Zbigniew Michalewicz

Algorytmy genetyczne

+

struktury danych

=

programy ewolucyjne

Wydawnictwa Naukowo-Techniczne

Algorytmy genetyczne

+

struktury danych

=

programy ewolucyjne

z angielskiego przełożył
dr hab. inż.
Zbigniew Nahorski
Instytut Badań Systemowych
Polskiej Akademii Nauk

wydanie drugie

Zbigniew Michalewicz

Algorytmy genetyczne

+

struktury danych

=

programy ewolucyjne

Wydawnictwa Naukowo-Techniczne • Warszawa

Dane o oryginale:

Originally published in English under the title

„Genetic Algorithms + Data Structures = Evolution Programs"
by Zbigniew Michalewicz

Copyright © Springer-Verlag Berlin Heidelberg 1992, 1994, 1996
All Rights Reserved

Wydanie polskie jest tłumaczeniem trzeciego, poprawionego
i rozszerzonego wydania angielskiego z 1996 r.

Redaktor *Zuzanna Grzejszczak*
Okładkę i strony tytułowe projektowali *Paulina Brzósko*, *Paweł Pieśniewski*
Przygotowanie do druku *Marianna Szwagrzak*, *Marianna Zadrożna*
Opracowanie techniczne *Anna Szeląg*

Podręcznik akademicki dotowany przez Ministerstwo Edukacji Narodowej

© Copyright for the Polish edition by
Wydawnictwa Naukowo-Techniczne
Warszawa 1996, 1999

All Rights Reserved
Printed in Poland

Utwór w całości ani we fragmentach nie może być powielany
ani rozpowszechniany za pomocą urządzeń elektronicznych,
mechanicznych, kopiujących, nagrywających i innych
bez pisemnej zgody posiadacza praw autorskich.

Adres poczty elektronicznej: wnt@pol.pl
Strona WWW: www.wnt.com.pl

ISBN 83-204-2368-6

Następnemu pokoleniu:
Zbyszkowi, Kasi, Michałowi,
Tomkowi i Irence

Przedmowa do polskiego wydania

W Tyńcu, w gospodzie „Pod Lutym Turem",
należącej do opactwa, siedziało kilku ludzi
słuchając opowiadania wojaka bywalca,
który z dalekich stron przybywszy prawił im o przygodach,
jakich na wojnie i w czasie podróży doznał.

Henryk Sienkiewicz, *Krzyżacy*

Z wielu powodów bardzo jestem rad z ukazania się polskiego wydania tej książki. Przede wszystkim jest to, w pewnym sensie, zwrot zaciągniętego długu wobec Polski. Całe bowiem wykształcenie zdobyłem właśnie tu. Najpierw skończyłem (w 1974 r.) studia na Studium Matematyczno-Technicznym (przy Wydziale Elektroniki) Politechniki Warszawskiej (wydział ten później zmienił nazwę na Studium Podstawowych Problemów Techniki), które było przez wiele lat znakomicie prowadzone przez Profesora Wojciecha Żakowskiego. Po obronie pracy magisterskiej pod kierunkiem Profesora Antoniego Mazurkiewicza (i po rocznej służbie wojskowej w Dęblinie i Świdwinie!) podjąłem swoją pierwszą pracę w Instytucie Podstaw Informatyki Polskiej Akademii Nauk w Warszawie. Miałem szczęście być jednym z bardzo niewielu doktorantów przedwcześnie zmarłego Profesora Witolda Lipskiego; moja praca doktorska (obroniona w 1981 r. w tymże Instytucie) dotyczyła pewnych problemów bezpieczeństwa baz danych. W czerwcu 1982 r. rozpocząłem wykłady w Nowej Zelandii, w Victoria University of Wellington. Był to wspaniały okres w moim życiu, który trwał przez pełnych siedem lat. W lecie 1989 r. nastąpiła kolejna przeprowadzka, tym razem do Karoliny Północnej, gdzie pracuję do dziś (University of North Carolina w Charlotte). Po przybyciu do USA, pod wpływem mojego serdecznego przyjaciela, Andrzeja Jankowskiego, zacząłem się interesować mało stosunkowo wówczas znanymi algorytmami genetycznymi. Już po pierwszym eksperymencie wiedziałem, że pozostanę przy tej tematyce przez wiele lat.

Polska nie jest dla mnie tylko wspomnieniem; nadal utrzymuję żywe stosunki z polskimi naukowcami. Od 1991 r., z inicjatywy Profesorów Mirosława Dąbrowskiego, Zdzisława Pawlaka i Macieja Michalewicza, zaczęto organizować czerwcowe spotkania polskich naukowców (krajowych i zagranicznych), pracujących w dziedzinie sztucznej inteligencji. Spotkania te, choć Mirka nie ma już wśród nas, są kontynuowane do dziś przy dodatkowym poparciu Profe-

sora Piotra Dembińskiego. Dwa lata temu zostałem ponownie zatrudniony w swoim macierzystym instytucie (Instytut Podstaw Informatyki PAN) na stanowisku profesora; przynajmniej raz do roku przyjeżdżam do Warszawy z wykładami na temat algorytmów ewolucyjnych. Tematyka ta powoli zdobywa popularność w Polsce; w czerwcu 1996 r. w Zakopanem odbyły się dwie imprezy z nią związane. Podczas IX Międzynarodowego Sympozjum Systemów Inteligentnych (IX International Symposium on Methodologies for Intelligent Systems; ISMIS) zorganizowałem sesję specjalną poświęconą algorytmom ewolucyjnym; parę dni poźniej, również w Zakopanem, wziąłem udział w pierwszej Krajowej Konferencji Algorytmów Ewolucyjnych. Dlatego myślę, że tłumaczenie tej książki ukazuje się we właściwym czasie i mam nadzieję, że przyczyni się do dalszego wzrostu popularności tej tematyki w Polsce.

Na zakończenie chciałbym serdecznie podziękować wszystkim moim kolegom z Instytutu Podstaw Informatyki Polskiej Akademii Nauk, z Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, a także pracownikom z kilku zakładów Politechniki Warszawskiej, którzy brali żywy udział w moich seminariach w ostatnich kilku latach, oraz rodakom, którzy byli współautorami kilku moich prac: Jarosławowi Arabasowi, Cezaremu Janikowowi, Andrzejowi Jankowskiemu, Jackowi Krawczykowi, Maciejowi Michalewiczowi, Janowi Mulawce, Zbigniewowi Rasiowi, Andrzejowi Szałasowi i Krzysztofowi Trojanowskiemu. Jestem bardzo wdzięczny Panu Profesorowi Leonardowi Bolcowi za to, że w lecie 1991 roku, po jednym z moich wykładów w Warszawie, namówił mnie do napisania tej właśnie książki (pierwsze wydanie ukazało się pod koniec lata 1992 roku). Bardzo dziękuję Zbigniewowi Nahorskiemu za wspaniałe tłumaczenie z angielskiego na polski; było to zadaniem trudnym i zostało wykonane wyśmienicie! Chciałbym także podziękować swojej rodzinie: Żonie Ewie i Synowi Zbyszkowi za to, że byli tak wspaniałymi kompanami w już ponad czternastoletniej tułaczce po świecie, jak również Mamie, Bratu i Siostrze za to, że dzięki nim każdy przyjazd do Polski jest specjalnym wydarzeniem w moim życiu.

Warszawa
czerwiec 1996

Przedmowa do trzeciego wydania

Publiczność zawsze najbardziej
oklaskuje ostatnią pieśń.

Homer, *Odyseja*[1)]

W czasie światowego kongresu na temat inteligencji komputerowej (Orlando, 27 czerwca – 2 lipca 1994 r.) Ken De Jong, jeden z wykładowców plenarnych i obecny redaktor naczelny czasopisma *Evolutionary Computation*, powiedział, że w tej chwili w zdecydowanej większości programów metod ewolucyjnych używa się reprezentacji niebinarnych. Wygląda więc na to, że to podstawowe pojęcie metod ewolucyjnych (lub algorytmów ewolucyjnych, programów ewolucyjnych itp.) jest szeroko zaakceptowane przez większość praktyków z tej dziedziny. W rezultacie w większości zastosowań metod ewolucyjnych przetwarza się populację osobników, przy czym każdy osobnik reprezentuje potencjalne rozwiązanie rozpatrywanego zadania, a proces selekcji jest ukierunkowany: lepsze osobniki mają większe szanse na przeżycie i reprodukcję. Jednocześnie reprezentacja osobników i zbiór operatorów, które zmieniają ich kod genetyczny, są często dobierane do zadania. Wobec tego naprawdę jest mało ważne dalsze wyjaśnianie, że uwzględnienie wiedzy specyficznej dla zadania za pomocą reprezentacji i specjalizowanych operatorów może istotnie poprawić działanie systemu ewolucyjnego. Jednak wiele udanych rozwiązań takich systemów hybrydowych [89]:

> „...wyprowadziło zastosowanie prostych algorytmów genetycznych daleko poza nasze początkowe teorie i rozumienie, tworząc potrzebę ich ponownego przejrzenia i rozszerzenia".

Wydaje mi się, że jest to jeden z najbardziej wyzywających celów dla naukowców zajmujących się dziedziną obliczeń ewolucyjnych. Niektóre ostatnie wyniki potwierdzają osiągnięcia obliczeniowe, podając pewne podstawy teoretyczne (patrz na przykład prace Nicka Radcliffe'a [315], [316], [317] na temat

[1)] Przekład Jana Parandowskiego, wyd. Czytelnik, 1981 (przyp. tłum.).

formalnej analizy i odpowiednich rekombinacji). Jednak jest niezbędne dalsze badanie różnych czynników wpływających na możliwości ewolucyjnych metod rozwiązywania zadań optymalizacyjnych.

Pomimo tej zmiany w pojmowaniu metod ewolucyjnych początkowa organizacja książki pozostała w tym wydaniu niezmieniona. Wprowadzenie na przykład nie zawiera prawie żadnych zmian (z wyjaśnieniem odejścia od algorytmów genetycznych z kodami binarnymi w kierunku bardziej złożonych systemów uzależnionych od zadania). Książka nadal składa się z trzech części, w których omówiłem algorytmy genetyczne (najlepiej znane metody w dziedzinie obliczeń ewolucyjnych), optymalizację numeryczną i różne zastosowania programów ewolucyjnych. Jednak w stosunku do poprzedniego wydania w obecnym wprowadziłem kilka zmian. Prócz drobniejszych zmian, poprawek i modyfikacji w większości rozdziałów (łącznie z dodatkiem A) główne różnice są następujące:

- ze względu na nowe wyniki związane z numeryczną optymalizacją z ograniczeniami napisałem od nowa rozdział 7;
- rozdział 11 istotnie przepracowałem, włączając doń kilka nowych wyników;
- dołączyłem nowy rozdział 13, w którym omawiam oryginalne metody programowania ewolucyjnego i całkiem nowy schemat programowania genetycznego;
- do rozdziału 14 dołączyłem materiał zawarty w zakończeniu drugiego wydania;
- rozdział 15 zawiera ogólny przegląd metod heurystycznych i sposobów uwzględniania ograniczeń w metodach ewolucyjnych;
- zakończenie napisałem od nowa, uwzględniając omówienie obecnych kierunków badań w dziedzinie metod ewolucyjnych; ze względu na te zmiany trzeba było także zmienić odwołania do literatury na początku tego rozdziału;
- dodatki A i C zawierają kilka funkcji testowych (bez i z ograniczeniami), których można użyć do różnych eksperymentów z metodami ewolucyjnymi;
- w dodatku D omówiłem kilka możliwych ćwiczeń; ta część może być przydatna, jeżeli książka jest używana jako podręcznik do wykładu z ćwiczeniami.

Mam szczerą nadzieję, że zmiany te jeszcze zwiększą poczytność tej książki.

Tak jak przy pierwszym i drugim wydaniu, mam przyjemność podziękować za pomoc kilku współautorom, którzy pracowali ze mną w czasie ostatnich dwóch lat. Wiele wyników uzyskanych w trakcie tej współpracy włączyłem do tej książki. Na liście nowych współautorów (nie wymienionych w przedmowach do poprzednich wydań) znajdują się (w kolejności alfabetycznej): Tom Cassen, Michael Cavaretta, Dipankar Dasgupta, Dusan Esquivel, Raul Gallard, Sridhar Isukapali, Rodolphe Le Riche, Li-Tine Li, Hoi-Shan Lin, Rafic Makki, Maciej Michalewicz, Mohammed Moinuddin, Subbu Muddappa, Girish Nazhiyath, Robert Reynolds, Marc Schenauer i Kalpathi Sub-

ramanian. Podziękowanie należy się Sicie Raghavanowi, który ulepszył prosty algorytm genetyczny z kodem rzeczywistym zawarty w dodatku A, i Girishowi Nazhiyathowi, który napisał nową wersję systemu GENOCOP III (opisanego w rozdziale 7), uwzględniającą nieliniowe ograniczenia. Dziękuję wszystkim, którzy poświęcili swój czas na podzielenie się ze mną ich osądem książki. Są oni przede wszystkim odpowiedzialni za większość zmian dokonanych w tym wydaniu. W szczególności wyrażam moją wdzięczność Thomasowi Bäckowi i Davidowi Fogelowi, z którymi pracowałem nad książką pt. *Handbook of Evolutionary Computation* [17] i redaktorowi odpowiedzialnemu z Wydawnictwa Springer-Verlag, Hansowi Wossnerowi, za pomoc w czasie pracy nad książką, Chciałbym też z wdzięcznością wspomnieć o grancie (IRI-9322400) przyznanym przez National Science Foundation, który pomógł mi przygotować to wydanie. Ze względu na ten grant byłem w stanie wprowadzić wiele zmian (poprawa rozdziału 7, nowy rozdział 15). Bardzo doceniam pomoc Larry'ego Reekera, dyrektora programowego National Science Foundation. Chciałbym także podziękować moim doktorantom z UNC-Charlotte, Universidad Nacional de San Luis i Linköping University, którzy brali udział w moich wykładach w roku 1994/95. Jak zwykle, lubiłem te wykłady i okazywały się one bardzo pouczające.

Charlotte
październik 1995

Zbigniew Michalewicz

Przedmowa do drugiego wydania

Ponieważ dobór naturalny działa tylko
dla dobra każdej istoty żywej, wszelkie
dalsze cielesne i duchowe przymioty
dążyć będą do doskonałości.

Charles Darwin, *O powstawaniu gatunków*[1)]

Dziedzina obliczeń ewolucyjnych osiągnęła stan dojrzałości. Regularnie odbywa się kilka uznanych międzynarodowych konferencji, w których udział biorą setki uczestników (International Conferences on Genetic Algorithms – ICGA [167], [171], [344], [32], [129], Parallel Problem Solving from Nature – PPSN [351], [251], Annual Conferences on Evolutionary Programming – EP [123], [124], [378]). Organizuje się nowe coroczne konferencje na ten temat (IEEE International Conferences on Evolutionary Computation [275], [276]). Prócz tego odbywają się dziesiątki seminariów, specjalnych sesji na konferencjach i lokalnych konferencji każdego roku na całym świecie. Nowe czasopismo *Evolutionary Computation* wydawane przez MIT Press [87] jest poświęcone wyłącznie metodom obliczeń ewolucyjnych. Wiele innych czasopism wydało numery specjalne poświęcone obliczeniom ewolucyjnym (np. [118], [263]). Wspaniałe artykuły przeglądowe [28], [29], [320], [397], [119] i opracowania zamykają tę bardziej lub mniej kompletną bibliografię dziedziny [161], [336], [297]. Jest także *The Hitch-Hiker's Guide to Evolutionary Computation* przygotowany przez Jörga Heitköttera [177] z Uniwersytetu w Dortmundzie, dostępny w sieci Internet w grupie zainteresowań comp.ai.genetic.

Ten trend przyśpieszył przygotowanie przeze mnie drugiego, rozszerzonego wydania tej książki. Tak jak to było w pierwszym wydaniu, książka składa się głównie z artykułów, które opublikowałem w ciągu ostatnich kilku lat. Z tego powodu prezentuje ona raczej osobiste spojrzenie na dziedzinę obliczeń ewolucyjnych niż zbalansowany przegląd dokonań w tej dziedzinie. W konsekwencji nie jest to typowy podręcznik, jednak w wielu wyższych szkołach używano pierwszego wydania jako podstawy wykładu o „obliczeniach ewolucyjnych". Aby pomóc potencjalnym przyszłym studentom, włączyłem do książki

[1)] Dzieła wybrane, t. 2. Przekład Szymona Dicksteina i Józefa Nusbauma-Hilarowicza, Państwowe Wydawnictwo Rolnicze i Leśne, Warszawa 1959 (przyp. tłum.).

kilka dodatkowych elementów (dodatek z prostym programem genetycznym, krótkie odnośniki do innych osiągnięć w tej dziedzinie, skorowidz itp.). Jednak nie dodałem zadań na końcu rozdziałów. Powodem jest to, że dziedzina obliczeń ewolucyjnych jest nadal bardzo młoda i jest w niej wiele obszarów wymagających dalszych badań. Powinny być one łatwo zauważalne w tekście, gdzie stawia się więcej pytań niż podaje odpowiedzi. Celem autora tej książki jest przedstawienie dziedziny obliczeń ewolucyjnych w łatwy sposób oraz omówienie jej prostoty i elegancji na wielu ciekawych przykładach. Pisanie programu ewolucyjnego dla danego zagadnienia powinno być przyjemnym przedsięwzięciem. Książka może służyć jako przewodnik pomocny w osiągnięciu tego zadania.

Z okazji obecnego wydania poprawiłem wiele błędów drukarskich z pierwszego wydania. Organizację książki pozostawiłem jednak bez zmian. Jest nadal 12 rozdziałów, wprowadzenie i zakończenie, ale niektóre rozdziały zostały znacznie przeze mnie powiększone. Trzy nowe podpunkty dodałem do rozdziału 4 (o algorytmach genetycznych z odwzorowaniami zwężającymi, o algorytmach genetycznych ze zmiennym rozmiarem populacji i o metodzie rozwiązywania zadania załadunku z ograniczeniami). Kilka informacji o kodzie Graya włączyłem do rozdziału 5. Rozdział 7 rozszerzyłem o punkt zawierający opis systemu GENOCOP. Przedstawia on wstępne doświadczenia i wyniki, dyskusję dalszych modyfikacji systemu i opis pierwszych doświadczeń z zadaniami optymalizacji z nieliniowymi ograniczeniami (GENOCOP II). Dodałem krótki punkt o optymalizacji wielomodalnej i wielokryterialnej do rozdziału 8. W rozdziale 11 doszedł punkt o planowaniu drogi w środowisku ruchomego robota. Zakończenie rozszerzyłem o wyniki ostanich doświadczeń potwierdzających hipotezę, że specyficzna dla rozpatrywanego zagadnienia wiedza poprawia działanie algorytmu wyrażone przez czas obliczeń i uzyskaną dokładność, ale jednocześnie zawęża możliwość jego stosowania. Czytelnik znajdzie w nim także kilka informacji o algorytmach kulturowych. Wydanie drugie ma skorowidz, którego nie było w pierwszym wydaniu, i dodatek, w którym znajduje się program w języku C dla prostego algorytmu genetycznego, przeznaczonego dla początkujących Czytelników. Jasność góruje w nim nad efektywnością. W tekście występują również inne drobniejsze zmiany. Kilka podpunktów usunąłem, dodałem lub zmodyfikowałem. W wydaniu tym jest też ponad sto nowych odwołań do literatury.

Tak jak w pierwszym wydaniu, chciałbym z przyjemnością podziękować moim współautorom, którzy pracowali ze mną w ciągu ostatnich dwóch lat. Dużo wyników uzyskanych w trakcie tej współpracy włączyłem do książki. Na liście tych współautorów znajdują się: Jarosław Arabas, Naguib Attia, Hoi-Shan Lin, Thomas Logan, Jan Mulawka, Swarnalatha Swaminathan, Andrzej Szałas i Jing Xiao. Podziękowania należą się także Denisowi Cormierowi, który napisał algorytm genetyczny umieszczony w dodatku. Chciałbym także podziękować wszystkim osobom, które podzieliły się ze mną swoimi uwagami o książce. To oni przede wszystkim odpowiadają za większość zmian wprowa-

dzonych w tym wydaniu. Chciałbym także podziękować wszystkim moim doktorantom z UNC-Charlotte i North Carolina State University, którzy wzięli udział w wykładach telewizyjnych na jesieni 1992 r. (i w konsekwencji musieli używać pierwszego wydania tej książki jako podręcznika). Było to dla mnie bardzo miłe i pożyteczne doświadczenie.

Charlotte
marzec 1994

Zbigniew Michalewicz

Przedmowa do pierwszego wydania

– Czego uczy wasz Mistrz? –
zapytał jeden z odwiedzających.
– Niczego – odpowiedział uczeń.
– Dlaczego więc przemawia?
– On jedynie wskazuje drogę –
nie naucza niczego.

Anthony de Mello, *Minuta mądrości*[1)]

W ciągu ostatnich trzech dziesięcioleci narastało zainteresowanie algorytmami, które naśladowały procesy natury. Masowe pojawienie się komputerów równoległych spowodowało, że algorytmy te nabrały praktycznego znaczenia. Najlepiej znane algorytmy z tej klasy to programowanie ewolucyjne, algorytmy genetyczne, strategie ewolucyjne, symulowane wyżarzanie, systemy klasyfikujące i sieci neuronowe. Ostatnio (1-3 października 1990 r.) w Uniwersytecie w Dortmundzie w Niemczech odbyło się pierwsze seminarium na temat algorytmów równoległych rozwiązywania zagadnień z natury [351].

W książce dyskutuje się pewną podklasę tych algorytmów – tę, która jest oparta na zasadzie ewolucji (przeżywania najbardziej dopasowanych osobników). W takich algorytmach populacja osobników (potencjalne rozwiązanie) podlega sekwencji transformacji jednoargumentowych (typu mutacji) i wieloargumentowych (typu krzyżowania). Te osobniki walczą o przetrwanie – w schemacie selekcji, ukierunkowanym na bardziej dopasowanych, wybiera się następne pokolenie. Po kilku pokoleniach program zbiega się. Najlepsze osobniki reprezentują optymalne rozwiązanie.

Jest wiele różnych algorytmów w tej kategorii. Aby podkreślić podobieństwo między nimi, używamy wspólnego terminu „programy ewolucyjne".

Programy ewolucyjne można rozumieć jako uogólnienie algorytmów genetycznych. Klasyczne algorytmy genetyczne operują na ciągach binarnych o stałej długości, co nie musi zachodzić w przypadku programów ewolucyjnych. Programy ewolucyjne zawierają także zróżnicowane operacje „genetyczne", podczas gdy klasyczne algorytmy genetyczne używają tylko binarnego krzyżowania i mutacji.

Początku algorytmów genetycznych należy szukać we wczesnych latach pięćdziesiątych, kiedy kilku biologów użyło komputerów do symulacji syste-

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

mów biologicznych [154]. Jednakże dopiero prace wykonane w późnych latach sześćdziesiątych i wczesnych siedemdziesiątych w Uniwersytecie w Michigan pod kierownictwem Johna Hollanda doprowadziły do takiego pojęcia algorytmów genetycznych, jak się je rozumie współcześnie. Zainteresowanie algorytmami genetycznymi rośnie gwałtownie. Ostatnia czwarta międzynarodowa konferencja na temat algorytmów genetycznych (San Diego, 13-16 lipca 1991 r.) zebrała około 300 uczestników.

W książce przedstawiłem wyniki trzyletnich badań, począwszy od początku 1989 r. w Uniwersytecie Wiktorii w Wellington w Nowej Zelandii aż do końca 1991 r. (od lipca 1989 r. byłem w Uniwersytecie Karoliny Północnej w Charlotte). W tym czasie opracowywałem i eksperymentowałem z różnymi modyfikacjami algorytmów genetycznych, używając rozmaitych struktur danych do reprezentacji chromosomów i różnych wykonywanych na nich operacji „genetycznych". Ze względu na moje wcześniejsze prace z bazami danych [256], [260], gdzie ograniczenia grają istotną rolę, większość programów ewolucyjnych opracowałem dla zagadnień z ograniczeniami.

Idea programów ewolucyjnych (w sensie prezentowanym w tej książce) powstała dość wcześnie [204], [277] i była później poparta seriami doświadczeń. Mimo że programy ewolucyjne w większości nie mają silnego podparcia teoretycznego, wyniki doświadczeń były więcej niż zachęcające. Bardzo często działały one dużo lepiej od klasycznych algorytmów genetycznych, od systemów będących w sprzedaży, czy też innych, najlepiej znanych algorytmów dla poszczególnych klas zadań.

Inni badacze, na różnych etapach swoich poszukiwań, prowadzili badania, które były doskonałymi przykładami metod „programowania ewolucyjnego". Niektóre z nich są dyskutowane w tej książce. W rozdziale 8 przedstawiłem przegląd strategii ewolucyjnych, metody opracowanej w Niemczech przez I. Rechenberga i H. -P. Schwefela [319], [348] dla zadań optymalizacji parametrycznej. Wiele osób badało właściwości systemów ewolucyjnych dla zadań szeregowania, włączając w to szeroko znane „zadanie komiwojażera" (rozdział 10).

W rozdziale 11 omówiłem systemy dla szerokiej gamy zadań łącznie z zadaniami na grafach, harmonogramowania i przydziału. W rozdziale 12 opisałem konstrukcję programu ewolucyjnego do uczenia indukcyjnego w przestrzeni atrybutów, rozwiniętą przez C. Janikowa [200]. W zakończeniu krótko przedyskutowałem programy ewolucyjne do generowania programów w LISP-ie, ukierunkowane na rozwiązywanie konkretnych zadań, rozwinięte przez J. Kozę [228] oraz zaprezentowałem pomysły nowego środowiska programowego.

Książka jest zorganizowana w następujący sposób. We wprowadzeniu znajduje się ogólne przedstawienie zagadnienia oraz prezentacja podstawowego celu książki. Ponieważ programy ewolucyjne działają na tej samej zasadzie co algorytmy genetyczne, część I tej książki jest przeglądem tej tematyki. Wyjaśniłem w niej, co to są algorytmy genetyczne, jak one działają i dlaczego (rozdziały 1-3). W ostatnim rozdziale części I (rodział 4) przedstawiłem kilka wybranych zagadnień (programy wyboru, skalowanie itp.) z zakresu algorytmów genetycznych.

W części II badałem pojedyncze struktury danych: reprezentację z wektorami zmiennopozycyjnymi, dopiero ostatnio szerzej zaakceptowaną przez środowisko związane z algorytmami genetycznymi [78]. Rozważona jest tylko optymalizacja numeryczna. Przedstawiłem porównanie eksperymentalne reprezentacji binarnych i zmiennopozycyjnych (rozdział 5) i omówiłem nowe operatory „genetyczne" odpowiedzialne za dokładne dostrojenie lokalne (rozdział 6). W rozdziale 7 podałem dwa programy ewolucyjne dla zadań z ograniczeniami: system GENOCOP do optymalizacji funkcji z liniowymi ograniczeniami i system GAFOC do zadań sterowania optymalnego. Rozpatrzone są różne przykłady testowe. Wyniki programów ewolucyjnych porównałem także z otrzymanymi z systemów komercyjnych. Ostatni rozdział tej części (rozdział 8) poświęciłem przeglądowi strategii ewolucyjnych oraz opisowi innych metod.

W części III przedstawiłem programy ewolucyjne zbudowane w ciągu ostatnich lat w celu sprawdzenia możliwości ich zastosowania w różnych skomplikowanych zagadnieniach. Zaprezentowane są dalsze eksperymenty z porządkującymi programami ewolucyjnymi, programami ewolucyjnymi z macierzami i grafami w charakterze struktur chromosomów. Omówiłem także zastosowania programu ewolucyjnego do uczenia maszynowego, porównując go z innymi podejściami.

Tytuł tej książki naśladuje słynne wyrażenie użyte przez N. Wirtha 15 lat temu w tytule jego książki *Algorytmy + struktury danych = programy* [406]. Obu książkom przyświeca ta sama idea. Aby zbudować udany program (w szczególności program ewolucyjny), należy użyć odpowiednich struktur danych (struktury danych w przypadku programów ewolucyjnych odpowiadają reprezentacji chromosomów) oraz odpowiednich algorytmów (odpowiadają one operatorom „genetycznym" do transformacji jednego lub więcej pojedynczych chromosomów).

Książka jest przeznaczona dla szerokiego kręgu czytelników: doktorantów, programistów, badaczy, inżynierów, projektantów, każdego, kto ma do czynienia z trudnymi zadaniami optymalizacji. W szczególności książka powinna zainteresować środowisko zajmujące się badaniami operacyjnymi, gdyż wiele rozważanych w niej zadań (zadanie komiwojażera, szeregowanie, zadania transportowe) są właśnie z ich dziedziny zainteresowań. Do śledzenia przedstawionego tu materiału wystarczy znajomość matematyki na poziomie szkoły średniej i podstawowych pojęć programowania.

Podziękowania

Mam przyjemność podziękować kilku osobom i instytucjom za pomoc w wysiłku przygotowania tej książki.

Podziękowania należą się wielu współautorom, którzy pracowali ze mną na różnych etapach tego zamierzenia. Są to (w kolejności alfabetycznej):

Paul Elia, Lindsay Groves, Mathew Hobbs, Cezary Janikow, Andrzej Jankowski, Mohammad Kazemi, Jacek Krawczyk, Zbigniew Raś, Joseph Schell, David Seniv, Don Shoff, Jim Stevens, Tony Vignaux i George Windholz.

Chciałbym także podziękować moim doktorantom, którzy wzięli udział w kursie Algorytmy genetyczne I, który prowadziłem w UNC-Charlotte na jesieni 1990 r., oraz moim studentom, którzy napisali prace magisterskie na temat różnych modyfikacji algorytmów genetycznych. Byli to: Jason Foodman, Jay Livingstone, Jeffrey B. Rayfield, David Seniw, Jim Stevens, Charles Strickland, Swarnalatha Swaminathan i Keith Wharton. Jim Stevens dostarczył mi opowiadanie o królikach i lisach (rozdział 1).

Na podziękowania zasłużyły również następujące osoby: Abdollah Homaifar z Uniwersytetu Stanu Karoliny Północnej, Kenneth Massa z Uniwersytetu Loyola (Nowy Orlean) i kilku kolegów z UNC-Charlotte: Mike Allen, Rick Lejk, Zbigniew Raś, Harold Reiter, Joe Schell i Barry Wilkinson, którzy przejrzeli niektóre części tej książki.

Chciałbym podziękować za pomoc w odzyskaniu wszystkich rozdziałów tej książki (skasowanych przez przypadek) Dougowi Gulletowi, Jerry'emu Holtowi i Dwayne'owi McNeilowi (Computing Services, UNC-Charlotte) oraz Janowi Thomasowi za ogólną pomoc.

Chciałbym także rozszerzyć moje podziękowania na redaktorów serii „Artificial Intelligence" w wydawnictwie Springer-Verlag: Leonarda Bolca, Donalda Lovelanda i Hansa Wössnera za zainicjowanie pomysłu tej książki i ich pomoc w trakcie jej przygotowania. Specjalnego podziękowania wymaga J. Andrew Ross, redaktor języka angielskiego w Springer-Verlag, za istotną pomoc w dopracowaniu stylistycznym książki.

Chciałbym podziękować za serię grantów z Centrum Superobliczeń Karoliny Północnej (1990-1991), co pozwoliło mi wykonać setki doświadczeń opisanych w tekście.

W książce umieściłem wiele cytatów z innych opublikowanych prac, jak również z moich opublikowanych artykułów. W związku z tym chciałbym podziękować za wszystkie zezwolenia na użycie tych materiałów w tej publikacji otrzymane od następujących wydawnictw: Pitman Publishing Company, Birkhauser Verlag AG, John Wiley and Sons Ltd., Complex Systems Publications Inc., Addison-Wesley Publishing Company, Kluwer Academic Publishers, Chapman and Hall Ltd., Scientific, Technical and Medical Publishers, Prentice Hall, IEEE, Association for Computing Machinery, Morgan Kaufmann Publishers Inc. i indywidualnych osób: Davida Goldberga, Freda Glovera, Johna Grefenstette'a, Abdollaha Homaifara i Johna Kozy.

Na koniec chcę podziękować mojej rodzinie za jej cierpliwość i pomoc podczas (długiego) lata 1991 r.

Charlotte
styczeń 1992

Zbigniew Michalewicz

Spis treści

Wprowadzenie 25

Część I. Algorytmy genetyczne 37

1. Algorytmy genetyczne: co to jest? 39
   1.1. Optymalizacja prostej funkcji 44
      1.1.1. Reprezentacja 45
      1.1.2. Populacja początkowa 46
      1.1.3. Funkcja oceny 46
      1.1.4. Operatory genetyczne 47
      1.1.5. Parametry 48
      1.1.6. Wyniki obliczeń 48
   1.2. Dylemat więźnia 49
      1.2.1. Reprezentacja strategii 49
      1.2.2. Schemat algorytmu genetycznego 50
      1.2.3. Wyniki obliczeń 50
   1.3. Zadanie komiwojażera 51
   1.4. Algorytmy wzrostu, symulowane wyżarzanie a algorytmy genetyczne 53
   1.5. Wnioski 57

2. Algorytmy genetyczne: jak one działają? 58

3. Algorytmy genetyczne: dlaczego one działają? 71

4. Algorytmy genetyczne: wybrane zagadnienia 84
   4.1. Mechanizm próbkowania 85
   4.2. Charakterystyki funkcji 93
   4.3. Algorytmy genetyczne z odwzorowaniem zwężającym 95

4.4. Algorytmy genetyczne ze zmienną liczebnością populacji 100
4.5. Algorytmy genetyczne, ograniczenia i zadanie załadunku 109
4.5.1. Zero-jedynkowe zadanie załadunku i dane testowe 110
4.5.2. Opis algorytmów 111
4.5.3. Obliczenia i wyniki 114
4.6. Inne pomysły 118

Część II. Optymalizacja numeryczna 125

5. Binarnie czy zmiennopozycyjnie? 127
5.1. Przykład testowy 130
5.2. Dwie wersje 130
5.2.1. Wersja binarna 131
5.2.2. Wersja zmiennopozycyjna 131
5.3. Obliczenia 131
5.3.1. Losowa mutacja i krzyżowanie 132
5.3.2. Mutacja nierównomierna 133
5.3.3. Inne operatory 135
5.4. Efektywność czasowa obliczeń 136
5.5. Wnioski 136

6. Dokładne dostrajanie lokalne 138
6.1. Przykłady testowe 139
6.1.1. Zadanie liniowo-kwadratowe 140
6.1.2. Zadanie zbierania plonów 141
6.1.3. Zadanie pchania wózka 141
6.2. Program ewolucyjny optymalizacji numerycznej 142
6.2.1. Reprezentacja 142
6.2.2. Operatory specjalizowane 142
6.3. Obliczenia i wyniki 144
6.4. Programy ewolucyjne a inne metody 146
6.4.1. Zadanie liniowo-kwadratowe 146
6.4.2. Zadanie zbierania plonów 147
6.4.3. Zadanie pchania wózka 147
6.4.4. Znaczenie mutacji nierównomiernej 149
6.5. Wnioski 150

7. Zadania z ograniczeniami 152
7.1. Program ewolucyjny: system GENOCOP 153
7.1.1. Przykład 157
7.1.2. Operatory 158
7.1.3. Testowanie systemu GENOCOP 162
7.2. Optymalizacja nieliniowa: GENOCOP II 167
7.3. Inne metody 174

7.3.1. Pięć zadań testujących 178
7.3.2. Eksperymenty 181
7.4. Inne możliwości 184
7.5. GENOCOP III 188

8. Strategie ewolucyjne i inne metody 192
8.1. Rozwój strategii ewolucyjnych 193
8.2. Porównanie strategii ewolucyjnych i operatorów genetycznych 197
8.3. Optymalizacja funkcji wielomodalnych i wielokryterialnych 202
8.3.1. Optymalizacja wielomodalna 202
8.3.2. Optymalizacja wielokryterialna 205
8.4. Inne programy ewolucyjne 207

Część III. Programy ewolucyjne 213

9. Zadanie transportowe 215
9.1. Liniowe zadanie transportowe 215
9.1.1. Klasyczne algorytmy genetyczne 218
9.1.2. Uwzględnianie wiedzy specyficznej dla zadania 220
9.1.3. Macierz jako struktura reprezentacji 223
9.1.4. Wnioski 231
9.2. Nieliniowe zadanie transportowe 231
9.2.1. Reprezentacja 232
9.2.2. Inicjalizacja 232
9.2.3. Ocena 232
9.2.4. Operatory 232
9.2.5. Parametry 233
9.2.6. Zadania testujące 234
9.2.7. Obliczenia i wyniki 237
9.2.8. Wnioski 242

10. Zadanie komiwojażera 245

11. Programy ewolucyjne dla różnych zadań dyskretnych 274
11.1. Harmonogramowanie 274
11.2. Układanie planu lekcji 281
11.3. Podział obiektów i grafów 283
11.4. Planowanie drogi w środowisku ruchomego robota . . 289
11.5. Uwagi 298

12. Uczenie maszynowe 304
12.1. Podejście Michigan 307
12.2. Podejście Pitt 312

12.3. Program ewolucyjny: system GIL 313
12.3.1. Struktura danych 314
12.3.2. Operatory genetyczne 315
12.4. Porównanie 318
12.5. REGAL 319

13. Programowanie ewolucyjne a programowanie genetyczne 321
13.1. Programowanie ewolucyjne 321
13.2. Programowanie genetyczne 323

14. Hierarchia programów ewolucyjnych 326

15. Programy ewolucyjne i heurystyki 345
15.1. Metody i heurystyki: podsumowanie 348
15.2. Rozwiązania dopuszczalne i niedopuszczalne 350
15.3. Heurystyki do oceny osobników 353

16. Zakończenie 367

Dodatek A 376

Dodatek B 387

Dodatek C 391

Dodatek D 497

Literatura 401

Słownik angielsko-polski 417

Słownik polsko-angielski 421

Skorowidz nazwisk 425

Skorowidz rzeczowy 427

Wprowadzenie

Widziałem również pod słońcem,
że to nie szybcy wygrywają bieg, ani mężni wojnę,
że to nie mędrcom przypada chleb,
ani rozumnym bogactwo,
ani też uznanie tym, co posiadają wiedzę,
lecz czas i przypadek ustalają los wszystkich.

Pismo Święte Starego i Nowego Testamentu, Księga Koheleta, 9[1)]

W ciągu ostatnich 30 lat narastało zainteresowanie systemami, w których do rozwiązywania zadań stosuje się zasady ewolucji i dziedziczności. W systemach tych występuje populacja potencjalnych rozwiązań, zawierają one pewien proces selekcji, oparty na dopasowaniu osobników, i pewne operatory „genetyczne". Jednym z typów takich systemów jest klasa strategii ewolucyjnych, to znaczy algorytmów, które naśladują zasady ewolucji w naturze przy rozwiązywaniu zadań optymalizacji parametrycznej [319], [348] (Rechenberg, Schwefel). Programowanie ewolucyjne Fogla [126] jest metodą przeszukiwania przestrzeni stanów małych automatów o skończonej pamięci. Metody przeszukiwania rozproszonego Glovera [142] utrzymują populację punktów odniesienia i generują potomstwo za pomocą ich liniowych kombinacji. Innym typem systemów używających zasad ewolucji są algorytmy genetyczne Hollanda [188]. W 1990 r. Koza [231] zaproponował system oparty na zasadach ewolucji, zwany *programowaniem genetycznym*, w którym poszukuje się programu komputerowego najlepiej dopasowanego do rozwiązania danego zadania.

Będziemy tu używać wspólnego terminu *programy ewolucyjne* dla wszystkich systemów korzystających z zasady ewolucji, łącznie z wymienionymi wcześniej. Struktura programu ewolucyjnego jest przedstawiona na rys. 0.1.

Program ewolucyjny jest algorytmem probabilistycznym, w którym generuje się populację osobników $P(t) = \{x_1^t, ..., x_n^t\}$ w każdej iteracji t. Każdy osobnik przedstawia możliwe rozwiązanie rozpatrywanego zadania i w programie ewolucyjnym jest reprezentowany przez (być może skomplikowaną) strukturę danych S. Każde rozwiązanie x_i^t ocenia się na podstawie pewnej miary jego „dopasowania". Tak więc nową populację (w iteracji $t+1$) tworzy się przez selekcję osobników najlepiej dopasowanych (faza selekcji). Pewne

[1)] Przekład zespołowy pod redakcją Michała Petera, Księgarnia Św. Wojciecha, 1992 (przyp. tłum.).

osobniki nowej populacji podlegają dodatkowo transformacji (faza zmiany) za pomocą operatorów „genetycznych", dając w ten sposób nowe rozwiązanie. Mogą to być transformacje jednoargumentowe m_i (typu mutacji), w których nowe osobniki powstają przez małą zmianę pojedynczego osobnika ($m_i : S \to S$), i transformacje wieloargumentowe c_j (typu krzyżowania), w których nowe osobniki powstają przez łączenie części z kilku (dwóch lub więcej) osobników ($c_j : S \times ... \times S \to S$). Po kilku krokach generacji program zbiega się i spodziewamy się, że najlepsze osobniki reprezentują rozwiązanie leżące blisko optymalnego (rozwiązanie rozsądne).

```
procedure program ewolucyjny
begin
   t ← 0
   ustal początkowe P(t)
   oceń P(t)
   while (not warunek zakończenia) do
   begin
      t ← t + 1
      wybierz P(t) z P(t − 1)
      zmień P(t)
      oceń P(t)
   end
end
```

Rys. 0.1. Struktura programu ewolucyjnego

Rozważmy pewien ogólny przykład. Przypuśćmy, że szukamy grafu, który powinien spełniać pewne wymagania (powiedzmy, że szukamy optymalnych połączeń w sieci telekomunikacyjnej zgodnie z pewnymi wymaganiami, takimi jak koszt wysłania wiadomości, niezawodność itp.). Każdy osobnik populacji występującej w programie ewolucyjnym przedstawia możliwe rozwiązanie zadania, to znaczy graf. Początkowa populacja grafów $P(0)$ (generowana losowo lub za pomocą pewnej metody heurystycznej) jest punktem startowym ($t = 0$) programu ewolucyjnego. Funkcja oceniająca jest na ogół znana – uwzględnia ona wymagania zadania. Funkcja oceniająca określa dopasowanie rozwiązania, rozróżniając osobniki lepsze i gorsze. Można utworzyć kilka operatorów mutacji, które przekształcają graf. Można także rozważyć kilka operatorów krzyżowania, które składają nowy graf ze struktur dwóch lub więcej grafów. Bardzo często operatory takie uwzględniają wiedzę specyficzną dla danego problemu. Na przykład, jeżeli graf, którego poszukujemy, jest spójny i niecykliczny, to możliwe mutacje mogą polegać na usunięciu łuku grafu i dodaniu nowego, który połączy powstałe wcześniej dwa rozłączne podgrafy. Inna możliwość to określenie mutacji niezależnie od zadania i włączenie do funkcji oceniającej składnika karzącego za grafy, które nie są drzewami.

Rzecz jasna, dla danego zadania można stworzyć wiele programów ewolucyjnych. Takie programy mogą się różnić wieloma szczegółami. Mogą mieć inne struktury danych reprezentujące poszczególne osobniki, inne operatory „genetyczne" do transformacji osobników, inne metody tworzenia populacji początkowej, inne sposoby uwzględniania ograniczeń występujących w zadaniu, inne parametry (rozmiar populacji, prawdopodobieństwa użycia poszczególnych operatorów itp.). Jednak działają one według tej samej zasady: populacja osobników podlega transformacji i w trakcie procesu ewolucyjnego osobniki starają się przetrwać.

Pomysł programowania ewolucyjnego nie jest nowy i przewijał się w literaturze przez ostanich 30 lat [126], [188], [348]. Od tego czasu powstało wiele różnych systemów ewolucyjnych. W tym tekście omawiamy te różne wzorce programów ewolucyjnych pod względem ich podobieństw. Zresztą główne różnice między nimi są ukryte na niższym poziomie. Nie będziemy omawiać filozoficznych różnic między różnymi metodami ewolucyjnymi (na przykład czy działają one na poziomie genotypów, czy fenotypów), ale raczej omówimy je pod względem budowy programu ewolucyjnego dla poszczególnych klas zadań. Polecamy użycie właściwych (być może skomplikowanych) struktur danych (do reprezentacji chromosomów) łącznie z rozszerzonym zbiorem operatorów genetycznych, gdy tymczasem na przykład klasyczne algorytmy genetyczne używają dla swoich osobników łańcuchów binarnych o skończonej długości (jako chromosomu lub struktury danych S) i dwóch operatorów: binarnej mutacji i binarnego krzyżowania. Inaczej mówiąc, struktura algorytmu genetycznego jest taka sama jak programu ewolucyjnego (rys. 0.1), a różnice są ukryte na niższym poziomie. W programach ewolucyjnych chromosomy nie muszą być reprezentowane przez łańcuchy binarne, a proces zmian dopuszcza inne operatory „genetyczne" odpowiednie dla zadanej struktury i zadania.

Nie jest to całkiem nowy kierunek. Jeszcze w 1985 r. De Jong [84] napisał:

> Co zrobić, kiedy elementy przeszukiwanej przestrzeni są w sposób naturalny reprezentowane przez bardziej skomplikowane struktury danych, takie jak macierze, drzewa, dwugrafy itp.? Czy należy starać się je „linearyzować" do łańcuchów, czy też istnieją sposoby twórczego przedefiniowania krzyżowania i mutacji, tak aby można było bezpośrednio pracować na takich strukturach. Nie znam żadnych rozwiązań idących w tym kierunku.

Jak wspomniano wcześniej, algorytmy genetyczne używają łańcuchów binarnych o stałej długości i tylko dwóch podstawowych operatorów genetycznych. Dwie znaczniejsze (wcześniejsze) publikacje o algorytmach genetycznych [188], [82] przedstawiają teorię i oprogramowanie takich algorytmów. Jak napisano w [155]:

> Wkład tej pracy [82] polegał na konsekwentnej abstrakcji i upraszczaniu. De Jong doszedł do czegoś nie przez unikanie uproszczeń, lecz ze względu na

ich stosowanie. [...] Książka Hollanda [188] położyła podstawy teoretyczne dla prac De Jonga i innych przez matematyczną identyfikację złożonej roli, jaką odgrywają w przeszukiwaniu genetycznym podzbiory podobieństw (schematy), rozerwania za pomocą operatorów minimalnych i selekcje reprodukcyjne. [...] Kolejni badacze mieli tendencje do rozumienia teoretycznych sugestii z [188] dosłownie, w ten sposób zwiększając sukces oprogramowania De Jonga, ze zręcznymi rozwiązaniami i operatorami.

Jednak w następnym punkcie Goldberg [155] pisze:

Jest interesujące, a może nawet śmieszne, że żaden z nich nie myślał, że jego praca będzie brana tak dosłownie. Chociaż oprogramowanie De Jonga ustaliło użyteczne metody zgodne z uproszczeniami teoretycznymi Hollanda, kolejni badacze starali się traktować oba osiągnięcia jako nienaruszalną świętość.

Wygląda na to, że „naturalna" reprezentacja potencjalnych rozwiązań danego zagadnienia i rodzina możliwych do zastosowania operatorów „genetycznych" może być bardzo użyteczna w przybliżaniu rozwiązań wielu zadań, a podejście z modelowaniem naśladującym naturę (programowaniem ewolucyjnym) jest obiecującym ogólnym kierunkiem rozwiązywania zagadnień. Niektórzy badacze próbowali już wcześniej używać innych reprezentacji, jak na przykład list uporządkowanych (do zagadnień pakowania), list zagnieżdżonych (do zadań szeregowania prac) czy list o zmiennej długości (do projektowania schematów półprzewodników). Podczas ostatnich 10 lat przedstawiano różne, związane z zastosowaniami modyfikacje algorytmów genetycznych [73], [167], [171], [173], [278], [363], [364], [392]. Te modyfikacje obejmowały łańcuchy o zmiennej długości (włączając w to łańcuchy, których elementy były ustalane za pomocą wyrażeń *if-then-else* [363]), struktury bogatsze niż łańcuchy binarne (na przykład macierze [392]) oraz eksperymenty ze zmodyfikowanymi operatorami genetycznymi, aby sprostać wymaganiom poszczególnych zastosowań [270]. W [285] opisano algorytm genetyczny z propagacją wstecz (metoda nauczania w sieciach neuronowych), jako operatorem, oraz z mutacją i krzyżowaniem dostosowanym do dziedziny sieci neuronowych. Davis i Coombs [65], [76] opisali algorytm genetyczny, który wykonywał jeden krok w procesie projektowania sieci komunikacyjnej z pakietem przełączającym. Użyta przez nich reprezentacja nie była binarna, a oprócz tego wprowadzili pięć operatorów „genetycznych" (opartych na wiedzy, statystycznych, numerycznych). Operatory te znacznie się różniły od binarnej mutacji i krzyżowania. Inni badacze, przy rozważaniu rozwiązywania zadania szeregowania prac [21], napisali:

Aby poprawić działanie algorytmu i powiększyć przestrzeń rozwiązań wymyślono reprezentację chromosomu, w której jest pamiętana informacja istotna dla zadania. Wprowadzono także związane z zadaniem operatory połączeń, w których uwzględnia się dodatkową informację.

Można przytoczyć wiele podobnych cytatów. Wygląda na to, że dla zapisu rozwiązywanego zadania większość badaczy wprowadzała „modyfikacje" w swoich rozwiązaniach algorytmów genetycznych albo przez użycie niełańcuchowej reprezentacji chromosomów, albo wprowadzanie specyficznych dla zadania operatorów genetycznych. W [228] Koza zauważył:

> Reprezentacja jest kluczową rzeczą w pracach nad algorytmami genetycznymi, ponieważ schemat reprezentacji może istotnie ograniczyć okno, przez które system widzi swój świat. Jednakże, jak zauważyli Davis i Steenstrup [74] „W całej pracy Hollanda i w pracach wielu jego studentów chromosomy są łańcuchami binarnymi". Schematy reprezentacji oparte na łańcuchach są trudne i nienaturalne w wielu zadaniach i potrzeba szerszych reprezentacji była widoczna już od pewnego czasu [84], [85], [86].

Różne niestandardowe rozwiązania tworzono dla konkretnych zadań. Po prostu klasyczne algorytmy genetyczne trudno było stosować w tych zadaniach bezpośrednio i pewne modyfikacje struktur chromosomów były koniecznością. W tej książce świadomie odchodzimy od klasycznych algorytmów genetycznych, które operują na łańcuchach binarnych. Szukamy bogatszych struktur danych i możliwych do zastosowania z nimi operatorów „genetycznych" dla różnorodnych zadań. Eksperymentując z takimi strukturami i operatorami, otrzymaliśmy systemy, które nie były już algorytmami genetycznymi, a przynajmniej nie były algorytmami klasycznymi. Tytuły wielu opracowań zaczynały się od: „Zmodyfikowany algorytm genetyczny..." [271], „Specjalizowany algorytm genetyczny..." [201], „Niestandardowy algorytm genetyczny..." [278]. Było także odczucie, że termin „algorytmy genetyczne" może być mylący w stosunku do rozwijanych systemów. Davis opracował kilka niestandardowych systemów z wieloma operatorami ukierunkowanymi na zadanie. W [77] napisał on:

> Zauważyłem kiwanie głowami nad systemem innych badaczy zajmujących się algorytmami genetycznymi [...] i szczere zdumienie, że system, który opracowaliśmy, jest algorytmem genetycznym (gdyż nie używaliśmy reprezentacji binarnej, binarnego krzyżowania i binarnej mutacji).

Możemy dodatkowo zapytać na przykład, czy strategia ewolucyjna jest algorytmem genetycznym? A czy odwrotne pytanie jest prawdziwe? Aby uniknąć wszystkich spraw związanych z klasyfikacją systemów ewolucyjnych, nazywamy je po prostu „*programami ewolucyjnymi*".

Dlaczego odchodzimy od algorytmów genetycznych w kierunku bardziej elastycznych programów ewolucyjnych? Mimo że są one elegancko opracowane teoretycznie, algorytmy genetyczne nie zdały egzaminu w wielu zagadnieniach praktycznych. Wygląda na to, że główna przyczyna tych niepowodzeń jest taka sama, jak i ich sukcesu – niezależność od rozpatrywanej dziedziny.

Jedną z konsekwencji wygody stosowania algorytmów genetycznych (w sensie ich niezależności od dziedziny) jest niemożliwość uwzględniania w nich nietrywialnych ograniczeń. Jak wspomniano wcześniej, w większości prac związanych z algorytmami genetycznymi chromosomy są łańcuchami binarnymi – ciągami zer i jedynek. Istotnym zagadnieniem przy wyborze reprezentacji chromosomu dla rozwiązania zadania jest możliwość uwzględnienia ograniczeń nałożonych na rozwiązanie. Jak napisano w [74]:

> Ograniczenia, których nie wolno przekroczyć, można uwzględnić przez nałożenie wysokich kar na osobniki, które je naruszają, przez nałożenie umiarkowanych kar lub przez utworzenie dekoderów reprezentacji, które uniemożliwiają tworzenie osobników naruszających ograniczenia. Każde z tych rozwiązań ma zalety i wady. Jeżeli wprowadzimy wysokie kary do procedury oceniającej, a dziedzina jest taka, że tworzenie osobników naruszających ograniczenia jest wysoce prawdopodobne, to ryzykujemy tym, że utworzymy algorytm genetyczny, który większość czasu spędzi na ocenianiu nieprawidłowych osobników. Ponadto może się zdarzyć, że znajdzie się prawidłowy osobnik, który wyrzuci wszystkie pozostałe poza ograniczenia. Wtedy populacja zbiegnie się do niego, nie znajdując lepszych osobników, gdyż możliwe ścieżki do innych prawidłowych osobników wymagają utworzenia osobników nieprawidłowych, jako struktur przejściowych, a kary za przekroczenie ograniczeń spowodują, że dalsza reprodukcja takich struktur będzie mało prawdopodobna. Jeżeli nałożymy umiarkowane kary, to system może rozwinąć osobniki, które naruszają ograniczenia, ale są oceniane lepiej niż te, które ich nie naruszają, ponieważ główna część funkcji oceniającej może mieć przy umiarkowanej karze decydujące znaczenie. Jeżeli zaś wbudujemy do procedury oceniającej „dekoder", który w inteligentny sposób nie pozwoli na utworzenie nieprawidłowych osobników z istniejących chromosomów, to często otrzymamy program o zbytnich wymaganiach obliczeniowych. Ponadto nie wszystkie ograniczenia można łatwo uwzględnić w ten sposób.

(Przykłady dekoderów i algorytmów naprawy oraz kilka funkcji kary opisano w p. 4.5 przy rozważaniu zadania upakowania).

W programowaniu ewolucyjnym zagadnienie spełniania ograniczeń ma inny charakter. Nie chodzi tu o wybór funkcji oceniającej i kar, lecz raczej o wybór „najlepszej" reprezentacji chromosomowej rozwiązań oraz dobrych operatorów genetycznych, tak aby spełnić wszystkie ograniczenia występujące w zadaniu. Każdy operator genetyczny powinien przekazywać pewne charakterystyczne struktury od rodziców do potomstwa, więc struktury reprezentacji grają istotną rolę przy definiowaniu operatorów genetycznych. Ponadto różne struktury reprezentacji dają różne możliwości uwzględniania ograniczeń, co komplikuje zagadnienie jeszcze bardziej. Te dwa składniki (reprezentacja i operatory) są ze sobą związane. Wygląda więc na to, że każde zadanie wymaga dokładnej analizy, z której powinna wyniknąć odpowiednia reprezentacja oraz

odpowiednie dla niej operatory genetyczne. Glover w swoich badaniach nad rozwiązywaniem zadania konfiguracji złożonej klawiatury [141] napisał:

> Chociaż odporny charakter schematu poszukiwań za pomocą algorytmów genetycznych dobrze odpowiada wymaganiom zadania konfiguracji klawiatury, jednak reprezentacja binarna i wyidealizowane operatory nie do końca pasują do [...] wymaganych ograniczeń. Na przykład, jeżeli użyjemy trzech bitów do opisania każdego klawisza prostej klawiatury o 40 tylko klawiszach, to można łatwo wykazać, że tylko jedna z każdych 1016 dowolnie wybranych struktur 120-bitowych jest strukturą o właściwej konfiguracji klawiatury.

A oto inny cytat z pracy De Jonga [88], gdzie rozważa się zadanie komiwojażera:

> Za pomocą standardowych operatorów mutacji i krzyżowania algorytm genetyczny będzie przeszukiwał przestrzeń wszystkich *kombinacji* miast, gdy tymczasem należy przeszukiwać przestrzeń wszystkich *permutacji*. Oczywisty kłopot polega na tym, że gdy N (liczba miast do objechania) wzrasta, to przestrzeń permutacji jest znikomo małym podzbiorem przestrzeni kombinacji i mocne heurystyki próbkujące algorytmów genetycznych stają się bezsilne ze względu na zły wybór reprezentacji.

We wczesnych stadiach sztucznej inteligencji programy rozwiązywania zadań były powszechnie projektowane jako ogólne narzędzia do złożonych zadań. Jednakże, jak się okazało, trzeba było uwzględniać w nich wiedzę specyficzną dla zadania ze względu na niemożliwą do opanowania złożoność tych systemów. Teraz historia się powtarza. Aż do niedawna algorytmy genetyczne były rozumiane jako ogólne narzędzie użyteczne w optymalizacji wielu trudnych zadań. Jednak konieczność uwzględniania w ogólnych algorytmach genetycznych wiedzy specyficznej dla zadania była zauważana od pewnego czasu w niektórych artykułach naukowych [10], [128], [131], [170], [370]. Wygląda więc na to, że algorytmy genetyczne (jako powszechne programy rozwiązywania zadań) są zbyt niezależne od rozpatrywanej dziedziny, aby mogły być użyteczne w wielu zastosowaniach. Tak więc nie jest dziwne, że programy ewolucyjne, z wbudowaną w strukturach danych chromosomów wiedzą specyficzną dla zadania i specyficznymi operatorami „genetycznymi", działają lepiej.

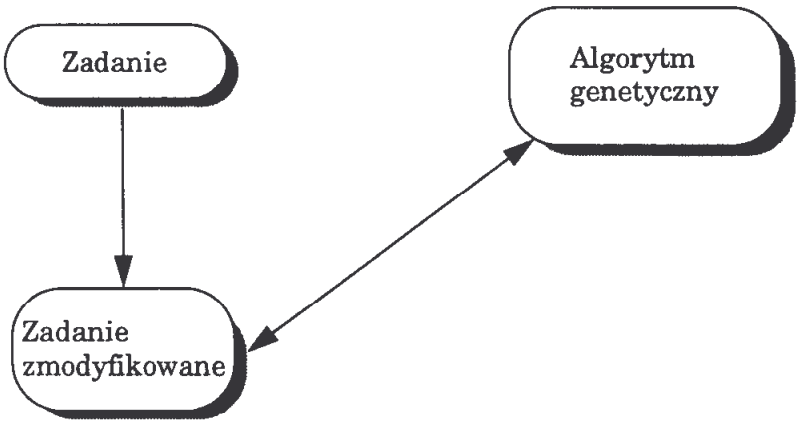
Podstawowa różnica koncepcyjna między klasycznymi algorytmami genetycznymi a programami ewolucyjnymi jest przedstawiona na rysunkach 0.2 i 0.3. Klasyczne algorytmy genetyczne, które operują na łańcuchach binarnych, wymagają modyfikacji pierwotnego zadania do odpowiedniej (nadającej się dla algorytmów genetycznych) postaci. W to może wchodzić odwzorowanie między potencjalnymi rozwiązaniami a binarną reprezentacją, rozważenie dekoderów lub algorytmów naprawy itp. Zazwyczaj nie jest to zbyt proste.

Programy ewolucyjne nie wymagają jednak zmiany zadania, lecz modyfikacji reprezentacji chromosomowej potencjalnych rozwiązań (używając „natu-

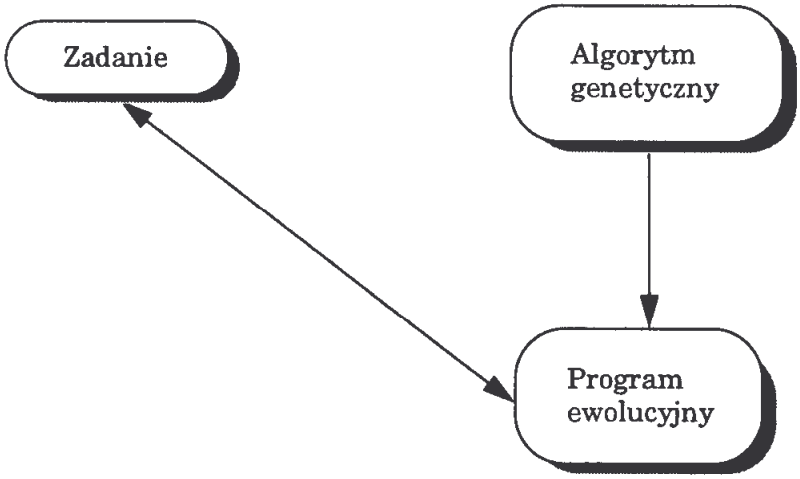
ralnych" struktur danych) i zastosowania odpowiednich operatorów „genetycznych".

Inaczej mówiąc, aby rozwiązać nietrywialne zadanie za pomocą programu ewolucyjnego, możemy albo przetransformować zadanie do postaci odpowiedniej dla algorytmu genetycznego (rys. 0.2), albo przetransformować algorytm genetyczny, tak aby odpowiadał on zadaniu (rys. 0.3). Rzecz jasna, klasyczne algorytmy genetyczne są związane z pierwszym podejściem, a programy ewolucyjne z drugim. Tak więc idea programów ewolucyjnych jest całkiem prosta i korzysta z następującego motta:

Nie przyszła góra do Mahometa, Mahomet przyszedł do góry.



Rys. 0.2. Schemat algorytmu genetycznego



Rys. 0.3. Schemat programu ewolucyjnego

Nie jest to zupełnie nowa idea. W [77] Davis napisał:

Już od pewnego czasu wydawało mi się, że nie można objąć większości zadań z rzeczywistości, uwzględniając tylko binarne reprezentacje i zbiór operatorów składający się tylko z binarnego krzyżowania i binarnej mutacji. Jednym z powodów tego jest to, że prawie każda praktyczna dziedzina ma własną

wiedzę, którą można uwzględnić, kiedy rozważa się transformację rozwiązania z tej dziedziny [...]. Wydaje mi się, że algorytmy genetyczne są odpowiednimi algorytmami, które można stosować w znacznej większości rzeczywistych zastosowań. Wydaje mi się także, że należy uwzględniać istniejącą wiedzę w algorytmach, wbudowując ją w dekodery lub rozszerzając odpowiednio zbiór operatorów.

Tutaj nazywamy takie zmodyfikowane algorytmy genetyczne „*programami ewolucyjnymi*".

Jest dosyć trudno przeciągnąć linię rozdzielającą algorytmy genetyczne i programy ewolucyjne. Kiedy program ewolucyjny jest algorytmem genetycznym? Gdy zachowuje populację potencjalnych rozwiązań? Gdy potencjalne rozwiązania mają binarną reprezentację? Gdy proces selekcji bazuje na dopasowaniu osobników? Gdy występują operatory rekombinacji? Gdy jest spełnione twierdzenie o schematach? Gdy jest prawdziwa hipoteza o blokach budujących? Czy wtedy, kiedy są spełnione wszystkie powyższe wymagania? Czy program ewolucyjny dla zadania komiwojażera z reprezentacją w postaci wektorów całkowitoliczbowych i operatorem PMX (rozdz. 10) jest algorytmem genetycznym? Czy program ewolucyjny dla zadania transportowego z reprezentacją macierzową i arytmetycznym operatorem krzyżowania (rozdz. 9) jest algorytmem genetycznym? W tej książce nie odpowiemy na te pytania. Zamiast tego przedstawimy pewne interesujące wyniki użycia metod programowania ewolucyjnego w różnorodnych zadaniach.

Jak wspomniano wcześniej, kilku badaczy zauważyło wagę różnych modyfikacji. W [78] Davis napisał:

Kiedy mówię do użytkownika, wyjaśniam mu, że planuję hybrydowe połączenie metody algorytmów genetycznych ze znanym algorytmem, stosując następujące trzy zasady:

- *Zachowanie kodowania*, tzn. użycie kodowania ze znanego algorytmu w algorytmie hybrydowym.
- *Łączenie tam, gdzie jest to możliwe*, tzn. uwzględnianie dodatnich cech znanego algorytmu w algorytmie hybrydowym.
- *Adaptację operatorów genetycznych*, tzn. utworzenie operatorów krzyżowania i mutacji w nowym programie przez analogię z operatorami krzyżowania i mutacji dla łańcuchów binarnych oraz dołączenie heurystyk związanych z dziedziną, jako dodatkowych operatorów.

[...] Dla algorytmów utworzonych na podstawie tych trzech zasad używam nazwy hybrydowe algorytmy genetyczne.

Wygląda na to, że hybrydowe algorytmy genetyczne i programy ewolucyjne łączy wspólna idea: odejście od klasycznych algorytmów genetycznych z łańcuchami binarnymi w kierunku bardziej skomplikowanych systemów, za-

wierających odpowiednie struktury danych (zachowanie kodowania) i odpowiednich operatorów genetycznych (adaptacja operatorów genetycznych). Davis zakłada jednak istnienie jednego lub większej liczby algorytmów tradycyjnych w rozpatrywanej dziedzinie i na podstawie takich algorytmów konstruuje hybrydowe algorytmy genetyczne. Przy naszym podejściu z programowaniem ewolucyjnym nie przyjmujemy żadnych założeń tego rodzaju. Wszystkie systemy ewolucyjne omawiane dalej w książce były zbudowane od początku.

Jakie są mocne i słabe strony programowania ewolucyjnego? Wygląda na to, że główną siłą metody programowania ewolucyjnego jest możliwość jej szerokiego zastosowania. W książce próbujemy opisać różnorodne zadania i omówić dla nich konstrukcję programu ewolucyjnego. Bardzo często wyniki są nadzwyczajne. Systemy działają dużo lepiej niż dostępne oprogramowanie komercyjne. Innym mocnym punktem programów ewolucyjnych jest to, że są one z natury równoległe. Jak stwierdzono w [154]:

> W świecie, w którym algorytmy sekwencyjne są przerabiane na równoległe za pomocą niezliczonych sztuczek i łamańców, jest niemałą ironią, że algorytmy genetyczne (algorytmy wysoko równoległe) są przerabiane na sekwencyjne za pomocą równie nienaturalnych sztuczek i wykrętów.

Jest to oczywiście także prawdą dla każdego (używającego populacji) programu ewolucyjnego. Natomiast musimy przyznać, że programy ewolucyjne mają słabą podstawę teoretyczną. Takiej podstawy dotychczas nie opracowano. Eksperymentowanie z różnymi strukturami danych i modyfikacje krzyżowania i mutacji wymaga starannej analizy, która zagwarantuje rozsądne działanie algorytmu.

Jednak pewne programy ewolucyjne mają teoretyczne podstawy. Można wykazać, że strategie ewolucyjne zastosowane do zadań regularnych (rozdz. 8) są zbieżne. Algorytmy genetyczne mają twierdzenie o schematach (rozdz. 3), które wyjaśnia dlaczego one działają. Dla innych programów ewolucyjnych bardzo często mamy tylko interesujące wyniki.

W ogólności strategie rozwiązywania zadań oparte na sztucznej inteligencji są dzielone na metody „mocne" i „słabe". W metodach słabych robi się mało założeń o dziedzinie zastosowań, wobec czego mogą one być szeroko używane. Jednak zdarza się, że wykazują one kombinatorycznie eksplodujący koszt uzyskania rozwiązania, gdy stosuje się je do większych zadań [90]. Można tego uniknąć, przyjmując mocne założenia o dziedzinie zastosowań i konsekwentnie używając tych założeń w metodzie rozwiązywania zadania. Lecz wadą takich mocnych metod jest ich ograniczone zastosowanie. Bardzo często wymagają one istotnego przeprojektowania, nawet wtedy, kiegdy chce się ich użyć w zbliżonym zadaniu.

Programy ewolucyjne znajdują się gdzieś między metodami słabymi a mocnymi. Niektóre programy ewolucyjne (jak algorytmy genetyczne) są całkiem słabe i nie wymagają żadnych założeń o dziedzinie zastosowania. Inne

programy (GENOCOP lub GENETIC-2) są bardziej uzależnione od zadań, z różnym stopniem uzależnienia. Na przykład GENOCOP (rozdz. 7), podobnie jak strategie ewolucyjne (rozdz. 8), opracowano do rozwiązywania zadań optymalizacji parametrycznej. System ten może uwzględnić każdą funkcję celu i każdy zbiór ograniczeń liniowych. GENETIC-2 (rozdz. 9) jest przeznaczony do zadań transportowych. Inne systemy (patrz rozdz. 10 i 11) są odpowiednie dla zadań optymalizacji kombinatorycznej jak harmonogramowanie, zadanie komiwojażera lub zadań kolorowania grafów. Ciekawe zastosowanie programu ewolucyjnego do indukcyjnego uczenia reguł decyzyjnych jest omawiane w rozdz. 12.

Zabawne, że algorytmy genetyczne są uważane za metody słabe. Przy nietrywialnych ograniczeniach zmieniają się one przecież szybko w metody mocne. Czy przyjmiemy funkcje kary, dekodery czy też algorytmy naprawy, to muszą one być związane ze specyficznym zastosowaniem. Programy ewolucyjne (uważane za metody znacznie mocniejsze i zależne od zadania) nagle wydają się znacznie słabsze (dyskutujemy o tym w rozdz. 14). Przedstawia to wielki potencjał tkwiący w podejściu programowania ewolucyjnego.

Wszystkie te obserwacje wywoływały moje zainteresowanie badaniem właściwości różnych operatorów genetycznych zdefiniowanych na bogatszych strukturach niż łańcuchy binarne. Ponadto te badania doprowadziły do utworzenia nowej metodologii programowania (w [277] taką zaproponowaną metodologię nazwano EVA, od *Evolution progrAmming*). Z grubsza mówiąc, programista w takim środowisku wybiera struktury danych z odpowiednimi dla danego zadania operatorami genetycznymi oraz wybiera funkcję oceniającą i sposób wyboru populacji początkowej (pozostałe parametry są dostrajane przez inny algorytm genetyczny).

Jednak wiele jeszcze trzeba wykonać badań, zanim będziemy mogli zaproponować podstawową konstrukcję takiego środowiska programowego. Ta książka jest tylko pierwszym krokiem w tym kierunku i obejmuje badania różnych struktur i operatorów genetycznych, które składają się na programy ewolucyjne dla wielu zadań.

Powrócimy do pomysłu nowego środowiska programowego w podsumowaniu na końcu książki (rozdz. 14).

Część I
Algorytmy genetyczne

1

Algorytmy genetyczne: co to jest?

Mistrz stale podkreślał –
choć wyglądało to na paradoks –
że prawdziwym reformatorem może być
jedynie taki człowiek, który dostrzega,
że rzecz – w swym aktualnym stanie –
jest doskonała, i który pozwala jej być
taką, jaka jest.

Anthony de Mello, *Minuta mądrości*[1)]

Dla dużej klasy ważnych zadań nie opracowano dotąd dostatecznie szybkich algorytmów ich rozwiązywania. Wiele z nich to zadania optymalizacji występujące w zastosowaniach. Dla takich trudnych zadań optymalizacji często można znaleźć wydajny algorytm, który nie gwarantuje jednak uzyskania rozwiązania optymalnego, a tylko w przybliżeniu optymalne. Dla niektórych trudnych zadań optymalizacji możemy używać również algorytmów probabilistycznych. Te algorytmy także nie gwarantują, że zostanie uzyskane rozwiązanie optymalne, ale przez losowy wybór dostatecznie wielu „przedstawicieli" prawdopodobieństwo błędu można uczynić tak małym, jak tylko chcemy.

Takie algorytmy o wysokiej jakości są dostępne dla wielu ważnych praktycznych zadań optymalizacji [73]. Możemy na przykład użyć algorytmu symulowanego wyżarzania w zadaniu prowadzenia przewodów i rozmieszczania elementów przy projektowaniu VLSI lub w zadaniu komiwojażera. Także wiele innych kombinatorycznych zadań optymalizacji o wielkiej skali (z których wiele okazuje się NP-trudnymi; NP – *Nondeterministic Polynomial*, tzn. wielomianowe niedeterministyczne) można rozwiązać w przybliżeniu na obecnie używanych komputerach za pomocą podobnej metody Monte Carlo.

Ogólnie, jakąkolwiek czynność do wykonania można przedstawić jako rozwiązywanie zadania, co można z kolei rozumieć jako przeszukiwanie przestrzeni możliwych rozwiązań. Ponieważ zależy nam na „najlepszym" rozwiązaniu, możemy uważać to zadanie za proces optymalizacji. W małych przestrzeniach klasyczne metody pełnego przeszukiwania zwykle wystarczają. W większych przestrzeniach trzeba stosować specjalizowane metody sztucznej inteligencji. Wśród nich znajdują się algorytmy genetyczne. Są to algorytmy stochastyczne, których sposób przeszukiwania naśladuje pewne procesy naturalne: dziedziczenie genetyczne i darwinowską walkę o przeżycie. Jak napisano w [74]:

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

...przenośnia leżąca u podstaw algorytmów genetycznych jest związana z ewolucją w naturze. W trakcie ewolucji każdy gatunek styka się z problemem lepszej adaptacji do skomplikowanego i zmiennego środowiska. „Wiedza", jaką każdy gatunek uzyskał, jest wbudowana w układ chromosomów jego osobników.

Pomysł leżący u podstaw algorytmów genetycznych polega na zrobieniu tego samego, co robi natura. Weźmy dla przykładu króliki. W każdej chwili mamy jakąś populację królików. Niektóre z nich są szybsze i sprytniejsze od innych. Szybsze i sprytniejsze króliki mają większą szansę umknięcia przed lisem. Wobec tego więcej ich przeżywa, aby zrobić to, co króliki robią najlepiej, a mianowicie nowe króliki. Oczywiście, niektóre z wolniejszych i głupszych królików też przeżyją, bo mają szczęście. Ta ocalała populacja królików wydaje potomstwo. Jest ono dobrą mieszaniną materiału genetycznego. Niektóre wolne króliki krzyżują się z szybkimi, niektóre szybkie z szybkimi, niektóre sprytne z głupimi itd. A do tego natura dorzuca od czasu do czasu „dziką kartę", wprowadzając mutację do genetycznego materiału królików. Urodzone w wyniku takiego procesu króliki będą (średnio) szybsze i sprytniejsze niż te z początkowej populacji, ponieważ szybsi i sprytniejsi rodzice umknęli przed lisami. (Dobrze, że lisy podlegają temu samemu procesowi – inaczej króliki mogłyby stać się zbyt szybkie i sprytne, aby pozwolić się złapać lisom).

Algorytm genetyczny naśladuje krok po kroku procedurę przedstawioną w opowiadaniu o królikach. Zanim przyjrzymy się dokładniej strukturze algorytmu genetycznego, popatrzmy na historię genetyki (na podstawie [380]).

Podstawową zasadę wyboru naturalnego jako główną zasadę ewolucji sformułował C. Darwin dużo wcześniej przed odkryciem mechanizmu genetycznego. Nie znając podstawowych zasad dziedziczenia, Darwin wyobrażał je sobie jako stapianie lub mieszanie się cech, przypuszczając, że cechy rodziców mieszają się jak płyny w organizmie potomka. Jego teoria wyboru napotkała na poważne zarzuty, po raz pierwszy postawione przez F. Jenkinsa: przechodząc szybko przez okres zróżnicowania dziedzicznego, dojdziemy do poziomu, w którym nie będzie już możliwości wyboru w jednorodnej populacji (tzw. „zmora Jenkinsa").

Dopiero w 1865 r., gdy Mendel odkrył podstawowe prawa przekazywania cech dziedzicznych od rodziców do potomków, wskazujące na nieciągłą naturę tych cech, można było wyjaśnić „zmorę Jenkinsa", gdyż ze względu na ową nieciągłość nie może zachodzić „zanikanie" różnic dziedzicznych.

Prawa Mendla zostały szerzej poznane przez społeczność naukową dopiero wtedy, kiedy niezależnie odkryli je w 1900 r. H. de Vries, K. Correns i K. von Tschermak. Genetykę w pełni rozwinęli T. Morgan i jego współpracownicy, którzy udowodnili eksperymentalnie, że chromosomy są podstawowymi nośnikami dziedziczonej informacji i że geny, związane z dziedziczonymi cechami, są połączone w chromosomach liniowo. Późniejsze kolejne wyniki do-

świadczalne wykazały, że prawa Mendla odnoszą się do wszystkich organizmów rozmnażających się płciowo.

Jednakże prawa Mendla, nawet po tym, jak je ponownie odkryto, oraz teoria Darwina wyboru naturalnego pozostały niezależnymi, nie łączonymi razem pojęciami. A dodatkowo przeciwstawiano je sobie. Dopiero w latach dwudziestych (patrz na przykład klasyczna praca Četverikova [61]) dowiedziono, że teoria genetyczna Mendla i teoria Darwina o wyborze naturalnym w żaden sposób nie kolidują ze sobą, a ich szczęśliwe małżeństwo prowadzi do współczesnej teorii ewolucji.

Do opisu algorytmów genetycznych używa się słownictwa zapożyczonego z genetyki naturalnej. Mówimy o *osobnikach* (lub *genotypach*, *strukturach*) w populacji. Często osobniki te są także nazywane *łańcuchami* lub *chromosomami*. Może to być trochę mylące. Każda komórka organizmu danego gatunku zawiera pewną liczbę chromosomów (na przykład u człowieka 46). W tej książce mamy na myśli osobniki z pojedynczymi chromosomami, to znaczy chromosomami *haploidalnymi* (dodatkowe wiadomości o *diploidach* – parach chromosomów, dominacji i innych związanych z tym sprawach w powiązaniu z algorytmami genetycznymi można znaleźć w [154], a także w niedawnej pracy Greene'a [165] oraz Nga i Wonga [298]). Chromosomy składają się z *genów* (*cech*, *znaków*, *dekoderów*) uszeregowanych liniowo. Każdy gen decyduje o dziedziczności jednej lub kilku cech. Geny pewnych typów są umieszczone w pewnych miejscach chromosomu, zwanych *pozycją* (*locus* – miejscem w łańcuchu). Jakakolwiek cecha osobnika (taka jak kolor włosów) objawia się inaczej – mówimy, że gen jest w kilku stanach, zwanych *allelami* (wartościami cech).

Każdy genotyp (w tej książce pojedynczy chromosom) reprezentuje potencjalne rozwiązanie zadania (znaczenie poszczególnego chromosomu, tzn. jego *fenotyp*, jest definiowane zewnętrznie przez użytkownika). Proces ewolucyjny zachodzący w populacji chromosomów odpowiada przeszukaniu przestrzeni potencjalnych rozwiązań. Takie przeszukiwanie wymaga pogodzenia dwóch (w sposób oczywisty sprzecznych) celów: skorzystania z najlepszych dotychczasowych rozwiązań i szerokiego przebadania przeszukiwanej przestrzeni [46]. Metoda wzrostu jest przykładem strategii, która korzysta z najlepszego dotychczasowego rozwiązania w celu jego poprawienia. Przeszukiwanie losowe jest natomiast typowym przykładem strategii, w której bada się przeszukiwaną przestrzeń, nie zwracając uwagi na jej obiecujące regiony. Algorytmy genetyczne są klasą ogólniejszych metod przeszukiwania (niezależnych od dziedziny), w których utrzymuje się możliwie jak najlepszą równowagę między szerokim badaniem przestrzeni a korzystaniem z wcześniejszych wyników.

Algorytmy genetyczne były z powodzeniem stosowane w zadaniach optymalizacji, takich jak wytyczanie trasy połączeń kablowych, harmonogramowanie, sterowanie adaptacyjne, rozgrywanie gier, modelowanie poznawcze, zadania transportowe, zadanie komiwojażera, sterowanie optymalne, optymalizacja obsługi pytań w bazach danych itp. (patrz [15], [34], [45], [84], [121], [154],

[167], [170], [171], [273], [344], [129], [103], [391], [392]). Jednak De Jong [84] ostrzega przed patrzeniem na algorytmy genetyczne pod kątem narzędzia do optymalizacji:

> ...z powodu historycznego nagromadzenia prac i nacisku na zastosowania do optymalizacji funkcji łatwo wpaść w pułapkę uważania algorytmów genetycznych za algorytmy optymalizacyjne, co prowadzi do zdziwienia i rozczarowania, gdy nie znajdują one „oczywistych" optimów w jakiejś przeszukiwanej przestrzeni. Moja rada na uniknięcie takiej pułapki pojęciowej polega na myśleniu o algorytmach genetycznych jako o (bardzo wyidealizowanej) symulacji naturalnego procesu, który, jako taki, obejmuje cele i zamierzenia (jeżeli istnieją) tego procesu. Nie jestem pewny, czy ktokolwiek próbował określić cele i zamierzenia systemów ewolucyjnych. Jednak myślę, że można szczerze powiedzieć, że takie systemy nie są ogólnie rozumiane jako metody optymalizacyjne.

Z drugiej strony, optymalizacja jest jednym z głównych obszarów zastosowań algorytmów genetycznych. W [348] Schwefel napisał:

> Z trudem można znaleźć nowoczesne czasopismo, inżynierskie, ekonomiczne, matematyczne, fizyczne, z zakresu zarządzania czy nauk społecznych, w którym pojęcie „optymalizacja" nie występuje w wykazie haseł. Jeżeli abstrahujemy od punktu widzenia specjalistów, to powtarzającym się zadaniem jest wybór lepszej lub najlepszej (albo zgodnie z Leibnizem, optymalnej) możliwości spośród wielu możliwych stanów sprawy.

W czasie ostatniej dekady znaczenie optymalizacji wzrosło nawet bardziej – wiele ważnych zadań optymalizacji kombinatorycznej o dużej skali i zadań inżynierskich z wieloma ograniczeniami można za pomocą obecnych komputerów rozwiązać tylko w sposób przybliżony.

Algorytmy genetyczne nadają się do tak skomplikowanych zadań. Należą one do klasy algorytmów probabilistycznych, jednak różnią się znacznie od algorytmów czysto losowych, gdyż łączą elementy przeszukiwania bezpośredniego i stochastycznego. Z tego powodu algorytmy genetyczne są także bardziej niezawodne niż istniejące algorytmy bezpośredniego przeszukiwania. Inną ważną cechą takich metod przeszukiwania opartych na rozwiązaniach genetycznych jest to, że zachowują one całą populację potencjalnych rozwiązań, gdy tymczasem inne metody przetwarzają tylko jeden punkt przeszukiwanej przestrzeni.

W metodach wzrostu stosuje się poprawianie iteracyjne, bazując na jednym (bieżącym) punkcie przeszukiwanej przestrzeni. Podczas jednej iteracji nowy punkt wybiera się z otoczenia bieżącego punktu (dlatego też o tych metodach mówi się także jako o przeszukiwaniu otoczenia lub przeszukiwaniu lokalnym [233]). Jeżeli w nowym punkcie wartość funkcji celu jest lepsza (mniejsza przy minimalizacji i większa przy maksymalizacji), to nowy punkt staje się punktem bieżącym. Jeżeli nie, to wybiera się inny punkt i sprawdza się

go z punktem bieżącym. Jeżeli nie można uzyskać poprawy, poszukiwanie kończy się.

Jest oczywiste, że metody wzrostu prowadzą do lokalnych wartości optymalnych i że te wartości zależą od wyboru punktu startowego. Ponadto nie dają one żadnej informacji o błędzie względnym (w stosunku do optimum globalnego) znalezionego rozwiązania.

Aby zwiększyć szansę pełnego sukcesu, obliczenia w metodach wzrostu na ogół zaczyna się z dużej liczby różnych punktów startowych (nie muszą one być wybierane losowo – wybór punktu startowego dla następnego obliczania może zależeć od wyników poprzedniego obliczenia).

W metodzie symulowanego wyżarzania [1] wiele wad metod wzrostu jest wyeliminowane. Rozwiązania nie zależą już w nich od punktu startowego i są (na ogół) bliskie punktowi optymalnemu. Uzyskuje się to przez wprowadzenie prawdopodobieństwa p akceptacji (tzn. zamiany bieżącego punktu na nowy punkt). Jeżeli w nowym punkcie funkcja celu przyjmuje mniejszą wartość, to $p = 1$. Jeżeli jednak tak nie jest, to $p > 0$. W tym drugim przypadku prawdopodobieństwo akceptacji p jest funkcją zależną od wartości funkcji celu w punktach bieżącym i nowym oraz dodatkowego parametru sterującego T, zwanego *temperaturą*. W ogólności, im niższa jest temperatura T, tym mniejsza jest szansa zakceptowania nowego punktu. Podczas wykonywania algorytmu temperatura systemu T jest stopniowo obniżana. Algorytm zatrzymuje się przy pewnej małej wartości T, dla której faktycznie nie są już akceptowane żadne zmiany.

Jak wspomniano wcześniej, algorytm genetyczny wykonuje wielokierunkowe przeszukiwanie przez przekształcanie populacji potencjalnych rozwiązań i prowadzi do zbierania informacji genetycznej i jej wymiany między tymi kierunkami. Populacja podlega symulowanej ewolucji: w każdym następnym pokoleniu stosunkowo „dobre" rozwiązania reprodukują się, a stosunkowo „złe" wymierają. Aby rozróżnić rozwiązania, używamy funkcji celu (oceny), która odgrywa rolę środowiska.

Przykłady metod gradientowych, symulowanego wyżarzania i algorytmów genetycznych będą podane w dalszym ciągu tego rozdziału (p. 1.4).

Struktura prostego algorytmu genetycznego jest taka sama jak struktura jakiegokolwiek programu ewolucyjnego (patrz rys. 0.1 we wprowadzeniu). W iteracji t algorytm genetyczny zachowuje populację potencjalnych rozwiązań (chromosomów, wektorów) $P(t) = \{x_1^t, ..., x_n^t\}$. Każde rozwiązanie x_i^t jest oceniane pod kątem wartości jego „dopasowania". Następnie w iteracji $t + 1$ formuje się nową populację przez wybór osobników najlepiej dopasowanych. Niektórzy członkowie tej nowej populacji podlegają zmianom za pomocą krzyżowania i mutacji, tworząc w ten sposób nowe rozwiązania. Krzyżowanie prowadzi do połączenia cech dwóch rodzicielskich chromosomów w chromosomach dwóch potomków przez wymianę odcinków chromosomów rodziców. Dla przykładu, jeżeli rodzice są reprezentowani przez pięcioelementowe wektory $(a_1, b_1, c_1, d_1, e_1)$ i $(a_2, b_2, c_2, d_2, e_2)$, to krzyżowanie chromosomów po

drugim genie doprowadzi do potomków $(a_1, b_1, c_2, d_2, e_2)$ i $(a_2, b_2, c_1, d_1, e_1)$. Intuicyjnie można interpretować krzyżowanie jako wymianę informacji genetycznej między potencjalnymi rozwiązaniami.

Mutacja polega na losowej zmianie jednego lub więcej genów wybranego chromosomu, z prawdopodobieństwem równym częstości mutacji. Intuicyjnie operator mutacji można rozumieć jako wprowadzenie pewnej dodatkowej zmienności w populacji.

Algorytm genetyczny (jak każdy program ewolucyjny) dla każdego szczególnego zadania musi zawierać następujących 5 elementów:

- podstawową reprezentację potencjalnych rozwiązań zadania,
- sposób tworzenia początkowej populacji potencjalnych rozwiązań,
- funkcję oceniającą, która gra rolę środowiska i ocenia rozwiązania według ich „dopasowania",
- podstawowe operatory, które wpływają na skład populacji dzieci,
- wartości różnych parametrów używanych w algorytmie genetycznym (rozmiar populacji, prawdopodobieństwa użycia operatorów genetycznych itp.).

Omówimy główne cechy algorytmów genetycznych na 3 przykładach. W pierwszym użyjemy algorytmu genetycznego do optymalizacji prostej funkcji jednej zmiennej rzeczywistej. Drugi przykład zilustruje użycie algorytmu genetycznego do nauki strategii w prostej grze (dylemat więźnia). W trzecim przykładzie przedstawimy możliwe zastosowanie algorytmu genetycznego do zadania NP-trudnego, a mianowicie zadania komiwojażera.

1.1. Optymalizacja prostej funkcji

W tym punkcie przedstawimy podstawowe cechy algorytmu genetycznego do optymalizacji prostej funkcji jednej zmiennej. Funkcja ta ma postać

$$f(x) = x\sin(10\pi x) + 1{,}0$$

i jest wykreślona na rys. 1.1. Zadanie polega na znalezieniu x z przedziału $[-1, 2]$ maksymalizującego funkcję f, to znaczy znalezieniu x_0 takiego, że

$$f(x_0) \geqslant f(x) \text{ dla wszystkich } x \in [1, 2]$$

Funkcję f można łatwo zbadać. Należy znaleźć zera pierwszej pochodnej f'

$$f'(x) = \sin(10\pi x) + 10\pi x\cos(10\pi x) = 0$$

co można przekształcić do

$$\mathrm{tg}(10\pi x) = -10\pi x$$

Powyższe równanie ma oczywiście nieskończoną liczbę rozwiązań

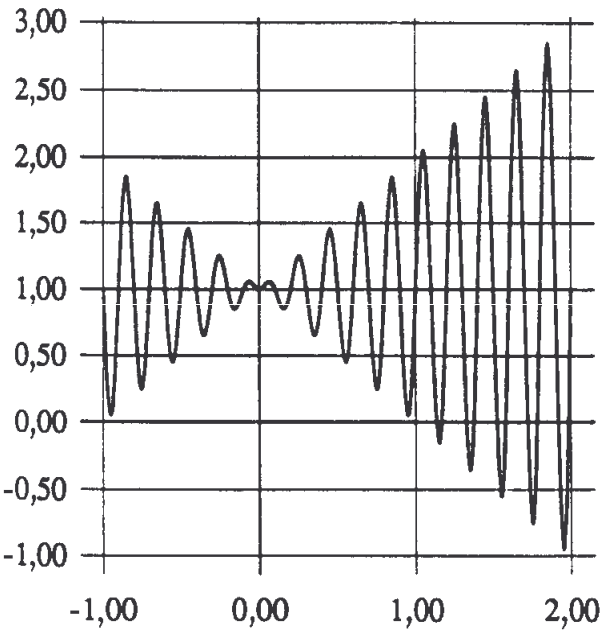
$$x_i = (2i - 1)/20 + \varepsilon_i, \quad i = 1, 2, \ldots$$

$$x_0 = 0$$

$$x_i = (2i + 1)/20 - \varepsilon_i, \quad i = -1, -2, \ldots$$

przy czym ε_i jest ciągiem malejących liczb rzeczywistych (zarówno dla $i = 1, 2, \ldots$, jak i $i = -1, -2, \ldots$), dążących do zera.

