Neural Networks*, Vol. 3, pp. 551-560.
236. Jacob, W., M. Gorges-Schleuter, and C. Blume (1992). Application of Genetic Algorithms to Task Planning and Learning. In [299], pp. 291-300.
237. Janikow, C. and Z. Michalewicz (1991). An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms. In [39], pp. 151-157.
238. Jog, P., J.Y. Suh, and D.V. Gucht (1989). The Effects of Population Size, Heuristic Crossover, and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem. In [409], pp. 110-115.
239. Johnson, D.S. (1990). Local Optimization and the Traveling Salesman Problem. In *Proceedings of the 17th Colloquium on Automata, Languages, and Programming*, M.S. Paterson, ed., Lecture Notes in Computer Science, Vol. 443, Springer, Berlin, pp. 446-461.
240. Johnson, D.S. (1995). Private communication.
241. Johnson, D.S. (1995). The Traveling Salesman Problem: A Case Study in Local Search. Presented during the Metaheuristics International Conference, Breckenridge, Colorado, July 22-26, 1995.
242. Johnson, D.S. (1995). The Traveling Salesman Problem: A Case Study. In [2], pp. 215-310.
243. Johnson, D.S., L.A. Bentley, L.A. McGeoch, and E.E. Rothberg (1999). Near-Optimal Solutions to Very Large Traveling Salesman Problems. In preparation.
244. Johnson, D.S., L.A. McGeoch, and E.E. Rothberg (1996). Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ACM, New York, and SIAM, Philadelphia, PA, pp. 341-350.
245. Johnson, L.W. and R.D. Riess (1982). *Numerical Analysis*, 2nd edition, John Wiley, New York, NY.
246. Johnson, R.W., M.E. Melich, Z. Michalewicz, and M. Schmidt (2004). Coevolutionary „TEMPO” Game. Technical Report, University of North Carolina at Charlotte.

247. Joines, J. and C. Houck (1994). On the Use of Non-stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems with GAs. In [365], pp. 579-584.
248. Judson, O.P. (1994). The Rise of the Individual-Based Model in Ecology. *Trends in Ecology and Evolution*, Vol. 9, pp. 9-14.
249. Julstrom, B.A. (1995). What Have You Done for me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm. In [130], pp. 81-87.
250. Kampfer, R.R. and M. Conrad (1983). Computational Modeling of Evolutionary Learning Processes in the Brain. *Bulletin of Mathematical Biology*, Vol. 45, No. 6, pp. 931-968.
251. Karp, R.M. (1977). Probabilistic Analysis of Partitioning Algorithm for the Traveling Salesman Problem in the Plane. *Mathematics of Operations Research*, Vol. 2, No. 3, pp. 209-224.
252. Kauffman, S.A. (1994). *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, London, UK.
253. Kaufman, H. (1967). An Experimental Investigation of Process Identification by Competitive Evolution. *IEEE Transactions on Systems, Science and Cybernetics*, Vol. SSC-3, No. 1, pp. 11-16.
254. Keane, A.J. (1996). A Brief Comparison of Some Evolutionary Optimization Methods. In *Modern Heuristic Search Methods*, V. Rayward-Smith, I. Osman, C. Reeves and G.D. Smith, eds., John Wiley, New York, NY, pp. 255-272.
255. Keeney, R. and H. Raiffa (1976). *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*. John Wiley, New York, NY.
256. Kelly, J.P., B.L. Golden, and A.A. Assad (1993). Large Scale Controlled Rounding Using Tabu Search with Strategic Oscillation. *Annals of Operations Research*, Vol. 41, pp. 69-84.
257. Kelly, J. and M. Laguna (1996). Article posted in *Genetic Algorithms Digest*, Vol. 10, No. 16.
258. Kendall, G. and G. Whitwell (2001). An Evolutionary Approach for the Tuning of a Chess Evaluation Function Using Population Dynamics. In *Proceedings of the 2001 Congress on Evolutionary Computation*, IEEE Press, Piscataway, NY, pp. 995-1002.
259. Kennedy, J. and R. Eberhart (1995). Particle Swarm Optimization. In *Proceedings IEEE International Conference on Neural Networks*, IEEE Press, Piscataway, NJ, pp. 1942-1948.
260. Kita, H., Y. Yabumoto, N. Mori, and Y. Nishikawa (1996). Multi-Objective Optimization by Means of the Thermodynamical Genetic Algorithm. In [480], pp. 504-512.
261. Klir, G.J. and B. Yuan (1995). *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice-Hall, Upper Saddle River, NJ.
262. Knowles, J.D. and D.W. Corne (2000). Approximating the Non-dominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, Vol. 8, No. 2, pp. 149-172.

263. Knox, J. (1994). Tabu Search Performance on the Symmetric Traveling Salesman Problem. *Computer Operations Research*, Vol. 21, No. 8, pp. 867-876.
264. Knuth, D.E. (2002). *Sztuka programowania, Tom 3: Sortowanie i wyszukiwanie*. Wydawnictwa Naukowo-Techniczne, Warszawa.
265. Konhauser, J.D.E., D. Velleman, and S. Wagon (1996). *Which Way Did the Bicycle Go?* Mathematical Association of America, Washington CD.
266. Korte, B. (1988). Applications of Combinatorial Optimization. Talk at the 13th International Mathematical Programming Symposium, Tokyo.
267. Koza, J.R. (1992). *Genetic Programming*. MIT Press, Cambridge, MA.
268. Koza, J.R. (1994). *Genetic Programming – 2*. MIT Press, Cambridge, MA.
269. Koza, J.R., D.E. Goldberg, D.B. Fogel, and R.L. Riolo, eds. (1996). *Proceedings of the 1st Annual Conference on Genetic Programming*. MIT Press, Cambridge, MA.
270. Koza, J.R., K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, and R.L. Riolo, eds. (1997). *Proceedings of the 2nd Annual Conference on Genetic Programming*. MIT Press, Cambridge, MA.
271. Koza, J.R., W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, D.E. Goldberg, H. Iba, and R.L. Riolo, eds. (1998). *Proceedings of the 3rd Annual Conference on Genetic Programming*. MIT Press, Cambridge, MA.
272. Kozieł, S. and Z. Michalewicz (1998). A Decoder-Based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In [113], pp. 231-240.
273. Kozieł, S. and Z. Michalewicz (1999). Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation*, Vol. 7, No. 1, pp. 19-44.
274. Krink, T., B.H. Mayoh, and Z. Michalewicz (1999). A PATCHWORK Model for Evolutionary Algorithms with Structured and Variable Size Populations. In *Proceedings of the Genetic and Evolutionary Computation Conference*, Orlando, Florida, pp. 1321-1328.
275. Krink, T. and F. Vollrath (1997). Analysing Spider Web-Building Behaviour with Rulebased Simulations and Genetic Algorithms. *Journal of Theoretical Biology*, Vol. 185, pp. 321-331.
276. Kursawe, F. (1991). A Variant of Evolution Strategies for Vector Optimization. In [420], pp. 193-197.
277. Langton, C.G., ed. (1989). *Artificial Life*. Addison-Wesley, Reading, MA.
278. Langdon, W.B. et al. (2002). *Proceedings of the Genetic and Evolutionary Conference 2002*, Morgan Kaufmann, San Mateo, CA.
279. Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1985). *The Traveling Salesman Problem*. John Wiley, Chichester, UK.
280. Le Riche, R., C. Knopf-Lenoir, and R.T. Haftka (1995). *A Segregated Genetic Algorithm for Constrained Structural Optimization*. In [130], pp. 558-565.
281. Lewis, J., E. Hart, and G. Ritchie (1998). A Comparison of Dominance Mechanism and Simple Mutation on Non-stationary Problems. In [113], pp. 139-148.