Zauważmy także, że funkcja f osiąga w x_i lokalne maksima, gdy i jest nieparzyste, a lokalne minima, gdy i jest parzyste (patrz rys. 1.1).



Rys. 1.1. Wykres funkcji $f(x) = x \cdot \sin(10\pi x) + 1{,}0$

Ponieważ dziedziną funkcji jest przedział $[-1, 2]$, zatem funkcja osiąga maksimum dla $x_{19} = 37/20 + \varepsilon_{19} = 1{,}85 + \varepsilon_{19}$, przy czym $f(x_{19})$ jest nieznacznie większe niż $f(1{,}85) = 1{,}85 \sin(18\pi + \pi/2) + 1{,}0 = 2{,}85$.

Przypuśćmy, że chcemy skonstruować algorytm genetyczny do rozwiązania powyższego zadania, tzn. znalezienia maksimum funkcji f. Omówimy po kolei główne składniki takiego algorytmu.

1.1.1. Reprezentacja

Do reprezentacji rzeczywistych wartości zmiennej x użyjemy chromosomu w postaci wektora binarnego. Długość tego wektora zależy od żądanej dokładności obliczenia rozwiązania, którą przykładowo przyjmiemy jako 6 cyfr po przecinku.

Dziedzina zmienności x ma długość 3. Żądana dokładność wymaga, aby przedział $[-1, 2]$ był podzielony na co najmniej $3 \cdot 1\,000\,000$ równych podprzedziałów. Oznacza to, że wektor binarny (chromosom) musi mieć 22 bity, gdyż

$$2\,097\,152 = 2^{21} < 3\,000\,000 \leqslant 2^{22} = 4\,194\,304$$

Odwzorowanie łańcucha binarnego $\langle b_{21}\, b_{20} \ldots b_0 \rangle$ w liczbę rzeczywistą x z zakresu $[-1, 2]$ jest proste i można je wykonać w dwóch krokach:

- przekształcenie łańcucha binarnego $\langle b_{21}\, b_{20} \ldots b_0 \rangle$ z sytemu dwójkowego na system dziesiętny: $(\langle b_{21}\, b_{20} \ldots b_0 \rangle)_2 = (\sum_{i=0}^{21} b_i\, 2^i)_{10} = x'$,
- obliczenie odpowiedniej liczby rzeczywistej x: $x = -1{,}0 + 3x'/(2^{22} - 1)$, gdzie $-1{,}0$ jest lewą granicą dziedziny, a 3 jest długością przedziału.

Na przykład chromosom

(1000101110110101000111)

przedstawia liczbę 0,637197, gdyż

$$x' = (1000101110110101000111)_2 = 2288967$$

i

$$x = -1{,}0 + 2\,288\,967 \cdot \frac{3}{4\,194\,303} = 0{,}637197$$

Oczywiście chromosomy

(0000000000000000000000) i (1111111111111111111111)

reprezentują odpowiednie granice dziedziny, $-1{,}0$ i $2{,}0$.

1.1.2. Populacja początkowa

Rozpoczęcie procesu jest bardzo proste. Tworzymy populację chromosomów, w której każdy chromosom jest wektorem binarnym o 22 bitach. Wszystkie 22 bity w każdym chromosomie są wyznaczone losowo.

1.1.3. Funkcja oceny

Funkcja oceny *eval* z binarnym wektorem **v** jako argumentem jest równa funkcji f:

$$\mathrm{eval}(\mathbf{v}) = f(x)$$

przy czym chromosom **v** reprezentuje wartość rzeczywistą x.

Jak zauważono wcześniej, funkcja oceny gra rolę środowiska, oceniając potencjalne rozwiązania według ich dopasowania. Na przykład 3 chromosomy

$$\mathbf{v}_1 = (100010111011010100011 1)$$

$$\mathbf{v}_2 = (000000111000000000 10000)$$

$$\mathbf{v}_3 = (11100000001111111000101)$$

odpowiadają wartościom $x_1 = 0{,}637197$, $x_2 = -0{,}958973$ i $x_3 = 1{,}627888$. Wobec tego funkcja oceny przyporządkuje im następujące oceny:

$$eval(\mathbf{v}_1) = f(x_1) = 1{,}586345$$

$$eval(\mathbf{v}_2) = f(x_2) = 0{,}078878$$

$$eval(\mathbf{v}_3) = f(x_3) = 2{,}250650$$

Oczywiście chromosom $\mathbf{v}_3$ jest z nich najlepszy, gdyż jego ocena jest najwyższa.

1.1.4. Operatory genetyczne

W fazie zmian w algorytmie genetycznym użyjemy dwóch klasycznych operatorów genetycznych: mutacji i krzyżowania.

Jak wspomniano wcześniej, mutacja zmienia jeden lub więcej genów (miejsc w chromosomie) z prawdopodobieństwem równym częstości mutacji. Przypuśćmy, że do mutacji wybrano piąty gen z chromosomu $\mathbf{v}_3$. Ponieważ piąty gen tego chromosomu jest 0, to byłby on zmieniony na 1. Tak więc chromosom $\mathbf{v}_3$ po tej mutacji wyglądałby następująco:

$$\mathbf{v}'_3 = (11101000001111111000101)$$

Ten chromosom reprezentuje liczbę $x'_3 = 1{,}721638$ i $f(x'_3) = -0{,}082257$. Oznacza to, że ta akurat mutacja doprowadziła do znacznego zmniejszenia oceny chromosomu $\mathbf{v}_3$. Z drugiej strony, gdyby do mutacji wybrano dziesiąty gen chromosomu $\mathbf{v}_3$, to otrzymalibyśmy

$$\mathbf{v}''_3 = (11100000011111111000101)$$

Odpowiada mu wartość $x''_3 = 1{,}630818$ i $f(x''_3) = 2{,}343555$, co jest poprawą w stosunku do wcześniejszej wartości $f(x_3) = 2{,}250650$.

Zilustrujemy teraz operator krzyżowania chromosomów $\mathbf{v}_2$ i $\mathbf{v}_3$. Przypuśćmy, że punkt zamiany wybrano (losowo) po piątym genie:

$$\mathbf{v}_2 = (00000|01110000000010000)$$

$$\mathbf{v}_3 = (11100|00000111111000101)$$

Dwóch wynikowych potomków będzie miało postać

$$\mathbf{v}_2' = (00000|00000111111000101)$$

$$\mathbf{v}_3' = (11100|01110000000010000)$$

Będą oni ocenieni następująco:

$$f(\mathbf{v}_2') = f(-0{,}998113) = 0{,}940865$$

$$f(\mathbf{v}_3') = f(1{,}666028) = 2{,}459245$$

Zauważmy, że drugi potomek ma lepszą ocenę od obu jego rodziców.

1.1.5. Parametry

W tym zadaniu użyliśmy następujących parametrów: rozmiar populacji $pop_size = 50$, prawdopodobieństwo krzyżowania $p_c = 0{,}25$, prawdopodobieństwo mutacji $p_m = 0{,}01$. Następny podpunkt przedstawia wyniki obliczeń dla takiego systemu genetycznego.

1.1.6. Wyniki obliczeń

W tablicy 1.1 przedstawiono numery pokoleń, w których zanotowano poprawę funkcji oceny, oraz wartość tej funkcji. Najlepszy chromosom w 150 pokoleniu miał postać

$$\mathbf{v}_{max} = (11110011010001000001 01)$$

co odpowiada wartości $x_{max} = 1{,}850773$.

Jak się tego można było spodziewać, $x_{max} = 1{,}85 + \varepsilon$ i $f(x_{max})$ jest nieco większe od 2,85.

Tablica 1.1. Wyniki po 150 pokoleniach

Numer pokolenia	Funkcja oceny
1	1,441942
6	2,250003
8	2,250283
9	2,250284
10	2,250363
12	2,328077
39	2,344251
40	2,345087
51	2,738930
99	2,849246
137	2,850217
145	2,850227

1.2. Dylemat więźnia

W tym punkcie wyjaśnimy, jak algorytm genetyczny może nauczyć się strategii postępowania w prostej grze zwanej *dylematem więźnia*. Przedstawiamy tu wyniki uzyskane przez Axelroda [14].

Dwaj więźniowie są przetrzymywani w odrębnych celach, bez możliwości komunikowania się ze sobą. Każdy z nich jest niezależnie namawiany do zdrady drugiego. Jeżeli jeden więzień zdradzi, będzie nagrodzony, a drugi więzień będzie ukarany. Jeżeli obaj zdradzą, obaj będą uwięzieni i torturowani. Jeżeli żaden nie zdradzi, obaj otrzymają umiarkowaną nagrodę. Tak więc samolubny wybór zdrady jest zawsze bardziej nagradzany niż współpraca – niezależnie, co zrobi drugi więzień – ale jeżeli obaj zdradzą, obaj wyjdą na tym gorzej, niż gdyby współpracowali. Dylemat więźnia polega więc na tym, czy zdradzić, czy też współpracować z pozostałym więźniem.

Dylemat więźnia można traktować jako grę między dwoma graczami. W kolejnych ruchach każdy gracz albo zdradza, albo współpracuje z drugim więźniem. W tablicy 1.2 przedstawiono listę wypłat dla każdego z graczy.

Tablica 1.2. Lista wypłat dla gry „dylemat więźnia"

Gracz 1	Gracz 2	P_1	P_2	Uwagi
Zdrada	Zdrada	1	1	Kara za wspólną zdradę
Zdrada	Współpraca	5	0	Zachęta do zdrady i zapłata przez naiwnego
Współpraca	Zdrada	0	5	Zapłata przez naiwnego i zachęta do zdrady
Współpraca	Współpraca	3	3	Nagroda za współpracę

Zastanowimy się teraz, jak można użyć algorytmu genetycznego do nauki strategii postępowania w dylemacie więźnia. W podejściu tym mamy utworzyć populację „graczy", z których każdy charakteryzuje się jakąś szczególną strategią. Na początku strategia każdego gracza jest wybierana losowo. Dalej, w każdym kolejnym kroku, gracze rozgrywają gry, a ich wyniki są notowane. Następnie niektórzy gracze są wybierani do następnego pokolenia, a niektórzy zaś do skojarzenia w pary. Otrzymany z nich nowy gracz otrzymuje strategię utworzoną ze strategii jego rodziców (krzyżowanie). Jak zwykle mutacja wprowadza dodatkową zmienność do strategii graczy przez przypadkowe zmiany w reprezentacjach tych strategii.

1.2.1. Reprezentacja strategii

Przede wszystkim potrzebujemy pewnego pomysłu na stworzenie reprezentacji strategii (tzn. potencjalnego rozwiązania). Dla uproszczenia rozważymy tylko strategie deterministyczne, a do wyboru następnego ruchu będziemy brali pod

uwagę tylko wyniki trzech ostatnich ruchów. Ponieważ są cztery możliwe wyniki każdego ruchu, zatem dla trzech poprzednich ruchów jest $4 \times 4 \times 4 = 64$ różnych możliwości.

Strategię tego typu można zapisać, wskazując, jaki ruch należy wykonać dla każdej z powyższych możliwości. A więc strategię może reprezentować łańcuch 64-bitowy (ewentualnie składający się z liter Z i W), wskazujący, jaki ruch należy wykonać dla każdej z 64 możliwości. Aby móc stosować strategię na początku gry, musimy zadać trzy hipotetyczne ruchy, które poprzedziły rozpoczęcie gry. To wymaga sześciu dodatkowych genów, co daje w sumie 70 miejsc w chromosomie.

Ten łańcuch 70 bitów zawiera informację, co powinien zrobić gracz w każdej możliwej sytuacji i wobec tego całkowicie określa poszczególną strategię. Łańcuch 70 genów służy także za chromosom gracza w procesie ewolucyjnym.

1.2.2. Schemat algorytmu genetycznego

Algorytm genetyczny Axelroda uczący się strategii gry w dylemacie więźnia działa w 4 krokach:

1. Wybierz początkową populację. Każdemu graczowi przyporządkowuje się losowo łańcuch 70-bitowy reprezentujący strategię, tak jak to przedstawiono powyżej.
2. Sprawdź każdego gracza, aby określić jego efektywność. Każdy gracz używa strategii zapisanej w swoim chromosomie, aby rozgrywać gry z pozostałymi graczami. Wynik gracza jest średnią ze wszystkich rozegranych przez niego gier.
3. Wybierz graczy do rozmnożenia. Gracz ze średnim wynikiem dostaje jednego partnera do skojarzenia, gracz z wynikiem o jedno odchylenie standardowe lepszym dostaje dwóch partnerów, a gracz z wynikiem o jedno odchylenie standardowe gorszym od średniego nie dostaje żadnego partnera.
4. Wygrywający gracze są losowo kojarzeni w celu utworzenia dwóch potomków z jednej pary. Strategia potomków jest tworzona ze strategii rodziców. Do tego służą dwa operatory genetyczne: krzyżowanie i mutacja.

Po tych czterech krokach otrzymujemy nową populację. Ta nowa populacja odzwierciedla wzory zachowania bliższe wygrywającym w poprzednim pokoleniu, a mniej podobne do przegrywających. W każdym nowym pokoleniu osobniki ze stosunkowo dobrymi wynikami będą miały większe szanse przekazania dalej części swoich strategii, podczas gdy te ze stosunkowo słabymi wynikami będą miały na to mniejsze szanse.

1.2.3. Wyniki obliczeń

Po uruchomieniu swojego programu Axelrod uzyskał wspaniałe wyniki. Z zupełnie przypadkowego punktu startowego algorytm genetyczny utworzył po-

pulację, której średni członek był tak udany, jak najlepsze znane algorytmy heurystyczne. Niektóre wzory zachowań rozwinęły się u zdecydowanej większości osobników. Oto one:

1. Nie zatapiaj łodzi. Przedłużaj współpracę po trzech kolejnych współpracach (tzn. W po (WW)(WW)(WW)[1]).
2. Bądź podatny na prowokację. Zdradzaj, jeżeli inny gracz zdradził cię po dłuższej współpracy (tzn. Z po (WW)(WW)(WZ)).
3. Akceptuj przeprosiny. Kontynuuj współpracę, gdy partner ją wznowił (tzn. W po (WZ)(ZW)(WW)).
4. Zapominaj. Współpracuj, gdy współpraca została nawiązana po jej naruszeniu (tzn. W po (ZW)(WW)(WW)).
5. Trzymaj się rozwiązania rutynowego. Zdradzaj po trzech kolejnych zdradach (tzn. Z po (ZZ)(ZZ)(ZZ)).

Więcej szczegółów można znaleźć w [10].

1.3. Zadanie komiwojażera

W tym punkcie opiszemy, jak można użyć algorytmu genetycznego do rozwiązywania zadania komiwojażera. Zauważmy, że jest tu przedstawione tylko jedno z możliwych podejść. W rozdziale 10 omówimy także inne podejścia do tego zadania.

Mówiąc prosto, komiwojażer musi odwiedzić wszystkie miasta na swoim terytorium dokładnie raz, następnie zaś wrócić do punktu startu. Znając koszty przejazdu między miastami, jak powinien on zaplanować swoją trasę, aby uzyskać minimalny koszt całej podróży?

Zadanie komiwojażera jest zadaniem typu optymalizacji kombinatorycznej. Pojawia się ono w wielu zastosowaniach. Istnieje kilka algorytmów podziału i oszacowań, algorytmów przybliżonych i algorytmów heurystycznych do rozwiązywania tego zadania. W czasie ostatnich kilku lat było też kilka prób poszukiwania przybliżonych rozwiązań zadania komiwojażera za pomocą algorytmów genetycznych [154, s. 166-179]. Tutaj przedstawimy jeden z nich.

Po pierwsze, powinniśmy się zastanowić nad ważnym pytaniem o reprezentację chromosomu. Czy powinniśmy przyjąć chromosom jako wektor całkowitoliczbowy, czy raczej przekształcić go w wektor binarny? W poprzednich dwóch przykładach (optymalizacja funkcji i dylemat więźnia) chromosom był reprezentowany (w sposób mniej lub bardziej naturalny) jako wektor binarny. To pozwalało na zastosowanie binarnej mutacji i krzyżowania. Używając

[1] Ostatnie trzy ruchy są zapisywane w postaci trzech par (a_1b_1) (a_2b_2) (a_3b_3), gdzie literą a oznaczono ruchy rozważanego gracza (W oznacza współpracę, a Z zdradę), a literą b ruchy innego gracza.

tych operatorów, uzyskiwaliśmy dopuszczalne potomstwo, to znaczy potomstwo leżące w przestrzeni przeszukiwań. Taki przypadek nie zachodzi w zadaniu komiwojażera. Przy binarnej reprezentacji dla zadania komiwojażera z n miastami każde miasto musiałoby być zakodowane za pomocą $[\log_2 n]$ bitów, a więc chromosom musiałby być łańcuchem o $n[\log_2 n]$ bitach. Mutacja może spowodować, że trasa będzie przechodziła dwukrotnie przez to samo miasto. Ponadto, dla zadania z 20 miastami (gdzie potrzebujemy 5 bitów do reprezentacji miasta), niektóre 5-bitowe sekwencje (na przykład 10101) nie będą odpowiadały żadnemu miastu. Na podobne trudności napotkamy przy użyciu operatora krzyżowania. Widać więc jasno, że jeżeli użyjemy takich operatorów mutacji i krzyżowania, jakie wprowadziliśmy wcześniej, to będziemy potrzebować jakiegoś „algorytmu naprawy". Taki algorytm „naprawiałby" chromosom, umieszczając go z powrotem w przestrzeni przeszukiwań.

Wydaje się, że reprezentacja w postaci wektora całkowitoliczbowego jest lepsza. Zamiast używać algorytmów naprawy możemy włączyć wiedzę o zadaniu do operatorów. W ten sposób unikałyby one w „inteligentny" sposób tworzenia niedopuszczalnych osobników. Przyjmijmy więc reprezentację całkowitoliczbową. Wektor $\mathbf{v} = \langle i_1 i_2 ... i_n \rangle$ reprezentuje trasę: z i_1 do i_2 itd., z i_{n-1} do i_n i z powrotem do i_1 ($\mathbf{v}$ jest permutacją liczb $\langle 1\ 2\ ...\ n \rangle$).

Do rozpoczęcia obliczeń możemy albo użyć algorytmów heurystycznych (na przykład możemy przyjąć kilka rozwiązań z algorytmu zachłannego dla zadania komiwojażera, zaczynając od różnych miast), albo zacząć od populacji utworzonej z losowych permutacji liczb $\langle 1\ 2\ ...\ n \rangle$.

Ocena chromosomów jest oczywista. Znając koszty przejazdu między miastami, możemy łatwo policzyć całkowity koszt podróży.

W zadaniu komiwojażera szukamy najlepszego uporządkowania miast w trakcie podróży. Dosyć łatwo więc jest wprowadzić pewne operatory jednoargumentowe (typu jednoargumentowego), które umożliwiają poszukiwanie lepszego uporządkowania łańcucha. Jednak stosując tylko operatory jednoargumentowe, mamy małą nadzieję na znalezienie nawet dobrego uporządkowania (nie mówiąc o najlepszym) [160]. Do tego moc algorytmów genetycznych wynika z ukształtowanej wymiany informacji genetycznej przy krzyżowaniu wysoce dopasowanych osobników. Tak więc to, czego potrzebujemy, to operatory podobne do krzyżowania, które korzystałyby z istotnych podobieństw między chromosomami. Do tego celu użyjemy pewnego wariantu operatora OX [71], który tworzy potomstwo dla zadanej dwójki rodziców, wybierając podciąg trasy od jednego z rodziców i zachowując odpowiednią kolejność miast u drugiego z rodziców. Na przykład, jeżeli rodzicami są

$\langle 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12 \rangle$ i

$\langle 7\ 3\ 1\ 11\ 4\ 12\ 5\ 2\ 10\ 9\ 6\ 8 \rangle$

a wybrano podciąg

$\langle 4\ 5\ 6\ 7 \rangle$

to otrzymany potomek ma postać

$$\langle 1\ 11\ 12\ 4\ 5\ 6\ 7\ 2\ 10\ 9\ 8\ 3 \rangle$$

Tak jak żądano, potomek zawiera strukturalne uwarunkowania obu rodziców. Rola rodziców może być następnie odwrócona przy tworzeniu drugiego potomka.

Algorytm genetyczny z powyższym operatorem jest lepszy od losowego przeszukiwania, ale pozostawia jeszcze wiele możliwości poprawy. Typowe (średnie z 20 obliczeń) wyniki zastosowania algorytmu do zadania o 100 losowo wygenerowanych miastach dały (po 2000 pokoleniach) koszt całej podróży wyższy o 9,4% od optymalnego.

Do pełnej dyskusji zadania komiwojażera, zagadnień reprezentacji i zastosowanych operatorów genetycznych odsyłamy czytelnika do rozdz. 10.

1.4. Algorytmy wzrostu, symulowane wyżarzanie a algorytmy genetyczne

W tym punkcie przedstawimy trzy algorytmy: wzrostu, symulowanego wyżarzania i genetyczny dla prostego zadania optymalizacji. Ten przykład podkreśli wyjątkowość algorytmu genetycznego.

Przeszukiwaną przestrzenią jest tu zbiór łańcuchów binarnych $\mathbf{v}$ o długości 30. Funkcja celu, którą należy maksymalizować, ma postać

$$f(\mathbf{v}) = |11 \cdot one(\mathbf{v}) - 150|$$

gdzie funkcja $one(\mathbf{v})$ podaje liczbę jedynek w łańcuchu $\mathbf{v}$. Na przykład dla następujących trzech łańcuchów:

$$\mathbf{v}_1 = (110110101110101111111011011011)$$

$$\mathbf{v}_2 = (111000100100110111001010100011)$$

$$\mathbf{v}_3 = (000010000011001000000010001000)$$

otrzymamy

$$f(\mathbf{v}_1) = |11 \cdot 22 - 150| = 92$$

$$f(\mathbf{v}_2) = |11 \cdot 15 - 150| = 15$$

$$f(\mathbf{v}_3) = |11 \cdot 6 - 150| = 84$$

gdyż $one(\mathbf{v}_1) = 22$, $one(\mathbf{v}_2) = 15$, $one(\mathbf{v}_3) = 6$.

Funkcja f jest liniowa i jej optymalizacja nie jest niczym nadzwyczajnym. Przyjęliśmy ją tylko do zilustrowania omawianych trzech algorytmów. Jednak interesującą właściwością funkcji f jest to, że ma ona tylko jedno globalne maximum dla

$$\mathbf{v}_g = (111111111111111111111111111111)$$

równe $f(\mathbf{v}_g) = |11 \cdot 30 - 150| = 180$ oraz jedno lokalne maksimum dla

$$\mathbf{v}_l = (000000000000000000000000000000)$$

równe $f(\mathbf{v}_l) = |11 \cdot 0 - 150| = 150$.

Istnieje kilka wersji algorytmu wzrostu. Różnią się one sposobem wyboru nowego łańcucha do porównania z łańcuchem bieżącym. Jedna z wersji prostego (iteracyjnego) algorytmu wzrostu (iteracje MAX) jest przedstawiona na rys. 1.2 (algorytm największego wzrostu). Początkowo rozważa się w nim wszystkie 30 sąsiednie łańcuchy i do porównania z bieżącym łańcuchem $\mathbf{v}_c$ wybiera się ten łańcuch $\mathbf{v}_n$, dla którego wartość $f(\mathbf{v}_n)$ jest największa. Jeżeli zachodzi $f(\mathbf{v}_c) < f(\mathbf{v}_n)$, to nowy łańcuch staje się łańcuchem bieżącym. Jeżeli nie, to lokalnie nie da się poprawić rozwiązania i algorytm osiągnął (lokalne lub globalne) optimum (*local* = TRUE). W takim przypadku następna iteracja ($t \leftarrow t + 1$) algorytmu polega na przyjęciu nowego łańcucha bieżącego losowo.

```
procedure iteracyjny algorytm największego wzrostu
begin
  t ← 0
  repeat
    local ← FALSE
    wybierz losowo bieżący łańcuch v_c
    oceń v_c
    repeat
      wybierz 30 nowych łańcuchów z otoczenia v_c przez zamianę
        pojedynczych bitów v_c
      wybierz łańcuch v_n spośród nowych łańcuchów z największą
        wartością funkcji celu f
      if f(v_c) < f(v_n)
        then v_c ← v_n
        else local ← TRUE
    until local
    t ← t + 1
  until t = MAX
end
```

Rys. 1.2. Prosty algorytm największego wzrostu

Zauważmy, że sukces lub porażka pojedynczej iteracji powyższego algorytmu wzrostu (tzn. uzyskanie globalnego lub lokalnego optimum) są uzależnione od łańcucha początkowego (wybieranego losowo). Jest oczywiste, że jeżeli łańcuch początkowy ma 13 lub mniej jedynek, to algorytm zawsze dojdzie do lokalnego optimum (porażka). Powód tego jest taki, że funkcja celu dla łańcucha z 13 jedynkami jest równa 7 i jakakolwiek jednokrokowa zmiana w kierunku optimum globalnego, to znaczy zwiększenie liczby jedynek do 14, zmniejszy wartość funkcji celu do 4. Natomiast każde zmniejszenie liczby jedynek zwiększy wartość funkcji celu. Funkcja celu dla łańcucha z 12 jedynkami jest równa 18, z 11 jedynkami 29 itd. To kieruje poszukiwania w złym kierunku, do lokalnego maksimum.

Dla zadań z wieloma lokalnymi punktami optymalnymi szansa znalezienia globalnego optimum w pojedynczej iteracji jest mała.

Struktura procedury symulowanego wyżarzania jest przedstawiona na rys. 1.3. Wywołanie funkcji *random*[0, 1) powoduje otrzymanie liczby losowej z przedziału [0, 1). Blok (warunek zakończenia) sprawdza, czy osiągnięto „równowagę termiczną", to znaczy czy rozkład prawdopodobieństwa nowego wybranego łańcucha zbliżył się do rozkładu Boltzmanna [1].

Czasami [2] ta powtarzająca się pętla jest wykonywana tylko k razy (k jest dodatkowym parametrem metody).

Temperaturę T obniża się z kroku na krok ($g(T, t) < T$ dla wszystkich t). Algorytm kończy pracę przy pewnej małej wartości T: (kryterium stopu) spra-

```
procedure symulowane wyżarzanie
begin
  t ← 0
  początkowa temperatura T
  wybierz bieżący łańcuch v_c losowo
  oceń v_c
  repeat
    repeat
      wybierz nowy łańcuch v_n z otoczenia v_c przez zmianę pojedynczego
        bitu w v_n
      if f(v_c) < f(v_n)
        then v_c ← v_n
        else if random[0,1) < exp{(f(v_c) − f(v_n)/T}
          then v_c ← v_n
    until (warunek zakończenia)
    T ← g(T,t)
    t ← t + 1
  until (kryterium stopu)
end
```

Rys. 1.3. Symulowane wyżarzanie

wdza, czy system jest „zamrożony", co oznacza, że właściwie nie będą akceptowane więcej żadne zmiany.

Jak wspomniano wcześniej, algorytm symulowanego wyżarzania może uciec z lokalnego optimum. Rozważmy łańcuch

$$\mathbf{v}_4 = (111000000100110111001010100000)$$

z 12 jedynkami, dla którego funkcja celu jest równa $f(\mathbf{v}_4) = |\ 11 \cdot 12 - 50| = 18$. Z $\mathbf{v}_4$, jako łańcucha startowego, algorytm wzrostu (jak przedstawiono to wcześniej) osiągnie lokalne maksimum

$$\mathbf{v}_l = (000000000000000000000000000000)$$

gdyż funkcja celu dla każdego łańcucha z 13 jedynkami (krok w kierunku globalnego optimum) jest równa 7 (mniej od 18). Jednak algorytm symulowanego wyżarzania przyjmie łańcuch z 13 jedynkami jako nowy łańcuch bieżący z prawdopodobieństwem

$$p = \exp\{(f(\mathbf{v}_n) - f(\mathbf{v}_c))/T\} = \exp\{(7 - 18)/T\}$$

co, na przykład dla temperatury $T = 20$, daje

$$p = e^{-11/20} = 0{,}57695$$

to oznacza, że szansa zaakceptowania jest większa od 50%.

Algorytmy genetyczne, zgodnie z dyskusją z p. 1.1, utrzymują populację łańcuchów. Dwa stosunkowo słabe łańcuchy

$$\mathbf{v}_5 = (111110000000110111001110100000) \quad \text{i}$$

$$\mathbf{v}_6 = (000000000001101110010101111111)$$

dla których funkcja celu jest równa 16, mogą utworzyć znacznie lepszego potomka (jeżeli punkt cięcia wypadnie gdziekolwiek między 5 a 12 elementem)

$$\mathbf{v}_7 = (111110000001101110010101111111)$$

Funkcja celu dla tego potomka jest

$$f(\mathbf{v}_7) = |11 \cdot 19 - 150| = 59$$

Dla zapoznania się ze szczegółową dyskusją tego i innych algorytmów (różne warianty algorytmów wzrostu, poszukiwania genetycznego i symulowanego wyżarzania) testowanych na kilku funkcjach o różnym charakterze odsyłamy czytelnika do [4]. Jest także możliwe zbudowanie hybryd, łączących kilka metod (łącznie z algorytmami genetycznymi) – patrz na przykład dynamiczną metodę wzrostu [92]. Kończymy ten punkt cytatem ze śmiesznej przesyłki przedstawionej w *Internecie* (comp.ai.neural-nets [377]). Podaje się w niej ciekawe porównanie metod wzrostu, symulowanego wyżarzania i algorytmów genetycznych:

Zauważmy, że we wszystkich omówionych dotąd metodach [wzrostu] kangur może liczyć najwyżej na znalezienie się na szczycie góry w pobliżu miejsca, gdzie zaczynał. Nie ma żadnej gwarancji, że będzie to Everest czy bardzo wysoka góra. Stosuje się różne metody, aby spróbować znaleźć rzeczywiste optimum globalne.

W symulowanym wyżarzaniu kangur jest pijany i skacze wkoło losowo przez dłuższy czas. Jednak stopniowo trzeźwieje i stara się wskoczyć na wzniesienie.

W algorytmach genetycznych jest spora liczba kangurów, które są zrzucone na spadochronach w Himalaje (jeżeli pilot się nie zagubił) w losowych miejscach. Kangury te nie wiedzą, że mają szukać szczytu Mont Everest. Jednak co kilka lat zabija się kangury na niskich wysokościach z nadzieją, że te, które zostały, będą owocne i rozmnożą się.

1.5. Wnioski

Trzy przykłady algorytmów genetycznych do optymalizacji funkcji, dylematu więźnia i zadania komiwojażera wskazują na szerokie możliwości zastosowań algorytmów genetycznych. Jednocześnie jednak zarysowały się potencjalne trudności. Kwestia reprezentacji w zadaniu komiwojażera nie była oczywista. Nowy operator (krzyżowanie OX) nie był wcale trywialny. Jakie dalsze kłopoty możemy jeszcze mieć w trudnych zadaniach? W pierwszym i trzecim przykładzie (optymalizacji funkcji i zadaniu komiwojażera) funkcja oceny była jasno określona. W drugim przykładzie (dylemat więźnia) prosta symulacja umożliwiła nam ocenę chromosomu (badamy efektywność każdego gracza: gracz stosuje strategię zdefiniowaną w swoim chromosomie do gry z innymi i jego końcowy wynik jest średnią z wyników uzyskanych we wszystkich grach). Jak należy postąpić, gdy funkcja oceny nie jest jasno określona? Na przykład, zadanie spełniania logicznego ma naturalną reprezentację (i-ty bit jest zapisem prawdziwości i-tej zmiennej binarnej), jednak wybór funkcji oceny jest daleki od oczywistego [90].

Pierwszy przykład optymalizacji funkcji bez ograniczeń pozwala użyć wygodnej reprezentacji, w której każdy łańcuch binarny odpowiada wartości z dziedziny funkcji $[-1, 2]$. Oznacza to, że jakakolwiek mutacja lub krzyżowanie prowadzi do prawidłowego potomka. To samo dotyczy drugiego przykładu – jakakolwiek kombinacja bitów jest prawidłową strategią. W trzecim zadaniu występuje proste ograniczenie – każde miasto może się znaleźć tylko raz na prawidłowej trasie. To powoduje pewne kłopoty: przyjęliśmy wektory całkowitoliczbowe (zamiast reprezentacji binarnej) i zmodyfikowaliśmy operację krzyżowania. Ale jak należy postępować ogólnie w zadaniu z ograniczeniami? Jakie mamy możliwości? Odpowiedzi nie są łatwe. Będziemy je rozważać dalej w książce.

2

Algorytmy genetyczne: jak one działają?

Wszystko ma swoją godzinę,
jest czas na wszelką sprawę pod niebem:
jest czas rodzenia i czas umierania,
czas sadzenia i czas wyrywania tego, co zasadzono
czas zabijania i czas leczenia,
czas burzenia i czas budowania.

Pismo Święte Starego i Nowego Testamentu, Księga Koheleta, 3[1)]

W tym rozdziale przedstawimy działanie algorytmu genetycznego dla prostego zadania optymalizacji. Zaczniemy od kilku ogólnych stwierdzeń, po czym nastąpi szczegółowy przykład.

Zauważmy przede wszystkim, że bez straty ogólności możemy rozważyć tylko zadanie maksymalizacji. Jeżeli nasze zadanie optymalizacji polega na minimalizacji funkcji f, to jest ono równoważne maksymalizacji funkcji g, przy czym $g = -f$, czyli

$$\min f(x) = \max g(x) = \max\{-f(x)\}$$

Dodatkowo możemy przyjąć, że funkcja f jest dodatnia w swojej dziedzinie. Jeżeli tak nie jest, to możemy dodać[2)] pewną dodatnią stałą C, gdyż

$$\max g(x) = \max\{g(x) + C\}$$

Przypuśćmy teraz, że chcemy maksymalizować funkcję k zmiennych $f(x_1, \ldots, x_k)$: $R^k \to R$. Przypuśćmy także, że każda zmienna x_i może przybierać wartości z przedziału $D_i = [a_i, b_i] \subseteq R$ i $f(x_1, \ldots .x_k) > 0$ dla wszystkich $x_i \in D_i$. Chcemy optymalizować funkcję f z pewną żądaną dokładnością. Przypuśćmy, że jest to sześć najbardziej znaczących cyfr dziesiętnych w każdej zmiennej.

Aby uzyskać taką dokładność, każdy przedział D_i należy podzielić na $(b_i - a_i) \cdot 10^6$ równych podprzedziałów. Oznaczmy przez m_i najmniejszą liczbę

[1)] Przekład zespołowy pod redakcją Michała Petera, Księgarnia Św. Wojciecha, 1992 (przyp. tłum.).

[2)] Oczywiście dotyczy to funkcji ograniczonych z dołu. W szczególności rozważane tu często funkcje, których dziedzina składa się ze skończonej liczby punktów, są zawsze ograniczone od dołu przez najmniejszą wartość funkcji w tych punktach (przyp. tłum.).

całkowitą taką, że $(b_i - a_i) \cdot 10^6 \leqslant 2^{m_i} - 1$. Wtedy reprezentacja, w której każda zmienna x_i jest zakodowana jako łańcuch binarny o długości m_i, w oczywisty sposób będzie spełniała wymagania dokładności. Wartość tego łańcucha można wyrazić w postaci dziesiętnej za pomocą wzoru

$$x_i = a_i + decimal(1001...001_2) \cdot (b_i - a_i)/2^{m_i} - 1$$

gdzie *decimal(łańcuch2)* jest równy dziesiętnej wartości łańcucha binarnego.

Każdy chromosom (jako potencjalne rozwiązanie) jest reprezentowany przez łańcuch o długości $m = \sum_{i=1}^{k} m_i$. Pierwsze m_1 bitów odpowiada zmiennej z przedziału $[a_1, b_1]$, druga grupa m_2 bitów odpowiada zmiennej z przedziału $[a_2, b_2]$, i tak dalej, aż do ostatniej grupy m_k bitów, które odpowiadają zmiennej z przedziału $[a_k, b_k]$.

W celu ustalenia populacji początkowej możemy po prostu wygenerować liczbę *pop_size* chromosomów losowo, bit po bicie. Jeżeli jednak dysponujemy pewną wiedzą o rozkładzie możliwych optimów, możemy użyć tej informacji do odpowiedniego rozmieszczenia początkowych (potencjalnych) rozwiązań.

Reszta algorytmu jest oczywista. W każdym pokoleniu oceniamy wszystkie chromosomy (używając funkcji f ograniczonej do zakodowanych sekwencji zmiennych), wybieramy nową populację zgodnie z rozkładem prawdopodobieństwa określonym na wartościach dopasowań i zmieniamy chromosomy w nowej populacji za pomocą operatorów mutacji i krzyżowania. Po pewnej liczbie pokoleń, kiedy nie obserwujemy już poprawy, najlepsze chromosomy reprezentują (mamy nadzieję, że globalne) rozwiązanie optymalne. Często zatrzymujemy algorytm po ustalonej liczbie iteracji, w zależności od szybkości i pamięci posiadanego komputera.

W procesie selekcji (wybór nowej populacji zgodnie z rozkładem prawdopodobieństwa określonym na wartościach dopasowań) używa się ruletki o wielkości pól zgodnej z wartościami dopasowań. Ruletkę taką konstruuje się następująco (poniżej zakładamy, że wartości dopasowania są dodatnie; jeżeli tak nie jest, należy odpowiednio przeskalować wartości – jest to omawiane w rozdz. 4):

- oblicz wartość dopasowania $eval(\mathbf{v}_i)$ dla każdego chromosomu $\mathbf{v}_i$ $(i = 1, ..., pop_size)$,
- oblicz całkowite dopasowanie populacji $F = \sum_{i=1}^{pop_size} eval(\mathbf{v}_i)$,
- oblicz prawdopodobieństwo wyboru p_i każdego chromosomu $\mathbf{v}_i$ $(i = 1, ..., pop_size)$ $p_i = eval(\mathbf{v}_i)/F$,
- oblicz dystrybuantę q_i dla każdego chromosomu $\mathbf{v}_i$ $(i = 1, ..., pop_size)$: $q_i = \sum_{j=1}^{i} p_j$.

Proces selekcji jest oparty na obrocie ruletką *pop_size* razy i wyborze za każdym razem jednego chromosomu do nowej populacji w następujący sposób:

- wygeneruj (zmiennopozycyjną) liczbę przypadkową r z zakresu [0, 1],
- jeżeli $r < q_1$, to wybierz chromosom $\mathbf{v}_1$; jeżeli nie, to wybierz chromosom $\mathbf{v}_i$ ($2 \leqslant i \leqslant pop_size$), dla którego zachodzi $q_{i-1} < r \leqslant q_i$.

Oczywiście pewne chromosomy będą wybrane więcej niż raz. Jest to zgodne z twierdzeniem o schematach (patrz następny rozdział), które mówi, że z najlepszych chromosomów powstaje więcej kopii, ze średnich tyle samo, a najgorsze wymierają.

Jesteśmy teraz gotowi zastosować operator mieszania, to znaczy krzyżowanie, do osobników z nowej populacji. Jak wspomniano wcześniej, jednym z parametrów systemu genetycznego jest prawdopodobieństwo krzyżowania p_c. Umożliwia nam ono obliczenie oczekiwanej liczby chromosomów $p_c \cdot pop_size$, które ulegną operacji krzyżowania. Postępujemy w następujący sposób:

Dla każdego chromosomu z (nowej) populacji:

- wygeneruj (zmiennopozycyjną) liczbę losową r z zakresu [0, 1],
- jeżeli $r < p_c$, to wybierz rozpatrywany chromosom do krzyżowania.

Następnie dobieramy wybrane chromosomy losowo w pary. Dla każdej pary chromosomów generujemy losową liczbę całkowitą *pos* z zakresu $[1, m-1]$ (m jest całkowitą długością – liczbą bitów – chromosomu). Liczba *pos* określa punkt cięcia chromosomu do wymiany. Tak więc dwa chromosomy

$$(b_1 b_2 \ldots b_{pos} b_{pos+1} \ldots b_m) \quad \text{i}$$

$$(c_1 c_2 \ldots c_{pos} c_{pos+1} \ldots c_m)$$

są zamieniane na parę potomków

$$(b_1 b_2 \ldots b_{pos} c_{pos+1} \ldots c_m)$$

$$(c_1 c_2 \ldots c_{pos} b_{pos+1} \ldots b_m)$$

Następna operacja, mutacja, jest wykonywana na bitach. Na podstawie parametru systemu genetycznego, prawdopodobieństwa mutacji p_m, możemy obliczyć oczekiwaną liczbę zmutowanych bitów $p_m \cdot m \cdot pop_size$. Każdy bit (we wszystkich chromosomach i w całej populacji) ma równe szanse na mutację, to znaczy zmiany z 0 na 1 lub odwrotnie. Postępujemy więc następująco:

Dla każdego chromosomu bieżącej (po krzyżowaniu) populacji i dla każdego bitu w chromosomie:

- wygeneruj (zmiennopozycyjną) liczbę losową r z zakresu [0, 1],
- jeżeli $r < p_m$, to zmutuj bit.

Po selekcji, krzyżowaniu i mutacji nowa populacja jest gotowa do kolejnej oceny. Ta ocena będzie użyta do określenia rozkładu prawdopodobieństwa (dla następnego procesu selekcji), to znaczy do konstruowania ruletki o wielkościach pól zgodnych z nowymi wartościami dopasowania. Reszta oceny

jest cyklicznym powtórzeniem wcześniejszych kroków (patrz rys. 0.1 we wprowadzeniu).

Przedstawimy teraz całe postępowanie na przykładzie. Uruchamiamy algorytm genetyczny do optymalizacji funkcji. Zakładamy, że liczebność populacji $pop_size = 20$ i że prawdopodobieństwa operatorów genetycznych wynoszą $p_c = 0{,}25$ i $p_m = 0{,}01$.

Przypuśćmy także, że maksymalizujemy następującą funkcję:

$$f(x_1, x_2) = 21{,}5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

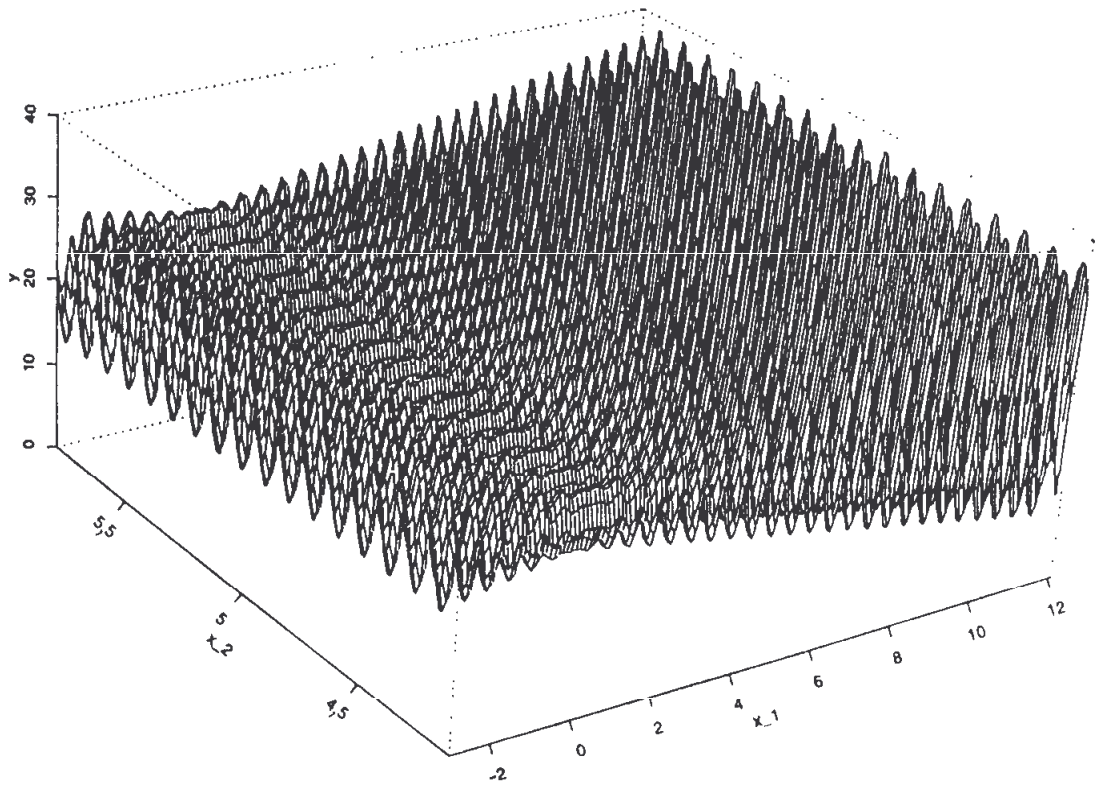
przy ograniczeniach $-3{,}0 \leqslant x_1 \leqslant 12{,}1$ i $4{,}1 \leqslant x_2 \leqslant 5{,}8$. Wykres funkcji f jest przedstawiony na rys. 2.1.

Przypuśćmy dalej, że żądana dokładność wynosi cztery najbardziej znaczące cyfry dla każdej zmiennej. Przedział zmienności x_1 ma długość 15,1. Z żądanej dokładności wynika, że przedział $[-3{,}0, 12{,}1]$ należy podzielić na przynajmniej $15{,}1 \cdot 10\,000$ równych podprzedziałów. Oznacza to, że początkowa część chromosomu musi zawierać 18 bitów, gdyż

$$2^{17} < 151000 \leqslant 2^{18}$$

Przedział zmienności x_2 ma długość 1,7. Z żądanej dokładności wynika, że przedział $[4{,}1, 5{,}8]$ należy podzielić na przynajmniej $1{,}7 \cdot 10\,000$ równych podprzedziałów. Oznacza to, że końcowa część chromosomu musi zawierać 15 bitów, gdyż

$$2^{14} < 17\,000 \leqslant 2^{15}$$



Rys. 2.1. Wykres funkcji $f(x_1, x_2) = 21{,}5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$

Całkowita długość chromosomu (wektora rozwiązań) wynosi więc $m = 18 + 15 = 33$ bitów. W pierwszych 18 bitach jest zakodowane x_1, a w pozostałych 15 bitach $(19 - 33)x_2$. Rozważmy przykładowy chromosom

(010001001011010000111110010100010)

Pierwsze 18 bitów

010001001011010000

reprezentuje

$$x_1 = -3{,}0 + decimal(010001001011010000_2)\cdot(12{,}1 - (-3{,}0))/(2^{18} - 1) =$$
$$= -3{,}0 + 70352\cdot 15{,}1/262143 = -3{,}0 + 4{,}052426 = 1{,}052426$$

Następne 15 bitów

111110010100010

reprezentuje

$$x_2 = 4{,}1 + decimal(111110010100010_2)\cdot(5{,}8 - 4{,}1)/(2^{15} - 1) =$$
$$= 4{,}1 + 31906\cdot 1{,}7/32767 = 4{,}1 + 1{,}655330 = 5{,}755330$$

Tak więc chromosom

(010001001011010000111110010100010)

odpowiada $(x_2, x_2) = (1{,}052426, 5{,}755330)$. Wartość dopasowania chromosomu wynosi $f(1{,}052426, 5{,}755330) = 20{,}252640$.

Do optymalizacji funkcji f za pomocą algorytmu genetycznego utworzymy populację o liczbie $pop_size = 20$ chromosomów. Wszystkie 33 bity w każdym chromosomie są na początku wybierane losowo.

Przypuśćmy, że po procesie początkowego wyboru chromosomów otrzymaliśmy następującą populację:

$\mathbf{v}_1$ = (100110100000001111111010011011111)
$\mathbf{v}_2$ = (111000100100110111001010100011010)
$\mathbf{v}_3$ = (000010000011001000001010111011101)
$\mathbf{v}_4$ = (100011000101101001111000001110010)
$\mathbf{v}_5$ = (000111011001010011010111111000101)
$\mathbf{v}_6$ = (000101000010010101001010111111011)
$\mathbf{v}_7$ = (001000100000110101111011011111011)
$\mathbf{v}_8$ = (100001100001110100010110101100111)
$\mathbf{v}_9$ = (010000000101100010110000001111100)
$\mathbf{v}_{10}$ = (000001111000110000011010000111011)
$\mathbf{v}_{11}$ = (011001111110110101100001101111000)
$\mathbf{v}_{12}$ = (110100010111101101000101010000000)
$\mathbf{v}_{13}$ = (111011111010001000110000001000110)
$\mathbf{v}_{14}$ = (010010011000001010100111100101001)

$\mathbf{v}_{15} = (111011101101110000100011111011110)$
$\mathbf{v}_{16} = (110011110000011111100001101001011)$
$\mathbf{v}_{17} = (011010111111001111010001101111101)$
$\mathbf{v}_{18} = (011101000000001110100111110101101)$
$\mathbf{v}_{19} = (000101010011111111110000110001100)$
$\mathbf{v}_{20} = (101110010110011110011000101111110)$

W kroku oceny dekodujemy każdy chromosom i obliczamy funkcję oceniającą dla właśnie rozkodowanych wartości (x_1, x_2). Dostajemy:

$eval(\mathbf{v}_1) = f(6{,}084492,\ 5{,}652242) = 26{,}019600$
$eval(\mathbf{v}_2) = f(10{,}348434,\ 4{,}380264) = 7{,}580015$
$eval(\mathbf{v}_3) = f(-2{,}516603,\ 4{,}390381) = 19{,}526329$
$eval(\mathbf{v}_4) = f(5{,}278638,\ 5{,}593460) = 17{,}406725$
$eval(\mathbf{v}_5) = f(-1{,}255173,\ 4{,}734458) = 25{,}341160$
$eval(\mathbf{v}_6) = f(-1{,}811725,\ 4{,}391937) = 18{,}100417$
$eval(\mathbf{v}_7) = f(-0{,}991471,\ 5{,}680258) = 16{,}020812$
$eval(\mathbf{v}_8) = f(4{,}910618,\ 4{,}703018) = 17{,}959701$
$eval(\mathbf{v}_9) = f(0{,}795406,\ 5{,}381472) = 16{,}127799$
$eval(\mathbf{v}_{10}) = f(-2{,}554851,\ 4{,}793707) = 21{,}278435$
$eval(\mathbf{v}_{11}) = f(3{,}130078,\ 4{,}996097) = 23{,}410669$
$eval(\mathbf{v}_{12}) = f(9{,}356179,\ 4{,}239457) = 15{,}011619$
$eval(\mathbf{v}_{13}) = f(11{,}134646,\ 5{,}378671) = 27{,}316702$
$eval(\mathbf{v}_{14}) = f(1{,}335944,\ 5{,}151378) = 19{,}876294$
$eval(\mathbf{v}_{15}) = f(11{,}089025,\ 5{,}054515) = 30{,}060205$
$eval(\mathbf{v}_{16}) = f(9{,}211598,\ 4{,}993762) = 23{,}867227$
$eval(\mathbf{v}_{17}) = f(3{,}367514,\ 4{,}571343) = 13{,}696165$
$eval(\mathbf{v}_{18}) = f(3{,}843020,\ 5{,}158226) = 15{,}414128$
$eval(\mathbf{v}_{19}) = f(-1{,}746635,\ 5{,}395584) = 20{,}095903$
$eval(\mathbf{v}_{20}) = f(7{,}935998,\ 4{,}757338) = 13{,}666916$

Jak widać, chromosom $\mathbf{v}_{15}$ jest najmocniejszy, a chromosom $\mathbf{v}_2$ najsłabszy.

Teraz konstruujemy ruletkę dla procesu selekcji. Pełne dopasowanie populacji wynosi

$$F = \sum_{i=1}^{20} eval(\mathbf{v}_i) = 387{,}776822$$

Prawdopodobieństwa wyboru p_i dla każdego chromosomu $\mathbf{v}_i$ $(i = 1, ..., 20)$ są równe:

$p_1 = eval(\mathbf{v}_1)/F = 0{,}067099$
$p_2 = eval(\mathbf{v}_2)/F = 0{,}019547$
$p_3 = eval(\mathbf{v}_3)/F = 0{,}050355$
$p_4 = eval(\mathbf{v}_4)/F = 0{,}044889$
$p_5 = eval(\mathbf{v}_5)/F = 0{,}065350$
$p_6 = eval(\mathbf{v}_6)/F = 0{,}046677$

$p_7 = eval(\mathbf{v}_7)/F = 0{,}041315$
$p_8 = eval(\mathbf{v}_8)/F = 0{,}046315$
$p_9 = eval(\mathbf{v}_9)/F = 0{,}041590$
$p_{10} = eval(\mathbf{v}_{10})/F = 0{,}054873$
$p_{11} = eval(\mathbf{v}_{11})/F = 0{,}060372$
$p_{12} = eval(\mathbf{v}_{12})/F = 0{,}038712$
$p_{13} = eval(\mathbf{v}_{13})/F = 0{,}070444$
$p_{14} = eval(\mathbf{v}_{14})/F = 0{,}051257$
$p_{15} = eval(\mathbf{v}_{15})/F = 0{,}077519$
$p_{16} = eval(\mathbf{v}_{16})/F = 0{,}061549$
$p_{17} = eval(\mathbf{v}_{17})/F = 0{,}035320$
$p_{18} = eval(\mathbf{v}_{18})/F = 0{,}039750$
$p_{19} = eval(\mathbf{v}_{19})/F = 0{,}051823$
$p_{20} = eval(\mathbf{v}_{20})/F = 0{,}035244$

Wartości dystrybuanty q_i dla każdego chromosomu $\mathbf{v}_i$ ($i = 1, ..., 20$) wynoszą:

$q_1 = 0{,}067099$
$q_2 = 0{,}086647$
$q_3 = 0{,}137001$
$q_4 = 0{,}181890$
$q_5 = 0{,}247240$
$q_6 = 0{,}293917$
$q_7 = 0{,}335232$
$q_8 = 0{,}381546$
$q_9 = 0{,}423137$
$q_{10} = 0{,}478009$
$q_{11} = 0{,}538381$
$q_{12} = 0{,}577093$
$q_{13} = 0{,}647537$
$q_{14} = 0{,}698794$
$q_{15} = 0{,}776314$
$q_{16} = 0{,}837863$
$q_{17} = 0{,}873182$
$q_{18} = 0{,}912932$
$q_{19} = 0{,}964756$
$q_{20} = 1{,}000000$

Jesteśmy teraz gotowi zakręcić ruletką 20 razy. Za każdym razem wybieramy jeden chromosom do nowej populacji. Przypuśćmy, że (losowy) ciąg 20 liczb z zakresu [0, 1] ma postać:

0,513870
0,175741
0,308652
0,534534
0,947628
0,171736
0,702231
0,226431
0,494773
0,424720
0,703899
0,389647
0,277226
0,368071
0,983437
0,005398
0,765682
0,646473
0,767139
0,780237

Pierwsza liczba $r = 0{,}513870$ jest większa od q_{10} i mniejsza od q_{11}, co oznacza, że chromosom $\mathbf{v}_{11}$ jest wybrany do nowej populacji. Druga liczba $r = 0{,}175741$ jest większa od q_3 i mniejsza od q_4, co oznacza, że $\mathbf{v}_4$ jest wybrane do nowej populacji itd.

W rezultacie nowa populacja składa się z następujących chromosomów:

$\mathbf{v}'_1 = (011001111110110101100001101111000)$ $(\mathbf{v}_{11})$
$\mathbf{v}'_2 = (100011000101101001111000001110010)$ $(\mathbf{v}_4)$

$\mathbf{v}'_3 = (001000100000110101111011011111011)$ $(\mathbf{v}_7)$
$\mathbf{v}'_4 = (011001111110110101100001101111000)$ $(\mathbf{v}_{11})$
$\mathbf{v}'_5 = (000101010011111111110000110001100)$ $(\mathbf{v}_{19})$
$\mathbf{v}'_6 = (100011000101101001111000001110010)$ $(\mathbf{v}_4)$
$\mathbf{v}'_7 = (111011101101110000100011111011110)$ $(\mathbf{v}_{15})$
$\mathbf{v}'_8 = (000111011001010011010111111000101)$ $(\mathbf{v}_5)$
$\mathbf{v}'_9 = (011001111110110101100001101111000)$ $(\mathbf{v}_{11})$
$\mathbf{v}'_{10} = (000010000011001000001010111011101)$ $(\mathbf{v}_3)$
$\mathbf{v}'_{11} = (111011101101110000100011111011110)$ $(\mathbf{v}_{15})$
$\mathbf{v}'_{12} = (010000000101100010110000001111100)$ $(\mathbf{v}_9)$
$\mathbf{v}'_{13} = (000101000010010101001010111111011)$ $(\mathbf{v}_6)$
$\mathbf{v}'_{14} = (100001100001110100010110101100111)$ $(\mathbf{v}_8)$
$\mathbf{v}'_{15} = (101110010110011110011000101111110)$ $(\mathbf{v}_{20})$
$\mathbf{v}'_{16} = (100110100000001111111010011011111)$ $(\mathbf{v}_1)$
$\mathbf{v}'_{17} = (000001111000110000011010000111011)$ $(\mathbf{v}_{10})$
$\mathbf{v}'_{18} = (111011111010001000110000001000110)$ $(\mathbf{v}_{13})$
$\mathbf{v}'_{19} = (111011101101110000100011111011110)$ $(\mathbf{v}_{15})$
$\mathbf{v}'_{20} = (110011110000011111100001101001011)$ $(\mathbf{v}_{16})$

Jesteśmy teraz gotowi do użycia operatora rekombinacji, krzyżowania, dla osobników nowego pokolenia (wektorów $\mathbf{v}'_i$). Prawdopodobieństwo krzyżowania $p_c = 0{,}25$, więc spodziewamy się, że (średnio) 25% chromosomów (tzn. 5 z 20) ulegnie krzyżowaniu. Postępujemy następująco. Dla każdego chromosomu z nowej populacji generujemy liczbę losową r z zakresu [0, 1]. Jeżeli $r < 0{,}25$, to wybieramy rozważany chromosom do krzyżowania.

Załóżmy, że ciąg liczb losowych jest następujący:

0,822951	0,031523
0,151932	0,869921
0,625477	0,166525
0,314685	0,674520
0,346901	0,758400
0,917204	0,581893
0,519760	0,389248
0,401154	0,200232
0,606758	0,355635
0,785402	0,826927

Oznacza to, że do krzyżowania są wybrane chromosomy $\mathbf{v}'_2$, $\mathbf{v}'_{11}$, $\mathbf{v}'_{13}$ i $\mathbf{v}'_{18}$. (Mieliśmy szczęście. Liczba wybranych chromosomów jest parzysta, więc możemy je łatwo połączyć. Gdyby liczba wybranych chromosomów była nieparzysta, musielibyśmy albo dodać, albo odjąć jeden chromosom, oczywiście także losowo). Teraz łączymy wybrane chromosomy losowo. Powiedzmy, że zostały połączone dwa pierwsze (tzn. $\mathbf{v}'_2$ i $\mathbf{v}'_{11}$) i dwa ostatnie (tzn. $\mathbf{v}'_{13}$ i $\mathbf{v}'_{18}$). Dla

każdej z tych dwóch par generujemy losową liczbę całkowitą *pos* z zakresu [1, 33] (33 jest całkowitą długością chromosomu, czyli liczbą zawartych w nim bitów). Liczba *pos* określa pozycję cięcia w chromosomach. Dla *pos* = 9 pierwsza para chromosomów wygląda więc następująco:

$$\mathbf{v}'_2 = (100011000|101101001111000001110010)$$
$$\mathbf{v}'_{11} = (111011101|101110000100011111011110)$$

Po odcięciu po 9. bicie i zamianie uzyskujemy parę potomków

$$\mathbf{v}''_2 = (100011000|101110000100011111011110)$$
$$\mathbf{v}''_{11} = (111011101|101101001111000001110010)$$

Dla drugiej pary chromosomów *pos* = 20

$$\mathbf{v}'_{13} = (000101000010010101 00|1010111111011)$$
$$\mathbf{v}'_{18} = (111011111010001000 11|0000001000110)$$

Są one zamienione na parę ich potomków

$$\mathbf{v}''_{13} = (000101000010010101 00|0000001000110)$$
$$\mathbf{v}''_{18} = (111011111010001000 11|1010111111011)$$

Obecna wersja populacji wygląda więc następująco:

$$\mathbf{v}'_1 = (011001111110110101100001101111000)$$
$$\mathbf{v}''_2 = (100011000101110000100011111011110)$$
$$\mathbf{v}'_3 = (001000100000110101111011011111011)$$
$$\mathbf{v}'_4 = (011001111110110101100001101111000)$$
$$\mathbf{v}'_5 = (000101010011111111110000110001100)$$
$$\mathbf{v}'_6 = (100011000101101001111000001110010)$$
$$\mathbf{v}'_7 = (111011101101110000100011111011110)$$
$$\mathbf{v}'_8 = (000111011001010011010111111000101)$$
$$\mathbf{v}'_9 = (011001111110110101100001101111000)$$
$$\mathbf{v}'_{10} = (000010000011001000001010111011101)$$
$$\mathbf{v}''_{11} = (111011101101101001111000001110010)$$
$$\mathbf{v}'_{12} = (010000000101100010110000001111100)$$
$$\mathbf{v}''_{13} = (000101000010010101000000001000110)$$
$$\mathbf{v}'_{14} = (100001100001110100010110101100111)$$
$$\mathbf{v}'_{15} = (101110010110011110011000101111110)$$
$$\mathbf{v}'_{16} = (100110100000001111111010011011111)$$
$$\mathbf{v}'_{17} = (000001111000110000011010000111011)$$
$$\mathbf{v}''_{18} = (111011111010001000111010111111011)$$
$$\mathbf{v}'_{19} = (111011101101110000100011111011110)$$
$$\mathbf{v}'_{20} = (110011110000011111100001101001011)$$

Następny operator, mutacja, odbywa się bezpośrednio na bitach. Prawdopodobieństwo mutacji wynosi $p_m = 0{,}01$, więc spodziewamy się, że (średnio) 1% bitów ulegnie mutacji. W całej populacji jest $m \times pop_size = 33 \times 20 = 660$ bitów. Oczekujemy więc (średnio) 6,6 mutacji na pokolenie. Każdy bit ma równe szanse na zmutowanie, więc dla każdego bitu w populacji generujemy liczbę losową r z zakresu [0, 1]. Jeżeli $r < 0{,}01$, to zmieniamy bit.

Oznacza to, że musimy wygenerować 660 liczb przypadkowych. W otrzymanej z generatora próbie 5 z nich bylo mniejszych od 0,01. Numery bitów i wartości liczb losowych są podane poniżej.

Pozycja bitu	Liczba przypadkowa
112	0,000213
349	0,009945
418	0,008809
429	0,005425
602	0,002836

Następująca tablica podaje przeliczania numerów bitów na numery chromosomów i pozycje w nich bitów.

Pozycja bitu	Numer chromosomu	Numer bitu w chromosomie
112	4	13
349	11	19
418	13	22
429	13	33
602	19	8

Oznacza to, że 4 chromosomy są zmienione przez operator mutacji, przy czym jeden z nich (trzynasty) ma zmienione dwa bity.

Końcowa populacja jest przedstawiona poniżej. Zmutowane bity są w niej wytłuszczone. W oznaczeniach zmodyfikowanych chromosomów opuszczamy apostrofy i wypisujemy populację jako nowe wektory $\mathbf{v}_i$:

$\mathbf{v}_1 = (011001111110110101100001101111000)$
$\mathbf{v}_2 = (100011000101110000100011111011110)$
$\mathbf{v}_3 = (001000100000110101111011011111011)$
$\mathbf{v}_4 = (011001111110\mathbf{0}101011000001101111000)$
$\mathbf{v}_5 = (000101010011111111110000110001100)$
$\mathbf{v}_6 = (100011000101101001111000001110010)$
$\mathbf{v}_7 = (111011101101110000100011111011110)$
$\mathbf{v}_8 = (000111011001010011010111111000101)$
$\mathbf{v}_9 = (011001111110110101100001101111000)$
$\mathbf{v}_{10} = (000010000011001000001010111011101)$
$\mathbf{v}_{11} = (111011101101101001\mathbf{0}11000001110010)$
$\mathbf{v}_{12} = (010000000101100010110000001111100)$

$\mathbf{v}_{13} = (000101000010010101000100001000111)$
$\mathbf{v}_{14} = (100001100001110100010110101100111)$
$\mathbf{v}_{15} = (101110010110011110011000101111110)$
$\mathbf{v}_{16} = (100110100000001111111010011011111)$
$\mathbf{v}_{17} = (000001111000110000011010000111011)$
$\mathbf{v}_{18} = (111011111010001000111010111111011)$
$\mathbf{v}_{19} = (111011100101110000100011111011110)$
$\mathbf{v}_{20} = (110011110000011111100001101001011)$

Zakończyliśmy w ten sposób jedną iterację (tzn. jedno pokolenie) pętli **while** w procedurze genetycznej (rys. 0.1 we wprowadzeniu). Ciekawe jest przestudiowanie wyników oceny nowej populacji. Podczas fazy oceny dekodujemy każdy chromosom i obliczamy wartość funkcji dopasowania we właśnie rozkodowanym punkcie (x_1, x_2). Otrzymujemy:

$eval(\mathbf{v}_1) = f(3{,}130078,\ 4{,}996097) = 23{,}410669$
$eval(\mathbf{v}_2) = f(5{,}279042,\ 5{,}054515) = 18{,}201083$
$eval(\mathbf{v}_3) = f(-0{,}991471,\ 5{,}680258) = 16{,}020812$
$eval(\mathbf{v}_4) = f(3{,}128235,\ 4{,}996097) = 23{,}412613$
$eval(\mathbf{v}_5) = f(-1{,}746635,\ 5{,}395584) = 20{,}095903$
$eval(\mathbf{v}_6) = f(5{,}278638,\ 5{,}593460) = 17{,}406725$
$eval(\mathbf{v}_7) = f(11{,}089025,\ 5{,}054515) = 30{,}060205$
$eval(\mathbf{v}_8) = f(-1{,}255173,\ 4{,}734458) = 25{,}341160$
$eval(\mathbf{v}_9) = f(3{,}130078,\ 4{,}996097) = 23{,}410669$
$eval(\mathbf{v}_{10}) = f(-2{,}516603,\ 4{,}390381) = 19{,}526329$
$eval(\mathbf{v}_{11}) = f(11{,}088621,\ 4{,}743434) = 33{,}351874$
$eval(\mathbf{v}_{12}) = f(0{,}795406,\ 5{,}381472) = 16{,}127799$
$eval(\mathbf{v}_{13}) = f(-1{,}811725,\ 4{,}209937) = 22{,}692462$
$eval(\mathbf{v}_{14}) = f(4{,}910618,\ 4{,}703018) = 17{,}959701$
$eval(\mathbf{v}_{15}) = f(7{,}935998,\ 4{,}757338) = 13{,}666916$
$eval(\mathbf{v}_{16}) = f(6{,}084492,\ 5{,}652242) = 26{,}019600$
$eval(\mathbf{v}_{17}) = f(-2{,}554851,\ 4{,}793707) = 21{,}278435$
$eval(\mathbf{v}_{18}) = f(11{,}134646,\ 5{,}666976) = 27{,}591064$
$eval(\mathbf{v}_{19}) = f(11{,}059532,\ 5{,}054515) = 27{,}608441$
$eval(\mathbf{v}_{20}) = f(9{,}211598,\ 4{,}993762) = 23{,}867227$

Zauważmy, że całkowite dopasowanie nowej populacji F wynosi 447,049688 i jest znacznie większe od całkowitego dopasowania poprzedniej populacji, która wynosiła 387,776822. Także najlepszy obecnie chromosom $(\mathbf{v}_{11})$ ma lepszą ocenę (33,351874) niż najlepszy chromosom $(\mathbf{v}_{15})$ poprzedniej populacji (30,060205).

Jesteśmy teraz gotowi do uruchomienia ponownie procesu selekcji i użycia operatorów genetycznych, oceny następnego pokolenia itd. Po 1000 pokoleń populacja składa się z:

$\mathbf{v}_1 = (111011110110011011100101010111011)$
$\mathbf{v}_2 = (111001100110000100010101010111000)$
$\mathbf{v}_3 = (111011110111011011100101010111011)$
$\mathbf{v}_4 = (111001100010000110000101010111001)$
$\mathbf{v}_5 = (111011110111011011100101010111011)$
$\mathbf{v}_6 = (111001100110000100000100010100001)$
$\mathbf{v}_7 = (110101100010010010001100010110000)$
$\mathbf{v}_8 = (111101100010001010001101010010001)$
$\mathbf{v}_9 = (111001100010010010001100010110001)$
$\mathbf{v}_{10} = (111011110111011011100101010111011)$
$\mathbf{v}_{11} = (110101100000010010001100010110000)$
$\mathbf{v}_{12} = (110101100010010010001100010110001)$
$\mathbf{v}_{13} = (111011110111011011100101010111011)$
$\mathbf{v}_{14} = (111001100110000100000101010111011)$
$\mathbf{v}_{15} = (111001101010111001010100110110001)$
$\mathbf{v}_{16} = (111001100110000101000100010100001)$
$\mathbf{v}_{17} = (111001100110000100000101010111011)$
$\mathbf{v}_{18} = (111001100110000100000101010111001)$
$\mathbf{v}_{19} = (111101100010001010001110000010001)$
$\mathbf{v}_{20} = (111001100110000100000101010111001)$

Wartości dopasowania wynoszą:

$eval(\mathbf{v}_1) = f(11{,}120940,\ 5{,}092514) = 30{,}298543$
$eval(\mathbf{v}_2) = f(10{,}588756,\ 4{,}667358) = 26{,}869724$
$eval(\mathbf{v}_3) = f(11{,}124627,\ 5{,}092514) = 30{,}316575$
$eval(\mathbf{v}_4) = f(10{,}574125,\ 4{,}242410) = 31{,}933120$
$eval(\mathbf{v}_5) = f(11{,}124627,\ 5{,}092514) = 30{,}316575$
$eval(\mathbf{v}_6) = f(10{,}588756,\ 4{,}214603) = 34{,}356125$
$eval(\mathbf{v}_7) = f(9{,}631066,\ 4{,}427881) = 35{,}458636$
$eval(\mathbf{v}_8) = f(11{,}518106,\ 4{,}452835) = 23{,}309078$
$eval(\mathbf{v}_9) = f(10{,}574816,\ 4{,}427933) = 34{,}393820$
$eval(\mathbf{v}_{10}) = f(11{,}124627,\ 5{,}092514) = 30{,}316575$
$eval(\mathbf{v}_{11}) = f(9{,}623693,\ 4{,}427881) = 35{,}477938$
$eval(\mathbf{v}_{12}) = f(9{,}631066,\ 4{,}427933) = 35{,}456066$
$eval(\mathbf{v}_{13}) = f(11{,}124627,\ 5{,}092514) = 30{,}316575$
$eval(\mathbf{v}_{14}) = f(10{,}588756,\ 4{,}242514) = 32{,}932098$
$eval(\mathbf{v}_{15}) = f(10{,}606555,\ 4{,}653714) = 30{,}746768$
$eval(\mathbf{v}_{16}) = f(10{,}588814,\ 4{,}214603) = 34{,}359545$
$eval(\mathbf{v}_{17}) = f(10{,}588756,\ 4{,}242514) = 32{,}932098$
$eval(\mathbf{v}_{18}) = f(10{,}588756,\ 4{,}242410) = 32{,}956664$
$eval(\mathbf{v}_{19}) = f(11{,}518106,\ 4{,}472757) = 19{,}669670$
$eval(\mathbf{v}_{20}) = f(10{,}588756,\ 4{,}242410) = 32{,}956664$

Jednak, jeżeli popatrzymy starannie na przebieg obliczeń, możemy odkryć, że we wcześniejszych pokoleniach wartość dopasowania niektórych chromosomów była wyższa niż wartość 35,477938 najlepszego chromosomu po 1000 pokoleń. Na przykład najlepszy chromosom w pokoleniu 396 miał wartość 38,827553. Przyczyną tego jest błąd stochastyczny związany z próbą. Omówimy tę sprawę w rozdz. 4.

Najlepsze osobniki można stosunkowo łatwo zapamiętywać w trakcie procesu ewolucyjnego. Na ogół (w programach algorytmów genetycznych) zapamiętuje się „najlepszego dotychczas" osobnika w oddzielnej zmiennej. W ten sposób algorytm podaje najlepszą wartość znalezioną w ciągu całego procesu (w odróżnieniu od najlepszej wartości w końcowej populacji).

3

Algorytmy genetyczne: dlaczego one działają?

Gatunki nie wzrastają w doskonałości:
słabi biorą wciąż górę nad silnymi,
– co stąd pochodzi, iż są w wielkiej liczbie,
przytem przeważają roztropnością.

Friedrich Nietzsche, *Zmierzch bożyszcz*[1)]

Teoretyczne podstawy algorytmów genetycznych opierają się na reprezentacji rozwiązań za pomocą łańcuchów binarnych i na pojęciu schematów (patrz na przykład [188]) – szablonie pozwalającym badać podobieństwa między chromosomami. Schemat buduje się, wprowadzając symbol *nieistotne* (*) do alfabetu genów. Schemat reprezentuje wszystkie łańcuchy (hiperpłaszczyznę lub podzbiór przestrzeni przeszukiwań), które zgadzają się z nim na wszystkich pozycjach innych niż „*".

Rozważmy na przykład łańcuch i schemat o długości 10. Do schematu (* 1 1 1 1 0 0 1 0 0) pasują dwa łańcuchy

{(0111100100), (1111100100)}

a do schematu (* 1 * 1 1 0 0 1 0 0) cztery łańcuchy

{(0101100100), (0111100100), (1101100100), (1111100100)}

Oczywiście schemat (1 0 0 1 1 1 0 0 0 1) reprezentuje tylko jeden łańcuch (1001110001), a schemat (**********) reprezentuje wszystkie łańcuchy o długości 10. Można zauważyć, że do każdego schematu pasuje dokładnie 2^r łańcuchów, gdzie r jest liczbą symboli *nieistotne* „*" w szablonie schematu. Z drugiej strony do każdego łańcucha o długości m pasuje 2^m schematów. Rozważmy na przykład łańcuch (1001110001). Pasuje do niego 2^{10} schematów

(1 0 0 1 1 1 0 0 0 1)
(* 0 0 1 1 1 0 0 0 1)

[1)] Przełożył Stanisław Wyrzykowski, wyd. Jakób Mortkowicz, 1905-1906, przedruk Wyd. „bis", 1991 (przyp. tłum.).

(1 * 0 1 1 1 0 0 0 1)
(1 0 * 1 1 1 0 0 0 1)

⋮

(1 0 0 1 1 1 0 0 0 *)
(* * 0 1 1 1 0 0 0 1)
(* 0 * 1 1 1 0 0 0 1)

⋮

(1 0 0 1 1 1 0 0 * *)
(* * * 1 1 1 0 0 0 1)

⋮

(* * * * * * * * * *)

Dla łańcuchów o długości m istnieje 3^m wszystkich możliwych schematów. W populacji o liczności n może być reprezentowanych od 2^m do $n \cdot 2^m$ różnych schematów.

Różne schematy mają różne właściwości. Zauważyliśmy już, że liczba symboli *nieistotne* * w schemacie określa liczbę łańcuchów, które do niego pasują. Wprowadza się dwie istotne właściwości schematu: *rząd i długość definiującą*. Będą one potrzebne do sformułowania twierdzenia o schematach.

Rzędem schematu S (oznaczanym przez $o(S)$) nazywamy liczbę pozycji 0 lub 1, to znaczy pozycji *ustalonych* (pozycji nie *nieistotnych*) w schemacie. Inaczej mówiąc, jest to długość szablonu minus liczba symboli *nieistotne* (*). Rząd określa specyficzność schematu. Na przykład następujące 3 schematy, każdy o długości 10

$S_1 = (* * * 0 0 1 * 1 1 0)$
$S_2 = (* * * * 0 0 * * 0 *)$
$S_3 = (1 1 1 0 1 * * 0 0 1)$

mają następujące rzędy:

$$o(S_1) = 6,\ o(S_2) = 3 \text{ i } o(S_3) = 8$$

więc schemat S_3 jest najbardziej specyficzny.

Pojęcie rzędu schematu jest użyteczne przy obliczaniu prawdopodobieństwa przeżycia schematu po mutacji. Będziemy to omawiać dalej w tym rozdziale.

Długość definiująca schematu S (oznaczana przez $\delta(S)$) jest odległością między pierwszą a ostatnią ustaloną pozycją schematu. Określa ona zwartość informacji zawartej w schemacie. Na przykład

$$\delta(S_1) = 10 - 4 = 6, \quad \delta(S_2) = 9 - 5 = 4, \quad \delta(S_3) = 10 - 1 = 9$$

Zauważmy, że schemat z pojedynczą ustaloną pozycją ma długość definiującą 0.

Pojęcie długości definiującej schematu jest użyteczne przy obliczaniu prawdopodobieństwa przeżycia po krzyżowaniu. Będziemy to omawiać dalej w tym rozdziale.

Zgodnie z wcześniejszą dyskusją, symulowany proces ewolucyjny algorytmu genetycznego składa się z czterech powtarzających się po sobie kroków

$t \leftarrow t + 1$
wyselekcjonuj $P(t)$ z $P(t-1)$
zastosuj operatory mieszania do $P(t)$
oceń $P(t)$

Pierwszy krok ($t \leftarrow t + 1$) po prostu przesuwa zegar ewolucji o jednostkę dalej. W czasie ostatniego kroku (oceń $P(t)$) oceniamy bieżącą populację. Główne przemiany w procesie ewolucji zachodzą w pozostałych krokach cyklu ewolucyjnego: selekcji i mieszania. Omówimy efekt tych dwóch kroków na spodziewanej liczbie schematów reprezentujących populację. Zaczniemy od kroku selekcji. Będziemy objaśniać wzory na ciągnącym się przez cały punkt przykładzie.

Przypuśćmy, że w populacji o liczności $pop_size = 20$ długość łańcucha (a w konsekwencji i długość szablonu schematu) wynosi $m = 33$ (tak jak to było w przykładzie ciągnącym się przez poprzedni rozdział). Załóżmy dalej, że (w chwili t) populacja składa się z następujących łańcuchów:

$\mathbf{v}_1 = (100110100000001111111010011011111)$
$\mathbf{v}_2 = (111000100100110111001010100011010)$
$\mathbf{v}_3 = (000010000011001000001010111011101)$
$\mathbf{v}_4 = (100011000101101001111000001110010)$
$\mathbf{v}_5 = (000111011001010011010111111000101)$
$\mathbf{v}_6 = (000101000010010101001010111111011)$
$\mathbf{v}_7 = (001000100000110101111011011111011)$
$\mathbf{v}_8 = (100001100001110100010110101100111)$
$\mathbf{v}_9 = (010000000101100010110000001111100)$
$\mathbf{v}_{10} = (000001111000110000011010000111011)$
$\mathbf{v}_{11} = (011001111110110101100001101111000)$
$\mathbf{v}_{12} = (110100010111101101000101010000000)$
$\mathbf{v}_{13} = (111011111010001000110000001000110)$
$\mathbf{v}_{14} = (010010011000001010100111100101001)$
$\mathbf{v}_{15} = (111011101101110000100011111011110)$
$\mathbf{v}_{16} = (110011110000011111100001101001011)$
$\mathbf{v}_{17} = (011010111111001111010001101111101)$
$\mathbf{v}_{18} = (011101000000001110100111110101101)$
$\mathbf{v}_{19} = (000101010011111111110000110001100)$
$\mathbf{v}_{20} = (101110010110011110011000101111110)$

Oznaczmy przez $\xi(S, t)$ liczbę łańcuchów w populacji w chwili t pasujących do schematu S. Na przykład dla schematu

$$S_0 = (****111**********)$$

$\xi(S_0, t) = 3$, gdyż do schematu S_0 pasują 3 łańcuchy: $\mathbf{v}_{13}$, $\mathbf{v}_{15}$ i $\mathbf{v}_{16}$. Zauważmy, że rząd schematu S_0 wynosi $o(S_0) = 3$, a jego długość definiująca $\delta(S_0) = 7 - 5 = 2$.

Inną właściwością schematu jest jego dopasowanie w chwili t, $eval(S, t)$. Jest ono zdefiniowane jako średnie dopasowanie z wszystkich łańcuchów populacji pasujących do schematu S. Przypuśćmy, że w pewnej populacji do schematu S pasuje p łańcuchów $\{\mathbf{v}_{i1}, ..., \mathbf{v}_{ip}\}$. Wtedy mamy

$$eval(S, t) = \sum_{j=1}^{p} eval(\mathbf{v}_{ij})/p$$

W kroku selekcji tworzy się pośrednią populację złożoną z $pop_size = 20$ wybranych łańcuchów. Każdy łańcuch początkowej populacji jest skopiowany zero, jeden lub więcej razy, zależnie od jego dopasowania. Jak widzieliśmy w poprzednim rozdziale, przy rozpatrywaniu każdego łańcucha $\mathbf{v}_i$ ma on prawdopodobieństwo selekcji $p_i = eval(\mathbf{v}_i)/F(t)$. ($F(t)$ jest całkowitym dopasowaniem populacji w chwili t, $F(t) = \sum_{i=1}^{20} eval(\mathbf{v}_i)$).

Po kroku selekcji spodziewamy się, że $\xi(S, t + 1)$ łańcuchów będzie pasowało do schematu S. Ponieważ, po pierwsze, prawdopodobieństwo selekcji (dla pojedynczej selekcji) przeciętnego łańcucha pasującego do schematu S wynosi $eval(S, t)/F(t)$, po drugie, liczba łańcuchów pasujących do schematu S jest równa $\xi(S, t)$, a po trzecie, liczba selekcji łańcuchów wynosi pop_size, więc oczywiście

$$\xi(S, t + 1) = \xi(S, t) \cdot pop_size \cdot eval(S, t)/F(t)$$

Wzór ten możemy przekształcić, biorąc pod uwagę, że średnie dopasowanie populacji wyraża się wzorem $\overline{F(t)} = F(t)/pop_size$, skąd

$$\xi(S, t + 1) = \xi(S, t) \cdot eval(S, t)/\overline{F(t)} \tag{3.1}$$

Inaczej mówiąc, liczba łańcuchów pasujących do schematu S zmienia się w populacji tak, jak stosunek dopasowania schematu do średniego dopasowania populacji. To oznacza, że schemat oceniany „powyżej średniej" uzyska zwiększoną liczbę łańcuchów w następnym pokoleniu, schemat oceniany „poniżej średniej" uzyska zmniejszoną liczbę łańcuchów, a średni schemat pozostanie na tym samym poziomie.

Długoterminowy efekt tej reguły jest także oczywisty. Jeżeli założymy, że schemat S ma dopasowanie powyżej średniej o ε% (tzn. $eval(S, t) = \overline{F(t)} + \varepsilon \cdot \overline{F(t)}$), to

$$\xi(S, t) = \xi(S, 0) \cdot (1 + \varepsilon)^t$$

oraz $\varepsilon = (eval(S, t) - \overline{F(t)})/\overline{F(t)}$ ($\varepsilon > 0$ dla schematów ocenianych powyżej średniej i $\varepsilon < 0$ dla schematów ocenianych poniżej średniej).

Jest to równanie wzrostu geometrycznego. Możemy teraz nie tylko powiedzieć, że schemat „powyżej średniej" uzyska zwiększoną liczbę łańcuchów w następnym pokoleniu, ale też, że taki schemat uzyska *wykładniczo* rosnącą liczbę łańcuchów w następnych pokoleniach.

Równanie (3.1) nazwiemy równaniem wzrostu schematu.

Powróćmy do przykładowego schematu S_0. Ponieważ w chwili t mamy 3 łańcuchy, a mianowicie $\mathbf{v}_{13}$, $\mathbf{v}_{15}$ i $\mathbf{v}_{16}$, pasujące do schematu S_0, dopasowanie $eval(S_0, t)$ schematu jest

$$eval(S_0, t) = (27{,}316702 + 30{,}060205 + 23{,}867227)/3 = 27{,}081378$$

Jednocześnie, średnie dopasowanie całej populacji jest

$$\overline{F(t)} = \sum_{i=1}^{20} eval(\mathbf{v}_i)/pop_size = 387{,}776822/20 = 19{,}388841$$

i stosunek dopasowania schematu S_0 do średniego dopasowania wynosi

$$eval(S_0)/\overline{F(t)} = 1{,}396751$$

Jeżeli schemat S_0 ma ocenę powyżej średniej, to uzyskuje wykładniczo rosnącą liczbę łańcuchów w następnych pokoleniach. W szczególności, jeżeli ocena schematu S_0 jest większa od średniej w stałym stosunku 1,396751, to w chwili $t+1$ możemy spodziewać się $3 \times 1{,}396751 = 4{,}19$ łańcuchów pasujących do S_0 (tzn. najprawdopodobniej 4 lub 5), w chwili $t+2$ możemy spodziewać się $3 \times 1{,}3967512 = 5{,}85$ takich łańcuchów (tzn. najprawdopodobniej 6) itd.

Intuicyjnie oznacza to, że schemat S_0 określa obiecującą część przeszukiwanej przestrzeni i że będzie ona sprawdzana w wykładniczo wzrastający sposób.

Sprawdźmy te przewidywania na naszym ciągnącym się przykładzie dla schematu S_0. W chwili t w populacji pasowały do schematu S_0 trzy łańcuchy: $\mathbf{v}_{13}$, $\mathbf{v}_{15}$ i $\mathbf{v}_{16}$. W poprzednim rozdziale symulowaliśmy proces selekcji w tej samej populacji. Nowa populacja składa się więc z następujących chromosomów:

$\mathbf{v}'_1$ = (01100111111011010110000110111 1000) ($\mathbf{v}_{11}$)
$\mathbf{v}'_2$ = (10001100010110100111100000111 0010) ($\mathbf{v}_4$)
$\mathbf{v}'_3$ = (00100010000011010111101101111 1011) ($\mathbf{v}_7$)
$\mathbf{v}'_4$ = (01100111111011010110000110111 1000) ($\mathbf{v}_{11}$)
$\mathbf{v}'_5$ = (00010101001111111111000011000 1100) ($\mathbf{v}_{19}$)
$\mathbf{v}'_6$ = (10001100010110100111100000111 0010) ($\mathbf{v}_4$)
$\mathbf{v}'_7$ = (11101110110111000010001111101 1110) ($\mathbf{v}_{15}$)
$\mathbf{v}'_8$ = (00011101100101001101011111100 0101) ($\mathbf{v}_5$)

$v'_9 = (0110011111101101011000011011110 00)$ (v_{11})
$v'_{10} = (0000100000110010000010101110111 01)$ (v_3)
$v'_{11} = (1110111011011100001000111110111 10)$ (v_{15})
$v'_{12} = (0100000001011000101100000011111 00)$ (v_9)
$v'_{13} = (0001010000100101010010101111111 011)$ (v_6)
$v'_{14} = (1000011000011101000101101011001 11)$ (v_8)
$v'_{15} = (1011100101100111100110001011111 10)$ (v_{20})
$v'_{16} = (1110011001100001010001000101000 01)$ (v_1)
$v'_{17} = (1110011001100001000001010101110 11)$ (v_{10})
$v'_{18} = (1110111110100010001100000010001 10)$ (v_{13})
$v'_{19} = (1110111011011100001000111110111 10)$ (v_{15})
$v'_{20} = (1100111100000111111000011010010 11)$ (v_{16})

Rzeczywiście, obecnie (w chwili $t + 1$) do schematu S_0 pasuje 5 łańcuchów: v'_7, v'_{11}, v'_{18}, v'_{19} i v'_{20}.

Jednak sama selekcja nie wprowadza żadnego nowego punktu (potencjalnego rozwiązania) z przeszukiwanej przestrzeni. W trakcie selekcji po prostu kopiuje się pewne łańcuchy, tworząc populację pośrednią. Tak więc dopiero drugi krok cyklu ewolucyjnego, rekombinacji, odpowiada za wprowadzenie nowych osobników do populacji. Robi się to za pomocą dwóch operatorów genetycznych: krzyżowania i mutacji. Przedyskutujemy teraz po kolei wpływ tych dwóch operacji na oczekiwaną liczbę schematów.