282. Lidd, M.L. (1991). Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms. Technical Report, MITRE Corporation.
283. Liepins, G.E., M.R. Hilliard, J. Richardson, and M. Palmer (1990). Genetic Algorithms Application to Set Covering and Traveling Salesman Problems. In *Operations Research and Artificial Intelligence: The Integration of Problem-solving Strategies*, D.E. Brown, and C.C. White, eds. Kluwer Academic, Norwell, MA, pp. 29-57.
284. Lin, S. and B.W. Kernighan (1973). An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, Vol. 21, pp. 498-516.
285. Lis, J. (1996). Parallel Genetic Algorithm with Dynamic Control Parameter. In [367], pp. 324-329.
286. Lis, J. and M. Lis (1996). Self-adapting Parallel Genetic Algorithm with the Dynamic Mutation Probability, Crossover Rate and Population Size. In *Proceedings of the 1st Polish National Conference on Evolutionary Computation*, J. Arabas, ed., Politechnika Warszawska, pp. 324-329.
287. Litke, J.D. (1984). An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes. *Communications of the ACM*, Vol. 27, No. 12, pp. 1227-1236.
288. Liu, Y. and X. Yao (1999). Ensemble Learning via Negative Correlation. *Neural Networks*, Vol. 12, No. 10, pp. 1399-1404.
289. Liu, Y. and X. Yao (1999). Simultaneous Training of Negatively Correlated Neural Networks in an Ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 29, No. 6, pp. 716-725.
290. Liu, Y., X. Yao, and T. Higuchi (2000). Evolutionary Ensembles with Negative Correlation Learning. *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 4, pp. 380-387.
291. Lord, W. (1955). *A Night to Remember*. Holt, Rinehart & Winston, Austin, TX.
292. Louis, S.J. and Z. Xu (1996). Genetic Algorithms for Open Shop Scheduling and Rescheduling. In *Proceedings of the ISCA Eleventh International Conference on Computers and Their Applications*, M.E. Cohen and D.L. Hudson, eds., pp. 99-102.
293. Luhandjula, M.K. (1989). Fuzzy Optimization: An Appraisal. *Fuzzy Sets and Systems*, Vol. 30, pp. 257-282.
294. Maa, C. and M. Shanblatt (1992). A Two-Phase Optimization Neural Network. *IEEE Transactions on Neural Networks*, Vol. 3, No. 6, pp. 1003-1009.
295. Macready, W.G. and D.H. Wolpert (1998). Bandit Problems and the Exploration/Exploitation Tradeoff. *IEEE Transactions on Evolutionary Computation*, Vol. 2, No. 1, pp. 2-22.
296. Madsen, K. and S. Zertchaninov (1998). A New Branch-and-Bound Method for Global Optimization. Technical Report IMM-REP-1998-05, Department of Mathematical Modelling, Technical University of Denmark.

297. Maekawa, K., N. Mori, H. Tamaki, H. Kita, and Y. Nishikawa (1996). A Genetic Solution for the Traveling Salesman Problem by Means of a Thermodynamical Selection Rule. In [367], pp. 529-534.
298. Mahfoud, S.W. (1997). Boltzmann Selection. In [26], pp. C2.5:l-4.
299. Männer, R. and B. Manderick, eds. (1992). *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature 2*. North-Holland, Amsterdam, The Netherlands.
300. Martin, W.N., J. Lienig, and J.P. Cohoon (1997). Island (Migration) Models: Evolutionary Algorithms Based on Punctuated Equilibria. In [26], Sect. C6.3.
301. Mathias, K. and D. Whitley (1993). Remapping Hyperspace During Genetic Search: Canonical Delta Folding. In [490], pp. 167-186.
302. Mathias, K. and D. Whitley (1992). Genetic Operators, the Fitness Landscape and the Traveling Salesman Problem. In [299], pp. 219-228.
303. McCulloch, W.S. and W. Pitts (1943). A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, Vol. 5, pp. 115-133.
304. McDonnell, J.R., R.G. Reynolds, and D.B. Fogel, eds. (1995). *Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA.
305. Merelo, J.-J., P. Adamidis, H.-G. Beyer, J.-L. Fernandez-Villacañas, and H.-P. Schwefel, eds. (2002). *Proceedings of the 7th Parallel Problem Solving from Nature Conference*. Lecture Notes in Computer Science, Vol. 2439, Springer, Berlin.
306. Merz, P. and B. Freisleben (1997). Genetic Local Search for the TSP: New Results. In [368], pp. 159-164.
307. Michalewicz, Z. (2003). *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. Wyd. 3. Wydawnictwa Naukowo-Techniczne, Warszawa.
308. Michalewicz, Z. (1995). Genetic Algorithms, Numerical Optimization and Constraints. In [130], pp. 151-158.
309. Michalewicz, Z. (1993). A Hierarchy of Evolution Programs: An Experimental Study. *Evolutionary Computation*, Vol. 1, No. 1, pp. 51-76.
310. Michalewicz, Z. (1994). Evolutionary Computation Techniques for Nonlinear Programming Problems. *International Transactions in Operational Research*, Vol. 1, No. 2, pp. 223-240.
311. Michalewicz, Z. (1995). Heuristic Methods for Evolutionary Computation Techniques. *Journal of Heuristics*, Vol. 1, No. 2, pp. 177-206.
312. Michalewicz, Z. and N. Attia (1994). Evolutionary Optimization of Constrained Problems. In [424], pp. 98-108.
313. Michalewicz, Z., D. Dasgupta, R.G. Le Riche, and M. Schoenauer (1996). Evolutionary Algorithms for Constrained Engineering Problems. *Computers & Industrial Engineering Journal*, Vol. 30, No. 4, pp. 851-870.
314. Michalewicz, Z., K. Deb, M. Schmidt, and T. Stidsen (1999). Towards Understanding Constrained-Handling Methods in Evolutionary Algorithms. In [370], pp. 583-590.

315. Michalewicz, Z., K. Deb, M. Schmidt, and T. Stidsen (2000). Test Case Generator for Constrained Parameter Optimization Techniques. *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 197-215.
316. Michalewicz, Z. and C. Janikow, C. (1996). GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints. *Communications of the ACM*, December, p. 118.
317. Michalewicz, Z. and G. Nazhiyath (1995). GENOCOP III: A Coevolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In [366], pp. 647-651.
318. Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems. In [163], pp. 305-312.
319. Michalewicz, Z. and M. Schoenauer (1996). Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation*, Vol. 4, No. 1, pp. 1-32.
320. Michalewicz, Z., G.A. Vignaux, and M. Hobbs (1991). A Non-standard Genetic Algorithm for the Nonlinear Transportation Problem. *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 307-316.
321. Miettinen, K. (1999). *Nonlinear Multiobjective Optimization*. Kluwer, Boston, MA.
322. Miller, G.A. (1956). The Magic Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *Psychological Review*, Vol. 63, pp. 81-97.
323. Minar, N., R. Burkhart, C. Langton, and M. Askenazi (1996). The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations. Working Paper, The Santa Fe Institute, rep. no.: 96-06-042.
324. Minsky, M.L. (1961). Steps Toward Artificial Intelligence. In *Proceedings of the IEEE*, Vol. 49, pp. 8-30.
325. Mizumoto, M. (1998). Defuzzification Methods. In [403], pp. B6.1:1-7.
326. Mori, N., S. Imanishi, H. Kita, and Y. Nishikawa (1997). Adaptation to Changing Environments by Means of the Memory Based Thermodynamical Genetic Algorithm. In [23], pp. 299-306.
327. Morris, P. (1993). The Breakout Method for Escaping from Local Minima. In *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI-93*, AAAI Press/The MIT Press, pp. 40-45.
328. Mühlenbein, H. (1992). How Genetic Algorithms Really Work: I. Mutation and Hill-Climbing. In [299], pp. 15-25.
329. Mühlenbein, H. (1991). Evolution in Time and Space - The Parallel Genetic Algorithm. In [380], pp. 316-337.
330. Mühlenbein, H., M. Gorges-Schleuter, and O. Krämer (1988). Evolution Algorithms in Combinatorial Optimization. *Parallel Computing*, Vol. 7, pp. 65-85.
331. Müller, J.P. (1998). Architectures and Applications of Intelligent Agents: A Survey. *The Knowledge Engineering Review*, Vol. 13, No. 4, pp. 353-380.