Zacznijmy od krzyżowania i rozważmy następujący przykład. Jak to omawiano wcześniej w tym rozdziale, pojedynczy łańcuch w populacji, powiedzmy v'_{18}

(1110111110100010001100000010001 10)

pasuje do 2^{30} schematów. W szczególności pasuje on do następujących dwóch schematów:

$S_0 = (****111**********)$ i

$S_1 = (111**********10)$

Przypuśćmy, że powyższy łańcuch wybrano do krzyżowania (jak się to zdarzyło w rozdz. 2). Przypuśćmy dalej (zgodnie z rozdz. 2, gdzie v'_{18} był skrzyżowany z v'_{13}), że wygenerowany punkt cięcia $pos = 20$. Widać, że schemat S_0 przetrwał to krzyżowanie, to znaczy jeden z potomków nadal pasuje do S_0. Przyczyna leży w tym, że cięcie zachowało sekwencję „111" leżącą na piątej, szóstej i siódmej pozycji w jednym z pary potomków, to znaczy

$v'_{18} = (11101111101000100011|0000001000110)$

$v'_{13} = (00010100001001010100|1010111111011)$

utworzyły

$$v''_{18} = (111011111101000100011|1010111111011)$$

$$v''_{13} = (000101000010010101 00|0000001000110)$$

Z drugiej strony schemat S_1 jest zniszczony. Żaden z potomków do niego nie pasuje. Przyczyna leży w tym, że ustalone pozycje „111" na początku szablonu i „10" na końcu znalazły się w różnych potomkach.

Jest chyba oczywiste, że długość definiująca schematu spełnia istotną rolę w prawdopodobieństwie jego przetrwania lub zniszczenia. Zauważmy, że długość definiująca schematu S_0 wynosi $\delta(S_0) = 2$, a schematu S_1 wynosi $\delta(S_1) = 32$.

W ogólności punkt cięcia jest ustalany równomiernie spośród $m - 1$ możliwych miejsc. Wynika z tego, że prawdopodobieństwo zniszczenia schematu S jest

$$p_d(S) = \delta(S)/(m - 1)$$

a w konsekwencji prawdopodobieństwo przetrwania wynosi

$$p_s(S) = 1 - \delta(S)/(m - 1)$$

I faktycznie prawdopodobieństwa przetrwania i zniszczenia schematów S_0 i S_1 w naszym przykładzie wynoszą

$$p_d(S_0) = 2/32, \quad p_s(S_0) = 30/32, \quad p_d(S_1) = 32/32 = 1, \quad p_d(S_1) = 0$$

a więc wynik był do przewidzenia.

Należy jednak zwrócić uwagę na to, że tylko pewne chromosomy podlegają krzyżowaniu, a prawdopodobieństwo wyboru do krzyżowania wynosi p_c. Oznacza to, że prawdopodobieństwo przetrwania schematu faktycznie wynosi

$$p_s(S) = 1 - p_c \cdot \delta(S)/(m - 1)$$

Ponownie wracając do naszego przykładu ($p_c = 0{,}25$) i schematu S_0

$$p_s(S_0) = 1 - 0{,}25 \cdot 2/32 = 63/64 = 0{,}984375$$

Zauważmy także, że chociaż punkt cięcia wypadł między ustalonymi pozycjami schematu, nadal jest szansa na to, iż schemat przetrwa. Jeśli na przykład *oba* łańcuchy v'_{18} i v'_{13} zaczynają się od „111" i kończą na „10", to schemat S_1 przetrwa krzyżowanie (jednakże prawdopodobieństwo takiego przypadku jest zupełnie małe). Z tego powodu powinniśmy zmodyfikować wzór na prawdopodobieństwo przetrwania schematu

$$p_s(S) \geqslant 1 - p_c \cdot \delta(S)/(m - 1)$$

Tak więc połączony efekt selekcji i krzyżowania daje nam nową postać równania wzrostu schematu

$$\xi(S, t+1) \geqslant \xi(S, t) \cdot eval(S, t)/\overline{F(t)}[1 - p_c \cdot \delta(S)/(m-1)] \tag{3.2}$$

Równanie (3.2) przedstawia oczekiwaną liczbę łańcuchów pasujących do schematu S w następnym pokoleniu jako funkcję obecnej liczby łańcuchów pasujących do schematu, względnego dopasowania schematu i jego długości definiującej. Rzecz jasna, schemat oceniany powyżej średniej, z krótką długością definiującą, będzie nadal miał pasujące łańcuchy z wykładniczo rosnącą szybkością. Dla schematu S_0

$$eval(S_0, t)/\overline{F(t)}[1 - p_c \cdot \delta(S)/(m-1)] = 1{,}396751 \cdot 0{,}984375 = 1{,}374927$$

Oznacza to, że krótki, oceniany powyżej średniej schemat S_0 nadal uzyska wykładniczo rosnącą liczbę łańcuchów w następnych pokoleniach. W chwili $(t+1)$ spodziewamy się mieć $3 \times 1{,}374927 = 4{,}12$ łańcuchów pasujących do S_0 (tylko trochę mniej niż 4,19 – wartość, którą otrzymaliśmy, biorąc pod uwagę tylko selekcję), a w chwili $(t+2)$:$3 \times 1{,}374927^2 = 5{,}67$ takich łańcuchów (ponownie nieco mniej niż 5,85).

Następny rozważany operator to mutacja. Operator mutacji losowo zmienia pojedyncze pozycje chromosomu z prawdopodobieństwem p_m. Zmiana następuje z 1 na 0 lub odwrotnie. Jest oczywiste, że wszystkie ustalone pozycje schematu muszą zostać nienaruszone, jeżeli schemat ma przetrwać mutację. Rozważmy, na przykład, ponownie jeden z łańcuchów w populacji, powiedzmy $\mathbf{v}'_{19}$

(111011101101110000100011111011110)

i schemat S_0

S_0 = (****111**********)

Załóżmy teraz, że w łańcuchu $\mathbf{v}'_{19}$ zaszła mutacja, to znaczy przynajmniej jeden bit został zmieniony, jak to się stało w poprzednim rozdziale. (Przypomnijmy sobie, że 4 łańcuchy uległy wtedy mutacji, przy czym jeden z nich $\mathbf{v}'_{13}$ uległ mutacji na dwóch pozycjach, a trzy inne – łącznie z $\mathbf{v}'_{19}$ – na jednej). Ponieważ $\mathbf{v}'_{19}$ doznał mutacji na ósmej pozycji, jego potomek

$\mathbf{v}''_{19}$ = (111011100101110000100011111011110)

nadal pasuje do schematu S_0. Jeżeli wybrana do mutacji pozycja leżałaby między 1 a 4 lub 8 a 33, to wynikowy potomek nadal pasowałby do S_0. Tylko 3 bity (piąty, szósty i siódmy - ustalone pozycje na schemacie S_0) są „ważne".

Mutacja przynajmniej jednego z nich zniszczy schemat S_0. Oczywiście liczba takich „ważnych" bitów równa się rzędowi schematu, to znaczy liczbie ustalonych pozycji.

Ponieważ prawdopodobieństwo zmiany pojedynczego bitu wynosi p_m, więc prawdopodobieństwo przetrwania bitu jest równe $1 - p_m$. Pojedyncza mutacja jest niezależna od innych mutacji, więc prawdopodobieństwo, że schemat przetrwa mutację (tzn. sekwencję mutacji bitów), wyraża się wzorem

$$p_s(S) = (1 - p_m)^{o(S)}$$

Ponieważ $p_m \ll 1$, a zatem powyższe prawdopodobieństwo można przybliżyć przez

$$p_s(S) \approx 1 - o(S)p_m$$

Ponownie wracając do naszego przykładowego schematu S_0 i ciągnącego się przykładu (gdzie $p_m = 0{,}01$), mamy

$$p_s(S_0) \approx 1 - 3 \cdot 0{,}01 = 0{,}97$$

Połączony wpływ selekcji, krzyżowania i mutacji daje nam nową postać równania wzrostu schematu

$$\xi(S, t+1) \geqslant \xi(S, t) \cdot eval(S, t)/\overline{F(t)}[1 - p_c \cdot \delta(S)/(m-1) - o(S)p_m] \qquad (3.3)$$

Tak jak w prostszych postaciach (równania (3.1) i (3.2)) równanie (3.3) podaje nam oczekiwaną liczbę łańcuchów pasujących do schematu S w następnym pokoleniu jako funkcję względnego dopasowania schematu, jego długości definiującej i rzędu. I ponownie widać, że liczba łańcuchów schematu ocenianego powyżej średniej, z krótką długością definiującą i niskim rzędem będzie wzrastała wykładniczo.

Dla schematu S_0 mamy

$$eval(S_0, t)/\overline{F(t)}[1 - p_c \cdot \delta(S_0)/(m-1) - o(S_0)p_m] = \\ = 1{,}396751 \cdot 0{,}954375 = 1{,}333024$$

Oznacza to, że krótki, niskiego rzędu i oceniany powyżej średniej schemat S_0 uzyska wykładniczo rosnącą liczbę łańcuchów w następnym pokoleniu. W chwili $(t + 1)$ spodziewamy się uzyskać $3 \times 1{,}333024 = 4{,}00$ łańcuchów pasujących do S_0 (niewiele mniej niż 4,19 – wartości uzyskanej przy uwzględnieniu tylko selekcji, lub 4,12 – wartości uzyskanej przy uwzględnieniu selekcji i krzyżowania). W chwili $(t + 2)$ spodziewamy się $3 \times 1{,}333024^2 = 5{,}33$ takich łańcuchów (znowu niewiele mniej niż 5,85 lub 5,67).

Zauważmy, że równanie (3.3) opiera się na założeniu, że funkcja dopasowania f jest dodatnia. Stosując algorytmy genetyczne w zadaniach optymaliza-

cji, gdzie optymalizowana funkcja może być ujemna, musimy zastosować dodatkowe przekształcenie zmieniające ją na dodatnią funkcję dopasowania. Będziemy omawiali to zagadnienie w następnym rozdziale.

Podsumowując, z równania wzrostu (3.1) wynika, że selekcja zwiększa liczbę łańcuchów pasujących do schematu ocenianego powyżej średniej i że ta zmiana jest wykładnicza. Sama selekcja nie wprowadza nowych schematów (nie reprezentowanych w populacji początkowej dla $t = 0$). To jest właśnie przyczyna wprowadzenia operacji krzyżowania – aby umożliwić ukierunkowaną, ale jednocześnie losową wymianę informacji genetycznej. Dodatkowo operator mutacji wprowadza większą zmienność w populacji. Połączony (rozrywający) efekt tych dwóch operacji nie jest istotny, jeżeli schemat jest krótki i niskiego rzędu. Końcowy wynik przedstawiony równaniem wzrostu (3.3) można wyrazić jako twierdzenie.

Twierdzenie 1 (Twierdzenie o schematach)
Krótkie, niskiego rzędu i oceniane powyżej średniej schematy uzyskują wykładniczo rosnącą liczbę łańcuchów w kolejnych pokoleniach.

Z twierdzenia tego natychmiast wynika, że algorytmy genetyczne badają przeszukiwaną przestrzeń przez krótkie, niskiego rzędu schematy, które z kolei są używane do wymiany informacji genetycznej podczas krzyżowania.

Hipoteza 1 (Hipoteza o blokach budujących)
Algorytm genetyczny poszukuje działania zbliżonego do optymalnego przez zestawianie krótkich, niskiego rzędu schematów o dużej wydajności działania, zwanych blokami budującymi.

Jak podano w [154]:

> Tak jak dziecko tworzy wspaniałe fortece przez ułożenie prostych bloczków drewnianych, tak algorytm genetyczny poszukuje działania zbliżonego do optymalnego przez zestawienie krótkich, niskiego rzędu schematów o dużej wydajności działania.

W tym rozdziale mieliśmy wspaniały przykład bloku budującego

$$S_0 = (****111**********)$$

S_0 jest krótkim schematem o niskim rzędzie, a do tego (przynajmniej we wczesnych populacjach) był oceniany powyżej średniej. Ten schemat wpływał na ruch w kierunku optimum.

Chociaż prowadzono badania w celu dowodu powyższej hipotezy [38], w większości nietrywialnych zastosowań polegamy głównie na wynikach empirycznych. W ciągu ostatnich piętnastu lat algorytmy genetyczne były często używane, co potwierdziło prawdziwość hipotezy o blokach budujących w wielu

różnych zagadnieniach. Niemniej jednak hipoteza sugeruje, że kodowanie w algorytmie genetycznym jest krytyczne dla jego działania i że przy takim kodowaniu powinno się brać pod uwagę pojęcie krótkich bloków budujących.

Wcześniej w tym rozdziale stwierdziliśmy, że populacja *pop_size* osobników o długości m przetwarza co najmniej 2^m i co najwyżej 2^{pop_size} schematów. Część z nich jest przetwarzana w użyteczny sposób. Tworzą one (pożądaną) liczbę wykładniczo rosnących łańcuchów i nie są rozrywane przez krzyżowania i mutacje (co może się łatwiej zdarzyć schematom o większej długości definiującej i wyższym rzędzie).

Holland [188] wykazał, że przynajmniej pop_size^3 z nich jest przetwarzanych użytecznie. Nazwał on tę właściwość *wewnętrzą równoległością*, gdyż uzyskuje się ją bez żadnej dodatkowej pamięci lub przetwarzania. Warto może zauważyć, że w populacji o *pop_size* łańcuchach jest reprezentowanych dużo więcej niż *pop_size* schematów.[1)]

Jest to być może jedyny znany przykład wykładniczej eksplozji, która pracuje na naszą korzyść zamiast na niekorzyść.

W tym rozdziale podaliśmy uznane wyjaśnienie, dlaczego algorytmy genetyczne pracują. Zauważmy jednak, że hipoteza o blokach budujących jest jedynie aktem wiary, który dla niektórych zadań może być łatwo naruszony. Przypuśćmy na przykład, że dwa krótkie schematy o niskim rzędzie (obecnie rozważymy schematy o całkowitej długości wynoszącej 11 pozycji)

$$S_1 = (1\ 1\ 1\ *\ *\ *\ *\ *\ *\ *\ *) \quad \text{i}$$

$$S_2 = (*\ *\ *\ *\ *\ *\ *\ *\ *\ 1\ 1)$$

są oba oceniane powyżej średniej, ale ich kombinacja

$$S_3 = (1\ 1\ 1\ *\ *\ *\ *\ *\ *\ 1\ 1)$$

ma dopasowanie gorsze niż

$$S_4 = (0\ 0\ 0\ *\ *\ *\ *\ *\ *\ 0\ 0)$$

Załóżmy dalej, że optymalnym łańcuchem jest $s_0 = (11111111111)$ (pasuje on do S_3). Algorytm genetyczny może mieć trudności w zbiegnięciu do s_0, gdyż może zbiegać do punktów w rodzaju (00011111100). Jest to nazywane zawodem [38], [154]. Pewne bloki budujące (krótkie schematy o niskim rzędzie) mogą źle ukierunkować algorytm i spowodować jego zbieżność do punktów suboptymalnych.

[1)] Ostatnio Bertoni i Dorigo [36] wykazali, że oszacowanie $pop\text{-}size^3$ jest prawdziwe tylko w szczególnym przypadku, kiedy *pop-size* jest proporcjonalne do 2^l, oraz przeprowadzili ogólniejszą analizę zagadnienia.

Zjawisko zawodu jest silnie połączone z pojęciem *epistazy*, które (dla algorytmów genetycznych) oznacza silne uzależnienie między genami w chromosomie.[1] Inaczej mówiąc, epistaza podaje, na ile wpływ jednego genu na dopasowanie łańcucha zależy od wartości innych genów. Dla danego zadania wysoka epistaza oznacza, że bloki budujące nie mogą się formować i w rezultacie zadanie jest zawodne.

Zaproponowano trzy sposoby postępowania w przypadku zawodu (patrz [155]). W pierwszym zakłada się początkową umiejętność zakodowania funkcji celu w odpowiedni sposób (aby uzyskać „ścisłe" bloki budujące). Na przykład początkowa wiedza o funkcji celu i w konsekwencji o zawodzie mogłaby wpłynąć na inne zakodowanie, w którym pięć bitów potrzebnych do optymalizacji funkcji jest przyległych zamiast rozdzielonych sześcioma innymi pozycjami.

W drugim sposobie używa się trzeciego operatora genetycznego, *inwersji*. Pojedyncza inwersja jest (podobnie jak mutacja) operatorem jednoargumentowym. Wybiera ona dwa punkty w łańcuchu i odwraca kolejność bitów między tymi punktami, pamiętając jednak „znaczenie" bitów. Oznacza to, że musimy rozpoznawać bity w łańcuchu. Robimy to przez utrzymywanie bitów razem z zapisem ich początkowej pozycji. Na przykład łańcuch

$$s = ((1, 0)(2, 0)(3, 0)|(4, 1)(5, 1)(6, 0)(7, 1)|(8, 0)(9, 0)(10, 0)(11, 1))$$

z dwoma zaznaczonymi punktami po inwersji staje się łańcuchem

$$s' = ((1, 0)(2, 0)(3, 0)|(7, 1)(6, 0)(5, 1)(4, 1)|(8, 0)(9, 0)(10, 0)(11, 1))$$

Algorytm genetyczny z inwersją, jako jednym z operatorów, poszukuje najlepszego ustawienia bitów do formowania bloków budujących. Na przykład rozważany wcześniej schemat

$$S_3 = (1\ 1\ 1\ *\ *\ *\ *\ *\ *\ 1\ 1)$$

przepisany w postaci

$$S_3 = ((1, 1)(2, 1)(3, 1)(4, *)(5, *)(6, *)(7, *)(8, *)(9, *)(10, 1)(11, 1))$$

może być (po udanej inwersji) przegrupowany do postaci

$$S_3 = ((1, 1)(2, 1)(3, 1)(11, 1)(10, 1)(9, *)(8, *)(7, *)(6, *)(5, *)(4, *))$$

tworząc ważny blok budujący. Jednak, jak zauważono w [155]:

> We wcześniejszej pracy [160] tłumaczono, że inwersja – operator jednoargumentowy – nie była użyteczna w wydajnym poszukiwaniu ścisłych bloków budujących, ponieważ nie ma możliwości zestawiania właściwej operatorom

[1] Genetycy używają terminu epistaza do opisu efektu maskowania lub przełączania. Gen jest epistatyczny, jeżeli jego obecność tłumi efekt innego genu znajdującego się w innym miejscu.

binarnym. Przedstawmy to inaczej, inwersja jest tym dla kolejności, czym mutacja dla alleli. Obie wykonują dobrą robotę przeciwko wstrzymującemu przeszukiwanie brakowi różnorodności, ale żadna nie jest na tyle mocna, aby szukać dobrych struktur, czy to allelicznych czy permutacyjnych, na własną rękę, kiedy dobre struktury wymagają epistatycznego oddziaływania poszczególnych części.

Trzeci sposób walki z zawodem zaproponowano ostatnio [155], [159]. Jest to nieporządny algorytm genetyczny. Ponieważ nieporządne algorytmy genetyczne mają ciekawe właściwości, rozpatrzymy je pokrótce w następnym rozdziale (p. 4.6).

4

Algorytmy genetyczne: wybrane zagadnienia

Pewien człowiek zobaczył motyla
usiłującego wydobyć się z kokonu.
Wydawało mu się, że motyl robi to zbyt wolno,
zaczął więc delikatnie dmuchać.
Ciepło powietrza przyśpieszyło proces,
wyłonił się jednak nie motyl,
lecz jakiś stwór z poszarpanymi skrzydłami.

Antonio de Mello, *Minuta mądrości*[1)]

Teoria algorytmów genetycznych daje pewne wyjaśnienie dlaczego, dla danego sformułowania zadania, możemy uzyskać zbieżność do szukanego punktu optymalnego. Niestety, zastosowania praktyczne nie zawsze idą w parze z teorią, a najważniejsze tego przyczyny są następujące:

- kodowanie powoduje, że algorytmy genetyczne pracują w innej przestrzeni niż przestrzeń zadania,
- hipotetycznie nieograniczona liczba iteracji musi być skończona,
- hipotetycznie nieograniczona liczebność populacji musi być skończona.

Jednym ze skutków powyższych przyczyn jest, w pewnych warunkach, niemożliwość znalezienia optymalnego rozwiązania. Jest to spowodowane przedwczesną zbieżnością algorytmu do lokalnego optimum. Przedwczesna zbieżność jest wspólnym problemem zarówno algorytmów genetycznych, jak i innych algorytmów optymalizacyjnych. Jeżeli zbieżność zachodzi zbyt szybko, to często traci się wartościową informację rozwijającą się w części populacji. Rozwiązania komputerowe algorytmów genetycznych mają skłonność zbiegać się zbyt wcześnie, zanim zostanie znalezione rozwiązanie optymalne. Jak stwierdzono w [46]:

> ...Chociaż działanie większości rozwiązań komputerowych jest porównywalne lub lepsze od działania innych metod przeszukiwania, to [algorytmy genetyczne] nadal zawodzą wysokie oczekiwania zrodzone przez teorię. Problem polega na tym, że teoria wskazuje na graniczne stosunki między łańcuchami i właściwościami przeszukiwań, gdy tymczasem jakiekolwiek rozwiązanie komputerowe używa skończonej populacji oraz skończonego zbioru punktów, w których sprawdza się funkcję (punktów próbkujących). Oszacowania oparte

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

na skończonej próbie są obarczone związanymi z tym błędami i prowadzą do trajektorii poszukiwań znacznie różniącej się od przewidzianej teoretycznie. Ten problem objawia się w praktyce jako zbyt wczesna strata różnorodności populacji, co prowadzi do przeszukiwania zbiegającego się do rozwiązania suboptymalnego.

Eshelman i Schaffer [105] omawiają kilka strategii zwalczania przedwczesnej zbieżności. Obejmują one (1) strategię doboru zwaną *zapobieganiem kazirodztwu*[1], (2) użycie jednorodnego krzyżowania (patrz p. 4.6), (3) wykrywanie jednakowych łańcuchów w populacji (podobne do modeli ze współczynnikiem zatłoczenia, patrz p. 4.1).

Jednak większość badań w tej dziedzinie jest związana z:

- zakresem i rodzajem błędów wprowadzonych przez mechanizm wyboru punktów próbnych,
- charakterystyką samej funkcji.

Te dwa zagadnienia są blisko związane. Jednak będziemy je omawiać po kolei (p. 4.1 i p. 4.2). Dodatkowe dwa punkty zawierają wyniki związane ze zbieżnością pewnej klasy algorytmów genetycznych (zwanych algorytmami genetycznymi z odwzorowaniem zwężającym), w których korzysta się z twierdzenia Banacha o punkcie stałym (p. 4.3), oraz z pewnymi eksperymentami z algorytmami genetycznymi ze zmienną liczebnością populacji (p. 4.4). Ostatni punkt zawiera kilka dodatkowych pomysłów na zwiększenie przeszukiwania genetycznego.

4.1. Mechanizm próbkowania

Wygląda na to, że w procesie ewolucyjnym związanym z przeszukiwaniem genetycznym występują dwa istotne zagadnienia: różnorodność populacji i napór selekcyjny. Te dwa czynniki są silnie powiązane. Zwiększenie naporu selekcyjnego zmniejsza różnorodność populacji i odwrotnie. Inaczej mówiąc, silny napór selekcyjny „wspomaga" przedwczesną zbieżność algorytmów genetycznych, za to mały napór selekcyjny może spowodować, że przeszukiwanie będzie mało efektywne. Dlatego jest ważne, aby utrzymać równowagę między tymi czynnikami. Mechanizmy próbkowania usiłują osiągnąć ten cel. Jak zauważył Whitley [395]:

Można uważać, że są tylko dwa podstawowe czynniki (i być może tylko dwa) w przeszukiwaniu genetycznym: różnorodność genetyczna i napór selek-

[1] Dodatkowe informacje o metodach zapobiegania kazirodztwu w zadaniu komiwojażera można znaleźć na końcu rozdz. 10.

cyjny. [...] W pewnym sensie jest to inna strona idei przebadania w opozycji do wykorzystania, którą dyskutowali Holland i inni. Wiele parametrów, które są używane do „strojenia" przeszukiwania genetycznego pośrednimi sposobami wpływa na napór selekcyjny i różnorodność populacji. Kiedy napór selekcyjny zwiększa się, przeszukiwanie skupia się na czołowych osobnikach populacji. Ale z powodu tego „wykorzystania" traci się różnorodność genetyczną. Zmniejszając napór selekcyjny (lub przyjmując większą populację), zwiększamy „badanie", gdyż w ten sposób wprowadza się do przeszukiwania więcej genotypów, a tym samym więcej schematów.

Pierwszą i być może najbardziej docenianą pracę na ten temat napisał De Jong [82] w 1975 r. Rozważył on kilka odmian prostej selekcji przedstawionej w poprzednim rozdziale. Pierwsza odmiana, nazwana *modelem elitarnym*, kładzie większy nacisk na utrzymanie najlepszych chromosomów. Druga odmiana, *model wartości oczekiwanej*, zmniejsza błąd stochastyczny selekcji. Robi się to przez wprowadzenie licznika dla każdego chromosomu $\mathbf{v}$, ustawianego na początku na wartość $f(\mathbf{v})/\bar{f}$ i zmniejszanego o 0,5 lub 1, odpowiednio dla krzyżowania lub mutacji, za każdym razem, gdy chromosom jest wybrany do reprodukcji. Kiedy licznik chromosomu schodzi poniżej zera, wtedy tego chromosomu nie uwzględnia się dalej w procesie selekcji. W trzeciej odmianie, *elitarnym modelu wartości oczekiwanej*, łączy się dwie pierwsze odmiany. W czwartym modelu, *modelu ze współczynnikiem zatłoczenia*, nowo wygenerowany chromosom wymienia „starego", a skazany na zagładę chromosom jest wybierany spośród takich, które są podobne do nowego.

W 1981 r. Brindle [49] rozważał pewne modyfikacje: *próbkowanie deterministyczne, stochastyczne próbkowanie na podstawie reszty bez zamiany, stochastyczne próbkowanie na podstawie reszty z zamianą i turniej stochastyczny*. Jego badania potwierdziły wyższość niektórych modyfikacji nad prostą selekcją. W szczególności udana i przyjęta przez wielu badaczy za wzorcową była *metoda stochastycznego próbkowania na podstawie reszty z zamianą*, w której punkty do badania otrzymuje się z części całkowitej wartości oczekiwanej pojawienia się każdego chromosomu w nowej populacji i w której chromosomy rywalizują o pozostałe miejsca w populacji na podstawie części ułamkowej. W 1987 r. Baker [23] przedstawił wyczerpujące studium teoretyczne tych modyfikacji, używając do tego pewnych dobrze zdefiniowanych miar. Przedstawił tam także nową, ulepszoną wersję zwaną *stochastycznym próbkowaniem uniwersalnym*. W metodzie tej używa się obrotu jednego koła ruletki. Koło to, które ma standardową konstrukcję (rozdz. 2), jest obracane z równo rozmieszczonymi znacznikami o liczbie równej liczebności populacji.

Inne metody próbkowania populacji polegają na wprowadzaniu sztucznych wag. Chromosomy są wybierane proporcjonalnie do ich kolejności, a nie do rzeczywistych wartości oceny (patrz np. [22], [395]). Te metody są oparte na wierze, że wspólną przyczyną szybkiej (przedwczesnej) zbieżności jest istnienie *superosobników*, którzy są znacznie lepsi niż średnie dopasowanie populacji.

Każdy superosobnik ma dużą liczbę potomstwa i (ze względu na stałą liczebność populacji) nie pozwala innym osobnikom na tworzenie potomków w następnym pokoleniu. W kilku pokoleniach superosobnik może wyeliminować pożądany materiał genetyczny i spowodować zbieżność do (pewnie lokalnego) optimum.

Istnieje wiele metod ustalania liczby potomków na podstawie uporządkowania. Na przykład Baker [22] przyjmuje wartość podaną przez użytkownika, MAX, jako granicę górną oczekiwanej liczby osobników, i prowadzi linię prostą przez MAX, tak aby powierzchnia pod linią była równa liczebności populacji. W ten sposób możemy łatwo określić różnicę między oczekiwaną liczbą potomków u „sąsiednich" osobników. Na przykład dla MAX = 2,0 i $pop_size = 50$ różnica między oczekiwaną liczbą potomków między „sąsiednimi" osobnikami wynosi 0,04.

Inna możliwość polega na przyjęciu parametru podanego przez użytkownika q i określeniu funkcji liniowej, na przykład

$$prob(rank) = q - (rank - 1)r$$

lub funkcji nieliniowej, jak na przykład

$$prob(rank) = q(1 - q)^{rank - 1}$$

Obie funkcje podają prawdopodobieństwo wyboru w pojedynczej selekcji osobnika znajdującego się na miejscu $rank$ ($rank = 1$ oznacza najlepszego osobnika, $rank = pop_size$ najgorszego).

Oba te sposoby pozwalają użytkownikowi wpływać na napór selekcyjny algorytmu. W przypadku funkcji liniowej z żądania

$$\sum_{i=1}^{pop_size} prob(i) = 1$$

wynika, że

$$q = r(pop_size - 1)/2 + 1/pop_size$$

Jeżeli $r = 0$ (a w konsekwencji $q = 1/pop_size$), to nie ma w ogóle naporu selekcyjnego. Wszystkie osobniki mają takie samo prawdopodobieństwo wyboru. Z drugiej strony, jeżeli $q - (pop_size - 1)r = 0$, to

$$r = 2/(n(n - 1)) \quad \text{i} \quad q = 2/n$$

co prowadzi do maksymalnego naporu selekcyjnego. Inaczej mówiąc, jeżeli wybierzemy funkcję liniową do określania prawdopodobieństw dla uporządkowanych osobników, to jeden parametr q, zmieniający się między $1/pop_size$

a $2/pop_size$ może sterować naporem selekcyjnym algorytmu. Na przykład, jeżeli $pop_size = 100$ i $q = 0{,}015$, to $r = q/(pop_size - 1) = 0{,}00015151515$ i $prob(1) = 0{,}015$, $prob(2) = 0{,}0148484848$, ..., $prob(100) = 0{,}00000000000000000051$.

Dla funkcji nieliniowej parametr $q \in (0, 1)$ nie zależy od liczebności populacji. Większe wartości q powodują mocniejszy napór selekcyjny algorytmu. Na przykład, jeżeli $q = 0{,}1$ i $pop_size = 100$, to $prob(1) = 0{,}100$, $prob(2) = 0{,}1 \cdot 0{,}9 = 0{,}090$, $prob(3) = 0{,}1 \cdot 0{,}9 \cdot 0{,}9 = 0{,}081$, ..., $prob(100) = 0{,}000003$. Zauważmy, że[1)]

$$\sum_{i=1}^{pop_size} prob(i) = \sum_{i=1}^{pop_size} q(1-q)^{i-1} \approx 1^2$$

Tego typu podejścia, chociaż wykazano, że w pewnych przypadkach poprawiają zachowanie się algorytmu genetycznego, mają oczywiste wady. Po pierwsze przenoszą one na użytkownika odpowiedzialność, kiedy użyć tego mechanizmu. Po drugie pomijają one informację o względnych ocenach różnych chromosomów. Po trzecie traktują one wszystkie przypadki jednakowo, nie biorąc pod uwagę wielkości zadania. Na koniec procedura selekcji oparta na uporządkowaniu nie jest zgodna z twierdzeniem o schematach. Jak jednak wykazały badania [23], [395], zapobiegają one kłopotom ze skalowaniem (omawianym w następnym punkcie), lepiej sterują naporem selekcyjnym i (łącznie z reprodukcją po jednym osobniku na raz) bardziej skupiają przeszukiwania.

Inna metoda selekcji, zwana *selekcją turniejową* [159], uwzględnia pomysł uporządkowania w bardzo ciekawy i efektywny sposób. W metodzie tej (w pojedynczej iteracji) wybiera się pewną liczbę k osobników i selekcjonuje najlepszego z tego zbioru k-elementowego do następnego pokolenia. Ten proces powtarza się pop_size razy. Rzecz jasna, duża wartość k zwiększa napór selekcyjny tej procedury. Typowa wartość dogodna w wielu zastosowaniach to $k = 2$ (tak zwany *rozmiar turnieju*). Można też dodać posmak symulowanego wyżarzania, uwzględniając selekcję Boltzmanna, w której dwa elementy: i oraz j rywalizują ze sobą, a zwycięzcę określa się zgodnie z wyrażeniem

$$\frac{1}{1 + e^{\frac{f(i) - f(j)}{T}}}$$

gdzie T jest temperaturą, a $f(i)$ oraz $f(j)$ są wartościami funkcji celu odpowiednio dla elementu i oraz j (wyrażenie odpowiada zadaniu minimalizacji).

Bäck i Hoffmeister klasyfikują w [16] procedury selekcji. Dzielą je oni na metody *dynamiczne* i *statyczne*. Statyczna selekcja oznacza, że prawdopodobieństwa wyboru są stałe dla wszystkich pokoleń (na przykład selekcja na podstawie uporządkowania). W selekcji dynamicznej natomiast takiego wyma-

---

[1)] Łatwo można zamienić w tym wzorze $\approx$ na $=$. Wystarczy określić $prob(i) = cq(1-q)^{i-1}$, gdzie $c = 1/[1 - (1-q)^{pop_size}]$.

gania nie ma (na przykład selekcja proporcjonalna). W innym podziale procedur selekcji rozróżnia się metody *wygaszające* i *zachowujące*. W selekcji zachowującej są niezerowe prawdopodobieństwa wyboru dla każdego osobnika. Natomiast w selekcji wygaszającej nie. Selekcje wygaszające dzielą się dalej na selekcje *lewe* i *prawe*. W lewej selekcji wygaszającej nie dopuszcza się najlepszych osobników do reprodukcji, aby uniknąć przedwczesnej zbieżności (w prawej tego nie ma). Dodatkowo niektóre procedury selekcji są *wyłączne*, co oznacza, że rodzice mogą reprodukować tylko w jednym pokoleniu (to znaczy czas życia każdego osobnika jest ograniczony tylko do jednego pokolenia, niezależnie od ich dopasowania). Wrócimy do zagadnienia wyłącznych selekcji wygaszających w rozdz. 8, gdzie będziemy omawiać strategie ewolucyjne i porównywać je z algorytmami genetycznymi. Niektóre selekcje są *pokoleniowe*, co oznacza, że zbiór rodziców jest stały tak długo, aż będą utworzone wszystkie dzieci dla następnej generacji. W selekcjach *w locie* potomek wymienia swojego rodzica natychmiast. Niektóre selekcje są *elitarne*, co oznacza, że niektórzy (lub wszyscy) rodzice mogą wchodzić do selekcji razem z ich potomkami. Mieliśmy już do czynienia z takimi selekcjami w modelu elitarnym [82].

W większości eksperymentów opisywanych w tej książce używaliśmy nowej, dwukrokowej odmiany podstawowego algorytmu selekcji. Jednak ta modyfikacja nie jest jakimś nowym mechanizmem selekcyjnym. Można w niej używać którejkolwiek metody próbkowania wymyślonej wcześniej. Jest ona zaprojektowana do zmniejszania (możliwych) niechcianych wpływów pewnych cech funkcji. Podpada ona pod kategorię selekcji dynamicznych, zachowujących, pokoleniowych i elitarnych.

Struktura zmodyfikowanych algorytmów genetycznych jest przedstawiona na rys. 4.1. W stosunku do klasycznych algorytmów genetycznych modyfikacja polega na tym, że w zmodyfikowanych algorytmach genetycznych nie wykonu-

```
procedure modGA
begin
  t ← 0
  nadaj wartość P(t)
  oceń P(t)
  while (not warunek zakończenia) do
  begin
    t ← t + 1
    wybierz rodziców z P(t − 1)
    wybierz martwych z P(t − 1)
    utwórz P(t): zreprodukuj rodziców
    oceń P(t)
  end
end
```

Rys. 4.1. Algorytm modGA

jemy kroku selekcji „wybierz $P(t)$ z $P(t-1)$", ale wybieramy niezależnie r (niekoniecznie różnych) chromosomów do reprodukcji i r (różnych) chromosomów do wyginięcia. Te selekcje wykonuje się, biorąc pod uwagę względne dopasowanie łańcuchów. Łańcuch oceniany lepiej od średniej ma większe szanse na wybranie do reprodukcji. Łańcuch oceniany gorzej od średniej ma większe szanse na wyginięcie. Po krokach „wybierz rodziców" i „wybierz ginących" w populacji znajdują się trzy (niekoniecznie rozdzielne) grupy łańcuchów:

- r (niekoniecznie różnych) łańcuchów przeznaczonych do reprodukcji (rodziców),
- dokładnie r łańcuchów do wyginięcia i
- pozostale łańcuchy, zwane *neutralnymi*.

Liczba łańcuchów neutralnych w pokoleniu (co najmniej $pop_size - 2r$ i co najwyżej $pop_size - r$) zależy od liczby wybranych różnych rodziców i od liczby tych samych łańcuchów w kategoriach „rodzic" i „ginący". Następnie tworzy się nową populację $P(t+1)$ składającą się z $pop_size - r$ łańcuchów (wszystkie łańcuchy z wyjątkiem wybranych do wyginięcia) i r potomków od r rodziców.

Tak jak go przedstawiono, algorytm ma jeden potencjalnie dyskusyjny krok: jak wybrać r chromosomów do wyginięcia. Oczywiście chcielibyśmy wykonać ten wybór w taki sposób, aby mocniejsze chromosomy miały mniejsze szanse wyginięcia. Uzyskujemy to, zmieniając metodę tworzenia nowej populacji $P(t+1)$ na następującą:

krok 1: Wybierz r rodziców z $P(t)$. Każdy wybrany chromosom (lub raczej każda z wybranych kopii pewnego chromosomu) jest zaznaczany jako przeznaczony do jednej ustalonej operacji genetycznej.
krok 2: Wybierz $pop_size - r$ różnych chromosomów z $P(t)$ i skopiuj je do $P(t+1)$.
krok 3: Niech r chromosomów rodzicielskich utworzy dokładnie r potomków.
krok 4: Wprowadź tych r potomków do populacji $P(t+1)$.

W powyższej selekcji kroki 1 i 2 wykonuje się, biorąc pod uwagę dopasowanie chromosomów (stochastyczna uniwersalna metoda próbkowania).

Między różnymi procedurami selekcji omawianymi wcześniej a opisaną powyżej zachodzi kilka istotnych różnic. Po pierwsze zarówno rodzice, jak i potomkowie mają w niej dużą szansę znalezienia się w nowym pokoleniu. Oceniane powyżej średniej osobniki mają dużą szansę wybrania na rodziców (krok 1), a jednocześnie wybrania do nowej populacji w ramach $pop_size - r$ elementów (krok 2). Jeżeli tak się stanie, to jeden (lub więcej) z ich potomków zajmie jedno z r wakujących miejsc. Po drugie stosujemy operatory genetyczne na całych osobnikach w przeciwieństwie do indywidualnych bitów, jak w klasycznej mutacji. Prowadzi to do jednakowego traktowania wszystkich operatorów używanych w programie ewolucyjnym (program ewolucyjny GENOCOP

używa sześciu operatorów „genetycznych", patrz rozdz. 7). Tak więc, jeżeli używamy trzech operatorów (na przykład mutacji, krzyżowania i inwersji), to niektórzy rodzice ulegną mutacji, inni krzyżowaniu, a reszta inwersji.

Zmodyfikowane podejście ma podobne właściwości teoretyczne jak klasyczny algorytm genetyczny. Możemy przepisać równanie wzrostu (3.3) z rozdz. 3 w postaci

$$\xi(S, t+1) \geqslant \xi(S, t) \cdot p_s(S) \cdot p_g(S) \tag{4.1}$$

gdzie $p_s(S)$ reprezentuje prawdopodobieństwo przeżycia schematu S, a $p_g(S)$ prawdopodobieństwo wzrostu schematu S. Wzrost schematu S następuje w czasie etapu selekcji (faza wzrostu), gdzie kilka kopii schematów ocenianych powyżej średniej przepisuje się do nowej populacji. Prawdopodobieństwo $p_g(S)$ wzrostu schematu S wynosi $p_g(S) = eval(S, t)/\overline{F(t)}$. Dla schematów ocenianych powyżej średniej zachodzi $p_g(S) > 1$. Następnie schemat wybranego chromosomu musi przeżyć operatory genetyczne: krzyżowanie i mutację (faza kurczenia). Jak przedstawiono w rozdz. 3, prawdopodobieństwo $p_s(S)$ przeżycia schematu S wynosi

$$p_s(S) = 1 - p_c \cdot \delta(S)/(m-1) - p_m \cdot o(S) < 1$$

Ze wzoru (4.1) wynika, że dla krótkich schematów o niskim rzędzie $p_s(S) \cdot p_g(S) > 1$. Z tego powodu schematy te wygrywają wykładniczo rosnącą liczbę losowań w kolejnych pokoleniach. To samo zachodzi w zmodyfikowanej wersji algorytmu. Oczekiwana liczba chromosomów schematu S w zmodyfikowanym algorytmie genetycznym jest także iloczynem liczby chromosomów w starej populacji $\xi(S, t)$, prawdopodobieństwa przeżycia ($p_s(S) < 1$) i prawdopodobieństwa wzrostu $p_g(S)$. Jedyna różnica polega na innej interpretacji faz wzrostu i kurczenia i ich wzajemnej kolejności. W zmodyfikowanym algorytmie genetycznym faza kurczenia zachodzi wcześniej i w jej trakcie wybiera się do nowej populacji $n - r$ chromosomów. Prawdopodobieństwo przeżycia definiuje się jako ułamek chromosomów ze schematu S nie przeznaczonych do wyginięcia. Faza wzrostu zachodzi później i przejawia się pojawieniem się r nowych potomków. Prawdopodobieństwo wzrostu $p_g(S)$ schematu S jest prawdopodobieństwem, że schemat S poszerzy się o nowe potomstwo uzyskane z r rodziców. I znowu, dla krótkich schematów o niskim rzędzie zachodzi $p_s(S) \cdot p_g(S) > 1$ i takie schematy wygrywają wykładniczo rosnącą liczbę losowań w kolejnych pokoleniach.

Jednym z pomysłów w zmodyfikowanym algorytmie genetycznym jest lepsze wykorzystanie dostępnej pamięci, związane z liczebnością populacji. Nowy algorytm unika pozostawiania dokładnych wielokrotnych kopii tego samego chromosomu w nowej populacji (co ciągle może się zdarzyć przez przypadek lub w inny sposób, ale jest mało prawdopodobne). Algorytm klasyczny jest bardzo wrażliwy na tworzenie się takich wielokrotnych kopii. Ponadto takie

wielokrotne pojawianie się superosobników tworzy możliwość reakcji łańcuchowej. Tworzy się szansa na jeszcze większą liczbę takich dokładnych kopii w następnej populacji itd. W ten sposób ograniczona liczebnie populacja może w rzeczywistości zawierać ciągle malejącą liczbę różnych chromosomów. Zmniejszenie przestrzeni przeszukiwań pogarsza działanie algorytmu. Zauważmy, że w podstawach teoretycznych algorytmów genetycznych zakłada się nieskończoną liczebność populacji. W zmodyfikowanych algorytmach genetycznych możemy mieć wielu członków rodziny chromosomu, ale każdy z nich jest inny (przez rodzinę rozumiemy potomków tego samego rodzica).

Jako przykład rozważmy chromosom z oczekiwaną liczbą wystąpień $p = 3$ w $P(t + 1)$. Przypuśćmy także, że klasyczny algorytm genetyczny ma prawdopodobieństwo krzyżowania i mutacji $p_c = 0{,}3$ i $p_m = 0{,}003$, co spotyka się dosyć często. Po selekcji, a przed reprodukcją, będzie dokładnie $p = 3$ kopie tego chromosomu w $P(t + 1)$. Po reprodukcji, przy założonej długości chromosomu $m = 20$, oczekiwana liczba dokładnych kopii tego chromosomu pozostających w $P(t + 1)$ będzie wynosiła $p \cdot (1 - p_c - p_m \cdot m) = 1{,}92$. A więc możemy z zapasem powiedzieć, że następna populacja będzie zawierała dwie dokładne kopie tego chromosomu, zmniejszając tym samym liczbę różnych chromosomów w populacji.

Zmiana w zmodyfikowanym algorytmie genetycznym opiera się na pomyśle zapożyczonym z modelu ze współczynnikiem zatłoczenia [82], gdzie nowo utworzony chromosom zmienia któregoś ze starszych. Różnica polega na tym, że w modelu ze współczynnikiem zatłoczenia ginący chromosom wybiera się z tych, które przypominają nowego, podczas gdy w zmodyfikowanym algorytmie genetycznym ginące chromosomy to te, które mają mniejsze dopasowania.

Zmodyfikowane algorytmy genetyczne dla małych wartości parametru r należą do klasy algorytmów genetycznych z ustalonym stanem [394], [382]. Główna różnica między algorytmami genetycznymi a algorytmami genetycznymi z ustalonym stanem polega na tym, że w tych ostatnich zmienia się tylko kilku członków populacji (w każdym pokoleniu). Występuje także podobieństwo między zmodyfikowanym algorytmem genetycznym a systemem klasyfikujący (rozdz. 12). Składniki genetyczne systemu klasyfikacji zmieniają populację tak mało, jak to jest tylko możliwe. W zmodyfikowanych algorytmach genetycznych możemy regulować tę zmianę za pomocą parametru r, który określa liczbę chromosomów przeznaczonych do reprodukcji i liczbę chromosomów przeznaczonych do wyginięcia. W zmodyfikowanym algorytmie genetycznym $pop_size - r$ chromosomów przenosi się do nowej populacji bez żadnej zmiany. W szczególności, przy $r = 1$, tylko jeden chromosom podlega zmianie w każdym pokoleniu. Ostatnio Mühlenbein [289] zaproponował płodzące algorytmy genetyczne, w których r najlepszych osobników jest wybieranych i łączonych w pary losowo tak długo, aż liczba potomków osiągnie liczebność populacji. Pokolenie potomków zamienia populację rodziców, a najlepsze osobniki znalezione do tej pory pozostają w populacji.

4.2. Cechy funkcji

Zmodyfikowany algorytm genetyczny zawiera inny mechanizm tworzenia nowej populacji ze starej. Jednak wydaje się, że można by było znaleźć dodatkowe sposoby przezwyciężania kłopotów związanych z cechami optymalizowanej funkcji. Z upływem czasu wyodrębniły się trzy główne kierunki. Jeden z nich zapożyczył z metody symulowanego wyżarzania zmianę entropii systemu (patrz np. [359], gdzie autorzy sterują szybkością zbieżności populacji za pomocą operatora termodynamicznego, który zależy od temperatury).

Inny kierunek jest oparty na przyporządkowywaniu prób reprodukcyjnych zgodnie z uporządkowaniem, a nie faktyczną wartością oceny (jak to przedstawiono w poprzednim punkcie), gdyż uporządkowanie automatycznie wprowadza jednostajne skalowanie populacji.

Ostatni kierunek skupia się na próbie ustalenia samej funkcji przez wprowadzenie mechanizmu skalowania. Za Goldbergiem [154, s. 122-124] podzielimy takie mechanizmy na trzy części.

1. *Skalowanie liniowe.* W tej metodzie faktyczne dopasowanie chromosomu ulega następującemu przeskalowaniu $f'_i = a * f_i + b$. Parametry a i b są zazwyczaj wybierane tak, aby średnie dopasowanie było odwzorowane w siebie, a najlepsze dopasowanie było określoną wielokrotnością średniego dopasowania. Ten mechanizm, choć niewątpliwie silny, może wprowadzać ujemne oceny, które trzeba umieć później zinterpretować. Dodatkowo parametry a i b są zazwyczaj ustalone na całe trwanie populacji i nie zależą od zadania.
2. *Obcinanie na poziomie odchylenia standardowego.* Ta metoda jest pomyślana jako ulepszenie skalowania liniowego, zarówno w celu rozwiązania problemu negatywnych ocen, jak też wprowadzenia informacji uzależnionej od zadania do samego przekształcenia. Nowe dopasowanie jest tu obliczane według wzoru $f'_i = f_i + (\bar{f} - c * \sigma)$, gdzie c jest małą liczbą całkowitą (zazwyczaj cyfrą z zakresu 1 do 5), a σ jest odchyleniem standardowym populacji. Możliwe ujemne oceny f' są zamienione na zerowe.
3. *Skalowanie zgodne z prawem potęgowym.* W tej metodzie początkowe dopasowanie jest podnoszone do potęgi $f'_i = f_i^k$, przy pewnym k bliskim 1. Parametr k skaluje funkcję f. W niektórych pracach [138] twierdzi się, że wybór k powinien zależeć od zadania. W tej samej pracy autor użył $k = 1{,}005$ i uzyskał pewną poprawę w obliczeniach.

Najbardziej zauważalnym problemem związanym z charakterystyką rozważanej funkcji są różnice we względnym dopasowaniu. Jako przykład rozważmy dwie funkcje: $f_1(x)$ i $f_2(x) = f_1(x) + const$. Ponieważ są one właściwie takie same (to znaczy mają to samo optimum), powinny dla nich wystąpić te same trudności w optymalizacji. Jednak, jeżeli $const \gg \bar{f}_1(x)$, to zbieżność dla funkcji $f_2(x)$ będzie dużo wolniejsza niż dla funkcji $f_1(x)$. Faktycznie, w eks-

tremalnym przypadku, do optymalizacji drugiej funkcji będzie użyte całkowicie przypadkowe przeszukiwanie. Takie postępowanie może być tolerowane we wczesnych etapach życia populacji, ale będzie wyniszczające później. Dla odmiany zbieżność dla $f_1(x)$ może zachodzić zbyt szybko, wpychając algorytm w lokalne optimum.

Dodatkowo, ze względu na ograniczoną liczebność populacji, zachowanie się algorytmu genetycznego może zmieniać się między kolejnymi obliczeniami. Jest to spowodowane błędami związanymi ze skończonymi punktami próbnymi. Rozważmy funkcję $f_3(x)$ oraz punkt $x_i^t \in P(t)$ w pobliżu pewnego optimum lokalnego, przy czym $f(x_i^t)$ jest dużo większe od średniego dopasowania $\bar{f}(x^t)$ (tzn. x_i^t jest superosobnikiem). Załóżmy dalej, że nie ma x_j^t leżącego w pobliżu globalnego maksimum. Tak może być w przypadku bardzo niegładkiej funkcji. W takim przypadku zachodzi szybka zbieżność do lokalnego optimum. Z tego powodu populacja $P(t+1)$ staje się przesycona elementami z otoczenia tego rozwiązania, co zmniejsza szanse ogólnego badania potrzebnego do poszukiwania innych optimów. O ile takie zachowanie jest dopuszczalne w późniejszych etapach ewolucji, a nawet pożądane w całkiem końcowych etapach, to bardzo przeszkadza w etapach wczesnych. Ponadto zazwyczaj stare populacje (w dalekich etapach algorytmu) są nasycone chromosomami o podobnym dopasowaniu, ponieważ wszystkie one są blisko spokrewnione (przez proces łączenia w pary). Wobec tego, używając tradycyjnej metody selekcji, punkty próbne faktycznie stają się przypadkowe. Takie zachowanie jest dokładnie przeciwne najbardziej pożądanemu, w którym wpływ spokrewnionych chromosomów na proces selekcji jest zmniejszany w początkowych etapach i zwiększany w późniejszych.

Jeden z najlepiej znanych systemów, GENESIS 1.2ucsd ma dwa parametry do sterowania przeszukiwaniem w zależności od charakterystyki optymalizowanej funkcji: skalowane okno i współczynnik obcinania na poziomie odchylenia standardowego. Przy minimalizacji funkcji zazwyczaj funkcja oceny *eval* jest

$$eval(x) = F - f(x)$$

gdzie F jest stałą, taką że $F > f(x)$ dla wszystkich x. Jak to omawiano wcześniej, zły wybór F może mieć nieszczęśliwy wpływ na przeszukiwanie, a ponadto można nie znać F zawczasu. Okno skalujące W w GENESIS 1.2ucsd pozwala użytkownikowi ustalać, jak często zmieniać F. Jeżeli $W > 0$, to system podstawia na F największą wartość $f(x)$, która pojawiła się w ostatnich W pokoleniach. Wartość $W = 0$ oznacza okno nieskończone, to znaczy $F = \max\{f(x)\}$ po wszystkich ocenach. Jeżeli $W < 0$, to użytkownik może użyć innej metody przedstawionej wcześniej: obcinania na poziomie odchylenia standardowego.

Należy też zwrócić uwagę na znaczenie warunku zakończenia przyjętego w algorytmie. W najprostszym warunku zakończenia sprawdza się bieżący numer pokolenia. Przeszukiwanie kończy się, gdy liczba pokoleń przekroczy

wcześniej podaną stałą. Zgodnie z rys. 0.1 (we wprowadzeniu) taki warunek zakończenia wyraża się przez „$t \geqslant T$" dla pewnej stałej T. W wielu wersjach programów ewolucyjnych nie wszystkie osobniki muszą być oceniane. Niektóre z nich przechodzą z jednego pokolenia do drugiego bez żadnych zmian. W takich przypadkach mogłoby być istotne (dla porównania z innymi, tradycyjnymi algorytmami) obliczanie liczby wywołań funkcji (zazwyczaj ta liczba jest proporcjonalna do liczby pokoleń) i kończenie przeszukiwania, kiedy liczba wywołań przekroczy wcześniej zadaną stałą.

Jednak w powyższych warunkach zakończenia zakłada się wiedzę użytkownika o cechach funkcji, co wpływa na czas przeszukiwania. W wielu przypadkach jest dosyć trudno stwierdzić, że całkowita liczba pokoleń (lub obliczeń funkcji) powinna wynosić, powiedzmy, 10 000. Wygląda na to, że byłoby dużo lepiej, gdyby algorytm kończył przeszukiwanie, gdy szansa na znaczącą poprawę jest niewielka.

Są dwie główne grupy warunków zakończenia, w których przy podjęciu decyzji o zakończeniu bierze się pod uwagę stan przeszukiwania. W jednej grupie rozważa się strukturę chromosomu (genotyp), w drugiej – znaczenie poszczególnego chromosomu (fenotyp). W warunkach zakończenia z pierwszej grupy mierzy się zbieżność populacji przez sprawdzanie liczby ustalonych alleli, przy czym allel uważa się za ustalony, jeżeli pewny wcześniej ustalony procent populacji ma tę samą (lub zbliżoną – dla niebinarnych reprezentacji) wartość tego allela. Jeżeli liczba ustalonych alleli przekroczy pewien procent wszystkich alleli, to przeszukiwanie kończy się. W warunkach zakończenia z drugiej grupy mierzy się postępy algorytmu w zadanej wcześniej liczbie pokoleń. Jeżeli postęp jest mniejszy od pewnego epsilonu (będącego parametrem metody), to przeszukiwanie kończy się.

4.3. Algorytmy genetyczne z odwzorowaniem zwężającym

Zbieżność algorytmów genetycznych jest jednym z najbardziej wyzywających zagadnień teoretycznych w dziedzinie obliczeń ewolucyjnych. Kilka osób badało to zagadnienie z różnych punktów widzenia. Goldberg i Segrest [163] przedstawili analizę algorytmów genetycznych za pomocą skończonych łańcuchów Markowa (populacja o skończonej liczebności, tylko reprodukcja i mutacja). Davis i Principe [80] badali możliwość przeniesienia teoretycznych podstaw algorytmu symulowanego wyżarzania na modele algorytmów genetycznych w postaci łańcuchów Markowa. Eiben, Aarts i Van Hee [98] zaproponowali abstrakcyjny algorytm genetyczny, w którym ujednolicono algorytmy genetyczne i symulowanego wyżarzania. Przedstawili oni analizę takich abstrakcyjnych algorytmów genetycznych za pomocą łańcuchów Markowa i podali wa-

runki, przy których ten proces ewolucyjny znajduje optimum z prawdopodobieństwem 1. Kingdom [224] badał punkty startowe, zbieżność i klasę zadań, które trudno jest rozwiązać za pomocą algorytmów genetycznych. Uogólnił on pojęcie rywalizujących schematów i podał prawdopodobieństwo ich zbieżności. Kilku badaczy rozważało także różne definicje zadań zawodnych [154]. Ostatnio Rudolph [334] dowiódł, że klasyczne algorytmy genetyczne nigdy nie zbiegają się do optimum globalnego, natomiast zbiegają do niego wersje zmodyfikowane, które zachowują najlepsze rozwiązania w populacji (to znaczy model elitarny).

Jeden z możliwych sposobów badania zbieżności algorytmu genetycznego korzysta z twierdzenia Banacha o punkcie stałym [386]. Podaje się w nim intuicyjne wyjaśnienie zbieżności algorytmów genetycznych (bez modelu elitarnego). Wymaga się jedynie, aby kolejne populacje były lepsze (nie dotyczy to wymagania poprawy najlepszych osobników). Twierdzenie Banacha o punkcie stałym dotyczy odwzorowań zwężających w przestrzeniach metrycznych. Mówi ono, że każde takie odwzorowanie f ma jednoznacznie określony punkt stały, to znaczy element x taki, że $f(x) = x$. Metody korzystające z punktu stałego są ogólnie uznawane za mocne narzędzia określania semantyki obliczeń. Na przykład semantyka znaczeniowa programu lub obliczeń jest zazwyczaj określana jako najmniejszy punkt stały odwzorowania ciągłego zdefiniowanego na odpowiedniej pełnej siatce. Jednak, inaczej niż w tradycyjnej semantyce znaczeniowej, są to przestrzenie metryczne pozwalające w prosty i naturalny sposób wyrażać semantykę algorytmów genetycznych. Algorytmy genetyczne można przedstawić jako transformację populacji. Przypuśćmy więc, że potrafimy określić takie przestrzenie metryczne, w których te transformacje są zwężające. W takim przypadku otrzymujemy semantykę algorytmów genetycznych jako punkty stałe tych transformacji. Ponieważ każda taka transformacja ma jednoznacznie określony punkt stały, więc otrzymujemy jako prosty wniosek zbieżność algorytmów genetycznych.

Intuicyjnie przestrzeń metryczna jest uporządkowaną parą zbioru i funkcji, która pozwala mierzyć odległość między jakąkolwiek parą elementów tego zbioru. Odwzorowanie f określone na elementach takiego zbioru jest zwężające, jeżeli odległość między $f(x)$ i $f(y)$ jest mniejsza[1] niż odległość między x a y.

Zdefiniujemy teraz podstawowe pojęcia w sposób formalny. Określmy zbiór liczb rzeczywistych przez R. Zbiór S wraz z funkcją δ: $S \times S \to R$ jest *przestrzenią metryczną*, jeżeli dla dowolnych elementów $x, y \in S$ są spełnione następujące warunki:

- $\delta(x, y) \geqslant 0$ i $\delta(x, y) = 0$ wtedy i tylko wtedy, gdy $x = y$
- $\delta(x, y) = \delta(y, x)$
- $\delta(x, y) + \delta(y, z) \geqslant \delta(x, z)$

[1] Faktycznie, jak to zobaczymy później, określenie *mniejsza* jest tu trochę mocniejsze niż zwykle.

Funkcja δ jest nazywana *odległością*. Zazwyczaj oznaczamy przestrzeń metryczną przez $\langle S, \delta \rangle$.

Niech $\langle S, \delta \rangle$ będzie przestrzenią metryczną i niech $f: S \to S$ będzie odwzorowaniem. Powiemy, że f jest *zwężające*, jeżeli istnieje stała $\varepsilon \in [0, 1)$ taka, że dla wszystkich $x, y \in S$

$$\delta(f(x), f(y)) \leqslant \varepsilon * \delta(x, y)$$

Aby sformułować twierdzenie Banacha, musimy zdefiniować pojęcie przestrzeni metrycznych zupełnych. Mówimy, że ciąg $p_0, p_1, \ldots$ elementów przestrzeni metrycznej $\langle S, \delta \rangle$ jest *ciągiem Cauchy'ego*, jeżeli dla każdego $\varepsilon > 0$ istnieje k takie, że dla wszystkich $m, n > k$ zachodzi $\delta(p_m, p_n) < \varepsilon$. Mówimy, że przestrzeń metryczna jest zupełna, jeżeli każdy ciąg Cauchy'ego $p_0, p_1, \ldots$ ma granicę $p = \lim_{n \to \infty} p_n$.

Jesteśmy teraz gotowi, aby sformułować twierdzenie Banacha. Dowód tego twierdzenia był po raz pierwszy podany w [25] i można go znaleźć w większości podręczników topologii (np. [94], s. 60).

Twierdzenie [25]
Niech $\langle S, \delta \rangle$ będzie przestrzenią metryczną zupełną i niech $f: S \to S$ będzie odwzorowaniem zwężającym. Wtedy f ma jednoznaczny punkt stały $x \in S$ taki, że dla każdego $x_0 \in S$

$$x = \lim_{i \to \infty} f^i(x_0)$$

przy czym $f^0(x_0) = x_0$ oraz $f^{i+1}(x_0) = f(f^i(x_0))$

Twierdzenie Banacha ma bardzo intuicyjne zastosowanie w algorytmach genetycznych. Mianowicie, jeżeli utworzymy przestrzeń metryczną S w taki sposób, że jej elementy są populacjami, każde odwzorowanie zwężające f ma jednoznaczny punkt stały, który zgodnie z twierdzeniem Banacha można uzyskać przez iterowanie f, poczynając od *dowolnie wybranej* populacji początkowej $P(0)$. A więc, jeżeli określimy odpowiednią przestrzeń metryczną, w której algorytmy genetyczne są zwężające, to będziemy mogli wykazać zbieżność tych algorytmów do tego samego punktu stałego, niezależnie od wyboru początkowej populacji. Wykażemy, że taka konstrukcja jest możliwa dla nieznacznie zmodyfikowanego algorytmu genetycznego zwanego algorytmem genetycznym z odwzorowaniem zwężającym.

Bez straty ogólności założymy, że mamy do czynienia z zadaniem maksymalizacji, to znaczy zadaniem, w którym rozwiązanie x_i jest lepsze od rozwiązania x_j wtedy i tylko wtedy, gdy $eval(\bar{x}_i) > eval(\bar{x}_j)$.

Założymy też, że liczebność populacji $pop_size = n$ jest ustalona, to znaczy każda populacja składa się z n osobników, $P = \{\bar{x}_1, ..., \bar{x}_n\}$. Ponadto przyjmiemy funkcję oceniającą *Eval* dla populacji P, na przykład w postaci

$$Eval(P) = \frac{1}{n} \sum_{\bar{x}_i \in P} eval(\bar{x}_i)$$

gdzie *eval* jest „dopasowaniem" osobnika $\bar{x}_i$ z populacji P.

Zbiór S składa się ze wszystkich możliwych populacji, to znaczy do S należą wszystkie wektory $\{\bar{x}_1, ..., \bar{x}_n\}$.

Zdefiniujmy funkcję (odległość) δ: $S \times S \to R$ i odwzorowanie zwężające f: $S \to S$ w przestrzeni metrycznej $\langle S, \delta \rangle$. Odległość δ w przestrzeni metrycznej populacji S można zdefiniować następująco:

$$\delta(P_1, P_2) = \begin{cases} 0 & \text{jeżeli } P_1 = P_2 \\ |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)| & \text{w odwrotnym przypadku} \end{cases}$$

gdzie M jest ograniczeniem górnym funkcji *eval* w rozważanej dziedzinie, to znaczy $eval(\bar{x}) \leqslant M$ dla wszystkich osobników $\bar{x}$ (w rezultacie $Eval(P) \leqslant M$ dla wszystkich możliwych populacji). Rzeczywiście

- $\delta(P_1, P_2) \geqslant 0$ dla każdej populacji P_1 i P_2, ponadto $\delta(P_1, P_2) = 0$ wtedy i tylko wtedy, gdy $P_1 = P_2$
- $\delta(P_1, P_2) = \delta(P_2, P_1)$ oraz
- $\delta(P_1, P_2) + \delta(P_2, P_3) = |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)| + |1 + M - Eval(P_2)| + |1 + M - Eval(P_3)| \geqslant |1 + M - Eval(P_1)| + |1 + M - Eval(P_3)| = \delta(P_1, P_3)$

W konsekwencji $\langle S, \delta \rangle$ jest przestrzenią metryczną.

Ponadto przestrzeń metryczna $\langle S, \delta \rangle$ jest zupełna. Jest tak, ponieważ dla każdego ciągu Cauchy'ego populacji $P_1, P_2, ...$ istnieje takie k, że dla wszystkich $n > k$ zachodzi $P_n = P_k$. Oznacza to, że wszystkie ciągi Cauchy'ego P_i mają granicę dla $i \to \infty$.[1)]

Jesteśmy teraz gotowi rozważyć przekształcenie zwężające f: $S \to S$, które jest po prostu pojedynczą iteracją w przebiegu[2)] algorytmu genetycznego (patrz rys. 4.3) przy założeniu, że nastąpiła poprawa (wyrażona przez funkcję *Eval*) populacji $P(t + 1)$ w stosunku do populacji $P(t)$. Inaczej mówiąc, iteracja t algorytmu genetycznego będzie przyjęta za operator przekształcenia zwężającego f wtedy i tylko wtedy, gdy $Eval(P(t)) < Eval(P(t + 1))$. Jeżeli nie ma poprawy, nie bierzemy takiej iteracji pod uwagę, to znaczy powtarzamy proces selekcji i rekombinacji od nowa.

[1)] Zauważmy, że w przypadku algorytmów genetycznych zasadniczo mamy do czynienia ze skończonymi przestrzeniami metrycznymi, gdyż w każdej populacji jest tylko skończona liczba elementów. Wobec tego wymaganie Banacha zupełności przestrzeni jest w tym przypadku zawsze spełnione. Jednak w naszym przypadku $\langle S, \delta \rangle$ jest zupełna dla każdego zbioru S.

[2)] Przez przebieg rozumiemy tutaj każdy zaobserwowany ciąg obliczeń.

```
procedure CM-GA
begin
  t = 0
  nadaj wartość P(t)
  oceń P(t)
  while (not warunek zakończenia) do
  begin contractive mapping f(P(t)) → P(t + 1)
    t = t + 1
    wybierz P(t) z P(t − 1)
    wykonaj operacje rekombinacji na P(t)
    oceń P(t)
    if Eval(P(t − 1)) ⩾ Eval(P(t))
      then t = t − 1
  end
end
```

Rys. 4.2. Algorytm genetyczny z odwzorowaniem zwężającym

Struktura takiego zmodyfikowanego algorytmu genetycznego (algorytmu genetycznego z przekształceniem zwężającym) jest podana na rys. 4.2.

Zmodyfikowana iteracja algorytmu genetycznego z przekształceniem zwężającym rzeczywiście spełnia wymagania przekształcenia zwężającego. Jest oczywiste, że gdy iteracja f: $P(t) \to P(t+1)$ poprawia ocenę populacji wyrażoną funkcją *Eval*, to znaczy jeżeli

$$Eval(P_1(t)) < Eval(f(P_1(t))) = Eval(P_1(t+1)) \qquad \text{oraz}$$

$$Eval(P_2(t)) < Eval(f(P_2(t))) = Eval(P_2(t+1))$$

to

$$\delta(f(P_1(t)), f(P_2(t))) = |1 + M - Eval(f(P_1(t)))| + |1 + M - Eval(f(P_2(t)))| <$$

$$< |1 + M - Eval(P_1(t))| + |1 + M - Eval(P_2(t))| = \delta(P_1(t), P_2(t))$$

Ponadto, ponieważ zawsze mamy do czynienia z jakimś rozwiązaniem komputerowym algorytmu, poprawa jest nie mniejsza niż najmniejsza liczba rzeczywista w słowie komputerowym.

Podsumowując, algorytm genetyczny z przekształceniem zwężającym spełnia założenia twierdzenia Banacha o punkcie stałym: przestrzeń populacji $\langle S, \delta \rangle$ jest przestrzenią metryczną zupełną, a iteracja f: $P(t) \to P(t+1)$ (która poprawia ocenę populacji wyrażoną funkcją *Eval*) jest zwężająca. W rezultacie

$$P^* = \lim_{i \to \infty} f^i(P(0))$$

czyli algorytm genetyczny z przekształceniem zwężającym zbiega do populacji P^*, która jest jednoznacznym punktem stałym w przestrzeni wszystkich populacji.

Oczywiście P^* jest globalnym optimum. Zauważmy, że funkcja *Eval* była zdefiniowana przez

$$Eval(P) = \frac{1}{n} \sum_{\bar{x}_i \in P} eval(\bar{x}_i)$$

co oznacza, że punkt stały P^* jest osiągany, gdy wszystkie osobniki w populacji mają tę samą wartość (globalne maksimum). Ponadto P^* nie zależy od populacji początkowej $P(0)$.

Interesujący problem pojawia się wtedy, kiedy funkcja oceniająca *eval* ma więcej niż jedno maksimum. W takim przypadku algorytmy genetyczne z odwzorowaniem zwężającym faktycznie nie tworzą odwzorowania zwężającego, gdyż dla optymalnych populacji P_1, P_2

$$\delta(f(P_1), f(P_2)) = \delta(P_1, P_2)$$

Można jednak wykazać w tym przypadku, że algorytmy genetyczne z odwzorowaniem zwężającym zbiegają do jednej z możliwych populacji optymalnych. Wynika to z faktu, że każdy przebieg algorytmu zbiega do jednej z optymalnych populacji.

Na pierwszy rzut oka ten wynik wydaje się być dziwny. W przypadku algorytmu genetycznego z odwzorowaniem zwężającym wybór populacji początkowej może wpłynąć tylko na szybkość zbieżności. Zaproponowany algorytm genetyczny z odwzorowaniem zwężającym (na podstawie twierdzenia Banacha) powinien zawsze zbiegać się (w nieskończonym czasie). Jednak jest możliwe, że (w pewnym kroku algorytmu) nie można zaakceptować nowej populacji przez dłuższy czas i algorytm zapętla się, próbując znaleźć nową populację $P(t)$. Innymi słowy, operatory mutacji i krzyżowania zastosowane do szczególnej suboptymalnej populacji nie mogą wytworzyć „lepszej" populacji i algorytm zapętla się, próbując wykonać następny *zbiegający się* krok. Odległość δ między populacjami oraz funkcja oceniająca *Eval* były tu wybrane możliwie najprościej. Wybór ten może jednak wpłynąć na szybkość zbieżności i wydaje się zależeć od zastosowania.

4.4. Algorytmy genetyczne ze zmienną liczebnością populacji

Liczebność populacji jest jednym z najważniejszych parametrów do wyboru przez użytkownika algorytmów genetycznych, w wielu zastosowaniach wprost krytycznym. Jeżeli liczebność populacji jest zbyt mała, algorytmy genetyczne mogą zbiegać się zbyt szybko. Jeżeli jest zbyt duża, mogą one niepotrzebnie zużywać zasoby komputerowe i czas oczekiwania na poprawę może być zbyt

długi. Jak przedstawialiśmy to wcześniej (p. 4.1), w procesie ewolucyjnym z przeszukiwaniem genetycznym trzeba brać pod uwagę dwa istotne wskaźniki: różnorodność populacji i napór selekcyjny. Oba te wskaźniki zależą od liczebności populacji.

Kilka osób rozważało zagadnienie liczebności populacji w algorytmach genetycznych pod różnym kątem. Grefenstette [169] użył metaalgorytmów genetycznych do sterowania parametrami innego algorytmu genetycznego (łącznie z liczebnością populacji i metodą selekcji). Goldberg [151], [153] przedstawił teoretyczną analizę optymalnej liczebności populacji. Studium wpływu parametrów sterujących (działających na bieżąco do optymalizacji funkcji) na przeszukiwanie genetyczne jest przedstawione w [343]. Wyniki eksperymentów z liczebnością populacji podano w [206] i [59]. Ostatnio Smith [362] zaproponował algorytm, w którym liczebność populacji jest ustalana przy uwzględnieniu prawdopodobieństwa błędu selekcji.

W tym punkcie omówimy algorytm genetyczny ze zmienną liczebnością populacji [12]. Algorytm ten nie zawiera żadnej odmiany mechanizmu selekcji rozważanego wcześniej (p. 4.1), ale wprowadza pojęcie „wieku" chromosomu, co jest równoważne liczbie pokoleń, przez które chromosom pozostaje „żywy". Tak więc wiek chromosomu zastępuje pojęcie selekcji i, ponieważ zależy on od dopasowania osobnika, wpływa na liczebność populacji w każdym kroku procesu. Wygląda również na to, że takie podejście jest bardziej „naturalne" niż mechanizm selekcyjny rozważony wcześniej. Zresztą proces starzenia jest dobrze znany w naturalnym środowisku.

Wygląda na to, że proponowana metoda zmiennej liczebności populacji jest podobna do pewnych strategii ewolucyjnych (rozdz. 8) i innych metod, w których potomkowie rywalizują z rodzicami o przeżycie. Jednak ważna różnica polega na tym, że w innych metodach liczebność populacji pozostaje stała, podczas gdy w algorytmach genetycznych ze zmienną liczebnością populacji zmienia się ona w czasie.

Dodatkowa motywacja dla tej pracy wynikała z następujących obserwacji: tylko kilka osób badało możliwość wprowadzenia w algorytmach genetycznych adaptowanych prawdopodobieństw w operatorach genetycznych [77], [115], [343], [367], [368]. W innych metodach, jak strategie ewolucyjne [349], użyto adaptowanych prawdopodobieństw już jakiś czas temu (przedyskutujemy krótko te aspekty w p. 4.6 i rozdz. 8). Wydaje się, że rozsądne jest założenie, że na różnych etapach procesu ewolucyjnego operatory będą miały inną wagę i że system powinien mieć możliwość samodostrajania częstotliwości i zakresu ich zmian.To samo powinno być prawdziwe dla liczebności populacji. Na różnych etapach procesu ewolucyjnego mogą być „optymalne" różne liczebności populacji. Dlatego jest istotne, aby eksperymentować z regułami heurystycznymi dostrajania liczebności populacji do potrzeb bierzącego etapu poszukiwań.

Algorytm genetyczny ze zmienną liczebnością populacji przekształca w chwili t populację $P(t)$ chromosomów. Podczas kroku rekombinacji tworzy się z $P(t)$ nową populację pomocniczą (jest to populacja potomków). Liczebność

populacji pomocniczej jest proporcjonalna do liczebności początkowej populacji, zawiera ona $AuxPopSize(t) = \lfloor PopSize(t)^* \rfloor \rho$ chromosomów (będziemy nazywali *ρ współczynnikiem reprodukcji*). Każdy chromosom z populacji może być wybrany do reprodukcji (tzn. do umieszczenia potomka w populacji pomocniczej) z jednakowym prawdopodobieństwem, *niezależnym* od wartości dopasowania. Potomków tworzy się za pomocą operatorów genetycznych (krzyżowania i mutacji) użytych do wybranych chromosomów. Ponieważ wybór chromosomów nie zależy od ich wartości dopasowania – co oznacza, że nie ma kroku selekcji jako takiego – wprowadzamy pojęcie *wieku* chromosomu i parametr *czasu życia*.

Struktura algorytmu genetycznego ze zmienną liczebnością populacji jest przedstawiona na rys. 4.3.

```
procedure GAVaPS
begin
   t = 0
   nadaj wartość P(t)
   oceń P(t)
   while (not warunek zakończenia) do
   begin
      t = t + 1
      podnieś wiek każdego osobnika o 1
      wykonaj operacje rekombinacji na P(t)
      oceń P(t)
      usuń z P(t) wszystkie osobniki o wieku większym od ich czasu życia
   end
end
```

Rys. 4.3. Algorytm genetyczny ze zmienną liczebnością populacji

Parametr czasu życia jest przyporządkowany raz każdemu chromosomowi podczas kroku oceny (członkom populacji pomocniczej, albo po wybraniu wszystkich chromosomów, albo po kroku mieszania). Pozostaje on stały (dla danego chromosomu) w czasie procesu ewolucyjnego, to znaczy od narodzin chromosomu do jego śmierci. Oznacza to, że dla „starych" chromosomów ich czasy życia *nie są* liczone ponownie. Śmierć chromosomu następuje, gdy jego wiek, to znaczy liczba pokoleń, w ciągu których chromosom pozostawał żywy (na początku ustawiana na zero), przekroczy jego czas życia. Inaczej mówiąc, czas życia chromosomu określa liczbę pokoleń algorytmu genetycznego ze zmienną liczebnością, w ciagu którego chromosom jest utrzymywany w populacji. Kiedy jego czas życia wyczerpie się, chromosom umiera. Tak więc liczebność populacji po jednej iteracji wynosi

$$PopSize(t + 1) = PopSize(t) + AuxPopSize(t) - D(t)$$

przy czym $D(t)$ jest liczbą chromosomów, które wyginęły w pokoleniu t.

Istnieje wiele możliwych strategii ustalania czasów życia. To pewne, że ustalanie stałej wartości (większej od jedności) niezależnie od jakichkolwiek statystyk związanych z przeszukiwaniem spowoduje wykładniczy wzrost liczebności populacji. Ponadto, ponieważ nie ma tu mechanizmu selekcji jako takiego, nie istnieje napór selekcyjny. Ustalanie więc stałej wartości czasu życia spowodowałoby słabe działanie algorytmu. Aby wprowadzić napór selekcyjny, należy uwzględnić bardziej wyrachowane obliczanie czasu życia. Strategie obliczania czasu życia powinny: (1) wzmacniać osobniki z dopasowaniem powyżej średniego (a w konsekwencji osłabiać osobniki z dopasowaniem poniżej średniego), (2) dostrajać liczebność populacji odpowiednio do bieżącego etapu przeszukiwania (w szczególności chronić przed wykładniczym wzrostem populacji i zmniejszać koszty symulacji). Wzmocnienie lepiej dopasowanych osobników powinno powodować większe niż średnie wprowadzanie ich potomków do populacji pomocniczej. Ponieważ prawdopodobieństwo wejścia każdego osobnika do mieszania genetycznego jest równe, zatem oczekiwana liczba potomków osobnika jest proporcjonalna do jego czasu życia (gdyż czas życia określa liczbę pokoleń, w których osobnik znajduje się w populacji). A więc osobniki o wartościach dopasowania powyżej średnich powinny być obdarowane większymi czasami życia. Przy wyliczaniu czasu życia należy wziąć pod uwagę stan przeszukiwania genetycznego. Z tego powodu stosujemy kilka miar stanu przeszukiwań: *AvgFit*, *MaxFit* i *MinFit* reprezentują odpowiednio średnie, największe i najmniejsze wartości dopasowań w bieżącej populacji, a *AbsFitMax* i *AbsFitMin* oznaczają największą i najmniejszą wartość dopasowania znalezioną dotychczas. Należy także podkreślić, że wyliczanie czasu życia powinno być łatwe obliczeniowo, aby oszczędzać zasoby komputerowe.

Mając na względzie powyższe uwagi, zaprogramowano kilka strategii wyliczania czasu życia i użyto ich w obliczeniach. Parametr czasu życia dla i-tego osobnika ($lifetime[i]$) można określać na podstawie (tak jak w poprzednim punkcie zakładamy, że mamy zadanie maksymalizacji z dodatnią funkcją oceniającą):

(1) przyporządkowania proporcjonalnego

$$\min(MinLT + \eta \cdot fitness[i]/AvgFit,\ MaxLT)$$

(2) przyporządkowania liniowego

$$MinLT + 2\eta \cdot (fitness[i] - AbsFitMin)/(AbsFitMax - AbsFitMin)$$

(3) przyporządkowania biliniowego

$$MinLT + \eta \cdot (fitness[i] - MinFit)/(AvgFit - minFit), \quad \text{jeżeli } AvgFit \geqslant fitness[i]$$

$$1/2(MinLT + MaxLT) + \eta \cdot (fitness[i] - AvgFit)/(MaxFit - AvgFit), \quad \text{jeżeli } AvgFit < fitness[i]$$

gdzie $MaxLT$ i $MinLT$ są odpowiednio największym i najmniejszym dopuszczalnym czasem życia (te wartości są podane jako parametry algorytmu genetycznego ze zmienną liczebnością), a $\eta = 1/2(MaxLT - MinLT)$.

Pierwsza strategia (przyporządkowanie proporcjonalne) pochodzi z idei selekcji ruletkowej. Czas życia poszczególnych osobników jest proporcjonalny do ich dopasowań (w ramach limitu *MinLT* i *MaxLT*). Jednak strategia ta ma poważną wadę – nie korzysta z informacji o „obiektywnej dobroci" osobnika, którą można uzyskać, porównując ich dopasowania z najlepszymi wartościami znalezionymi do tej pory. Ta uwaga motywuje zainteresowanie strategią liniową. W tej strategii czas życia jest wyliczany zgodnie z dopasowaniem osobnika w stosunku do najlepszej obecnie wartości. Jednak, jeżeli wielu osobników ma dopasowania równe lub w przybliżeniu równe wartości najlepszej, to taka strategia prowadzi do przyporządkowania długich czasów życia, zwiększając w ten sposób liczebność populacji. Na koniec, w strategii biliniowej próbuje się znaleźć kompromis między obiema poprzednimi strategiami. Powiększa ona różnice między czasami życia prawie najlepszych osobników, korzystając z informacji o średnim dopasowaniu, jednak uwzględniając też największe i najmniejsze dopasowanie znalezione dotychczas.

Algorytm genetyczny ze zmienną liczebnością populacji przetestowano na następujących funkcjach:

G1:	$-x\sin(10\pi x) + 1$	$-2{,}0 \leqslant x \leqslant 1{,}0$
G2:	$\text{integer}(8x)/8$	$0{,}0 \leqslant x \leqslant 1{,}0$
G3:	$x \cdot \text{sgn}(x)$	$-1{,}0 \leqslant x \leqslant 2{,}0$
G4:	$0{,}5 + (\sin^2\sqrt{x^2+y^2} - 0{,}5)/(1 + 0{,}001(x^2+y^2))^2$	$-100 \leqslant x \leqslant 100$

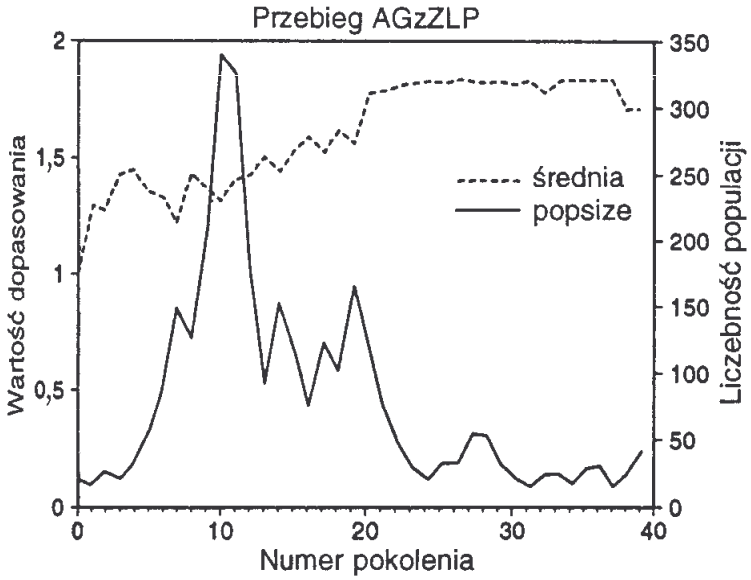
Funkcje te wybrano, aby pokryć szerokie spektrum możliwych do optymalizacji typów funkcji. Funkcje G1 i G2 są multimodalne z wieloma lokalnymi maksimami. Funkcji G2 nie da się optymalizować za pomocą metod gradientowych, bo nie jest ona różniczkowalna. Funkcja G3 reprezentuje zadanie, znane jako „zadanie zawodne" [154]. Maksymalizując taką funkcję, można łatwo znaleźć dwa kierunki wzrostu, przy czym ograniczenia są ustawione w taki sposób, że jedynie dla jednego z nich uzyskuje się globalne maksimum. Przy użyciu metody gradientowej z losowym wyznaczaniem punktów prowadzi to do częstego znajdowania maksimum lokalnego.

Działanie algorytmu genetycznego ze zmienną liczebnością populacji przetestowano i porównano z działaniem prostego algorytmu genetycznego Goldberga [154]. Metody kodowania zadania oraz operatory genetyczne były identyczne dla obu algorytmów (użyto prostego kodowania binarnego i dwóch operatorów: mutacji i krzyżowania jednopunktowego).

Do obliczeń zrobiliśmy następujące założenia. Początkowa liczebność populacji wynosiła 20. W przypadku prostego algorytmu genetycznego początkowa liczebność populacji pozostawała stała w całej symulacji. Współczynnik reprodukcji ρ ustalono na 0,4 (ten parametr nie ma znaczenia dla prostego algorytmu genetycznego). Współczynnik mutacji wynosił 0,015, a krzyżowania 0,65. Długość chromosomu była równa 20. We wszystkich obliczeniach przyjmowaliśmy, że minimalny i maksymalny czas życia były stałe i równe $MaxLT = 7$ i $MinLT = 1$.

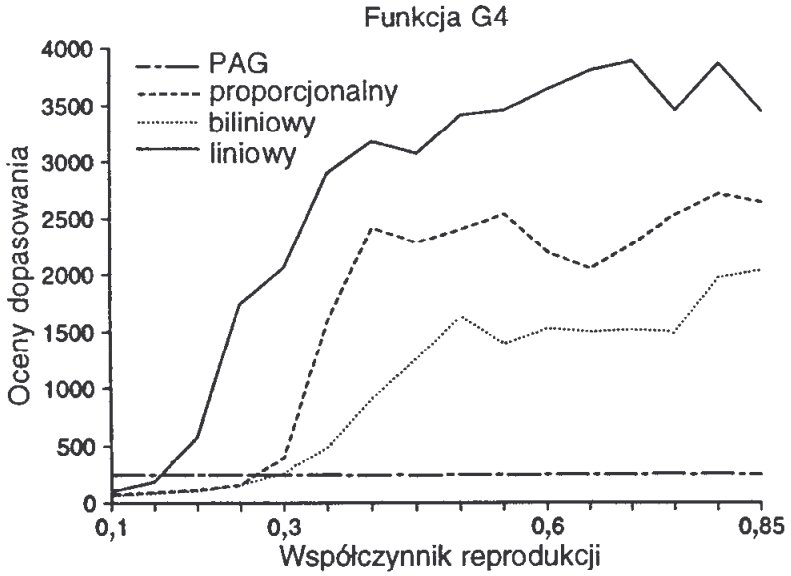
Do porównania prostego algorytmu genetycznego z algorytmem genetycznym ze zmienną liczebnością populacji wybrano dwa wskaźniki: koszt algorytmu, reprezentowany przez *evalnum* (średnią liczbę obliczania funkcji we wszystkich obliczeniach), i działanie, reprezentowane przez *avgmax* (średnią z największych wartości znalezionych we wszystkich obliczeniach). Oba algorytmy miały takie same warunki zakończenia. Kończyły one obliczenia, kiedy nie było żadnej poprawy liczonej w najlepszych wartościach znalezionych w kolejnych $conv = 20$ pokoleniach. Początkową populację ustalono w sposób losowy i wykonano 20 niezależnych przebiegów. Następnie miary działania i kosztu uśrednionio po tych 20 przebiegach, co dawało przedstawione wyniki. W czasie testowania wpływu pojedynczych parametrów na działanie i koszt, wartości omówionych powyżej parametrów utrzymywano stałe, z wyjątkiem tego, którego wpływ testowano.

Na rysunku 4.4 przedstawiono $PopSize(t)$ i średnie dopasowanie populacji w pojedynczym przebiegu algorytmu genetycznego ze zmienną liczebnością populacji dla funkcji G4 i biliniowym wyliczaniem czasu życia (podobne obserwacje można zrobić dla innych funkcji i innych strategii przyporządkowywania czasów życia). Kształt krzywej $PopSize(t)$ jest interesujący. Na początku, kiedy zmienność dopasowania jest stosunkowo duża, liczebność populacji rośnie. Oznacza to, że algorytm genetyczny ze zmienną liczebnością populacji dokonuje szerokiego przeszukiwania w celu znalezienia optimów. Jak tylko zostanie zlokalizowane otoczenie optimum, algorytm zaczyna zbiegać się i liczebność populacji ulega redukcji. Jednak nadal zachodzą poszukiwania możliwości poprawy. Kiedy taka możliwość się pojawi, zachodzi następna „eksplozja demograficzna", po której następuje ponownie etap zbieżności. Wygląda na to, że algorytm genetyczny ze zmienną liczebnością populacji prowadzi proces *samodostrajania* przez dobór liczebności populacji na każdym etapie procesu ewolucyjnego.

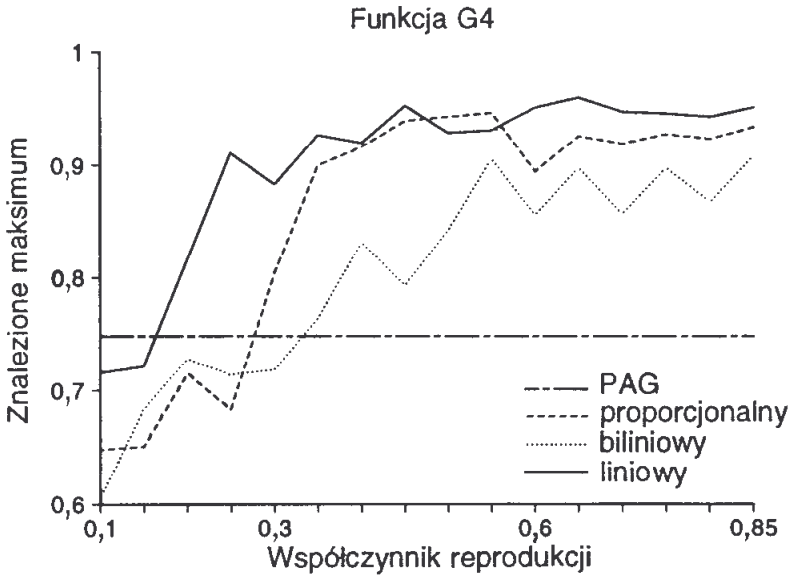


Rys. 4.4. $PopSize(t)$ oraz średnie dopasowanie populacji dla pojedynczego przebiegu algorytmu genetycznego ze zmienną liczebnością populacji (AGzZLP)

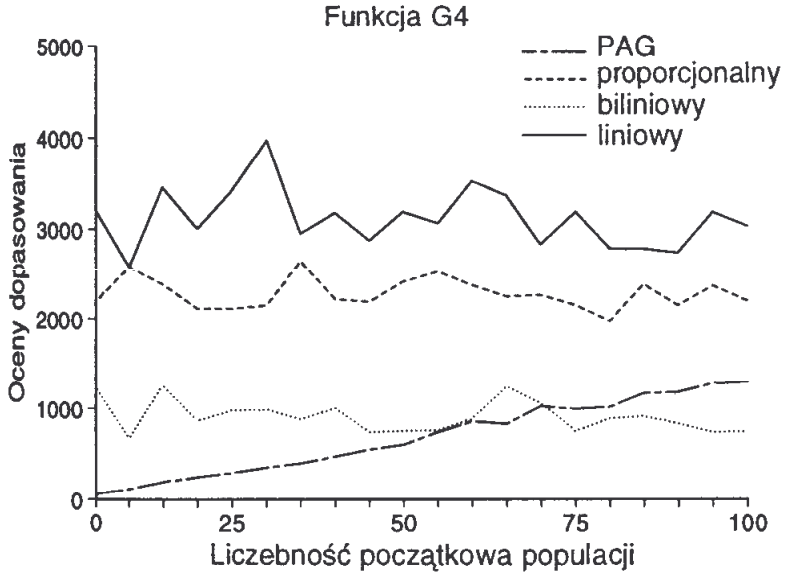
Na rysunkach 4.5-4.6 przedstawiono wpływ współczynnika reprodukcji na działanie algorytmu genetycznego ze zmienną liczebnością populacji. Dla prostego algorytmu genetycznego ta wartość nie ma znaczenia (ponieważ w tym przypadku następuje pełna zamiana starej populacji przez nową). W przypadku algorytmu genetycznego ze zmienną liczebnością populacji ta wartość silnie wpływa na koszt symulacji, który może spaść przy zmniejszeniu współczynnika reprodukcji, jednak bez straty dokładności (patrz odpowiednia wartości *avg-max*). Sądząc z obliczeń, wygląda na to, że „optymalny" wybór ρ wynosi w przybliżeniu 0,4.