332. Myung, H., J.-H. Kim, and D.B. Fogel (1995). Preliminary Investigation Into a Two-Stage Method of Evolutionary Optimization on Constrained Problems. In [304], pp. 449-463.
333. Nadhamuni, P.Y.R. (1995). Application of Co-evolutionary Genetic Algorithm to a Game. Master's Thesis, Department of Computer Science, University of North Carolina, Charlotte, NC.
334. Nagata, Y. and S. Kobayashi (1997). Edge Assembly Crossover: A High-Power Genetic Algorithm for the Traveling Salesman Problem. In [23], pp. 450-457.
335. Ng, K.P. and K.C. Wong (1995). A New Diploid Scheme and Dominance Change Mechanism for Non-stationary Function Optimization. In [130], pp. 159-166.
336. Nix, A.E. and M.D. Vose (1992). Modelling Genetic Algorithms with Markov Chains. *Annals of Mathematics and Artificial Intelligence*, Vol. 5, pp. 79-88.
337. Nurnber, H.-T. and H.-G. Beyer (1997). The Dynamics of Evolution Strategies in the Optimization of Traveling Salesman Problem. In [10], pp. 349-359.
338. Oliver, I.M., D.J. Smith, and J.R.C. Holland (1987). A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In [200], pp. 224-230.
339. Orvosh, D. and L. Davis (1993). Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints. In [172], p.650.
340. Osyczka, A. (2002). *Evolutionary Algorithms for Single and Multicriteria Design Optimization*. Physica-Verlag, Heidelberg, Germany.
341. Osyczka, A. and S. Kundu (1995). A New Method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm. *Structural Optimization*, Vol.10, No. 2, pp. 94-99.
342. Padberg, M. and G. Rinaldi (1986). Optimization of a 532-City Symmetric Traveling Salesman Problem. Technical Report IASI-CNR, Italy.
343. Palmer, C.C. and A. Kershenbaum (1994). Representing Trees in Genetic Algorithms. In [365], pp. 379-384.
344. Papadimitriou, Ch.H. and K. Steiglitz (1982). *Combinatorial Optimization*. Prentice-Hall, Englewood Cliffs, NJ.
345. Papoulis, A. (1991). *Probability, Random Variables, and Stochastic Processes*. 3rd edition, McGraw-Hill, New York, NY.
346. *Parabola*, University of New South Wales, Vol. 14, pp. 30-31, 1978, problem PARAB 348.
347. Pardalos, P. (1994). On the Passage from Local to Global in Optimization. In *Mathematical Programming*, J.R. Birge and K.G. Murty, eds., University of Michigan, 1994.
348. Paredis, J. (1992). Exploiting Constraints as Background Knowledge for Genetic Algorithms: A Case-Study for Scheduling. In [299], pp. 229-238.
349. Paredis, J. (1993). Genetic State-Space Search for Constrained Optimization Problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, CA.

350. Paredis, J. (1994). Co-evolutionary Constraint Satisfaction. In [84], pp. 46-55.
351. Paredis, J. (1995). The Symbiotic Evolution of Solutions and Their Representations. In [130], pp. 359-365.
352. Paredis, J. (1999). Coevolutionary Algorithms. In *Handbook of Evolutionary Computation*, 2nd edition, T. Bäck, D.B. Fogel, and Z. Michalewicz, eds., Institute of Physics, pp. 224-238.
353. Parmee, I. and G. Purchase (1994). The Development of Directed Genetic Search Technique for Heavily Constrained Design Spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control*, University of Plymouth, UK, pp. 97-102.
354. Pearl, J. (1984). *Heuristics*. Addison-Wesley, Reading, MA.
355. Pollack, J.B. and A.D. Blair (1998). Co-evolution in the Successful Learning of Backgammon Strategy. *Machine Learning*, Vol. 32, pp. 225-240.
356. Pollack, J.B., A.D. Blair, and M. Land (1997). Coevolution of a Backgammon Player. In *Proceedings of the Fifth International Conference on Artificial Life*, MIT Press, Cambridge, MA, pp. 92-98.
357. Pólya, G. (1964). *Jak to rozwiązać. Nowy aspekt metody matematycznej*. PWN, Warszawa.
358. Poggio, T. and F. Girosi (1990). Networks for Approximation and Learning. *Proceedings IEEE*, Vol. 78, pp. 1481-1497.
359. Porto, V.W., N. Saravanan, D. Waagen, and A.E. Eiben, eds. (1998). *Evolutionary Programming VII*, Lecture Notes in Computer Science, Vol. 1447, Springer, Berlin.
360. Potter, M. and K.A. De Jong (1994). A Cooperative Coevolutionary Approach to Function Optimization. In [84], pp. 249-257.
361. Potter, M.A. and K.A. De Jong (2000). Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, Vol. 8, No. 1, pp. 1-29.
362. Powell, D. and M.M. Skolnick (1993). Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints. In [172], pp. 424-430.
363. Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vettering (1989). *Numerical Recipes in Pascal: The Art of Scientific Computing*. Cambridge University Press, Cambridge, MA.
364. Price, K.V. (1997). Differential Evolution vs. the Functions of the 2nd ICEO. In [368], pp. 153-157.
365. *Proceedings of the 1994 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1994.
366. *Proceedings of the 1995 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1995.
367. *Proceedings of the 1996 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1996.
368. *Proceedings of the 1997 IEEE Conference on Evolutionary Computation*. IEEE Press, Piscataway, NJ, 1997.

369. *Proceedings of the 1998 IEEE Conference on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1998.
370. *Proceedings of the 1999 Congress on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1999.
371. *Proceedings of the 2000 Congress on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1999.
372. *Proceedings of the 2001 Congress on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1999.
373. *Proceedings of the 2002 Congress on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1999.
374. *Proceedings of the 2003 Congress on Evolutionary Computation.* IEEE Press, Piscataway, NJ, 1999.
375. Radcliffe, N.J. (1991). Equivalence Class Analysis of Genetic Algorithms. *Complex Systems*, Vol. 5, No. 2, pp. 183-205.
376. Ramsey, C.L. and J.J. Grefenstette (1993). Case-Based Initialization of Genetic Algorithms. In 172 , pp. 84-91.
377. Ranjithan, S., J.W. Eheart, and J.C. Liebman (1992). Incorporating Fixed-Cost Component of Pumping into Stochastic Groundwater Management: A Genetic Algorithm-Based Optimization Approach. *Eos Transactions AGU*, Spring meeting supplement, Vol. 73, No. 14, p.125.
378. Ratschek, H. and J. Rokne (1995). Global Unconstrained Optimization. In *Handbook of Global Optimization*, R. Horst and P.M. Pardalos, eds., Kluwer Publishing, Norwell, MA, pp. 779-796.
379. Raup, D.M. (1991). *Extinction: Bad Genes or Bad Luck?* W.W. Norton and Company, New York, NY .
380. Rawlins, G.J.E., ed. (1991). *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA.
381. Ray, T., K. Tai, and K.C. Seow (2001). An Evolutionary Algorithm for Multi-objective Optimization. *Engineering Optimization*, Vol. 33, No. 3, pp. 399-424.
382. Ray, T.S. (1992). An Approach to the Synthesis of Life. In *Artificial Life II*, C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, eds., Addison-Wesley, Reading, MA, pp. 371-408.
383. Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Verlag, Stuttgart.
384. Reed, J., R. Toombs, and N.A. Barricelli (1967). Simulation of Biological Evolution and Machine Learning. I. Selection of Self-Reproducing Numeric Patterns by Data Processing Machines, Effects of Hereditary Control, Mutation Type and Crossing. *Journal of Theoretical Biology*, Vol. 17, pp. 319-342.
385. Reinelt, G. (1991). TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal on Computing*, Vol. 3, No. 4, pp. 376-384.
386. Resende, M.G.C. and J.P. de Sousa, eds. (2003). *Metaheuristics: Computer Decision-Making.* Kluwer Academic Publishers, New York, NY .

387. Reynolds, C. (1994). Competition, Coevolution and the Game of Tag. In *Proceedings of Artificial Life IV*, R. Brooks and P. Maes, eds., MIT Press, Cambridge, MA, pp. 56-69.
388. Reynolds, R.G. (1994). An Introduction to Cultural Algorithms. In 424 , pp. 131-139.
389. Reynolds, R.G., Z. Michalewicz, and M. Cavaretta (1995). Using Cultural Algorithms for Constraint Handling in GENOCOP. In 304 , pp. 298-305.
390. Reynolds, R.G. and W. Sverdlik (1993). Solving Problems in Hierarchically Structured Systems Using Cultural Algorithms. In 153 , pp. 144-153.
391. Richardson, J.T., M.R. Palmer, G. Liepins, and M. Hilliard (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In 409 , pp. 191-197.
392. Riesbeck, C.K. and R.C. Schank (1989). *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ.
393. Romero, C. (1991). *Handbook of Critical Issues in Goal Programming*. Pergamon Press, Oxford, UK.
394. Ronald, E. (1995). When Selection Meets Seduction. In 130 , pp. 167-173.
395. Rosenberg, R.S. (1967). Simulation of Genetic Populations with Biochemical Properties. PhD Dissertation, University of Michigan, Ann Arbor, MI. *Dissertation Abstracts International*, 28(7), 2732B. (University Microfilms No. 67-17, 836).
396. Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, Vol. 65, pp. 386-408.
397. Rosin, C.D. and R.K. Belew (1995). Methods for Competitive Co-evolution: Finding Opponents Worth Beating. In 130 , pp. 373-380.
398. Rosin, C.D. and R.K. Belew (1997). New Methods for Competitive Evolution. *Evolutionary Computation*, Vol. 5, pp. 1-29.
399. Rudolph, G. (1996). Convergence of Evolutionary Algorithms in General Search Space. In 367 , pp. 50-54.
400. Rudolph, G. (1998). On a Multi-Objective Evolutionary Algorithm and Its Convergence to the Pareto Set. In 369 , pp. 511-516.
401. Rudolph, G. (2001). Evolutionary Search under Partially Ordered Fitness Sets. In *Proceedings of the International Symposium on Information Science Innovations in Engineering of Natural and Artificial Intelligent Systems (ISI 2001)*, pp. 818-822.
402. Rudolph, G. and J. Sprave (1995). A Cellular Genetic Algorithm with Self-Adjusting Acceptance Threshold. In *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, IEE, London, pp. 365-372.
403. Rupsini, E.H., P.P. Bonissone, and W. Pedrycz, eds. (1998). *Handbook of Fuzzy Computation*, IOP, Philadelphia, PA.

404. Ryan, C. (1996). The Degree of Oneness. In *Proceedings of the First Online Workshop on Soft Computing*, Furuhashi, T., Morikawa, K., Miyata, . and Tkeuchi, I., eds., pp. 43-48.
405. Sakawa, M. (1993). *Fuzzy Sets and Interactive Multiobjective Optimization*. Plenum Press, New York, N .
406. San Pedro, J. and F. Burstein (2003). A Framework for Case-Based Fuzzy Multicriteria Decision Support for Tropical Cyclone Forecasting. In *Proceedings of the 36th Hawaii International Conference on Systems Sciences*, IEEE, New York, N .
407. Saravanan, N. and D.B. Fogel (1994). Learning Strategy Parameters in Evolutionary Programming: An Empirical Study. In 424 , pp. 269-280.
408. Saravanan, N., D.B. Fogel, and K.M. Nelson (1995). A Comparison of Methods for Self-Adaptation in Evolutionary Algorithms. *BioSystems*, Vol. 36, pp. 157-166.
409. Schaffer, J.D., ed. (1989). *Proceedings of the 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA.
410. Schaffer, J.D. (1984). Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms. PhD Dissertation, Vanderbilt University, Nashville, TN.
411. Schaffer, J.D. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In 199 , pp. 93-100.
412. Schaffer, J.D., R. Caruana, L. Eshelman, and R. Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In 409 , pp. 51-60.
413. Schaffer, J.D. and A. Morishima (1987). An Adaptive Crossover Distribution Mechanism for Genetic Algorithms. In 200 , pp. 36-40.
414. Schoenauer, M., K. Deb, G. Rudolph, X. Yao, E. Lutton, J.-J. Merelo, and H.-P. Schwefel, eds. (2000). *Proceedings of the 6th Parallel Problem Solving from Nature Conference*. Lecture Notes in Computer Science, Vol. 1917, Springer, Berlin.
415. Schoenauer, M. and Z. Michalewicz (1996). Evolutionary Computation at the Edge of Feasibility. In 480 , pp. 245-254.
416. Schoenauer, M. and S. Xanthakis (1993). Constrained GA Optimization. In 172 , pp. 573-580.
417. Schraudolph, N. and R. Belew (1992). Dynamic Parameter Encoding for Genetic Algorithms. *Machine Learning*, Vol. 9, No. 1, pp. 9-21.
418. Schwefel, H.-P. (1981). *Numerical Optimization for Computer Models*, John Wiley, Chichester, UK.
419. Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*, John Wiley, New York, N .
420. Schwefel, H.-P. and R. Manner, eds. (1991). *Proceedings of the 1st Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science, Vol. 496, Springer, Berlin.

421. Sebald, A.V. and J. Schlenzig (1994). Minima Design of Neural-net Controllers for Uncertain Plants. *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, pp. 73-82.
422. Sebald A.V. and K. Chellapilla (1998). On Making Problems Evolutionarily Friendly. Part 1: Evolving the Most Convenient Representations. In 359 , pp. 271-280.
423. Sebald, A.V. and D.B. Fogel (1992). Design of Fault Tolerant Neural Networks for Pattern Classification. In 153 , pp. 90-99.
424. Sebald, A.V. and L.J. Fogel, eds. (1994). *Proceedings of the Third Annual Conference on Evolutionary Programming*, World Scientific, River Edge, NJ.
425. Sedgewick, R. (1988). *Algorithms*. 2nd edition, Addison-Wesley, Reading, MA.
426. Sedgewick, R. (1977). Permutation Generation Methods. *Computing Surveys*, Vol. 9, No. 2, pp. 137-164.
427. Selman, B., H.A. Kautz, and B. Cohen (1993). Local Search Strategies for Satisfiability Testing. Presented at the 2nd DIMACS Challenge on Cliques, Coloring, and Satisfiability, Rutgers University, NJ.
428. Selman, B., H.J. Levesque, and D.G. Mitchell (1992). A New Method for Solving Hard Satisfiability Problems. In *Proceedings AAAI-92*, San Jose, CA, pp. 440-446.
429. Selman, B. and H.A. Kautz (1993). Domain-Independent E tensions to GSAT: Solving Large Structured Satisfiability Problems. In *Proceedings IJCAI-93*, Chambery, France.
430. Seniw, D. (1991). A Genetic Algorithm for the Traveling Salesman Problem. MSc Thesis, University of North Carolina at Charlotte, NC.
431. Shaefer, C.G. (1987). The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique. In 200 , pp. 50-55.
432. Shapiro, J.F. (1979). *Mathematical Programming*. John Wiley, Chichester, UK.
433. Siegel, S. (1956). *Non-parametric Statistics*. McGraw-Hill, New York, N .
434. Śmierzchalski, R. and Z. Michalewicz (2000). Modeling of Ship Trajectory in Collision Situations by an Evolutionary Algorithm. *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 3, pp. 227-241.
435. Smith, A. and D. Tate (1993). Genetic Optimization Using A Penalty Function. In 172 , pp. 499-503.
436. Smith, J.E. (1997). Self Adaptation in Evolutionary Algorithms. PhD thesis, University of the West of England, Bristol, UK.
437. Smith, J.E. and T.C. Fogarty (1996). Self-Adaptation of Mutation Rates in a Steady-State Genetic Algorithm. In 367 , pp. 318-323.
438. Smith, J.E. and T.C. Fogarty (1996). Adaptively Parameterised Evolutionary Systems: Self Adaptive Recombination and Mutation in a Genetic Algorithm. In 480 , pp. 441-450.
439. Smith, J.E. and T.C. Fogarty (1997). Operator and Parameter Adaptation in Genetic Algorithms. *Soft Computing*, Vol. 1, No. 2, pp. 81-87.