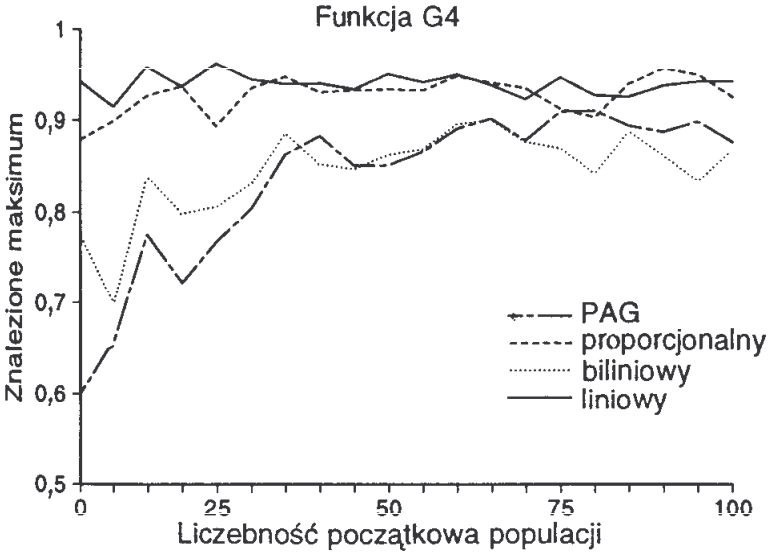
Rys. 4.5. Porównanie prostego algorytmu genetycznego (PAG) i algorytmu genetycznego ze zmienną liczebnością populacji: liczba ocen w funkcji współczynnika reprodukcji



Rys. 4.6. Porównanie prostego algorytmu genetycznego (PAG) i algorytmu genetycznego ze zmienną liczebnością populacji: średnie działanie w funkcji współczynnika reprodukcji



Rys. 4.7. Porównanie prostego algorytmu genetycznego (PAG) i algorytmu genetycznego ze zmienną liczebnością populacji: liczba ocen w funkcji liczebności początkowej populacji



Rys. 4.8. Porównanie prostego algorytmu genetycznego (PAG) i algorytmu genetycznego ze zmienną liczebnością populacji: średnie działanie w funkcji liczebności początkowej populacji

Na rysunkach 4.7-4.8 przedstawiono wpływ liczebności początkowej populacji na działanie (*avgmax*) i koszt obliczeń (*evalnum*) algorytmu. W przypadku prostych algorytmów genetycznych liczebność populacji (we wszystkich przebiegach) była stała i równa jej wartości początkowej. Tak jak się spodziewano, w prostym algorytmie genetycznym małe liczebności populacji powodują niskie koszty i słabe działanie. Zwiększanie liczebności populacji początkowo poprawia działanie, ale też zwiększa koszt obliczeń. Po tym następuje etap „nasycenia działania", gdy tymczasem koszt stale rośnie liniowo. W przypadku algorytmu genetycznego ze zmienną liczebnością populacji początkowa li-

czebność populacji praktycznie nie wpływa ani na działanie (bardzo dobre), ani na koszt (rozsądny i dostateczny jak na bardzo dobre działanie).

Podobne obserwacje można zrobić, analizując koszt i działanie obu algorytmów dla pozostałych funkcji G1-G3. Należy jednak zauważyć, że zachowanie się (mierzone działaniem w stosunku do kosztu) prostego algorytmu genetycznego jest optymalne przy pewnych liczebnościach populacji we wszystkich czterech zadaniach. Natomiast algorytm genetyczny ze zmienną liczebnością populacji przyjmuje liczebność populacji właściwą dla zadania i stanu przeszukiwania. W tablicy 4.1 przedstawiono działanie i koszt symulacji uzyskane z obliczeń dla wszystkich testowanych funkcji G1-G4. Kolumny: PAG (prosty algorytm genetyczny), AGzZLP(1) (algorytm genetyczny ze zmienną liczebnością populacji), AGzZLP(2) i AGzZLP(3) zawierają najlepszą znalezioną wartość (W) i liczbę obliczeń funkcji (O) dla PAG i AGzZLP, z przydziałem czasu życia odpowiednio proporcjonalnym, liniowym i biliniowym. Optymalne liczebności populacji dla prostego algorytmu genetycznego dla funkcji G1-G4 wynosiły odpowiednio 75, 15, 75 i 100.

Tablica 4.1. Porównanie trzech strategii

Typ algorytmu	Funkcja							
	G1		G2		G3		G4	
	W	O	W	O	W	O	W	O
PAG	2,814	1467	0,875	345	1,996	1420	0,959	2186
AGzZLP(1)	2,831	1708	0,875	970	1,999	1682	0,969	2133
AGzZLP(2)	2,841	3040	0,875	1450	1,999	2813	0,970	3739
AGzZLP(3)	2,813	1538	0,875	670	1,999	1555	0,972	2106

Strategia liniowa (2) charakteryzuje się najlepszym działaniem i (niestety) najwyższym kosztem. Na drugim końcu strategia biliniowa jest najtańsza, ale działalnie przy niej nie jest tak dobre, jak w przypadku liniowym. W końcu strategia proporcjonalna (1) prowadzi do średniego działania ze średnim kosztem. Należy zauważyć, że algorytm genetyczny ze zmienną liczebnością populacji i jakąkolwiek strategią przydziału czasu życia (1)-(3) w większości testów działał lepiej niż prosty algorytm genetyczny. Koszt algorytmu genetycznego ze zmienną liczebnością populacji (w porównaniu z prostym algorytmem genetycznym) jest wyższy. Jednak wyniki dla prostego algorytmu genetycznego są podane dla optymalnych liczebności populacji. Jeżeli, na przykład, liczebność populacji dla prostego algorytmu genetycznego w obliczeniach dla funkcji G2 wynosiła 75 (zamiast optymalnej 15), to koszt symulacji prostego algorytmu genetycznego był równy 1035.

Wiedza o właściwym wyborze parametrów algorytmów genetycznych jest wciąż tylko fragmentaryczna i pochodzi z doświadczenia. Wśród tych parametrów najważniejsza wydaje się być liczebność populacji, gdyż istotnie wpływa ona na koszty symulacji algorytmu genetycznego. Być może najlepszym

sposobem jej ustalenia jest pozwolić się jej nastroić samej zgodnie z potrzebami algorytmu genetycznego. Jest to właśnie pomysł leżący u podstaw algorytmu genetycznego ze zmienną liczebnością populacji: na różnych etapach procesu przeszukiwania mogą być optymalne inne liczebności populacji. Przedstawione wyniki mają jednak wstępny charakter i przydział czasu życia wymaga jeszcze dalszych badań.

4.5. Algorytmy genetyczne, ograniczenia i zadanie załadunku

Jak przedstawiono we wprowadzeniu, metody uwzględniania ograniczeń w algorytmach genetycznych można podzielić na kilka grup. Jeden ze sposobów traktowania punktów próbnych, które naruszają ograniczenia, polega na wygenerowaniu potencjalnego rozwiązania bez uwzględniania ograniczeń, a następnie karania ich przez obniżanie „dobroci" funkcji oceny. Inaczej mówiąc, zadanie z ograniczeniami jest przekształcane do zadania bez ograniczeń przez dołączenie kary za naruszanie ograniczeń. Kary te są włączone do funkcji oceniającej. Oczywiście istnieje wiele możliwych funkcji kar, których można tu użyć. W niektórych funkcjach używa się kary stałej. Inne funkcje kary zależą od stopnia naruszenia ograniczeń. Im większe naruszenie, tym większą nakłada się karę (wzrost tej funkcji wraz z wielkością naruszenia ograniczeń może być logarytmiczny, liniowy, kwadratowy, wykładniczy itp.).

Inną wersją metody kary jest eliminacja rozwiązań niedopuszczalnych z populacji (jest to zastosowanie najsurowszej kary, kary śmierci). Ta metoda jest użyta z powodzeniem w strategiach ewolucyjnych (rozdz. 8) do zadań optymalizacji numerycznej. Jednak ma ona wady. W pewnych zadaniach prawdopodobieństwo wygenerowania (za pomocą standardowych operatorów genetycznych) rozwiązania dopuszczalnego jest stosunkowo małe i algorytm zużywa znaczny czas na ocenianie niedopuszczalnych osobników. Ponadto w podejściu tym rozwiązania niedopuszczalne nie wpływają na zasób genów populacji.

Inna grupa metod uwzględniania ograniczeń jest związana z zastosowaniem specjalnych algorytmów naprawy do „korygowania" wygenerowanych rozwiązań niedopuszczalnych. Takie algorytmy naprawy mogą być także wymagające obliczeniowo i wynikające z tego algorytmy muszą być dostosowywane do poszczególnych zastosowań. Ponadto w pewnych zadaniach proces korygowania rozwiązań może być tak samo trudny jak zadanie początkowe.

Trzecie podejście koncentruje się na użyciu specjalnych przekształceń reprezentacji (dekoderów), które gwarantują (lub przynajmniej zwiększają prawdopodobieństwo) generacji rozwiązania dopuszczalnego lub stosują operatory zależne od zadania, zachowujące dopuszczalność rozwiązań. Jednak dekodery

mają często duże wymagania obliczeniowe [73], nie wszystkie ograniczenia mogą być łatwo uwzględnione w ten sposób, a otrzymane algorytmy muszą być dostosowywane do poszczególnych zastosowań.

W tym punkcie przebadamy omówione wyżej metody na pewnym szczególnym zero-jedynkowym zadaniu załadunku. Zadanie to jest łatwe do sformułowania, jednak podjęcie w nim decyzji należy do rodziny problemów NP-trudnych. Ocena zalet i wad metod uwzględniania ograniczeń dla tego zadania z jednym ograniczeniem jest ciekawym przykładem. Wnioski mogą odnosić się do wielu zadań optymalizacji kombinatorycznej. Należy jednak zaznaczyć, że głównym celem tego punktu jest przedstawienie pojęć dekodera, algorytmu naprawy i funkcji kary (omówionych krótko we wprowadzeniu) na jednym szczególnym przykładzie. W żadnym przypadku nie jest to wyczerpujący przegląd możliwych metod. Z tego powodu nie przedstawiamy rozwiązań optymalnych dla testów, a tylko porównujemy przedstawiane metody.

4.5.1. Zero-jedynkowe zadanie załadunku i dane testowe

Istnieją różne zadania typu załadunku (plecakowe), w których jest dany zbiór artykułów (rzeczy) wraz z ich wartościami i rozmiarami. Należy wybrać jeden lub więcej rozłącznych podzbiorów, tak aby suma rozmiarów w każdym podzbiorze nie przekroczyła zadanego ograniczenia (pojemności plecaka) i aby suma z wybranych wartości była maksymalna [252]. Wiele zadań z tej klasy jest NP-trudnych i w wielu przypadkach takie zadania mogą być rozwiązywane tylko za pomocą algorytmów heurystycznych. Zadanie wybrane do eksperymentów to zero-jedynkowe zadanie załadunku. Zagadnienie polega na wyborze, dla danych zbiorów wag $W[i]$, zysków $P[i]$ i pojemności C, wektora binarnego $\mathbf{x} = \langle x[1], ..., x[n] \rangle$, takiego że

$$\sum_{i=1}^{n} x[i] \cdot W[i] \leqslant C$$

i dla którego

$$\mathrm{P}(\mathbf{x}) = \sum_{i=1}^{n} x[i] \cdot P[i]$$

jest maksymalne.

Jak zauważono wcześniej, w tym punkcie analizujemy eksperymentalnie zachowanie się kilku algorytmów opartych na algorytmie genetycznym na kilku losowo wygenerowanych zadaniach testowych. Ponieważ trudności w takich zadaniach są głównie spowodowane korelacją między zyskami a wagami [252], rozważono trzy losowo wygenerowane zbiory danych:

- *nieskorelowane*
  $W[i] :=$ przypadkowe([1, v]) o rozkładzie jednostajnym
  $P[i] :=$ przypadkowe([1, v]) o rozkładzie jednostajnym
- *słabo skorelowane*
  $W[i] :=$ przypadkowe([1, v]) o rozkładzie jednostajnym
  $P[i] := W[i] +$ przypadkowe([$-r$, r]) o rozkładzie jednostajnym
  (jeżeli dla pewnego i wartość $P[i] \leqslant 0$, to taka wartość zysku jest pomijana i obliczenia powtarza się tak długo, aż $P[i] > 0$).
- *silnie skorelowane*
  $W[i] :=$ przypadkowe([1, v]) o rozkładzie jednostajnym
  $P[i] := W[i] + r$

Z większej korelacji wynika mniejsza wartość różnicy

$$\max_{i=1\ldots n} \{P[i]/W[i]\} - \min_{i=1\ldots n} \{P[i]/W[i]\}$$

i zgodnie z [252] dla zadań z większymi korelacjami można się spodziewać większych trudności.

Dane wygenerowano przy następujących wartościach parametrów: $v = 10$ i $r = 5$. Do testów użyliśmy trzech zbiorów danych każdego typu zawierających odpowiednio $n = 100$, 250 i 500 artykułów. Ponownie ulegając sugestii z [252], rozważyliśmy dwa typy zadań załadunku:

- *z ograniczoną pojemnością*
  Zadanie z pojemnością $C_1 = 2v$. W tym przypadku rozwiązanie optymalne zawiera bardzo mało artykułów. Powierzchnia, w której warunki nie są spełnione, zajmuje prawie całą dziedzinę.
- *ze średnią pojemnością*
  Zadanie z pojemnością $C_2 = 0{,}5 \sum_{i=1}^{n} W[i]$. W tym przypadku rozwiązanie optymalne zawiera około połowy artykułów.

Jak opisano w [252], dalsze powiększanie pojemności C nie zwiększa w istotny sposób czasu obliczeń algorytmu klasycznego.

4.5.2. Opis algorytmów

Zaprogramowano i przetestowano trzy typy algorytmów: algorytmy z funkcją kary ($A_p[i]$, gdzie i jest wskaźnikiem algorytmu w tej klasie), algorytmy z metodami naprawy ($A_r[i]$) i algorytmy z dekoderami ($A_d[i]$). Opiszemy teraz po kolei te algorytmy.

Algorytmy $A_p[i]$

We wszystkich algorytmach z tej klasy rozwiązanie problemu **x** jest reprezentowane przez łańcuchy binarne o długości n. Artykuł i-ty jest ładowany do plecaka wtedy i tylko wtedy, gdy $x[i] = 1$.

Dopasowanie $eval(\mathbf{x})$ łańcucha jest dane wyrażeniem

$$eval(\mathbf{x}) = \sum_{i=1}^{n} x[i] \cdot P[i] - Pen(\mathbf{x})$$

przy czym funkcja kary $Pen(\mathbf{x})$ jest równa zero dla rozwiązań dopuszczalnych **x**, to znaczy rozwiązań takich, że $\sum_{i=1}^{n} x[i] \cdot W[i] \leqslant C$, a większa od zera dla pozostałych.

Istnieje wiele możliwych strategii ustalania wartości kary. Tutaj rozważymy tylko trzy przypadki, w których wzrost funkcji kary wraz ze stopniem przekroczenia ograniczeń jest odpowiednio logarytmiczny, liniowy i kwadratowy:

- $A_p[1]$: $Pen(\mathbf{x}) = \log_2(1 + \rho \cdot (\sum_{i=1}^{n} x[i] \cdot W[i] - C))$
- $A_p[2]$: $Pen(\mathbf{x}) = \rho \cdot (\sum_{i=1}^{n} x[i] \cdot W[i] - C)$
- $A_p[3]$: $Pen(\mathbf{x}) = (\rho \cdot (\sum_{i=1}^{n} x[i] \cdot W[i] - C))^2$

We wszystkich trzech przypadkach $\rho = \max_{i=1\ldots n} \{P[i]/W[i]\}$.

Algorytmy $A_r[i]$

Tak jak w poprzedniej klasie algorytmów, rozwiązanie problemu **x** jest reprezentowane przez łańcuchy binarne o długości n. Artykuł i-ty jest ładowany do plecaka wtedy i tylko wtedy, gdy $x[i] = 1$.

Dopasowanie $eval(\mathbf{x})$ łańcucha jest wyznaczane przez

$$eval(\mathbf{x}) = \sum_{i=1}^{n} x'[i] \cdot P[i]$$

przy czym wektor $\mathbf{x}'$ jest naprawioną wersją oryginalnego wektora **x**.

Są tu dwie interesujące kwestie. Po pierwsze możemy rozważać różne metody naprawy. Po drugie pewien procent naprawionych chromosomów może wymienić chromosomy z populacji. Taki ułamek wymiany może się zmieniać od 0% do 100%. Ostatnio Orvosh i Davis [301] opisali tak zwaną regułę 5%, która mówi, że przy wymianie z prawdopodobieństwem 5% działanie algorytmu jest lepsze niż przy jakiejkolwiek innej (w szczególności jest ono lepsze niż przy strategiach „nie wymieniaj żadnego" i „wymieniaj wszystkie").

Zaprogramowano i przetestowano dwa różne algorytmy naprawy. Oba są oparte na tej samej procedurze przedstawionej na rys. 4.9.

```
procedure repair (x)
begin
  plecak przepełniony := false
  x' := x
  if Σ_{i=1}^{n} x'[i] · W[i] > C
    then plecak przepełniony := true
  while (plecak przepełniony) do
  begin
    i := select artykuł z plecaka
    usuń wybrany artykuł z plecaka
      tzn. x'[i] := 0
    if Σ_{i=1}^{n} x'[i] · W[i] ≤ C
    then plecak przepełniony := false
  end
end
```

Rys. 4.9. Procedura naprawy

Rozważone tu dwa algorytmy naprawy różnią się tylko procedurą selekcji **select**, w której wybiera się artykuł do wyjęcia z plecaka:

- $A_r[1]$ (naprawa losowa). Procedura **select** wybiera losowo element z plecaka.
- $A_r[2]$ (naprawa zachłanna). Wszystkie artykuły w plecaku są ustawiane w porządku malejącym względem stosunków zysku do wagi. Procedura **select** wybiera zawsze do usunięcia ostatni artykuł (z listy artykułów dostępnych).

Algorytmy $A_d[i]$

Najbardziej naturalne dekodery dla zadania załadunku korzystają z reprezentacji całkowitoliczbowej. Tutaj użyliśmy reprezentacji porządkowej wybranych artykułów (patrz rozdz. 10 po więcej szczegółów o tej reprezentacji). Każdy chromosom jest wektorem n liczb całkowitych, i-ty składnik wektora jest liczbą całkowitą z zakresu od 1 do $n - i + 1$. Reprezentacja porządkowa korzysta z listy L artykułów, a dekodowanie za pomocą wektora następuje przez wybór odpowiednich artykułów z bieżącej listy. Na przykład dla listy artykułów $L = (1, 2, 3, 4, 5, 6)$ dekodowanie za pomocą wektora $\langle 4, 3, 4, 1, 1, 1 \rangle$ daje następujący ciąg artykułów: 4, 3, 6 (ponieważ 6 jest czwartym elementem bieżącej listy po usunięciu z niej 4 i 3), 1, 2, 5. W tej metodzie chromosom (wektor) może być oczywiście interpretowany jako strategia dołączania artykułu do

(częściowego) rozwiązania. Dodatkowo krzyżowanie jednopunktowe dowolnych dwóch rodziców będzie dawało dopuszczalnego potomka. Operator mutacji definiuje się w podobny sposób jak dla reprezentacji binarnej. Jeżeli i-ty gen ulegnie mutacji, to przyjmuje on wartość losową (z rozkładem jednostajnym) z zakresu $[1, n - i + 1]$. Algorytm dekodowania jest przedstawiony na rys. 4.10.

```
procedure decode (x)
begin
   build lista L elementów
   i := 1
   WeightSum := 0
   ProfitSum := 0
   while i ⩽ n do
   begin
      j := x[i]
      usuń j-ty element z listy L
      if WeightSum + W[j] ⩽ C then
      begin
         WeightSum := WeightSum + Weight[j]
         ProfitSum := ProfitSum + Profit[j]
      end
      i := i + 1
   end
end
```

Rys. 4.10. Procedura dekodowania dla reprezentacji porządkowej

Przedstawione tu dwa algorytmy korzystające z metody dekodowania różnią się tylko procedurą **build**:

- $A_d[1]$ (dekodowanie losowe). W tym algorytmie procedura **build** tworzy listę L artykułów takich, że kolejność pozycji na liście odpowiada kolejności artykułu w zbiorze wejściowym (który jest przypadkowy).
- $A_d[2]$ (dekodowanie zachłanne). Procedura **build** tworzy listę L artykułów w porządku malejącym względem stosunków zysku do wagi. Dekodowanie wektora **x** następuje na podstawie uporządkowanego zbioru (występują tu pewne podobieństwa do metody $A_r[2]$). Na przykład $x[i] = 23$ jest interpretowane jako 23 artykuł (w malejącym porządku względem stosunków zysku do wagi) na bieżącej liście L.

4.5.3. Obliczenia i wyniki

We wszystkich obliczeniach liczebność populacji była stała i równa 100. Także prawdopodobieństwa mutacji i krzyżowania były ustalone odpowiednio na

0,05 i 0,65. Ocenę działania algorytmu podejmowano na podstawie najlepszych rozwiązań z 500 pokoleń. Sprawdzono empirycznie, że po takiej liczbie pokoleń nie zaobserwowano poprawy. Wyniki przedstawione w tabl. 4.2 są średnimi wartościami z 25 obliczeń. Dokładnych rozwiązań nie podano. Tablica zawiera tylko względne efektywności różnych algorytmów. Zauważmy, że zbiory danych nie były uporządkowane (dowolna kolejność artykułów, nie związania z ich stosunkami $P[i]/W[i]$). Pojemności C_1 i C_2 oznaczają odpowiednio pojemności ograniczone i średnie (p. 2).

Tablica 4.2. Wyniki obliczeń

Korelacja	Liczba pozycji	Typ pojemności	Metoda						
			$A_p[1]$	$A_p[2]$	$A_p[3]$	$A_r[1]$	$A_r[2]$	$A_d[1]$	$A_d[2]$
brak	100	C_1	*	*	*	62,9	94,0	63,5	59,4
		C_2	398,1	341,3	342,6	344,6	371,3	354,7	353,3
	250	C_1	*	*	*	62,6	135,1	58,0	60,4
		C_2	919,6	837,3	825,5	842,2	894,4	867,4	857,5
	500	C_1	*	*	*	63,9	156,2	61,0	61,4
		C_2	1712,2	1570,8	1565,1	1577,4	1663,2	1602,8	1597,0
słaba	100	C_1	*	*	*	39,7	51,0	38,2	38,4
		C_2	408,5	327,0	328,3	330,1	358,2	333,6	332,3
	250	C_1	*	*	*	43,7	74,0	42,7	44,7
		C_2	920,8	791,3	788,5	798,4	852,1	804,4	799,0
	500	C_1	*	*	*	44,5	93,8	43,2	44,5
		C_2	1729,0	1531,8	1532,0	1538,6	1624,8	1548,4	1547,1
mocna	100	C_1	*	*	*	61,6	90,0	59,5	59,5
		C_2	741,7	564,5	564,4	566,5	577,0	576,2	576,2
	250	C_1	*	*	*	65,5	117,0	65,5	64,0
		C_2	1631,9	1339,5	1343,4	1345,8	1364,4	1366,4	1359,0
	500	C_1	*	*	*	67,5	120,0	67,1	64,1
		C_2	3051,6	2703,8	2700,8	2709,5	2748,1	2738,0	2744,0

* oznacza, że w zadanym przedziale czasowym nie znaleziono dopuszczalnego rozwiązania.

Wyniki dla metod $A_r[1]$ i $A_r[2]$ uzyskano, przyjmując regułę 5% naprawy. Zbadano także, czy reguła 5% nadaje się dla zadania zero-jedynkowego załadunku (reguła ta była opracowana w czasie obliczeń dla dwóch innych zadań kombinatorycznych: projektowania sieci i kolorowania grafu [301]). Dla porównania wybrano zbiór danych testowych o słabej korelacji między wagami a zyskami. Wszystkie parametry były ustalone, a wartości współczynnika naprawy zmieniały się od 0% do 100%. Zaobserwowano, że reguła 5% ma wpływ na algorytm genetyczny. Wyniki (dla algorytmu $A_r[2]$) zebrano w tabl. 4.3.

Główne wnioski wyciągnięte z obliczeń można podsumować następująco:

- Funkcje kary $A_p[i]$ (dla wszystkich i) nie dają dobrych wyników dla zadań z ograniczonymi pojemnościami plecaka (C_1). Tak jest dla dowolnej liczby artykułów (n = 100, 250 i 500) i dowolnej korelacji.

- Sądząc tylko z wyników obliczeń dla zadań ze średnimi pojemnościami plecaka (C_2), algorytm $A_p[1]$ z logarytmiczną funkcją kary jest wyraźnie najlepszy. Wygrywa on z innymi algorytmami we wszystkich przypadkach (nieskorelowane, słabo lub silnie skorelowane, z n = 100, 250, i 500 artykułami). Jednak, jak zauważono powyżej, zawodzi on dla zadań z ograniczoną pojemnością.
- Sądząc tylko z wyników obliczeń dla zadań z ograniczoną pojemnością plecaka (C_1), metoda naprawy $A_r[2]$ (naprawa zachłanna) wygrywa z innymi metodami we wszystkich testowanych przypadkach.

Tablica 4.3. Wpływ współczynnika naprawy na działanie algorytmu $A_r[2]$

Korelacja	słaba					
Liczba pozycji	100		250		500	
Typ pojemności	C_1	C_2	C_1	C_2	C_1	C_2
Współczynnik naprawy						
0	94,0	371,3	134,1	895,0	158,0	1649,1
5	94,0	371,7	135,1	891,4	155,8	1648,5
10	94,0	370,2	135,1	889,5	157,3	1640,3
15	94,0	368,3	135,3	895,4	156,6	1646,2
20	94,0	372,0	135,6	905,7	155,8	1643,6
25	94,0	370,2	135,1	894,0	155,8	1644,0
30	94,0	367,2	135,1	895,7	157,3	1648,0
35	94,0	370,3	136,1	896,6	156,3	1643,3
40	94,0	368,6	134,3	886,5	156,1	1648,4
45	94,0	369,0	135,3	891,5	156,6	1649,0
50	94,0	371,7	134,6	891,0	156,1	1641,5
55	94,0	371,3	135,0	895,7	157,0	1647,0
60	94,0	369,6	135,0	894,0	156,1	1645,0
65	94,0	370,0	135,1	893,2	156,6	1642,8
70	94,0	367,6	135,0	893,4	156,1	1640,9
75	94,0	367,7	135,3	895,7	157,1	1648,1
80	94,0	368,2	134,3	898,5	155,8	1648,5
85	94,0	364,7	135,6	897,4	156,1	1646,2
90	94,0	368,7	134,3	885,2	156,1	1648,9
95	94,0	371,2	135,0	890,5	155,3	1642,3
100	94,0	370,2	134,6	901,0	156,3	1646,1

Te wyniki są intuicyjnie całkiem jasne. W przypadku ograniczonej pojemności plecaka tylko bardzo małe ułamki możliwych podzbiorów artykułów tworzą rozwiązania dopuszczalne. W konsekwencji większość metod z funkcjami kary zawodzi. Jest to często spotykany przypadek w wielu zadaniach kombinatorycznych. Im mniejszy stosunek między dopuszczalną częścią przeszukiwanej przestrzeni a całą przeszukiwaną przestrzenią, tym trudniej jest metodom z funkcjami kary uzyskać dopuszczalne rozwiązanie. Zaobserwowano to już w [332], gdzie autorzy napisali:

W zadaniach rzadkich [zgrubne funkcje kary] rzadko znajdują rozwiązania. Rozwiązania, które przypadkiem udało się znaleźć, są słabe. Powód tego jest jasny. Dla rzadkiego obszaru dopuszczalnego jest mało prawdopodobne, aby początkowa populacja zawierała rozwiązanie. Ponieważ [zgrubne funkcje kary] nie rozróżniają rozwiązań niedopuszczalnych, algorytm genetyczny błądzi w koło bez celu. Jeżeli w wyniku jakiejś szczęśliwej mutacji albo dziwnego krzyżowania okaże się, że potomek wpadł w obszar dopuszczalny, to staje się on *superosobnikiem*, którego materiał genetyczny szybko dominuje w populacji, co prowadzi do przedwczesnej zbieżności.

Algorytmy naprawy natomiast dają sobie dobrze radę. Przy średniej pojemności plecaka metoda logarytmicznej funkcji kary (to znaczy małych kar) jest najlepsza. Warto zauważyć, że rozmiar zadania nie wpływa na wnioski.

Jak zauważono wcześniej, obliczenia te nie dają jeszcze pełnego obrazu. Dużo dodatkowych obliczeń planuje się na najbliższą przyszłość. Dla algorytmów z funkcjami kary warto by było poeksperymentować z innymi wzorami. Wszystkie rozważane kary miały postać $Pen(\mathbf{x}) = f(\mathbf{x})$, przy czym f była funkcją logarytmiczną, liniową lub kwadratową. Inna możliwość, to spróbować funkcji $Pen(\mathbf{x}) = a + f(\mathbf{x})$ z pewną stałą a. Dawałoby to pewną minimalną karę dla każdego niedopuszczalnego wektora. Dalej można by było przeprowadzić eksperymenty z *dynamicznymi* funkcjami kary, które zależą od dodatkowych parametrów, takich jak numer pokolenia (było to zrobione w [267] dla optymalizacji z ciągłymi zmiennymi) lub właściwości przeszukiwania (patrz [360], gdzie dynamicznych funkcji kary użyto w zadaniu rozplanowania ustawienia urządzeń: kiedy znajduje się lepsze rozwiązania dopuszczalne lub niedopuszczalne, kara nałożona na dane rozwiązanie niedopuszczalne zmienia się). Także *adaptacyjne* funkcje kary wydają się warte sprawdzenia. Poza tym prawdopodobieństwa użycia operatorów mogą być ustalane adaptacyjnie (jak w strategiach ewolucyjnych). Początkowe obliczenia wykazują, że pewne zalety ma adaptacyjne ustalanie liczebności populacji (p. 4.4). Tak więc idea adaptacyjnych funkcji kary jest warta lepszego przyjrzenia się. W swojej najprostszej wersji współczynnik kary może być częścią wektora rozwiązań i podlegać wszystkim genetycznym (losowym) zmianom (odwrotnie niż w dynamicznych funkcjach kary, gdzie taki współczynnik kary jest zmieniany regularnie jako funkcja, na przykład, numeru pokolenia).

Jest także możliwe sprawdzenie wielu metod naprawy, łącznie z innymi wskaźnikami niż stosunek zysku do wagi. Mogłoby być także interesujące połączenie metod funkcji kary i algorytmów naprawy. Rozwiązania niedopuszczalne dla algorytmu $A_p[i]$ byłyby wtedy naprawiane na dopuszczalne.

W grupie dekoderów należałoby sprawdzić różne (całkowitoliczbowe) reprezentacje (jak zrobiono dla zadania komiwojażera w rozdz. 10): reprezentacje przyległe (z krzyżowaniem ze zmianą łuków, krzyżowaniem na kawałkach tras lub krzyżowaniem heurystycznym) lub reprezentacje ścieżek (z krzyżowaniem PMX, OX lub CX albo nawet z krzyżowaniem z kombinacją łuków).

Ciekawe byłoby porównanie użyteczności tych reprezentacji i operatorów w zadaniu zero-jedynkowego załadunku (jak to zrobiono dla zadania załadunku i harmonogramowania [370]). Jest całkiem możliwe, że jakieś nowe krzyżowanie zależne od zadania prowadziłoby do najlepszych wyników.

4.6. Inne pomysły

W poprzednim punkcie tego rozdziału omówiliśmy pewne sprawy związane z usunięciem możliwych błędów w mechanizmie próbkowania. Głównym celem tych badań było powiększenie przeszukiwania genetycznego oraz walka z przedwczesną zbieżnością algorytmów genetycznych. Przez ostatnich kilka lat były podejmowane wysiłki znalezienia lepszego przetwarzania schematów, przy innych podejściach. W tym punkcie omówimy niektóre z nich.

Pierwszy kierunek jest związany z operatorem genetycznym krzyżowania. Naśladuje on proces biologiczny, jednak ma pewne wady. Na przykład przypuśćmy, że mamy dwa bardzo dobre schematy

$$S_1 = (0\ 0\ 1\ *\ *\ *\ *\ *\ *\ *\ *\ 0\ 1) \quad \text{i}$$

$$S_2 = (*\ *\ *\ *\ 1\ 1\ *\ *\ *\ *\ *\ *\ *)$$

W populacji istnieją dwa łańcuchy $\mathbf{v}_1$ i $\mathbf{v}_2$ pasujące odpowiednio do S_1 i S_2:

$$\mathbf{v}_1 = (\mathbf{001}0001101\mathbf{01}) \quad \text{i}$$

$$\mathbf{v}_2 = (1110110001000)$$

Oczywiście z krzyżowania nie można uzyskać pewnych kombinacji cech zakodowanych w chromosomach. Nie można uzyskać łańcucha pasującego do schematu

$$S_3 = (0\ 0\ 1\ *\ 1\ 1\ *\ *\ *\ *\ *\ 0\ 1)$$

gdyż pierwszy schemat byłby zniszczony.

Dodatkowym argumentem przeciwko klasycznemu krzyżowaniu jednopunktowemu jest asymetria między mutacją a krzyżowaniem. Mutacja zależy od długości chromosomu, a krzyżowanie nie. Na przykład, jeżeli prawdopodobieństwo mutacji wynosi $p_m = 0{,}01$, a długość chromosomu wynosi 100, to oczekiwana liczba zmutowanych bitów w chromosomie równa się 1. Jeżeli długość chromosomu wynosi 1000, to oczekiwana liczba zmutowanych bitów w chromosomie równa się 10. W obu jednak przypadkach krzyżowanie jednopunktowe łączy dwa łańcuchy przez cięcie w jednym miejscu, niezależnie od długości łańcucha.

Niektórzy badacze (patrz np. [104] lub [382]) eksperymentowali z innymi krzyżowaniami. Na przykład w krzyżowaniu dwupunktowym wybiera się dwa punkty cięcia i jest wymieniany materiał chromosomowy między nimi. Oczywiście w ten sposób z łańcuchów $\mathbf{v}_1$ i $\mathbf{v}_2$ można utworzyć parę potomków

$$\mathbf{v}_1' = (\mathbf{001}|01\mathbf{100}|01\mathbf{001}) \quad \text{i}$$

$$\mathbf{v}_2' = (111|00011|01000)$$

przy czym $\mathbf{v}_1'$ pasuje do

$$S_3 = (0\ 0\ 1\ *\ 1\ 1\ *\ *\ *\ *\ *\ 0\ 1)$$

co nie jest możliwe dla krzyżowania jednopunktowego.

Podobnie istnieją schematy, których nie może utworzyć krzyżowanie dwupunktowe. Możemy wtedy próbować krzyżowania wielopunktowego [104] będącego naturalnym uogólnieniem krzyżowania dwupunktowego. Zauważmy jednak, że ponieważ w krzyżowaniu wielopunktowym trzeba zamieniać między rodzicami odcinki (uzyskane po pocięciu chromosomu na s kawałków), zatem ich liczba musi być parzysta. Znaczy to, że krzyżowanie wielopunktowe nie jest naturalnym rozszerzeniem krzyżowania jednopunktowego.

Schaffer i Morishima [341] prowadzili eksperymenty z krzyżowaniem, w którym następuje adaptacja rozkładu punktów cięcia do zachodzących procesów (przeżycia najlepiej dopasowanych i rekombinacji). Dokonano tego, wprowadzając specjalne znaki do reprezentacji łańcuchów. Zaznaczają one miejsca w łańcuchu, gdzie zaszło cięcie. Było to zrobione z nadzieją, że miejsca cięcia będą podlegały procesowi ewolucji – jeżeli pewne miejsce prowadzi do słabych potomków, to zaginie (i odwrotnie). Eksperymenty wykazały [341], że adaptacyjne krzyżowanie dawało wyniki równie dobre lub lepsze niż klasyczne algorytmy genetyczne dla zbioru zadań testowych. Spears [367] eksperymentował z adaptacją szczególnych krzyżowań (rozważono dwa krzyżowania: jednopunktowe i jednorodne), rozszerzając reprezentację chromosomu o jeden bit.

Niektórzy [104] eksperymentowali z innymi krzyżowaniami: krzyżowaniem odcinkowym i krzyżowaniem tasowanym. Krzyżowanie odcinkowe jest pewnym wariantem krzyżowania wielopunktowego, ze zmieniającą się liczbą punktów cięcia. Liczba (ustalona) punktów cięcia (lub odcinków) jest w nich zastąpiona współczynnikiem zmiany odcinka. Określa on prawdopodobieństwo, że odcinek zakończy się w rozważanym punkcie łańcucha. Na przykład, jeżeli współczynnik zmiany odcinka jest równy $s = 0{,}2$, to, zaczynając od początku odcinka, szansa zakończenia odcinka w każdym kolejnym punkcie wynosi 0,2. Inaczej mówiąc, przy współczynniku zmiany odcinka $s = 0{,}2$ oczekiwana liczba odcinków wyniesie $m/5$. Jednak w odróżnieniu od krzyżowania wielopunktowego liczba cięć będzie się zmieniać. Krzyżowanie tasowane można rozumieć jako dodatkowy mechanizm, który można stosować z innymi

krzyżowaniami. Nie zależy ono od liczby punktów cięć. W krzyżowaniu tasowanym następuje (1) losowe ustawienie (tasowanie) miejsc bitów w dwóch dobranych łańcuchach, (2) krzyżowanie łańcuchów, to znaczy wymiana odcinków między punktami cięć, (3) przywracanie miejsc bitów w łańcuchach.

Dalszym uogólnieniem krzyżowania jedno-, dwu- i wielopunktowego jest krzyżowanie jednorodne [366], [382]. Dla każdego bitu pierwszego potomka decyduje się w nim (z pewnym prawdopodobieństwem p), od którego z rodziców pobierze się wartość. Drugi potomek otrzymuje bit od pozostałego rodzica. Dla przykładu, przy $p = 0{,}5$ (krzyżowanie 0,5-jednorodne) z łańcuchów

$$\mathbf{v}_1 = (0010001101001) \quad \text{i}$$

$$\mathbf{v}_2 = (1110110001000)$$

można otrzymać następującą parę potomków:

$$\mathbf{v}_1' = (0_1 0_1 1_2 0_1 1_2 1_2 0_2 1_1 0_1 1_2 0_1 0_1 0_2) \quad \text{i}$$

$$\mathbf{v}_2' = (1_2 1_2 1_1 0_2 0_1 0_1 1_1 0_2 0_2 1_1 0_2 0_2 1_1)$$

gdzie indeksy dolne 1 i 2 wskazują, od którego rodzica (odpowiednio wektor $\mathbf{v}_1$ lub $\mathbf{v}_2$) uzyskano bit. Jeżeli $p = 0{,}1$ (krzyżowanie 0,1-jednorodne), to z łańcuchów $\mathbf{v}_1$ i $\mathbf{v}_2$ otrzymamy

$$\mathbf{v}_1' = (0_1 0_1 1_1 0_1 1_2 0_1 1_1 1_1 0_1 1_2 0_1 0_1 1_1) \quad \text{i}$$

$$\mathbf{v}_2' = (1_2 1_2 1_2 0_2 0_1 1_2 0_2 0_2 0_2 1_1 0_2 0_2 0_2)$$

Ponieważ w krzyżowaniu jednorodnym wymienia się raczej bity niż odcinki, można w nim uzyskać połączenie cech niezależne od ich wzajemnego położenia. W pewnych zadaniach [382] ta możliwość przeważa nad wadą związaną z niszczeniem bloków budujących. Jednak w innych zadaniach krzyżowanie jednorodne jest gorsze od krzyżowania dwupunktowego. Syswerda [382] porównał teoretycznie krzyżowanie 0,5-jednorodne z krzyżowaniami jedno- i dwupunktowym. Spears i De Jong [366] przedstawił analizę krzyżowania p-jednorodnego, to znaczy krzyżowania, w którym występuje średnio mp punktów cięcia.

Eshelman [104] opisuje kilka eksperymentów z różnymi operatorami krzyżowania. Z porównania wynika, że „przegrywające" jest krzyżowanie jednopunktowe, ale nie ma zdecydowanego zwycięzcy. Ogólny komentarz do tych eksperymentów stwierdza, że każde z krzyżowań jest szczególnie użyteczne w pewnych klasach zadań, a słabo wypada w innych. Wzmacnia to nasz pomysł operatorów zależnych od zadania, co doprowadziło do programów ewolucyjnych.

Mühlenbein i Voigt [290] badali właściwości nowego operatora rekombinacji zwanego *rekombinacją puli genów*, w którym geny są wybierane losowo z puli genów wybranych rodziców. Interesującą cechą tego operatora jest to, że

uwzględnia się w nim tak zwane *orgie*, w których kilku rodziców tworzy potomka. Takie krzyżowanie wielorodzicielskie było także badane przez Eibena i innych [100], gdzie rozważono *metody przeglądania genów* (w których tworzy się jednego potomka z kilku rodziców). Renders i Bersini [324] eksperymentowali z *krzyżowaniem simpleksowym* dla zadań optymalizacji numerycznej. To krzyżowanie wymaga obliczenia centroidu dla grupy rodziców i przesunięcia od najgorszego osobnika ponad punkt centroidu. Także w metodach szukania rozproszonego [142] proponuje się użycie wielu rodziców.

Kilka osób przyglądało się także skutkom zmian parametrów sterujących w algorytmie genetycznym (liczebność populacji, prawdopodobieństwa operacji) na działanie systemu. Grefenstette [169] zastosował metaalgorytm genetyczny do sterowania parametrów innego algorytmu genetycznego. Goldberg [153] przedstawił analizę teoretyczną optymalnej liczebności populacji. Kompletne studium wpływu parametrów sterujących na przeszukiwanie genetyczne (działanie na bieżąco dla optymalizacji funkcji) przedstawiono w [343]. Wyniki sugerują, że (1) mutacja gra istotniejszą rolę niż się wydawało wcześniej (mutacja jest uważana za operację drugoplanową), (2) istotność współczynnika krzyżowania jest mniejsza niż oczekiwano, (3) strategia przeszukiwania oparta tylko na selekcji i mutacji może być mocną strategią przeszukiwania nawet bez pomocy krzyżowania (podobnie jak w strategiach ewolucyjnych przedstawionych w rozdz. 8). Jednak w algorytmach genetycznych ciągle brakuje dobrych przepisów heurystycznych określania dobrych wartości parametrów. Nie ma jednego wspólnego ustalenia, które pasowałoby do wszystkich rozważanych zadań. Wygląda na to, że znalezienie dobrych wartości parametrów algorytmu genetycznego jest nadal bardziej sztuką niż nauką.

Do tego rozdziału omawialiśmy dwa podstawowe operatory genetyczne: krzyżowanie (jednopunktowe, dwupunktowe, jednorodne itp.) i mutację, które stosowano do osobników lub pojedynczych bitów z pewną stałą częstością (prawdopodobieństwem krzyżowania p_c i mutacji p_m). Jak widzieliśmy w przykładzie przewijającym się przez rozdz. 2, było możliwe użycie krzyżowania i mutacji dla tego samego osobnika (na przykład $\mathbf{v}_{13}$). Faktycznie te dwa podstawowe operatory można traktować jako jeden operator rekombinacji, operator „krzyżowania i mutacji", gdyż oba te operatory można stosować do osobników w tym samym czasie. Jedna z możliwości eksperymentowania z operatorami genetycznymi polega na niezależnym ich używaniu. Tylko jeden z nich, a nie oba, jest używany w trakcie reprodukcji [78]. Istnieje kilka zalet takiej separacji. Po pierwsze po krzyżowaniu nie zachodzi mutacja, co czyni cały proces prostszym pojęciowo. Po drugie łatwiej rozszerzyć listę operatorów genetycznych przez dodanie nowych. Taka lista może się składać z kilku operatorów zależnych od zadania. Jest to dokładnie pomysł prowadzący do programowania ewolucyjnego: dużo zależnych od zadania „operatorów genetycznych" stosowanych do struktur danych osobników. Przypomnijmy także, że dla programów ewolucyjnych przedstawionych w tej książce opracowano specjalny podprogram selekcji modGA (w poprzednim punkcie), który ułatwia zastoso-

wanie powyższego pomysłu. Możemy ponadto pójść dalej. Każdy operator może mieć własne dopasowanie, które podlegałoby także pewnemu procesowi ewolucyjnemu. Wybór i zastosowanie operatora następuje losowo, zgodnie z ich dopasowaniami. Nie jest to nowy pomysł i był on wspominany od pewnego czasu [77], [115], [78], ale w metodach programowania ewolucyjnego nabiera nowego znaczenia.

Innym interesującym kierunkiem poszukiwania lepszego przekształcania schematów, wspomnianym już w poprzednim rozdziale przy okazji zadań zawodnych, a zaproponowanym ostatnio [155], [159], jest nieporządny algorytm genetyczny.

Nieporządny algorytm genetyczny różni się od klasycznego algorytmu genetycznego w wielu punktach: reprezentacji, operatorów, liczebności populacji, selekcji i faz procesu ewolucyjnego. Omówimy je krótko po kolei.

Przede wszystkim każdy bit chromosomu jest oznaczany nazwą (numerem) – tego samego chwytu użyliśmy, omawiając operator inwersji w poprzednim rozdziale. Dodatkowo łańcuchy mają zmienną długość i nie ma wymagania, aby miały one pełny komplet genów. Łańcuch może zawierać geny nadmiarowe, a nawet sprzeczne. Na przykład następujące łańcuchy są dopuszczalnymi chromosomami w nieporządnym algorytmie genetycznym

$$\mathbf{v}_1 = ((7, 1)(1, 0))$$

$$\mathbf{v}_2 = ((3, 1)(9, 0)(3, 1)(3, 1)(3, 1))$$

$$\mathbf{v}_3 = ((2, 1)(2, 0)(4, 1)(5, 0)(6, 0)(7, 1)(8, 1))$$

Pierwsza liczba w każdym nawiasie podaje miejsce, a druga liczba wartość bitu. Tak więc pierwszy łańcuch, $\mathbf{v}_1$, opisuje dwa bity: bit 1 na pozycji 7 i bit 0 na pozycji 1.

Przy ocenie takiego łańcucha mamy do czynienia z nadmiarowością (łańcuch $\mathbf{v}_3$, gdzie na pozycji 2 znajdują się dwa bity) i niedookreśleniem (wszystkie trzy wektory są niedookreślone, jeżeli przyjmiemy, że mają one 9 bitów). Z nadmiarowością możemy sobie dać radę na kilka sposobów. Można na przykład użyć pewnych procedur głosowania (deterministycznych lub probabilistycznych) lub uporządkowania według miejsc. Trudniej jest postępować z niedookreśleniem i w tej mierze odsyłamy czytelnika do informacji źródłowej [155], [158], [159].

Oczywiście wprowadzenie łańcuchów o zmiennej długości, nadmiarowych lub niedookreślonych wymaga zmiany używanych operatorów. Prostą wymianę zamienia się na dwa (nawet prostsze) operatory: *łączenia i cięcia*. Operator łączenia skleja (konkatenuje) dwa wybrane łańcuchy (z zadanym prawdopodobieństwem łączenia). Na przykład łącząc łańcuchy $\mathbf{v}_1$ i $\mathbf{v}_2$, dostajemy

$$\mathbf{v}_4 = ((7, 1)(1, 0)(3, 1)(9, 0)(3, 1)(3, 1)(3, 1))$$

Operator cięcia rozcina (z pewnym prawdopodobieństwem cięcia) wybrany łańcuch na pozycji określonej w sposób losowy wzdłuż jego długości. Na przykład rozcinając łańcuch v_3 na pozycji czwartej, otrzymamy

$$v_5 = ((2, 1)(2, 0)(4, 1)(5, 0)) \quad \text{i}$$

$$v_6 = ((6, 0)(7, 1)(8, 1))$$

Operator mutacji nie ulega zmianie. Zamienia on 0 na 1 (lub odwrotnie) z pewnym zadanym prawdopodobieństwem.

Istnieją jeszcze inne różnice między algorytmem genetycznym a nieporządnym algorytmem genetycznym. W nieporządnym algorytmie genetycznym (do rzetelnej selekcji niezależnie od wyskalowania funkcji) używa się selekcji turniejowej [155]. Dzieli się w nim także proces ewolucji na dwie fazy (w pierwszej fazie wybiera się bloki budujące, a dopiero w drugiej fazie wywołuje się operatory genetyczne), liczebność zaś populacji zmienia się w czasie procesu.

Nieporządne algorytmy genetyczne były testowane na kilku funkcjach zawodnych z bardzo dobrymi wynikami [155], [158]. Jak przedstawia Goldberg [158]:

> Przyjęto trudne funkcje testowe i w dwóch turach eksperymentów nieporządny algorytm genetyczny był w stanie znaleźć optimum globalne. [...] We wszystkich obliczeniach w obu turach eksperymentów nieporządny algorytm genetyczny zbiegał do globalnych optimów funkcji testowych. Dla porównania, prosty algorytm genetyczny z przypadkowym uporządkowaniem łańcucha był w stanie znaleźć poprawne rozwiązanie tylko dla 25% funkcji.

a w [158]:

> Ponieważ nieporządne algorytmy mogą zbiegać się w trudnych zadaniach można przypuszczać, że znajdą one optima globalne we wszystkich innych zadaniach z ograniczoną zawodnością. Ponadto nieporządne algorytmy genetyczne są ukształtowane tak, aby zbiegały się w czasie rosnącym tylko wielomianowo w funkcji liczby zmiennych decyzyjnych na zwykłym komputerze i logarytmicznie na komputerze równoległym. Na koniec nieporządne algorytmy genetyczne są praktycznym narzędziem, którego można użyć do wspięcia się na drabinę zawodności funkcji, otrzymując użyteczne i stosunkowo niedrogie wyniki po drodze.

Były także inne próby powiększenia przeszukiwania genetycznego. Whitley i in. [400] zaproponowali ostatnio modyfikację algorytmów genetycznych zwaną delta-kodowaniem. Schraudoph i Belew [347] zaproponowali strategię dynamicznego kodowania parametrów, w której dynamicznie dostosowuje się dokładność zakodowanych osobników. Te algorytmy będą omawiane w dalszym ciągu książki (rozdz. 8).

Część II
Optymalizacja numeryczna

5

Binarnie czy zmiennopozycyjnie

Wprawdzie w klasztorze obowiązywały reguły,
ale Mistrz zawsze przestrzegał przed tyranią prawa.

Antonio de Mello, *Minuta mądrości*[1)]

Jak przedstawiono w poprzednim rozdziale, w zastosowaniach algorytmów genetycznych występują problemy, które opóźniają, a nawet uniemożliwiają znalezienie rozwiązań optymalnych z założoną dokładnością. Jedną z konsekwencji jest przedwczesna zbieżność całej populacji do lokalnego optimum (rozdz. 4). Inne konsekwencje to brak możliwości dokładnego lokalnego dostrojenia oraz niemożność działania przy nietrywialnych ograniczeniach (rozdz. 6 i 7).

Reprezentacja binarna używana tradycyjnie w algorytmach genetycznych ma pewne wady, gdy stosuje się ją do rozwiązywania wielowymiarowych zadań wymagających dużej dokładności. Na przykład przy 100 zmiennych i dziedzinie o zakresach $[-500, 500]$ oraz żądanej dokładności 6 cyfr po przecinku długość binarnego wektora rozwiązań wynosi 3000. To z kolei prowadzi do przestrzeni przeszukiwań o liczebności rzędu 10^{1000}. Dla takich zadań algorytmy genetyczne działają słabo.

Alfabet binarny oferuje maksymalną liczbę schematów na bit informacji przy jakimkolwiek kodowaniu [154] i w konsekwencji reprezentacja rozwiązań w postaci łańcuchów binarnych zdominowała badania nad algorytmami genetycznymi. Kodowanie to ułatwia także analizę teoretyczną i umożliwia wprowadzenie eleganckich operatorów genetycznych. Ale wyniki związane z „wewnętrzną równoległością" nie zależą od zastosowania łańcuchów binarnych [9] i może warto przeprowadzić eksperymenty z większymi alfabetami i (ewentualnie) nowymi operatorami genetycznymi. W szczególności w zadaniach optymalizacji parametrycznej z ciągłymi dziedzinami możemy eksperymentować z genami zakodowanymi za pomocą liczb rzeczywistych oraz odpowiednio dla nich określonymi operatorami „genetycznymi". Goldberg napisał w [157]:

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

Użycie kodowania za pomocą liczb rzeczywistych lub genów w zapisie zmiennopozycyjnym ma długą, choć może kontrowersyjną, historię w sztucznej genetyce i schematach przeszukiwania ewolucyjnego, a ich zastosowanie wydaje się być ostatnio wzrastające. To wzrastające zastosowanie jest trochę dziwne dla badaczy znających teorię podstawowych algorytmów genetycznych [154], [188], ponieważ prosta analiza wydaje się wskazywać na to, że zwiększone przetwarzanie schematów uzyskuje się przy niskich liczbach kardynalnych, co jest w sposób oczywisty sprzeczne z empirycznymi odkryciami, że kodowanie za pomocą liczb rzeczywistych daje dobre wyniki w wielu zadaniach praktycznych.

W tym rozdziale opiszemy wyniki obliczeń dla różnych modyfikacji operatorów genetycznych z reprezentacją zmiennopozycyjną. Głównym celem takich rozwiązań jest (zgodnie z zasadą programowania ewolucyjnego) przybliżenie algorytmu genetycznego do przestrzeni zadania. Taki ruch wymusza, ale także pozwala na to, aby operatory były bardziej zależne od zadania, przez wykorzystanie pewnych specyficznych właściwości przestrzeni rzeczywistych. Na przykład dla takiej reprezentacji dwa punkty leżące blisko siebie w przestrzeni reprezentacji będą także leżały blisko w przestrzeni zadania, i na odwrót. Nie jest to zawsze prawdziwe przy reprezentacji binarnej, gdzie odległość między reprezentacjami jest zazwyczaj definiowana jako liczba pozycji o różnych bitach. Jest jednak możliwe zredukowanie tej sprzeczności przez użycie kodu Graya.

Procedurę konwersji liczby binarnej $\mathbf{b} = \langle b_1, ..., b_m \rangle$ w liczbę w kodzie Graya $g = \langle g_1, ..., g_m \rangle$ i odwrotnie przedstawiono na rys. 5.1. Parametr m oznacza liczbę bitów w tych reprezentacjach.

```
procedure Binary-to-Gray
begin
  g_1 = b_1
  for k = 2 to m do
     g_k = b_{k-1} XOR b_k
  end

procedure Gray-to-Binary
begin
  value = g_1
  b_1 = value
  for k = 2 to m do
  begin
     if g_k = 1 then value = NOT value
     b_k = value
  end
end
```

Rys. 5.1. Procedury zamiany kodu binarnego na kod Graya i odwrotnie

W tablicy 5.1 podano pierwszych 16 liczb binarnych łącznie z odpowiadającymi im kodami Graya.

Tablica 5.1. Kod binarny i kod Graya

binarny	Graya
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Zauważmy, że reprezentacja w kodzie Graya ma tę właściwość, że dowolne dwa punkty leżące obok siebie w przestrzeni zadania różnią się tylko jednym bitem. Inaczej mówiąc, zwiększenie parametru o jeden krok odpowiada zmianie jednego bitu w kodzie. Zauważmy też, że istnieją inne równoważne procedury konwersji kodów binarnych i Graya. Dla przykładu (przypadek $m=4$) para macierzy

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad A^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

prowadzi do następującej transformacji:

$$\mathbf{g} = A\mathbf{b} \quad \text{i} \quad \mathbf{b} = A^{-1}\mathbf{g}$$

gdzie operatorem mnożenia jest modulo 2.

Jednak tu używamy reprezentacji zmiennopozycyjnej, gdyż jest ona pojęciowo bliższa przestrzeni zadania, a także ułatwia łatwe i efektywne zaprogramowanie operatorów domkniętych i dynamicznych (patrz także rozdz. 6 i 7). Następnie porównamy empirycznie rozwiązania binarne i zmiennopozycyjne z różnymi nowymi operatorami na wielu przykładach testowych. W tym rozdziale objaśniamy różnice między reprezentacjami binarnymi a zmiennopozy-

cyjnymi na typowym przykładzie testowym dotyczącym zadania sterowania. Jest to zadanie liniowo-kwadratowe, które jest szczególnym przypadkiem zadania rozważanego w następnym rozdziale (razem z dwoma innymi zadaniami sterowania) ilustrującego postęp naszego programu ewolucyjnego w zakresie przedwczesnej zbieżności i dokładnego lokalnego dostrojenia. Jak się można spodziewać, wyniki są lepsze niż dla reprezentacji binarnej. Ten sam wniosek wyciągnęli inni badacze, np. [78], [408].

5.1. Przykład testowy

Do eksperymentowania wybraliśmy następujące zadanie sterowania:

$$J = \min(x_N^2 + \sum_{k=0}^{N-1} (x_k^2 + u_k^2))$$

przy ograniczeniach

$$x_{k+1} = x_k + u_k, \quad k = 0, 1, ..., N-1$$

gdzie x_0 jest zadanym stanem początkowym, $x_k \in R$ jest stanem, a $\mathbf{u} \in R^N$ jest poszukiwanym wektorem sterowania. Wartość optymalną można obliczyć analitycznie

$$J^* = K_0 x_0^2$$

gdzie K_k jest rozwiązaniem równania Riccatiego

$$K_k = 1 + K_{k+1}/(1 + K_{k+1}) \quad \text{i} \quad K_N = 1$$

Podczas obliczeń chromosom był reprezentowany przez wektor sterowania $\mathbf{u}$. Założono także stałą dziedzinę $\langle -200, 200 \rangle$ dla każdego u_i (rzeczywiste rozwiązania znajdowały się w tym zakresie w przeprowadzonych testach). We wszystkich obliczeniach przyjęto $x_0 = 100$ i $N = 45$, to znaczy chromosom $\mathbf{u} = \langle u_0, ..., u_{44} \rangle$, co daje wartość optymalną $J^* = 16180{,}4$.

5.2. Dwie wersje

Do badań wybrano dwie wersje algorytmów genetycznych różniące się reprezentacjami i stosowanymi operatorami genetycznymi. Poza tym są one równoważne. Takie podejście daje nam lepsze podstawy do bezpośredniego porównania. W obu wersjach zastosowano ten sam mechanizm selekcji: uniwersalne próbkowanie stochastyczne [23].

5.2.1. Wersja binarna

W wersji binarnej każdy element wektora chromosomu zakodowano za pomocą tej samej liczby bitów. Dla ułatwienia szybkiego dekodowania na bieżąco każdy element jest zapisany w osobnym słowie pamięci (w ogólności może on zajmować więcej niż jedno słowo, jeżeli liczba bitów związanych z elementem przekracza długość słowa; taki przypadek jest łatwym uogólnieniem). W ten sposób elementy mogą być traktowane jako liczby całkowite, co zwalnia z konieczności dekodowania binarnego lub dziesiętnego. Dalej, każdy chromosom jest wektorem składającym się z N słów, tj. tylu słów, ile jest elementów w chromosomie (lub wielokrotności tej liczby w przypadku, gdy do zapisania odpowiedniej liczby bitów potrzeba kilku słów).

Dokładność takiego podejścia zależy (przy ustalonej dziedzinie) od liczby rzeczywiście używanych bitów i jest równa $(UB - LB)/(2^n - 1)$, przy czym UB i LB są ograniczeniami dziedziny, a n jest liczbą bitów w elemencie chromosomu.

5.2.2. Wersja zmiennopozycyjna

W wersji zmiennopozycyjnej każdy wektor chromosomu zakodowano jako wektor liczb zmiennopozycyjnych, o tej samej długości, jak wektor rozwiązania. Każdy element musiał mieścić się w wymaganym zakresie, a operatory określono tak, aby to wymaganie było zachowane.

Dokładność tego podejścia zależy od używanego komputera, ale zazwyczaj jest znacznie wyższa niż przy reprezentacji binarnej. Oczywiście zawsze można zwiększyć dokładność reprezentacji binarnej, wprowadzając większą liczbę bitów, ale zwalnia to znacznie działanie algorytmu (patrz p. 5.4).

Dodatkowo reprezentacja zmiennopozycyjna może objąć całkiem duże dziedziny (lub przypadki o nieznanych dziedzinach). Natomiast przy reprezentacji binarnej jesteśmy zmuszeni zrezygnować z dokładności ze wzrostem wymiaru dziedziny, przy ustalonej liczbie bitów. Dla reprezentacji zmiennopozycyjnej jest także znacznie łatwiej zaprojektować specjalistyczne narzędzia ułatwiające postępowanie w przypadku nietrywialnych ograniczeń. Jest to dokładnie omówione w rozdz. 7.

5.3. Obliczenia

Przeprowadzono dwie serie obliczeń na stacji roboczej DEC3100. Wszystkie przedstawione tu wyniki są wartościami średnimi uzyskanymi z 10 niezależnych przebiegów algorytmu. We wszystkich obliczeniach liczebność populacji była stała i równa 60, a liczba iteracji wynosiła 20 000. Jeżeli nie będzie to

wyraźnie zaznaczone, w reprezentacji binarnej do zakodowania jednej zmiennej (jednego elementu wektora rozwiązania) użyto $n = 30$ bitów, co daje $30 \cdot 45 = 1350$ bitów w całym wektorze.

Ze względu na możliwe różnice w interpretacji operatora mutacji przyjęliśmy prawdopodobieństwo modyfikacji chromosomu tak, aby porównywać reprezentacje zmiennopozycyjne i binarne w uczciwy sposób. Wszystkie obliczenia wykonano dla operatorów ustawionych tak, aby uzyskać takie same współczynniki. Wobec tego numery iteracji są w przybliżeniu równe liczbie ocen funkcji.

5.3.1. Losowa mutacja i krzyżowanie

W tej części eksperymentu przeprowadzono obliczenia z operatorami, które są równoważne (przynajmniej dla reprezentacji binarnej) z operatorami tradycyjnymi.

Reprezentacja binarna

W reprezentacji binarnej użyto tradycyjnych operatorów mutacji i krzyżowania. Jednak, aby były one bardziej zbliżone do użytych w wersji zmiennopozycyjnej, zezwolono na krzyżowanie tylko między elementami. Prawdopodobieństwo krzyżowania ustalono na 0,25, a prawdopodobieństwo mutacji zmieniało się tak, aby uzyskać żądany współczynik modyfikacji chromosomu (co przedstawiono w tabl. 5.2).

Tablica 5.2. Zależność prawdopodobieństwa modyfikacji chromosomu i współczynnika mutacji

Wersja	Prawdopodobieństwo modyfikacji chromosomu				
	0,6	0,7	0,8	0,9	0,95
binarna, p_m	0,00047	0,00068	0,00098	0,0015	0,0021
zmiennopozycyjna, p_m	0,014	0,02	0,03	0,045	0,061

Reprezentacja zmiennopozycyjna

Operator krzyżowania był bardzo podobny do użytego w wersji binarnej (punkty cięcia między cyframi liczby zmiennopozycyjnej) i użyty z tym samym prawdopodobieństwem (0,25). Mutacja, którą nazywamy losową, odnosi się do jednej z cyfr liczby zmiennopozycyjnej, a nie do bitu. Wynikiem takiej mutacji jest liczba przypadkowa z zakresu $\langle LB, UB \rangle$.

Wyniki

Wyniki (tabl. 5.3) są nieznacznie lepsze dla przypadku binarnego. Jednak jest je dosyć trudno ocenić dokładniej, gdyż wszystkie znajdują się daleko od roz-

wiązania optymalnego (16180,4). Na dodatek wystąpiło interesujące zjawisko wskazujące, że wersja zmiennopozycyjna jest bardziej stabilna, z dużo mniejszym odchyleniem standardowym.

Tablica 5.3. Wyniki średnie w funkcji prawdopodobieństwa modyfikacji chromosomu

Wersja	Prawdopodobieństwo modyfikacji chromosomu					Odchylenie standardowe
	0,6	0,7	0,8	0,9	0,95	
binarna	42179	46102	29290	52769	30573	31212
zmiennopozycyjna	46594	41806	47454	69624	82371	11275

Warto zauważyć jeszcze, że powyższe obliczenia nie były całkiem uczciwe w stosunku do reprezentacji zmiennopozycyjnej. Jej losowa mutacja zachowuje się „bardziej" losowo niż mutacja dla wersji binarnej, w której losowa zmiana bitu nie oznacza utworzenia całkowicie losowej wartości z rozpatrywanej dziedziny. Dla ilustracji rozważmy następujące pytanie. Jakie jest prawdopodobieństwo, że po mutacji element znajdzie się nie dalej niż $\delta\%$ zakresu dziedziny (400, gdyż dziedzina jest $\langle -200, 200\rangle$) od swojej starej wartości? Odpowiedź jest następująca.

Wersja zmiennopozycyjna. Takie prawdopodobieństwo oczywiście znajduje się w przedziale $\langle\delta, 2\delta\rangle$. Dla przykładu przy $\delta = 0{,}05$ znajduje się ono w przedziale $\langle 0{,}05, 0{,}1\rangle$.

Wersja binarna. Tutaj musimy rozważyć liczbę mało znaczących bitów, które można bezpiecznie zmienić. Zakładając, że długość elementu wynosi $n = 30$, a długość możliwej zmiany m, musi zachodzić $m \leqslant n + \log_2\delta$. Ponieważ m jest liczbą całkowitą, więc $m = [n + \log_2\delta] = 25$ i poszukiwane prawdopodobieństwo wynosi $m/n = 25/30 = 0{,}833$, a więc jest całkiem inne niż poprzednio.

Spróbujemy znaleźć metodę skompensowania tej wady w następnym punkcie.

5.3.2. Mutacja nierównomierna

W tej części eksperymentu użyto, jako dodatek do operatorów opisanych w p. 5.3.1, specjalnej mutacji dynamicznej mającej na celu zarówno poprawę dopasowania pojedynczego elementu, jak i skompensowanie zbytniej losowości mutacji w wersji zmiennopozycyjnej. Nazywamy to *mutacją nierównomierną*. Pełne omówienie tego operatora przedstawimy w następnych dwóch rozdziałach.

Reprezentacja zmiennopozycyjna

Definiujemy nowy operator następująco. Jeżeli $s_v^t = \langle v_1, ..., v_m \rangle$ jest chromosomem (t jest numerem pokolenia), a element v_k jest wybrany do mutacji, to wynik jest wektorem $s_v^{t+1} = \langle v_1, ..., v'_k, ..., v_m \rangle$, gdzie

$$v'_k = \begin{cases} v_k + \Delta(t, UB - v_k), & \text{jeżeli wylosowaną losowo liczbą jest 0} \\ v_k - \Delta(t, v_k - LB), & \text{jeżeli wylosowaną losowo liczbą jest 1} \end{cases}$$

a LB i UB są dolnym i górnym ograniczeniem zmiennej v_k. Wartości funkcji $\Delta(t, y)$ znajdują się w zakresie $[0, y]$, przy czym prawdopodobieństwo, że $\Delta(t, y)$ jest bliskie 0 wzrasta ze wzrostem t. Ta właściwość powoduje, że operator ten początkowo (kiedy t jest małe) przeszukuje równomiernie całą przestrzeń, a w późniejszych etapach przeszukuje lokalnie, w ten sposób zwiększając prawdopodobieństwo utworzenia nowej liczby zbliżonej do swojej poprzedniczki, a nie liczby całkiem losowej. Użyto następującej funkcji:

$$\Delta(t, y) = y(1 - r^{(1 - t/T)b})$$

gdzie r jest liczbą losową z przedziału $[0, 1]$, T jest maksymalną liczbą pokoleń, a b jest parametrem systemu określającym stopień zależności od numeru iteracji (przyjęto $b = 5$).

Reprezentacja binarna

Aby być więcej niż uczciwym w stosunku do reprezentacji binarnej, zamodelowano także dla niej operator dynamiczny, mimo że był on wprowadzony głównie w celu poprawienia mutacji zmiennopozycyjnej. Jest on podobny do mutacji zmiennopozycyjnej, ale z inaczej zdefiniowanym v'_k:

$$v'_k = mutate(v_k, \nabla(t, n))$$

gdzie $n = 30$ jest liczbą bitów w elemencie chromosomu, $mutate(v_k, pos)$ oznacza zmutowanie bitu pos w k-tym elemencie (bit 0 jest najmniej znaczący), zaś

$$\nabla(t, n) = \begin{cases} \lfloor \Delta(t, n) \rfloor, & \text{jeżeli wylosowaną losowo cyfrą jest 0} \\ \lceil \Delta(t, n) \rceil, & \text{jeżeli wylosowaną losowo cyfrą jest 1} \end{cases}$$

z odpowiednio dobranym parametrem b funkcji Δ, jeżeli żądamy podobnego zachowania się obu operatorów (użyto $b = 1{,}5$).

Wyniki

Powtórzyliśmy obliczenia podobne do tych z p. 5.3.1, stosując także mutację nierównomierną z tym samym współczynnikiem, jak w mutacji zdefiniowanej poprzednio.

Obecnie wersja zmiennopozycyjna wykazuje lepsze średnie działanie (tabl. 5.4). Prócz tego ponownie wyniki dla wersji binarnej są bardziej niestabilne. Warto jednak tu dodać, że pomimo gorszej średniej wersja binarna dała dwa najlepsze pojedyncze wyniki w tej rundzie (16205 i 16189).

Tablica 5.4. Wyniki średnie w funkcji prawdopodobieństwa modyfikacji chromosomu

Wersja	Prawdopodobieństwo modyfikacji chromosomu		Odchylenie standardowe
	0,8	0,9	
binarna	35265	30373	40256
zmiennopozycyjna	20561	26164	2133

5.3.3. Inne operatory

W tej części eksperymentu zdecydowano się na zaprogramowanie i użycie tylu dodatkowych operatorów, ile tylko można było łatwo zdefiniować w obu przestrzeniach reprezentacji.

Reprezentacja binarna
Jako dodatek do operatorów opisanych poprzednio zaprogramowano krzyżowanie wielopunktowe, a także zezwolono na krzyżowanie bitów elementów. Prawdopodobieństwo użycia operatora wielopunktowego dla pojedynczego elementu było sterowane za pomocą parametru systemu (ustalonego na 0,3).

Reprezentacja zmiennopozycyjna
Tutaj także zaprogramowano podobne krzyżowanie wielopozycyjne. Dodatkowo zaprogramowano krzyżowanie arytmetyczne jedno- i dwupunktowe. Zamiast wymieniać, uśredniają one wartości z dwóch elementów w wybranych punktach. Operatory te mają taką właściwość, że każdy element nowego chromosomu pozostaje nadal w dziedzinie. Więcej szczegółów o tych operatorach przedstawiono w następnych dwóch rozdziałach.

Wyniki
Wersja zmiennopozycyjna jest tu wyraźnie lepsza (tabl. 5.5). Nawet jeżeli najlepsze wyniki nie różnią się zbytnio od siebie, to uzyskane są one tylko za pomocą wersji zmiennopozycyjnej.

Tablica 5.5. Wyniki średnie w funkcji prawdopodobieństwa modyfikacji chromosomu

Wersja	Prawdopodobieństwo modyfikacji chromosomu			Odchylenie standardowe	Najlepszy
	0,7	0,8	0,9		
binarna	23814	19234	27456	6078	16188,2
zmiennopozycyjna	16248	16798	16198	54	16182,1

5.4. Efektywność czasowa obliczeń

Często słychać narzekania na słabą efektywność czasową algorytmów genetycznych przy rozwiązywaniu nietrywialnych zadań. W tym punkcie porównamy efektywność czasową obu rozpatrywanych wersji. W tablicy 5.6 przedstawiono wyniki uzyskane dla obliczeń opisanych w p. 5.3.3.

Tablica 5.6. Czas CPU[s] w funkcji liczby elementów

Wersja	Liczba elementów (N)				
	5	15	25	35	45
binarna	1080	3123	5137	7177	9221
zmiennopozycyjna	184	398	611	823	1072

W tablicy 5.6 porównano czasy CPU dla obu wersji przy zmiennej liczbie elementów w chromosomie. Wersja zmiennopozycyjna jest znacznie szybsza, nawet przy umiarkowanej liczbie 30 bitów na zmienną w wersji binarnej. Dla dużych dziedzin i dużej dokładności obliczeń całkowita długość chromosomu wzrasta i względna różnica powiększa się, jak to wynika z tabl. 5.7.

Tablica 5.7. Czas CPU[s] w funkcji liczby bitów w elemencie, $N = 45$

Wersja	Liczba bitów w elemencie					
	5	10	20	30	40	50
binarna	4426	5355	7438	9219	10981	12734
zmiennopozycyjna	1072 (stałe)					

5.5. Wnioski

Przeprowadzone eksperymenty wykazały, że reprezentacja zmiennopozycyjna jest szybsza, daje bardziej zbliżone wyniki w różnych przebiegach i prowadzi do lepszej dokładności (w szczególności przy większych dziedzinach, gdzie kodowanie binarne wymaga zbyt długich reprezentacji). Jednocześnie jej działanie można poprawić, wprowadzając specjalne operatory, aby uzyskać dużą (lepszą niż dla reprezentacji binarnych) dokładność. Dodatkowo dla reprezentacji zmiennopozycyjnej, intuicyjnie bliższej przestrzeni zadania, jest łatwiej opracować inne operatory uwzględniające nietrywialne ograniczenia związane ze specyfiką zadania (rozdz. 7).

Powyższe wnioski zgadzają się z podanymi w [157] powodami, dla których użytkownicy genetycznych metod ewolucyjnych preferują reprezentacje zmiennopozycyjne: (1) wygodą z jednoznaczną zależnością między genem a zmienną,

(2) unikaniem klifów Hamminga i innych sztucznych chwytów w mutacjach operujących na łańcuchach binarnych traktowanych jako liczby całkowite binarne bez znaku, (3) mniejszą liczbą pokoleń potrzebnych do dostosowania się populacji.

W tym miejscu zachęcamy Czytelnika do wykonania kilku obliczeń (patrz dodatek D, ćwiczenie 3). Należy wybrać kilka funkcji testowych (na przykład z dodatku B) i przeprowadzić obliczenia z trzema systemami z algorytmami genetycznymi, z reprezentacjami: binarną, Graya i zmiennopozycyjną. Do dwóch pierwszych systemów użyć GENESIS 1.2ucsd, do trzeciego GENOCOP-u (rozdz. 7).

6

Dokładne dostrajanie lokalne

Kilka tygodni później, gdy jeden z odwiedzających
zapytał go, czego uczy swoich uczniów, odpowiedział:
– Uczę ich ustalania właściwej hierarchii wartości:
lepiej mieć pieniądze, niż je liczyć;
lepiej doświadczyć czegoś, niż dać definicję.

Anthony de Mello, *Minuta mądrości*[1)]

Algorytmy genetyczne wykazują nieodłączne trudności z wykonywaniem lokalnego przeszukiwania w zastosowaniach numerycznych. Holland zasugerował [188], że algorytmy genetyczne powinny być używane jako preprocesory do wykonania początkowego przeszukania, po czym należałoby przełączyć proces przeszukiwania na system, który może użyć wiedzy o dziedzinie zastosowań do ukierunkowania poszukiwania lokalnego. Jak stwierdzono w [170]:

> „Podobnie jak w naturalnych systemach genetycznych algorytmy genetyczne rozwijają się, zmieniając rozkład substruktur o wysokiej efektywności działania w całkowitej populacji, nie skupiając uwagi na strukturach indywidualnych. Dlatego, kiedy tylko algorytmy genetyczne znajdą rejon o wysokiej efektywności działania, może być wygodniej wywołać program szukania lokalnego, który przeprowadzi optymalizację osobników w końcowej populacji."

Szukanie lokalne wymaga użycia schematów o wyższych rzędach i większych długościach definiujących niż sugerowane przez twierdzenie o schematach. Prócz tego istnieją zadania, w których dziedzina parametrów jest nieograniczona, liczba parametrów jest duża, a wymagana jest duża dokładność. W takich wypadkach długość (binarnego) wektora rozwiązań jest dosyć duża (dla 100 zmiennych z dziedziną w zakresie [−500, 500] i wymaganą dokładnością 6 cyfr po przecinku długość binarnego wektora rozwiązań wynosi 3000). Jak wspomniano w poprzednim rozdziale, w takich zadaniach algorytmy genetyczne działają raczej słabo.

Aby zwiększyć możliwości dokładnego lokalnego dostrajania się algorytmów genetycznych, co jest koniecznością dla zadań wymagających dużej do-

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

kładności, opracowaliśmy specjalny operator mutacji, którego działanie jest całkiem inne od tradycyjnego. Przypomnijmy, że tradycyjna mutacja zmienia jeden bit chromosomu na raz. W takiej zmianie używa się tylko lokalnej wiedzy, gdyż tylko bit podlegający mutacji jest znany. Taki bit, jeżeli znajduje się w lewej części sekwencji kodu zmiennej, istotnie wpływa na bezwzględną wielkość efektu mutacji tej zmiennej. Natomiast mutacja bitów leżących daleko na prawo w tej sekwencji ma całkiem mały wpływ. Zdecydowaliśmy się użyć tej wiedzy o pozycji bitu w następujący sposób: ze wzrostem wieku populacji bity znajdujące się bardziej na prawo w sekwencji kodu zmiennej otrzymują większe prawdopodobieństwo mutacji, znajdujące się zaś na lewo mniejsze. Inaczej mówiąc, taka mutacja prowadzi do globalnego przeszukiwania przestrzeni na początku procesu iteracyjnego, aby coraz bardziej przechodzić do lokalnego w późniejszych stadiach. Nazywamy to *mutacją nierównomierną* i omawiamy ją dalej w tym rozdziale. Na razie jednak opiszemy zadania, których użyjemy do testowania tego nowego operatora.

6.1. Przykłady testowe

W ogólności zagadnienie opracowania i zaprogramowania algorytmu rozwiązywania zadań optymalnego sterowania jest trudne. Mocno reklamowane programowanie dynamiczne jest metodą matematyczną, której można użyć w wielu zagadnieniach, a w szczególności do sterowania optymalnego [37]. Jednak metoda ta załamuje się na zadaniach o umiarkowanej wymiarowości i złożoności, cierpiąc na tak zwane „przekleństwo wymiarowości" [33].

Zadania sterowania optymalnego są trudne do rozwiązywania numerycznego. Programy optymalizacji dynamicznej dostępne dla szerokiego kręgu użytkowników pochodzą na ogół z pakietów optymalizacji statycznej [50] i nie używają metod specyficznych dla optymalizacji dynamicznej. Tak więc dostępne programy nie używają w sposób bezpośredni hamiltonianów, warunków transwersalności itp. Jednakże, gdyby nawet używały metod specyficznych dla optymalizacji dynamicznej, to laikowi byłoby jeszcze trudniej się nimi posługiwać.

O ile autorowi wiadomo, dopiero od niedawna zaczęto stosować algorytmy genetyczne do rozwiązywania zadań sterowania optymalnego w systematyczny sposób [273], [271]. Wydaje się, że poprzednie programy z algorytmami genetycznymi były zbyt słabe, aby dać sobie radę z zadaniami, gdzie była wymagana duża dokładność. W tym rozdziale przedstawiono naszą modyfikację algorytmu genetycznego przeznaczoną do polepszenia jego działania. Wskazano na jakość i zakres zastosowania opracowanego systemu za pomocą studium porównawczego dla kilku zadań optymalizacji dynamicznej. Później system ten będzie rozszerzony przez włączenie do niego typowych ograniczeń dla takich zadań optymalizacji – piszemy o tym w następnym rozdziale (p. 7.2). Jako

odniesienie dla tych przypadków testowych (jak i wielu innych eksperymentów omawianych dalej w książce) używamy standardowego pakietu obliczeniowego stosowanego do rozwiązywania takich zadań: Student Version of General Algebraic Modelling System z modułem optymalizującym MINOS [50]. Do końca książki będziemy się powoływali na ten pakiet za pomocą skrótu GAMS.

Za zadania testowe dla programu ewolucyjnego przyjęto trzy proste zagadnienia sterowania optymalnego z czasem ciągłym (często używane w zastosowaniach sterowania optymalnego): zadanie liniowo-kwadratowe, zadanie zbierania plonów i (zdyskretyzowane) zadanie pchania wózka. Omówimy je po kolei.

6.1.1. Zadanie liniowo-kwadratowe

Pierwsze zadanie testowe jest jednowymiarowym zadaniem liniowo-kwadratowym

$$\min q x_N^2 + \sum_{k=0}^{N-1} (s x_k^2 + r u_k^2) \tag{6.1}$$

przy ograniczeniach

$$x_{k+1} = a x_k + b u_k, \qquad k = 0, 1, \ldots, N-1 \tag{6.2}$$

przy czym x_0 jest zadane, a, b, q, s, r są danymi stałymi, $x_k \in R$ jest stanem i $u_k \in R$ jest sterowaniem systemu.

Optymalne rozwiązanie (6.1) przy ograniczeniach (6.2) jest następujące:

$$J^* = K_0 x_0^2 \tag{6.3}$$

gdzie K_k jest rozwiązaniem równania Riccatiego

$$K_k = s + r a^2 K_{k+1}/(r + b^2 K_{k+1}), \qquad K_N = q \tag{6.4}$$

W dalszym ciągu zadanie (6.1) przy ograniczeniach (6.2) będzie rozwiązywane dla zbioru parametrów przedstawionego w tabl. 6.1.

Tablica 6.1. Dziesięć przypadków testowych

Przypadek	N	x_0	s	r	q	a	b
I	45	100	1	1	1	1	1
II	45	100	10	1	1	1	1
III	45	100	1000	1	1	1	1
IV	45	100	1	10	1	1	1
V	45	100	1	1000	1	1	1
VI	45	100	1	1	0	1	1
VII	45	100	1	1	1000	1	1
VIII	45	100	1	1	1	0,01	1
IX	45	100	1	1	1	1	0,01
X	45	100	1	1	1	1	100

W obliczeniach ustalono wartość N równą 45 i był to najdłuższy horyzont, dla którego można było jeszcze uzyskać rozwiązanie z GAMS.

6.1.2. Zadanie zbierania plonów

Zadanie zbierania plonów definiuje się następująco:

$$\max \sum_{k=0}^{N-1} \sqrt{u_k} \tag{6.5}$$

przy ograniczeniu będącym równaniem wzrostu

$$x_{k+1} = a \cdot x_k - u_k \tag{6.6}$$

i jednym ograniczeniu równościowym

$$x_0 = x_N \tag{6.7}$$

przy czym stan początkowy x_0 jest zadany, a jest stałą, zaś $x_k \in R$ i $u_k \in R^+$ są odpowiednio stanem i (nieujemnym) sterowaniem.

Optymalna wartość J^* wyrażenia (6.5) przy ograniczeniach (6.6) i (6.7) wynosi

$$J^* = \sqrt{\frac{x_0 \cdot (a^N - 1)^2}{a^{N-1} \cdot (a - 1)}} \tag{6.8}$$

Zadanie (6.5) przy ograniczeniach (6.6) i (6.7) rozwiązywano dla $a = 1{,}1$, $x_0 = 100$ i $N = 2, 4, 10, 20, 45$.

6.1.3. Zadanie pchania wózka

Zadanie pchania wózka polega na maksymalizacji całkowitej odległości $x_1(N)$ przebytej w zadanym czasie (powiedzmy jednostkowym) po odjęciu całkowitego wysiłku. System jest opisany modelem drugiego rzędu

$$x_1(k+1) = x_2(k) \tag{6.9}$$

$$x_2(k+1) = 2x_2(k) - x_1(k) + \frac{1}{N^2} u(k) \tag{6.10}$$

a wskaźnik jakości ma postać

$$x_1(N) - \frac{1}{2N} \sum_{k=0}^{N-1} u^2(k) \tag{6.11}$$

Optymalna wartość wskaźnika (6.11) dla tego zadania wynosi

$$J^* = \frac{1}{3} - \frac{3N-1}{6N^2} - \frac{1}{2N^3} \sum_{k=0}^{N-1} k^2 \tag{6.12}$$

Zadanie pchania wózka rozwiązywano przy różnych wartościach $N = 5$, 10, 15, 20, 25, 30, 35, 40, 45. Zauważmy, że N odpowiada liczbie kroków dyskretyzacji (równoważnego zadania ciągłego), a nie rzeczywistej długości horyzontu optymalizacji, który przyjęliśmy za 1.

6.2. Program ewolucyjny dla optymalizacji numerycznej

Program ewolucyjny, który utworzyliśmy dla zadań optymalizacji numerycznej, zawiera reprezentację zmiennopozycyjną i pewne nowe (specjalizowane) operatory genetyczne. Omówimy je po kolei.

6.2.1. Reprezentacja

Przy reprezentacji zmiennopozycyjnej każdy chromosom jest kodowany jako wektor zmiennopozycyjny o tej samej długości co wektor rozwiązań. Każdy element jest na początku wybrany z zadanej dziedziny, a operatory są starannie określone tak, aby zachować te ograniczenia (taka trudność nie występuje przy reprezentacji binarnej, ale określenie operatorów jest tu dosyć proste; nie uważamy tego za wadę, natomiast ma to zalety przedstawione dalej).

Dokładność takiego podejścia zależy od używanego komputera, ale na ogół jest znacznie lepsza niż dla reprezentacji binarnej. Oczywiście możemy zawsze zwiększyć dokładność reprezentacji binarnej, wprowadzając więcej bitów, ale spowalnia to znacznie algorytm, jak to omawialiśmy w poprzednim rozdziale.

Dodatkowo reprezentacja zmiennopozycyjna może objąć całkiem duże dziedziny (lub przypadki nieznanych dziedzin). Natomiast w reprezentacji binarnej przy ustalonej liczbie bitów traci się dokładność ze wzrostem wymiarowości dziedziny. Dla reprezentacji zmiennopozycyjnej jest także dużo łatwiej zaprojektować specjalne narzędzia dla nietrywialnych ograniczeń. Dyskutujemy to w pełni w następnym rozdziale.

6.2.2. Operatory specjalizowane

Operatory, których używamy, różnią się zupełnie od klasycznych, ponieważ działają w innych przestrzeniach (liczb rzeczywistych). Jednak z powodu intuicyjnych podobieństw podzielimy je na standardowe klasy: mutacje i krzyżo-

wania. W dodatku niektóre operatory są nierównomierne, to znaczy ich działanie zależy od wieku populacji.