440. Smith, R. (1993). Adaptively Resizing Populations: An Algorithm and Analysis. In 172 , p. 653.
441. Smith, R. (1997). Population Size. In 26 , pp. E1.1:1-5.
442. Soule, T. and J.A. Foster (1997). Code Size and Depth Flows in Genetic Programming. In 270 , pp. 313-320.
443. Spears, W.M. (1995). Adapting Crossover in Evolutionary Algorithms. In 304 , pp. 367-384.
444. Spears, W.M. (1996). Simulated Annealing for Hard Satisfiability Problems. In *Cliques. Coloring, and Satisfiability: Second DIMACS Implementation Challenge*, D.S. Johnson and M.A. Trick, eds., DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26, American Mathematical Society, pp. 533-558.
445. Spears, W.M. (1994). Simple Subpopulation Schemes. In 424 , pp. 296-307.
446. Spears, W.M. and K.A. De Jong (1991). On the Virtues of Parametrized Uniform Crossover. In 39 , pp. 230-236.
447. Spector, L. et al. (2001). *Proceedings of the Genetic and Evolutionary Conference 2001*, Morgan Kaufmann, San Mateo, CA.
448. Srinivas, N. and K. Deb (1994). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, Vol. 2, No. 3, 1994, pp. 221-248.
449. Srinivas, M. and L.M. Patnaik (1994). Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 24, No. 4, pp. 17-26.
450. Standish, R., M.A. Bedau, and H. Abbass, eds. (2003). *Artificial Life VII: Proceedings of the Eighth International Conference on Artificial Life*. MIT Press, Cambridge, MA.
451. Stanley, K.O. and R. Miikkulainen (2002). Continual Coevolution through Complexification. In *Proceedings 2002 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, San Francisco, CA, pp. 113-120.
452. Starkweather, T., S. McDaniel, K. Mathias, C. Whitley, and D. Whitley (1991). A Comparison of Genetic Sequencing Operators. In 39 , pp. 69-76.
453. Steele, J.M. (1986). Probabilistic Algorithm for the Directed Traveling Salesman Problem. *Mathematics of Operations Research*, Vol. 11, No. 2, pp. 343-350.
454. Stein, D. (1977). Scheduling Dial a Ride Transportation Systems: An Asymptotic Approach. PhD Dissertation, Harvard University, MA.
455. Steinhaus, H. (1964). *One Hundred Problems in Elementary Mathematics*. Dover Publications, New York, NY .
456. Stewart, I. (1999) A Puzzle for Pirates. *Scientific American*, May 1999, pp. 98-99.
457. Suh, J.- . and Van D. Gucht (1987). Incorporating Heuristic Information into Genetic Search. In 200 , pp. 100-107.

458. Surry, P.D., N.J. Radcliffe, and I.D. Boyd (1995). A Multi-Objective Approach to Constrained Optimization of Gas Supply Networks. In *Proceedings of the AISB-95 Workshop on Evolutionary Computing*, T. Fogarty, ed., Lecture Notes in Computer Science, Vol. 993, Springer, Berlin, pp. 166-180.
459. Suzuki, J. (1993). A Markov Chain Analysis on a Genetic Algorithm. In 172 , pp. 146-153.
460. Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. In 409 , pp. 2-9.
461. Syswerda, G. (1991). Schedule Optimization Using Genetic Algorithms. In 91 , pp. 332-349.
462. Syswerda, G. and J. Palmucci (1991). The Application of Genetic Algorithms to Resource Scheduling. In 39 , pp. 502-508.
463. Tao, G. and Z. Michalewicz (1998). Evolutionary Algorithms for the TSP. In 113 , pp. 803-812.
464. Tate, D.M. and E.A. Smith (1993). Expected Allele Coverage and the Role of Mutation in Genetic Algorithms. In 172 , pp. 31-37.
465. Taylor, R. and A. Wiles (1995). Ring-Theoretic Properties of Certain Hecke Algebras. *Annals of Mathematics*, Vol. 142, pp. 553-573.
466. Thierens, D. (1996). Dimensional Analysis of Allele-Wise Mixing Revisited. In 480 , pp. 255-265.
467. Thierens, D. and D.E. Goldberg (1993). Mixing in Genetic Algorithms. In 172 , pp. 38-45.
468. Trojanowski, K., Z. Michalewicz, and J. Xiao (1997). Adding Memory to the Evolutionary Planner Navigator. In 368 , pp. 483-487.
469. Trojanowski, K. and Z. Michalewicz (1999). Searching for Optima in Non-stationary Environments. In 370 , pp. 1845-1852.
470. Tuson, A. and P. Ross (1996). Cost Based Operator Rate Adaptation: An Investigation. In 480 , pp. 461-469.
471. Ulder, N.L.J., E.H.L. Aarts, H.-J. Bandelt, P.J.M. van Laarhoven, and E. Pesch (1990). Genetic Local Search Algorithms for the Traveling Salesman Problem. In 420 , pp. 109-116.
472. Ursem, R.K., T. Krink, M.T. Jensen, and Z. Michalewicz (2000). Analysis and Modeling of Control Tasks in Dynamic Systems. *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 4, pp. 378-389.
473. Valenzuela, C.L. and A.J. Jones (1994). Evolutionary Divide and Conquer (I): A Novel Genetic Approach to the TSP. *Evolutionary Computation*, Vol. 1, No. 4, pp. 313-333.
474. Valenzuela, C.L. and L.P. Williams (1997). Improving Simple Heuristic Algorithms for the Traveling Salesman Problem Using a Genetic Algorithm. In 23 , pp. 458-464.
475. Valenzuela-Rendón, M. and E. Uresti Charre (1997). A Non-generational Genetic Algorithm for Multiobjective Optimization. In 23 , pp. 658-665.

476. Van Laarhoven, P.J.M. and E.H.L. Aarts (1987). *Simulated Annealing: Theory and Applications*. D. Reidel, Dordrecht, The Netherlands.
477. Vavak, F., T.C. Fogarty, and K. Jukes (1996). A Genetic Algorithm with Variable Range of Local Search for Tracking Changing Environments. In 480 , pp. 376-385.
478. Vavak, F., K. Jukes, and T.C. Fogarty (1997). Learning the Local Search Range for Genetic Optimisation in Nonstationary Environments. In 368 , pp. 355-360.
479. Veldhuizen, D.V. (1999). Multiobjective Evolutionary Algorithms: Classifications, Analysis, and New Innovations. PhD Dissertation, Air Force Institute of Technology, Dayton, OH.
480. Voigt, H.-M., W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds. (1996). *Proceedings of the 4th Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science, Vol. 1141, Springer, Berlin.
481. Von Neumann, J. and O. Morgenstern (1944). *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ.
482. Vose, M.D. (1992). Modeling Simple Genetic Algorithms. In 490 , pp. 63-74.
483. Voss, S. (1993). Tabu Search: Applications and Prospects. Technical Report, Technische Hochshule, Darmstadt, Germany.
484. Waagen, D., P. Diercks, and J. McDonnell (1992). The Stochastic Direction Set Algorithm: A Hybrid Technique for Finding Function E trema. In 152 , pp. 35-42.
485. Watson, J., C. Ross, V. Eisele, J. Denton, J. Bins, C. Guerra, D. Whitley, and A. Howe (1998). The Traveling Salesrep Problem, Edge Assembly Crossover, and 2-opt. In 113 , pp. 823-832.
486. White, T. and F. Oppacher (1994). Adaptive Crossover Using Automata. In 84 , pp. 229-238.
487. Whitley, D. (1988). GENITOR: A Different Genetic Algorithm. *Proceedings of the Rocky Mountain Conference on Artificial Intelligence*, Denyer, CO.
488. Whitley, D. (1989). The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In 409 , pp. 116-121.
489. Whitley, D. (1990). GENITOR II: A Distributed Genetic Algorithm. *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 2, pp. 189-214.
490. Whitley, L.D., ed. (1992). *Foundations of Genetic Algorithms 2*, Morgan Kauffman, San Mateo, CA.
491. Whitley, D., V.S. Gordon, and K. Mathias (1996). Lamarckian Evolution, the Baldwin Effect and Function Optimization. In 84 , pp. 6-15.
492. Whitley, D., K. Mathias, and P. Fitzhorn (1991). Delta Coding: An Iterative Strategy for Genetic Algorithms. In 39 , pp. 77-84.
493. Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building Better Test Functions. In 130 , pp. 239-246.
494. Whitley, D., T. Starkweather, and D'A Fuquay (1989). Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In 409 , pp. 133-140.

495. Whitley, D., T. Starkweather, and D. Shaner (1990). Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination. In 91 , pp. 350-372.
496. Whitley, D. et al. (2000). *Proceedings of the Genetic and Evolutionary Conference 2000*, Morgan Kaufmann, San Mateo, CA.
497. Wienke, P.B., C. Lucasius, and G. Kateman (1992). Multicriteria Target Optimization of Analytical Procedures Using a Genetic Algorithm. *Analytical Chimica Acta*, Vol. 265, No. 2, pp. 211-225.
498. Wiles, A. (1995). Modular Elliptic Curves and Fermat's Last Theorem. *Annals of Mathematics*, Vol. 142, pp. 443-551.
499. Wolpert, D.H. and W.G. Macready (1997). No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 67-82.
500. Wu, A., R.K. Lindsay, and R.L. Riolo (1997). Empirical Observation on the Roles of Crossover and Mutation. In 23 , pp. 362-369.
501. Xiao, J., Z. Michalewicz, L. Zhang, and K. Trojanowski (1997). Adaptive Evolutionary Planner Navigator for Mobile Robots. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 18-28.
502. ager, R.R. (1988). On Ordered Weighted Averaging Aggregation Operators in Multicriteria Decision Making. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 18, pp. 183-190.
503. ager, R.R., D.P. Filev (1995). *Podstawy modelowania i sterowania rozmytego*, Wydawnictwa Naukowo-Techniczne, Warszawa.
504. ao, X., G. Lin, and . Liu (1997). An Analysis of Evolutionary Algorithms Based on Neighbourhood and Step Sizes. In 10 , pp. 297-307.
505. Zadeh, L.A. (1965). Fuzzy Sets. *Information and Control*, Vol. 8, pp. 338-352.
506. Zadeh, L.A. (1973). Outline of a New Approach to the Analysis of Complex Systems and Decision Processes. *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-3:1, pp. 28-44.
507. Zitzler, E., K. Deb, and L. Thiele (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, Vol. 8, No. 2, pp. 173-195.
508. Zitzler, E., K. Deb, L. Thiele, C.A. Coello Coello, and D. Corne, eds. (2001). *Evolutionary Multi-Criterion Optimization. First International Conference EMO 2001*. Lecture Notes in Computer Science, Vol. 1993, Springer, Berlin.
509. Zitzler, E. and L. Thiele (1998). Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In 113 , pp. 292-301.

Skorowidz

- α -cięcie 432, 433
- δ -ścieżka 94
- ϵ -ograniczenie 499, 512

- AB*-cykl 255
- ACO 535
 - adaptacyjna teoria rezonansu 410
 - adaptacyjne sterowanie parametrami 328
 - agent 472
 - autonomiczny 537
 - inteligentny 472
 - komunikujący się 538, 539
 - mobilny 533
 - reaktywny 537, 539
 - wnioskujący 537, 539
 - algorytm 2-opt 93, 252, 257
 - 3-opt 252
 - A^* 135
 - adaptacyjny 187
 - ewolucyjny 184, 198, 320, 342, 356, 516
 - – dziel i rządź 254
 - – stanu stabilnego 216
 - – z segregacją 297
 - genetyczny 184
 - – klasyczny 342
 - – sortowania bez zdominowania 513
 - ANN 382
 - ARMA 512
 - ARMAX 512
 - arytmetyka rozmyta 433
 - aspiracja zgodna z domyślnym ruchem 160
 - z kierunkiem poszukiwań 160
 - – z wpływem 161
 - automat komórkowy 204
 - Mealy'ego 203
 - ze skońzoną liczbą stanów 203, 211, 468, 512

- bijekcja 201
- binarna macierz pierwszeństwa 242
- bisekcja 98
- błąd drugiego rodzaju 408, 573
 - pierwszego rodzaju 408, 573
 - typu I 408, 573
 - typu II 408, 573
- cel 61, 63
- centralne twierdzenie graniczne 562
- centroid 429
- chromosom 173
- ciąg liczb pseudolosowych 568
- CX 234, 236
- cykl 230
- czas wielomianowy 145

- dekoder 281, 305
- deterministyczne sterowanie parametrem 328
- diagram Venna 556
- dopuszczalność 137
- dowód przez sprzeczność 49
- drzewo analizy syntaktycznej 212
 - poszukiwań binarnych 86

- EAX 254
- EEX 254
- efekt Baldwina 189, 278
- eksperyment 555
- eksploracja 171
 - danych 403
- eliminacja Gaussa 104
- EP/N 370
- ER 239
- estymator nieobciążony 570
 - obciążony 570
- Evolutionary Planner/Navigator 370
- ewolucja lamarkowska 188, 251, 278
 - różnicowa 536
 - sieci neuronowych 408
- E-zbiór 255

- fałszywy alarm 408, 573
- fenotyp 188

- funkcja bazowa radialna 410
 - charakterystyczna 416
 - dyskryminacyjna, liniowa 387, 388
 - esowata 392
 - gęstości 182
 - – gaussowska 396
 - – prawdopodobieństwa 561
 - gradientu 102
 - kary 183, 276, 290
 - masy prawdopodobieństwa 561
 - oceny 35, 48, 61, 63, 97, 176, 204, 206, 327, 328, 355
 - – dla osobników niedopuszczalnych 275
 - – numeryczna 64
 - – pochodna 68
 - – porządkująca 63
 - odległości 66
 - przynależności 416
 - sigmoidalna 392, 395
 - signum 340
 - wiarygodności 568

- GENOCOP 285, 286
 - II 292
 - III 278, 302, 460
- genotyp 188
- geometria euklidesowa 179
- gęstość prawdopodobieństwa 561
- gradient 103
- granica Helda-Karpa 260
 - Pareto 508
- gra z sumą zerową 452
- grupowanie rozmyte 429
- GSAT 152

- hesjan 103
- heurystyka najbliższego sąsiada 117
- hipoteza alternatywna 573
 - zerowa 572
- hybrydyzacja 251, 529