Grupa mutacji:

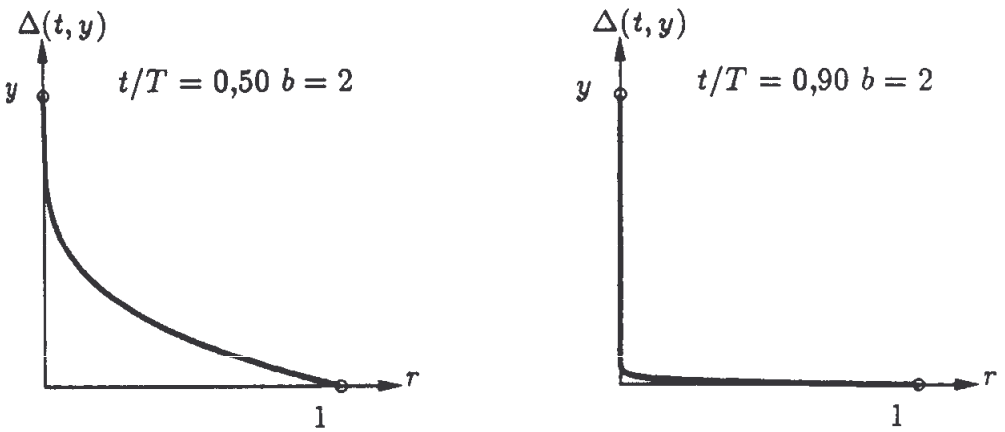
- **mutacja równomierna**, zdefiniowana podobnie do wersji klasycznej: jeżeli $x_i^t = \langle v_1, ..., v_n \rangle$ jest chromosomem, to każdy element v_k ma dokładnie równe szanse na to, aby ulec procesowi mutacji. Wynikiem pojedynczego zastosowania tego operatora jest wektor $\langle v_1, ..., v_k', ..., v_n \rangle$, dla $1 \leqslant k \leqslant n$, gdzie v_k' jest losową wartością z dziedziny odpowiedniego parametru $domain_k$.
- **mutacja nierównomierna** jest jednym z operatorów związanych z możliwością dokładnego dostrojenia się systemu. Jest on określony następująco: jeżeli $s_v^t = \langle v_1, ..., v_m \rangle$ jest chromosomem i element v_k został wybrany do tej mutacji (dziedziną v_k jest $[l_k, u_k]$), to wynikiem jest wektor $s_v^{t+1} = \langle v_1, ..., v_k', ..., v_m \rangle$, dla $k \in \{1, ..., n\}$, gdzie

$$v_k' = \begin{cases} v_k + \Delta(t, u_k - v_k), & \text{jeżeli liczbą losową jest } 0 \\ v_k - \Delta(t, v_k - l_k), & \text{jeżeli liczbą losową jest } 1 \end{cases}$$

przy czym funkcja $\Delta(t, y)$ przyjmuje wartości w zakresie $[0, y]$, a prawdopodobieństwo, że $\Delta(t, y)$ jest bliskie 0, wzrasta ze wzrostem t. Ta właściwość powoduje, że algorytm przeszukuje początkowo przestrzeń w sposób jednorodny (kiedy t jest małe) i bardzo lokalnie w późniejszych etapach. Stosowano następującą funkcję:

$$\Delta(t, y) = y(1 - r^{(1 - t/T)^b})$$

gdzie r jest liczbą losową z zakresu $[0, 1]$, T jest maksymalną liczbą pokoleń, a b jest parametrem systemu określającym stopień niejednorodności. Na rysunku 6.1 przedstawiono funkcję Δ dla dwóch wybranych chwil. Określa on jasno zachowanie się tego operatora.



Rys. 6.1. Funkcja $\Delta(t, y)$ dla dwóch wybranych czasów

Ponadto oprócz standardowego sposobu zastosowania mutacji mamy pewien nowy mechanizm. Mutacja nierównomierna odnosi się do całego wektora rozwiązań, a nie do pojedynczego jego elementu, co powoduje, że cały wektor jest nieco przesunięty w przestrzeni.

Grupa krzyżowań:

- **krzyżowanie proste**, zdefiniowane w zwykły sposób, ale dla danego chromosomu x dopuszczalne punkty cięcia znajdują się tylko między wartościami v,
- **krzyżowanie arytmetyczne** zdefiniowane jako liniowa kombinacja dwóch wektorów: jeżeli krzyżuje się s_v^t i s_w^t, to wynikający z tego potomkowie są wyznaczani następująco: $s_v^{t+1} = a s_w^t + (1-a) s_v^t$ oraz $s_w^{t+1} = a s_v^t + (1-a) s_w^t$. Ten operator zależy od parametru a, który jest albo stałą (jednorodne krzyżowanie arytmetyczne), albo zmienną, której wartość zależy od wieku populacji (niejednorodne krzyżowanie arytmetyczne).

Tutaj ponownie mamy pewien nowy mechanizm przy zastosowaniu operatorów, na przykład krzyżowanie arytmetyczne można stosować albo do wybranych elementów dwóch wektorów, albo do całych wektorów.

6.3. Obliczenia i wyniki

W tym punkcie przedstawimy wyniki programu ewolucyjnego dla zadania sterowania optymalnego. We wszystkich zadaniach testowych liczebność populacji była ustalona na 70, a każdy przebieg obejmował 40 000 pokoleń. Dla każdego zadania testowego wykonywano losowo trzy przebiegi i podano najlepsze wyniki. Należy jednak zwrócić uwagę na to, że odchylenie standardowe wyników takich przebiegów było pomijalnie małe. Wartości początkowe wektorów $\langle u_0, ..., u_{N-1} \rangle$ ustalano losowo (ale w zadanej dziedzinie). Tablice 6.2, 6.3 i 6.4 podają otrzymane wartości łącznie z wynikami pośrednimi w pewnym zakresie pokoleń. Na przykład wartość w kolumnie „10 000" oznacza pośredni wynik po 10 000 pokoleń w czasie obliczania 40 000 pokoleń. Warto zauważyć, że te wartości są gorsze, niż gdyby były liczone przy obliczaniu tylko 10 000 pokoleń, ze względu na charakter pewnych operatorów genetycznych. W następnym punkcie porównamy te wyniki z wynikami dokładnymi oraz z wynikami uzyskanymi z pakietu GAMS.

Zauważmy, że w zadaniu ((6.5)-(6.7)) występuje ograniczenie nałożone na końcowy stan. Różni się ono od zadania ((6.1)-(6.2)) tym, że nie każdy przypadkowy wektor początkowy $\langle u_0, ..., u_{N-1} \rangle$ o elementach będących dodatnimi liczbami rzeczywistymi prowadzi do dopuszczalnego ciągu x_k (patrz warunek (6.6)), takiego że $x_0 = x_N$ dla zadanych wartości a i x_0. W naszym programie ewolucyjnym wygenerowano przypadkowy ciąg $u_0, ..., u_{N-2}$ i przyjęto $u_{N-1} = a x_{N-1} - x_N$. Uzyskując ujemne u_{N-1}, odrzucano cały ciąg i powtarza-

no generację ciągu sterowań (opisujemy ten proces w szczegółach w następnym rozdziale, w p. 7.2). Ta sama trudność pojawiała się w procesie reprodukcji. Potomek (po pewnych operacjach genetycznych) nie musiał spełniać ograniczenia $x_0 = x_N$. W takim przypadku zamieniano ostatni element wektora potomka u przy użyciu wzoru $u_{N-1} = a\,x_{N-1} - x_N$. I ponownie przy ujemnym u_{N-1} nie wprowadzano takiego potomka do nowej populacji.

Tablica 6.2. Program ewolucyjny dla zadania liniowo-kwadratowego (6.1)-(6.2)

	Pokolenia							Mnożnik
Przypadek	1	100	1000	10 000	20 000	30 000	40 000	
I	17904,4	3,87385	1,73682	1,61859	1,61817	1,61804	1,61804	10^4
II	13572,3	5,56187	1,35678	1,11451	1,09201	1,09162	1,09161	10^5
III	17024,8	2,89355	1,06954	1,00952	1,00124	1,00102	1,00100	10^7
IV	15082,1	8,74213	4,05532	3,71745	3,70811	3,70162	3,70160	10^4
V	5968,42	12,2782	2,69862	2,85524	2,87645	2,87571	2,87569	10^5
VI	17897,7	5,27447	2,09334	1,61863	1,61837	1,61805	1,61804	10^4
VII	2690258	18,6685	7,23567	1,73564	1,65413	1,61842	1,61804	10^4
VIII	123,942	72,1958	1,95783	1,00009	1,00005	1,00005	1,00005	10^4
IX	7,28165	4,32740	4,39091	4,42524	4,31021	4,31004	4,31004	10^5
X	9971341	148233	16081,0	1,48445	1,00040	1,00010	1,00010	10^4

Tablica 6.3. Program ewolucyjny dla zadania zbierania plonów (6.5)-(6.7)

	Pokolenia						
N	1	100	1000	10 000	20 000	30 000	40 000
2	6,3310	6,3317	6,3317	6,3317	6,3317	6,3317	6,331738
4	12,6848	12,7127	12,7206	12,7210	12,7210	12,7210	12,721038
8	25,4601	25,6772	25,9024	25,9057	25,9057	25,9057	25,905710
10	32,1981	32,5010	32,8152	32,8209	32,8209	32,8209	32,820943
20	65,3884	68,6257	73,1167	73,2372	73,2376	73,2376	73,237668
45	167,1348	251,3241	277,3990	279,0657	279,2612	279,2676	279,271421

Tablica 6.4. Program ewolucyjny dla zadania pchania wózka (6.9)-(6.10)

	Pokolenia						
N	1	100	1000	10 000	20 000	30 000	40 000
5	−3,008351	0,081197	0,119979	0,120000	0,120000	0,120000	0,120000
10	−5,668287	−0,011064	0,140195	0,142496	0,142500	0,142500	0,142500
15	−6,885241	−0,012345	0,142546	0,150338	0,150370	0,150370	0,150371
20	−7,477872	−0,126734	0,149953	0,154343	0,154375	0,154375	0,154377
25	−8,668933	−0,015673	0,143030	0,156775	0,156800	0,156800	0,156800
30	−12,257346	−0,194342	0,123045	0,158241	0,158421	0,158426	0,158426
35	−11,789546	−0,236753	0,110964	0,159307	0,159586	0,159592	0,159592
40	−10,985642	−0,235642	0,072378	0,160250	0,160466	0,160469	0,160469
45	−12,789345	−0,342671	0,072364	0,160913	0,161127	0,161152	0,161152

Jest to jedyne zadanie testowe w tym rozdziale, które zawiera nietrywialne ograniczenia. Ogólne zagadnienie sterowania optymalnego z ograniczeniami omówimy w następnym rozdziale.

6.4. Programy ewolucyjne a inne metody

W tym punkcie porównamy otrzymane wyżej wyniki z dokładnymi rozwiązaniami oraz z rozwiązaniami uzyskanymi za pomocą pakietu obliczeniowego GAMS.

6.4.1. Zadanie liniowo-kwadratowe

Dokładne rozwiązanie zadania dla wartości parametrów podanych w tabl. 6.1 obliczono za pomocą wzorów (6.3) i (6.4).

Dla podkreślenia działania i konkurencyjności programu ewolucyjnego rozwiązano te same zadania testowe za pomocą pakietu GAMS. Porównanie nie jest całkowicie uczciwe w stosunku do programu ewolucyjnego, gdyż GAMS używa metody poszukiwań szczególnie przystosowanej do zadań liniowo-kwadratowych. Tak więc zadanie (6.1)-(6.2) musiało być łatwym przypadkiem dla tego pakietu. Jeżeli jednak dla takich zadań testowych program ewolucyjny okazałby się konkurencyjny, lub bliski takiemu, byłaby to wskazówka, że powinien on zachowywać się w ogólnym przypadku lepiej. W tablicy 6.5 podano wyniki, przy czym w kolumnie D podano procentowy błąd względny.

Tablica 6.5. Porównanie rozwiązań dla zadania liniowo-kwadratowego

	Rozwiązanie dokładne	Program ewolucyjny		GAMS	
Przypadek	Wartość	Wartość	*D*	Wartość	*D*
I	16180,3399	16180,3928	0,000%	16180,3399	0,000%
II	109160,7978	109161,0138	0,000%	109160,7978	0,000%
III	10009990,0200	10010041,3789	0,000%	10009990,0200	0,000%
IV	37015,6212	37016,0426	0,000%	37015,6212	0,000%
V	287569,3725	287569,4357	0,000%	287569,3725	0,000%
VI	16180,3399	16180,4065	0,000%	16180,3399	0,000%
VII	16180,3399	16180,3784	0,000%	16180,3399	0,000%
VIII	10000,5000	10000,5000	0,000%	10000,5000	0,000%
IX	431004,0987	431004,4182	0,000%	431004,0987	0,000%
X	10000,9999	10001,0038	0,000%	10000,9999	0,000%

Jak widać, działanie GAMS dla zadania liniowo-kwadratowego jest doskonałe. Jednak nie jest już tak w przypadku drugiego zadania testowego.

6.4.2. Zadanie zbierania plonów

Przede wszystkim żadne rozwiązanie GAMS nie pokrywało się z analitycznym. Różnica między rozwiązaniami wzrastała wraz z horyzontem optymalizacji, jak pokazano w tabl. 6.6, a dla $N > 4$ system nie był w stanie znaleźć jakiegokolwiek rozwiązania.

Tablica 6.6. Porównanie rozwiązań dla zadania zbierania plonów

N	Rozwiązanie dokładne	GAMS		GAMS +		Algorytm genetyczny	
		Wartość	D	Wartość	D	Wartość	D
2	6,331738	4,3693	30,99%	6,3316	0,00%	6,3317	0,000%
4	12,721038	5,9050	53,58%	12,7210	0,00%	12,7210	0,000%
8	25,905710	*		18,8604	27,20%	25,9057	0,000%
10	32,820943	*		22,9416	30,10%	32,8209	0,000%
20	73,237681	*		*		73,2376	0,000%
45	279,275275	*		*		279,2714	0,001%

* oznacza, że GAMS nie znalazł rozsądnej wartości.

Wydaje się, że GAMS jest wrażliwy na niewypukłość zadania optymalizacji i na liczbę zmiennych. Nie bardzo pomogło nawet dodanie do zadania dodatkowego ograniczenia ($u_{k+1} > 0{,}1u_k$) w celu zawężenia zbioru rozwiązań dopuszczalnych tak, aby algorytm GAMS nie „gubił się"[1] (patrz kolumna „GAMS+"). Jak wynika z tej kolumny, dla dostatecznie długich horyzontów optymalizacji nie ma szansy uzyskania z GAMS zadowalającego rozwiązania.

6.4.3. Zadanie pchania wózka

Dla zadania pchania wózka zarówno GAMS, jak i program ewolucyjny dały bardzo dobre wyniki (tabl. 6.7). Jednak warto zwrócić uwagę na zależność między czasami, jakich potrzebują oba algorytmy do wykonania zadania.

W większości programów optymalizacyjnych czas potrzebny do zbiegnięcia się algorytmu zależy od liczby zmiennych decyzyjnych. Ta zależność dla programowania dynamicznego jest wykładnicza (przekleństwo wymiarowości). Dla metod szukania (takich jak GAMS) jest ona zwykle „gorsza niż liniowa".

W tablicy 6.8 podano liczbę iteracji, jakiej potrzebował program ewolucyjny do osiągnięcia dokładnego rozwiązania (z zaokrągleniem do sześciu miejsc dziesiętnych), potrzebny na to czas i całkowity czas wykonania 40 000 iteracji (dla nieznanego dokładnego rozwiązania nie jesteśmy w stanie podać dokładności uzyskiwanego rozwiązania). Jest także dany czas obliczeń dla GAMS. Zauważmy jednak, że GAMS był liczony na PC Zenith z-386/20, a program ewolucyjny na stacji DEC-3100.

[1] Jest to „nieuczciwe" w stosunku do algorytmu genetycznego, który pracował bez żadnej pomocy.

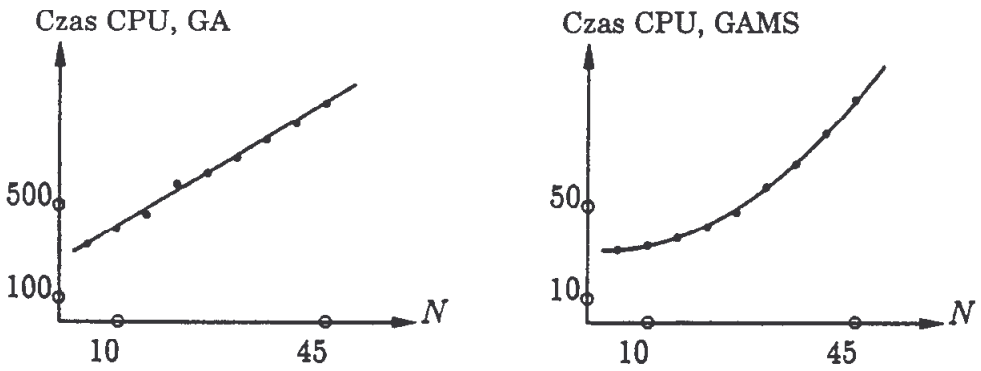
Tablica 6.7. Porównanie rozwiązań dla zadania pchania wózka

	Rozwiązanie dokładne	GAMS		AG	
N	Wartość	Wartość	*D*	Wartość	D
5	0,120000	0,120000	0,000%	0,120000	0,000%
10	0,142500	0,142500	0,000%	0,142500	0,000%
15	0,150370	0,150370	0,000%	0,150370	0,000%
20	0,154375	0,154375	0,000%	0,154375	0,000%
25	0,156800	0,156800	0,000%	0,156800	0,000%
30	0,158426	0,158426	0,000%	0,158426	0,000%
35	0,159592	0,159592	0,000%	0,159592	0,000%
40	0,160469	0,160469	0,000%	0,160469	0,000%
45	0,161152	0,161152	0,000%	0,161152	0,000%

Tablica 6.8. Czasy obliczeń programu ewolucyjnego i GAMS dla zadania pchania wózka (6.9)-(6.10)

N	Potrzebna liczba iteracji	Potrzebny czas (CPU[s])	Czas dla 40000 iteracji (CPU[s])	Czas dla GAMS (CPU[s])
5	6234	65,4	328,9	31,5
10	10231	109,7	400,9	33,1
15	19256	230,8	459,8	36,6
20	19993	257,8	590,8	41,1
25	18804	301,3	640,4	47,7
30	22976	389,5	701,9	58,2
35	23768	413,6	779,5	68,0
40	25634	467,8	850,7	81,3
45	28756	615,9	936,3	95,9

Jest jasne, że program ewolucyjny jest wolniejszy od GAMS – występuje różnica zarówno w bezwzględnych czasach CPU, jak i używanych komputerach. Porównajmy jednak nie czasy potrzebne obu systemom do zakończenia obliczeń, ale raczej szybkość wzrostu tych czasów w funkcji rozmiaru zadania. Na rysunku 6.2 pokazano szybkość wzrostu potrzebną do uzyskania wyników dla programu ewolucyjnego i GAMS.



Rys. 6.2. Czas obliczeń w funkcji wymiarowości zadania (*N*)

Te wykresy mówią same za siebie. Chociaż program ewolucyjny jest ogólnie wolniejszy, jego szybkość wzrostu, zbliżona do liniowej, jest znacznie większa niż dla GAMS (która jest przynajmniej kwadratowa). Podobne wyniki uzyskano dla zadania liniowo-kwadratowego i zadania zbierania plonów.

6.4.4. Znaczenie mutacji nierównomiernej

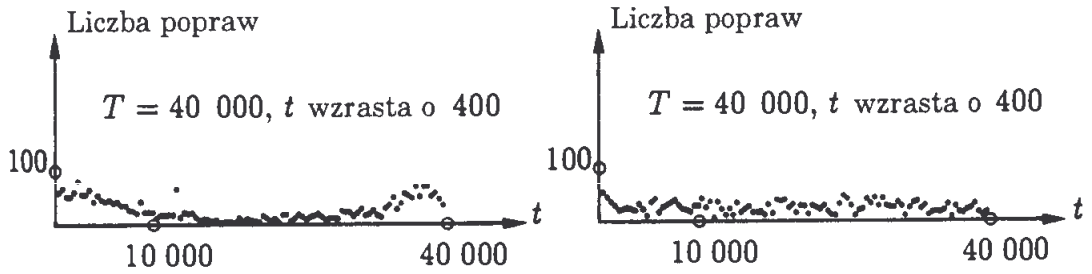
Ciekawe jest porównanie tych wyników z rozwiązaniami dokładnymi oraz otrzymanymi za pomocą innego algorytmu genetycznego, dokładnie takiego samego, ale z *mutacją nierównomierną*. W tablicy 6.9 podsumowano wyniki. W kolumnie D podano procentowy błąd względny.

Tablica 6.9. Porównanie rozwiązań dla liniowo-kwadratowego zadania sterowania dynamicznego

Przypadek	Rozwiązanie dokładne	AG z mutacją nierównomierną		AG bez mutacji nierównomiernej	
	Wartość	Wartość	*D*	Wartość	D
I	16180,3399	16180,3939	0,000%	16234,3233	0,334%
II	109160,7978	109163,0278	0,000%	113807,2444	4,257%
III	10009990,0200	10010391,3989	0,004%	10128951,4515	1,188%
IV	37015,6212	37016,0806	0,001%	37035,5652	0,054%
V	287569,3725	287569,7389	0,000%	298214,4587	3,702%
VI	16180,3399	16180,6166	0,002%	16238,2135	0,358%
VII	16180,3399	16188,2394	0,048%	17278,8502	6,786%
VIII	10000,5000	10000,5000	0,000%	10000,5000	0,000%
IX	431004,0987	431004,4092	0,000%	431610,9771	0,141%
X	10000,9999	10001,0045	0,000%	10439,2695	4,380%

Algorytm genetyczny z *mutacją nierównomierną* wyraźnie bije pozostałe, jeśli chodzi o dokładność uzyskanego rozwiązania optymalnego. Podczas gdy rzadko myli się on o więcej niż kilka tysięcznych procenta, to inne uzyskują z trudem jeden procent. Ponadto zbiegał się on szybciej do rozwiązania.

Ilustracją efektu działania *mutacji nierównomiernej* na proces ewolucyjny jest rys. 6.3. Nowa mutacja powoduje spory przyrost liczby popraw zaobserwowanych w populacji w jej końcówce życia. Mniejsze liczby takich popraw przed tym czasem łącznie z faktycznie szybszą zbieżnością wyraźnie wskazują na lepsze całkowite przeszukiwanie.



Rys. 6.3. Liczba popraw w przypadku I – liniowo-kwadratowego zadania sterowania dynamicznego

6.5. Wnioski

W tym rozdziale badaliśmy nowy operator, mutację nierównomierną, wprowadzony w celu poprawy możliwości dokładnego lokalnego dostrajania się algorytmów genetycznych. Eksperymenty wypadły pomyślnie dla trzech zadań sterowania optymalnego z czasem dyskretnym, wybranych jako zadania testowe. Wyniki były szczególnie zachęcające, jeżeli chodzi o bliskość rozwiązań numerycznych z analitycznymi. Także czasy obliczeń były rozsądne (dla 40 000 pokoleń kilka minut CPU na CRAY Y-MP i do 15 minut na stacji DEC-3100).

Wyniki numeryczne porównano z uzyskanymi za pomocą pakietu obliczeniowego GAMS, korzystającego z metod przeszukiwania. Podczas gdy program ewolucyjny dawał wyniki porównywalne z rozwiązaniami analitycznymi dla wszystkich zadań testowych, to GAMS zawiódł dla jednego z nich. Opracowany program ewolucyjny wykazał pewne dodatnie cechy, nie zawsze obecne w innych (gradientowych) systemach:

- Optymalizowana funkcja dla programu ewolucyjnego nie musi być ciągła. Tymczasem niektóre pakiety optymalizacji w ogóle nie akceptują takich funkcji.
- Niektóre pakiety funkcjonują na zasadzie wszystko albo nic. Użytkownik musi czekać, aż program skończy się. Niekiedy nie ma możliwości uzyskania częściowych (lub przybliżonych) wyników we wczesnych etapach. Programy ewolucyjne dają użytkownikowi dodatkową możliwość, gdyż może on sprawdzać „stan poszukiwań" w czasie obliczeń i podejmować odpowiednie decyzje. W szczególności użytkownik może zadać czas obliczeń, za jaki jest gotów zapłacić (dłuższy czas prowadzi do lepszej dokładności odpowiedzi).
- Złożoność obliczeniowa programów ewolucyjnych wzrasta liniowo, gdy tymczasem większość metod przeszukiwania jest bardzo wrażliwa na długość horyzontu optymalizacji. Jak zwykle możemy łatwo poprawić działanie systemu, stosując oprogramowanie równoległe. Jest to często trudne dla innych metod optymalizacji.

Ostatnio pojawiło się wiele ciekawych rozwiązań w dziedzinie algorytmów ewolucyjnych do poprawy możliwości ich dokładnego lokalnego dopasowania. Są to między innymi algorytm delta-kodowania, dynamiczne kodowanie parametrów, strategia ARGOT, strategie IRM (mechanizmu odpornej rekrutacji), rozszerzone programowanie ewolucyjne, ewolucja ziarnista czy algorytm genetyczny odcinkowy. Krótki opis tych rozwiązań przedstawimy w ostatnim punkcie rozdz. 8, po omówieniu systemu GENOCOP (rozdz. 7) i strategii ewolucyjnych (rozdz. 8).

Podejmuje się także inne działania mające na celu (bardziej lub mniej bezpośrednie) dokładne dostrojenie lokalne. Obejmują one badania Arabasa i innych [11], gdzie wprowadzono dla strategii ewolucyjnych krzyżowania adaptacyjne pośrednie i jednorodne (patrz rozdz. 8). Natomiast Hinterding [184] eks-

perymentował z mutacją genów (które odpowiadają zmiennym) w odróżnieniu od mutacji bitów. Z tego punktu widzenia autorzy analizowali istotność kodowania (Graya czy binarne), ziarnistość oraz częstość mutacji genów.

Ciekawe wyniki przedstawili także Srinivas i Patnaik [368], którzy eksperymentowali z adaptacyjnymi prawdopodobieństwami mutacji i krzyżowania dla utrzymania różnorodności populacji (i utrzymania możliwości zbiegania się algorytmu). W ich podejściu prawdopodobieństwa tych operatorów zmieniały się w zależności od wartości dopasowań rozwiązań. „Dobre" rozwiązania są chronione, a „słabe" rozrywane. Dokładniej

$$p_c = \begin{cases} k_1 \cdot (f_{max} - f')/(f_{max} - \bar{f}), & \text{jeżeli } f' \leqslant \bar{f} \\ k_3, & \text{w pozostałym przypadku} \end{cases}$$

oraz

$$p_m = \begin{cases} k_2 \cdot (f_{max} - f)/(f_{max} - \bar{f}), & \text{jeżeli } f \leqslant \bar{f} \\ k_4, & \text{w pozostałym przypadku} \end{cases}$$

gdzie k_1 oraz k_2 są dodatnimi stałymi (nie większymi niż jeden), f_{max} i $\bar{f}$ – oznaczają odpowiednio wartość maksymalną i wartość średnią funkcji dopasowania f w bieżącej populacji, f oznacza wartość funkcji dopasowania dla danego rozwiązania, a f' większą wartość (dla dwóch rozwiązań wybranych do krzyżowania). Zauważmy, że: (1) wartość $f_{max} - \bar{f}$ jest podstawowa dla powyższego wyrażenia, jest ona także ważna przy ocenie zbieżności algorytmu, (2) p_c i p_m są zerami dla rozwiązania z największym dopasowaniem, (3) $p_c = k_1$ i $p_m = k_2$ dla rozwiązania z $f = \bar{f}$, (4) $p_c = k_3$ i $p_m = k_4$ dla rozwiązań znajdujących się poniżej średniej. Właściwy wybór wartości k_1, k_2, k_3 i k_4 oraz wyniki obliczeń są przedstawione w [368].

7

Zadania z ograniczeniami

Pewien podróżny poszukujący Boga zapytał Mistrza:
– Jak odróżnić prawdziwego nauczyciela od fałszywego,
gdy już powrócę do swojej ojczyzny?
Mistrz odrzekł:
– Dobry nauczyciel mówi jak postępować,
zły – opowiada teorie.
– Ale jak będę mógł odróżnić
dobre postępowanie od złego?
– W taki sam sposób, jak rolnik odróżnia
uprawę dobrą od złej.

Anthony de Mello, *Minuta mądrości*[1)]

Ogólne zadanie programowania nieliniowego ($\mathcal{NLP}$) polega na wyznaczeniu $\mathbf{x}$, dla którego zachodzi

$$\text{optimum } f(\mathbf{x}), \text{ gdzie } \mathbf{x} = (x_1, \ldots, x_q) \in R^q$$

przy $p \geqslant 0$ ograniczeniach równościowych

$$c_i(\mathbf{x}) = 0, \quad i = 0, \ldots, p$$

i $m - p \geqslant 0$ ograniczeniach nierównościowych

$$c_i(\mathbf{x}) \leqslant 0, \quad i = p + 1, \ldots, m$$

Nie są znane metody wyznaczania globalnego maksimum (lub minimum) dla ogólnego zadania programowania nieliniowego. Optimum lokalne można niekiedy wyznaczyć, jeżeli funkcja celu f i ograniczenia c_i spełniają pewne warunki. Opracowano kilka algorytmów dla zadań bez ograniczeń (na przykład metoda bezpośredniego przeszukiwania, metoda gradientowa) i zadań z ograniczeniami (te algorytmy klasyfikuje się zazwyczaj jako metody bezpośrednie i pośrednie). W metodach pośrednich poszukuje się rozwiązania, tworząc jedno lub więcej zadań liniowych z zadania początkowego. W metodach bezpośrednich próbuje się natomiast osiągnąć punkty dopuszczalne. W tym celu przekształca się początkowe zadanie do zadania bez ograniczeń i używa się następnie metod gradientowych z pewnymi modyfikacjami [387]. Mimo aktywnych badań i postępu w optymalizacji globalnej w ostatnich latach [114], można chyba uczciwie powiedzieć, że nie widać na razie nadziei na opracowanie

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

efektywnej procedury rozwiązywania w ten sposób ogólnego zadania nieliniowego. Jak napisano w [172]:

> Nadzieje na opracowanie jednego ogólnego programu, który by pasował do każdego modelu nieliniowego, są nierealne. Zamiast tego należy próbować wybrać taki program, który będzie pasował do rozwiązywanego zadania. Jeżeli zadanie nie pasuje do żadnej kategorii oprócz „ogólnej" lub jeżeli należy koniecznie znaleźć globalne optimum (z wyjątkiem, kiedy nie ma szans na napotkanie wielokrotnych lokalnych optimów), trzeba być przygotowanym na zastosowanie metod, które sprowadzają się do całkowitego przeszukiwania, co jest zadaniem trudnym do wykonania.

Istnieją też inne trudności związane z zastosowaniem tradycyjnych metod optymalizacyjnych. Na przykład większość opracowanych metod ma lokalny zasięg, zależą one od istnienia pochodnych i nie są dostatecznie odporne na nieciągłości, rozległe wielomodalności lub występowanie zakłóceń w przeszukiwanej przestrzeni. Dlatego warto sprawdzić inne (heurystyczne) metody, które mogą być przydatne w wielu zadaniach praktycznych.

W tym rozdziale omówimy kilka metod, które opracowano dla zadań programowania nieliniowego. Zaczniemy od opisu systemu GENOCOP przeznaczonego dla wypukłych przestrzeni przeszukiwań. W następnych punktach prześledzimy inne podejścia ewolucyjne do zadań programowania nieliniowego i opiszemy dwa systemy: GENOCOP II i GENOCOP III.

7.1. Program ewolucyjny: system GENOCOP

Wiele osób [78], [408] badało algorytmy genetyczne z reprezentacjami zmiennopozycyjnymi. Jednak zadanie optymalizacji, jakie rozważali, było zdefiniowane tylko w przestrzeni $\mathscr{D} \subseteq R^q$, gdzie $\mathscr{D} = \prod_{k=1}^{q} \langle l_k, r_k \rangle$, to znaczy każda zmienna x_k była ograniczona przez zadany przedział $\langle l_k, r_k \rangle (1 \leqslant k \leqslant q)$. Tymczasem wydaje się, że powinno się rozważać także inne ograniczenia. Jak zauważono w [66]:

> Trochę obserwacji i zastanowienia wystarczy, aby zauważyć, że wszystkie praktyczne zadania optymalizacyjne są faktycznie zadaniami z ograniczeniami. Przypuśćmy, że znamy wyrażenie opisujące końcowy związek powstający w naczyniu z zachodzącymi reakcjami chemicznymi i chcemy zmaksymalizować ilość otrzymanego związku. Musimy wtedy wziąć pod uwagę ograniczenia stechiometryczne wynikające z bilansu materiałowego substancji reagujących i produktu końcowego, prawa rządzące przepływem materiału do i z naczynia oraz inne warunki. Wszystko to stanowi dodatkowe ograniczenia na zmienne optymalizowanej funkcji.

W zadaniach optymalizacji z ograniczeniami kształt geometryczny zbioru rozwiązań w R^q jest chyba najbardziej decydującym czynnikiem charakteryzującym zadanie, jeżeli chodzi o stopień trudności, jaki można napotkać przy próbie rozwiązania tego zadania [66]. Jest tylko jeden specjalny typ zbiorów – zbiór wypukły – dla którego opracowano liczącą się teorię.

W tym punkcie rozważymy następujące zadanie optymalizacji:

optymalizuj $f(x_1, ..., x_q) \in R$

gdzie $(x_1, ..., x_q) \in \mathscr{D} \subseteq R^q$ i $\mathscr{D}$ jest *zbiorem wypukłym*.

Dziedzina $\mathscr{D}$ jest określona przez zakres określoności zmiennych ($l_k \leqslant x_k \leqslant r_k$ dla $k = 1, ..., q$) i przez zbiór ograniczeń $\mathscr{C}$. Z wypukłości zbioru $\mathscr{D}$ wynika, że w każdym punkcie przeszukiwanej przestrzeni $(x_1, ..., x_q) \in \mathscr{D}$ istnieje dopuszczalny zakres zmienności $\langle left(k), right(k)\rangle$ zmiennej x_k ($1 \leqslant k \leqslant q$) przy ustalonych pozostałych zmiennych x_i ($i = 1, ..., k-1, k+1, ..., q$). Inaczej mówiąc, dla danego $(x_1, ..., x_k, ..., x_q) \in \mathscr{D}$

$$y \in \langle left(k), right(k)\rangle \text{ wtedy i tylko wtedy, gdy } (x_1, ..., x_{k-1}, y, x_{k+1} ..., x_q) \in \mathscr{D}$$

gdzie wszystkie x_i ($i = 1, ..., k-1, k+1, ..., q$) są stałe. Założymy także, że zakresy $\langle left(k), right(k)\rangle$ można obliczyć[1].

Dla przykładu, jeżeli $\mathscr{D} \subseteq R^2$ zdefiniujemy następująco:

$$-3 \leqslant x_1 \leqslant 3$$

$$0 \leqslant x_2 \leqslant 8$$

$$x_1^2 \leqslant x_2 \leqslant x_1 + 4$$

to dla danego punktu $(2,5) \in \mathscr{D}$

$$left(1) = 1,\ right(1) = \sqrt{5}$$
$$left(2) = 4,\ right(2) = 6$$

Oznacza to, że pierwsza składowa wektora (2,5) może się zmieniać od 1 do $\sqrt{5}$ (przy stałym $x_2 = 5$), a druga składowa od 4 do 6 (przy stałym $x_1 = 2$).

Oczywiście, jeżeli zbiór ograniczeń $\mathscr{C}$ jest pusty, to przeszukiwana przestrzeń $\mathscr{D} = \prod_{k=1}^{q} \langle l_k, r_k\rangle$ jest wypukła, a przy tym $left(k) = l_k$ oraz $right(k) = r_k$ dla $k = 1, ..., q$.

---

[1] Zauważmy, że dla punktów leżących na brzegu dziedziny $\mathscr{D}$ może zachodzić $left(k) = right(k)$, to znaczy zakres zmienności może się ograniczyć do jednego punktu. Jest tak na przykład dla punktu $x_1 = \frac{1+\sqrt{17}}{2}$, $x_2 = \frac{9+\sqrt{17}}{2}$ w przedstawionym tuż dalej przykładzie (przyp. tłum.).

Powyższa właściwość jest bardzo ważna dla wszystkich operatorów mutacji. Jeżeli zmienna x_k ma być zmutowana, to zakres mutacji wynosi $\langle left(k), right(k)\rangle$, a potomek jest zawsze dopuszczalny.

Inna właściwość wypukłych przestrzeni przeszukiwań gwarantuje, że dla każdych dwóch punktów $\mathbf{x}_1$ i $\mathbf{x}_2$ ze zbioru rozwiązań $\mathscr{D}$ kombinacja liniowa $a\mathbf{x}_1 + (1 - a)\mathbf{x}_2$, gdzie $a \in [0, 1]$, jest także punktem w $\mathscr{D}$. Jest to istotna właściwość dla krzyżowania arytmetycznego.

Rozważymy szczególną klasę zadań optymalizacji, które są zdefiniowane na wypukłych dziedzinach. Te zadania można sformułować następująco:

Optymalizuj funkcję $f(x_1, x_2, ..., x_q)$ przy następujących ograniczeniach liniowych:

1. Ograniczenia nałożone na dziedzinę: $l_i \leqslant x_i \leqslant u_i$ dla $i = 1, 2, ..., q$. Zapiszemy to $\mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}$, gdzie $\mathbf{l} = \langle l_1, ..., l_q\rangle$, $\mathbf{u} = \langle u_1, ..., u_q\rangle$, $\mathbf{x} = \langle x_1, ..., x_q\rangle$.
2. Ograniczenia równościowe: $A\mathbf{x} = \mathbf{b}$, gdzie $\mathbf{x} = \langle x_1, ..., x_q\rangle$, $A = (a_{ij})$, $\mathbf{b} = \langle b_1, ..., b_p\rangle$, $1 \leqslant i \leqslant p$ i $1 \leqslant j \leqslant q$ (p jest liczbą równości).
3. Ograniczenia nierównościowe: $C\mathbf{x} \leqslant \mathbf{d}$, gdzie $\mathbf{x} = \langle x_1, ..., x_q\rangle$, $C = (c_{ij})$, $\mathbf{d} = \langle d_1, ..., d_m\rangle$, $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant q$ (m jest liczbą nierówności).

To sformułowanie jest dostatecznie ogólne, aby objąć dużą klasę występujących w badaniach operacyjnych zadań optymalizacji z liniowymi ograniczeniami i dowolną funkcją celu. Rozważony dalej przykład nieliniowego zadania transportowego jest jednym z zadań w tej klasie.

Opracowany system GENOCOP (*GEnetic algorithm for Numerical Optimization for COnstrained Problems*) obejmuje sposób postępowania z ograniczeniami, który jest zarówno ogólny, jak i niezależny od zadania. Zawiera on pewne wcześniejsze pomysły, ale przedstawione w zupełnie innym kontekście. Główna idea polega na: (1) eliminacji ograniczeń równościowych, (2) starannym określeniu specjalnych operatorów genetycznych, które gwarantują, że wszystkie chromosomy zawierają się w ograniczonej przestrzeni rozwiązań. Można to zrobić efektywnie przynajmniej dla liniowych ograniczeń, które obejmują wiele ciekawych zadań optymalizacji, chociaż otrzymane wyniki niekoniecznie przenoszą się na przypadki z ograniczeniami nieliniowymi.

W niektórych metodach optymalizacji, jak na przykład w programowaniu liniowym, ograniczenia równościowe są chętnie widziane, gdyż wiadomo, że optimum, jeżeli istnieje, znajduje się na ograniczeniu zbioru wypukłego. Nierówności zamienia się na równości przez dodanie pomocniczych zmiennych i poszukuje się rozwiązania, przechodząc z jednego wierzchołka do następnego wokół zbioru.

Natomiast dla metody, w której rozwiązania są generowane przypadkowo, takie ograniczenia równościowe są zawadą. W programie GENOCOP są one eliminowane na samym początku, łącznie z taką samą liczbą poszukiwanych zmiennych. Tym sposobem odcina się część przeszukiwanej przestrzeni. Pozostałe ograniczenia, w postaci nierówności liniowych, wyznaczają zbiór wypuk-

ły, który należy przeszukać, aby znaleźć rozwiązanie. Wypukłość przeszukiwanej przestrzeni zapewnia, że liniowa kombinacja rozwiązań jest rozwiązaniem dopuszczalnym, bez konieczności sprawdzania ograniczeń. Ten fakt jest wykorzystywany cały czas w opisywanym programie. Nierówności mogą być używane do tworzenia ograniczeń na poszczególne zmienne. Są to ograniczenia dynamiczne, gdyż zależą od wartości innych zmiennych i muszą być za każdym razem obliczane.

Przypuśćmy, że zbiór ograniczeń liniowych jest przedstawiony w postaci macierzowej

$$A\mathbf{x} = \mathbf{b}$$

Założymy, że wśród nich jest p równań niezależnych (istnieją proste sposoby sprawdzenia tego), to znaczy, że p zmiennych $x_{i_1}, x_{i_2}, \ldots, x_{i_p}$ ($\{i_1, \ldots, i_p\} \subseteq \{1, 2, \ldots, q\}$) jest określonych przez inne zmienne. Mogą one być wyeliminowane z zadania w następujący sposób.

Dzielimy macierz A pionowo na dwie macierze A_1 i A_2 tak, aby j-ta kolumna macierzy A należała do A_1 wtedy i tylko wtedy, gdy $j \in \{i_1, \ldots, i_p\}$. Wtedy istnieje A_1^{-1}. Podobnie dzielimy macierz C i wektory $\mathbf{x}$, $\mathbf{l}$ i $\mathbf{u}$ (to znaczy są one równe $\mathbf{x}_1 = \langle x_{i_1}, \ldots, x_{i_p}\rangle$, $\mathbf{l}_1 = \langle l_{i_1}, \ldots, l_{i_p}\rangle$ i $\mathbf{u}_1 = \langle u_{i_1}, \ldots, u_{i_p}\rangle$). Wtedy

$$A_1\mathbf{x}^1 + A_2\mathbf{x}^2 = \mathbf{b}$$

skąd łatwo wynika

$$\mathbf{x}^1 = A_1^{-1}\mathbf{b} - A_1^{-1}A_2\mathbf{x}^2$$

Stosując powyższy sposób, możemy wyeliminować zmienne $x_{i_1}, \ldots, x_{i_p}$, zamieniając je liniowymi kombinacjami pozostałych zmiennych. Jednak każda zmienna x_{i_j} ($j = 1, 2, \ldots, p$) jest dodatkowo ograniczona przez nierówności $l_{i_j} \leqslant x_{i_j} \leqslant u_{i_j}$. Eliminując wszystkie zmienne x_{i_j}, musimy więc dodać do początkowego zbioru nierówności nowy zbiór

$$\mathbf{l}_1 \leqslant A_1^{-1}\mathbf{b} - A_1^{-1}A_2\mathbf{x}^2 \leqslant \mathbf{u}_1$$

Początkowy zbiór nierówności

$$C\mathbf{x} \leqslant \mathbf{d}$$

można przedstawić w postaci

$$C_1\mathbf{x}^1 + C_2\mathbf{x}^2 \leqslant \mathbf{d}$$

co można przekształcić do

$$C_1(A_1^{-1}\mathbf{b} - A_1^{-1}A_2\mathbf{x}^2) + C_2\mathbf{x}^2 \leqslant \mathbf{d}$$

Tak więc, po wyeliminowaniu p zmiennych $x_{i_1}, \ldots, x_{i_p}$, końcowy zbiór ograniczeń składa się tylko z następujących nierówności:

1. Początkowych ograniczeń $\mathbf{l}_2 \leqslant \mathbf{x}^2 \leqslant \mathbf{u}_2$.
2. Nowych nierówności $\mathbf{l}_1 \leqslant A_1^{-1}\mathbf{b} - A_1^{-1}A_2\mathbf{x}^2 \leqslant \mathbf{u}_1$.
3. Początkowych nierówności (po usunięciu zmiennych x^1)
   $(C_2 - C_1A_1^{-1}A_2)\mathbf{x}^2 \leqslant \mathbf{d} - C_1A_1^{-1}\mathbf{b}$.

7.1.1. Przykład

Zacznijmy od przykładu i załóżmy, że chcemy optymalizować funkcję sześciu zmiennych

$$f(x_1, x_2, x_3, x_4, x_5, x_6)$$

przy następujących ograniczeniach:

$$2x_1 + x_2 + x_3 = 6$$
$$x_3 + x_5 - 3x_6 = 10$$
$$x_1 + 4x_4 = 3$$
$$x_2 + x_5 \leqslant 120$$
$$-40 \leqslant x_1 \leqslant 20,\ 50 \leqslant x_2 \leqslant 75$$
$$0 \leqslant x_3 \leqslant 10,\ 5 \leqslant x_4 \leqslant 15$$
$$0 \leqslant x_5 \leqslant 20,\ -5 \leqslant x_6 \leqslant 5$$

Możemy skorzystać z obecności trzech niezależnych równości i wyrazić trzy zmienne jako funkcje pozostałych czterech:

$$x_1 = 3 - 4x_4$$
$$x_2 = -10 + 8x_4 + x_5 - 3x_6$$
$$x_3 = 10 - x_5 + 3x_6$$

W ten sposób zredukowaliśmy początkowe zadanie do zadania optymalizacji z funkcją celu zależną od trzech zmiennych x_4, x_5 i x_6:

$$g(x_4, x_5, x_6) = f((3 - 4x_4), (-10 + 8x_4 + x_5 - 3x_6), (10 - x_5 + 3x_6), x_4, x_5, x_6)$$

przy następujących ograniczeniach (tylko nierównościowych):

$-10 + 8x_4 + 2x_5 - 3x_6 \leqslant 120$, (początkowe ograniczenie $x_2 + x_5 \leqslant 120$),
$50 \leqslant 3 - 4x_4 \leqslant 20$, (początkowe ograniczenie $50 \leqslant x_1 \leqslant 20$),

$-20 \leqslant -10 + 8x_4 + x_5 - 3x_6 \leqslant 75$, (początkowe ograniczenie $-20 \leqslant x_2 \leqslant 75$),
$0 \leqslant 10 - x_5 + 3x_6 \leqslant 10$, (początkowe ograniczenie $0 \leqslant x_3 \leqslant 10$),
$5 \leqslant x_4 \leqslant 15$, $0 \leqslant x_5 \leqslant 20$ i $-5 \leqslant x_6 \leqslant 5$.

Można je jeszcze bardziej uprościć, na przykład nierówność drugą i piątą można zastąpić jedną

$$5 \leqslant x_4 \leqslant 10{,}75$$

To przekształcenie kończy pierwszy krok naszego algorytmu, eliminację równości. Uzyskana przestrzeń rozwiązań jest oczywiście wypukła. Jak pisaliśmy wcześniej, z wypukłości przestrzeni rozwiązań wynika, że dla każdego punktu dopuszczalnego (x_1, x_2, x_3) istnieje dopuszczalny zakres $\langle left(k), right(k)\rangle$ zmiennej x_k $(1 \leqslant k \leqslant 3)$ przy ustalonych pozostałych zmiennych. Na przykład dla przestrzeni rozwiązań dopuszczalnych zdefiniowanej powyżej i dla dopuszczalnego punktu $(x_4, x_5, x_6) = (10, 8, 2)$

$$left(1) = 7{,}25,\ right(1) = 10{,}375$$

$$left(2) = 6,\ right(2) = 11$$

$$left(3) = 1,\ right(3) = 2{,}666$$

($left(1)$ i $right(1)$ są zakresami pierwszej składowej wektora (10, 8, 2), to znaczy zmiennej x_4 itd.). Oznacza to, że pierwsza składowa wektora (10, 8, 2) może się zmieniać od 7,25 do 10,375 (gdy $x_5 = 8$ i $x_6 = 2$ są stałe), druga składowa tego wektora może się zmieniać od 6 do 11 (gdy $x_4 = 10$ i $x_6 = 2$ są stałe), a trzecia składowa od 1 do 2,666 (gdy $x_4 = 10$ i $x_5 = 8$ są stałe).

System GENOCOP próbuje znaleźć początkowe (dopuszczalne) rozwiązanie, próbkując rejon rozwiązań dopuszczalnych. Jeżeli pewna zadana wcześniej liczba prób jest nieudana, to system pyta użytkownika o dopuszczalne rozwiązanie początkowe. Populacja początkowa składa się z identycznych kopii takiego punktu początkowego (niezależnie od tego, czy został on znaleziony, czy podany przez użytkownika).

System GENOCOP zawiera kilka operatorów, które okazały się użyteczne w wielu zadaniach testowych. Omówimy je po kolei w następnym punkcie.

7.1.2. Operatory

W tym punkcie opiszemy sześć operatorów genetycznych z reprezentacją zmiennopozycyjną, które są używane w zmodyfikowanej wersji systemu GENOCOP. Pierwsze trzy są operatorami jednoargumentowymi (typu mutacji), a pozostałe trzy są dwuargumentowe (różne typy krzyżowania).

Mutacja równomierna

Ten operator wymaga jednego rodzica **x** i tworzy jednego potomka **x**′. Operator wybiera losowo składową $k \in (1, ..., q)$ wektora $\mathbf{x} = (x_1, ..., x_k, ..., x_q)$ i tworzy $\mathbf{x}' = (x_1, ..., x_k', ..., x_q)$, gdzie x_k' przyjmuje wartość losową (z równomiernym rozkładem prawdopodobieństwa) z przedziału $\langle left(\mathrm{k}), right(\mathrm{k})\rangle$.

Operator ten odgrywa istotną rolę we wczesnych fazach procesu ewolucyjnego, kiedy jest dozwolony dowolny ruch w przestrzeni rozwiązań. W szczególności jest to podstawowy operator, gdy populacja początkowa składa się z wielokrotnej kopii tego samego (dopuszczalnego) punktu. Takie sytuacje mogą się zdarzać całkiem często w zadaniach optymalizacji z ograniczeniami, kiedy użytkownik zadaje punkt początkowy obliczeń. Dodatkowo taki pojedynczy punkt startowy (poza wadami) ma dużą zaletę: pozwala on utworzyć proces iteracyjny, w którym następna iteracja startuje z najlepszego punktu poprzedniej iteracji. Taką metodę stosowano przy opracowaniu systemu dla zadań z nieliniowymi ograniczeniami w przestrzeniach, które nie musiały być wypukłe (GENOCOP II).

Także w późniejszych fazach procesu ewolucyjnego operator ten wprowadza ewentualne skoki daleko od lokalnego optimum w poszukiwaniu lepszego punktu.

Mutacja brzegowa

Ten operator wymaga także jednego rodzica **x** i tworzy jednego potomka **x**′. Jest to odmiana mutacji równomiernej, w której x_k' jest równe albo $left(k)$, albo $right(k)$ z jednakowym prawdopodobieństwem.

Operator ten jest pomyślany dla zadań optymalizacji, w których rozwiązanie optymalne leży albo na, albo blisko brzegu dopuszczalnej przestrzeni przeszukiwań. W rezultacie, jeżeli zbiór ograniczeń $\mathscr{C}$ jest pusty, a ograniczenia na zmienne są oddalone od siebie, operator ten jest zawadą. Ale może się on okazać wyjątkowo przydatny, gdy istnieją ograniczenia. Użyteczność tego operatora można wykazać na prostym przykładzie zadania programowania liniowego. W takim przypadku wiemy, że optimum globalne leży na ograniczeniu przeszukiwanej przestrzeni.

Przykład 7.1

Rozważmy następujące zadanie testowe [387]:

$$\max f(x_1, x_2) = 4x_1 + 3x_2$$

przy następujących ograniczeniach:

$$2x_1 + 3x_2 \leqslant 6$$
$$-3x_1 + 2x_2 \leqslant 3$$
$$2x_1 + x_2 \leqslant 4$$
$$0 \leqslant x_i \leqslant 2,\ i = 1,2$$

Znane optimum globalne wynosi $(x_1, x_2) = (1{,}5,\ 1{,}0)$ i wtedy $f(1{,}5,\ 1{,}9) = 9{,}0$.

Aby zbadać użyteczność tego operatora przy optymalizacji powyższego zadania, wykonano dziesięć obliczeń z wszystkimi operatorami i dziesięć następnych bez mutacji brzegowej. Program z mutacją brzegową znalazł łatwo optimum globalne we wszystkich przypadkach, średnio po 32 pokoleniach, gdy tymczasem bez tego operatora nawet po 100 pokoleniach najlepszy znaleziony punkt (z dziesięciu obliczeń) wynosił $\mathbf{x} = (1{,}501,\ 0{,}997)$ z $f(\mathbf{x}) = 8{,}996$ (najgorszy punkt to $\mathbf{x} = (1{,}576,\ 0{,}847)$ z $f(\mathbf{x}) = 8{,}803$).

Mutacja nierównomierna

Jest to (jednoargumentowy) operator odpowiedzialny za umiejętność dokładnego dostrojenia się systemu. Jest zdefiniowany następująco. Jeżeli do mutacji wybrano element x_k rodzica $\mathbf{x}$, to wynikiem jest $\mathbf{x}' = \langle x_1, \ldots, x_k', \ldots, x_q \rangle$, gdzie

$$x_k' = \begin{cases} x_k + \Delta(t,\ right(k) - x_k), & \text{jeżeli losową liczbą jest } 0 \\ x_k - \Delta(t,\ x_k - left(\mathrm{k})), & \text{jeżeli losową liczbą jest } 1 \end{cases}$$

Funkcja $\Delta(t, y)$ przyjmuje wartości z przedziału $[0, y]$, przy czym prawdopodobieństwo, że $\Delta(t, y)$ jest bliskie 0, wzrasta, gdy t wzrasta (t jest numerem pokolenia). Ta właściwość powoduje, że operator początkowo przeszukuje równomiernie całą przestrzeń (kiedy t jest małe), a w dalszych etapach tylko lokalne otoczenie. Użyto następującej funkcji:

$$\Delta(t, y) = yr(1 - t/T)^b$$

gdzie r jest liczbą losową z przedziału $[0, 1]$, T jest najwyższym numerem pokolenia, a b jest parametrem systemu określającym stopień niejednorodności.

Krzyżowanie arytmetyczne

Ten operator dwuargumentowy definiuje się jako liniową kombinację dwóch wektorów. Jeżeli krzyżowaniu mają podlegać $\mathbf{x}_1$ i $\mathbf{x}_2$, to potomkowie są wyznaczani następująco: $\mathbf{x}_1' = a\mathbf{x}_1 + (1 - a)\mathbf{x}_2$ oraz $\mathbf{x}_2' = a\mathbf{x}_2 + (1 - a)\mathbf{x}_1$. W operatorze tym używa się losowej wartości $a \in [0, 1]$, co zawsze gwarantuje, że są to punkty dopuszczalne ($\mathbf{x}_1', \mathbf{x}_2' \in \mathscr{D}$). Takie krzyżowanie było nazywane *gwarantowanym średnim krzyżowaniem* [77] (gdy $a = 1/2$), *krzyżowaniem pośrednim* [18], *krzyżowaniem liniowym* [408] i *krzyżowaniem arytmetycznym* [268], [269].

Istotność krzyżowania arytmetycznego przedstawimy na następującym przykładzie.

Przykład 7.2

Rozważmy następujące zadanie [114]:

$$\min f(x_1, x_2, x_3, x_4, x_5) = -5\sin(x_1)\sin(x_2)\sin(x_3)\sin(x_4)\sin(x_5) +$$
$$-\sin(5x_1)\sin(5x_2)\sin(5x_3)\sin(5x_4)\sin(5x_5)$$

przy ograniczeniach

$$0 \leqslant x_i \leqslant \pi \quad \text{dla} \quad 1 \leqslant i \leqslant 5$$

Znane minimum globalne to $(x_1, x_2, x_3, x_4, x_5) = (\pi/2, \pi/2, \pi/2, \pi/2, \pi/2)$ z $f(\pi/2, \pi/2, \pi/2, \pi/2, \pi/2) = -6$.

Wydaje się, że program bez krzyżowania arytmetycznego zbiega się wolniej. Po 50 pokoleniach średnia wartość z najlepszych punktów (z 10 obliczeń) wynosiła $-5{,}9814$, a średnia wartość najlepszych punktów po 100 pokoleniach wynosiła $-5{,}9966$. Natomiast te same średnie dla programu z krzyżowaniem arytmetycznym wynosiły odpowiednio $-5{,}9930$ i -5.9996.

Ponadto pojawiło się ciekawe zjawisko, że wyniki przy krzyżowaniu arytmetycznym były bardziej stabilne, ze znacznie mniejszym odchyleniem standardowym najlepszych rozwiązań (otrzymanych z 10 obliczeń).

Krzyżowanie proste

Ten operator dwuargumentowy definiuje się następująco. Przy krzyżowaniu $\mathbf{x}_1 = (x_1, ..., x_q)$ i $\mathbf{x}_2 = (y_1, ..., y_q)$ po pozycji k-tej potomkowie są następujący: $\mathbf{x}_1' = (x_1, ..., x_k, y_{k+1}, ..., y_q)$ oraz $\mathbf{x}_2' = (y_1, ..., y_k, x_{k+1}, ..., x_q)$. Taki operator może utworzyć potomka poza dziedziną $\mathscr{D}$. Aby tego uniknąć, zastosujemy właściwość przestrzeni wypukłych mówiącą, że istnieje takie $a \in [0, 1]$, że

$$\mathbf{x}_1' = \langle x_1, ..., x_k, y_{k+1} a + x_{k+1}(1 - a), ..., y_q a + x_q(1 - a) \rangle$$

i

$$\mathbf{x}_2' = \langle y_1, ..., y_k, x_{k+1} a + y_{k+1}(1 - a), ..., x_q a + y_q(1 - a) \rangle$$

są dopuszczalne.

Jedynym pytaniem bez odpowiedzi jest: jak wyznaczyć największe a, aby uzyskać największą możliwą wymianę informacji. Najprostszą metodą byłoby zacząć od $a = 1$ i, jeżeli potomek nie należy do $\mathscr{D}$, zmniejszać a, mnożąc je przez pewną stałą $1/\rho$. Po ρ próbach $a = 0$ i obaj potomkowie są w $\mathscr{D}$, gdyż są wtedy identyczni ze swoimi rodzicami. Konieczność wyznaczenia takiego największego przyrostu nie jest w ogólności duża i maleje szybko z czasem życia populacji.

Wygląda na to, że zalety krzyżowania prostego są takie same, jak krzyżowania arytmetycznego (do eksperymentowania użyto zadania z testu 6 w następnym punkcie). Wyniki wskazują, że system bez krzyżowania prostego dawał nawet mniej stabilne rozwiązania niż system bez krzyżowania arytmetycznego, a odchylenia standardowe najlepszych rozwiązań otrzymanych z 10 obliczeń były znacznie większe. Także najgorsze rozwiązanie otrzymane po 100 pokoleniach miało wartość $-5{,}9547$, znacznie gorszą od najgorszego rozwiązania otrzymanego przy obecności wszystkich operatorów ($-5{,}9982$) lub najgorszego rozwiązania otrzymanego bez krzyżowania arytmetycznego ($-5{,}9919$).

Krzyżowanie heurystyczne

Ten operator [408] jest wyjątkowy z następujących powodów: (1) do określenia kierunku poszukiwań używa on wartości funkcji celu, (2) tworzy on tylko jednego potomka i (3) może w ogóle nie utworzyć potomka.

Operator ten tworzy jednego potomka $\mathbf{x}_3$ z dwojga rodziców $\mathbf{x}_1$ i $\mathbf{x}_2$ zgodnie z następującą regułą:

$$\mathbf{x}_3 = r(\mathbf{x}_2 - \mathbf{x}_1) + \mathbf{x}_2$$

gdzie r jest liczbą przypadkową leżącą między 0 a 1, zaś rodzic $\mathbf{x}_2$ jest nie gorszy od $\mathbf{x}_1$, to znaczy $f(\mathbf{x}_2) \geqslant f(\mathbf{x}_1)$ dla zadania maksymalizacji i $f(\mathbf{x}_2) \leqslant f(\mathbf{x}_1)$ dla zadania minimalizacji.

Ten operator może utworzyć potomka, który nie jest dopuszczalny. W takim przypadku generuje się inną wartość losową r i tworzy innego potomka. Jeżeli po w próbach operator nie znalazł nowego punktu, który spełnia ograniczenia, to poddaje się i nie tworzy nowego potomka.

Wygląda na to, że krzyżowanie heurystyczne wpływa na dokładność znalezionego rozwiązania. Jego głównymi zadaniami są: (1) dokładne dostrojenie lokalne i (2) poszukiwanie w obiecującym kierunku.

7.1.3. Testowanie systemu GENOCOP

Do oceny systemu GENOCOP wybrano starannie zbiór zadań testowych, które pozwalają przedstawić działanie algorytmu i wykazać, że potrafi on rozwiązywać praktyczne zadania. Poniżej omówimy osiem zadań obejmujących funkcje kwadratowe, nieliniowe i nieciągłe z kilkoma ograniczeniami liniowymi.

Wszystkie obliczenia przeprowadzono na stacji SUN SPARC 2. We wszystkich eksperymentach przyjęto następujące parametry:

pop_size = 70, $k = 28$ (liczba rodziców w etapie klasyfikacji w każdym pokoleniu) i $b = 2$ (współczynnik mutacji nierównomiernej).

(Trzecia wersja systemu GENOCOP jest dostępna na ftp.uncc.edu na ścieżce coe/evol, plik GENOCOP3.0.tar.Z). Każde zadanie było liczone 10 razy. We wszystkich zadaniach liczba pokoleń wynosiła 500 lub 1000 (z wyjątkiem testu 6, gdzie system liczył aż do 10 000 pokoleń). Osiem zadań testowych i wyniki obliczeń przedstawiono poniżej.

Test 1

Zadanie jest następujące [114]:

$$\min f(\mathbf{x}, y) = -10{,}5x_1 - 7{,}5x_2 - 3{,}5x_3 - 2{,}5x_4 - 1{,}5x_5 - 10y - 0{,}5 \sum_{i=1}^{5} x_i^2$$

przy ograniczeniach

$$6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leqslant 6{,}5, \qquad 10x_1 + 10x_3 + y \leqslant 20$$

$$0 \leqslant x_i \leqslant 1, \qquad 0 \leqslant y$$

Rozwiązaniem globalnym jest $(\mathbf{x}^*, y^*) = (0, 1, 0, 1, 1, 20)$ z $f(\mathbf{x}^*, y^*) = -213$.

GENOCOP znalazł rozwiązania bardzo bliskie optimum we wszystkich dziesięciu obliczeniach. Typowy znaleziony punkt to

(0,000000, 1,000000, 0,000000, 0,999999, 1,000000, 20,000000)

z wartością funkcji celu $-213{,}0$. Pojedynczy przebieg dla 1000 iteracji zajmował 21 s czasu CPU.

Test 2

Zadanie jest następujące [186]:

$$\min f(\mathbf{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_i + \ldots + x_{10}} \right)$$

przy ograniczeniach

$$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2, \qquad x_4 + 2x_5 + x_6 + x_7 = 1$$

$$x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1, \qquad x_i \geqslant 0{,}000001, \ (i = 1, \ldots, 10)$$

gdzie

$$c_1 = -6{,}089;\ c_2 = -17{,}164;\ c_3 = -34{,}054;\ c_4 = -5{,}914;\ c_5 = -24{,}721;$$

$$c_6 = -14{,}986;\ c_7 = -24{,}100;\ c_8 = -10{,}708;\ c_9 = -26{,}662;$$

$$c_{10} = -22{,}179$$

Najlepsze poprzednio znane rozwiązanie [186] było następujące:

$$\mathbf{x}^* = (0{,}1773548,\ 0{,}8200180,\ 0{,}8825646,\ 0{,}007233256,\ 0{,}4907851,$$
$$0{,}0004335469,\ 0{,}1727298,\ 0{,}007765639\ 0{,}1984929,\ 0{,}05269826)$$

z $f(\mathbf{x}^*)) = -47{,}707579$.

GENOCOP znalazł punkty z lepszą wartością funkcji celu niż powyższa we wszystkich dziesięciu przebiegach. Najlepsze znalezione rozwiązanie to

$$\mathbf{x}^* = (0{,}04034785,\ 0{,}15386976,\ 0{,}77497089,\ 0{,}00167479,\ 0{,}48468539,$$
$$0{,}00068965,\ 0{,}02826479,\ 0{,}01849179,\ 0{,}03849563,\ 0{,}10128126)$$

z wartością funkcji celu $-47{,}760765$. Pojedynczy przebieg o 1000 iteracji zajął 56 s czasu CPU.

Test 3

Zadanie jest następujące [114]:

$$\min f(\mathbf{x}, \mathbf{y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i$$

przy ograniczeniach

$$\begin{aligned}
&2x_1 + 2x_2 + y_6 + y_7 \leqslant 10, && 2x_1 + 2x_3 + y_6 + y_8 \leqslant 10\\
&2x_2 + 2x_3 + y_7 + y_8 \leqslant 10, && -8x_1 + y_6 \leqslant 0\\
&-8x_2 + y_7 \leqslant 0, && -8x_3 + y_8 \leqslant 0\\
&-2x_4 - y_1 + y_6 \leqslant 0, && -2y_2 - y_3 + y_7 \leqslant 0\\
&-2y_4 - y_5 + y_8 \leqslant 0, && 0 \leqslant x_i \leqslant 1,\ i = 1, 2, 3, 4\\
&0 \leqslant y_i \leqslant 1,\ i = 1, 2, 3, 4, 5, 9, && 0 \leqslant y_i,\ i = 6, 7, 8
\end{aligned}$$

Minimum globalne jest równe $(\mathbf{x}^*, \mathbf{y}^*) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ z $f(\mathbf{x}^*, \mathbf{y}^*) = -15$.

GENOCOP znalazł optimum we wszystkich dziesięciu przebiegach. Typowy punkt optymalny wyglądał następująco:

(1,000000, 1,000000, 1,000000, 1,000000, 0,999995, 1,000000, 0,999999,

1,000000, 1,000000, 2,999984, 2,999995, 2,999995, 0,999999)

z wartością funkcji celu −14,999965. Pojedynczy przebieg o 1000 iteracji zajął 41 s czasu CPU.

Test 4

Zadanie jest następujące [113]:

$$\max f(\mathbf{x}) = \frac{3x_1 + x_2 - 2x_3 + 0{,}8}{2x_1 - x_2 + x_3} + \frac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3}$$

przy ograniczeniach

$$\begin{aligned}
&x_1 + x_2 - x_3 \leqslant 1, && -x_1 + x_2 - x_3 \leqslant -1\\
&12x_1 + 5x_2 + 12x_3 \leqslant 34{,}8, && 12x_1 + 12x_2 + 7x_3 \leqslant 29{,}1\\
&-6x_1 + x_2 + x_3 \leqslant -4{,}1, && 0 \leqslant x_1,\ i = 1, 2, 3
\end{aligned}$$

Maksimum globalnym jest $\mathbf{x}^* = (1, 0, 0)$ i $f(\mathbf{x}^*) = 2{,}471428$.

GENOCOP znalazł optimum we wszystkich dziesięciu przebiegach. Pojedynczy przebieg o 500 iteracjach zajął 9 s czasu CPU.

Test 5

Zadanie jest następujące [64]:

$$\min f(\mathbf{x}) = x_1^{0,6} + x_2^{0,6} + 6x_1 - 4x_3 + 3x_4$$

przy ograniczeniach

$$\begin{aligned} &-3x_1 + x_2 - 3x_3 = 0, && x_1 + 2x_3 \leqslant 4 \\ &x_2 + 2x_4 \leqslant 4, && x_1 \leqslant 3 \\ &x_4 \leqslant 1, && 0 \leqslant x_i,\ i = 1,\ 2,\ 3,\ 4 \end{aligned}$$

Najlepszym globalnym rozwiązaniem jest $\mathbf{x}^* = (4/3,\ 4,\ 0,\ 0)$ z $f(\mathbf{x}^*) = -4{,}5142$.

GENOCOP znalazł ten punkt we wszystkich przebiegach. Pojedynczy przebieg o 500 iteracjach zajął 9 s czasu CPU.

Test 6

Zadanie jest następujące [64]:

$$\min f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + \\ + (1 - x_3)^2 + 10{,}1(x_2 - 1)^2 + (x_4 - 1)^2 + 19{,}8(x_2 - 1)(x_4 - 1)$$

przy ograniczeniach

$$-10{,}0 \leqslant x_i \leqslant 10{,}0, \qquad i = 1, 2, 3, 4$$

Minimum globalne wynosi $\mathbf{x}^* = (1,\ 1,\ 1,\ 1)$ i $f(\mathbf{x}^*) = 0$.

GENOCOP zbliżył się bardzo blisko do optimum we wszystkich dziesięciu przebiegach. Typowy znaleziony punkt to

$$(x_1,\ x_2,\ x_3,\ x_4) = (1{,}000044,\ 1{,}000087,\ 0{,}999954,\ 0{,}999909)$$

z wartością funkcji celu 0,00000001. Pojedynczy przebieg o 10 000 iteracji zajął 159 s czasu CPU.

Test 7

Zadanie jest następujące [114]:

$$\min f(x,\ \mathbf{y}) = 6{,}5x - 0{,}5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5$$

przy ograniczeniach

$$\begin{aligned} &x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leqslant 16 \\ &-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leqslant -1 \\ &2x + 0{,}5y_1 + 0{,}2y_2 - 3y_3 - y_4 - 4y_5 \leqslant 24 \\ &0{,}2x + 2y_1 + 0{,}1y_2 - 4y_3 + 2y_4 + 2y_5 \leqslant 12 \end{aligned}$$

$$-0{,}1x - 0{,}5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leqslant 3$$

$$y_3 \leqslant 1,\ y_4 \leqslant 1 \text{ i } y_5 \leqslant 2$$

$$x \geqslant 0,\ y_i \geqslant 0, \text{ dla } 1 \leqslant i \leqslant 5$$

Minimum globalne wynosi $(x^*, \mathbf{y}^*) = (0, 6, 0, 1, 1, 0)$ i $f(x^*, \mathbf{y}^*) = -11$.

GENOCOP znalazł optimum we wszystkich przebiegach. Pojedynczy przebieg o 1000 iteracji zajął 23 s czasu CPU.

Test 8

Zadanie złożono z trzech różnych zadań [186] w następujący sposób:

$$\min f(\mathbf{x}) = \begin{cases} f_1 = x_2 + 10^{-5}(x_2 - x_1)^2 - 1{,}0 & \text{gdy } 0 \leqslant x_1 < 2 \\ f_2 = \dfrac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3 & \text{gdy } 2 \leqslant x_1 < 4 \\ f_3 = \dfrac{1}{3}(x_1 - 2)^3 + x_2 - \dfrac{11}{3} & \text{gdy } 4 \leqslant x_1 \leqslant 6 \end{cases}$$

przy ograniczeniach

$$x_1/\sqrt{3} - x_2 \geqslant 0$$

$$-x_1 - 3x_2 + 6 \geqslant 0$$

$$0 \leqslant x_1 \leqslant 6 \text{ i } x_2 \geqslant 0$$

Funkcja f ma trzy globalne minima

$$\mathbf{x}_1^* = (0, 0),\ \mathbf{x}_2^* = (3, \sqrt{3}) \text{ i } \mathbf{x}_3^* = (4, 0)$$

We wszystkich przypadkach $f(\mathbf{x}_i^*) = -1$ $(i = 1, 2, 3)$.

Wykonano trzy odrębne eksperymenty. W eksperymencie k $(k = 1, 2, 3)$ wszystkie funkcje f_i z wyjątkiem f_k zwiększono o 0,5. W rezultacie minimum globalne w pierwszym eksperymencie wynosiło $\mathbf{x}_1^* = (0, 0)$, w drugim $\mathbf{x}_2^* = (3, \sqrt{3})$, a w trzecim $\mathbf{x}_3^* = (4, 0)$.

GENOCOP znalazł globalne optima we wszystkich przebiegach i we wszystkich trzech przypadkach. Pojedynczy przebieg o 500 iteracjach zajął 9 s czasu CPU.

Podsumowanie

Jak wykazano powyżej, GENOCOP działał bardzo dobrze dla wielu zadań testowych z ograniczeniami liniowymi. Jednak nie jest jasne, jak uogólnić GENOCOP, aby działał dla ograniczeń nieliniowych (tzn. zadań w klasie NP). Niektóre zestawy ograniczeń nieliniowych mogą nadal dawać wypukłe przestrzenie przeszukiwań, co jest właściwością istotną dla wielu operatorów (wszy-

stkie mutacje, krzyżowanie arytmetyczne). Jednak nawet w takich przypadkach wyznaczanie zakresów *left*(k) i *right*(k) może być obliczeniowo trudne. Inna możliwość to pokryć przestrzeń rozwiązań rodziną zbiorów wypukłych (niekoniecznie rozłączną) i puszczać GENOCOP na każdej z nich. I tu także pojawiają się problemy obliczeniowe.

Z powyższych względów poszukano wzorów w tradycyjnych metodach optymalizacji. Dwie szczególne metody (opisane w dalszych punktach) wybrano w celu „dopasowania" ich do systemu GENOCOP.

7.2. Optymalizacja nieliniowa: GENOCOP II

W tym punkcie omówimy nowy system hybrydowy GENOCOP II przeznaczony do rozwiązywania zadań programowania nieliniowego. Pomysł tego systemu wzięto z ostatnich osiągnięć w dziedzinie optymalizacji [13] i powiązano z iteracyjnym działaniem systemu GENOCOP (w tym punkcie będziemy się odwoływać do systemu GENOCOP jako GENOCOP I, aby odróżnić go od GENOCOP II).

W metodach korzystających z pojęć analizy matematycznej zakłada się, że funkcja $f(\mathbf{x})$ i wszystkie ograniczenia są dwukrotnie różniczkowalnymi[1)] funkcjami $\mathbf{x}$. Ogólne podejście w większości metod polega na przetransformowaniu nieliniowego zadania programowania na sekwencję rozwiązywalnych podzadań. Nakład pracy potrzebny do rozwiązania podzadań zmienia się znacznie w zależności od metody. Wymagają one obliczania bezpośredniego (lub pośredniego) drugich pochodnych (przetransformowanej) funkcji celu, co w niektórych metodach może być źle uwarunkowane, prowadząc do złego działania algorytmu.

W ostatnich 30 latach na badania związane z nieliniową optymalizacją położono duży nacisk, co doprowadziło do postępu zarówno w teorii, jak i praktyce [114]. Opracowano w tej dziedzinie kilkanaście metod, a wśród nich sekwencyjną metodę kwadratowej funkcji kary [51], [13], metodę rekurencyjnego programowania kwadratowego [40], metody karanych trajektorii [291] i metodę SOLVER [112].

Jedno z tych podejść, sekwencyjna metoda kwadratowej funkcji kary, leży u podstaw pomysłu użytego w systemie GENOCOP II. W metodzie tej zamienia się zadanie $\mathcal{NLP}$ na zadanie $\mathcal{NLP}'$

$$\text{optymalizuj } F(\mathbf{x}, r) = f(\mathbf{x}) + \frac{1}{2r}\,\overline{C}^T\,\overline{C}$$

gdzie $r > 0$, a $\overline{C}$ jest wektorem wszystkich aktywnych ograniczeń $c_1, \ldots, c_l$.

---

1) Istnieją różne metody optymalizacji funkcji zdefiniowanych na ciągłych dziedzinach, w tym także nie zakładające w ogóle różniczkowalności funkcji (przyp. tłum.).

Fiacco i McCormick [253] wykazali, że rozwiązania $\mathcal{NLP}$ i $\mathcal{NLP}'$ są takie same, gdy $r \to 0$. Wydawało się więc, że można by było rozwiązać $\mathcal{NLP}'$, po prostu minimalizując $F(\mathbf{x}, r)$ dla ciągu malejących dodatnich wartości r za pomocą metody Newtona [111]. Ta nadzieja trwała jednak krótko, gdyż minimalizacja $F(\mathbf{x}, r)$ stawała się coraz bardziej źle uwarunkowana numerycznie dla mniejszych wartości r. Murray [291] wykazał, że jest to spowodowane złym uwarunkowaniem hesjanu funkcji $F(\mathbf{x}, r)$, gdy $r \to 0$. Ponieważ nie było nadziei na łatwe przezwyciężenie tego problemu, metoda ta powoli wyszła z użycia. Nieco później Broyden i Attia [52], [51] przedstawili metodę pokonania trudności numerycznych za pomocą prostej kwadratowej funkcji kary. Obliczanie kierunku poszukiwań nie wymaga w niej rozwiązywania żadnego układu równań liniowych i można się spodziewać, że będzie wymagała mniej obliczeń niż inne algorytmy. Metoda ta zawiera także automatyczne obliczanie początkowej wartości parametru r i jego kolejnych wartości [51].

Powyższą metodę wraz z istniejącym systemem GENOCOP I użyto do zbudowania nowego systemu GENOCOP II. Strukturę systemu GENOCOP II przedstawiono na rys. 7.1.

procedure GENOCOP II
begin
  $t \leftarrow 0$
  rozdziel zbiór ograniczeń C na
    $C = L \cup N_e \cup N_i$
  wybierz punkt startowy $\mathbf{x}_s$
  wstaw ograniczenia aktywne do zbioru ograniczeń aktywnych A
    $A \leftarrow N_e \cup V$
  ustaw karę $\tau \leftarrow \tau_0$
  while (**not** warunek zakończenia) **do**
  begin
    $t \leftarrow t + 1$
    wykonaj GENOCOP I dla funkcji
      $$F(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \bar{A}^T \bar{A}$$
      z liniowymi ograniczeniami L
      i punktem startowym $\mathbf{x}_s$
    zapamiętaj najlepszego osobnika $\mathbf{x}^*$:
      $\mathbf{x}_s \leftarrow \mathbf{x}^*$
    uaktualnij A:
      $A \leftarrow A - S \cup V$
    zmniejsz karę τ:
      $\tau \leftarrow g(\tau, t)$
  end
end

Rys. 7.1. Struktura systemu GENOCOP II

Pierwsza faza algorytmu (zanim wejdzie on do pętli *while*) składa się z kilku kroków. Parametrowi t (którego wartość mówi o wykonanej liczbie iteracji, to znaczy liczbie wywołań algorytmu GENOCOP I) nadaje się wartość zero. Zbiór wszystkich ograniczeń C dzieli się na trzy podzbiory: ograniczeń liniowych L, nieliniowych ograniczeń równościowych N_e i nieliniowych ograniczeń nierównościowych N_i. Wybiera się (lub zwraca się o to do użytkownika) punkt startowy $\mathbf{x}_s$ (niekoniecznie dopuszczalny) dla dalszego procesu optymalizacji. Zbiór ograniczeń aktywnych A składa się początkowo z elementów N_e i zbioru $V \subseteq N_i$ naruszonych ograniczeń z N_i. Ograniczenie $c_j \in N_i$ jest naruszone w punkcie $\mathbf{x}$ wtedy i tylko wtedy, gdy $c_j(\mathbf{x}) > \delta$ ($j = p + 1, \ldots, m$), gdzie δ jest parametrem metody. Na koniec ustala się początkowy współczynnik kary w systemie τ (parametr metody) jako równy τ_0.

W głównej pętli algorytmu użyto systemu GENOCOP I do optymalizacji zmodyfikowanej funkcji

$$F(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \overline{A}^T \overline{A}$$

z liniowymi ograniczeniami L. Zauważmy, że początkowa populacja systemu GENOCOP I składa się z *pop_size* identycznych kopii (początkowego punktu w pierwszej iteracji i zapamiętanego najlepszego punktu w dalszych) – kilka operatorów mutacji wprowadza różnorodność do populacji we wczesnych etapach procesu. Kiedy GENOCOP I zbiegnie się, zapamiętuje się jego najlepszego osobnika $\mathbf{x}^*$ i przyjmuje się go następnie jako punkt startowy w następnym pokoleniu. Jednak następną iterację wykonuje się ze zmniejszoną wartością parametru kary ($\tau \leftarrow g(\tau, t)$) i z nowym zbiorem aktywnych ograniczeń A:

$$A \leftarrow A - S \cup V$$

gdzie S i V są odpowiednio podzbiorami N_i ograniczeń spełnionych i naruszonych w $\mathbf{x}^*$. Zauważmy, że zmniejszenie τ powoduje zwiększenie kary.

Działanie algorytmu przedstawimy na następującym przykładzie. Zadanie polega na wyznczeniu

$$\min f(\mathbf{x}) = x_1 x_2^2$$

przy jednym ograniczeniu nieliniowym

$$c_1\colon 2 - x_1^2 - x_2^2 \geqslant 0$$

Znane minimum globalne to $\mathbf{x}^* = (-0{,}816497, -1{,}154701)$ z $f(\mathbf{x}^*) = -1{,}088662$. Dopuszczalnym punktem początkowym jest $\mathbf{x}_0 = (-0{,}99, -0{,}99)$. Po pierwszej iteracji (zbiór A jest pusty) system GENOCOP II zbiegł do $\mathbf{x}_1 = (-1{,}5, -1{,}5)$, $f(\mathbf{x}_1) = -3{,}375$. Punkt $\mathbf{x}_1$ narusza ograniczenie c_1, które staje się aktywne.

Punkt $\mathbf{x}_1$ przyjmuje się jako punkt startowy w drugiej iteracji. Druga iteracja ($\tau = 10^{-1}$, $A = \{c_1\}$) kończy się na $\mathbf{x}_2 = (-0{,}831595, -1{,}179690)$ i $f(\mathbf{x}_2) = -1{,}122678$. Punkt $\mathbf{x}_2$ przyjmuje się za punkt startowy w trzeciej iteracji. Trzecia iteracja ($\tau = 10^{-2}$, $A = \{c_1\}$) kończy się na $\mathbf{x}_3 = (-0{,}815862, -1{,}158801)$ i $f(\mathbf{x}_3) = -1{,}09985$. Ciąg punktów $\mathbf{x}_t$ (gdzie $t = 4, 5, \ldots$ są numerami iteracji algorytmu) zbliża się do optimum.

Do oceny metody GENOCOP II, przedstawienia działania algorytmu i wykazania, że jest on w praktyce skuteczny, wybrano zbiór zadań testowych. Przedstawionych pięć zadań obejmuje funkcje kwadratowe, nieliniowe i nieciągłe z kilkoma ograniczeniami nieliniowymi.

Wszystkie obliczenia przeprowadzono na stacji SUN SPARC 2. We wszystkich eksperymentach przyjęto następujące parametry systemu GENOCOP I:

pop_size = 70, $k = 28$ (liczba rodziców w każdym pokoleniu),
$b = 6$ (współczynnik mutacji nierównomiernej),
$\delta = 0{,}01$ (parametr określający, czy ograniczenie jest aktywne, czy nie).

W większości przypadków początkowy współczynnik kary τ_0 ustalano na 1 (to znaczy $g(\tau, 0) = 1$) oraz przyjmowano $g(\tau, t) = 10^{-1} g(\tau, t-1)$.

Każde zadanie testowe przeliczano za pomocą systemu GENOCOP II dziesięć razy. W większości przypadków liczba pokoleń potrzebna do zbiegnięcia się systemu GENOCOP II wynosiła 1000 (trudniejsze zadania wymagały większej liczby iteracji). Nie podajemy czasów obliczeń tych zadań testowych, gdyż system GENOCOP II nie jest jeszcze w pełni gotowy. Działanie systemu symulowano, wykonując jego zewnętrzną pętlę ręcznie. Kiedy GENOCOP II zbiegł się, najlepszy punkt wprowadzano jako punkt startowy do następnej iteracji, sprawdzano stan naruszania ograniczeń i dostosowywano odpowiednio funkcję oceniającą.

Test 1

Zadanie (wzięte z [186]) wygląda następująco:

$$\min f(\mathbf{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

przy nieliniowych ograniczeniach

$$c_1\colon x_1 + x_2^2 \geqslant 0$$

$$c_2\colon x_1^2 + x_2 \geqslant 0$$

i ograniczeniach na zmienne

$$-0{,}5 \leqslant x_1 \leqslant 0{,}5 \text{ oraz } x_2 \leqslant 1{,}0$$

Znanym globalnym minimum jest $\mathbf{x}^* = (0{,}5, 0{,}25)$ z $f(\mathbf{x}^*) = 0{,}25$. Dopuszczalny punkt startowy to $\mathbf{x}_0 = (0, 0)$.

GENOCOP II znalazł dokładne optimum w jednej iteracji, gdyż żadne ograniczenie nieliniowe nie jest aktywne w optimum.

Test 2

Zadanie (wzięte z [113]) wygląda następująco:

$$\min f(x, y) = -x - y$$

przy nieliniowych ograniczeniach

$$c_1\colon y \leqslant 2x^4 - 8x^3 + 8x^2 + 2$$

$$c_2\colon y \leqslant 4x^2 - 32x^3 + 88x^2 - 96x + 36$$

i ograniczeniach na zmienne

$$0 \leqslant x \leqslant 3 \text{ oraz } 0 \leqslant y \leqslant 4$$

Znane minimum globalne wynosi $\mathbf{x}^* = (2{,}3295,\ 3{,}1783)$ z $f(\mathbf{x}^*) = -5{,}5079$. Dopuszczalny punkt startowy to $\mathbf{x}_0 = (0,\ 0)$. Region rozwiązań dopuszczalnych jest prawie rozłączny.

GENOCOP II zbliżył się bardzo blisko do optimum w czwartej iteracji. Działanie systemu przedstawiono w tabl. 7.1.

Tablica 7.1. Wyniki systemu GENOCOP II dla testu 2

Numer iteracji	Najlepszy punkt	Ograniczenia aktywne
0	(0, 0)	brak
1	(3, 4)	c_2
2	(2,06, 3,98)	c_1, c_2
3	(2,3298, 3,1839)	c_1, c_2
4	(2,3295, 3,1790)	c_1, c_2

W iteracji 0 najlepszym punktem jest punkt startowy.

Test 3

Zadanie (wzięte z [113]) wygląda następująco:

$$\min f(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

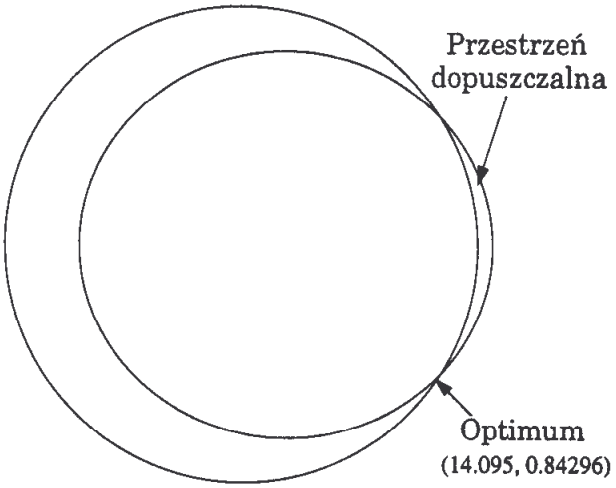
przy nieliniowych ograniczeniach

$$c_1\colon (x_1 - 5)^2 + (x_2 - 5)^2 - 100 \geqslant 0$$

$$c_2\colon -(x_1 - 6)^2 - (x_2 - 5)^2 + 82{,}81 \geqslant 0$$

i ograniczeniach na zmienne

$$13 \leqslant x_1 \leqslant 100 \text{ oraz } 0 \leqslant x_2 \leqslant 100$$



Rys. 7.2. Przestrzeń dopuszczalna w teście 3

Znane minimum globalne wynosi $\mathbf{x}^* = (14{,}095,\ 0{,}84296)$ z $f(\mathbf{x}^*) = -6961{,}81381$ (patrz rys. 7.2). Punkt startowy nie spełniający ograniczeń to $\mathbf{x}_0 = (20{,}1,\ 5{,}84)$.

GENOCOP II zbliżył się bardzo blisko do optimum w dwunastej iteracji. Działanie systemu przedstawiono w tabl. 7.2.

Tablica 7.2. Wyniki systemu GENOCOP II dla tekstu 3

Numer iteracji	Najlepszy punkt	Ograniczenia aktywne
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 iteracji 0 najlepszym punktem jest punkt startowy.

Test 4

Zadanie (wzięte z [39]) wygląda następująco:

$$\min f(x_1, x_2) = 0{,}01x_1^2 + x_2^2$$

przy nieliniowych ograniczeniach

$$c_1\colon x_1x_2 - 25 \geqslant 0$$

$$c_2\colon x_1^2 + x_2^2 - 25 \geqslant 0$$

i ograniczeniach na zmienne

$$2 \leqslant x_1 \leqslant 50 \text{ oraz } 0 \leqslant x_2 \leqslant 50$$

Znane minimum globalne wynosi $\mathbf{x}^* = (\sqrt{250}, \sqrt{2{,}5}) = (15{,}811388, 1{,}581139)$ z $f(\mathbf{x}^*) = 5{,}0$. Punkt startowy (nie spełniający ograniczeń) to $\mathbf{x}_0 = (2, 2)$.

Standardowy schemat chłodzenia (to jest $g(\tau, t) = 10^{-1} g(\tau, t-1)$) nie dawał dobrych wyników, jednak kiedy proces chłodzenia spowolniono (to znaczy $g(\tau, 0) = 5$ i $g(\tau, \mathrm{t}) = 2^{-1}\, g(\tau, t-1)$), wtedy system zbliżył się łatwo do optimum (tabl. 7.3). Powstaje oczywiście pytanie, jak ustalać temperaturę dla danego zadania. Będzie to celem dalszych badań.

Tablica 7.3. Wyniki systemu GENOCOP II dla testu 4

Numer iteracji	Najlepszy punkt	Ograniczenia aktywne
0	(2, 2)	c_1, c_2
1	(3,884181, 3,854748)	c_1
2	(15,805878, 1,581057)	c_1
3	(15,811537, 1,580996)	c_1

W iteracji 0 najlepszym punktem jest punkt startowy.

Test 5

Końcowe zadanie (wzięte z [186]) wygląda następująco:

$$\min f(\mathbf{x}) = (x_1 - 2)^2 + (x_2 - 1)^2$$

przy nieliniowych ograniczeniach

$$c_1\colon\ -x_1^2 + x_2 \geqslant 0$$

i liniowym ograniczeniu

$$x_1 + x_2 \leqslant 2$$

Znane minimum globalne wynosi $\mathbf{x}^* = (1, 1)$ z $f(\mathbf{x}^*) = 1$. Punkt startowy (dopuszczalny) to $\mathbf{x}_0 = (0, 0)$.

GENOCOP II zbliżył się bardzo blisko do optimum w szóstej iteracji. Działanie systemu przedstawiono w tabl. 7.4.

Tablica 7.4. Wyniki systemu GENOCOP II dla testu 5

Numer iteracji	Najlepszy punkt	Ograniczenia aktywne
0	(0, 0)	c_1
1	(1,496072, 0,503928)	c_1
2	(1,020873, 0,979127)	c_1
3	(1,013524, 0,986476)	c_1
4	(1,002243, 0,997757)	c_1
5	(1,000217, 0,999442)	c_1
6	(1,000029, 0,999971)	c_1

W iteracji 0 najlepszym punktem jest punkt startowy.