- ICG 257
- informacja lokalna 68
- inicjowanie 217, 288

- IPD 466, 469
iterowany dylemat więźnia 376, 466, 468
- kara 279, 295
– dostosowująca się 295
– dynamiczna 291
– statyczna 290
– śmierci 277, 296
– wyżarzana 292
- klauzula 93
- kodowanie binarne 518
- koewolucja 338, 453, 459, 464, 473, 474, 539
– ze współpracą 478
– z konkurencyjnością 474
- kombinacja 559
– liniowa 286
- korelacja liniowa 575
- kryterium aspiracji 159
– informacyjne Akaike 512
– Metropolis 338
- krzywe wypełniające przestrzeń 226
- krzyżowanie 174, 208, 210, 258, 288, 336
– arytmetyczne 209, 320, 328
– cykliczne 234
– dwupunktowe 249, 320, 518
– geometryczne 209, 288
– heurystyczne 230
– jednopunktowe 185, 249, 320, 467
– jednostajne 320
– krawędziowe 239
– macierzowe 249
– oparte na kolejności 236
– – na pozycjach 237
– przez naprzemienne wybieranie krawędzi 230
– przez wymianę podtras 230
– sferyczne 289
– z częściowym odwzorowaniem (PMX) 234
– ze składaniem krawędzi (EAX) 254
– z wymianą krawędzi (EEX) 254
– – podtras (SXX) 254
– z zachowaniem porządku (OX) 234, 235
- kurtoza 565
- liczba rozmyta 432, 433
- Lisp 461
- lista priorytetowa kryteriów 510
– tabu 536
- logika rozmyta 506, 507
- lokalna optymalizacja 226, 251
- łańcuch Markowa 473
- macierz Hessego 103
– pierwszeństwa, binarna 242
- maszyna Boltzmanna 401
- mechanizm samoadaptujących się rozmiarów kroków mutacji 326
- selekcji Boltzmanna 338
- mediana 563
- metoda agregacji wielu celów 497
– Brenta 104, 362
– ϵ -ograniczeń 499
– funkcji odległości 498
– gradientowa 283
– hybrydowa 308
– kary śmierci 277, 296
– Lagrange'a 308
– metryczna z wagami 498
– najbliższego sąsiada 226
– Newtona 100
– Newtona-Gaussa 104
– pamięci behawioralnej 297
– podziału i ograniczeń 87, 131
– poszukiwania, lokalna 69
– programowania celu 499
– prób i błędów 319, 437
– przyblizonego wnioskowania Mamdaniego 421
– quasi-newtonowska 104
– roju cząsteczek 536
– rozmytych c -średnich 429
– selekcji z kryterium 509
– siecznych 102
– sympleks 105
– średniej arytmetycznej 497
– testowania hipotez, nieparametryczna 574
– turniejowa 275

metoda wielokrotnego generowania dziecka 257
 – wspinania się 69, 71
 – współdzielenia wartości 511
 – zachłanna 124
 – zstępowania 69
 – z zapewnieniem przewagi punktów dopuszczalnych 300
 metody analityczne 98
 – deterministyczne 98
 – losowe 98
 – numeryczne 98
 – oparte na pochodnych 98
 – optymalizacyjne gradientowe 82
 metryka euklidesowa 227
 miara probabilistyczna 417
 migracja 338, 531
 – osobników 188
 MIP 403
 mnożenie macierzy 124
 mnożnik Lagrange'a 309
 moda zmiennej 562
 model 542
 – aktywności neuronów 386
 – autoregresywny 209
 – drobnoziarnisty 532
 – dyfuzyjny 532
 – gruboziarnisty 530
 – koewolucyjny 188, 284
 – liniowy 43, 217
 – migracyjny 530
 – patchworkowy 533
 – problemu 35, 40
 – średnich krocących autoregresji 512
 – wyspowy 530
 – z sąsiedztwem 532
 modyfikator stopniujący 420
 MOGA 514, 518, 519
 moment centralny n -ty 565
 – zmiennej losowej 565
 MPX 252
 mutacja 288, 337, 467
 – Gaussa 185, 213, 322, 325
 – jednowierzchołkowa 213
 – niejednostajna 335
 – obcinająca 213
 – powiększająca 213
 – przez przełączenie 185

mutacja zamieniająca 213
 MX 249
 naprzemienne wybieranie krawędzi 230
 neurony progowe 384
 nierówność Czebyszewa 567
 nisza 513
 niszowanie 366
 niszowy genetyczny algorytm Pareto 514
 NLP 38, 62, 83, 152, 181, 270, 582, 583, 584
 NPGA 514
 NSGA 514
 obszar krytyczny 573
 – nieroziążywalny 39
 – przyciągania 70
 ocenianie jakości rozwiązań 207
 odchylenie standardowe 155, 562, 564
 oddzielanie osobników i ograniczeń 282
 odległość euklidesowa 67, 209
 – Hamminga 67, 68, 208, 332, 368
 odwzorowanie jednego przełączenia 67
 – zamiany w parze 67
 ograniczenie 46
 – aktywne 97
 – liniowe 285, 286
 – łagodne 47, 48
 operator 46
 – cięcia i łączenia 233
 – graniczny 284
 – heurystycznej inwersji 250
 – iloczynu 243
 – inver-over 238, 257, 258
 – inwersji 237
 – k -krawędziowy 210
 – krzyżowania 229, 234–238
 – logiki większościowej 320
 – naprawy 228
 – odwracania 185
 – rekombinacji 178, 229, 251
 – – macierzowej 249
 – – skanujący 284

- operator różnicowania 179, 208, 211, 228, 238, 280, 328
 - selekcji 328
 - sumy 243, 244
 - transpozycji 185
 - zamiany 185
- opis rozmyty 496
- optimum lokalne 68, 92, 145
- optymalizacja 284
 - kolonii mrówek 535
 - kombinatoryczna 225, 283, 407
 - lokalna 226, 251
 - numeryczna 71
 - regulatora CMAC 459
 - wielokryterialna 282, 495, 516, 521
 - w sensie Pareto 495, 516
- optymalne prawdopodobieństwo operatorów 323
- optymalny rozmiar populacji 323
- oscylacja strategiczna 282
- oscylator jednostronny 283
- osobnik dopuszczalny 276, 277
 - niedopuszczalny 275, 276, 277
- otoczenie 66, 69, 174
- OWA 504
- OX 235, 253

- pamięć 156
 - asocjacyjna 401
 - behawioralna 282, 297
 - częstości użycia 159
 - niedawnych wartości 159
- parametr 187
- perceptron 386
 - wielowarstwowy 396
- permutacja 37, 61, 179, 204, 209, 228, 559
 - indeksowana 480
- plateau 93
- PMX 234
- pochodna kierunkowa 102
- podejmowanie decyzji, wielokryterialne 499, 506
 - podejście ewolucyjne 178
 - minimaksowe 498
 - z pamięcią behawioralną 282
 - z wektorem celu 498
- podstawa 200
- podwójne minimalne drzewa rozpinające 226
- podział na pasy 226
- pominięcie 408
- poprawianie niedopuszczalnych osób 302
- rozwiązań 307
- populacja 328, 339
 - początkowa 175, 179, 217
- porządek leksykograficzny 512
- poszukiwanie na ślepo 200
 - z tabu 186
- poziom istotności 573
 - krytyczny 283
- prawdopodobieństwo operatorów, optymalne 323
- sukcesu 559
- warunkowe 556
- prawo kazirodzka k -tego stopnia 253
 - wielkich liczb 566
- problem Czebyszewa z wagami 498
 - dyliżansu 122
 - El Farol 472
 - holistyczny 479
 - identyfikacji systemu 512
 - komiwojażera *zob.* TSP
 - liniowo separowalny 323
 - minimalizacji największego odchylenia 498
 - NP-trudny 145, 225
 - optymalizacyjny 65
 - planowania 71
 - plecakowy 278
 - pokrycia zbiorami 278
 - programowania nieliniowego 38, 62
 - punktu stałego 101
 - rozpoznawania wzorców 403
 - spełnialności formuł boolowskich 36, 61
 - spełniania ograniczeń 284
 - wykrywania 403
 - z ograniczeniami 97
 - – równościowymi 97
 - – w postaci nierówności 97
 - procedura GSAT 92
 - Newtona-Gaussa 198
 - podziału i ograniczania 231

- procedura transpozycyjna 181
- programowanie dynamiczne 121, 126
 - ewolucyjne 184
 - genetyczne 185
 - liniowe 105
 - nieliniowe 270
 - ciągłe 297
- propagacja wsteczna 392
- próbkowanie jednostajne 209
 - zgodnie z rozkładem dwumianowym 209
- przedział ufności 571
- przekonanie 534
- przekształcenie unarne 184
- przenoszenie 238
- przesłanka statystycznie istotna 573
- przestrzeń euklidesowa 179
 - poszukiwań 35, 61, 62, 84, 97, 131
 - , dopuszczalna część 65, 97, 284
 - próbkowa 555
 - przekonań 309
 - rozwiązań dopuszczalnych, wypukła 287
 - stanów, wartościowana 499
- przeszukiwanie lokalne 91, 98, 103, 146
 - typu „najpierw najlepszy” 135, 136
 - w głąb 87
 - wyczerpujące 82, 85, 131
 - z tabu 82, 145, 146, 147, 155, 161
- przybliżenie 43
- punkt krzyżowania 249, 337
 - siodłowy 430
 - stały 401