Podsumowanie

Z tą metodą jest związanych kilka ciekawych uwag. Jak każda metoda oparta na algorytmie genetycznym, nie wymaga ona obliczania bezpośredniego lub pośredniego gradientu lub hesjanu funkcji celu lub ograniczeń. W rezultacie nie wykazuje ona wad związanych ze złym uwarunkowaniem hesjanu, co zwykle występuje w niektórych metodach wykorzystujących analizę matematyczną.

Należy zauważyć, że zamiast systemu GENOCOP I można by było użyć w wewnętrznej pętli GENOCOP II jakiegokolwiek algorytmu genetycznego. W takim przypadku powinno się rozważyć umieszczenie wszystkich ograniczeń (liniowych i nieliniowych) w zbiorze ograniczeń aktywnych A (elementy zbioru L należy rozdzielić między N_e a N_i). Jednak taka metoda jest dużo wolniejsza i mniej efektywna. Bardziej efektywne jest oddzielne traktowanie ograniczeń liniowych (jak to się robi w GENOCOP I).

7.3. Inne metody

W trakcie ostatnich lat zaproponowano kilka metod postępowania z ograniczeniami w algorytmach genetycznych dla zadań optymalizacji numerycznej. Większość z nich korzysta z pojęcia funkcji kary, która wprowadza karę za rozwiązania niedopuszczalne, to znaczy[1)]

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{jeżeli } \mathbf{x} \text{ jest dopuszczalne} \\ f(\mathbf{x}) + penalty(\mathbf{x}), & \text{jeżeli } \mathbf{x} \text{ jest niedopuszczalne} \end{cases}$$

gdzie $penalty(\mathbf{x})$ wynosi zero, gdy ograniczenia nie są naruszone i jest dodatnie w odwrotnym przypadku. W większości metod do ustalenia kary przyjmuje się zbiór funkcji f_j $(1 \leqslant j \leqslant m)$. Funkcja f_j mierzy naruszenie j-tego ograniczenia w następujący sposób:

$$f_j(\mathbf{x}) = \begin{cases} \max\{0,\ g_j(\mathbf{x})\}, & \text{jeżeli } 1 \leqslant j \leqslant q \\ |h_j(\mathbf{x})|, & \text{jeżeli } q+1 \leqslant j \leqslant m \end{cases}$$

Jednak metody te różnią się w wielu istotnych szczegółach związanych z tym, w jaki sposób określono i użyto funkcji kary dla rozwiązań niedopuszczalnych. W następnych podpunktach omówimy je po kolei. Metody są przedstawione zgodnie z malejącą liczbą wymaganych parametrów.

Metoda 1

Metodę tę zaproponowli Homaifar i in. [195]. Zakłada się w niej, że każdemu ograniczeniu przyporządkowano rodzinę przedziałów związanych z odpowiednimi współczynnikami kar. Metoda działa następująco:

[1)] Do końca tego punktu przyjmiemy, że mamy do czynienia z zadaniem minimalizacji.

- dla każdego ograniczenia utwórz kilka (l) poziomów jego naruszenia,
- dla każdego poziomu naruszenia i każdego ograniczenia ustal współczynnik kary R_{ij} ($i = 1,2$, ..., l, $j = 1, 2, ..., m$), wyższe poziomy naruszenia mają większe wartości współczynników,
- rozpocznij od losowej populacji osobników (dopuszczalnej lub niedopuszczalnej),
- przetwarzaj populację, oceniaj osobniki na podstawie następującego wyrażenia:

$$eval(\mathbf{x}) = f(\mathbf{x}) = \sum_{j=1}^{m} R_{ij} f_j^2(\mathbf{x})$$

Wadą tej metody jest liczba parametrów. Dla m ograniczeń metoda wymaga: m parametrów do ustalenia liczby przedziałów dla każdego ograniczenia (w [195] przyjęto te same parametry dla wszystkich ograniczeń $l = 4$), l parametrów dla każdego ograniczenia (to znaczy razem $l \times m$ parametrów), które podają granice przedziałów (poziomów naruszenia), oraz l parametrów dla każdego ograniczenia (to znaczy razem $l \times m$ parametrów) ze współczynnikami kary R_{ij}. Tak więc metoda wymaga w sumie $m(2l + 1)$ parametrów dla m ograniczeń. I tak na przykład dla $m = 5$ ograniczeń i $l = 4$ poziomów naruszenia musimy podać aż 45 parametrów! I oczywiście wyniki zależą od tych parametrów. Prawdopodobnie dla danego zadania istnieje optymalny zestaw parametrów, dla którego system wyznacza rozwiązanie bliskie optymalnemu, może być jednak trudno go znaleźć.

Metoda 2

Druga metoda była zaproponowana przez Joinesa i Houcka [210]. W odróżnieniu od poprzedniej metody, autorzy przyjęli tu kary dynamiczne. Osobniki są oceniane (w iteracji t) za pomocą następującej funkcji:

$$eval(\mathbf{x}) = f(\mathbf{x}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\mathbf{x})$$

gdzie C, α i β są stałymi. Niezłe wartości tych parametrów (podane w [210]) to $C = 0{,}5$, $\alpha = \beta = 2$. Metoda ta wymaga znacznie mniejszej liczby parametrów (niezależnej od liczby ograniczeń) niż pierwsza metoda. Zamiast ustalania kilku poziomów naruszania ograniczeń, karę za rozwiązanie niedopuszczalne zwiększa się zgodnie z mnożnikiem $(C \times t)^{\alpha}$. Pód koniec procesu (dla dużych wartości numeru pokolenia t) mnożnik ten przyjmuje duże wartości.

Metoda 3

Trzecią metodę zaproponowali Schoenauer i Xanthakis [346]. Działa ona następująco:

- rozpocznij od losowej populacji osobników (dopuszczalnych lub niedopuszczalnych),
- ustaw $j = 1$ (j jest licznikiem ograniczeń),
- przetwarzaj populację z $eval(\mathbf{x}) = f_j(\mathbf{x})$ tak długo, aż zadany jej procent (tak zwany *próg uporządkowania* φ) będzie dopuszczalny dla tego ograniczenia,
- przyjmij $j = j + 1$,
- bieżąca populacja jest punktem startowym do następnej fazy przetwarzania, w której $eval(\mathbf{x}) = f_j(\mathbf{x})$; w tej fazie punkty, które nie spełniają jednego z ograniczeń $1, 2, \ldots, j-1$, są wyrzucane z populacji; kryterium stopu jest znowu spełnianie j-tego ograniczenia przez procent populacji zadany progiem uporządkowania φ.
- jeżeli $j < m$, to powtórz dwa ostatnie kroki; jeżeli nie ($j = m$), to optymalizuj funkcję celu, to znaczy $eval(\mathbf{x}) = f(\mathbf{x})$, odrzucając niedopuszczalne osobniki.

Metoda wymaga liniowego uporządkowania wszystkich ograniczeń, które są przetwarzane po kolei. Nie jest jasne, jaki jest wpływ kolejności ustawienia ograniczeń na wynik algorytmu. Nasze doświadczenia wskazują na to, że różne uporządkowanie prowadzi do innych wyników (w sensie całkowitego czasu obliczania i dokładności).

W sumie metoda wymaga 3 parametrów: współczynnika rozdziału σ, progu uporządkowania φ i uporządkowania ograniczeń. Jest ona zupełnie różna od poprzednich dwóch metod i, w ogólności, różni się ona od innych podejść z funkcjami kary, gdyż przetwarza tylko po jednym ograniczeniu na raz. A w ostatnim kroku algorytmu w metodzie optymalizuje się samą funkcję f bez składnika kary.

Metoda 4

Przyjmijmy za czwartą metodę GENOCOP II. Jak przedstawiono wcześniej, jest to jedyna metoda z omówionych tutaj, w której rozróżnia się między ograniczeniami liniowymi a nieliniowymi. Algorytm utrzymuje dopuszczalność względem ograniczeń liniowych za pomocą zestawu analitycznie podanych operatorów, które przetwarzają rozwiązanie dopuszczalne (w sensie ograniczeń liniowych) na inne rozwiązanie dopuszczalne. W każdej iteracji algorytm bierze pod uwagę tylko ograniczenia aktywne, kara za niedopuszczalne rozwiązania wzrasta ze wzrostem temperatury τ.

Metoda ma jeszcze jedną wyjatkową cechę: zaczyna ona od pojedynczego punktu[1]. W rezultacie jest stosunkowo łatwo porównać tę metodę z innymi klasycznymi metodami optymalizacji, których działanie testuje się (dla danego zadania) z pewnego punktu startowego.

---

[1] Ta cecha nie jest podstawowa. Jedynym istotnym wymaganiem jest to, aby następna populacja zawierała najlepsze osobniki z poprzedniej populacji.

Metoda wymaga podania temperatury początkowej i temperatury „zamrażania", odpowiednio τ_0 i τ_f, oraz schematu chłodzenia dla obniżania temperatury τ. Standardowe wartości (podane w [267]) to $\tau_0 = 1$, $\tau_{i+1} = 0{,}1\ \tau_i$ z $\tau_f = 0{,}000001$.

Metoda 5

Piątą metodę opracowali Powell i Skolnick [313]. Jest to klasyczna metoda funkcji kary z jednym ważnym wyjątkiem. Każdy osobnik jest oceniany za pomocą funkcji

$$eval(\mathbf{x}) = f(\mathbf{x}) + r \sum_{j=1}^{m} f_j(x) + \lambda(t, \mathbf{x})$$

gdzie r jest stałą. Jest tu jednak także składnik $\lambda(t, \mathbf{x})$. Jest to jeszcze jedna funkcja zależąca od iteracji, która wpływa na ocenę rozwiązań niedopuszczalnych. Chwyt polega na tym, że w metodzie rozróżnia się osobniki dopuszczalne i niedopuszczalne za pomocą dodatkowego algorytmu heurystycznego (zasugerowanego wcześniej w [332]): dla każdego osobnika dopuszczalnego $\mathbf{x}$ i niedopuszczalnego $\mathbf{y}$ zachodzi $eval(\mathbf{x}) < eval(\mathbf{y})$, to znaczy dowolne rozwiązanie dopuszczalne jest lepsze od każdego niedopuszczalnego. Można to uzyskać na wiele sposobów, jedna z możliwości polega na przyjęciu

$$\lambda(t, \mathbf{x}) = \begin{cases} 0, & \text{jeżeli } \mathbf{x} \in \mathscr{F} \\ \max\{0, \max\limits_{x \in \mathscr{F}} \{f(\mathbf{x})\} + \\ \qquad - \min\limits_{x \notin \mathscr{F}} \{f(\mathbf{x}) + r \sum\limits_{j=1}^{m} f_i(\mathbf{x})\}\}, & \text{w przeciwnym razie} \end{cases}$$

gdzie $\mathscr{F}$ oznacza dopuszczalny obszar przestrzeni przeszukiwań. Inaczej mówiąc, dla osobników niedopuszczalnych zwiększa się karę, ich wartość nie może być lepsza od wartości najgorszego osobnika dopuszczalnego (to znaczy $\max\limits_{x \in \mathscr{F}}\{f(\mathbf{x})\}$)[1].

Metoda 6

W ostatniej metodzie odrzuca się osobniki niedopuszczalne (kara śmierci). Metoda ta była używana w strategiach ewolucyjnych [18], programowaniu ewolucyjnym dostosowanym do optymalizacji numerycznej [117] i symulowanym wyżarzaniu.

---

[1] Powell i Skolnick uzyskali ten sam wynik, rzutując oceny rozwiązań dopuszczalnych na przedział $(-\infty, 1)$, a rozwiązania niedopuszczalne na przedział $(1, \infty)$. Dla selekcji z uporządkowaniem i turniejowej ta różnica w rozwiązaniach nie jest istotna.

7.3.1. Pięć zadań testujących

Przy wyborze poniższych pięciu zadań testujących wzięto pod uwagę: (1) typ funkcji celu, (2) liczbę zmiennych, (3) liczbę ograniczeń, (4) typ ograniczeń, (5) liczbę aktywnych ograniczeń w optimum i (6) stosunek rozmiaru dopuszczalnej przestrzeni przeszukiwań do całej przestrzeni przeszukiwań. Nie twierdzimy, że zaproponowane testy G1-G5 wyczerpują wszystkie możliwości, jednak mogą one stanowić poręczny zestaw wstępnych testów dla innych metod z ograniczeniami.

Test 1
Zadanie [114] polega na 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

$$\begin{aligned}
&2x_1 + 2x_2 + x_{10} + x_{11} \leqslant 10, \quad 2x_1 + 2x_3 + x_{10} + x_{12} \leqslant 10\\
&2x_2 + 2x_3 + x_{11} + x_{12} \leqslant 10, \quad -8x_1 + x_{10} \leqslant 0, \quad -8x_2 + x_{11} \leqslant 0\\
&-8x_3 + x_{12} \leqslant 0, \quad -2x_4 - x_5 + x_{10} \leqslant 0, \quad -2x_6 - x_7 + x_{11} \leqslant 0\\
&-2x_8 - x_9 + x_{12} \leqslant 0, \quad 0 \leqslant x_i \leqslant 1, i = 1, \ldots, 9\\
&0 \leqslant x_i \leqslant 100, i = 10, 11, 12, 0 \leqslant x_3 \leqslant 1
\end{aligned}$$

Zadanie ma 9 ograniczeń liniowych, a funkcja G1 jest kwadratowa z minimum globalnym w

$$\mathbf{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$$

przy czym $G1(\mathbf{x}^*) = -15$. Sześć (z dziesięciu) ograniczeń jest aktywnych w optimum globalnym (wszystkie oprócz następujących trzech: $-8x_1 + x_{10} \leqslant 0$, $-8x_2 + x_{11} \leqslant 0$, $-8x_3 + x_{12} \leqslant 0$).

Test 2
Zadanie [186] polega na minimalizacji funkcji

$$G2(\mathbf{x}) = x_1 + x_2 + x_3$$

gdzie

$$\begin{aligned}
&1 - 0{,}0025(x_4 + x_6) \geqslant 0, \quad 1 - 0{,}0025(x_5 + x_7 - x_4) \geqslant 0\\
&1 - 0{,}01(x_8 - x_5) \geqslant 0, \quad x_1x_6 - 833{,}33252x_4 - 100x_1 + 83333{,}333 \geqslant 0\\
&x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geqslant 0,
\end{aligned}$$

$$x_3x_8 - 1250000 - x_3x_5 + 2500x_5 \geqslant 0$$

$$100 \leqslant x_1 \leqslant 1000, \quad 1000 \leqslant x_i \leqslant 10000, \; i = 2, 3, \; 10 \leqslant x_i \leqslant 1000$$

$$i = 4, \ldots, 8$$

Zadanie ma 3 ograniczenia liniowe i 3 ograniczenia nieliniowe, a funkcja G2 jest liniowa z minimum globalnym w

$$\mathbf{x}^* = (579{,}3167,\ 1359{,}943,\ 5110{,}071,\ 182{,}0174,\ 295{,}5985,\ 217{,}9799,\ 286{,}4162,\ 395{,}5979)$$

przy czym $G2(\mathbf{x}^*) = 7049{,}330923$. Wszystkie sześć ograniczeń jest aktywnych w optimum globalnym.

Test 3

Zadanie [186] polega na minimalizacji funkcji

$$G3(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

gdzie

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geqslant 0$$
$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geqslant 0$$
$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geqslant 0$$
$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geqslant 0$$
$$-10{,}0 \leqslant x_i \leqslant 10{,}0, \quad i = 1, \ldots, 7$$

Zadanie ma 4 ograniczenia nieliniowe, a funkcja G3 jest nieliniowa z minimum globalnym w

$$\mathbf{x}^* = (2{,}330499,\ 1{,}951372,\ -0{,}4775414,\ 4{,}365726,\ -0{,}6244870,\ 1{,}038131,\ 1{,}594227)$$

przy czym $G3(\mathbf{x}^*) = 680{,}6300573$. Dwa (z czterech) ograniczenia są aktywne w optimum globalnym (pierwsze i ostatnie).

Test 4

Zadanie [186] polega na minimalizacji funkcji

$$G4(\mathbf{x}) = e^{x_1x_2x_3x_4x_5}$$

przy ograniczeniach

$$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10, \quad x_2x_3 - 5x_4x_5 = 0$$
$$x_1^3 + x_2^3 = -1, \quad -2{,}3 \leqslant x_i \leqslant 2{,}3 \quad i = 1, 2,$$
$$-3{,}2 \leqslant x_i \leqslant 3{,}2 \quad i = 3, 4, 5$$

Zadanie ma 3 ograniczenia nieliniowe, a funkcja G4 jest nieliniowa z minimum globalnym w

$$\mathbf{x}^* = (-1{,}717143,\ 1{,}595709,\ 1{,}827247,\ -0{,}7636413,\ -0{,}7636450)$$

przy czym $G4(\mathbf{x}^*) = 0{,}0539498478$.

Test 5

Zadanie [186] polega na minimalizacji funkcji

$$\begin{aligned} G5(\mathbf{x}) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + \\ & + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + \\ & + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \end{aligned}$$

gdzie

$$\begin{aligned} & 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geqslant 0 \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \geqslant 0 \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geqslant 0 \\ & -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geqslant 0 \\ & -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geqslant 0 \\ & -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geqslant 0 \\ & -0{,}5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geqslant 0 \\ & 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geqslant 0 \\ & -10{,}0 \leqslant x_i \leqslant 10{,}0,\ i = 1, \ldots, 10 \end{aligned}$$

Zadanie ma 3 ograniczenia liniowe i 5 ograniczeń nieliniowych, a funkcja G5 jest kwadratowa i ma minimum globalne w

$$\begin{aligned} \mathbf{x}^* = {} & (2{,}171996,\ 2{,}363683,\ 8{,}773926,\ 5{,}095984,\ 0{,}9906548,\ 1{,}430574, \\ & 1{,}321644,\ 9{,}828726,\ 8{,}280092,\ 8{,}375927) \end{aligned}$$

przy czym $G5(\mathbf{x}^*) = 24{,}3062091$. Sześć (z ośmiu) ograniczeń jest aktywnych w optimum globalnym (wszystkie oprócz dwóch ostatnich).

Podsumowanie

Podsumowanie wszystkich zadań testowych jest zawarte w tabl.7.5. Dla każdego zadania testowego (ZT) podano w niej liczbę zmiennych n, typ funkcji f, stosunek rozmiaru dopuszczalnej przestrzeni przeszukiwań do całej ρ, liczbę ograniczeń w każdej z kategorii (ograniczenia liniowe LI, ograniczenia nieliniowe NI) i liczbę ograniczeń aktywnych w optimum a.

Tablica 7.5. Podsumowanie pięciu testów

ZT	n	Typ f	ρ	LI	NE	NI	a
1	13	kwadratowa	0,0111%	9	0	0	6
2	8	liniowa	0,0010%	3	0	3	6
3	7	wielomianowa	0,5121%	0	0	4	2
4	5	nieliniowa	0,0000%	0	3	0	3
5	10	kwadratowa	0,0003%	3	0	5	6

Stosunek ρ ustalono eksperymentalnie, generując 1 000 000 losowych punktów w przestrzeni rozwiązań i sprawdzając, czy są one dopuszczalne. LI, NE i NI oznaczają odpowiednio liczby ograniczeń liniowych nierównościowych, ograniczeń nieliniowych równościowych i nierównościowych.

7.3.2. Eksperymenty

We wszystkich eksperymentach przyjęto reprezentację zmiennopozycyjną, nieliniową selekcję z uporządkowaniem, mutację gaussowską, krzyżowanie arytmetyczne i heurystyczne. Prawdopodobieństwa we wszystkich operatorach były równe 0,08, a liczebność populacji wynosiła 70. Dla wszystkich metod system działał do 5 000 pokoleń.

Wyniki zebrano w tabl. 7.6 i 7.7, gdzie przedstawiono (dla wszystkich metod) wyniki najlepsze (kolumna d), średnie (kolumna s) i najgorsze (kolumna z) (z 10 niezależnych obliczeń) i liczbę naruszonych ograniczeń w rozwiązaniu średnim (kolumna o). Ciąg trzech liczb podaje liczbę naruszeń ograniczeń z zakresem naruszenia odpowiednio od 1,0 do 10,0, od 0,1 do 1,0 i od 0,001 do 0,1 (ciąg trzech zer oznacza rozwiązanie dopuszczalne). Jeżeli przynajmniej jedno ograniczenie zostało naruszone o więcej niż 10,0 (jeśli chodzi o funkcję f_j), to takie rozwiązanie było uważane za „niedobre". W niektórych przypadkach było trudno określić „najlepsze" rozwiązanie ze względu na zależność między wartością funkcji celu a liczbą naruszonych ograniczeń. Tablice podają najmniejszą wartość funkcji celu (dla najlepszego rozwiązania), w rezultacie niektóre wartości są „lepsze" niż wartości w minimum globalnym.

Trudno jest zrobić całkowitą analizę wszystkich sześciu metod na podstawie pięciu zadań testowych. Wydaje się jednak, że metoda 1 daje dobre wyniki tylko wtedy, kiedy poziomy naruszeń i współczynniki kary R_{ij} są dopasowane do zadania (na przykład arbitralny wybór 4 poziomów naruszeń z karami 100, 200, 500 i 1000 był dobry dla testów 1 i 3, a nie najlepszy dla innych zadań, gdzie inne wartości tych parametrów były lepsze). Także te poziomy naruszeń i współczynniki kary nie przeszkodziły systemowi w naruszeniu 3 ograniczeń w teście 2, gdyż „nagroda" była zbyt duża, aby się jej oprzeć (wartość funkcji w optimum poza obszarem dopuszczalnym wynosi 2100 dla $\mathbf{x} = (100{,}00, 1000{,}00, 1000{,}00, 128{,}33, 447{,}95, 336{,}07, 527{,}85, 578{,}08)$ przy stosunkowo niedużych karach). We wszystkich testach (z wyjątkiem testu 4) metoda znajdowała rozwiązania, które były tylko stosunkowo niewiele niedopuszczalne, co jest interesującą właściwością tej metody.

Tablica 7.6. Wyniki eksperymentów

ZT	Dokładne optimum		Metoda 1	Metoda 2	Metoda 3
1	– 15,000	*d*	–15,002	–15,000	–15,000
		s	–15,002	–15,000	–15,000
		z	–15,001	–14,999	–14,998
		o	0, 0, 4	0, 0, 0	0, 0, 0
2	7049,331	*d*	2282,723	3117,242	7485,667
		s	2449,798	4213,497	8271,292
		z	2756,679	6055,211	8752,412
		o	0, 3, 0	0, 3, 0	0, 0, 0
3	680,630	*d*	680,771	680,787	680,836
		s	681,262	681,111	681,175
		z	689,660	682,798	685,640
		o	0, 0, 1	0, 0, 0	0, 0, 0
4	0,054	*d*	0,084	0,059	*
		s	0,955	0,812	
		z	1,000	2,542	
		o	0, 0, 0	0, 0, 0	
5	24,306	*d*	24,690	25,486	—
		s	29,258	26,905	
		z	36,060	42,358	
		o	0, 1, 1	0, 0, 0	

Dla każdej metody (1, 2 i 3) podano najlepszy (*d*), średni (*s*) i najgorszy (*z*) wynik (z 10 niezależnych obliczeń) oraz liczbę (*o*) naruszonych ograniczeń w średnim rozwiązaniu. Ciąg trzech liczb oznacza liczbę naruszeń ograniczeń odpowiednio o więcej niż 1,0, więcej niż 0,1 i więcej niż 0,001. Znaki „*" i „-" oznaczają odpowiednio „w tym teście metoda nie była liczona" oraz „wyniki były niedobre".

Metoda 2 dawała lepsze wyniki niż poprzednia dla prawie wszystkich testów: dla testu 1 (gdzie wszystkie rozwiązania były dopuszczalne), testów 2 i 4 (gdzie naruszenia ograniczeń były dużo mniejsze) i testu 3 (gdzie odchylenie standardowe wyników było mniejsze). Jednak wydaje się, że w metodzie 2 nakłada się zbyt duże kary. Często mnożnik $(C \times t)^{\alpha}$ rósł za bardzo, aby mógł być użyteczny. System ma małe szanse, aby uciec z lokalnych optimów. W większości eksperymentów najlepsze osobniki znaleziono we wczesnych pokoleniach. Warto także zauważyć, że metoda ta dała bardzo dobre wyniki dla testów 1 i 5, gdzie funkcja celu była kwadratowa.

Metody 3 nie użyto w teście 4 i nie dała ona dobrych wyników w teście 5, w którym stosunek ρ jest bardzo mały (najmniejszy poza testem 4). Metoda jest wyraźnie bardzo wrażliwa na rozmiar obszaru dopuszczalnego w przestrzeni rozwiązań. Także pewne dodatkowe eksperymenty wykazały, że kolejność rozpatrywanych ograniczeń wpływa wyraźnie na wyniki. Z drugiej strony metoda działała bardzo dobrze dla testu 1 i 3, a dla testu 2 dała niezłe wyniki.

Tablica 7.7. Kontynuacja tablicy 7.6. Wyniki obliczeń dla metod 4, 5, 6

Metoda 4	Metoda 5	Metoda 6	Metoda 6(*f*)
−15,000 −15,000 −15,000 0, 0, 0	−15,000 −15,000 −14,999 0, 0, 0	—	−15,000 −14,999 −13,616 0, 0, 0
7377,976 8206,151 9652,901 0, 0, 0	2101,367 2101,411 2101,551 1, 2, 0	—	7872,948 8559,423 8668,648 0, 0, 0
680,642 680,718 680,955 0, 0, 0	680,805 682,682 685,738 0, 0, 0	680,934 681,771 689,442 0, 0, 0	680,847 681,826 689,417 0, 0, 0
0,054 0,064 0,557 0, 0, 0	0,067 0,091 0,512 0, 0, 0	*	*
18,917 24,418 44,302 0, 1, 0	17,388 22,932 48,866 1, 0, 0	—	25,653 27,116 32,477 0, 0, 0

Wyniki dla metody 6 obejmują także wyniki eksperymentu, w którym początkowa populacja była dopuszczalna (metoda 6(*f*)).

Metoda 4 działała bardzo dobrze dla testów 1, 3 i 4, gdzie uzyskała najlepsze wyniki. Także nieźle działała ona dla testu 2 (gdzie ograniczenia liniowe doprowadziły do błędów metod 1 i 2). Jednak w teście 5 dała ona bardzo słabe wyniki w porównaniu z metodami 1, 2 i 6(f). Wygląda na to, że ograniczenia liniowe w tym zadaniu przeszkadzają systemowi w lepszym zbliżeniu się do optimum. Jest to ciekawy przykład niszczącego efektu ograniczania populacji tylko do obszaru dopuszczalnego (w stosunku do ograniczeń liniowych). Dodatkowe eksperymenty wykazały, że metoda jest bardzo wrażliwa na schemat chłodzenia. Na przykład wyniki dla testu 5 istotnie poprawiono, zmieniając schemat chłodzenia ($\tau_{i+1} = 0{,}01\ \tau_i$).

Metoda 5 miała trudności ze znalezieniem dopuszczalnego rozwiązania dla testu 2. Podobnie jak w metodach 1 i 2 algorytm zapętlał się na rozwiązaniach niedopuszczalnych. We wszystkich innych testach metoda działała stabilnie i nieźle, dając rozwiązania dopuszczalne (testy 1, 3 i 5) lub nieznacznie niedopuszczalne (test 4). Dodatkowe eksperymenty (nie przedstawione w tablicy) dotyczyły obliczeń z dopuszczalnymi populacjami początkowymi. Dla testu 2 wyniki były prawie identyczne jak dla metody 6 (f). Natomiast dla testu 5 wyniki były wspaniałe (najlepsze ze wszystkich metod).

Metoda 6 nie dała (poza testem 3) dobrych wyników. Aby przetestować właściwie tę metodę, trzeba było zaczynać od populacji dopuszczalnej (metoda 6 (f)). Ten inny sposób rozpoczynania obliczeń powoduje, że porównanie staje się jeszcze trudniejsze. Jednak, co ciekawe, metoda dawała zupełnie słabe wyniki. Nie była ona tak stabilna, jak inne metody w najłatwiejszym teście 1 (była to jedyna metoda, która dała rozwiązanie $-13{,}616$, daleko od optimum), a dla testu 2 tylko w jednym obliczeniu dała wartość poniżej 8000.

Żaden parametr (liczba ograniczeń liniowych, nieliniowych, ograniczeń aktywnych, stosunek ρ, typ funkcji, liczba zmiennych) nie okazał się istotny przy ocenie trudności zadania dla metod ewolucyjnych. Na przykład wszystkie metody zbliżały się całkiem blisko do optimum w testach 1 i 5 (gdzie odpowiednio zachodziło $\rho = 0{,}0111\%$ i $\rho = 0{,}0003\%$), gdy tymczasem większość metod miała trudności z testem 2 (gdzie $\rho = 0{,}0010\%$). Metody z różnym powodzeniem radziły sobie też z dwiema funkcjami kwadratowymi (testy 1 i 5) z podobnymi liczbami ograniczeń (odpowiednio 9 i 8) i tą samą liczbą ograniczeń aktywnych w optimum (6). Wygląda na to, że inne właściwości funkcji (kształt funkcji celu w powiązaniu z kształtem obszaru dopuszczalnego) dają dopiero dosyć istotne pojęcie o trudności zadania. Także kilka metod było dosyć wrażliwych na istnienie rozwiązań dopuszczalnych w populacji początkowej.

7.4. Inne możliwości

Jak zauważono wcześniej, badano także heurystyki do ustalania funkcji kary. W [332] sformułowano pewne hipotezy:

- kary, które są funkcjami odległości od obszaru dopuszczalnego działają lepiej niż te, które zależą po prostu od liczby naruszonych ograniczeń,
- dla zadań z małą liczbą ograniczeń i małą liczbą rozwiązań globalnie optymalnych znalezienie rozwiązań z karami, które zależą tylko od liczby naruszonych ograniczeń, jest mało prawdopodobne,
- dobre funkcje kary można utworzyć z dwóch wielkości: *maksymalnego kosztu ukończenia i oczekiwanego kosztu ukończenia*,
- kary powinny być bliskie oczekiwanemu kosztowi zakończenia, ale nie powinny zbyt często spadać poniżej. Im dokładniejsza kara, tym lepiej będą znalezione rozwiązania. Kiedy kara jest zbyt często mniejsza od kosztu ukończenia, wtedy poszukiwanie może nie przynieść skutku.

a w [358]

- algorytm genetyczny ze zmiennym współczynnikiem kary jest lepszy od algorytmu z ustalonym współczynnikiem kary.

przy czym zmienność współczynnika kary ustalono za pomocą reguł heurystycznych.

Tę ostatnią hipotezę rozpatrywali następnie Smith i Tate [360]. W swojej pracy eksperymentowali oni z dynamicznymi karami, w których kara zależy od liczby naruszonych ograniczeń, najlepszej znalezionej wartości funkcji celu dla rozwiązań dopuszczalnych i najlepszej znalezionej w ogóle wartości funkcji celu.

Metodę dostosowywania kar opracowali także Bean i Hadj-Alouane [27], [174]. Przyjęli oni funkcje kary, w której jeden składnik zależy od procesu przeszukiwania. Każdy osobnik jest oceniany za pomocą wyrażenia

$$eval\,\bar{X} = f(\bar{X}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\bar{X})$$

gdzie $\lambda(t)$ jest poprawiana w każdym pokoleniu t w następujący sposób:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & \text{jeżeli } \bar{B}(i) \in \mathscr{F} \text{ dla wszystkich } t-k+1 \leqslant i \leqslant t \\ \beta_2 \cdot \lambda(t), & \text{jeżeli } \bar{B}(i) \notin \mathscr{F} \text{ dla wszystkich } t-k+1 \leqslant i \leqslant t \\ \lambda(t), & \text{w przeciwnym razie} \end{cases}$$

a $\bar{B}(i)$ oznacza osobnika o najlepszej wartości funkcji *eval* w pokoleniu i, β_1, $\beta_2 > 1$ oraz $\beta_1 \neq \beta_2$ (aby uniknąć zapętlenia). Inaczej mówiąc, w metodzie: (1) zmniejsza się składnik kary $\lambda(t+1)$ w pokoleniu $t+1$, jeżeli wszystkie najlepsze osobniki w ostatnich k pokoleniach były dopuszczalne oraz (2) zwiększa się karę, jeżeli wszystkie najlepsze osobniki w ostatnich k pokoleniach nie były dopuszczalne. Jeżeli w ostatnich k pokoleniach wśród najlepszych osobników były osobniki dopuszczalne i niedopuszczalne, to $\lambda(t+1)$ pozostaje bez zmiany.

Powyższe podejście zastosowano w zadaniach programowania całkowitoliczbowego. Zadanie rozważane w [174] jest następujące:

$$\min \mathbf{cx}$$

przy ograniczeniach

$$A\mathbf{x} - \mathbf{b} \geqslant 0 \tag{1}$$

$$\sum_{j=1}^{n_i} x_{ij} = 1, \qquad \text{dla } i = 1, \ldots, m \tag{2}$$

$$x_{ij} \in \{0, 1\} \tag{3}$$

Dla każdego zbioru, $i = 1, \ldots, m$, równania (2) i (3) wymuszają, aby dokładnie jedna zmienna w $\{x_{ij}\}_{j=1}^{n_i}$ była równa jeden. Macierz A ma wymiar $k \times n$ ($n = \Sigma\, i = 1^m n_i$), a $\mathbf{b}$ jest ustalonym wektorem k-wymiarowym.

Jak podano w [174], najbardziej skutecznymi metodami rozwiązywania tego zadania są metody podziału i ograniczeń, które używają bądź programowania liniowego, bądź relaksacji Lagrange'a, bądź też ich odmian. W relak-

sacji Lagrange'a pomija się niektóre ograniczenia przez przyjęcie ważonej sumy kar za naruszenie ograniczeń. „Właściwe" wagi mogą prowadzić do bardzo dobrych oszacowań, a nawet optymalnego rozwiązania początkowego zadania. W typowej metodzie relaksacji Lagrange'a zamienia się zadanie początkowe (ograniczenia (1)) na następujące zadanie:

$$\min \mathbf{cx} - \lambda(A\mathbf{x} - \mathbf{b})$$

przy ograniczeniach

$$E\mathbf{x} = e_m,\ x_{ij} \in \{0, 1\}$$

(ograniczenia $E\mathbf{x} = e_m$ są ograniczeniami z wieloma wyborami, przy czym e_m jest wektorem jedynek).

W zaproponowanym podejściu zamieniono zadanie początkowe przez

$$\min \mathbf{cx} + p_\lambda(\mathbf{x})$$

przy ograniczeniach

$$E\mathbf{x} = e_m, \qquad x_{ij} \in \{0, 1\}$$

gdzie $p_\lambda(\mathbf{x}) = \sum_{i=1}^{k} \lambda_t[\min\{0, A_i\mathbf{x} - b_i\}]^2$. Jest to funkcja nieliniowa i do optymalizacji tego wyrażenia użyto algorytmu genetycznego.

W tej metodzie są ciekawe pomysły. Po pierwsze wyniki eksperymentów wskazują, że zaczynanie od dużych wartości λ nie prowadzi do efektywnych algorytmów. Wobec tego proponowany algorytm dopasowuje wektor λ podczas obliczeń: przyjmuje się ciąg rosnących λ. Eksperymenty [174] wskazały, że szybkość zmian tego współczynika jest bardzo ważna (podobne wnioski wyciągnięto na podstawie wyników obliczeń za pomocą systemu GENOCOP II): powolne zmiany mogą poprawić jakość rozwiązań kosztem szybkości poprawy, a szybkie zmiany mogą powodować nieefektywną ewolucję genetyczną. Dodatkowo w zaproponowanym algorytmie genetycznym stosuje się metodę losowych kluczy, w której rozwiązanie jest reprezentowane jako wektor liczb losowych. Ich uporządkowanie dekoduje rozwiązanie (zauważmy podobieństwo tej metody i zastosowania strategii ewolucyjnych do zadania komiwojażera, patrz rozdz. 8, gdzie omawia się strategie ewolucyjne, i rozdz. 10, gdzie omawia się zadanie komiwojażera). Rozwiązanie jest reprezentowane przez łańcuch o długości równej liczbie zmiennych w zbiorze wielokrotnych wyborów.

Warto także omówić inne metody postępowania z ograniczeniami. Jedna z nich (GENOCOP III, omawiany w następnym punkcie) jest oparta na algorytmach naprawy: rozwiązanie niedopuszczalne jest „wpychane" do obszaru dopuszczalnego i ocenia się jego naprawioną wersję. Można także zbudować algorytmy hybrydowe, w których stosuje się pewne deterministyczne procedury optymalizacji. Krótki przegląd i opis jednej szczegółowej metody można znaleźć w [294].

Dodatkowa możliwość może obejmować użycie wartości funkcji celu f i kar f_j jako elementów wektora i zastosowanie metod wielokryterialnych do minimalizacji wszystkich składników wektora. Inaczej mówiąc, funkcja celu f i miary naruszenia ograniczeń f_j (dla m ograniczeń) tworzą $(m + 1)$-wymiarowy wektor $\mathbf{v}$:

$$\mathbf{v} = (f, f_1, \ldots, f_m)$$

Stosując metody optymalizacji wielokryterialnej, możemy spróbować zminimalizować jego składniki. Idealne rozwiązanie x powinno dawać $f_j(x) = 0$ dla $1 \leqslant i \leqslant m$ i $f(x) \leqslant f(y)$ dla wszystkich dopuszczalnych y (zadanie minimalizacji). Udane rozwiązanie programowe tego sformułowania przedstawili ostatnio Surry i in. [376].

Jeszcze inne podejście zaproponowali ostatnio Le Riche i in. [239]. Autorzy utworzyli (rozdzielony) algorytm genetyczny, w którym używa się dwóch wartości parametrów kar zamiast jednego (dla każdego ograniczenia). Dążą oni do uzyskania równowagi między dużymi a średnimi karami przez utrzymywanie podpopulacji osobników. Populacja jest podzielona na dwie współpracujące grupy, a osobniki w każdej grupie są oceniane przy użyciu jednego lub dwóch parametrów kar.

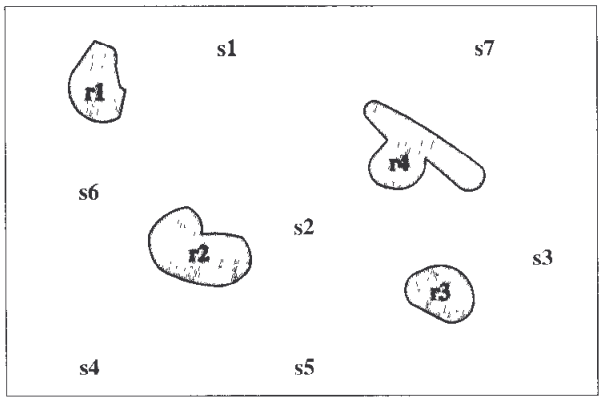
Interesujące podejście opisał także ostatnio Paredis [308]. Metoda (opisana w kontekście zadań spełniających ograniczenia) bazuje na dwóch współewolujących modelach, w których populacje potencjalnych rozwiązań współewolują z populacją ograniczeń. Bardziej dopasowane rozwiązania spełniają więcej ograniczeń, gdy tymczasem bardziej dopasowane ograniczenia są naruszone przez więcej rozwiązań. Oznacza to, że osobniki z populacji rozwiązań są rozważane na całej przestrzeni przeszukiwań i że nie rozróżnia się osobników dopuszczalnych od niedopuszczalnych. Jednak ocenę osobnika określa się na zasadzie miar naruszania ograniczeń f_j, lepsze f_j (na przykład aktywne ograniczenia) bardziej wpływają na ocenę rozwiązania. Byłoby ciekawe zaadoptowanie tego podejścia do zadań optymalizacji numerycznej z ograniczeniami i porównanie go z innymi metodami. Ale największą trudnością do rozwiązania przy takiej adaptacji wydaje się być to samo, co we wszystkich innych metodach: jak zbalansować nacisk na dopuszczalność rozwiązania z naciskiem na minimalizację funkcji celu.

Badania algorytmów kulturowych [327], [328], [330], [331] wywodzą się z obserwacji, że kultura może być innym sposobem dziedziczenia. Nie jest jednak jasne, jakie są odpowiednie struktury i jednostki do reprezentacji adaptacji i przekazywania informacji kulturowej. Nie jest też jasne, jak opisać wzajemne oddziaływanie na siebie ewolucji naturalnej i kultury. Reynolds opracował kilka modeli do badania właściwości algorytmów kulturowych. W tych modelach do ograniczenia kombinacji postaw, jakie mogą przyjmować osobniki, użyto przestrzeni wiary. Zmiany w przestrzeni wiary reprezentują zmiany makroewolucyjne, a zmiany w populacji osobników zmiany mikroewolucyjne. Obie zmiany są łagodzone przez połączenie komunikacyjne.

Ogólna intuicja związana z przestrzeniami wiary to zachowanie wierzeń związanych z zachowaniami „akceptowanymi" na poziomie postaw (i oczywiście odcięcie wierzeń nieakceptowanych). Wierzenia akceptowane służą jako ograniczenia, które kierują populacją postaw. Wygląda na to, że algorytmy kulturowe mogą być bardzo ciekawym narzędziem dla zadań optymalizacji numerycznej, gdzie ograniczenia wpływają bezpośrednio na przeszukiwanie (w rezultacie przeszukiwanie w przestrzeniach ograniczonych może być bardziej wydajne niż w nieograniczonych!). Całkiem niedawno Reynolds i in. [329] badali możliwość zastosowania algorytmów kulturowych w optymalizacji numerycznej z ograniczeniami. Pierwsze eksperymenty wskazują na duże potencjalne zalety tego podejścia.

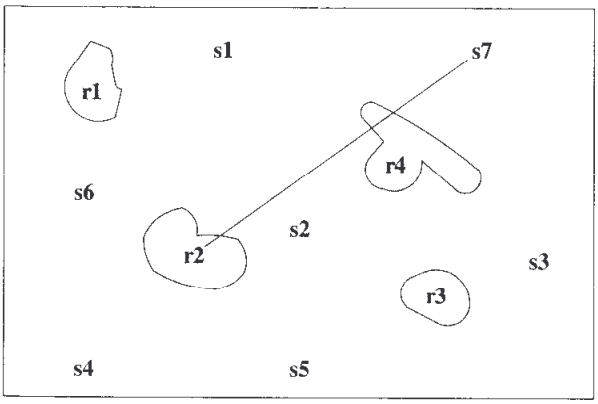
7.5. GENOCOP III

Ta metoda obejmuje początkowy system GENOCOP (opisany w p. 7.1), jednak rozszerzony do dwóch oddzielnych populacji, przy czym rozwój jednej z nich wpływa na ocenę osobników w drugiej populacji. Pierwsza populacja P_s składa się z tak zwanych punktów przeszukujących, które spełniają ograniczenia liniowe zadania (jak w początkowym systemie GENOCOP). Dopuszczalność (w sensie ograniczeń liniowych) tych punktów utrzymuje się, jak poprzednio, za pomocą specjalizowanych operatorów. Druga populacja P_r składa się z tak zwanych punktów odniesienia. Te punkty są całkowicie dopuszczalne, to znaczy spełniają wszystkie ograniczenia. (Jeżeli GENOCOP III ma trudności z wyborem takich punktów odniesienia na początku, pyta o to użytkownika.



Rys. 7.3. Populacja $P_s = \{\bar{S}_1, \bar{S}_2, \bar{S}_3, \bar{S}_4, \bar{S}_5\}$ i populacja $P_r = \{\bar{R}_1, \bar{R}_2, \bar{R}_3, \bar{R}_4\}$

W przypadkach, gdy stosunek ρ rozmiarów obszaru dopuszczalnego i całej przestrzeni rozwiązań jest bardzo mały, może się zdarzyć, że początkowy zbiór punktów odniesienia składa się z wielu kopii jednego punktu dopuszczalnego). Te dwie populacje zilustrowano na rys. 7.3.



Rys. 7.4. Ocena niedopuszczalnego punktu $\bar{S}$

Dopuszczalne punkty odniesienia $\bar{R}$ są oceniane bezpośrednio przez funkcję oceniającą (to znaczy $eval(\bar{R}) = f(\bar{R})$). Natomiast punkty niedopuszczalne są „naprawiane" do oceny, a proces naprawy przebiega następująco. Załóżmy, że punkt przeszukujący $\bar{S}$ nie jest w pełni dopuszczalny. W takim przypadku system wybiera jeden z punktów odniesienia, powiedzmy $\bar{R}$ (lepsze punkty odniesienia mają większe szanse na wybranie, używa się metody wyboru opartej na nieliniowym uporządkowaniu), i tworzy losowe punkty $\bar{Z}$ z odcinka łączącego $\bar{S}$ i $\bar{Z}$ za pomocą losowej generacji liczb a z zakresu [0, 1]: $\bar{Z} = a\bar{S} + (1 - a)\bar{R}$. Proces naprawy jest przedstawiony na rys. 7.4.

Kiedy znajdzie się punkt dopuszczalny $\bar{Z}$, wtedy $eval(\bar{S}) = eval(\bar{Z}) = f(\bar{Z})$[1]. Dodatkowo, jeżeli $f(\bar{Z})$ jest lepszy niż $f(\bar{R})$, to punkt $\bar{Z}$ zastępuje punkt $\bar{R}$ jako nowy punkt odniesienia. $\bar{Z}$ zastępuje także $\bar{S}$ z pewnym prawdopodobieństwem wymiany p_r.

W systemie GENOCOP III nie występuje wiele wad obecnych w innych systemach. Wprowadza się w nim tylko kilka dodatkowych parametrów (liczność populacji punktów odniesienia, prawdopodobieństwo wymiany). Uzyskuje się w nim zawsze rozwiązanie dopuszczalne. Obszar dopuszczalny jest przeszukiwany przez rzutowanie punktów przeszukujących. Otoczenie lepszych punktów odniesienia jest badane częściej. Niektóre punkty odniesienia są przesuwane do populacji punktów przeszukujących, gdzie są one przetwarzane przez specjalizowane operatory (które zachowują ograniczenia liniowe).

Początkową wersję GENOCOP-u III (dostępną przez ftp.uncc.edu, katalog coe/evol, plik genocopIII.tar.Z) przetestowano na kilku zadaniach (opisanych w p. 7.3.1) G1 – G5. Wyniki z GENOCOP-u III dla testu G1 są identyczne jak dla oryginalnego GENOCOP-u, gdyż przy braku ograniczeń nieliniowych nie ma potrzeby użycia populacji punktów odniesienia. Z pozostałych czterech testów wykonano eksperymenty na trzech (G2, G3 i G5). Zadanie G4 zawiera nieliniowe równości NE, a obecna wersja GENOCOP-u III jeszcze takich przypadków nie obejmuje.

[1] Oczywiście dla różnych pokoleń ten sam punkt $\bar{S}$ może być różnie oceniony ze względu na losową naturę procesu naprawy.

GENOCOP III przeliczał 5000 iteracji (jak inne systemy omawiane w p. 7.3). We wszystkich operatorach przyjęto prawdopodobieństwa równe 0,08, a liczności obu populacji wynosiły 70.

Wyniki były bardzo dobre. Na przykład dla zadania G2 najlepszy wynik wynosił 7286,650. Jest on dużo lepszy niż najlepszy wynik najlepszego systemu z omawianych w p. 7.3.1 (dla tego zadania był to GENOCOP II, który uzyskal 7377,976). Podobnie działał on dla dwu pozostałych zadań: G3 (wartość 7377,976) i G5 (wartość 25,883). Dodatkowa ciekawa obserwacja dotyczy stabilności systemu. GENOCOP III dawał bardzo małe odchylenie standardowe wyników. Na przykład w zadaniu G3 wszystkie wyniki znajdowały się między 680,640 a 680,889. Natomiast inne systemy dawały bardzo różne wyniki (między 680,642 a 689,660, patrz [266]).

Oczywiście wszystkie punkty końcowe $\overline{X}$ były dopuszczalne, co nie było pewne przy innych systemach (na przykład GENOCOP II dał wartość 18,917 w zadaniu G5, systemy oparte na metodach Homaifara, Lai i Qi oraz Powella i Skolnicka dały odpowiednio wyniki 2282,723 i 2101,367 w zadaniu G2).

Dodatkowy interesujący test pojawił się ostatnio. Zadanie [220] polega na maksymalizacji funkcji

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

gdzie

$$\prod_{i=1}^{n} x_i > 0{,}75, \quad \sum_{i=1}^{n} x_i < 7{,}5n \quad \text{i} \quad 0 < x_i < 10 \quad \text{dla} \quad 1 \leqslant i \leqslant n$$

Zadanie ma dwa ograniczenia nieliniowe, funkcja f jest nieliniowa, a jego maksimum globalne jest nieznane.

Keane [220] napisał:

> Używam obecnie do jego rozwiązywania równoległego algorytmu genetycznego z kodowaniem dwunastobitowym, krzyżowaniem, inwersją, mutacją, tworzeniem niszy i zmodyfikowaną funkcją kary Fiacca-McCormicka dla ograniczeń. Dla $n = 20$ dostałem wartości około 0,76 po 20 000 ocen.

Przeprowadzono obliczenia za pomocą GENOCOP-u III dla $n = 20$ i $n = 50$. W pierwszym przypadku najlepsze znalezione rozwiązanie (w 10 000 pokoleniach) wynosiło

$$\begin{aligned}\mathbf{x} = (&3{,}16311359,\ 3{,}13150430,\ 3{,}09515858,\ 3{,}06016588,\ 3{,}03103566,\\ &2{,}99158549,\ 2{,}95802593,\ 2{,}92285895,\ 0{,}48684388,\ 0{,}47732279,\\ &0{,}48044473,\ 0{,}48790911,\ 0{,}48450437,\ 0{,}44807032,\ 0{,}46877760,\\ &0{,}45648506,\ 0{,}44762608,\ 0{,}44913986,\ 0{,}44390863,\ 0{,}45149332)\end{aligned}$$

gdzie $f(\mathbf{x}) = 0{,}80351067$. W drugim przypadku ($n = 50$) najlepsze znalezione rozwiązanie (w 10 000 pokoleń) wynosiło

$$\begin{aligned}\mathbf{x} = (&6{,}28006029,\ 3{,}16155291,\ 3{,}15453815,\ 3{,}14085174,\ 3{,}12882447,\\ &3{,}11211085,\ 3{,}10170507,\ 3{,}08703685,\ 3{,}07571769,\ 3{,}06122732,\\ &3{,}05010581,\ 3{,}03667951,\ 3{,}02333045,\ 3{,}00721049,\ 2{,}99492717,\\ &2{,}97988462,\ 2{,}96637058,\ 2{,}95589066,\ 2{,}94427204,\ 2{,}92796040,\\ &0{,}40970641,\ 2{,}90670991,\ 0{,}46131119,\ 0{,}48193336,\ 0{,}46776962,\\ &0{,}43887550,\ 0{,}45181099,\ 0{,}44652876,\ 0{,}43348753,\ 0{,}44577143,\\ &0{,}42379948,\ 0{,}45858049,\ 0{,}42931050,\ 0{,}42928645,\ 0{,}42943302,\\ &0{,}43294361,\ 0{,}42663351,\ 0{,}43437257,\ 0{,}42542559,\ 0{,}41594154,\\ &0{,}43248957,\ 0{,}39134723,\ 0{,}42628688,\ 0{,}42774364,\ 0{,}41886297,\\ &0{,}42107263,\ 0{,}41215360,\ 0{,}41809589,\ 0{,}41626775,\ 0{,}42316407)\end{aligned}$$

gdzie $f(\mathbf{x}) = 0{,}83319378$[1].

Jak widać, GENOCOP III jest obiecującym narzędziem dla zadań optymalizacji nieliniowej z ograniczeniami. Jednak zawiera jeszcze wiele miejsc, które wymagają lepszego przyjrzenia się i eksperymentowania. Obejmują one badanie istotności stosunku ρ. Zauważmy, że jest możliwe przedstawienie pewnych ograniczeń liniowych jako nieliniowe. Taka zmiana wprowadziłaby zmianę przestrzeni punktów odniesienia na mniejszą i przestrzeni liniowo dopuszczalnych punktów odniesienia na większą. Nie wiadomo jednak, jak te zmiany wpłynęłyby na działanie systemu.

Inna grupa eksperymentów mogłaby być związana z jednym parametrem: prawdopodobieństwem wymiany p_r. We wszystkich eksperymentach przedstawionych powyżej $p_r = 0{,}15$.

Planujemy także (w bliskiej przyszłości) rozszerzyć GENOCOP III, tak aby obejmował on nieliniowe ograniczenia równościowe. Będzie to wymagało dodatkowego parametru (ε) określającego dokładność obliczeń. Wszystkie równości nieliniowe $h_j(\overline{X}) = 0$ (dla $j = q + 1, \ldots, m$) będą wtedy zamienione przez parę nierówności

$$-\varepsilon \leqslant h_j(\overline{X}) \leqslant \varepsilon$$

Ta nowa wersja GENOCOP-u III powinna umożliwić bezpośrednie obliczenie zadania G4.

---

[1] Nie jest to optimum globalne. Biltchev [43] podał wartość funkcji celu 0,8348.

8

Strategie ewolucyjne i inne metody

To nie było najlepsze, że mieliśmy myśleć jednakowo.
To różnice w opiniach powodują,
że odbywają się wyścigi konne.

Mark Twain, *Pudd'nhead Wilson*

Strategie ewolucyjne są algorytmami, które naśladują zasady ewolucji naturalnej w metodach rozwiązywania zadań optymalizacji parametrycznej [18], [348]. Opracowano je w Niemczech w latach sześćdziesiątych. Jak napisano w [348]:

> W 1963 r. dwaj studenci Politechniki Berlińskiej spotkali się i wkrótce zaczęli wspólnie wykonywać eksperymenty w tunelu wiatrów Instytutu Inżynierii Przepływów. W czasie poszukiwania optymalnych kształtów ciał w przepływach, co było wówczas robione za pomocą pracochłonnych eksperymentów intuicyjnych, powstał pomysł postępowania systematycznego. Jednak próby ze współrzędnymi i prostymi strategiami gradientowymi nie dawały wyników. Wtedy jeden ze studentów, Ingo Rechenberg, obecnie profesor bioniki i inżynierii ewolucyjnej, wpadł na pomysł wprowadzania losowych zmian w parametrach określających kształt, zgodnie z zasadami mutacji naturalnej. W ten sposób narodziła się strategia ewolucyjna.

(Drugim studentem był Hans-Paul Schwefel, obecnie profesor informatyki i kierownik katedry analizy systemowej).

Wczesne strategie ewolucyjne można uważać za programy ewolucyjne, w których używa się reprezentacji zmiennopozycyjnej, z mutacją jako jedynym operatorem rekombinacyjnym. Zastosowano je w wielu zadaniach optymalizacji z ciągłymi parametrami. Dopiero ostatnio rozszerzono je na zadania dyskretne [18], [178].

W tym rozdziale opiszemy wczesne strategie ewolucyjne używające populacji dwuelementowych i operatora mutacji oraz różne strategie ewolucyjne z populacjami wieloelementowymi (p. 8.1). W punkcie 8.2 porównamy strategie ewolucyjne z algorytmami genetycznymi, a w p. 8.4 przedstawimy inne strategie zaproponowane ostatnio przez różnych badaczy.

8.1. Ewolucja strategii ewolucyjnych

Najwcześniejsze strategie ewolucyjne używały populacji składających się tylko z jednego osobnika. W procesie ewolucji brał też udział tylko jeden operator genetyczny – mutacja. Jednak ciekawy pomysł (czego nie ma w algorytmach genetycznych) polegał na reprezentacji osobnika jako pary wektorów zmiennopozycyjnych, to znaczy $\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma})$. Pierwszy wektor $\mathbf{x}$ reprezentuje punkt w przestrzeni przeszukiwań, a drugi $\boldsymbol{\sigma}$ jest wektorem odchyleń standardowych. Mutację wykonuje się, zamieniając $\mathbf{x}$ na

$$\mathbf{x}^{t+1} = \mathbf{x}^t + N(0, \boldsymbol{\sigma})$$

gdzie $N(0, \boldsymbol{\sigma})$ jest wektorem niezależnych liczb gaussowskich o zerowej średniej i odchyleniach standardowych $\boldsymbol{\sigma}$. (Jest to zgodne z obserwacjami biologicznymi, że mniejsze zmiany zdarzają się częściej niż większe). Potomek (zmutowany osobnik) jest przyjmowany za nowego członka populacji (zastępuje swojego rodzica) wtedy i tylko wtedy, gdy ma lepsze dopasowanie i są spełnione wszystkie ograniczenia (jeżeli występują). Na przykład, jeżeli f jest maksymalizowaną funkcją celu bez ograniczeń, to potomek $(\mathbf{x}^{t+1}, \boldsymbol{\sigma})$ zastępuje swojego rodzica $(\mathbf{x}^t, \boldsymbol{\sigma})$ wtedy i tylko wtedy, gdy $f(\mathbf{x}^{t+1}) > f(\mathbf{x}^t)$. W innym przypadku potomek jest odrzucany i populacja nie zmienia się.

Przedstawmy pojedynczy krok takiej strategii ewolucyjnej na przykładzie zadania maksymalizacji, które rozważaliśmy w rozdz. 2 (dla prostego algorytmu genetycznego):

$$f(x_1, x_2) = 21{,}5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

przy czym $-3{,}0 \leqslant x_1 \leqslant 12{,}1$ oraz $4{,}1 \leqslant x_2 \leqslant 5{,}8$.

Jak opisano wcześniej, populacja składa się z jednego osobnika $(\mathbf{x}, \boldsymbol{\sigma})$, przy czym $\mathbf{x} = (x_1, x_2)$ jest punktem w przestrzeni przeszukiwań ($-3{,}0 \leqslant x_1 \leqslant 12{,}1$ i $4{,}1 \leqslant x_2 \leqslant 5{,}8$), a $\boldsymbol{\sigma} = (\sigma_1, \sigma_2)$ zawiera dwa odchylenia standardowe, które będą użyte w operatorze mutacji. Załóżmy, że w pewnej chwili t jedynym elementem populacji jest następujący osobnik:

$$(\mathbf{x}^t, \boldsymbol{\sigma}) = ((5{,}3,\ 4{,}9),\ (1{,}0,\ 1{,}0))$$

i że mutacja wprowadziła następującą zmianę:

$$x_1^{t+1} = x_1^t + N(0,\ 1{,}0) = 5{,}3 + 0{,}4 = 5{,}7$$
$$x_2^{t+1} = x_2^t + N(0{,}1,\ 0) = 4{,}9 - 0{,}3 = 4{,}6$$

Ponieważ

$$f(\mathbf{x}^t) = f(5{,}3,\ 4{,}9) = 18{,}383705 < 24{,}849532 = f(5{,}7,\ 4{,}6) = f(\mathbf{x}^{t+1})$$

i zarówno x_1^{t+1}, jak i x_2^{t+1} mieszczą się w swoich zakresach ograniczoności, więc potomek zastąpi rodzica w tej jednoelementowej populacji.

Mimo że populacja składa się z jednego osobnika, który podlega mutacji, przedstawiona powyżej strategia ewolucyjna jest nazywana *strategią ewolucyjną dwuelementową*. Powodem jest to, że potomek rywalizuje ze swoim rodzicem i na etapie rywalizacji są (czasowo) dwa osobniki w populacji.

Wektor odchyleń standardowych $\boldsymbol{\sigma}$ nie zmienia się w procesie ewolucyjnym. Jeżeli składniki tego wektora są jednakowe, to znaczy $\boldsymbol{\sigma} = (\sigma, ..., \sigma)$, a zadanie optymalizacji jest regularne[1)], to można udowodnić twierdzenie o zbieżności [18].

Twierdzenie 1 (twierdzenie o zbieżności)
Dla $\boldsymbol{\sigma} > 0$ i regularnego zadania optymalizacji z $f_{opt} > -\infty$ (dla minimalizacji) lub $f_{opt} < \infty$ (dla maksymalizacji) zachodzi

$$p\{\lim_{t\to\infty} f(\mathbf{x}^t) = f_{opt}\} = 1$$

Twierdzenie o zbieżności mówi, że dla dostatecznie długiego przeszukiwania znajdzie się globalne optimum z prawdopodobieństwem 1. Nie daje to jednak żadnego klucza do szybkości tej zbieżności (ilorazu odległości przebytej w kierunku optimum i liczby pokoleń, przy której uzyskano tę odległość). Dla zoptymalizowania szybkości zbieżności Rechenberg zaproponował *„regułę 1/5 sukcesu"*:

> Stosunek φ udanych mutacji do wszystkich mutacji powinien wynosić 1/5. Jeżeli φ jest większe od 1/5, to zwiększ wariancję operatora mutacji, w odwrotnym przypadku zmniejsz ją.

Reguła 1/5 sukcesu pojawiła się jako wynik procesu optymalizacji szybkości zbieżności dwóch funkcji (tak zwanego modelu korytarzowego i sferycznego; szczegóły można znaleźć w [18]). Reguła ta jest stosowana co k pokoleń (k jest jeszcze jednym parametrem tej metody). Reguła 1/5 sukcesu wygląda więc następująco:

$$\boldsymbol{\sigma}^{t+1} = \begin{cases} c_d \cdot \boldsymbol{\sigma}^t, & \text{dla } \varphi(k) < 1/5 \\ c_i \cdot \boldsymbol{\sigma}^t, & \text{dla } \varphi(k) > 1/5 \\ \boldsymbol{\sigma}^t, & \text{dla } p_s(k) = 1/5 \end{cases}$$

gdzie $\varphi(k)$ jest współczynnikiem sukcesów operatora mutacji w poprzednich k pokoleniach, a $c_i > 1$, $c_d < 1$ regulują szybkość wzrostu lub malenia wariancji mutacji. Schwefel w swoich eksperymentach [348] użył następujących wartości: $c_d = 0{,}82$, $c_i = 1{,}22 = 1/0{,}82$.

[1)] Zadanie optymalizacji jest regularne, jeżeli funkcja celu f jest ciągła, dziedzina funkcji jest zbiorem domkniętym, dla wszystkich $\varepsilon > 0$ zbiór wszystkich wewnętrznych punktów dziedziny, w których funkcja różni się od wartości optymalnej mniej niż o ε, jest niepusty, a dla wszystkich $\mathbf{x}_0$ zbiór punktów, w których funkcja przyjmuje wartości mniejsze lub równe $f(\mathbf{x}_0)$, jest zbiorem domkniętym (dla zadania minimalizacji – dla zadania maksymalizacji zależność jest odwrotna).

Intuicyjną przyczyną wprowadzenia reguły 1/5 sukcesu jest wzrost wydajności szukania. Jeżeli przynosi on sukces, niech się odbywa „większymi" krokami, jeżeli nie, niech kroki będą krótsze. Jednak takie szukanie może doprowadzić do przedwczesnej zbieżności dla pewnych klas funkcji. Było to przyczyną ulepszenia metody – zwiększenia liczebności populacji.

Wieloelementowa strategia ewolucyjna różni się od poprzedniej strategii dwuelementowej liczebnością populacji (*pop_size* > 1). Dodatkowymi cechami wieloelementowej strategii ewolucyjnej są:

- wszystkie osobniki w populacji mają te same prawdopodobieństwa łączenia się w pary,
- istnieje możliwość wprowadzenia operatora rekombinacji (w dziedzinie algorytmów genetycznych nazywanego *jednorodnym krzyżowaniem*), gdzie dwoje (losowo wybranych) rodziców

$$(\mathbf{x}^1, \boldsymbol{\sigma}^1) = ((x_1^1, \ldots, x_n^1), (\sigma_1^1, \ldots, \sigma_n^1)) \quad \text{i}$$

$$(\mathbf{x}^2, \boldsymbol{\sigma}^2) = ((x_1^2, \ldots, x_n^2), (\sigma_1^2, \ldots, \sigma_n^2))$$

tworzą potomka

$$(\mathbf{x}, \boldsymbol{\sigma}) = ((x_1^{q_1}, \ldots, x_n^{q_n}), (\sigma_1^{q_1}, \ldots, \sigma_n^{q_n}))$$

gdzie $q_i = 1$ lub $q_i = 2$ z jednakowym prawdopodobieństwem dla wszystkich $i = 1, \ldots, n$.

Operator mutacji i odchylenie standardowe $\boldsymbol{\sigma}$ pozostają bez zmian.

Występuje jednak nadal podobieństwo między strategiami ewolucyjnymi dwuelementowymi i wieloelementowymi. W obu z nich tworzy się jednego potomka. W strategii dwuelementowej potomek rywalizuje ze swoim rodzicem. W strategii wieloelementowej usuwa się najsłabszego osobnika (spośród *pop_size* + 1 osobników, to znaczy początkowych *pop_size* osobników i potomka). Wygodna notacja, która wyraża także dalsze udoskonalenia strategii ewolucyjnych, ma postać

(1 + 1)-SE dla strategii ewolucyjnej dwuelementowej i

(μ + 1)-SE dla strategii ewolucyjnej wieloelementowej

gdzie $\mu = pop_size$.

Strategie ewolucyjne wieloelementowe uległy później dalszemu rozwojowi [348], aby zakończyć jako strategie

($\mu + \lambda$)-SE i (μ, λ)-SE

Główną ideą tych strategii była samoadaptacja parametrów sterujących (jak wariancja mutacji) zamiast zmiany ich wartości w sposób deterministyczny.

Strategia $(\mu + \lambda)$-SE jest naturalnym rozszerzeniem strategii ewolucyjnej wieloelementowej $(\mu + 1)$-SE, w której μ osobników tworzy λ potomków. Nowa (chwilowa) populacja $(\mu + \lambda)$ osobników jest następnie zmniejszana w procesie selekcji ponownie do μ osobników. Natomiast w strategii (μ, λ)-SE μ osobników tworzy λ potomków $(\lambda > \mu)$, a w procesie selekcji wybiera się nową populację μ osobników tylko ze zbioru λ potomków. W ten sposób życie każdego osobnika ogranicza się do jednego pokolenia. Pozwala to strategii (μ, λ)-SE działać lepiej w zadaniach, w których optimum zmienia się w czasie, lub w zadaniach, w których funkcja celu jest zaszumiona.

Operatory zastosowane w strategiach $(\mu + \lambda)$-SE i (μ, λ)-SE mają dwa poziomy uczenia się. Ich parametr sterujący $\boldsymbol{\sigma}$ nie jest już stały, ani nie zmienia się zgodnie z pewnym algorytmem deterministycznym (jak reguła 1/5 sukcesu), ale jest włączony w strukturę osobników i podlega procesowi ewolucji. Aby utworzyć potomka, system wykonuje kilka kroków:

- Wybierz dwa osobniki

  $$(\mathbf{x}^1, \boldsymbol{\sigma}^1) = ((x_1^1, ..., x_n^1), (\sigma_1^1, ..., \sigma_n^1)) \quad \text{i}$$

  $$(\mathbf{x}^2, \boldsymbol{\sigma}^2) = ((x_1^2, ..., x_n^2), (\sigma_1^2, ..., \sigma_n^2))$$

  i użyj operatora rekombinacji (krzyżowania). Są dwa typy krzyżowań:

  – dyskretne, w których nowym potomkiem jest

  $$(\mathbf{x}, \boldsymbol{\sigma}) = ((x_1^{q_1}, ..., x_n^{q_n}), (\sigma_1^{q_1}, ..., \sigma_n^{q_n}))$$

  gdzie $q_i = 1$ lub $q_i = 2$ (więc każdy składnik pochodzi z pierwszego lub z drugiego wybranego rodzica),

  – pośrednie, w którym nowego potomka tworzy się według wzoru

  $$(\mathbf{x}, \boldsymbol{\sigma}) = (((x_1^1 + x_1^2)/2, ..., (x_n^1 + x_n^2)/2), ((\sigma_1^1 + \sigma_1^2)/2, ..., (\sigma_n^1 + \sigma_n^2)/2))$$

  Każdy z tych dwóch operatorów można także zastosować w sposób globalny, kiedy wybiera się nową parę rodziców dla każdego składnika wektora potomka.
- Zastosuj do otrzymanego potomka $(\mathbf{x}, \boldsymbol{\sigma})$ mutację. Otrzymanym nowym potomkiem będzie $(\mathbf{x}', \boldsymbol{\sigma}')$, przy czym

  $$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0, \Delta\sigma)} \quad \text{i}$$

  $$\mathbf{x}' = \mathbf{x} + N(0, \boldsymbol{\sigma}')$$

  gdzie $\Delta\boldsymbol{\sigma}$ jest parametrem metody.

Aby poprawić szybkość zbieżności strategii ewolucyjnych Schwefel [348] wprowadził dodatkowy parametr sterujący $\boldsymbol{\theta}$. Jest on związany ze skorelowaniem mutacji. Dla strategii ewolucyjnych dyskutowanych dotychczas (z wartościami σ_i przyporządkowanymi każdemu x_i) preferowany kierunek poszuki-

wań mógł być wybrany tylko wzdłuż osi układu współrzędnych. Teraz każdy osobnik populacji jest reprezentowany przez

$$(\mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$$

Operatory rekombinacji są podobne do omawianych we wcześniejszych punktach, a mutacja tworzy potomka $(\mathbf{x}', \boldsymbol{\sigma}', \boldsymbol{\theta}')$ z $(\mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\theta})$ w następujący sposób:

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0,\, \Delta\sigma)}$$

$$\boldsymbol{\theta}' = \boldsymbol{\theta} + N(0, \Delta\boldsymbol{\theta}) \qquad \text{i}$$

$$\mathbf{x}' = \mathbf{x} + C(0, \boldsymbol{\sigma}', \boldsymbol{\theta}')$$

gdzie $\Delta\boldsymbol{\theta}$ jest dodatkowym parametrem metody, a $C(0, \boldsymbol{\sigma}', \boldsymbol{\theta}')$ oznacza wektor niezależnych losowych liczb gaussowskich z zerową średnią i odpowiednią gęstością prawdopodobieństwa (szczegóły można znaleźć w [348] lub [18]).

Strategie ewolucyjne działają dobrze w zadaniach numerycznych, gdyż były one (przynajmniej początkowo) przeznaczone dla zadań optymalizacji funkcji (rzeczywistych). Są przykłady programów ewolucyjnych, które używają odpowiednich struktur danych (wektorów zmiennopozycyjnych rozszerzonych o parametry strategii sterującej) i operatorów „genetycznych" związanych z zadaniem.

Ciekawe może być porównanie algorytmów genetycznych i strategii ewolucyjnych, ich różnic i podobieństw, ich mocnych i słabych stron. Omówimy te sprawy w następnym punkcie.

8.2. Porównanie strategii ewolucyjnych i operatorów genetycznych

Główna różnica między strategiami ewolucyjnymi a algorytmami genetycznymi leży w dziedzinie przeszukiwań. Strategie ewolucyjne rozwinięto jako metody optymalizacji numerycznej. Stosują one specjalne procedury poszukiwania wzrostu funkcji z samoadaptującymi się krokami $\boldsymbol{\sigma}$ i kątami odchylenia $\boldsymbol{\theta}$. Dopiero niedawno zastosowano strategie ewolucyjne w zadaniach optymalizacji dyskretnej [178]. Natomiast algorytmy genetyczne były pomyślane jako metody adaptacyjnego przeszukiwania (ogólnego przeznaczenia), które w sposób wykładniczy zwiększają liczbę prób dla schematów ocenianych powyżej średniej. Algorytmy genetyczne były używane w różnych zastosowaniach, a optymalizacja parametryczna była tylko jedną z ich dziedzin zastosowania.

Z tego powodu nie jest uczciwe porównywanie czasów i dokładności działania strategii ewolucyjnych oraz algorytmów genetycznych, używając do ich porównania optymalizacji funkcji numerycznej. Jednak zarówno strategie ewo-

lucyjne, jak i algorytmy genetyczne są przykładami programów ewolucyjnych, więc ogólna dyskusja ich podobieństw i różnic jest całkiem naturalna.

Główne podobieństwo strategii ewolucyjnych i algorytmów genetycznych polega na tym, że w obu znajduje się populacja potencjalnych rozwiązań, a zasada selekcji opiera się na przeżyciu bardziej dopasowanych osobników. Jak wspomniano w wielu miejscach, strategie ewolucyjne używają wektorów zmiennopozycyjnych, gdy tymczasem klasyczne operatory algorytmów genetycznych używają wektorów binarnych.

Druga różnica między algorytmami genetycznymi a strategiami ewolucyjnymi jest ukryta w samym procesie selekcji. W pojedynczym pokoleniu strategii ewolucyjnych μ rodziców tworzy pośrednią populację składającą się z λ potomków utworzonych za pomocą operatorów rekombinacji i mutacji (dla strategii (μ, λ)-SE) i ewentualnie μ rodziców (dla strategii $(\mu + \lambda)$-SE). Następnie w procesie selekcji redukuje się liczebność tej populacji pośredniej z powrotem do μ osobników przez wyrzucenie z populacji osobników najgorzej dopasowanych. Ta populacja μ osobników tworzy nowe pokolenie. W jednym pokoleniu w algorytmie genetycznym w procedurze selekcji wybiera się *pop_size* osobników z populacji o liczebności *pop_size*. Osobniki są wybierane z powtórzeniami, to znaczy mocne osobniki mają dużą szansę na wielokrotny wybór do nowej populacji. Jednocześnie szansę wyboru ma nawet najsłabszy osobnik.

W strategiach ewolucyjnych procedura selekcji jest deterministyczna. Wybiera się w niej najlepszych μ z $\mu + \lambda$ (dla strategii $(\mu + \lambda)$-SE) lub λ (dla strategii (μ, λ)-SE) osobników (bez powtórzeń). Natomiast w algorytmach genetycznych selekcja jest losowa. Przy wyborze *pop_size* z *pop_size* osobników (z powtórzeniami) szansa wyboru jest proporcjonalna do dopasowania osobnika. Oczywiście w niektórych algorytmach genetycznych stosuje się selekcję na podstawie uporządkowania, jednak nawet w nich mocne osobniki mogą być wybrane kilka razy. Inaczej mówiąc, selekcja w strategiach ewolucyjnych jest statyczna, wygaszająca, a (dla strategii (μ, λ)-SE) pokoleniowa, podczas gdy w algorytmach genetycznych jest ona dynamiczna, zachowująca i w locie (patrz rozdz. 4).

Kolejność selekcji i rekombinacji jest trzecią różnicą między algorytmami genetycznymi a strategiami ewolucyjnymi. W strategiach ewolucyjnych proces selekcji następuje po operatorze rekombinacji, gdy tymczasem w algorytmach genetycznych kolejność tych kroków jest odwrotna. W strategiach ewolucyjnych potomek jest wynikiem krzyżowania dwojga rodziców i następującej po tym mutacji. Kiedy pośrednia populacja $\mu + \lambda$ (lub λ) osobników jest gotowa, wtedy procedura selekcji redukuje z powrotem jej liczebność do μ osobników. W algorytmach genetycznych najpierw wybiera się populację pośrednią. Następnie stosuje się operatory genetyczne (krzyżowanie i mutację) do niektórych osobników (wybranych zgodnie z prawdopodobieństwem krzyżowania) i niektórych genów (wybranych zgodnie z prawdopodobieństwem mutacji).

Następną różnicą między strategiami ewolucyjnymi a algorytmami genetycznymi jest to, że parametry reprodukcji w algorytmach genetycznych (praw-

dopodobieństwo krzyżowania, prawdopodobieństwo mutacji) są stałe w procesie ewolucji, gdy tymczasem w strategiach ewolucyjnych zmienia się je ($\boldsymbol{\sigma}$ i $\boldsymbol{\theta}$) w czasie. Podlegają one mutacji i krzyżowaniu razem z wektorem rozwiązań $\mathbf{x}$, gdyż za osobnika przyjmuje się trójkę ($\mathbf{x}$, $\boldsymbol{\sigma}$, $\boldsymbol{\theta}$). Jest to dosyć istotne – samoadaptacja parametrów sterujących w strategiach ewolucyjnych jest odpowiedzialna za dokładne lokalne dostrojenie się systemu.

W strategiach ewolucyjnych i algorytmach genetycznych inaczej postępuje się także z ograniczeniami. W strategiach ewolucyjnych przyjmuje się, że zbiór $q \geqslant 0$ nierówności

$$g_1(\mathbf{x}) \geqslant 0, \ldots, g_q(\mathbf{x}) \geqslant 0$$

jest częścią zadania optymalizacji. Jeżeli w którejś iteracji potomek nie spełnia tych ograniczeń, to taki potomek jest dyskwalifikowany, to znaczy nie umieszcza się go w nowej populacji. Jeżeli stosunek pojawiania się takich nieprawidłowych potomków jest zbyt duży, poprawia się parametry sterujące, na przykład zmniejszając składniki wektora $\boldsymbol{\sigma}$. Główna strategia w algorytmach genetycznych (omawiana już w rozdz. 7) postępowania z ograniczeniami polega na nałożeniu kary na osobniki, które je naruszają. Powodem jest to, że dla mocno ograniczonych zadań po prostu nie można pomijać nieprawidłowych potomków (algorytmy genetyczne nie poprawiają swoich parametrów sterujących), gdyż wtedy algorytmy stałyby w miejscu przez większość czasu. Jednocześnie często różne dekodery lub algorytmy naprawy są zbyt kosztowne, aby ich używać (wysiłek przy skonstruowaniu dobrego algorytmu naprawy jest porównywalny z wysiłkiem rozwiązania zadania). Metody funkcji kar mają wiele wad, jedną z nich jest zależność od rozwiązywanego zadania.

Z powyższej dyskusji wynika, że strategie ewolucyjne i algorytmy genetyczne różnią się wieloma szczegółami. Jednak przyglądając się bliżej rozwojowi strategii ewolucyjnych i algorytmów genetycznych w ciągu ostatnich dwudziestu lat, trzeba przyznać, że różnica między nimi staje się coraz mniejsza.

Pomówmy więc znowu o pewnych sprawach związanych ze strategiami ewolucyjnymi i algorytmami genetycznymi, tym razem z perspektywy historii.

Całkiem wcześnie pojawiły się oznaki, że algorytmy genetyczne mają pewne trudności w wykonaniu przeszukiwania lokalnego w zastosowaniach numerycznych (patrz rozdz. 6). Wiele osób eksperymentowało z różnymi reprezentacjami (kody Graya, liczby zmiennopozycyjne) i różnymi operatorami, aby poprawić działanie systemów z algorytmami genetycznymi. Obecnie nie jest to już podstawowa różnica między algorytmami genetycznymi a strategiami ewolucyjnymi. Większość programów z algorytmami genetycznymi dla zadań optymalizacji parametrycznej używa reprezentacji zmiennopozycyjnych [78], dostosowując do tego odpowiednio operatory (patrz rozdz. 4). Wygląda na to, że dziedzina algorytmów genetycznych zapożyczyła ten pomysł reprezentacji wektorów od strategii ewolucyjnych.

Wyniki naszych esperymentów z programami ewolucyjnymi doprowadziły do ciekawego spostrzeżenia: ani krzyżowanie, ani mutacja nie są same skutecz-

ne w procesie ewolucyjnym. Oba operatory (a raczej obie rodziny operatorów) są niezbędne w zapewnieniu dobrego działania systemu. Operatory krzyżowania są bardzo istotne w przeszukiwaniu obiecujących obszarów przestrzeni przeszukiwań i są odpowiedzialne za wczesną (ale nie przedwczesną) zbieżność. W wielu systemach, w szczególności tych, które pracują z bogatszymi strukturami danych (część III książki), zmniejszenie częstotliwości krzyżowania pogarsza ich działanie. Jednocześnie prawdopodobieństwo zastosowania mutacji jest w nich całkiem duże: system GENETIC-2 (rozdz. 9) używa dużego stosunku mutacji 0,2.

Podobny wniosek wyciągnięto w dziedzinie strategii ewolucyjnych. W rezultacie wprowadzono do strategii ewolucyjnych operator krzyżowania. Zauważmy, że wczesne strategie ewolucyjne bazowały tylko na operatorze mutacji, a operator krzyżowania pojawił się znacznie później [348]. Wygląda więc na to, że wynik rywalizacji między algorytmami genetycznymi a strategiami ewolucyjnymi jest remisowy: dziedzina strategii ewolucyjnych zapożyczyła pomysł operatora krzyżowania od algorytmów genetycznych.

Jest więcej interesujących spraw dotyczących powiązań między strategiami ewolucyjnymi a algorytmami genetycznymi. Ostatnio pewne operatory krzyżowania pojawiły się w algorytmach genetycznych i strategiach ewolucyjnych jednocześnie [269], [270], [352]. Dwa wektory $\mathbf{x}_1$ i $\mathbf{x}_2$ mogą utworzyć dwóch potomków $\mathbf{y}_1$ i $\mathbf{y}_2$, którzy są liniowymi kombinacjami ich rodziców, to znaczy

$$\mathbf{y}_1 = a \cdot \mathbf{x}_1 + (1 - a) \cdot \mathbf{x}_2 \qquad \text{oraz}$$

$$\mathbf{y}_2 = (1 - a) \cdot \mathbf{x}_1 + a \cdot \mathbf{x}_2$$

Takie krzyżowanie nazywa się:

- w algorytmach genetycznych *gwarantowanym średnim krzyżowaniem* [77] (kiedy $a = 1/2$) lub *krzyżowaniem arytmetycznym* [269], [270],
- w strategiach ewolucyjnych *natychmiastowym krzyżowaniem* [352].

Samoadaptacja parametrów sterujących w strategiach ewolucyjnych ma swój odpowiednik w badaniach nad algorytmami genetycznymi. W ogólności pomysł adaptowania algorytmu genetycznego podczas jego działania pojawił się jakiś czas temu: w systemie ARGOT [354] adaptuje się reprezentację osobników (patrz p. 8.4). Problem adaptowania parametrów sterujących zauważono także od pewnego czasu w algorytmach genetycznych [169], [77], [115]. Było jasne od początku, że znalezienie dobrego zestawu parametrów algorytmów genetycznych dla konkretnego zadania nie jest łatwą sprawą. Zaproponowano kilka rozwiązań. Jedno z nich [169] polega na zastosowaniu nadzorującego algorytmu genetycznego do optymalizacji parametrów „właściwego" algorytmu genetycznego dla pewnej klasy zadań. Rozważane parametry to liczebność populacji, współczynnik krzyżowania, współczynnik mutacji, luka pokoleniowa (procent populacji do wymiany w każdym pokoleniu), okno skalujące i strategia selekcji (zwykła czy elitarna). W innych rozwiązaniach [77] adaptuje

się prawdopodobieństwa operatorów genetycznych. Idea polega na tym, że prawdopodobieństwo zastosowania operatora zmienia się proporcjonalnie do zaobserwowanego działania osobników utworzonych przez ten operator. Intuicyjnie polega to na tym, że operatory wykonujące „dobrą robotę" powinny wykonywać ją częściej. W [115] autor badał cztery strategie przyporządkowania operatorowi mutacji prawdopodobieństwa: (1) stałe prawdopodobieństwo, (2) wykładniczo malejące, (3) wykładniczo rosnące, (4) kombinację (2) i (3).

Jeżeli sobie przypomnimy mutację nierównomierną (opisaną w rozdz. 6), to też zauważymy, że operator ten zmienia działanie w trakcie procesu ewolucji.

Porównajmy w skrócie program ewolucyjny działający na zasadzie genetycznej GENOCOP (rozdz. 7) ze strategią ewolucyjną. Oba systemy utrzymują populację potencjalnych rozwiązań i używają pewnych programów selekcji do rozróżnienia między „dobrymi" a „złymi" osobnikami. Oba używają reprezentacji zmiennopozycyjnej. Dają one dużą dokładność obliczeń (strategia ewolucyjna przez adaptację parametrów sterujących, a GENOCOP przez nierównomierną mutację). Oba systemy zawierają sposoby polepszenia działania przy wystąpieniu ograniczeń: GENOCOP przy wystąpieniu ograniczeń liniowych, a strategia ewolucyjna przez uwzględnianie ograniczeń nierównościowych. W obu systemach można łatwo uwzględnić sposoby postępowania z ograniczeniami użyte w drugim systemie. Operatory są podobne. W jednym jest krzyżowanie pośrednie, w drugim krzyżowanie arytmetyczne. Czy są więc one rzeczywiście różne?

Ciekawe porównanie strategii ewolucyjnych i algorytmów genetycznych pod względem strategii ewolucyjnych zaprezentowano w [187].

Kilka lat temu [117] metody programowania ewolucyjnego (patrz p. 13.1, gdzie opisano początkowe metody programowania ewolucyjnego) były uogólnione tak, aby mogły rozwiązywać zadania optymalizacji numerycznej. Są one bardzo podobne do strategii ewolucyjnych. Używają reprezentacji zmiennopozycyjnej i mutacji jako podstawowego operatora. Podstawowe różnice między strategiami ewolucyjnymi a metodami programowania ewolucyjnego można podsumować następująco [19]:

- w programowaniu ewolucyjnym nie używa się operatorów rekombinacji,
- w programowaniu ewolucyjnym stosuje się selekcję probabilistyczną (selekcję turniejową), gdy tymczasem w strategiach ewolucyjnych wybiera się μ najlepszych osobników do następnego pokolenia,
- w programowaniu ewolucyjnym wartości dopasowania uzyskuje się z funkcji celu przez ich skalowanie z ewentualnym wymuszeniem losowych zmian,
- odchylenie standardowe mutacji dla każdego osobnika oblicza się jako pierwiastek kwadratowy z liniowego przekształcenia jego własnej wartości dopasowania.

Więcej informacji o programowaniu ewolucyjnym zastosowanym do optymalizacji numerycznej i porównanie eksperymentalne metod strategii ewolucyjnych i programowania ewolucyjnego można znaleźć w [19], [117] i [121].

8.3. Optymalizacja funkcji wielomodalnych i wielokryterialnych

W większości rozdziałów książki przestawialiśmy metody wyznaczania pojedynczego globalnego optimum funkcji. Jednak w wielu przypadkach albo funkcja może mieć kilka optimów, które chcemy wyznaczyć (optymalizacja wielomodalna), albo jest więcej niż jedna funkcja celu do optymalizacji (optymalizacja wielokryterialna). Oczywiście do rozwiązywania takich rodzajów zadań są potrzebne nowe metody. Omówimy je po kolei.

8.3.1. Optymalizacja wielomodalna

W wielu zastosowaniach może być ważne wyznaczenie wszystkich optimów danej funkcji[1]. Dla takiej optymalizacji wielomodalnej zaproponowano kilka metod ewolucyjnych. W pierwszej metodzie stosuje się iteracje. Po prostu powtarza się w niej obliczenia algorytmu kilka razy. Jak stwierdzono w [30], jeżeli wszystkie optima mają równe prawdopodobieństwo znalezienia, to liczba niezależnych obliczeń powinna wynosić

$$p \sum_{i=1}^{p} \frac{1}{i} \approx p(\gamma + \log p)$$

gdzie p jest liczbą optimów, a $\gamma \approx 0{,}577$ jest stałą Eulera. Niestety, w większości rzeczywistych zastosowań znalezienie optimów nie jest jednakowo prawdopodobne, więc liczba niezależnych obliczeń powinna być znacznie większa. Można także użyć programowania równoległego, gdzie kilka populacji rozwija się (w niezależny sposób, to znaczy bez komunikacji) w tym samym czasie.

Goldberg i Richardson [162] opisali metodę z wykorzystaniem podziału. Metoda umożliwia utworzenie stabilnych podpopulacji (gatunków) o różnych łańcuchach. W ten sposób algorytm bada wiele optimów równolegle (artykuł zawiera też świetny przegląd innych metod, w których zastosowano podobne pomysły z niszami i tworzeniem gatunków). Funkcja podziału określa zmniejszenie dopasowania osobnika w stosunku do sąsiada oddalonego o pewną odległość *dist* [162][2]. Funkcję podziału *sh* definiuje się jako funkcję odległości z następującymi właściwościami:

[1] Przez „wszystkie" optima rozumiemy optima nas interesujące, to znaczy optima powyżej pewnego progu.

[2] Odległość *dist* można zdefiniować na poziomie genotypu lub fentypu, to znaczy na łańcuchach lub ich interpretacjach.

- $0 \leqslant sh(dist) \leqslant 1$, dla dowolnej odległości *dist*,
- $sh(0) = 1$,
- $\lim\limits_{dist \to \infty} = 0$.

Jest wiele funkcji podziału, które spełniają powyższe warunki. Jedna z możliwości, jaką zasugerowano w [162] to

$$sh(dist) = \begin{cases} 1 - \left(\dfrac{dist}{\sigma_{sh}}\right)^{\alpha}, & \text{jeśli } dist < \sigma_{sh} \\ 0, & \text{w przeciwnym przypadku} \end{cases}$$

gdzie σ_{sh} i α są stałymi (omówienie znaczenia tych stałych można znaleźć w [162]).

Nowe (podzielone) dopasowanie osobnika **x** jest określone przez

$$eval'(\mathbf{x}) = eval(\mathbf{x})/m(\mathbf{x})$$

gdzie $m(\mathbf{x})$ jest liczbą osobników w niszy zajmowanej przez szczególnego osobnika **x**:

$$m(\mathbf{x}) = \sum_{y} sh(dist(\mathbf{x}, \mathbf{y}))$$

W powyższym wzorze suma po wszystkich **y** w populacji dotyczy też samego łańcucha **x**. W rezultacie, jeżeli łańcuch jest sam we własnej niszy, jego wartość dopasowania nie zmniejsza się ($m(\mathbf{x}) = 1$). W innym przypadku funkcja dopasowania zmniejsza się proporcjonalnie do liczby i bliskości sąsiednich punktów.

Oznacza to, że kiedy sąsiaduje ze sobą wielu osobników, to wpływają one na liczebność niszy, w ten sposób obniżając własne wartości dopasowań. W rezultacie w tej metodzie ogranicza się niekontrolowany wzrost poszczególnych gatunków w populacji [154].

Ostatnio Beasley, Bull i Martin [30] opisali nową metodę (nazwaną *metodą sekwencyjnych nisz*) do optymalizacji funkcji wielomodalnych. Jest ona pozbawiona kilku niedogodności metody podziału (na przykład wydłużenia czasu liczenia z powodu obliczania podzielonych dopasowań, wielkości populacji, która powinna być proporcjonalna do liczby optimów). Zaproponowany algorytm także używa funkcji odległości *dist* i funkcji oceny *eval*. Jest on oparty na następującym pomyśle: jeżeli już znaleziono optimum, to można zmodyfikować funkcje oceny tak, aby wyeliminować możliwość ponownego znalezienie tego samego (już wyznaczonego) rozwiązania, gdyż nie ma potrzeby ponownego wykrywania tego samego optimum. W pewnym sensie w następnych obliczeniach takiego algorytmu genetycznego korzysta się z wiedzy uzyskanej

z poprzednich obliczeń (w odróżnieniu od prostej metody iteracyjnej, w której każde obliczenie zaczyna się od losowo wygenerowanej populacji). Podstawowe kroki tego algorytmu wyglądają następująco (na podstawie [30]):

1. Rozpocznij: podstaw za zmodyfikowaną funkcję dopasowania początkową funkcję dopasowania.
2. Wykonaj przebieg algorytmu genetycznego ze zmodyfikowaną funkcją dopasowania, zapamiętując najlepszego osobnika znalezionego w tym przebiegu.
3. Popraw zmodyfikowaną funkcję dopasowania, aby spowodować obniżenie[1)] w obszarze wokół najlepszego osobnika, tworząc w ten sposób nową funkcję dopasowania.
4. Jeżeli jest potrzebne dopasowanie osobnika według początkowej funkcji (na przykład do sprawdzenia, czy przekracza on próg rozwiązania), to podaj to jako rozwiązanie.
5. Jeżeli nie znaleziono wszystkich rozwiązań, to wróć do kroku 2.

Szczegółowe omówienie algorytmu i wyników obliczeń można znaleźć w [30].

Ostatnio Spears [365] zaproponował jeszcze inną metodę. W opisanym algorytmie użyto podziału i ograniczonego doboru. Jednak zrezygnowano z odległości metrycznej i przyjęto koncepcję etykiet. Każdy osobnik w populacji ma etykietę (w obliczeniach opisanych w [365] etykieta jest łańcuchem o n bitach, w rezultacie etykiety mogłyby być użyte dla 2^n podpopulacji). Aby wyjaśnić intuicyjne znaczenie zaproponowanego algorytmu zacytujmy [365]:

> Przypuśćmy, że mamy prostą funkcję z dwoma maksimami, przy czym jedno jest dwa razy wyższe od drugiego. Załóżmy dalej, że przyjmujemy jeden bit do oznaczenia każdego z osobników. Każdy taki znacznik jest generowany początkowo losowo. Na początku obliczeń mamy wtedy dwie podpopulacje o w przybliżeniu równych licznościach. Z powodu losowego próbkowania obie te podpopulacje mogą w rezultacie znaleźć się na wyższym maksimum, albo obie na niższym. Jednak niekiedy (z powodu losowego próbkowania) każda z nich może się skierować do innego maksimum. Jeżeli nie mamy podziału dopasowań, to osobniki z wyższego maksimum zawsze będą miały więcej dzieci niż osobniki z niższego i w końcu podpopulacja z niższego maksimum zniknie. Jednak z podziałem dopasowania wyższe maksimum może podtrzymywać tylko dwa razy więcej osobników niż niższe (gdyż jest ono dwa razy wyższe). [...] Mechanizm podziału dopasowania w sposób dynamiczny dostosował obserwowane dopasowanie, tak że dwa maksima mają tę samą obserwowaną wysokość. W wyniku tego obie podpopulacje mogą przeżyć w stabilny sposób. Ponadto ograniczony dobór zapobiega krzyżowaniu między osobnikami z dwóch maksimów, co mogłoby często tworzyć osobniki o niższym dopasowaniu.

Więcej szczegółów i wyniki obliczeń można znaleźć w [365].

[1)] Przy opisie algorytmu założono, że mamy do czynienia z zadaniem maksymalizacji.

W niedawnym artykule Mahfoud [249] porównuje kilka metod z niszami pogrupowanymi w dwie grupy: sekwencyjne i równoległe. Metody równoległe tworzą i utrzymują nisze równocześnie z populacją. Metody sekwencyjne lokują nisze czasowo. Wyniki wskazują na to, że metody równoległe są lepsze od sekwencyjnych. Szczegóły o ich porównaniu i pełne omówienie zalet metod równoległych można znaleźć w [249].

8.3.2. Optymalizacja wielokryterialna

W wielu rzeczywistych zadaniach decyzyjnych trzeba jednocześnie optymalizować wiele celów. Jak podano w [182]:

> Jednym z możliwych sposobów [...] jest zastosowanie długoterminowej maksymalizacji zysku jako jedynego celu. Na pierwszy rzut oka ten sposób wygląda zachęcająco. W szczególności cel długoterminowej maksymalizacji zysku jest na tyle dobrze określony, że można go wygodnie używać, a do tego jest na tyle ogólny, że może objąć podstawowe cele większości organizacji. Faktycznie, niektórym wydaje się, że inne rozsądne cele można sprowadzić do tego jednego. Jednak jest to tak duże uproszczenie, że trzeba zachować wyjątkową ostrożność! Wiele badań wykazało, że dla spółek amerykańskich charakterystycznym celem, zamiast maksymalizacji zysku, jest godziwy zysk połączony z innymi wymaganiami. W szczególności typowym wymaganiem może być utrzymanie stałego zysku, zwiększenie (lub utrzymanie) udziału w sprzedaży, zróżnicowanie towarów, utrzymanie stabilnych cen, poprawienie morale pracowników, utrzymanie kontroli nad spółką w ramach rodziny oraz poprawienie prestiżu spółki. Te cele mogą być zgodne z długofalową maksymalizacją zysku, ale ich wzajemna zależność jest na tyle mało znana, że może być niewygodne skonstruowanie z nich jednego celu.

Takie zadania optymalizacji wielokryterialnej wymagają odrębnych metod, które różnią się od standardowych metod optymalizacji z jedną funkcją celu. Jest całkiem jasne, że jeżeli są dwa cele do optymalizacji, to można by znaleźć rozwiązanie najlepsze w stosunku do jednego celu i inne rozwiązanie, najlepsze w stosunku do drugiego celu.

Wygodnie jest sklasyfikować możliwe rozwiązania zadań optymalizacji wielokryterialnej jako rozwiązania *zdominowane* i *niezdominowane* (*paretooptymalne*). Rozwiązanie $\mathbf{x}$ jest zdominowane, jeżeli istnieje dopuszczalne rozwiązanie $\mathbf{y}$ nie gorsze od $\mathbf{x}$ we wszystkich współrzędnych, to znaczy dla wszystkich celów $f_i (i = 1, ..., k)$

$$f_i(\mathbf{x}) \leqslant f_i(\mathbf{y}) \qquad \text{dla wszytkich} \qquad 1 \leqslant i \leqslant k$$ [1)]

[1)] Dla zadania maksymalizacji, w przeciwnym wypadku nierówność „mniejsze lub równe" należy zamienić na nierówność „większe lub równe".

Jeżeli rozwiązanie nie jest zdominowane przez żadne inne rozwiązanie dopuszczalne, nazywamy je *rozwiązaniem niezdominowanym* (*paretooptymalnym*). Wszystkie paretooptymalne rozwiązania mogą być ważne. Idealnie byłoby, gdyby system wyznaczał zbiór wszystkich punktów paretooptymalnych.

Istnieją klasyczne metody optymalizacji wielokryterialnej [369]. Wśród nich jest metoda ważonych celów, w której poszczególne funkcje celów f_i łączy się w jedną funkcję celu F:

$$F(\mathbf{x}) = \sum_{i=1}^{k} w_i f_i(\mathbf{x})$$

gdzie wagi $w_i \in [0, 1]$ i $\sum_{i=1}^{k} w_i = 1$. Inne wektory wag dają inne rozwiązania paretooptymalne. Inna metoda (metoda funkcji odległości) łączy poszczególne funkcje celów w jedną za pomocą wektora popytu $\mathbf{y}$:

$$F(\mathbf{x}) = \left(\sum_{i=1}^{k} |f_i(\mathbf{x}) - y_i|^r \right)^{\frac{1}{r}}$$

gdzie (zazwyczaj) $r = 2$ (odległość euklidesowska).

Optymalizacja wielokryterialna wzbudziła pewne zainteresowanie w dziedzinie algorytmów genetycznych. W 1984 r. Schaffer [339] opracował program VEGA (*Vector Evaluated Genetic Algorithm*), który był rozszerzeniem programu GENESIS [166] na funkcje wielokryterialne. Główny pomysł zastosowany w systemie VEGA polegał na podziale populacji na podpopulacje (o jednakowych liczebnościach). Każda podpopulacja była „odpowiedzialna" za jeden cel. Procedura selekcji przebiegała niezależnie dla każdego celu, ale krzyżowanie odbywało się przez granice podpopulacji. Opracowano też i badano inne heurystyki (na przykład schemat redystrybucji bogactwa, plan wspólnego wychowywania) w celu zmniejszenia tendencji systemu do zbiegania się do osobników, które nie są najlepsze w stosunku do żadnego celu.

Ostatnio Srinivas i Deb [369] zaproponowali nowy algorytm, NSGA (*Nondominated Sorting Genetic Algoritm*), stosujący kilka warstw klasyfikacji osobników. Przed selekcją populacja jest porządkowana pod względem dominacji. Wszystkie osobniki niezdominowane zalicza się do jednej kategorii (ze sztuczną wartością dopasowania proporcjonalną do liczebności populacji, aby zapewnić tym osobnikom równy potencjał reprodukcyjny). Dla utrzymania różnorodności populacji stosuje się dzielenie tej sztucznej funkcji dopasowania (patrz poprzedni punkt). Następnie tę grupę już sklasyfikowanych osobników pomija się i rozważa się następną warstwę niezdominowanych osobników. Ten proces ciagnie się tak długo, aż sklasyfikuje się wszystkie osobniki w populacji. Pełny opis systemu i pierwsze wyniki obliczeniowe można znaleźć w [369].

Ostatnio Fonseca i Fleming [127] opublikowali przegląd algorytmów ewolucyjnych do optymalizacji wielokryterialnej. Zawarli oni w nim opis obu ro-

dzajów metod (tych, które łączą wiele funkcji celów w jedną funkcję celu i podają jedną wartość, i tych, które uwzględniają paretooptymalność i podają zbiór wartości) oraz postawili kilka pytań, na które nie ma jeszcze odpowiedzi.

8.4. Inne programy ewolucyjne

Jak już pisaliśmy wcześniej, (klasyczne) algorytmy genetyczne nie są odpowiednim narzędziem do dokładnego dopasowania lokalnego. Z tego powodu algorytmy genetyczne dają mniej dokładne rozwiązania zadań optymalizacji numerycznej niż na przykład strategie ewolucyjne, oczywiście o ile reprezentacja osobników w algorytmach genetycznych nie jest zmieniona z binarnej na zmiennopozycyjną, a system (ewolucyjny) nie ma specjalizowanych operatorów (jak mutacja nierównomierna, rozdz. 6). Jednak w ostatniej dekadzie były inne próby poprawienia (bezpośrednio lub pośrednio) tej cechy algorytmów genetycznych.

Ciekawą modyfikację algorytmów genetycznych, zwaną *delta-kodowaniem*, zaproponowali Whitley i in. [400]. Głównym pomysłem tej strategii jest traktowanie osobników w populacji nie jako potencjalnych rozwiązań zadania, ale jako dodatkowe (małe) wartości (zwane *delta-wartościami*), które dodaje się do bieżącego potencjalnego rozwiązania. Uproszczony algorytm delta-kodowania jest przedstawiony na rys. 8.1.

Procedure Delta Kodowanie
begin
  użyj algorytmu genetycznego na poziomie x
  zapamiętaj najlepsze rozwiązanie (**x**)
  while (**not** warunek zakończenia) **do**
  begin
    użyj algorytmu genetycznego na poziomie δ
    zapamiętaj najlepsze rozwiązanie ($\boldsymbol{\delta}$)
    zmodyfikuj najlepsze rozwiązanie (poziom x):
     $\mathbf{x} \leftarrow \mathbf{x} + \boldsymbol{\delta}$
  end
end

Rys. 8.1. Uproszczony algorytm delta-kodowania

W algorytmie delta-kodowania stosuje się metody algorytmów genetycznych na dwóch poziomach: poziomie potencjalnych rozwiązań zadania (poziom x) i (faza iteracyjna) poziomie delta-zmian (poziom δ). Najlepsze rozwiązanie znalezione na poziomie x w jednym wywołaniu algorytmu genetycz-

nego jest zapamiętywane (x) i używane jako punkt odniesienia. Następnie wykonuje się kilka iteracji wewnętrznego algorytmu genetycznego (poziom δ). Zakończenie pojedynczego przebiegu algorytmu genetycznego na tym poziomie (to znaczy do zbiegnięcia się algorytmu genetycznego) powoduje uzyskanie najlepszego wektora modyfikacji $\boldsymbol{\delta}$, który służy do poprawy wartości $\mathbf{x}$. Po poprawie wykonuje sie następną iterację. Każde wywołanie algorytmu genetycznego w fazie iteracyjnej powoduje losową generację początkowej populacji wektora $\boldsymbol{\delta}$. Oczywiście do oceny $\boldsymbol{\delta}$ używamy oceny $\mathbf{x} + \boldsymbol{\delta}$.

Oryginalny algorytm delta-kodowania jest bardziej złożony, gdyż działa na łańcuchach binarnych. W ten sposób delta-kodowanie zachowuje podstawy teoretyczne algorytmów genetycznych (gdyż w każdej iteracji wykonuje się jeden przebieg algorytmu genetycznego). Warunki zakończenia dla algorytmów genetycznych na obu poziomach są wyrażone za pomocą odległości Hamminga między najlepszym i najgorszym członkiem populacji (algorytm kończy się, gdy odległość Hamminga jest nie większa niż 1). Dodatkowo występuje zmienna *len* oznaczająca liczbę bitów reprezentujących pojedynczą składową wektora $\boldsymbol{\delta}$ (właściwie tylko *len* − 1 bitów reprezentuje bezwzględną wartość składowej, ostatni bit jest zarezerwowany na znak). Jeżeli najlepsze rozwiązanie z poziomu δ jest wektorem

$$\boldsymbol{\delta} = (0, 0,...0)$$

(to znaczy nie ma poprawki najlepszego potencjalnego rozwiązania $\mathbf{x}$), to zmienną *len* zwiększa się o jeden (w celu zwiększenia dokładności rozwiązania). Jeżeli nie, to zmniejsza się ją o jeden. Zauważmy także, że w delta-kodowaniu mutacja nie jest konieczna ze względu na generację nowych populacji na poziomie δ w każdej iteracji.

Możemy uprościć oryginalny algorytm delta-kodowania (taki uproszczony algorytm znajduje się na rys. 8.1) i poprawić jego dokładność i czas działania, jeżeli oba wektory $\mathbf{x}$ i $\boldsymbol{\delta}$ przedstawimy w reprezentacji zmiennopozycyjnej.

Niektóre pomysły związane z delta-kodowaniem pojawiły się wcześniej w literaturze. Na przykład Schraudoph i Belew [347] zaproponowali strategię dynamicznego kodowania parametrów, w której dokładność zakodowanego osobnika jest poprawiana dynamicznie. W tym systemie każda składowa wektora rozwiązania jest reprezentowana przez łańcuch binarny o stałej długości. Jednak kiedy (w którejś z iteracji) algorytm zbiega się, najbardziej znaczący bit rozwiązania jest pomijany (oczywiście po zapamiętaniu), a pozostałe bity przesuwa się o jedną pozycję w lewo i wprowadza nowy bit. Ten nowy bit na najmniej znaczącej pozycji zwiększa dokładność przez dokładniejszy podział przeszukiwanej przestrzeni. Proces powtarza się dopóty, dopóki nie jest spełniony pewien globalny warunek zakończenia.

Pomysł generacji nowej populacji w każdej iteracji omawiano w [153], gdzie Goldberg badał właściwości systemu z małą populacją, ale generowaną na nowo za każdym razem, kiedy algorytm genetyczny się zbiega (przy zapa-

miętaniu, oczywiście, najlepszych osobników). Schemat takiej strategii (zwanej *selekcją seryjną*) jest przedstawiony na rys. 8.2.

Generacja populacji w każdej iteracji wprowadza różnorodność między osobnikami z dodatnim efektem w osiąganych przez system wynikach [153].

```
procedure Selekcja Seryjna
begin
  wygeneruj (małą) populację
  while (not warunek zakończenia) do
  begin
    użyj algorytmu genetycznego
    zapamiętaj najlepsze rozwiązanie (x)
    utwórz nową populację, przekazując najlepsze osobniki z populacji,
    która się zbiegła, i generując losowo pozostałe osobniki
  end
end
```

Rys. 8.2. Algorytm genetyczny z generacją populacji

Niektóre zaproponowane strategie zawierają składniki uczące się, w podobny sposób jak w strategach ewolucyjnych. Grefenstette [169] zaproponował optymalizację parametrów sterujących w algorytmach genetycznych (liczebność populacji, współczynniki krzyżowania i mutacji itp.) za pomocą innego, nadzorującego algorytmu genetycznego. Shaefer [354] dyskutował strategię ARGOT (*Adaptive Reprezentation Genetic Optimizer Technique*), w której system uczy się najlepszej wewnętrznej reprezentacji dla osobników.

Inny ciekawy program ewolucyjny selekcyjnej strategii ewolucyjnej (IRM; *Immune Recruitment Mechanism* – mechanizm odpornej rekrutacji) zaproponowano ostatnio w [35] jako metodę optymalizacji w przestrzeniach rzeczywistych (podobny system optymalizacji funkcji w przestrzeniach Hamminga nazwano GIRM). Ta strategia łączy pewne poprzednio już prezentowane pomysły ukierunkowywania poszukiwań w pożądanym kierunku (jak na przykład w przeszukiwaniu tabu). Jak we wszystkich metodach programowania ewolucyjnego, potomka generuje się z bieżącej populacji. W klasycznych algorytmach genetycznych taki potomek zastępuje swojego rodzica. W strategiach ewolucyjnych potomek rywalizuje ze swoim rodzicem (we wczesnych strategiach ewolucyjnych), z rodzicami i innymi potomkami ($(\mu + \lambda)$-ES) lub z innymi potomkami ((μ, λ)-ES). W systemach IRM potomek musi zdać dodatkowy test podobieństwa ze swoimi sąsiadami. Test sprawdza, czy jest on dostatecznie podobny do najbliższych sąsiadów.

Ogólnie, ewentualny kandydat k spełnia test pokrewieństwa, jeżeli

$$\sum_i m(k, i) \cdot f_i > T$$

gdzie i zmienia się wraz z różnymi gatunkami obecnymi w populacji, f_i jest zagęszczeniem gatunku i, $m(k, i)$ jest funkcją pokrewieństwa dla gatunku k oraz i, a T jest progiem rekrutacji.

W strategii IRM ukierunkowuje się poszukiwanie przez akceptację tylko tych osobników, które zdały test pokrewieństwa. Podobne pomysły przedstawiał Glover [142], [145] w przeszukiwaniu rozproszonym i przeszukiwaniu tabu. Metody przeszukiwania rozproszonego, jak inne programy ewolucyjne, utrzymują populację potencjalnych rozwiązań (wektory $\mathbf{x}^i$ są nazywane *punktami odniesienia*). W tej strategii łączy się preferowane podzbiory punktów odniesienia w celu wygenerowania punktu próbnego (potomka) za pomocą ważonej liniowej kombinacji i wybiera się najlepszego członka jako źródło nowych punktów odniesienia (nowej populacji). Nowym pomysłem jest tutaj użycie *krzyżowania wieloosobowego* (nazywanego tutaj *kombinacją ważoną*), w którym do wytworzenia potomka przyczynia się kilkoro rodziców (więcej niż dwoje). W [145] Glover rozszerzył pomysł przeszukiwania rozproszonego przez połączenie go z przeszukiwaniem tabu – metodą, w której ogranicza się wybór nowego potomka (wymaga to pamięci, w której przechowuje się zbiór wcześniej rozważanych osobników) [143], [144]. Struktura algorytmu przeszukiwania rozproszonego i tabu jest przedstawiona na rys. 8.3.

```
procedure Przeszukiwanie Rozproszone i Tabu
begin
  t = 0
  ustal początkowe P(t)
  oceń P(t)
  sklasyfikuj P(t)
  while (not warunek zakończenia) do
  begin
    t = t + 1
    utwórz R(t)
    oceń R(t)
    wybierz P(t) z P(t − 1) i R(t)
    sklasyfikuj P(t)
  end
end
```

Rys. 8.3. Przeszukiwanie rozproszone i tabu

Po inicjalizacji i ocenie algorytm przeszukiwania rozproszonego i tabu klasyfikuje (krok klasyfikuj $P(t)$) populacje rozwiązań $\bar{X}_1, \ldots, \bar{X}_{pop_size}$ na kilka grup. Obejmują one: (1) zbiór historycznych elitarnych generatorów V składający się z pewnej (ustalonej) liczby najlepszych rozwiązań w całym procesie, (2) zbiór generatorów tabu $T \subseteq V$ składający się z rozwiązań obecnie wyłączonych z rozważania, (3) zbiór wybranych historycznych generatorów V^* składający

się z najlepszych elementów $V - T$ oraz (4) zbiór wybranych obecnie generatorów S^* składający się z najlepszych elementów S. Krok klasyfikacji (klasyfikuj $P(t)$) powtarza się później w fazie iteracyjnej algorytmu.

W każdej iteracji tworzy się zbiór $R(t)$ punktów próbnych. Punkty próbne odpowiadają potomkom populacji $P(t)$. Są one oceniane, a niektóre z nich włączane do nowej populacji (wybierz $P(t)$ z $P(t-1)$ oraz $R(t)$).

Ostatnio David Fogel zastosował pomysły programowania ewolucyjnego [126] do optymalizacji funkcji rzeczywistych [119]. Te rozszerzenia zawierają samoadaptację niezależnych wariancji i procedury optymalizacji macierzy kowariancji.

Maniezzo opracował pomysł ewolucji ziarnistej [250], gdzie algorytm umożliwia równoległą ewolucję dyskretyzacji funkcji celu i dokładności dyskretyzacji (czyli granulacji). Osobniki mają zmienną długość z kodowaniem interpretowanym zgodnie z przyjętym poziomem dokładności dyskretyzacji podanym w chromosomie.

Także pomysł algorytmu genetycznego, który przetwarza odcinki, (zwanego *algorytmem genetycznym odcinkowym*) był badany przez Musellego i Ridella [293]. Algorytm genetyczny odcinkowy łączy w sobie pomysły z algorytmów genetycznych i symulowanego wyżarzania. Operatory genetyczne (krzyżowanie, które tworzy nowy odcinek z dwóch odcinków; łączenie, które tworzy potomka z dwóch rodziców jako część wspólną dwóch odcinków; oraz mutację, która szuka lepszego odcinka (jeden rodzic)).

Wydaje się, że najbardziej obiecujący kierunek poszukiwania „optymalnego programu optymalizacji" znajduje się gdzieś pośrodku powyższych pomysłów. Każda strategia wprowadza nowe doświadczenia, które mogą być użyteczne w opracowaniu programu ewolucyjnego dla pewnej klasy zadań. Jak stwierdził Glover [145]:

> Zastosowanie kombinacji strukturalnych pozwala łączyć wektory składowe w sposób, który różni się zasadniczo od [klasycznych] operatorów krzyżowania. Połączenie takich pomysłów z algorytmami genetycznymi może otworzyć furtkę do nowych typów procedur przeszukiwań. Fakt, że ważone i adaptacyjne kombinacje struktur mogą być łatwo tworzone do badania kontekstów, w których krzyżowanie nie ma ewidentnego znaczenia (lub występują z nim trudności w zapewnieniu dopuszczalności), sugeruje, że takie procedury łączonego przeszukiwania mogą mieć wyższość tam, gdzie algorytmy genetyczne mają ograniczone zastosowanie.

Aby stało się to bardziej przejrzyste, przejdźmy do następnego rozdziału.

Część III
Programy ewolucyjne

9

Zadanie transportowe

Konieczność nie zna prawa.

Publilius Syrus, *Moral Sayings*

W rozdziale 7 porównaliśmy różne sposoby uwzględniania ograniczeń w algorytmach genetycznych. Wydaje się, że dla szczególnych klas zadań (jak na przykład zadanie transportowe) możemy postąpić lepiej. Mianowicie możemy użyć odpowiedniejszej (naturalnej) struktury danych (w zadaniu transportowym macierzy) i specjalizowanych operatorów genetycznych, które operują na macierzach. Taki program ewolucyjny powinien być lepszy niż GENOCOP. GENOCOP optymalizuje dowolną funkcję z liniowymi ograniczeniami, gdy tymczasem nowy program ewolucyjny optymalizuje tylko zadania transportowe (te zadania mają dokładnie $n + k - 1$ ograniczeń równościowych, gdzie n i k oznaczają odpowiednio liczbę punktów nadania i odbioru, patrz opis procesu transportowego poniżej). Jednak warto byłoby sprawdzić, co możemy zyskać, wprowadzając do programu ewolucyjnego dodatkowo wiedzę specyficzną dla zadania.

W punkcie 9.1 przedstawiono program ewolucyjny dla liniowego zadania transportowego[1], a w punkcie 9.2 dla nieliniowego zadania transportowego[2].

9.1. Liniowe zadanie transportowe

Zadanie transportowe (patrz na przykład [387]) jest jednym z najprostszych zadań kombinatorycznych z ograniczeniami. Poszukuje się w nim planu najtańszego transportu jednego towaru z pewnej liczby punktów nadania do pew-

[1] Fragmenty tekstu przedrukowane, za pozwoleniem, z *IEEE Transsaction on Systems, Man, and Cybernetics*, Vol. 21, No. 2, s. 445-452, 1991.

[2] Fragmenty tekstu przedrukowane, za pozwoleniem, z *ORSA Journal on Computing*, Vol. 3, No. 4, s. 307-316, 1991.

nej liczby punktów odbioru. Wymaga ono podania poziomu zapasu towaru w każdym punkcie nadania, wielkości zapotrzebowania w każdym punkcie odbioru oraz kosztów transportu z każdego punktu nadania do każdego punktu odbioru.

Ponieważ w zadaniu występuje tylko jeden towar, punkty odbioru mogą uzupełniać swoje zapotrzebowanie z jednego lub więcej punktów nadania. Celem jest wyznaczenie ilości towaru wysłanego z każdego punktu nadania do każdego punktu odbioru, aby zminimalizować całkowity koszt transportu.

Zadanie transportowe jest *liniowe*, jeżeli koszt przejazdu jest wprost proporcjonalny do ilości transportowanego towaru. Jeżeli tak nie jest, to jest ono *nieliniowe*. O ile zadania liniowe można rozwiązać metodami badań operacyjnych, o tyle dla nieliniowych brak jest ogólnej metodyki rozwiązywania.

Załóżmy, że jest n punktów nadania i k punktów odbioru. Poziom zapasów w punkcie nadania i jest równy $sour(i)$, a zapotrzebowanie w punkcie odbioru j wynosi $dest(j)$. Jednostkowy koszt transportu między punktem nadania i oraz odbioru j wynosi $cost(i, j)$. Jeżeli x_{ij} jest ilością towaru transportowanego z punktu nadania i do punktu przeznaczenia j, to możemy zapisać zadanie transportowe następująco:

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

przy ograniczeniach

$$\sum_{j=1}^{k} x_{ij} \leqslant sour(i), \qquad \text{dla } i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} \geqslant dest(j), \qquad \text{dla } j = 1, 2, \ldots, k$$

$$x_{ij} \geqslant 0, \qquad \text{dla } i = 1, 2, \ldots, n \quad \text{i} \quad j = 1, 2, \ldots, k$$

W pierwszym zestawie ograniczeń żąda się, aby całkowita ilość towaru wysłana z punktu nadania nie przekraczała zapasu w tym punkcie. W drugim zestawie wymaga się, aby całkowita ilość towaru przysłanego do punktu odbioru spełniała jego zapotrzebowanie. Jeżeli $f_{ij}(x_{ij}) = cost_{ij}\, x_{ij}$ dla wszystkich i oraz j, to zadanie jest liniowe.

W powyższym zadaniu całkowite zapasy $\sum_{i=1}^{k} sour(i)$ muszą być równe przynajmniej całkowitemu zapotrzebowaniu $\sum_{j=1}^{n} dest(j)$. Kiedy całkowite zapasy są równe całkowitemu zapotrzebowaniu, to zadanie nazywa się *zbilansowanym zadaniem transportowym*. Różni się ono od powyżej zapisanego tylko tym, że odpowiednie ograniczenia są równościowe, to znaczy

$$\sum_{j=1}^{k} x_{ij} = sour(i), \qquad \text{dla } i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = dest(j), \qquad \text{dla } j = 1, 2, ..., k$$

Jeżeli wszystkie $sour(i)$ oraz $dest(j)$ są liczbami całkowitymi, to każde rozwiązanie zbilansowanego zadania transportowego jest całkowitoliczbowe, to znaczy wszystkie x_{ij} ($i = 1, 2, ..., n$; $j = 1, 2, ..., k$) są liczbami całkowitymi. Ponadto liczba dodatnich liczb całkowitych wśród x_{ij} wynosi co najwyżej $k + n - 1$. W tym punkcie założymy, że mamy do czynienia ze zbilansowanym liniowym zadaniem transportowym. Zaczniemy od przykładu. Inne informacje o zadaniu transportowym i bilansowaniu można znaleźć w podstawowych podręcznikach badań operacyjnych, takich jak [387].

Przykład 9.1

Załóżmy, że mamy 3 punkty nadania i 4 punkty odbioru. Zapasy wynoszą:

$$sour[1] = 15, \quad sour[2] = 25 \text{ oraz } sour[3] = 5$$

a zapotrzebowanie:

$$dest[1] = 5, \quad dest[2] = 15, \quad dest[3] = 15 \text{ oraz } dest[4] = 10$$

Zauważmy, że całkowite zapasy i całkowite zapotrzebowanie są równe 45.

Jednostkowe koszty transportu $cost(i, j)$ ($i = 1, 2, 3$ oraz $j = 1, 2, 3, 4$) są podane poniżej w tabeli.

Koszty

10	0	20	11
12	7	9	20
0	14	16	18

Optymalne rozwiązanie jest przedstawione poniżej. Całkowity koszt wynosi 315. Rozwiązanie składa się z całkowitoliczbowych wartości x_{ij}.

Transportowana ilość

	5	**15**	**15**	**10**
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

□

9.1.1. Klasyczne algorytmy genetyczne

Przez „klasyczne" algorytmy genetyczne rozumiemy oczywiście te, w których chromosomy (to znaczy reprezentacje rozwiązań) są łańcuchami binarnymi – ciągami zer i jedynek. Bezpośrednim sposobem zdefiniowania wektora binarnego dla rozwiązania w zadaniu transportowym jest utworzenie wektora $[v_1, v_2, ..., v_p]$ $(p = nk)$ takiego, że każda jego składowa v_i $(i = 1, 2, ..., p)$ jest wektorem binarnym $[w_0^i, ..., w_s^i]$ i reprezentuje liczbę całkowitą związaną z wierszem j i kolumną m macierzy alokacji, gdzie $j = \lfloor (i-1)/k + 1 \rfloor$, a $m = (i-1) \, mod \, k + 1$. Długość wektorów w (parametr s) określa największą liczbę całkowitą $(2^{s+1} - 1)$, którą mogą one reprezentować.

Omówmy krótko wpływ powyższej reprezentacji na spełnienie ograniczeń, funkcji oceny i operatorów genetycznych.

Spełnienie ograniczeń:
Jest oczywiste, że każdy wektor rozwiązań musi spełniać następujące warunki:

- $v_q \geqslant 0$, dla wszystkich $q = 1, 2, ..., k \cdot n$
- $\sum_{i=c \cdot k+1}^{c \cdot k+k} v_i = sour[c+1]$, dla $c = 0, 1, ..., n-1$
- $\sum_{j=m, krok\, k}^{k \cdot n} v_j = dest[m]$, dla $m = 1, 2, ..., k$

Zauważmy, że pierwsze ograniczenie jest zawsze spełnione (interpretujemy sekwencję 0 i 1 jako dodatnią liczbę całkowitą). Pozostałe dwa ograniczenia podają całkowite wartości dla każdego punktu nadania i każdego punktu odbioru, chociaż te wyrażenia nie są symetryczne.

Funkcja oceny:
Naturalna funkcja oceny wyraża całkowity koszt transportu towaru z punktu nadania do punktu odbioru i jest dana wyrażeniem

$$eval(\langle v_1, v_2, ..., v_p \rangle) = \sum_{i=1}^{p} v_i \cdot cost[j][m]$$

gdzie $j = \lfloor (i-1)/k + 1 \rfloor$ oraz $m = (i-1) \, mod \, k + 1$.

Operatory genetyczne:
Nie ma naturalnej definicji operatorów genetycznych dla zadań transportowych z powyższą reprezentacją. Mutację zazwyczaj definiuje się jako zmianę jednego bitu w wektorze rozwiązania. Odpowiada to zmianie jednej wartości całkowitej v_i. To z kolei wymaga w naszym zadaniu serii zmian w różnych miejscach (przynajmniej trzech zmian), aby zachować ograniczenia równościo-

we. Zauważmy także, że musimy zawsze pamiętać, w którym wierszu i rzędzie zrobiono zmianę – mimo reprezentacji wektorowej myślimy i działamy na wierszach i kolumnach (punktach nadania i odbioru). Jest to także powód występowania takich skomplikowanych wzorów. Pierwszą oznaką tego skomplikowania jest utrata symetrii w opisie ograniczeń.

Jest jeszcze więcej nie rozwiązanych zagadnień. Mutację rozumie się jako minimalną zmianę wektora rozwiązań, ale jak zauważyliśmy wcześniej, pojedyncza zmiana w jednej liczbie całkowitej wywołuje przynajmiej trzy inne zmiany w odpowiednich miejscach. Przypuśćmy, że wybrano dwa punkty losowe (v_i i v_m, przy czym $i < m$) tak, że nie należą one do tego samego wiersza ani kolumny. Załóżmy dalej, że v_i, v_j, v_k, v_m ($i < j < k < m$) są składnikami wektora rozwiązań (wybranego do mutacji), tak że v_i i v_k oraz v_j i v_m są z tej samej kolumny, a v_i i v_j oraz v_k i v_m są z tego samego wiersza.

To znaczy w reprezentacji macierzowej

$$\begin{array}{ccccc}
\dots & . & \dots & . & \dots \\
\dots & . & \dots & . & \dots \\
\dots & v_i & \dots & v_j & \dots \\
\dots & . & \dots & . & \dots \\
\dots & . & \dots & . & \dots \\
\dots & v_k & \dots & v_m & \dots \\
\dots & . & \dots & . & \dots \\
\dots & . & \dots & . & \dots
\end{array}$$

Próbując teraz określić najmniejszą zmianę w wektorze rozwiązania, napotykamy na trudność. Czy powinniśmy zwiększyć czy zmniejszyć v_i? Możemy przyjąć, że zmiana wynosi 1 (najmniejsza możliwa zmiana) lub jest jakąś liczbą losową ze zbioru $\{0, 1, ..., v_i\}$. Jeżeli zwiększymy wartość v_i o pewną stałą C, to musimy zmniejszyć każdą wartość v_j i v_k o tę samą stałą. Co się stanie, jeżeli $v_j < C$ lub $v_k < C$? Możemy ustalić $C = \min(v_i, v_j, v_k)$, ale wtedy większość mutacji nie powodowałaby żadnych zmian, gdyż prawdopodobieństwo wyboru trzech niezerowych elementów jest bliskie zeru (mniejsze niż $1/n$ dla wektorów o wymiarze n^2).

Tak więc metoda wywołująca pojedyncze zmiany bitu jest związana z niewydajnym operatorem mutacji ze złożonymi wyrażeniami na sprawdzanie odpowiednich wierszy i kolumn wybranego elementu.

Sytuacja staje się jeszcze bardziej skomplikowana, jeżeli spróbujemy naprawy chromosomów po operatorze krzyżowania. Rozcinanie wektora w przypadkowym punkcie może dać parę chromosomów naruszających liczne ograniczenia. Jeżeli będziemy się starać modyfikować te rozwiązania, aby spełnić wszystkie ograniczenia, stracą one całkiem podobieństwo do rodziców. Ponadto sposób zrobienia tego wcale nie jest oczywisty. Jeżeli wektor v znajduje się poza przestrzenią rozwiązań dopuszczalnych, jego „naprawianie" może być tak trudne, jak rozwiązanie początkowego zadania. Nawet jeżeli uda nam się

zbudować system bazujący na algorytmach naprawy, taki system będzie bardzo uzależniony od zadania z małą szansą na uogólnienie.

Podsumowując, powyższa reprezentacja wektorowa nie jest najbardziej odpowiednia do określania operatorów genetycznych w zadaniach z występującymi tu ograniczeniami.

9.1.2. Uwzględnianie wiedzy specyficznej dla zadania

Czy możemy poprawić reprezentację rozwiązań, zachowując jednocześnie podstawową strukturę reprezentacji wektorowej? Tak nam się wydaje, ale musimy uwzględnić w niej wiedzę specyficzną dla zadania.

Opiszmy najpierw sposób tworzenia rozwiązania, które spełnia wszystkie ograniczenia. Nazwiemy tę procedurę **inicjalizacją**. Będzie to podstawowy składnik operatora mutacji, kiedy będziemy rozważali operatory genetyczne dla struktur dwuwymiarowych. Tworzy ona taką macierz o najwyżej $k + n - 1$ niezerowych elementach, aby spełnione były wszystkie ograniczenia. Przedstawimy teraz krótki szkic algorytmu, a następnie objaśnimy go na macierzy z przykładu 9.1.

wejście: macierze $dest[k]$, $sour[n]$;

wyjście: macierz $(v)_{ij}$ taka, że $v_{ij} \geqslant 0$ dla wszystkich i oraz j,

$$\sum_{j=1}^{k} v_{ij} = dest[i] \text{ dla } i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} v_{ij} = sour[j] \text{ dla } j = 1, 2, ..., k$$

to znaczy są spełnione wszystkie ograniczenia.

procedure inicjalizacja;
begin
  ustaw wszystkie pozycje od 1 do $k \cdot n$ jako nie odwiedzone
  repeat
    wybierz losowo liczbę q z zakresu od 1 do $k \cdot n$ i ustaw
      odpowiednią pozycję jako odwiedzoną
    podstaw (wiersz) $i = \lfloor (q-1)/k + 1 \rfloor$
    podstaw (kolumnę) $j = (q-1) mod\, k + 1$
    podstaw $val = \min(sour[i], dest[j])$
    podstaw $v_{ij} = val$
    podstaw $sour[i] = sour[i] - val$
    podstaw $dest[j] = dest[j] - val$
  until wszystkie wartości będą odwiedzone
end

Przykład 9.2
Dla macierzy z przykładu 9.1, czyli

$$sour[1] = 15, \quad sour[2] = 25 \text{ oraz } sour[3] = 5$$

$$dest[1] = 5, \quad dest[2] = 15, \quad dest[3] = 15 \text{ oraz } dest[4] = 10$$

jest $3 \cdot 4 = 12$ pozycji, wszystkie na początku nie odwiedzone. Wybierzmy pierwszą liczbę losową, powiedzmy 10. Odpowiada ona numerowi wiersza $i = 3$ oraz numerowi kolumny $j = 2$. Mamy więc $val = \min(sour[3], dest[2]) = 5$, czyli $v_{32} = 5$. Zwróćmy uwagę, że po pierwszej iteracji $sour[3] = 0$ oraz $dest[2] = 10$.

Powtarzamy te obliczenia dla trzech następnych losowych (nie odwiedzonych) pozycji, powiedzmy 8, 5 i 3 (odpowiadających odpowiednio rzędowi 2 i kolumnie 4, rzędowi 2 i kolumnie 1 oraz rzędowi 1 i kolumnie 3). Uzyskana jak dotąd macierz v_{ij} ma następującą zawartość:

	0	**10**	**0**	**0**
0			15	
10	5			10
0		5		

Powyższe wartości $sour[i]$ i $dest[j]$ są dane po 4 iteracji.

Jeżeli dalszą sekwencją liczb losowych jest 1, 11, 4, 12, 7, 6, 9, 2, to końcowa otrzymana macierz (z pełną założoną sekwencją liczb losowych ⟨10, 8, 5, 3, 1, 11, 4, 12, 7, 6, 9, 2⟩) ma postać

	0	**0**	**0**	**0**
0	0	0	15	0
0	5	10	0	10
0	0	5	0	0

Oczywiście po 12 iteracjach dla wszystkich (lokalnych kopii) zachodzi $sour[i]$ i $dest[j] = 0$. Zauważmy także, że są sekwencje liczb, po których procedura **inicjalizacja** poda rozwiązania optymalne. Na przykład optymalne rozwiązanie (podane w przykładzie 9.1) można uzyskać po jakiejkolwiek z następujących sekwencji: ⟨7, 9, 4, 2, 6, *, *, *, *, *, *, *⟩, gdzie * oznacza wartość którejkolwiek nie odwiedzonej pozycji, a także po wielu innych sekwencjach.

Tym sposobem można wygenerować rozwiązanie dopuszczalne zawierające najwyżej $k + n - 1$ niezerowych elementów całkowitoliczbowych. Nie wygeneruje się natomiast innych rozwiązań, które są dopuszczalne, ale nie mają tej cechy. Procedura inicjalizacji musiałaby być niewątpliwie zmodyfikowana, gdybyśmy chcieli spróbować rozwiązywać nieliniową wersję zadania transportowego.

Ta wiedza o zadaniu i cechach rozwiązania pozwala na inną możliwość reprezentacji wektorowej rozwiązania zadania transportowego. Wektor rozwiązania jest w niej ciągiem kn różnych liczb całkowitych z zakresu $[1, kn]$, które (zgodnie z procedurą **inicjalizacja**) tworzą akceptowane rozwiązanie. Inaczej mówiąc, rozumiemy rozwiązanie jako permutację liczb i szukamy takiej permutacji, która odpowiada rozwiązaniu optymalnemu.

Omówimy teraz krótko, jak wpłynie ta reprezentacja na spełnienie ograniczeń, funkcję oceny i operatory genetyczne.

Spełnienie ograniczeń:
Jakakolwiek permutacja kn różnych liczb daje jednoznaczne rozwiązanie, które spełnia ograniczenia. Jest to zapewnione przez procedurę **inicjalizacja**.

Funkcja oceny:
Jest ona stosunkowo prosta. Każda permutacja odpowiada jednej macierzy, powiedzmy (v_{ij}). Funkcja oceny ma postać

$$\sum_{i=1}^{k} \sum_{j=1}^{n} v_{ij} \cdot cost[i][j]$$

Operatory genetyczne:
Są one również oczywiste:

- inwersja: każdy wektor rozwiązania $\langle x_1, x_2, ..., x_q \rangle$ $(q = kn)$ można łatwo odwrócić, tworząc inny wektor rozwiązania $\langle x_q, x_{q-1}, ..., x_1 \rangle$,
- mutacja: każde dwa elementy wektora rozwiązania $\langle x_1, x_2, ..., x_q \rangle$, powiedzmy x_i i x_j, można łatwo wymienić, co prowadzi do innego wektora rozwiązania,
- krzyżowanie: jest ono trochę bardziej skomplikowane. Zauważmy, że dowolny (ślepy) operator krzyżowania dawałby niedopuszczalne rozwiązania. Zastosowanie takiego operatora krzyżowania do ciągów

  $\langle 1, 2, 3, 4, 5, 6,| 7, 8, 9, 10, 11, 12 \rangle$ oraz

  $\langle 7, 3, 1, 11, 4, 12,| 5, 2, 10, 9, 6, 8 \rangle$

  dałoby w wyniku (kiedy punkt cięcia jest po 6 pozycji)

  $\langle 1, 2, 3, 4, 5, 6, 5, 2, 10, 9, 6, 8 \rangle$ oraz

  $\langle 7, 3, 1, 11, 4, 12, 7, 8, 9, 10, 11, 12 \rangle$

  żadne z nich nie jest rozwiązaniem dopuszczalnym. Musimy więc użyć pewnej formy heurystycznej operatora krzyżowania. Jest pewne podobieństwo między tymi ciągami wektorów rozwiązań a ciągami w zadaniu komiwojażera (patrz [170]). Tutaj użyjemy heurystycznego operatora krzyżowania

(z rodziny operatorów krzyżowania PMX, patrz [160] i rozdz. 10), który z dwojga rodziców tworzy potomka w następujący sposób:

1. utwórz kopię drugiego rodzica,
2. wybierz dowolną część pierwszego rodzica,
3. zrób najmniejsze zmiany w potomku potrzebne do uzyskania wybranego wzorca.

Na przykład, jeżeli rodzice są tacy, jak w powyższym przykładzie, a wybraną częścią jest

(4, 5, 6, 7)

to uzyskanym potomkiem będzie

⟨3, 1, 11, 4, 5, 6, 7, 12, 2, 10, 9, 8⟩

Tak jak żądano, potomek zawiera strukturalne zależności obu rodziców. Role rodziców można odwrócić dla utworzenia drugiego potomka.

Na podstawie tych zasad zaprogramowano system genetyczny GENETIC-1. Wyniki obliczeń za jego pomocą omówimy w następnym punkcie.

9.1.3. Macierz jako struktura reprezentacji

Być może najbardziej naturalną reprezentacją rozwiązania w zadaniu transportowym jest struktura dwuwymiarowa. Zresztą tak zadanie to jest przedstawiane i rozwiązywane ręcznie. Rozwiązanie jest tu reprezentowane przez macierz $V = (v_{ij})$ $(1 \leqslant i \leqslant k,\ 1 \leqslant j \leqslant n)$.

Zastanówmy się teraz, jak reprezentacja macierzowa wpłynie na spełnienie ograniczeń, funkcję oceny i operatory genetyczne.

Spełnienie ograniczeń:
Jest jasne, że każda macierz rozwiązania $V = (v_{ij})$ powinna spełniać następujące warunki:

- $v_{ij} \geqslant 0$, dla wszystkich $i = 1, ..., k$ oraz $j = 1, ..., n$
- $\sum_{i=1}^{k} v_{ij} = dest[j]$, dla $j = 1, ..., n$
- $\sum_{j=1}^{n} v_{ij} = sour[i]$, dla $i = 1, ..., k$

Są one podobne do zestawu ograniczeń w bezpośrednim podejściu (p. 9.1.2), ale ograniczenia są wyrażone w łatwiejszy i bardziej naturalny sposób.

Funkcja oceny:
Naturalną funkcją oceny jest funkcja celu

$$eval(v_{ij}) = \sum_{i=1}^{k} \sum_{j=1}^{n} v_{ij} \cdot const[j][m]$$

I ponownie wzór ten jest prostszy niż w podejściu bezpośrednim i szybszy do obliczenia niż w systemie GENETIC-1, gdzie każda sekwencja musi być przed oceną przetransformowana (inicjalizowana) na macierz.

Operatory genetyczne:
Określimy tu dwa operatory genetyczne: mutację i krzyżowanie. Trudno jest określić w tym przypadku rozsądny operator inwersji.

- mutacja
  Przypuśćmy że $\{i_1, i_2, ..., i_p\}$ jest podzbiorem $\{1, 2, ..., k\}$, a $\{j_1, j_2, ..., j_q\}$ jest podzbiorem $\{1, 2, ..., n\}$, tak że zachodzi $2 \leqslant p \leqslant k$, $2 \leqslant q \leqslant n$. Niech rodzicem wybranym do mutacji będzie macierz $V = (v_{ij})$ o wymiarach $k \times n$. Tworzymy z elementów macierzy V podmacierz $W = (w_{ij})$ o wymiarach $p \times q$ w następujący sposób: element $v_{ij} \in V$ wchodzi do W wtedy i tylko wtedy, gdy $i \in \{i_1, i_2, ..., i_p\}$ oraz $j \in \{j_1, j_2, ..., j_q\}$ (jeżeli $i = i_r$ oraz $j = j_s$, to element v_{ij} umieszcza się w wierszu r i kolumnie s macierzy W). Możemy teraz przyporządkować macierzy W nowe wartości $sour_W[i]$ oraz $dest_W[j]$ $(1 \leqslant i \leqslant p, 1 \leqslant j \leqslant q)$:

$$sour_W[i] = \sum_{j \in \{j_1, j_2, ..., j_q\}} v_{ij}, \qquad 1 \leqslant i \leqslant p$$

$$dest_W[j] = \sum_{i \in \{i_1, i_2, ..., i_p\}} v_{ij}, \qquad 1 \leqslant j \leqslant q$$

  Do przyporządkowania nowych wartości macierzy W możemy użyć procedury **inicjalizacja** (p. 9.1.3), aby były spełnione wszystkie ograniczenia na $sour_W[i]$ oraz $dest_W[j]$. Po tym zamieniamy odpowiednie elementy macierzy V nowymi elementami z macierzy W. W ten sposób są zachowane globalne ograniczenia (na $sour[i]$ oraz $dest[j]$).
  Następujący przykład zilustruje zastosowanie operatora mutacji.

Przykład 9.3.
Jest dane zadanie z 4 punktami nadania i 5 punktami odbioru oraz następującymi ograniczeniami:

$sour[1] = 8$, $sour[2] = 4$, $sour[3] = 12$ oraz $sour[4] = 6$

$dest[1] = 3$, $dest[2] = 5$, $dest[3] = 10$, oraz $dest[4] = 7$, $dest[5] = 5$

Przypuśćmy, że do mutacji wybrano następującą macierz V:

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

Dalej wybrano (losowo) dwa wiersze $\{2, 4\}$ oraz trzy kolumny $\{2, 3, 5\}$. Odpowiednia podmacierz W przyjmie postać

4	0	0
1	0	2

Zauważmy, że $sour_W[1] = 4$, $sour_W[2] = 3$, $dest_W[1] = 5$, $dest_W[2] = 0$, $dest_W[3] = 2$. Po ponownej inicjalizacji macierzy W może ona przyjąć następującą postać:

2	0	2
3	0	0

Więc, na koniec, macierz V potomka po mutacji będzie następująca:

0	0	5	0	3
0	2	0	0	2
0	0	5	7	0
3	3	0	0	0

□

- krzyżowanie

  Przypuśćmy, że jako rodziców do operacji krzyżowania wybrano dwie macierze $V_1 = (v_{ij}^1)$ i $V_2 = (v_{ij}^2)$. Poniżej przedstawimy szkielet algorytmu, którego używamy do utworzenia pary potomków V_3 i V_4.

  Utwórzmy dwie macierze tymczasowe: $DIV = (div_{ij})$ oraz $REM = (rem_{ij})$. Są one zdefiniowane następująco:

  $$div_{ij} = \lfloor (v_{ij}^1 + v_{ij}^2)/2 \rfloor$$

  $$rem_{ij} = (v_{ij}^1 + v_{ij}^2) \; mod 2$$

  Macierz DIV zawiera zaokrąglone średnie wartości z obu rodziców, a macierz REM zawiera zapis, czy było potrzebne zaokrąglenie. Macierz REM ma kilka interesujących właściwości. Liczba jedynek w każdym wierszu i każdej kolumnie jest parzysta. Inaczej mówiąc, wartości $sour_{REM}[i]$ oraz $dest_{REM}[j]$ (suma wartości odpowiednio w wierszu lub kolumnie macierzy

REM) są parzystymi liczbami rzeczywistymi. Użyjemy tej właściwości do rozkładu macierzy *REM* na dwie macierze REM_1 i REM_2 tak, aby

$$REM = REM_1 + REM_2$$

$$sour_{REM_1}[i] = sour_{REM_2}[i] = sour_{REM}[i]/2, \qquad \text{dla } i = 1, \ldots, k$$

$$dest_{REM_1}[j] = dest_{REM_2}[j] = dest_{REM}[j]/2, \qquad \text{dla } j = 1, \ldots, n$$

Następnie tworzymy potomka V_1 i V_2:

$$V_3 = DIV + REM_1$$

$$V_4 = DIV + REM_2$$

Następujący przykład zilustruje ten przypadek.

Przykład 9.4.
Rozważmy to samo zadanie co w przykładzie 9.1. Załóżmy, że do krzyżowania wybrano jako rodziców następujące macierze V_1 i V_2:

V_1

1	0	0	7	0
0	4	0	0	0
2	1	4	0	5
0	0	6	0	0

V_2

0	0	5	0	3
0	4	0	0	0
0	0	5	7	0
3	1	0	0	2

Macierze *DIV* i *REM* są następujące:

DIV

0	0	2	3	1
0	4	0	0	0
1	0	4	3	2
1	0	3	0	1

REM

1	0	1	1	1
0	0	0	0	0
0	1	1	1	1
1	1	0	0	0

a macierze REM_1 i REM_2:

REM_1

0	0	1	0	1
0	0	0	0	0
0	1	0	1	0
1	0	0	0	0

REM_2

1	0	0	1	0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0

W rezultacie potomkowie V_3 i V_4 są następujący:

V_3

0	0	3	3	2
0	4	0	0	0
1	1	4	4	2
2	0	3	0	1

V_4

1	0	2	4	1
0	4	0	0	0
1	0	5	3	3
1	1	3	0	1

□

Na podstawie powyższych zasad zbudowano system ewolucyjny GENETIC-2. Na początku przeprowadzono próby najpierw dostrojenia parametrów, a następnie porównania zmodyfikowanej wersji klasycznej (z reprezentacją wektorową) algorytmu GENETIC-1 z nową (z reprezentacją macierzową) wersją GENETIC-2 na standardowym liniowym zadaniu transportowym.

Naszym celem nie jest, oczywiście, porównanie metod genetycznych ze standardowymi algorytmami optymalizacji, gdyż jakiekolwiek byśmy wybrali metody pomiaru efektywności, to metody genetyczne nie będą w stanie z nimi rywalizować. Ta sytuacja zmienia się, gdy porównamy metody dla przypadku nieliniowego. Zamiast badać metody na podstawie wyników obliczeń dla zadań o różnych wymiarach i ze znanymi rozwiązaniami, skupimy się na efektach związanych z reprezentacją (porównanie systemów GENETIC-1 oraz GENETIC-2).

Zadania testowe składają się z kilku losowo wygenerowanych zadań o sztucznym charakterze oraz kilku opublikowanych przykładów. W zadaniach o sztucznym charakterze losowo wygenerowano koszty jednostkowe, wartości zaopatrzenia i zapotrzebowania, jednak tak, aby zadania były zbilansowane. Wydawało się nam, że opublikowane przykłady zawierają bardziej typowe struktury kosztów niż zadania wygenerowane sztucznie. Na przykład zadanie z produkcją i magazynem ma wyraźnie rozróżnialny schemat struktury kosztu, kiedy się je przedstawi jako zadanie transportowe.

W każdym przypadku najpierw rozwiązywano zadanie za pomocą standardowego algorytmu transportowego tak, że optimum było znane i można było na tej podstawie opracować kryterium stopu do późniejszego porównania metod.

Zadania miały ograniczone rozmiary ze względu na zastosowane do obliczeń komputery (Macintosh SE/30, Macintosh II, AT&T 3B2 i SUN 3/60, przy czym ten ostatni z systemem operacyjnym Unix). Są one przedstawione w tabl. 9.1.

Przy porównywaniu algorytmów optymalizacji należy się najpierw zdecydować, jakie zastosować kryteria. Jednym z oczywistych kryteriów jest liczba pokoleń potrzebna do osiągnięcia optimum, być może w połączeniu z czasem lub liczbą operacji potrzebnych do uzyskania każdego pokolenia. Poważnym problemem było to, że niekiedy algorytmy genetyczne potrzebowały dużo pokoleń do osiągnięcia optimum. Ponadto przy niektórych zestawach parame-

trów nie osiągnięto w ogóle optimum przed zakończeniem obliczeń. Liczba potrzebnych pokoleń także zależała wyraźnie od wygenerowanych losowo parametrów oraz od rozwiązywanego zadania. Wydaje się więc, że tego kryterium, chociaż jest ono naturalne, nie można łatwo użyć w przeprowadzonych eksperymentach.

Tablica 9.1. Przyjęte zadania

Nazwa zadania	Rozmiar	Literatura
zad 01 do zad 15	4 × 4 do 10 × 10	zadania wygenerowane losowo
sas218	5 × 5	[338], s. 218
taha170	3 × 4	[387], s. 170
taha197	5 × 4	[387], s. 197
win268	9 × 5	[405], s. 268

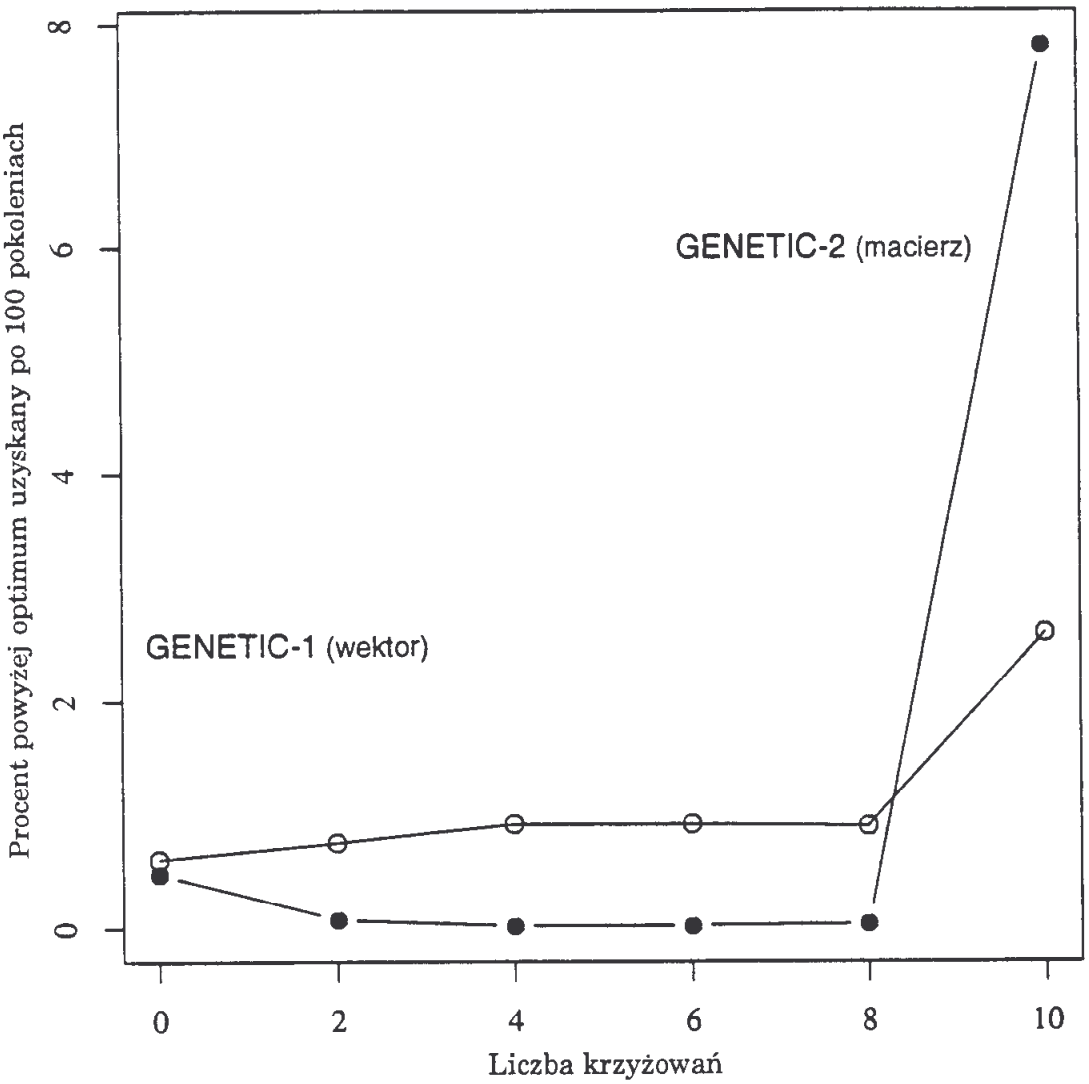
Inne, bardziej praktyczne kryterium można oprzeć na odległości między wartością optymalną a wartością osiągniętą w ustalonej liczbie pokoleń. Zdecydowano się obliczać uzyskany procent ponad wartość w optimum w 100 pokoleniach. Zrobiono także próby dla 1000 pokoleń, ale dla większości przypadków w badanych tu zadaniach rozwiązania uzyskane w tym czasie były optymalne albo bardzo bliskie optymalnych dla obu algorytmów i porównania były niemożliwe.

Inni badacze (patrz na przykład [169]) stwierdzili, że zmiana parametrów algorytmów genetycznych może zmienić ich działanie. Najpierw więc podamy wyniki z badań dostrojenia parametrów w obu metodach. Utrzymujemy stałą liczebność populacji 40 oraz liczbę rozwiązań wybranych do reprodukcji w każdym pokoleniu ustaloną na 40 (25% populacji). Ta ostatnia liczba jest także liczbą rozwiązań usuwanych w każdym pokoleniu. Możemy wtedy dopasować wartości parametrów mutacji *cross*, *inv* i *mut*, to znaczy liczbę rodziców wybranych do reprodukcji odpowiednio przez *krzyżowanie*, *inwersję* i *mutację*, tak aby ich suma wynosiła 10. Liczba rozwiązań wybranych do krzyżowania cross musi być parzysta, gdyż w krzyżowaniu używa się pary rodziców. Inwersja jest możliwa tylko w wersji z reprezentacją wektorową, czyli w GENETIC-1. Ustalono także parametr *sprob*, który występuje w geometrycznym rozkładzie prawdopodobieństwa zastosowanego do wyboru rodziców oraz usuwania rozwiązań.

W procesie dostrajania wykonano ponad 1000 obliczeń. Były one wykonywane dla pięciu różnych zestawów liczb losowych dla każdej wybranej kombinacji parametrów. Średnią wartość funkcji celu z tych pięciu obliczeń użyto do wyliczenia procentowej różnicy od znanej wartości optymalnej.

Na rysunku 9.1 jest przedstawiony przykładowy efekt zmiany liczby par w krzyżowaniu *cross* w dwóch programach dla jednego z opublikowanych

zadań, sas218. Podobne, choć nie identyczne wyniki uzyskano dla innych opublikowanych i wygenerowanych zadań. Ponieważ ustalono całkowitą liczbę rodziców, więcej krzyżowań oznaczało mniej mutacji, a dla modelu wektorowego także mniej inwersji. Każdy punkt na rysunku jest średnią z pięciu obliczeń z różnymi zestawami liczb losowych. Wykreślono procentową nadwyżkę nad wartością optymalną uzyskaną w 100 pokoleniach.



Rys. 9.1. Wyniki z dwóch algorytmów dla zadania sas218

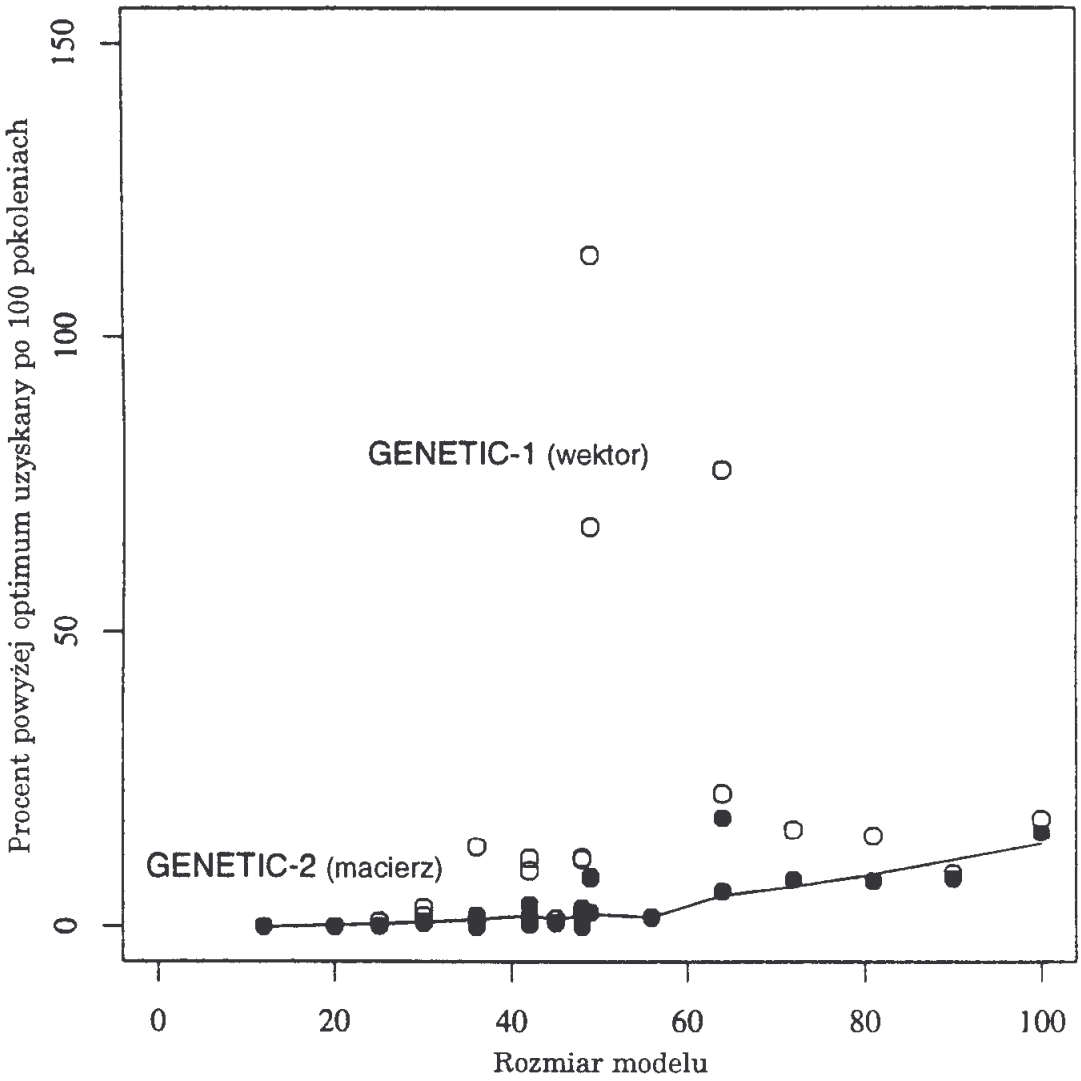
Ogólnie wersja macierzowa GENETIC-2 dała gładsze krzywe w funkcji liczby krzyżowanych par. Zazwyczaj, im mniej krzyżowań, tym lepiej, z najlepszymi wynikami uzyskanymi przy zerze krzyżowanych par. Ale wyniki zależą od wyboru zadania. Zadanie sas218 różni się od większości innych tym, że najlepsze wyniki uzyskano dla 2 do 4 krzyżowanych par. Wyniki z GENETIC-1 wykazują generalnie wzrastający procent przy wzroście liczby krzyżowań, chociaż nie zawsze uzyskano taką zależność. W obu przypadkach z rysunku wynika, że sytuacja pogarsza się, gdy do krzyżowania wybiera się wszystkie pary, nie zostawiając żadnego osobnika do losowej mutacji. Model

GENETIC-2 jest szczególnie wrażliwy na ten efekt i w takim przypadku jest dużo gorszy od GENETIC-1 w przedstawianych tu obliczeniach.

Dla badanej klasy zadań stwierdzono, chociaż wyniki różnią się dla poszczególnych zadań, że mała proporcja krzyżowań daje lepsze wyniki dla obu modeli, a dla modelu wektorowego najlepsze wyniki są przy zerowej inwersji.

Po wybraniu optymalnych parametrów przeprowadzono porównanie modeli, prowadząc obliczenia dla zestawu zadań. I ponownie podane wyniki są za każdym razem średnią z pięciu obliczeń dla różnych zbiorów liczb losowych.

Na rysunku 9.2 są pokazane wyniki obliczeń wykreślone względem wymiaru zadania ($n * k$) dla całego zestawu zadań przy 2 parach wybieranych do krzyżowania. W każdym przypadku system z reprezentacją macierzową GENETIC-2 działa lepiej (to znaczy dochodzi bliżej do optimum w 100 pokoleniach) niż system z reprezentacją wektorową GENETIC-1. Nigdy wersja wektorowa nie była lepsza od macierzowej w tym samym zadaniu. GENETIC-1 był także znacznie bardziej niepewny niż GENETIC-2. Jest to szczególnie frapujące w niektórych zadaniach, jak to widać z wartości oddalonych na rysunkach.



Rys. 9.2. Porównanie GENETIC-1 i GENETIC-2

9.1.4. Wnioski

Stwierdziliśmy więc, że w tych obliczeniach algorytm macierzowy (GENETIC-2) działa lepiej niż wersja wektorowa (GENETIC-1) przy zastosowaniu naszych kryteriów. Zwróćmy uwagę, że to porównanie dotyczy wersji macierzowej oraz *specjalnie opracowanego* modelu wektorowego. O ile algorytm macierzowy zawiera w sposób naturalny wiedzę specyficzną dla zadania, o tyle model wektorowy do działania wymaga dodatkowych założeń. Na podstawie szczegółowej analizy zadania trzeba było opracować specjalną procedurę **inicjalizacja**, aby wersja wektorowa mogła w ogóle pracować. Z tego powodu specjalnej wersji wektorowej (GENETIC-1) nie można łatwo uogólnić, gdy tymczasem podejście macierzowe (GENETIC-2) potencjalnie lepiej nadaje się do uogólnienia, także dla bardziej skomplikowanej nieliniowej wersji zadania transportowego. Tam GENETIC-1 nie może działać właściwie. Procedura **inicjalizacja**, będąca podstawą systemu, zależy bardzo od znajomości formy rozwiązania w przypadku liniowym. Kiedy rozwiązanie optymalne nie musi być macierzą liczb całkowitych i liczba elementów niezerowych może być znacznie większa niż $k + n - 1$, to system GENETIC-1 musi zawieść. Nawet jeżeli zmienimy procedurę **inicjalizacja** (na przykład dla każdego wybranego i-tego wiersza oraz j-tej kolumny przyporządkujemy zmiennej *val* liczbę losową z zakresu $[0, min(sour[i], dest[j])]$), to wektor reprezentujący ciąg punktów inicjalizacji powinien być rozszerzony, aby także objąć wybrane liczby losowe. Wszystkie te trudności są spowodowane sztuczną reprezentacją wektora rozwiązań.

Algorytm genetyczny jest bardzo wolny i w żaden sposób nie można go porównać ze specjalnymi metodami optymalizacji z wykorzystaniem standardowych algorytmów programowania liniowego. Te ostatnie rozwiązują zadania z liczbą iteracji rzędu wymiaru zadania ($n * k$), gdy tymczasem każde pokolenie w metodzie genetycznej wymaga utworzenia zbioru potencjalnych rozwiązań zadania. Jednak może ona być zastosowana w zadaniach nieliniowych i z ustalonym ładunkiem, gdzie standardowych metod transportowych nie da się użyć (patrz następny punkt).

9.2. Nieliniowe zadanie transportowe

Przedyskutujemy teraz program ewolucyjny dla zbilansowanego nieliniowego zadania transportowego, omawiając pięć składników algorytmów genetycznych: reprezentację, inicjalizację, ocenę, operatory i parametry. Algorytm nazwano GENETIC-2 (jak dla przypadku liniowego).

9.2.1. Reprezentacja

Tak jak w przypadku liniowym, do reprezentacji rozwiązania (chromosomu) dla zadania transportowego wybrano strukturę dwuwymiarową – macierz $V = (x_{ij})(1 \leqslant i \leqslant k,\ 1 \leqslant j \leqslant n)$. Tym razem każde x_{ij} jest liczbą rzeczywistą.

9.2.2. Inicjalizacja

Procedura inicjalizacji jest identyczna jak dla przypadku liniowego (p. 9.1.3). Tak jak w przypadku liniowym, tworzy ona macierz o najwyżej $k + n - 1$ niezerowych elementach, dla której są spełnione wszystkie ograniczenia. Chociaż są możliwe inne procedury inicjalizacji, ta metoda generuje rozwiązanie, które jest wierzchołkiem simpleksu, opisującego wypukły brzeg przestrzeni ograniczonych rozwiązań.

9.2.3. Ocena

W tym przypadku musimy minimalizować koszt, nieliniową funkcję elementów macierzy. Wybrano do tego kilka funkcji (p. 9.2.6), a wyniki obliczeń przedstawiono w p. 9.2.7.

9.2.4. Operatory

Określamy dwa operatory genetyczne: *mutację* i *krzyżowanie arytmetyczne*.

- **Mutacja**
  Określamy dwa typy mutacji. Pierwszy, **mutacja-1**, jest identyczny z tym, którego używaliśmy w przypadku liniowym. Wprowadza on tyle elementów zerowych do macierzy, ile potrzeba. Drugi, **mutacja-2**, jest zmodyfikowany, aby zamiast elementów zerowych wybierać wartości z zakresu określoności. Operator **mutacja-2** jest taki sam jak **mutacja-1** z tym wyjątkiem, że przy ponownym obliczaniu zawartości wybranych podmacierzy korzysta się ze zmodyfikowanej wersji podprogramu **inicjalizacja**. Zmiana ta polega na tym, że wiersz:

  set val = min(*sour*[*i*], *dest*[*j*])

  jest zastąpiony przez:

  set val_1 = min(*sour*[*i*], *dest*[*j*])
  if (*i* jest ostatnim możliwym wierszem) **or**
  (*j* jest ostatnią możliwą kolumną)
  then *val* = val_1
  else podstaw *val* = losowa (rzeczywista) liczba z zakresu [0, val_1]

Ta zmiana prowadzi do liczb rzeczywistych zamiast całkowitych, ale procedurę trzeba dalej zmodyfikować, gdyż obecnie tworzy macierz, która może naruszać ograniczenia.

Na przykład dla macierzy z przykładu 9.1 przypuśćmy, że ciągiem wybranych liczb jest (3, 6, 12, 8, 19, 1, 2, 4, 9, 11, 7, 5) i że pierwszą liczbą rzeczywistą wygenerowaną dla 3 (pierwszy wiersz, trzecia kolumna) jest 7,3 (która jest z zakresu [0,0, min(*sour*[1], *dest*[3])] = [0,0, 15,0]). Drugą rzeczywistą liczbą losową dla 6 (drugi wiersz, druga kolumna) jest 12,1, a reszta liczb rzeczywistych wygenerowanych przez nowy algorytm inicjalizacji to: 3,3, 5,0, 1,0, 3,0, 1,9, 1,7, 0,4, 0,3, 7,4, 0,5. Otrzymana macierz ma postać

	5,0	**15,0**	**15,0**	**10,0**
15,0	3,0	1,9	7,3	1,7
25,0	0,5	12,1	7,4	5,0
5,0	0,4	1,0	0,3	3,3

Możemy tylko wtedy spełnić ograniczenia, kiedy dodamy 1,1 do elementu x_{11}. Dlatego musimy dodać jeszcze końcowy wiersz do algorytmu **mutacja-2**:

zrób konieczne dodawania

co kończy modyfikację procedury **inicjalizacja**.

- **Krzyżowanie**
  Zaczynając od dwóch rodziców (macierze U i V), operator krzyżowania tworzy dwóch potomków X i Y, przy czym

  $$X = c_1 U + c_2 V \text{ oraz } Y = c_1 V + c_2 U$$

  (gdzie $c_1, c_2 \geqslant 0$ i $c_1 + c_2 = 1$). Ponieważ zbiór ograniczeń jest wypukły, operacja ta zapewnia, że obaj potomkowie są dopuszczalni, jeżeli tylko byli dopuszczalni rodzice. Jest to znaczne uproszczenie w stosunku do przypadku liniowego, gdzie istniało dodatkowe wymaganie, że wszystkie składowe macierzy muszą być liczbami całkowitymi.

9.2.5. Parametry

Prócz zbioru parametrów sterujących używanych w przypadku liniowym (liczebność populacji, współczynniki mutacji i krzyżowania, liczba początkowa do generacji liczb losowych itd.) potrzebnych jest tu jeszcze kilka innych. Są to: proporcje krzyżowania c_1 i c_2 oraz m_1, parametr określający proporcję stosowania mutacji-1 w stosowanych mutacjach.

9.2.6. Zadania testujące

W celu uzyskania pewnego wyobrażenia o użyteczności zaproponowanego algorytmu wybrano jeden przykład zadania transportowego o wymiarach 7 × 7 (tabl.9.2), na którym badano przydatność różnych funkcji celu.

Tablica 9.2. Zadanie transportowe o wymiarze 7 × 7

	20	20	20	23	26	25	26
27	x_1	x_2	x_3	x_4	x_5	x_6	x_7
28	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}
25	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}	x_{21}
20	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}
20	x_{29}	x_{30}	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
20	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}	x_{41}	x_{42}
20	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}

Zadanie polega na minimalizacji funkcji

$$f(\mathbf{x}) = f(x_1, \ldots, x_{49})$$

przy czternastu (trzynastu niezależnych) ograniczeniach równościowych:

$$\begin{aligned}
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 &= 27 \\
x_8 + x_9 + x_{10} + x_{11} + x_{12} + x_{13} + x_{14} &= 28 \\
x_{15} + x_{16} + x_{17} + x_{18} + x_{19} + x_{20} + x_{21} &= 25 \\
x_{22} + x_{23} + x_{24} + x_{25} + x_{26} + x_{27} + x_{28} &= 20 \\
x_{29} + x_{30} + x_{31} + x_{32} + x_{33} + x_{34} + x_{35} &= 20 \\
x_{36} + x_{37} + x_{38} + x_{39} + x_{40} + x_{41} + x_{42} &= 20 \\
x_{43} + x_{44} + x_{45} + x_{46} + x_{47} + x_{48} + x_{49} &= 20
\end{aligned}$$

$$\begin{aligned}
x_1 + x_8 + x_{15} + x_{22} + x_{29} + x_{36} + x_{43} &= 20 \\
x_2 + x_9 + x_{16} + x_{23} + x_{30} + x_{37} + x_{44} &= 20 \\
x_3 + x_{10} + x_{17} + x_{24} + x_{31} + x_{38} + x_{45} &= 20 \\
x_4 + x_{11} + x_{18} + x_{25} + x_{32} + x_{39} + x_{46} &= 23 \\
x_5 + x_{12} + x_{19} + x_{26} + x_{33} + x_{40} + x_{47} &= 26 \\
x_6 + x_{13} + x_{20} + x_{27} + x_{34} + x_{41} + x_{48} &= 25 \\
x_7 + x_{14} + x_{21} + x_{28} + x_{35} + x_{42} + x_{49} &= 26
\end{aligned}$$

Zachowanie się algorytmu optymalizacji nieliniowej zależało wyraźnie od funkcji celu. Oczywiście w innych metodach rozwiązywania może być inaczej.

Do testowania poklasyfikowano w sposób arbitralny funkcje celu na te, które mogłyby być uważane za zadania praktyczne badań operacyjnych (prak-

tyczne), te, które można głównie zobaczyć w podręcznikach optymalizacji (rozsądne), i te, które częściej widuje się jako przypadki trudne dla metod optymalizacji (inne). Można je krótko scharakteryzować następująco.

- Funkcje praktyczne
  Na ogół funkcje kosztów kawałkami liniowe. Pojawiają się one w zadaniach praktycznych albo z powodu ograniczonych danych, albo ponieważ przewóz zależy od obszarów działania, z różnym kosztem. Często nie są one gładkie i oczywiście pochodne mogą być nieciągłe. Na ogół sprawiają one trudności w metodach gradientowych, chociaż są możliwe aproksymacje funkcjami różniczkowalnymi. Przykłady: A(x) i B(x).
- Funkcje rozsądne
  Tu funkcje są gładkie i często są potęgami przewozów. Można je dalej klasyfikować jako funkcje wypukłe i wklęsłe. Przykłady: C(x) i D(x).
- Funkcje inne
  Te funkcje na ogół mają wielokrotne doliny (lub szczyty) z lokalnymi optimami, które sprawiają trudności w każdej metodzie gradientowej. Są one wymyślone jako ciężkie testy dla algorytmów optymalizacji i przypuszczalnie nieczęsto pojawiają się w pratyce. Przykłady: E(x) i F(x).

Podamy po dwa przykłady z każdej grupy użytych do testowania funkcji celu. Są to wszystko funkcje separowalne względem składników wektora rozwiązania, bez iloczynów tych składników. Ciągła wersja ich wykresów (zmodyfikowana przez system GAMS) jest przedstawiona na rys. 9.3.

- Funkcja A

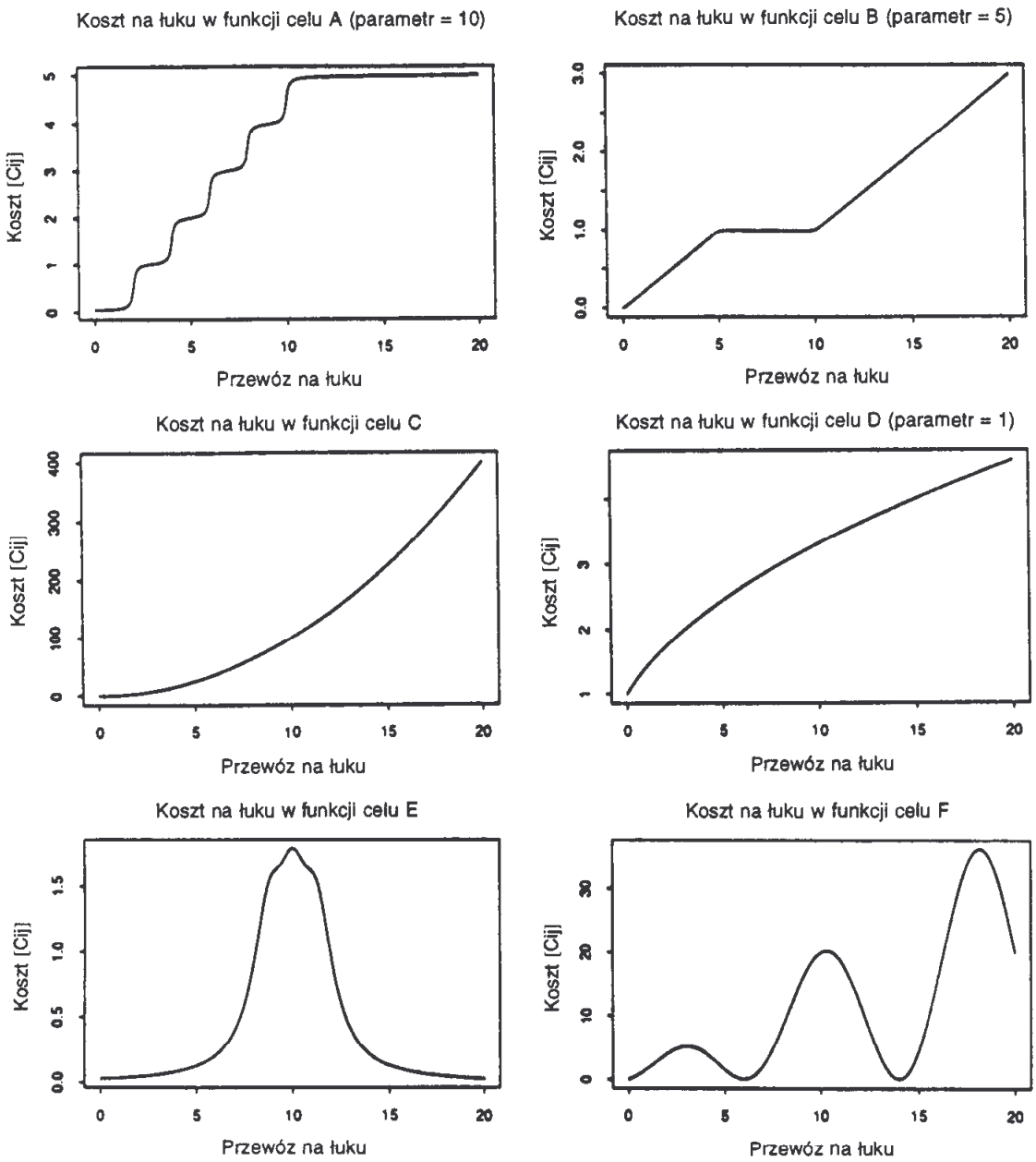
$$A(x) = \begin{cases} 0, & \text{jeżeli } 0 < x \leqslant S \\ c_{ij}, & \text{jeżeli } S < x \leqslant 2S \\ 2c_{ij}, & \text{jeżeli } 2S < x \leqslant 3S \\ 3c_{ij}, & \text{jeżeli } 3S < x \leqslant 4S \\ 4c_{ij}, & \text{jeżeli } 4S < x \leqslant 5S \\ 5c_{ij}, & \text{jeżeli } 5S < x \end{cases}$$

gdzie S jest mniejsze niż typowa wartość x.

- Funkcja B

$$B(x) = \begin{cases} c_{ij}\dfrac{x}{S}, & \text{jeżeli } 0 \leqslant x \leqslant S \\ c_{ij}, & \text{jeżeli } S < x \leqslant 2S \\ c_{ij}\left(1 + \dfrac{x - 2S}{S}\right), & \text{jeżeli } 2S < x \end{cases}$$

gdzie S jest rzędu typowej wartości x.



Rys. 9.3. Sześć funkcji testowych A - F

- Funkcja C

  $$C(x) = c_{ij}x^2$$

- Funkcja D

  $$D(x) = c_{ij}\sqrt{x}$$

- Funkcja E

  $$E(x) = c_{ij}\left(\frac{1}{1+(x-2S)^2} + \frac{1}{1+\left(x-\frac{9}{4}S\right)^2} + \frac{1}{1+\left(x-\frac{7}{4}S\right)^2}\right)$$

  gdzie S jest rzędu typowej wartości x.

- Funkcja F

$$F(x) = c_{ij}x\left(\sin\left(x\frac{5\pi}{4S}\right) + 1\right)$$

gdzie S jest rzędu typowej wartości x.

Funkcja celu w zadaniu transportowym ma postać

$$\sum_{ij} f(x_{ij})$$

gdzie $f(x)$ jest jedną z powyższych funkcji, parametry c_{ij} są wzięte z macierzy parametrów (patrz rys. 9.4), a S jest cechą zadania testowego.

Liczba punktów nadania: 7

Liczba punktów odbioru: 7

Wywozy z punktów nadania: 27 28 25 20 20 20 20

Przywozy do punktów odbioru: 20 20 20 23 26 25 26

Macierz parametrów łuków (punkty nadania na punkty odbioru):

0	21	50	62	93	77	1000
21	0	17	54	67	1000	48
50	17	0	60	98	67	25
62	54	60	0	27	1000	38
93	67	98	27	0	47	42
77	1000	67	1000	47	0	35
1000	48	25	38	42	35	0

Rys. 9.4. Opis zadania testowego

Do wyznaczenia S trzeba oszacować typową wartość x. Zrobiono to za pomocą próbnych obliczeń, na podstawie których oszacowano liczbę i wielkości niezerowych elementów x_{ij}. W ten sposób szacowano średni przewóz po każdym łuku i wyznaczano wartość S. Dla funkcji A przyjęto $S = 2$, a dla B, E i F użyto $S = 5$.

Zauważmy, że funkcja celu jest identyczna na każdym łuku, więc użyto macierzy kosztów, aby uzyskać ich zróżnicowanie. Z tej macierzy bierze się c_{ij}, które skalują podstawowy kształt funkcji, doprowadzając do „jednego stopnia swobody".

9.2.7. Obliczenia i wyniki

Testując algorytm GENETIC-2 na liniowych zadaniach transportowych (p. 9.1), porównywano jego rozwiązania ze znanym optimum wyznaczonym za

pomocą standardowych algorytmów. W związku z tym można było określić, na ile są efektywne te algorytmy w terminach absolutnych. W nieliniowych funkcjach celu optimum może nie być znane. Wobec tego testowanie sprowadza się do porównania z wynikami z innych metod, które jednak mogą zatrzymać się w lokalnym optimum.

Jak zwykle, porównamy algorytm GENETIC-2 z systemem GAMS, jako typowym przykładem metody o obecnej standardowej efektywności rozwiązań. Dla tego systemu, stosującego w zasadzie metodę gradientową, niektóre z przyjętych zadań okazały się zbyt trudne lub niemożliwe do rozwiązania. W tych przypadkach można było zmodyfikować funkcję celu tak, aby mógł on przynajmniej znaleźć przybliżone rozwiązanie.

Funkcja celu dla zadania transportowego miała więc postać

$$\sum_{ij} f(x_{ij})$$

gdzie $f(x)$ jest jedną z sześciu wybranych funkcji, parametry c_{ij} są wzięte z macierzy parametrów, a S jest cechą zadania testowego. S aproksymowano jako średni niezerowy przewóz po łuku określony na podstawie próbnych obliczeń, aby mieć pewność, że przewozy odbywają się w tej części, którą obejmuje funkcja celu.

W pewnym sensie pożądane byłoby użycie na każdym łuku całkiem losowej struktury funkcji celu. Zakładając, że naszym celem jest przedstawienie jak działa algorytm na różnych zadaniach, sprawa sprowadza się do pytania, jak duża jest potrzebna zmienność między łukami dla poszczególnej funkcji. Kiedy funkcja jest identyczna na każdym łuku, zadanie może mieć wiele rozwiązań z tym samym kosztem, co zmniejsza informację uzyskaną z analizy algorytmu.

W naszych obliczeniach użyto macierzy kosztów do wprowadzenia zmienności między łukami. Z macierzy tej bierze się c_{ij}, które skalują podstawowy kształt funkcji, wprowadzając w ten sposób „jeden stopień swobody". Więcej macierzy (wprowadzających więcej stopni swobody) nie potrzebowano.