- Regula Falsi 99
- regulator rozmyty 204, 423, 428
- reguła 5% 279
 - jednej piątej Rechenberga 325, 335
 - – – sukcesów 322
 - minimalnej kary 279
- relacja rozmyta 420
- relaksacja stochastyczna 150
- reprezentacja 61, 84, 200, 201, 202, 203, 280, 331
 - binarna 83, 228, 370
 - ciągła 322
 - osobników 328
- reprezentacja porządkowa 229, 232
 - ścieżkowa 229, 234, 236
 - w postaci listy sąsiedztwa 229
 - macierzy binarnej 242, 245, 249
 - zmiennopozycyjna 308
- robot typu Khepera 460
- rozdzielcość 90
- rozkład Cauchy'ego 572
 - Gaussa 154, 155, 568, 569, 575
 - jednostajny 568
 - t 562, 572, 574
 - wykładniczy 568
- rozmiar populacji, optymalny 323
- rozwiązywanie dopuszczalne 35, 61, 64, 270, 297
 - globalne 145
 - niedopuszczalne 278, 297
 - niezdominowane 495
 - zdominowane 495
- różnicowanie Gaussa 183, 308

- samoadaptacja 342
- samoadaptujące się sterowanie parametrem 329
- samoorganizujące się odwzorowania cech 410
- SA-SAT 152
- SAT 36, 61, 84, 152, 155, 175, 582
- sąsiedztwo lokalne 91
- schemat schładzania 294
- selekcja 214
 - deterministyczna 214
 - elitarna 216, 245, 515
 - leksykalna 510
 - Pareto 509
 - proporcjonalna 177, 215, 216, 289, 467, 518
 - rankingowa 183
 - stochastyczna 215
 - turniejowa 216, 301
 - według koła ruletki 177
- sieć Hopfielda 400
 - neuronowa 204, 386
 - – rozmyta 432, 435
 - – ze sprzężeniem do przodu 394, 396, 454
 - radialna 396

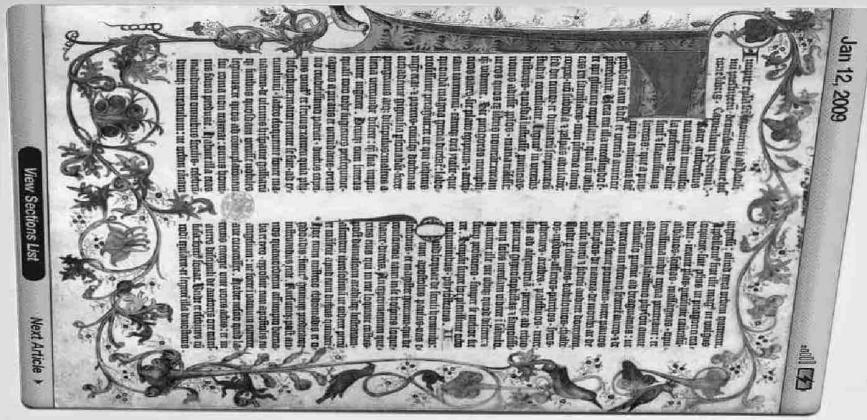
- sieć rekurencyjna 399
 - prosta 399
 - sortująca 456
 - wielu współpracujących programów 402
 - ze sprzężeniem do przodu 396, 399
- silna dominacja 495
- skośność 565
- sprzężenie zwrotne 399
- stagnacja w koewolucji 474
- standardowy szum Gaussa 363
- statystyka opisowa 563
- sterowanie parametrami, adaptacyjne 328
 - deterministyczne 328
 - samoadaptujące się 329
- stłaczanie 367
- stochastyczne schładzanie 150
- stopień swobody 562, 572
- strategia 122
 - ewolucyjna 184
 - minimaksowa 452
 - przeszukiwania, lokalna 68
 - otoczenia 68
 - sterująca 204
 - wet za wet 467
 - sukcesja elitarna 288, 289, 515
- suma kwadratów błędów 206, 393
- suwak logarytmiczny 415
- SWARM 533
- SXX 254
- symbol terminalny 203
- symulacja 207
 - ewolucji 467
- symulowane wyżarzanie 82, 145, 146, 150, 186, 292, 324, 338
 - typu Monte Carlo 150
- system hybrydowy 529
 - klasyfikatorów 184
 - koewolucyjny 454, 462
- sztuczna inteligencja rozproszona 538
- sztuczne życie 464, 533
- szukanie po prostych 118
- szum 44, 362
 - Gaussa, standardowy 363
- średnia zmiennej losowej 563
- średnie ważone uporządkowane 506
- środowisko ergodyczne 474
- tablica sympleksu 106
- temperatura 186
- TEMPO Military Planning Game 461
- tendencja centralna 564
- teoria rezonansu, adaptacyjna 410
- testowanie hipotez 572
- Tierra 465
- trasa 2-öptymalna 94
- trening 398
- TSP 36, 61, 84, 126, 152, 155, 178, 225, 278, 356, 406, 435, 478, 535, 579, 582, 583, 584
 - asymetryczny 37
 - symetryczny 37
- turniej losowy 215
- twierdzenie Eulera 50
 - graniczne centralne 565
 - o braku darmowych obiadów 335, 342
- uczenie bez nadzoru 406
 - konkurencyjne 404
 - metodą Hebb'a 410
 - pod nadzorem 406
 - ze wzmacnieniem 459
- uporządkowane średnie ważone 506
- Valuated State Space 499, 500–506
- VEGA 509, 510, 513
- wariancja 564
- wartościowana przestrzeń stanów 499
- wartość oczekiwana 562, 563
 - samoadaptująca się 335
- wartownik 90
- wektor dopuszczalny 105
 - poziomu popytu 498
 - zmiennopozycyjny 64
- wielokryterialne podejmowanie decyzji 499, 506
- wielokryterialny algorytm genetyczny 514

- właściwość wyłaniająca się 464
- wspinanie się 82
 - iteracyjne 146
 - po najbardziej stromym zboczu 70
 - probabilistyczne 150
 - stochastyczne 147
- wstawianie 238
 - najbliższego miasta 226
 - najdalszego miasta 226
- wymiana podtrąs 230
- wzajemna 238
- wynik fałszywie ujemny 573
- wyrażenie symboliczne 203, 212
- wyszukiwanie lokalne 82
 - z nawrotami 85

- zachowanie emergentne 537
- zadanie rozróżnienia wzorca wieloklasowego 509
- zakłócenia losowe 44
- zamiana dwukrawędziowa 93, 163, 210, 226
 - trójkrawędziowa 226
- zasada dziel i rządź 98, 119, 254
 - minimalnej długości opisu 512
 - zblizonej optymalności 283
- zbiór Pareto lokalny optymalny 495
 - optymalny 495
 - rozmyty 416, 417, 429
 - -, dopełnienie 418

- zbiór rozmyty, funkcja przynależności 416
 - -, nośnik 432
 - -, przecięcie 418
 - - pusty 418
 - -, suma 418
- zdarzenia niezależne 556
- zdarzenie 555
- złożenie typu max-min 421
- zmienna boolowska 36
 - losowa 557
 - - Bernoulliego 558, 568
 - - Cauchy'ego 209, 562, 564, 565, 569
 - - chi-kwadrat 562, 563, 565, 566, 567, 569
 - - ciągła 558
 - - dwumianowa 559, 565, 566, 568
 - - dyskretna 558
 - - Gamma 562
 - - Gaussa 182, 198, 208, 561, 563, 565, 567, 570, 571
 - - jednostajna 209, 561, 565, 566
 - - Poissona 209, 211, 560, 563, 565, 566, 567, 569
 - - wykładnicza 562, 563, 565, 566, 567
 - normalna 182, 561
- zmienność całkowita zmiennej 575
 - niewyjaśniona 575
 - wyjaśniona 575
- zrównoleglenie 188

Jan 12, 2009



społeczna edycji