Dla funkcji C, E i F użycie systemu GAMS było oczywiste – zrobiono to dla wprowadzonych wcześniej nieliniowych funkcji celu. Z powodu konieczności wyznaczania gradientu funkcji celu, GAMS nie mógł sobie dać wprost rady z funkcjami A, B i D. W przypadku A i B wyrażenia nie mogły być zapisane w GAMS, a w przypadku D (pierwiastek) napotkano trudności z wyznaczaniem gradientu w pobliżu zera. Z tego powodu zrobiono następujące modyfikacje zadań do obliczeń za pomocą GAMS:

- Funkcja A
  Do approksymacji każdego z pięciu kroków użyto oddzielnych funkcji arcus tangens. Wprowadzono parametr P_A do sterowania „dokładnością" dopasowania. Koszt na łuku [i, j] wynosił

$$c_{ij} \cdot \begin{pmatrix} \operatorname{arctg}(P_A(x_{ij} - S))/\pi + \frac{1}{2} + \\ \operatorname{arctg}(P_A(x_{ij} - 2S))/\pi + \frac{1}{2} + \\ \operatorname{arctg}(P_A(x_{ij} - 3S))/\pi + \frac{1}{2} + \\ \operatorname{arctg}(P_A(x_{ij} - 4S))/\pi + \frac{1}{2} + \\ \operatorname{arctg}(P_A(x_{ij} - 5S))/\pi + \frac{1}{2} \end{pmatrix}$$

- Funkcja B
  Ponownie użyto funkcji arcus tangens, tym razem do aproksymacji każdego z trzech gradientów. Do sterowania dokładnością dopasowania wprowadzono parametr P_B. Koszt na łuku [i, j] wynosił

$$c_{ij} \cdot \begin{pmatrix} \left(\frac{x_{ij}}{S}\right) \cdot (\operatorname{arctg}(P_B(x_{ij}))/\pi + \frac{1}{2}) + \\ \left(\frac{1 - x_{ij}}{S}\right) \cdot (\operatorname{arctg}(P_B(x_{ij} - S))/\pi + \frac{1}{2}) + \\ \left(\frac{x_{ij}}{S} - 2\right) \cdot (\operatorname{arctg}(P_B(x_{ij} - 2S))/\pi + \frac{1}{2}) \end{pmatrix}$$

- Funkcja D
  Aby uniknąć kłopotów z liczeniem gradientu w zerze lub w pobliżu zera zmieniono funkcję D na

$$D'(x) = D(x + \varepsilon)$$

Tak jak w przypadku liniowym, dla każdego zadania wykonano wiele obliczeń z różnymi wartościami powyższych parametrów modyfikujących krzywe i wybrano z nich najlepsze wyniki. Najlepszymi znalezionymi wartościami tych trzech parametrów były: P_A między 1 i 20, P_B bardzo duże (na przykład 1000) i ε (dla funkcji D) między 1 a 7. Końcowe wynikowe wartości obliczano zawsze po optymalizacji na podstawie początkowych funkcji, a nie zmodyfikowanych.

W podstawowym zestawie obliczeń użyto macierzy transportowych o wymiarach 10×10 dla każdej funkcji. Utworzono je ze zbioru niezależnych wartości c_{ij} o jednakowych rozkładach oraz losowo wybranych wektorach nadania i odbioru z pełnym przewozem 100 jednostek. Dla każdej kombinacji funkcji i macierzy wykonano 5 obliczeń z różnymi liczbami losowymi dla algorytmu genetycznego. Zadania liczono do 10 000 pokoleń.

Dla funkcji A przyjęto wartość S równą 2, a dla funkcji B, E i F wartość 5.

Zadania o 10×10 wierzchołkach osiągały granice możliwości wersji studenckiej GAMS (gdzie dopuszczalny rozmiar zadania jest ograniczony). Z wydruków kilku przykładowych zadań testowych dla systemu GAMS wynika, że z pełną wersją (w której wymiar zadania jest ograniczony przez dostępną pa-

mięć i wewnętrzne ograniczenia) powinno być możliwe obliczenie na komputerze AT z pamięcią 640 KB zadania o 25 × 25 wierzchołkach. Zwróćmy uwagę, że zadanie o N × N wierzchołkach jest traktowane przez GAMS jako zadanie z N^2 zmiennymi, 2N ograniczeniami i nieliniową funkcją celu. Rzecz jasna można by było liczyć większe zadania na większych komputerach lub ze specjalizowanym oprogramowaniem.

Jednak użycie dużo większych zadań do porównania systemów genetycznych z oprogramowaniem dla programowania nieliniowego może mieć ograniczoną wartość. Wyniki obliczeń dla zadań 10 × 10 przedstawiają tendencję systemu GAMS (a przypuszczalnie i innych podobnych systemów) do wpadania w lokalne (nieglobalne) optima. Pomijając czas spędzony na obliczaniu funkcji celu i używając liczby badanych rozwiązań jako miary czasu obliczeń, jest jasne, że standardowe metody programowania nieliniowego zawsze „skończą" szybciej niż systemy genetyczne. Jest to spowodowane tym, że typowo badają one tylko szczególną ścieżkę w strefie bieżącego optimum lokalnego. Zadziałają więc one dobrze tylko wtedy, kiedy optimum lokalne jest stosunkowo dobre.

Zestaw parametrów dla GENETIC-2 wybrano po doświadczeniach z zadaniami liniowymi oraz na podstawie obliczeń próbnych dla zadań nieliniowych. Liczebność populacji ustalono na 40. Współczynnik mutacji wynosił $p_m = 20\%$, a proporcja mutacji-1 wynosiła 50%. Współczynnik krzyżowania był równy $p_c = 5\%$. Proporcje krzyżowania wynosiły $c_1 = 0{,}35$ oraz $c_2 = 0{,}65$.

Może się wydawać, że wybrany współczynnik mutacji jest zbyt duży, a współczynnik krzyżowania zbyt mały w porównaniu z klasycznymi algorytmami genetycznymi. Jednak nasze operatory różnią się od klasycznych, ponieważ: (1) wybiera się rodziców do mutacji i krzyżowania, to znaczy *cała* struktura (w przeciwieństwie do pojedynczego bitu) podlega mutacji, (2) mutacja-1 „wypycha" potomków w kierunku powierzchni granicznej przestrzeni rozwiązań, gdy tymczasem krzyżowanie i mutacja-2 „wpychają" potomka do środka przestrzeni rozwiązań.

Zastosowanie wysokiego współczynnika mutacji może także sugerować, że algorytm jest prawie losowym przeszukiwaniem. Jednak algorytmy losowego przeszukiwania (współczynnik krzyżowania 0%) działają raczej słabo w porównaniu z dostrojonym algorytmem, którego się tu używa. Aby to zademonstrować, przedstawiamy w tabl. 9.3 kilka typowych wyników dla różnych wartości parametrów p_m i p_c dla funkcji A i F oraz konkretnego zadania transportowego o wymiarze 7 × 7. Podane wartości są średnimi z minimalnych kosztów osiągniętych po 5 obliczeniach dla różnych liczb losowych i przy 10 000 pokoleń.

Tablica 9.3. Wyniki dla różnych wartości p_c i p_m

Funkcja	$p_c = 0\%$ $p_m = 25\%$	$p_c = 25\%$ $p_m = 0\%$	$p_c = 5\%$ $p_m = 20\%$
A	45,8	181,0	0,0
F	178,7	189,6	110,9

Przyjęte zadanie transportowe o wymiarach 7×7 jest przedstawione na rys. 9.4. Znalezione rozwiązania za pomocą algorytmu GENETIC-2 z parametrami p_c i p_m znajdują się na rys. 9.5.

$p_c = 0\%$, $p_m = 25\%$, funkcja A, koszt = 45,8

20.00	0.00	0.00	1.00	2.00	2.00	2.00
0.00	20.00	1.00	2.00	2.00	2.00	1.00
0.00	0.00	19.00	0.00	2.00	1.00	3.00
0.00	0.00	0.00	20.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	20.00

$p_c = 0\%$, $p_m = 25\%$, funkcja F, koszt = 178,7

15.00	6.00	0.00	6.00	0.00	0.00	0.00
0.00	14.00	0.00	14.00	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
5.00	0.00	0.00	3.00	6.00	0.00	6.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	20.00	0.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

$p_c = 25\%$, $p_m = 0\%$, funkcja A, koszt = 181,2

18.25	0.00	0.11	1.81	3.30	3.53	0.00
0.00	18.21	3.94	3.05	0.00	1.97	0.82
1.75	1.79	13.24	1.46	1.46	1.48	3.83
0.00	0.00	1.91	14.87	1.18	0.35	1.69
0.00	0.00	0.72	0.97	18.10	0.21	0.00
0.00	0.00	0.02	0.71	1.96	17.30	0.00
0.00	0.00	0.05	0.14	0.00	0.16	19.65

$p_c = 25\%$, $p_m = 0\%$, funkcja F, koszt = 189,6

20.00	0.25	0.00	0.75	0.00	6.00	0.00
0.00	5.75	0.00	22.25	0.00	0.00	0.00
0.00	0.00	20.00	0.00	0.00	0.00	5.00
0.00	14.00	0.00	0.00	6.00	0.00	0.00
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	14.00	6.00
0.00	0.00	0.00	0.00	0.00	5.00	15.00

$p_c = 5\%$, $p_m = 20\%$, funkcja A, koszt = 0,0

19.87	0.00	0.68	1.80	1.33	1.80	1.51
0.08	20.00	1.00	1.90	1.48	1.61	1.92
0.00	0.00	18.32	1.89	1.08	1.78	1.93
0.05	0.00	0.00	17.09	1.91	0.96	0.00
0.00	0.00	0.00	0.00	19.92	0.00	0.08
0.00	0.00	0.00	0.00	0.00	18.60	1.40
0.00	0.00	0.00	0.31	0.28	0.25	19.16

$p_c = 5\%$, $p_m = 20\%$, funkcja F, koszt = 110,9

14.31	6.31	6.39	0.00	0.00	0.00	0.00
0.00	13.69	0.31	14.00	0.00	0.00	0.00
0.00	0.00	13.31	6.00	0.00	0.00	5.69
5.69	0.00	0.00	3.00	6.00	0.00	5.31
0.00	0.00	0.00	0.00	20.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	19.31	0.69
0.00	0.00	0.00	0.00	0.00	5.69	14.31

Rys. 9.5. Rozwiązania wyznaczone za pomocą GENETIC-2 dla różnych wartości p_c i p_m

Obliczenia za pomocą systemu GENETIC-2 wykonywano na stacji roboczej SUN SPARC 1, a GAMS na komputerze Olivetti 386. Chociaż porównanie szybkości obu tych komputerów jest trudne, trzeba powiedzieć, że na ogół GAMS kończył każde obliczenie znacznie wcześniej niż system genetyczny. Wyjątkiem był przypadek A (w którym GAMS oblicza liczne funkcje arctg), gdzie algorytmowi genetycznemu obliczenie zajęło nie więcej niż 15 minut, GAMS zaś potrzebował średnio dwa razy tyle. Dla przypadków A, B i D, w których dodatkowy parametr modyfikacji oznaczał, że GAMS musi wykonać wielokrotne obliczenia dla znalezienia najlepszego rozwiązania, system genetyczny był w sumie też znacznie szybszy.

Typowe porównanie optimów dla jednego zadania o wymiarze 10×10 wyznaczonych przez GENETIC-2 (uśrednionych po 5 obliczeniach z różnymi liczbami losowymi) oraz GAMS przedstawiono w tabl. 9.4, a ich opis na rys. 9.6.

Tablica 9.4. Porównanie GAMS i GENETIC-2

Funkcja	GAMS	GENETIC-2	% różnicy
A	281,0	202,0	– 28,1%
B	180,8	163,0	– 9,8%
C	4402,0	4556,2	+ 3,5%
D	408,4	391,1	– 4,2%
E	145,1	79,2	– 45,4%
F	1200,8	201,9	– 83,2%

Liczba punktów nadania: 10

Liczba punktów odbioru: 10

Wywozy z punktów nadania:	8	8	2	26	12	1	6	18	18	1
Przywozy do punktów odbioru:	19	2	33	5	11	11	2	14	2	1

Macierz parametrów łuków (punkty nadania na punkty odbioru):

15	3	23	1	19	14	6	16	41	33
13	17	30	36	20	17	26	19	3	33
37	17	30	5	48	27	8	25	36	21
13	13	31	7	35	11	20	41	34	3
31	24	8	30	28	33	2	8	1	8
32	36	12	9	18	1	44	49	11	11
49	6	17	0	42	45	22	9	10	47
2	21	18	40	47	27	27	40	19	42
13	16	25	21	19	0	32	20	32	35
23	42	2	0	9	30	5	29	31	29

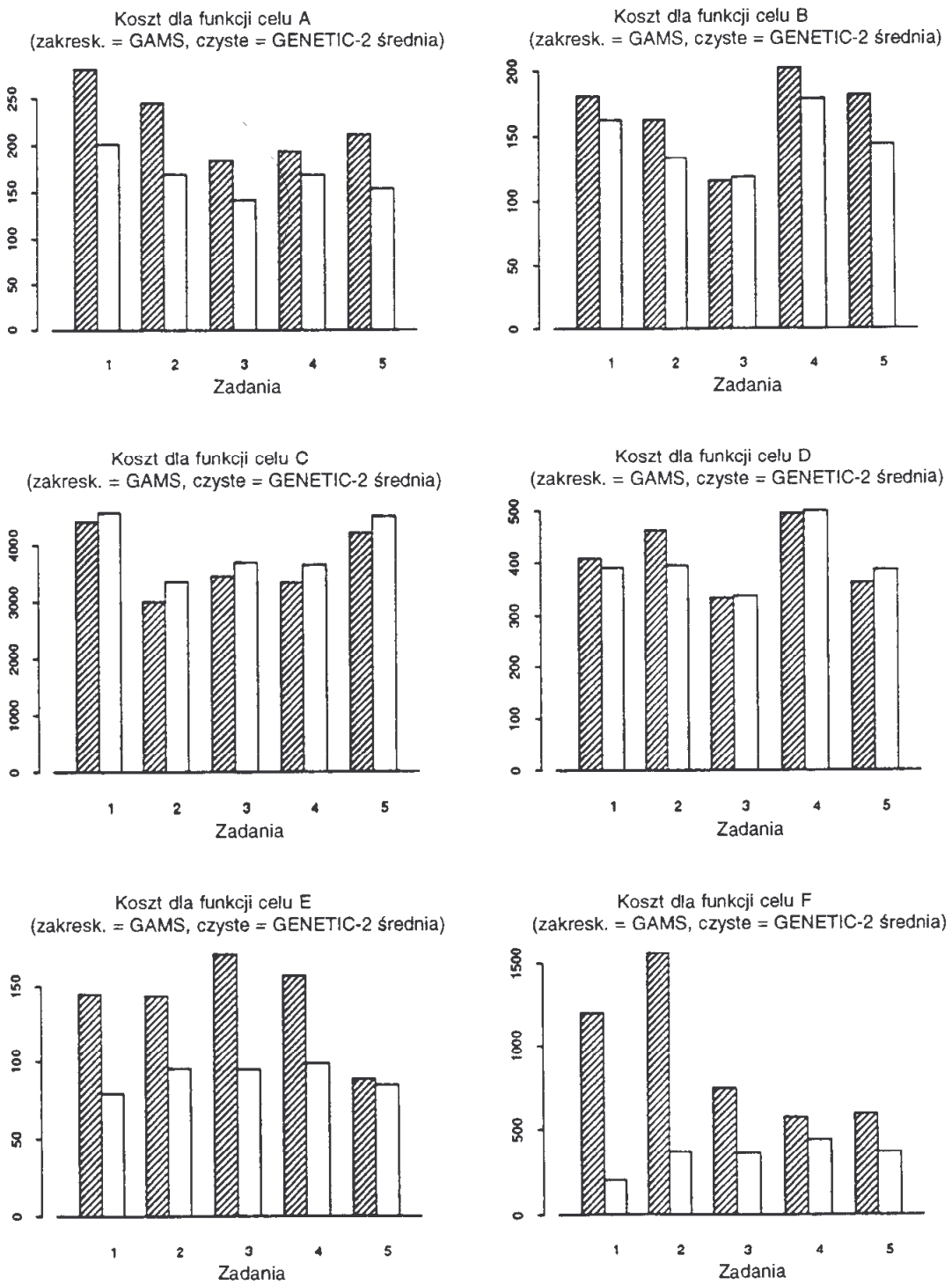
Rys. 9.6. Opis zadania testowego

Na rysunku 9.7 przedstawiano wyniki z wszystkich pięciu rozważanych zadań. W klasie zadań „praktycznych" A i B GENETIC-2 jest średnio lepszy niż GAMS o 24,5% w przypadku A i 11,5% w przypadku B. Dla „rozsądnych" funkcji wyniki były inne. W przypadku C (funkcja kwadratowa) system genetyczny wypadł gorzej o 7,5%, a w przypadku D (funkcja pierwiastkowa) system genetyczny był lepszy średnio tylko o 2,0%. Dla „innych" funkcji, E i F, system genetyczny przeważa. Miał on lepsze wyniki od GAMS o 33,0% i 54,5% przy średniej z pięciu zadań.

9.2.8. Wnioski

Naszym celem było zbadanie zachowania się pewnego typu algorytmu genetycznego w zadaniach z wieloma ograniczeniami i nieliniową funkcją celu. Do badania wybrano zadanie transportowe, gdyż charakteryzuje się ono stosunkowo prostym wypukłym zbiorem rozwiązań dopuszczalnych. Umożliwiło to łatwiejsze utrzymanie dopuszczalności rozwiązań. Można więc było badać jedynie wpływ funkcji celu na zachowanie się algorytmu.

Wyniki wskazały na efektywność metod genetycznych w wyznaczaniu globalnego optimum w trudnych zadaniach, chociaż GAMS działał też dobrze dla gładkich, monotonicznych funkcji „rozsądnych". Metody gradientowe najbardziej się nadają w takich sytuacjach. Dla funkcji C system GAMS wyznaczył lepsze rozwiązania dużo szybciej niż GENETIC-2.



Rys. 9.7. Wyniki

W zadaniach „praktycznych" metody gradientowe mają trudności „z zajrzeniem za róg" do nowych obszarów o lepszym koszcie. Algorytm typu genetycznego, o charakterze globalnym, może się przenieść łatwo do nowego obszaru, wyznaczając w ten sposób lepsze rozwiązania.

„Inne" zadania, chociaż także gładkie, mają istotne cechy strukturalne, które wybrano tak, aby powodować znaczne trudności dla metod gradiento-

wych. GENETIC-2 przewyższał tu GAMS nawet bardziej niż w zadaniach „praktycznych".

Ciekawe jest także porównanie GENOCOP-u (rozdz. 7) z GENETIC-2 (patrz tabl. 9.5). Na ogół ich wyniki są bardzo podobne. Jednak zauważmy znowu, że podejście macierzowe było przystosowane do specyficznego zadania (transportowego), gdy tymczasem GENOCOP nie jest uzależniony od zadania i działa bez wbudowanej wiedzy o dziedzinie. Inaczej mówiąc, o ile można spodziewać się, że GENOCOP będzie działał podobnie dobrze dla innych zadań z ograniczeniami, o tyle GENETIC-2 nie może w nich być w ogóle użyty.

Tablica 9.5. Porównanie GENETIC-2 i GENETIC-3

Funkcja	GENETIC-2	GENOCOP	% różnicy
A	00,0	24,15	
B	203,81	205,60	0,87%
C	2564,23	2571,04	0,26%
D	480,16	480,16	0,00%
E	204,73	204,82	0,04%
F	110,94	119,61	7,24%

Wyniki dla zadania o wymiarze 7 × 7 z funkcjami kosztu transportu A-F i macierzą kosztów podaną na rys. 7.3.

Przy porównywaniu trzech systemów (GAMS, GENOCOP, GENETIC-2) ważne jest, że dwa z nich, GAMS i GENOCOP, nie zależą od zadania. Mogą one optymalizować dowolną funkcję przy dowolnym zbiorze ograniczeń liniowych. Trzeci system, GENETIC-2, stworzono tylko dla zadań transportowych. Poszczególne ograniczenia są uwzględniane w strukturach danych macierzowych i specjalnych operatorach „genetycznych" (dalsze porównanie w rozdz. 14).

GENETIC-2 specjalnie przystosowano do zadań transportowych, ale jego istotną cechą jest to, że uwzględnia on wszystkie typy kosztów (które nie muszą być nawet ciągłe). Można go także zmodyfikować, aby objął wiele innych zadań z zakresu badań operacyjnych, włącznie z zadaniami alokacji i szeregowania. Wygląda to na obiecujący kierunek badań, który może zakończyć się opracowaniem ogólnej metody rozwiązywania macierzowych zadań optymalizacji z ograniczeniami.

10

Zadanie komiwojażera

Nie masz dla człowieka gorszego zła
nad włóczęgostwo.

Homer, *Odyseja*[1)]

W następnym rozdziale przedstawimy kilka przykładów programów ewolucyjnych przystosowanych do specyficznych zastosowań (rysowanie grafów, podział, szeregowanie). Zadanie komiwojażera jest właśnie jednym z takich zastosowań. Jednak traktujemy je jako oddzielne zadanie – matkę wszystkich zadań – i omawiamy w oddzielnym rozdziale. Jakie są tego przyczyny?

Jest ich wiele. Przede wszystkim zadanie komiwojażera jest koncepcyjnie bardzo proste: komiwojażer musi odwiedzić każde miasto na swoim terytorium dokładnie raz i wrócić do punktu startowego. Jak powinien on zaplanować swoją trasę, znając koszt przejazdu między miastami, aby zminimalizować całkowity koszt objazdu? Przestrzeń rozwiązań w zadaniu komiwojażera jest zbiorem permutacji n miast. Każda permutacja n miast jest rozwiązaniem (które jest pełnym objazdem n miast). Optymalnym rozwiązaniem jest permutacja, dla której koszt objazdu jest najmniejszy. Rozmiar przestrzeni rozwiązań wynosi $n!$.

Zadanie komiwojażera sformułowano dosyć dawno. Było ono zapisane już w 1759 r. przez Eulera (chociaż nie pod tą nazwą), który zajmował się rozwiązywaniem zadania ruchu konika po szachownicy. Prawidłowe rozwiązanie wymagało, aby konik odwiedził każde z 64 pól na szachownicy dokładnie raz w czasie całego ruchu.

Termin „komiwojażer" był po raz pierwszy użyty w 1932 r. w książce niemieckiej *Komiwojażer, jak i co powinien on robić, aby wykonać zlecenia i mieć powodzenie w interesach*, napisanej przez byłego komiwojażera (patrz [236]). Chociaż nie było to głównym tematem książki, zadanie komiwojażera i szeregowanie są omawiane w ostatnim jej rozdziale.

Zadanie komiwojażera w obecnym sformułowaniu wprowadziła RAND Corporation w 1948 r. Jej reputacja pomogła w tym, że stało się ono zadaniem

[1)] Przekład Jan Parandowski, wyd. Czytelnik, 1981 (przyp. tłum.).

znanym i popularnym. Zadanie komiwojażera także stało się popularne w tym czasie z powodu opracowania nowej metody programowania liniowego i prób rozwiązywania zadań kombinatorycznych.

Udowodniono, że zadanie komiwojażera jest NP-trudne [134]. Pojawia się ono w licznych zastosowaniach i liczba miast może być całkiem znaczna. Jak podano w [207]:

> W [246] wspomina się o zastosowaniu do wiercenia otworów w płytkach drukowanych z liczbą miast do 17 000. W [44] mówi się o przykładach z krystalografii rentgenowskiej z liczbą miast do 14 000, a twierdzi się, że przykłady związane z wytwarzaniem VLSI wymagają aż 1,2 miliona miast [227]. Ponadto 5 godzin liczenia na komputerze za wiele milionów dolarów, aby znaleźć optymalne rozwiązanie, może nie być najlepsze pod względem kosztu, jeżeli można uzyskać gorsze od niego o kilka procent w sekundy na PC. Z tego powodu wynika potrzeba metod heurystycznych.

W ciągu ostatnich kilku dziesięcioleci powstało kilka algorytmów wyznaczania rozwiązania bliskiego optymalnemu: algorytm najbliższego sąsiada, algorytm zachłanny, algorytm najbliższego wstawiania, najdalszego wstawiania, podwajanego najkrótszego drzewa rozpinającego, oddzielania powłoki wypukłej, krzywej wypełniającej przestrzeń, algorytmy Karpa, Litkego, Christofidesa itp. [207] (w niektórych z tych algorytmów zakłada się, że miasta odpowiadają punktom na płaszczyźnie przy pewnej standardowej metryce). Inna grupa algorytmów (2-opt, 3-opt, Lina-Kernighana) szuka lokalnych minimów – poprawy trasy przez lokalne przestawienia. Zadanie komiwojażera stało się także celem do rozwiązania w dziedzinie algorytmów genetycznych. Opisano kilka algorytmów genetycznych [131], [160], [168], [170], [206], [241], [288], [299], [353], [370], [375], [389], [402]. Mają one na celu wyszukanie rozwiązania bliskiego optymalnemu za pomocą populacji potencjalnych rozwiązań, które podlegają pewnym jednoargumentowym i binarnym przekształceniom („mutacjom" i „krzyżowaniom") za pomocą schematów selekcji biorących pod uwagę dopasowanie osobników. Ciekawe jest porównanie tych podejść, ze zwróceniem szczególnej uwagi na zastosowane reprezentacje i operatory genetyczne. Jest to celem tego rozdziału. Inaczej mówiąc, będziemy śledzić rozwój programów ewolucyjnych dla zadania komiwojażera.

Aby podkreślić pewne ważne cechy zadania komiwojażera, rozważmy najpierw zadanie spełniania postaci normalnej koniunkcyjnej. Wyrażenie logiczne w postaci normalnej koniunkcyjnej jest ciągiem zdań logicznych rozdzielonych operatorem boolowskim $\wedge$, zdanie jest ciągiem literałów oddzielonych operatorem boolowskim $\vee$, literał jest zmienną logiczną lub jej zaprzeczeniem, zmienna logiczna jest zmienną, która może przyjmować wartość PRAWDA lub FAŁSZ (1 lub 0).

Na przykład, następujące wyrażenie logiczne jest w postaci normalnej koniunkcyjnej:

$$(a \vee \bar{b} \vee c) \wedge (b \vee c \vee d \vee \bar{e}) \wedge (\bar{a} \vee c) \wedge (a \vee \bar{c} \vee \bar{e})$$

gdzie a, b, c, d i e są zmiennymi logicznymi, $\bar{a}$ określa zaprzeczenie zmiennej a ($\bar{a}$ ma wartość PRAWDA wtedy i tylko wtedy, gdy a ma wartość FAŁSZ).

Zadanie polega na określeniu, czy można uzyskać wartość całego wyrażenia PRAWDA przy przyporządkowaniu wartości PRAWDA pewnym zmiennym. Na przykład powyższe wyrażenie w postaci normalnej koniunkcyjnej ma kilka takich przyporządkowań, przy których całe wyrażenie przyjmuje wartość PRAWDA. Jest to chociażby przyporządkowanie z a = PRAWDA i c = PRAWDA.

Jeżeli spróbujemy zastosować algorytm genetyczny w zadaniu spełniania postaci normalnej koniunkcyjnej, to zauważymy, że trudno sobie wyobrazić zadanie z bardziej odpowiednią reprezentacją: wystarczy przyjąć wektor binarny o ustalonej długości (długość wektora odpowiada liczbie zmiennych). Ponadto nie ma zależności między bitami i jakakolwiek zmiana prowadzi do wektora dopuszczalnego. Możemy więc stosować mutacje i krzyżowania bez potrzeby dekodowania lub naprawy. Najtrudniejszym zadaniem jest wybór funkcji oceny. Zauważmy, że wyrażenia logiczne przyjmują wartości PRAWDA lub FAŁSZ i jeżeli jakieś konkretne przyporządkowanie spowoduje, że wyrażenie przyjmie wartość PRAWDA, to rozwiązanie będzie znalezione. Kłopot polega na tym, że podczas poszukiwania rozwiązania wszystkie chromosomy (wektory) w populacji mają ocenę FAŁSZ (dopóki nie znajdzie się rozwiązania), więc nie można rozróżnić „dobrych" chromosomów od „złych". W skrócie zadanie spełniania postaci normalnej koniunkcyjnej ma naturalną reprezentację bez naturalnej funkcji oceny. Szersze omówienie problemów związanych z wyborem odpowiedniej funkcji oceny można znaleźć w [90].

Natomiast zadanie komiwojażera ma wyjątkowo łatwą (i naturalną) funkcję oceny. Dla każdego potencjalnego rozwiązania (permutacji miast) możemy odwołać się do tablicy z odległościami między wszystkimi miastami i (po $n-1$ dodatkowych operacjach) możemy uzyskać całkowitą długość trasy. Tak więc w populacji tras możemy łatwo porównać każde dwie z nich. Jednak wybór reprezentacji trasy i użytych operatorów jest daleki od oczywistego.

Dodatkowym powodem omawiania zadania komiwojażera w oddzielnym rozdziale jest to, że podobnych metod używano w innych zadaniach szeregowania, jak harmonogramowanie i podział. Niektóre z nich są omawiane w następnym rozdziale.

Wśród badaczy algorytmów genetycznych panuje przekonanie, że reprezentacja binarna tras nie jest zbyt odpowiednia dla zadania komiwojażera. Nie jest trudno zauważyć dlaczego. W końcu chcemy znaleźć najlepszą permutację miast, to znaczy

$$(i_1, i_2, \ldots, i_n)$$

gdzie $(i_1, i_2, \ldots, i_n)$ jest permutacją ciągu $\{1, 2, \ldots, n\}$. Kodowanie binarne tych miast nie ma żadnych zalet. Wręcz przeciwnie, reprezentacja binarna wymaga-

łaby specjalnych algorytmów naprawy, gdyż zmiana jednego bitu może prowadzić do trasy niedopuszczalnej. Jak zauważono w [402]:

> Niestety, nie można praktycznie zakodować trasy w zadaniu komiwojażera jako łańcucha binarnego, w którym nie ma zależności od ustawienia lub w którym można w rozsądny sposób używać operatorów. Proste krzyżowanie łańcuchów powoduje podwójne wystąpienia miast lub ich brak. Aby więc rozwiązać to zadanie, trzeba użyć jakiejś odmiany standardowego krzyżowania genetycznego. Idealny operator kombinacyjny powinien wydobyć najważniejszą informację ze struktur rodziców w sposób niedestrukcyjny, zachowując ich znaczenie.

Warto zauważyć, że w ostatnim artykule Lidda [241] opisano algorytm genetyczny dla zadania komiwojażera z reprezentacja binarną i klasycznymi operatorami (krzyżowaniem i mutacją). Niedopuszczalne trasy są oceniane na podstawie pełnych (niekoniecznie dopuszczalnych) tras tworzonych przez algorytm zachłanny. Przedstawione wyniki są zadziwiająco dobre, jednak największe rozważane zadanie testowe zawiera tylko 100 miast.

W ciągu ostatnich kilku lat rozważano trzy reprezentacje wektorowe dla zadania komiwojażera: *przyległościową*, *porządkową* i *ścieżkową*. Każda z tych reprezentacji ma wlasne operatory „genetyczne". Omówimy je po kolei. Ponieważ jest stosunkowo latwo utworzyć pewien rodzaj operatora mutacji, który wprowadza male zmiany w trasie, skoncentrujemy się na operatorach krzyżowania. We wszystkich trzech reprezentacjach trasa oznacza ciąg miast. W poniższym omówieniu użyjemy wspólnego przykładu o 9 miastach ponumerowanych od 1 do 9.

Reprezentacja przyległościowa

W reprezentacji przyległościowej trasa jest reprezentowana jako ciąg n miast. Miasto j znajduje się na pozycji i wtedy i tylko wtedy, jeżeli trasa wiedzie z miasta i do miasta j. Na przykład wektor

(2 4 8 3 9 7 1 5 6)

przedstawia następującą trasę:

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

Każda trasa ma tylko jedną reprezentację przyległościową, jednak pewne reprezentacje przyległościowe mogą przedstawiać trasy niedopuszczalne, na przykład

(2 4 8 1 9 3 5 7 6)

czemu odpowiada

1 – 2 – 4 – 1

co jest (częściową) trasą z (niepełnym) cyklem.

W reprezentacji przyległościowej nie można użyć wprost klasycznego operatora krzyżowania. Może być w niej potrzebny algorytm naprawy. Określono i badano trzy operatory krzyżowania dla reprezentacji przyległościowej: *z wymianą krawędzi*, *z wydzielaniem podtras* i *heurystyczny* [168].

- W krzyżowaniu z wymianą krawędzi tworzy się potomka, wybierając (losowo) krawędź pierwszego rodzica, następnie wybierając odpowiednią krawędź drugiego rodzica itd. Operator rozszerza trasę przez dołączanie krawędzi na zmianę od rodziców. Jeżeli nowa krawędź (od jednego z rodziców) wprowadza cykl do bieżącej (nadal częściowej) trasy, to operator wybiera w zamian (losowo) krawędź z pozostałych krawędzi, taką która nie wprowadza cyklu. Na przykład pierwszym potomkiem z dwóch rodziców

  $p_1 = (2\ 3\ 8\ 7\ 9\ 1\ 4\ 5\ 6)$

  $p_2 = (7\ 5\ 1\ 6\ 9\ 2\ 8\ 4\ 3)$

  może być

  $o_1 = (2\ 5\ 8\ 7\ 9\ 1\ 6\ 4\ 3)$

  przy czym proces zaczął się od krawędzi (1, 2) z rodzica p_1, a jedyną losową krawędzią wprowadzoną w czasie procesu naprzemiennego dołączania krawędzi była (7,6) zamiast (7, 8), która wprowadziłaby niepełny cykl.
- W krzyżowaniu z wydzielaniem podtras tworzy się potomka, wybierając podtrasę o losowej długości od jednego z rodziców, a następnie od drugiego itd. – operator rozszerza trasę, wybierając krawędzie na zmianę od rodziców. I ponownie, gdy pewna krawędź (od któregoś z rodziców) wprowadza cykl do bieżącej (nadal częściowej) trasy, wówczas operator wybiera zamiast niej (losową) krawędź spośród nie wybranych jeszcze krawędzi tak, żeby nie doprowadzić do zamknięcia cyklu.
- W krzyżowaniu heurystycznym tworzy się potomka, wybierając przypadkowe miasto jako punkt startowy dla trasy potomka. Następnie porównuje się dwie krawędzie (z obu rodziców) wychodzące z tego miasta i wybiera krawędź lepszą (krótszą). Miasto po drugiej stronie wybranej krawędzi jest następnie punktem startowym do wyboru krótszej z krawędzi wychodzącej z tego miasta itd. Jeżeli na pewnym etapie nowa krawędź wprowadza cykl w trasie potomka, to dołącza się zamiast niej losowo krawędź spośród dotąd nie wybranych, aby nie doprowadzić do zamknięcia cyklu.

W [206] autorzy zmodyfikowali powyższe krzyżowanie heurystyczne, zmieniając dwie reguły: (1) jeżeli krótsza krawędź (od rodzica) wprowadza niepełny cykl w trasie potomka, sprawdź pozostałą (dłuższą) krawędź. Jeżeli dłuższa krawędź nie wprowadza cyklu, to zaakceptuj ją. Jeżeli wprowadza, to (2) wybierz najkrótszą krawędź z zestawu q losowo wybranych krawędzi (q jest parametrem metody).

W wyniku działania tego operatora łączy się krótkie ścieżki wybrane z tras rodziców. Jednak może on doprowadzić do niepożądanego krzyżowania się krawędzi. Dlatego też krzyżowanie heurystyczne nie jest odpowiednie dla dokładnej lokalnej poprawy. Suh i Gucht [375] zaproponowali dodatkowy operator heurystyczny (oparty na algorytmie 2-opt [245]), nadający się do lokalnego poprawiania. Operator ten wybiera losowo dwie krawędzie $(i\ j)$ oraz $(k\ m)$ i sprawdza, czy

$$dist(i, j) + dist(k, m) > dist(i, m) + dist(k, j)$$

gdzie $dist(a, b)$ jest podaną odległością między miastami a i b. Jeżeli tak jest, krawędzie $(i\ j)$ oraz $(k\ m)$ w bieżącej trasie są zamieniane na krawędzie $(i\ m)$ oraz $(k\ j)$.

Zaletą reprezentacji przyległościowej jest to, że umożliwia ona analizę schematów, podobną do omawianej w rozdz. 3, gdzie rozważano łańcuchy binarne. Schematy odpowiadają tu naturalnym blokom budującym, to znaczy krawędziom. Na przykład schemat

(* * * 3 * 7 * * *)

oznacza zbiór wszystkich tras z krawędziami (4 3) oraz (6 7). Jednak główną wadą tej reprezentacji są stosunkowo słabe wyniki stosowania wszystkich operatorów. Krzyżowanie z wymianą krawędzi często rozrywa dobre trasy przez operacje wymiany krawędzi z obu rodziców. Krzyżowanie z wydzielaniem podtras działa lepiej niż krzyżowanie z wymianą krawędzi, gdyż częstość rozrywania tras jest mniejsza. Jednak jego działanie jest nadal słabe. Krzyżowanie heurystyczne jest tu oczywiście najlepszym operatorem. Powodem jest to, że pierwsze dwa operatory są ślepe, to znaczy nie biorą pod uwagę rzeczywistej długości krawędzi. Natomiast krzyżowanie heurystyczne wybiera lepszą z dwóch możliwych krawędzi. Jest to powód, dla którego działa ono lepiej niż dwa pozostałe. Jednak działanie krzyżowania heurystycznego nie jest szczególnie dobre. W trzech przedstawionych obliczeniach [168] z 50, 100 i 200 miastami system znalazł trasy odległe o 25%, 16% i 27% od optimum odpowiednio w około 15 000, 20 000 i 25 000 pokoleń.

Reprezentacja porządkowa

W reprezentacji porządkowej trasa jest reprezentowana jako ciąg n miast, przy czym i-ty element jest liczbą z zakresu od 1 do $n - i + 1$. Idea reprezentacji porządkowej jest następująca. Mamy uporzadkowany ciąg miast C, który jest punktem odniesienia dla ciągów w reprezentacji porządkowej. Przypuśćmy dla przykładu, że taki uporządkowany ciąg (punkt odniesienia) to po prostu

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

Trasa

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

jest reprezentowana przez ciąg wskazań

$$l = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$$

co należy interpretować następująco:

- Pierwszą liczbą w ciągu l jest 1, więc weźmy pierwsze miasto z ciągu C jako pierwsze miasto trasy (miasto o numerze 1) i usuńmy je z C. Częściową trasą jest więc

  1

- Następną liczbą w ciągu l jest także 1, więc weźmy pierwsze miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 2) i usuńmy je z C. Częściową trasą jest

  1 – 2

- Następną liczbą w ciągu l jest 2, więc weźmy drugie miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 4) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4

- Następną liczbą w ciągu l jest 1, więc weźmy pierwsze miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 3) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3

- Następną liczbą w ciągu l jest 4, więc weźmy czwarte miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 8) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3 – 8

- Następną liczbą w ciągu l jest znowu 1, więc weźmy pierwsze miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 5) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3 – 8 – 5

- Następną liczbą w ciągu l jest 3, więc weźmy trzecie miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 9) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3 – 8 – 5 – 9

- Następną liczbą w ciągu l jest 1, więc weźmy pierwsze miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 6) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3 – 8 – 5 – 9 – 6

- Następną liczbą w ciągu l jest 1, więc weźmy pierwsze miasto z bieżącego ciągu C jako następne miasto trasy (miasto numer 7, ostatnie dostępne) i usuńmy je z C. Częściową trasą jest

  1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7

Główną zaletą reprezentacji porządkowej jest to, że działa przy niej klasyczne krzyżowanie. Jakiekolwiek dwie trasy w reprezentacji porządkowej, przecięte w pewnym punkcie i zamienione ze sobą, prowadzą do dwóch potomków, przy czym obaj z nich są trasami dopuszczalnymi. Na przykład dwóch rodziców

$p_1 = (1\ 1\ 2\ 1\ |\ 4\ 1\ 3\ 1\ 1)$ oraz

$p_2 = (5\ 1\ 5\ 5\ |\ 5\ 3\ 3\ 2\ 1)$

odpowiadających trasom

1 – 2 – 4 – 3 – 8 – 5 – 9 – 6 – 7 oraz

5 – 1 – 7 – 8 – 9 – 4 – 6 – 3 – 2

z punktem cięcia oznaczonym przez „|" utworzy następujących potomków:

$o_1 = (1\ 1\ 2\ 1\ 5\ 3\ 3\ 2\ 1)$ oraz

$o_2 = (5\ 1\ 5\ 5\ 4\ 1\ 3\ 1\ 1)$

którzy odpowiadają trasom

1 – 2 – 4 – 3 – 9 – 7 – 8 – 6 – 5 oraz

5 – 1 – 7 – 8 – 6 – 2 – 9 – 3 – 4

Łatwo zauważyć, że częściowe trasy po lewej stronie punktu cięcia nie zmieniły się, gdy tymczasem częściowe trasy po prawej stronie punktu cięcia są porozrywane w sposób losowy. Słabe wyniki eksperymentalne [168] wskazują na to, że ta reprezentacja z klasycznym krzyżowaniem nie jest odpowiednia dla zadania komiwojażera.

Reprezentacja ścieżkowa

Reprezentacja ścieżkowa jest chyba najbardziej naturalną reprezentacją trasy. Na przykład trasa

5 – 1 – 7 – 8 – 9 – 4 – 6 – 2 – 3

jest w niej po prostu reprezentowana przez

(5 1 7 8 9 4 6 2 3)

Aż do niedawna określano dla reprezentacji ścieżkowej trzy operatory krzyżowania: krzyżowanie *z częściowym odwzorowaniem* (PMX), *z porządkowaniem* (OX) oraz *cykliczne* (CX). Omówimy je po kolei.

- W PMX, zaproponowanym przez Goldberga i Lingle'a [160], tworzy się potomka, wybierając podtrasę od jednego rodzica i pozostawiając porządek i pozycje tak wielu miast drugiego rodzica, jak tylko jest to możliwe. Podtrasę wybiera się przez dwa losowe punkty cięcia, które służą jako granice dla operatorów wymieniających. Dla przykładu z dwóch rodziców (z dwoma punktami cięcia oznaczonymi przez „|")

  $p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$ oraz

  $p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$

  otrzymuje się dwóch potomków w następujący sposób. Po pierwsze odcinki między punktami cięcia są wymieniane (symbol „x" należy interpretować jako „obecnie nie znane")

  $o_1 = (\mathrm{x}\ \mathrm{x}\ \mathrm{x} \mid 1\ 8\ 7\ 6 \mid \mathrm{x}\ \mathrm{x})$ oraz

  $o_2 = (\mathrm{x}\ \mathrm{x}\ \mathrm{x} \mid 4\ 5\ 6\ 7 \mid \mathrm{x}\ \mathrm{x})$

  Ta wymiana określa też ciąg odwzorowań

  $1 \leftrightarrow 4,\quad 8 \leftrightarrow 5,\quad 7 \leftrightarrow 6$ oraz $6 \leftrightarrow 7$

  Wobec tego możemy wstawić dalsze miasta (od początkowych rodziców), dla których nie zachodzi konflikt:

  $o_1 = (\mathrm{x}\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid \mathrm{x}\ 9)$ oraz

  $o_2 = (\mathrm{x}\ \mathrm{x}\ 2 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$

  Na koniec pierwsze x w potomku o_1 (które powinno być 1, ale wystąpił konflikt) zamienia się na 4 ze względu na odwzorowanie $1 \leftrightarrow 4$. Podobnie drugie x w potomku o_1 zamienia się na 5, a x i x w potomku o_2 stają się 1 i 8. Potomkowie wyglądają więc następująco:

  $o_1 = (4\ 2\ 3 \mid 1\ 8\ 7\ 6 \mid 5\ 9)$ oraz

  $o_2 = (1\ 8\ 2 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$

  Krzyżowanie PMX wykorzystuje jednocześnie ważne podobieństwa w wartościach i uporządkowaniu, gdy używa się go łącznie z odpowiednim planem reprodukcji [160].
- W OX, zaproponowanym przez Davisa [71], tworzy się potomka, wybierając podtrasę od jednego rodzica i pozostawiając wzajemne uporządkowanie miast z drugiego rodzica. Na przykład z dwóch rodziców (z dwoma punktami cięcia oznaczonymi przez „|")

  $p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9)$ oraz

  $p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$

tworzy się potomków w następujący sposób. Po pierwsze odcinki między punktami cięć kopiuje się do potomków

$o_1 = (x\ x\ x \mid 4\ 5\ 6\ 7 \mid x\ x)$ oraz

$o_2 = (x\ x\ x \mid 1\ 8\ 7\ 6 \mid x\ x)$

Następnie, zaczynając od drugiego punktu cięcia jednego z rodziców, kopiuje się miasta z drugiego rodzica w tym samym porządku, omijając jedynie symbole już obecne w trasie. Po osiągnięciu końca łańcucha zaczynamy od pierwszego miejsca łańcucha. Ciąg miast drugiego rodzica (od drugiego punktu cięcia) jest następujący:

9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6

Po usunięciu miast 4, 5, 6 i 7, które znajdują się już w pierwszym potomku, dostajemy

9 – 3 – 2 – 1 – 8

Ten ciąg umieszcza się w pierwszym potomku (zaczynając od drugiego punktu cięcia)

$o_1 = (2\ 1\ 8 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$

Podobnie uzyskujemy drugiego potomka

$o_2 = (3\ 4\ 5 \mid 1\ 8\ 7\ 6 \mid 9\ 2)$

W krzyżowaniu OX korzysta się z tego, że w reprezentacji ścieżkowej ważne jest uporządkowanie miast (a nie ich pozycja), to znaczy dwie trasy

9 – 3 – 4 – 5 – 2 – 1 – 8 – 7 – 6 oraz

4 – 5 – 2 – 1 – 8 – 7 – 6 – 9 – 3

są faktycznie jednakowe.

- W CX, zaproponowanym przez Olivera [299], tworzy się potomków w taki sposób, że każde miasto (i jego pozycja) pochodzi od jednego z rodziców. Wyjaśnimy mechanizm krzyżowania cyklicznego na następującym przykładzie. Z dwóch rodziców

$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ oraz

$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

tworzy się pierwszego potomka, biorąc pierwsze miasto pierwszego rodzica

$o_1 = (1\ x\ x\ x\ x\ x\ x\ x\ x)$

Ponieważ każde miasto w potomku powinno być wzięte od jednego ze swoich rodziców (z tej samej pozycji), nie mamy teraz żadnego wyboru. Następ-

nym miastem musi być miasto 4, jako miasto od rodzica p_2 występujące zaraz za wybranym miastem 1. W p_1 miasto to jest na pozycji „4", więc

$o_1 = (1\ x\ x\ 4\ x\ x\ x\ x\ x)$

To z kolei wskazuje na miasto 8, jako miasto z rodzica p_2 występujące zaraz za wybranym miastem 4. Więc

$o_1 = (1\ x\ x\ 4\ x\ x\ x\ 8\ x)$

Trzymając się dalej tej reguły, następnymi miastami, które należy wstawić do pierszego potomka, będą 3 i 2. Zauważmy jednak, że wybór miasta 2 wymaga wyboru miasta 1, które już jest w potomku. W ten sposób skompletowaliśmy cykl

$o_1 = (1\ 2\ 3\ 4\ x\ x\ x\ 8\ x)$

Pozostałe miasta wstawiamy od 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 u rodziców.

Można określić inne operatory dla reprezentacji ścieżkowej. Na przykład Syswerda [383] użył dwóch zmodyfikowanych wersji operatora krzyżowania z uporządkowaniem. (Jego praca dotyczy zadania harmonogramowania i będzie omówiona w następnym rozdziale). W pierwszej modyfikacji (nazwanej *krzyżowaniem bazującym na porządku*) wybiera się (losowo) kilka miejsc w wektorze i porządek miast w wybranych pozycjach jednego rodzica jest narzucany odpowiednim pozycjom u drugiego rodzica. Rozważmy na przykład dwóch rodziców

$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ oraz

$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

Przypuśćmy, że wybranymi pozycjami są 3, 4, 6 i 9. To uporządkowanie miast z rodzica p_2 będzie narzucone rodzicowi p_1. Miasta na tych pozycjach (w zadanym porządku) w p_2 to 2, 8, 6 i 5. U rodzica p_1 te miasta znajdują się na pozycjach 2, 5, 6 i 8. U potomka elementy na tych pozycjach przestawia się, aby dopasować je do uporządkowania tych samych elementów u p_2 (porządek 2 – 8 – 6 – 5). Pierwszy potomek będzie więc kopią p_1 na wszystkich pozycjach z wyjątkiem pozycji 2, 5, 6 i 8:

$o_1 = (1\ x\ 3\ 4\ x\ x\ 7\ x\ 9)$

Pozostałe elementy wstawia się zgodnie z uporządkowaniem u rodzica p_2, to znaczy 2, 8, 6, 5, skąd

$$o_1 = (1\ 2\ 3\ 4\ 8\ 6\ 7\ 5\ 9)$$

Podobnie można utworzyć drugiego potomka

$$o_2 = (3\ 1\ 2\ 8\ 7\ 4\ 6\ 9\ 5)$$

Druga modyfikacja (zwana *krzyżowaniem bazującym na pozycjach*) jest bardziej podobna do oryginalnego krzyżowania z uporządkowaniem. Jedyną różnicą jest to, że w krzyżowaniu bazującym na pozycjach, zamiast wybierać jedną podtrasę miast do skopiowania, wybiera się (losowo) w tym celu kilka miast.

Warto zauważyć, że te dwa operatory (krzyżowanie bazujące na porządku i krzyżowanie bazujące na pozycjach) są, w pewnym sensie, równoważne. Krzyżowanie bazujące na porządku z pewną liczbą pozycji wybranych jako punkty cięcia i krzyżowanie bazujące na pozycjach z pozostałymi pozycjami jako punktami cięcia zawsze dadzą ten sam wynik. Oznacza to, że jeśli średnia liczba punktów cięcia wynosi $m/2$ (m jest całkowitą liczbą miast), to te dwa operatory powinny działać tak samo. Jednak jeżeli średnia liczba punktów cięcia wynosi, powiedzmy, $m/10$, to te dwa operatory wykazują inne cechy. Więcej informacji o tych dwóch operatorach i pewne wyniki teoretyczne i empiryczne z ich porównania można znaleźć w [154], [131], [299], [370] i [382].

Przeglądając różne operatory zmiany uporządkowania, które pojawiły się w ostatnich kilku latach, musimy wśród nich wymienić także operator inwersji. W zwykłej inwersji [188] wybiera się dwa punkty w chromosomie, tnie się go w tych punkach, a następnie łańcuch między tymi punktami odwraca się ze względu na kolejność. Na przykład z chromosomu

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

z punktami cięcia oznaczonymi przez „|" otrzymuje się

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

Taka prosta inwersja gwarantuje, że wynikowy potomek jest trasą dopuszczalną. Pewne badania teoretyczne [188] wskazują, że operator ten powinien być użyteczny w znajdowaniu dobrego uporządkowania łańcuchów. W [402] podano, że dla zadania komiwojażera z 50 miastami system z inwersją wypadł lepiej niż z operatorem „krzyżuj i koryguj". Jednak zwiększanie liczby punktów cięcia pogarsza działanie systemu. Także inwersja (podobnie jak mutacja) jest operatorem jednoargumentowym, który może tylko wspomagać operatory rekombinacyjne. Nie jest on zdolny do rekombinacji informacji sam z siebie. Badano kilka wersji operatora inwersji [154]. Holland [188] podaje modyfikację twierdzenia o schematach, włączając do niego efekt jego działania.

Należy także wymienić ostatnie próby rozwiązywania zadania komiwojażera za pomocą strategii ewolucyjnych [178], [352]. W jednej z tych prób [178]

eksperymentowano z czterema różnymi operatorami mutacji (mutacja jest nadal podstawowym operatorem w strategiach ewolucyjnych – patrz rozdz. 8):

- *inwersją* – jak opisano powyżej,
- *wstawieniem* – wybiera się miasto i wstawia je w losowym miejscu,
- *przemieszczeniem* – wybiera się podtrasę i wstawia się ją w losowym miejscu,
- *wzajemną wymianą* – wymienia się dwa miasta.

Używano także wersji operatora krzyżowania heurystycznego. W tej modyfikacji w tworzeniu potomka uczestniczy kilku rodziców. Po wyborze pierwszego miasta z trasy potomka (losowo), bada się wszystkich lewych i prawych sąsiadów tego miasta u wszystkich rodziców. Wybiera się to miasto, do którego odległość jest najmniejsza. Proces ten kontynuuje się tak długo, aż utworzy się całą trasę.

W innym zastosowaniu strategii ewolucyjnej [352] generuje się wektor (zmiennopozycyjny) o n składowych (n odpowiada liczbie miast). Następnie stosuje się strategię ewolucyjną, tak jak dla jakiegokolwiek zadania z ciągłymi zmiennymi. Chwyt jest ukryty w kodowaniu. Składowe wektora są sortowane i ich kolejność wyznacza 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 składnikiem wektora $\mathbf{v}$, druga z kolei, $-0{,}12$, jest piątym składnikiem wektora $\mathbf{v}$ itd.

W większości omawianych dotychczas operatorów bierze się pod uwagę miasta (to znaczy ich pozycje i uporządkowanie) w przeciwieństwie do krawędzi – połączeń między miastami. Co może być ważne, to nie tyle dokładna pozycja miasta w trasie, co jego powiązania z innymi miastami. Jak stwierdzili Hamafar i Guan [194]:

> Po starannym rozważeniu problemu możemy twierdzić, że podstawowymi blokami budującymi w zadaniu komiwojażera są krawędzie, a nie pozycje reprezentujące miasta. Miasto lub krótka ścieżka na danej pozycji bez przyległej lub otaczającej informacji ma małe znaczenie dla zbudowania dobrej trasy. Trudno jest stwierdzić, czy wstawienie miasta a na pozycji 2 jest lepsze niż na pozycji 5. Chociaż jest to przypadek ekstremalny, to kryje się za nim założenie, że dobry operator powinien wydobyć od rodziców tak dużo informacji o krawędziach, jak jest to możliwe. To założenie można częściowo wytłumaczyć na podstawie wyników obliczeń podanych w artykule Olivera [299], że OX jest o 11% lepszy niż PMX i o 15% lepszy niż krzyżowanie cykliczne.

Grefenstette [170] określił klasę operatorów heurystycznych, w których kładzie się nacisk na krawędzie. Działają one według następującego schematu:

1) wybierz losowo bieżące miasto potomka,
2) wybierz cztery krawędzie (po dwa od każdego rodzica) związane z bieżącym miastem c,
3) określ rozkład prawdopodobieństwa dla wybranych krawędzi, biorąc pod uwagę związany z nimi koszt; prawdopodobieństwo dla krawędzi związanej z poprzednio odwiedzonym miastem jest równe 0,
4) wybierz krawędź; jeżeli przynajmniej jedna krawędź ma niezerowe prawdopodobieństwo, to wybór następuje na podstawie powyższego rozkładu prawdopodobieństwa; jeżeli nie, to wybór następuje losowo (spomiędzy jeszcze nie odwiedzonych miast),
5) miasto „po drugiej stronie" wybranej krawędzi staje się bieżącym niastem c,
6) jeżeli trasa jest zakończona, to koniec; jeżeli nie, to idź do kroku 2.

Jak podano w [170], taki operator przenosi około 60% krawędzi od rodziców, co oznacza, że 40% jest wybieranych losowo.

Whitley, Starkweather i Fuquay [402] określili nowy operator krzyżowania: krzyżowanie *z rekombinacją krawędzi*, który przenosi ponad 95% krawędzi od rodziców do jednego potomka. Operator z rekombinacją krawędzi bada informację o krawędziach trasy. Na przykład dla trasy

(3 1 2 8 7 4 6 9 5)

krawędzie są następujące: (3 1), (1 2), (2 8), (8 7), (7 4), (4 6), (6 9), (9 5) i (5 3). Zresztą to krawędzie – a nie miasta – są związane z wartościami (odległościami) w zadaniu komiwojażera. Funkcja celu, którą należy zminimalizować, jest całkowitą długością krawędzi dopuszczalnej trasy. Pozycja miasta w trasie nie jest istotna: trasy są zamknięte. Także skierowanie krawędzi nie jest istotne: zarówno krawędź (3 1), jak i (1 3) wskazują tylko, że miasta 1 i 3 są bezpośrednio powiązane.

Ogólna idea leżąca u podstaw krzyżowania z rekombinacją krawędzi to budowanie potomka wyłącznie z krawędzi obecnych u obu rodziców. Robi się to za pomocą listy krawędzi tworzonej z tras obu rodziców. Lista krawędzi podaje, dla każdego miasta c, wszystkie inne miasta połączone z miastem c przynajmniej u jednego z rodziców. Oczywiście, dla każdego miasta c, na liście znajdują się przynajmniej dwa i 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 wygląda następująco:

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

Budowa potomka zaczyna się od wyboru początkowego miasta u jednego z rodziców. W [402] autorzy wybrali jedno z początkowych miast (to znaczy 1 lub 4 w powyższym przykładzie). Następnie wybiera się z listy krawędzi miasto z najmniejszą liczbą krawędzi. Jeżeli takich miast jest więcej, to wybiera się jedno z nich losowo. Taki wybór zwiększa szansę, że ukończymy trasę ze wszystkimi krawędziami wybranymi od rodziców. Przy losowym wyborze szansa błędu, to znaczy pozostania z miastem bez kontynuującej krawędzi, byłaby znacznie wyższa. Przypuśćmy, że wybraliśmy miasto 1. Jest ono bezpośrednio połączone z trzema innymi miastami: 9, 2 i 4. Następne miasto wybieramy spośród tych trzech. W naszym przykładzie miasta 4 i 2 mają trzy krawędzie, a miasto 9 ma ich cztery. Robimy losowy wybór między miastami 4 i 2 – przypuśćmy, że zostało wybrane miasto 4. I ponownie kandydatami na następne miasto w budowanej trasie są 3 i 5, gdyż są one bezpośrednio połączone z ostatnim miastem 4. Tym razem wybieramy miasto 5, gdyż ma ono tylko trzy krawędzie, gdy tymczasem miasto 3 ma cztery. Jak dotąd potomek jest następujący:

(1 4 5 x x x x x x)

Kontynuując tę procedurę, kończymy ją na potomku

(1 4 5 6 7 8 2 3 9)

który składa się wyłącznie z krawędzi wziętych od obu rodziców. W serii obliczeń [402] częstość błędu polegającego na braku krawędzi była mała (1% – 1,5%).

Operator z rekombinacją krawędzi testowano [402] na trzech zadaniach komiwojażera z 30, 50 i 75 miastami. We wszystkich przypadkach otrzymywano rozwiązania lepsze niż poprzednio „najlepsze znane" ciągi.

Dwa lata później krzyżowanie z rekombinacją krawędzi ulepszono [370]. Przyczyną było to, że „wspólne podtrasy" w krzyżowaniu z rekombinacją krawędzi nie zachowywały się. Na przykład, jeżeli lista krawędzi zawiera wiersz z trzema krawędziami:

miasto 4: krawędzie do innych miast: 3 5 1

to jedna z tych krawędzi się powtarza. Jeżeli wrócimy do poprzedniego przykładu, to jest to krawędź (4 5). Znajduje się ona u obu rodziców. Jednak jest ona zapisana tak samo jak inne krawędzie, na przykład (4 3) i (4 1), które znajdują się tylko u jednego rodzica. W zaproponowanym rozwiązaniu [370]

modyfikuje się listę krawędzi przez zapamiętanie dodatkowego wskaźnika przy miastach:

miasto 4: krawędzie do innych miast: 3 −5 1

Znak „−" po prostu oznacza, że zaznaczone miasto 5 powinno znaleźć się na liście dwukrotnie. 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 wygląda następująco:

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 budowy nowego potomka daje priorytet zaznaczonym miastom. Jest to ważne tylko w przypadkach, gdy na liście są trzy miasta, w pozostałych przypadkach albo nie ma zaznaczonych miast, albo oba miasta są zaznaczone. To rozszerzenie (z modyfikacją lepszego wyboru, kiedy potrzebna jest losowa selekcja) jeszcze poprawia działanie systemu [370].

Operator rekombinacji krawędzi wyraźnie wykazuje, że reprezentacja ścieżkowa może być zbyt uboga, aby objąć istotne właściwości trasy. Dlatego też uzupełniono ją o listę krawędzi. Czy inne reprezentacje nadają się lepiej dla zadania komiwojażera? Chyba nie możemy odpowiedzieć zdecydowanie „tak". Jednak warto zbadać inne, ewentualnie niewektorowe reprezentacje.

W czasie ostatnich dwóch lat były przynajmniej trzy niezależne próby skonstruowania programu ewolucyjnego z macierzową reprezentacją chromosomów. Zrobili to Fox i McMahon [131], Seniw [353] oraz Homaifar i Guan [194]. Omówimy je krótko po kolei.

Fox i McMahon [131] przyjęli za reprezentację trasy binarną macierz poprzednictwa M. Element macierzy m_{ij} w wierszu i oraz kolumnie j jest w niej jedynką wtedy i tylko wtedy, gdy miasto i znajduje się na trasie przed miastem j. Na przykład trasa

(3 1 2 8 7 4 6 9 5)

jest reprezentowana przez macierz z rys. 10.1.

	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

Rys. 10.1. Reprezentacja macierzowa trasy

W tej reprezentacji macierz M o wymiarze $n \times n$ reprezentująca trasę (całkowite uporządkowanie miast) ma następujące właściwości:

1) jest w niej dokładnie $n(n-1)/2$ jedynek,
2) $m_{ii} = 0$ dla wszystkich $1 \leqslant i \leqslant n$,
3) jeżeli $m_{ij} = 1$ i $m_{jk} = 1$, to $m_{ik} = 1$.

Jeżeli liczba jedynek w macierzy jest mniejsza niż $n(n-1)/2$ i są spełnione dwa pozostałe warunki, to miasta są częściowo uporządkowane. Oznacza to, że można uzupełnić taką macierz (przynajmniej na jeden sposób), tak aby dostać dopuszczalną trasę (pełne uporządkowanie miast). Jak stwierdzono w [131]:

> Reprezentacja ciągu za pomocą macierzy binarnej obejmuje całą informację o ciągu, łącznie z mikrotopologią połączeń poszczególnych miast ze sobą oraz makrotopologią miast poprzedzających i następujących po sobie. Binarną macierz reprezentacji można użyć do lepszego poznania istniejących operatorów i do opracowania nowych, które będzie można zastosować do ciągów, aby uzyskać pożądane efekty, nie naruszając niezbędnych cech tych ciągów.

Dwa nowe operatory opracowane w [131] to *przecięcie* i *zespolenie*. Oba są operatorami dwuargumentowymi (typu krzyżowania). Tak jak w innych programach ewolucyjnych (na przykład GENETIC-2 dla zadania transportowego, rozdz. 9), takie operatory łączą cechy obu rodziców, zachowując jednocześnie ograniczenia (wymagania).

Operator przecięcia korzysta z tego, że wyodrębnienie części wspólnej bitów w obu macierzach prowadzi do macierzy, w której: (1) liczba jedynek jest nie większa niż $n(n-1)/2$, (2) pozostałe dwa warunki są spełnione. Możemy więc uzupełnić taką macierz, aby uzyskać dopuszczalną trasę (pełne uporządkowanie miast).

Na przykład dwoje rodziców

$$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) \qquad p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$$

jest reprezentowanych przez dwie macierze (rys. 10.2). Przecięcie tych dwóch macierzy daje macierz przedstawioną na rys. 10.3.

	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

Rys. 10.2. Dwoje rodziców

	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

Rys. 10.3. Pierwsza faza operatora przecięcia

Z częściowego uporządkowania wynikającego z przecięcia wynika, że miasto 2 poprzedza miasta 2, 3, 5, 6, 7, 8 i 9; miasto 2 poprzedza miasta 3, 5, 6, 7, 8 i 9; miasto 3 poprzedza miasto 5; miasto 4 poprzedza miasta 5, 6, 7, 8 i 9, a miasta 5, 7 i 8 poprzedzają miasto 9.

W następnym etapie operatora przecięcia wybiera się jednego rodzica i „dodaje się" jedynki (które występują tylko u tego rodzica), uzupełniając macierz do ciągu przez analizę sum wierszy i kolumn. Na przykład macierz z rys. 10.4 może być możliwym wynikiem uzupełnienia w drugim etapie macierzy z rys. 10.3. Reprezentuje ona trasę (1 2 4 8 7 6 3 5 9).

Operator zespolenia korzysta z tego, że podzbiór bitów jednej macierzy można bezpiecznie połączyć z podzbiorem bitów drugiej macierzy, jeżeli mają one puste przecięcie. Operator dzieli zbiór miast na dwie rozłączne grupy (w [131] użyto do tego podziału specjalnej metody). Dla pierwszej grupy miast kopiuje się bity z pierwszej macierzy, a dla drugiej grupy z drugiej macierzy. Na koniec uzupełnia się macierz będącą reprezentacją ciągu przez analizę sumy wierszy i kolumn (jak w operatorze przecięcia).

Na przykład dla dwóch rodziców p_1 i p_2 podział miast na $\{1, 2, 3, 4\}$ i $\{5, 6, 7, 8\}$ daje macierz z rys. 10.5, którą się uzupełnia jak w operatorze przecięcia.

	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

Rys. 10.4. Końcowy wynik 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

Rys. 10.5. Pierwsza faza operatora połączenia

Wyniki eksperymentalne z różnym ustawieniem miast (losowe, pogrupowane, koncentryczne okręgi) wykazały interesujące cechy operatorów zespolenia i przecięcia, które prowadziły do poprawy nawet wtedy, kiedy nie używano opcji elitarystycznej (zachowywania najlepszych). Nie było tak ani w przypadku operatorów z rekombinacją krawędzi, ani operatorów PMX. Solidne porównanie kilku operatorów dwu- i jednoargumentowych (wymiana, pocięcie i inwersja) pod względem wyników i czasu działania oraz ich skomplikowania przedstawiono w [131].

Drugie podejście z zastosowaniem reprezentacji macierzowej przedstawił jeden z moich magistrantów David Seniw [353]. Element macierzy m_{ij} na przecięciu wiersza i oraz kolumny j jest jedynką wtedy i tylko wtedy, gdy trasa wiedzie z miasta i bezpośrednio do miasta j. Oznacza to, że w każdym wierszu i w każdej kolumnie macierzy jest tylko jeden niezerowy element (dla każdego miasta i jest dokładnie jedno miasto, które było odwiedzone przed nim, i dokładnie jedno miasto, które będzie odwiedzone po nim). Na przykład chromosom z rys. 10.6(a) reprezentuje trasę obejmującą miasta (1, 2, 4, 3, 8, 6, 5, 7, 9) odwiedzane w podanej kolejności. Zauważmy także, że w tej reprezentacji unika się problemu z podaniem miasta początkowego, to znaczy rys. 10.6(a) reprezentuje także trasy (2, 4, 3, 8, 6, 5, 7, 9, 1), (4, 3, 8, 6, 5, 7, 9, 1, 2) itp.

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	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

b)

	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	0	0	1	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	0	0	1
7	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	0
9	0	0	1	0	0	0	0	0	0

Rys. 10.6. Macierze binarne chromosomów

Warto zauważyć, że każda kompletna trasa jest reprezentowana przez macierz binarną z tylko jednym bitem jedynkowym w każdym wierszu i jednym bitem w każdej kolumnie. Jednak nie każda macierz o tej właściwości reprezentuje jedną trasę. Chromosomy z macierzy binarnych mogą reprezentować wielokrotne podtrasy. Każda z podtras może być odrębnym cyklem, bez połączenia z inną podtrasą w chromosomie. Na przykład chromosom z rys. 10.6(b) reprezentuje dwie podtrasy (1, 2, 4, 5, 7) i (3, 8, 6, 9).

Podtrasy były dopuszczane z nadzieją, że zajdzie naturalne pogrupowanie. Po zakończeniu programu ewolucyjnego najlepszy chromosom jest sprowadzany do jednej trasy przez kolejne łączenie par podtras za pomocą algorytmu deterministycznego. Podtrasy składające się z jednego miasta (trasa wychodząca z miasta i kończąca się natychmiast na nim), z zerowym kosztem przejazdu nie były dopuszczane. Ustalono najniższą granicę $q = 3$ miast w podtrasie, aby algorytm genetyczny nie sprowadził zadania komiwojażera do dużej liczby podtras obejmujących bardzo mało miast (q jest parametrem metody).

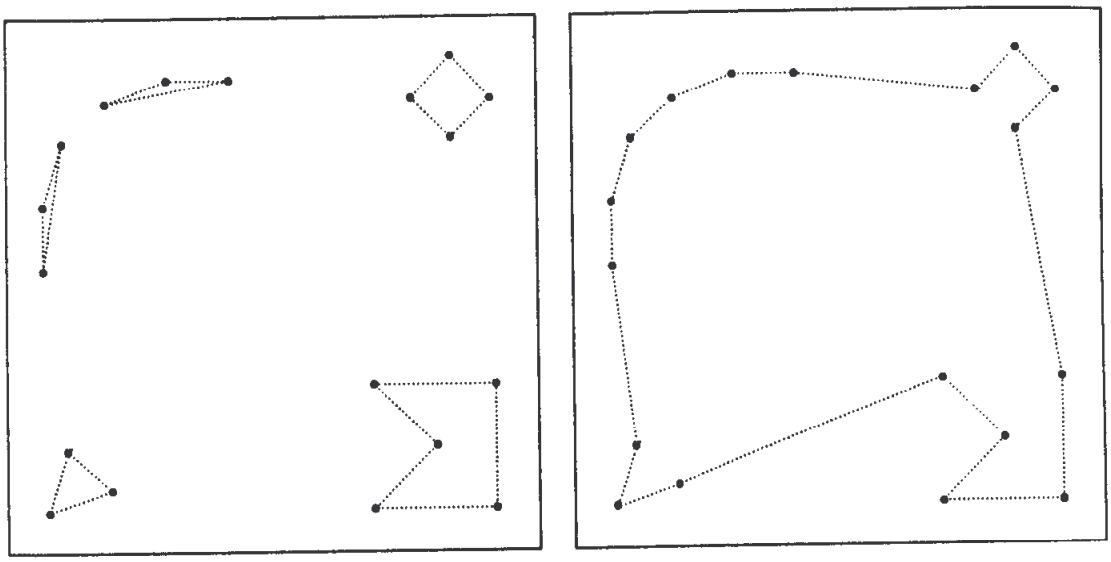
Na rysunku 10.7(a) przedstawiono podtrasy wynikłe z próbnego przebiegu algorytmu dla pewnej liczby miast celowo umieszczonych w grupach. Tak jak się spodziewano, algorytm wygenerował izolowane podtrasy. Na rysunku 10.7(b) przedstawiono całą trasę po połączeniu poszczególnych podtras.

Określono dwa operatory genetyczne: mutację i krzyżowanie. Operator mutacji losowo wybiera kilka wierszy i kolumn w chromosomie, usuwa zbiór bitów na przecięciach tych wierszy i kolumn, a następnie zamienia je w możliwie innych konfiguracjach.

Na przykład rozważmy macierz z rys. 10.6(a) reprezentującą trasę

(1, 2, 4, 3, 8, 6, 5, 7, 9)

Przypuśćmy, że wiersze 4, 6, 7, 9 oraz kolumny 1, 3, 5, 8 i 9 wybrano losowo do udziału w mutacji. Policzono sumy elementów w tych wierszach i kolumnach. Bity na przecięciu tych wierszy i kolumn usunięto i zamieniono w sposób losowy, jednak tak, aby wcześniej policzone sumy zgadzały się. To znaczy



Rys. 10.7. Rozdzielne podtrasy i końcowa trasa

a)

0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
1	0	0	0	0

b)

0	0	1	0	0
0	0	0	0	1
1	0	0	0	0
0	1	0	0	0

Rys. 10.8. Część chromosomu przed (a) i po (b) mutacji

podmacierz odpowiadająca wierszom 4, 6, 7 i 9 oraz kolumnom 1, 3, 5, 8 i 9 z macierzy początkowej (rys. 10.8(a)) zamienia się na inną macierz (rys. 10.8(b)).

Otrzymany chromosom zawiera dwie podtrasy

(1, 2, 4, 5, 7) i (3, 8, 6, 9)

i jest przedstawiony na rys. 10.6(b).

W operatorze krzyżowania zaczyna się od chromosomu dziecka, który ma wszystkie bity zerowe. Najpierw bada się w nim dwa chromosomy rodziców i kiedy odkryje się te same zbiory bitów (to znaczy 1) u obu rodziców (jednakowe wiersze i kolumny), to wstawia się odpowiedni bit u dziecka (pierwsza faza). Następnie na zmianę kopiuje się zbiory bitów od rodziców tak długo, aż nie pozostaną u żadnego rodzica bity, które można skopiować bez naruszenia podstawowych ograniczeń wynikających z konstrukcji chromosomu (druga faza). Na koniec, jeżeli jakiś wiersz chromosomu dziecka nadal nie zawiera zbioru bitów, wypełnia się je losowo (końcowa faza). Ponieważ krzyżowanie tradycyjnie tworzy dwa chromosomy dziecięce, więc działanie operatora powtarza się, zamieniając kolejność rodziców.

W poniższym przykładzie krzyżowania zaczyna się od chromosomu pierwszego rodzica z rys. 10.9(a) reprezentującego dwie podtrasy

(1, 5, 3, 7, 8) i (2, 4, 9, 6)

oraz chromosomu drugiego rodzica (rys. 10.9(b)) reprezentującego jedną trasę

(1, 5, 6, 2, 7, 8, 3, 4, 9)

Dwie fazy tworzenia pierwszego potomka przedstawiono na rys. 10.10.

a)

	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

b)

	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	1	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	1	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

Rys. 10.9. Pierwszy (a) i drugi (b) rodzic

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	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

b)

	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	1	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	0	0	0	0

Rys. 10.10. Potomek z krzyżowania po (a) fazie 1 oraz (b) fazie 2

Pierwszego potomka z krzyżowania, po końcowej fazie, przedstawiono na rys. 10.11. Reprezentuje on trasę

(1, 5, 6, 2, 3, 4, 9, 7, 8)

Drugi potomek reprezentuje

(1, 5, 3, 4, 9) i (2, 7, 8, 6)

Zauważmy, że u obu potomków znajdują się wspólne segmenty z chromosomów rodziców.

Ten program ewolucyjny nieźle działał na kilku zadaniach testowych zawierających od 30 do 512 miast. Jednak nie jest jasne, jak wpływa parametr q (najmniejsza liczba miast w podtrasie) na jakość końcowego rozwiązania. Także nie jest oczywiste, jak powinien wyglądać algorytm łączenia kilku pod-

tras w jedną podtrasę. Natomiast metoda wykazuje podobieństwo do algorytmu Litkego do rekurencyjnego grupowania [246], w którym rekurencyjnie zamienia się grupy o liczebności B na reprezentujące je pojedyncze miasta tak długo, aż pozostanie mniej niż B miast. Wtedy mniejsze zadanie rozwiązuje się optymalnie. Następnie rozwija się każdą grupę jedna po drugiej i algorytm włącza rozwinięty zbiór między dwóch sąsiadów bieżącej trasy. Takie podejście mogłoby być też użyteczne przy rozwiązywaniu zadania z wieloma komiwojażerami, gdzie kilku komiwojażerów powinno pokonać swoje oddzielne (nie nakładające się) trasy.

	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	1	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	0	1	0	0

Rys. 10.11. Potomek z krzyżowania po końcowej fazie

Trzecie podejście z reprezentacją macierzową zaproponowali ostatnio Homaifar i Guan [194]. Tak jak w poprzednich podejściach, element m_{ij} macierzy binarnej M jest równy 1 wtedy i tylko wtedy, gdy miasta i oraz j są połączone krawędzią. Jednak użyli oni innych operatorów krzyżowania i heurystycznej inwersji. Omówimy je po kolei.

Określono dwa macierzowe operatory krzyżowania (MX) [194]. Operatory te wymieniają wszystkie elementy macierzy rodziców znajdujące się albo po jednym punkcie cięcia (krzyżowanie jednopunktowe), albo między dwoma punktami cięcia (krzyżowanie dwupunktowe). Dodatkowo uruchamia się „algorytm naprawy", aby: (1) usunąć powtórzenia, to znaczy zapewnić, aby każdy wiersz i każda kolumna miały dokładnie jedną jedynkę, (2) rozciąć i połączyć cykle (jeżeli są), aby utworzyć trasę dopuszczalną.

Krzyżowanie dwupunktowe zilustrujemy następującym przykładem. Dwie macierze rodziców są przedstawione na rys. 10.12. Reprezentują one dopuszczalne trasy

(1 2 4 3 8 6 5 7 9) oraz (1 4 3 6 5 7 2 8 9)

Wybrano dwa punkty cięcia. Są to punkty między kolumnami 2 i 3 (pierwszy punkt) oraz kolumnami 6 i 7 (drugi punkt). Punkty cięcia przecinają macierze pionowo. W obu macierzach pierwsze dwie kolumny tworzą pierwszą część podziału, kolumny 3, 4, 5 i 6 środkową część, a ostatnie trzy kolumny trzecią część. W pierszym kroku dwupunktowego operatora wymienia się elementy

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	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

b)

	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	1	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	0	0	0	1
9	1	0	0	0	0	0	0	0	0

Rys. 10.12. Macierze binarne chromosomów z zaznaczonymi punktami cięcia

a)

	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

b)

	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

Rys. 10.13. Dwaj pośredni potomkowie po pierwszym kroku operatora macierzowego krzyżowania

obu macierzy znajdujące się między punktami cięcia (to znaczy w kolumnach 3, 4, 5 i 6). Ten pośredni wynik jest przedstawiony na rys. 10.13.

Obaj potomkowie (a) i (b) są niedopuszczalni. Jednak całkowita liczba jedynek w każdej z tych pośrednich macierzy jest prawidłowa (to znaczy wynosi 9). W pierwszym kroku „algorytmu naprawy" przesuwa się niektóre jedynki w macierzy w taki sposób, aby każdy wiersz i każda kolumna zawierała dokładnie jedną 1. Na przykład w potomku z rys. 10.13(a) podwójna jedynka znajduje się w wierszach 1 i 3. Algorytm może przesunąć element $m_{14} = 1$ do m_{84}, a element $m_{38} = 1$ do m_{28}. Podobnie w drugim potomku (rys. 10.13(b)) podwójna jedynka znajduje się w wierszu 2 i 8. Algorytm może przesunąć element $m_{24} = 1$ do m_{34}, a element $m_{86} = 1$ do m_{16}. Po zakończeniu pierwszego kroku algorytmu naprawy pierwszy potomek reprezentuje trasę (dopuszczalną)

(1 2 8 4 3 6 5 7 9)

a drugi trasę, która składa się z dwóch podtras

(1 6 5 7 2 8 9) oraz (3 4)

Drugi krok algorytmu naprawy należy zastosować tylko do drugiego potomka. W tym kroku algorytm rozcina i łączy potrasy, aby utworzyć trasę dopuszczalną. W fazie cięcia i łączenia bierze się pod uwagę krawędzie istniejące w początkowych rodzicach. Na przykład wybiera się krawędź (2 4) do połączenia tych dwóch podtras, gdyż ta krawędź istnieje u jednego z rodziców. Tak więc cała trasa (dopuszczalny drugi potomek) wygląda następująco:

(1 6 5 7 2 4 3 8 9)

Drugi operator użyty przez Homafaira i Guana [194] do uzupełnienia krzyżowania macierzowego to heurystyczna inwersja. Operator odwraca uporządkowanie miast między dwoma punktami cięcia (tak jak to robiła zwykła inwersja, omówiona wcześniej w tym rozdziale). Jeżeli odległość między dwoma punktami cięcia jest duża (inwersja wysokiego rzędu), to operator wykonuje lokalne przeszukiwanie. Jednak zachodzą dwie różnice między operatorem klasycznym i zaproponowaną tu inwersją. Po pierwsze wynikowy potomek jest akceptowany tylko wtedy, kiedy nowa trasa jest lepsza niż poprzednia. Druga różnica polega na tym, że procedura inwersji wybiera jedno miasto z trasy i szuka poprawy przez inwersje (najniższego możliwego) rzędu 2. Pierwsza inwersja, która spowoduje poprawę, kończy procedurę inwersji. Jeżeli takiej poprawy nie uzyskano, to wykonuje się inwersje rzędu 3 itp.

Przedstawione wyniki [194] wykazują, że program ewolucyjny z dwupunktowym krzyżowaniem macierzowym i inwersją działał dobrze dla zadań komiwojażera o $30 \div 100$ miastach. W najnowszych obliczeniach wynik uzyskany przez ten algorytm dla zadania z 318 miastami był tylko o 0,6% gorszy od rozwiązania optymalnego.

W tym rozdziale nie podaliśmy dokładnych wyników obliczeń dla różnych struktur danych i operatorów „genetycznych". Raczej zrobiliśmy ogólny przegląd licznych prób stworzenia udanego programu ewolucyjnego dla zadania komiwojażera. Jednym z powodów jest to, że większość cytowanych wyników mocno zależy od wielu szczegółów (liczebności populacji, liczby pokoleń, rozmiaru zadania itp.). Ponadto wiele wyników zostało podanych dla stosunkowo małych rozmiarów zadania komiwojażera (do 100 miast). Jak zauważono w [207]:

> Nie wygląda na to, aby przykłady tak małe jak 100 miast miały być obecnie uważane za dobrze pasujące do obecnego stanu wiedzy o optymalizacji globalnej. Aby być pewnym, że podejścia heurystyczne są rzeczywiście potrzebne, należy rozważyć przykłady znacznie większe.

Natomiast w większości prac cytowanych w tym rozdziale porównuje się zaproponowane rozwiązania z innymi. Do tych porównań używano dwóch rodzin przykładów testowych:

- Losowego zbioru miast. W tym przypadku użyteczna jest zależność empiryczna na spodziewaną długość L^* minimalnej trasy w zadaniu komiwojażera:

  $$L^* = k\sqrt{n \cdot R}$$

  gdzie n jest liczbą miast, R jest powierzchnią kwadratu, w którym losowo umieszczono miasta, a k jest stałą wyznaczoną empirycznie wynoszącą w przybliżeniu 0,765[1] (patrz [372]).
- Ogólnie dostępnego zbioru miast (częściowo z optymalnymi rozwiązaniami), znajdującego się na serwerze ftp softlib.rice.edu w katalogu /pub/tsplib.

Aby uzyskać całościowy pogląd na zastosowanie algorytmów genetycznych w zadaniach komiwojażera powinniśmy przedstawić prace innych badaczy, na przykład [288] i [389], którzy używali algorytmów genetycznych do lokalnej optymalizacji w zadaniach komiwojażera. Algorytmy optymalizacji lokalnej (2-opt, 3-opt, Lina-Kernighana) są dosyć efektywne. Jak napisano w [207]:

> Biorąc pod uwagę, że zadanie jest NP-trudne i wobec tego jest mało prawdopodobne, aby istniały algorytmy z wielomianowym czasem liczenia, przywiązano dużo wagi do opracowania efektywnych algorytmów przybliżonych, algorytmów szybkich, które próbują znaleźć trasy tylko bliskie optymalnym. Obecnie najlepsze praktycznie algorytmy tego typu opierają się na (lub pochodzą od) ogólnej metody zwanej *lokalną optymalizacją*, w której podane rozwiązanie jest poprawiane iteracyjnie za pomocą lokalnych zmian.

W algorytmach lokalnej optymalizacji dla danej (bieżącej) trasy określa się zbiór tras sąsiednich i zamienia się bieżącą trasę na (możliwie) lepszą sąsiednią. Ten krok powtarza się tak długo, aż osiągnie się lokalne optimum. Na przykład w algorytmie 2-opt określa się trasy sąsiednie jako trasy, które można uzyskać jedną z drugiej przez zmianę tylko dwóch krawędzi.

Algorytmy lokalnej optymalizacji są podstawą rozwoju algorytmów genetycznych lokalnego przeszukiwania [389], w których:

- używa się algorytmów lokalnego przeszukiwania do zamiany każdej trasy w bieżącej populacji (o liczebności μ) przez trasę lokalnie optymalną,

---

[1] Zgodnie z tym, co twierdzi David Johnson [209], takie porównanie eksperymentalne należy robić ostrożnie. Po pierwsze skłania się on obecnie ku nieco mniejszej ocenie stałej asymptotycznej, bardziej typu 0,7128 ($\pm$0,005). Po drugie stosunek $L^*/\sqrt{n}$ (dla $R = 1$) zbiega się bardzo wolno, nawet dla $n = 100\,000$ wynosi on 0,7134, a dla $n = 1000$ około 0,7306 (obie oceny wzięte z [209]). Po trzecie różnice między przypadkami są całkiem spore dla mniejszych rozmiarów. Dla $n = 1000$ odchylenie standardowe wynosi około 0,0064, a dla $n = 100$ jest równe 0,0224. Wniosek jest taki, że powinno się raczej porównywać długość trasy z ograniczeniem Helda-Karpa dla danego przypadku niż z optymalną oceną spodziewanej długości. Ograniczenie Helda-Karpa [179], [180] wymaga procesu iteracyjnego związanego z utworzeniem kilku rozpinających drzew oraz relaksacją Lagrange'a.

- rozszerza się populację o dodatkowe λ tras – potomków operatora rekombinacyjnego zastosowanego do niektórych tras bieżącej populacji,
- używa się (ponownie) algorytmu lokalnego przeszukiwania do zamiany każdego z λ potomków w rozszerzonej populacji na trasę lokalnie optymalną,
- zmniejsza się rozszerzoną populację do początkowej liczebności μ zgodnie z pewną regułą selekcji (przeżycia najbardziej dopasowanych),
- powtarza się ostatnie trzy kroki tak długo, aż będzie spełniony pewien warunek zakończenia (proces ewolucyjny).

Zauważmy, że istnieją pewne podobieństwa między algorytmami genetycznymi lokalnego przeszukiwania a strategią ewolucyjną $(\mu + \lambda)$ (rozdz. 8). Tak jak w $(\mu + \lambda)$-SE, μ osobników tworzy λ potomków i nowa (rozszerzona) populacja o $(\mu + \lambda)$ osobnikach jest zmniejszana w procesie selekcji znowu do μ osobników.

Powyższy algorytm genetyczny lokalnego przeszukiwania [389] jest podobny do algorytmu genetycznego dla zadania komiwojażera zaproponowanego wcześniej przez Mühlenbeina, Gorgesa-Schleutera i Krämera [288], gdzie zachęca się do „inteligentnej ewolucji" osobników. W algorytmie tym:

- używa się algorytmu lokalnego przeszukiwania (2-opt) do zamiany trasy w bieżącej populacji na trasę lokalnie optymalną,
- dobiera się partnerów w pary (osobniki oceniane powyżej średniej tworzą więcej potomków),
- przechodzi się przez proces reprodukcji (krzyżowanie i mutację),
- poszukuje się minimum dla każdego osobnika (zmniejszenie, rozwiązanie zadania, rozszerzenie),
- powtarza się ostatnie trzy kroki tak długo, aż spełni się pewien warunek stopu (proces ewolucyjny).

Krzyżowanie zastosowane w tym algorytmie jest wersją krzyżowania z uporządkowaniem (OX). Tak więc dwoje rodziców (z punktami cięcia oznaczonymi przez „|")

$$p_1 = (1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9) \qquad \text{oraz}$$

$$p_2 = (4\ 5\ 2 \mid 1\ 8\ 7\ 6 \mid 9\ 3)$$

tworzy potomka w następujący sposób. Po pierwsze odcinki między punktami cięcia kopiuje się do potomków

$$o_1 = (\mathrm{x}\ \mathrm{x}\ \mathrm{x} \mid 4\ 5\ 6\ 7 \mid \mathrm{x}\ \mathrm{x}) \qquad \text{oraz}$$

$$o_2 = (\mathrm{x}\ \mathrm{x}\ \mathrm{x} \mid 1\ 8\ 7\ 6 \mid \mathrm{x}\ \mathrm{x})$$

Następnie (zamiast zaczynać od drugiego punktu cięcia u jednego z rodziców, jak w przypadku OX) kopiuje się miasta od drugiego rodzica, zachowując kolejność od początku łańcucha, omijając jednak już wstawione oznaczenia:

$o_1 = (2\ 1\ 8 \mid 4\ 5\ 6\ 7 \mid 9\ 3)$ oraz

$o_2 = (2\ 3\ 4 \mid 1\ 8\ 7\ 6 \mid 5\ 9)$

Wyniki eksperymentalne były przynajmniej zachęcające. W zadaniu z 532 miastami algorytm znalazł trasę o długości 27 702, która jest odległa o 0,06% od rozwiązania optymalnego (27 686, znalezionego przez Padberga i Rinaldiego [302]).

Ostatnio opisano dwa inne podejścia. W pierwszym podejściu Craighurst i Martin [68] skoncentrowali się na badaniu powiązań między metodami zapobiegania kazirodztwu (patrz także rozdz. 4) a działaniem algorytmu genetycznego dla zadania komiwojażera. Autorzy używali w eksperymentach algorytmu genetycznego z następującymi cechami: liczebność populacji 128, selekcja z populacji oparta na rankingu, w którym 128 potomków konkuruje ze 128 rodzicami, selekcja dla operatora PMX oparta na rankingu, lokalny wzrost funkcji celu (2-opt) wykonuje się przy tworzeniu potomków, współczynnik mutacji wynosi 0,005, a warunkiem zakończenia jest 500 kolejnych pokoleń bez poprawy. Podejście do kazirodztwa opiera się na pojęciu rodziny. Autorzy rozważali tylko przodków dwóch osobników wybranych do krzyżowania i wprowadzili kilka zakazów kazirodztwa. Zakaz kazirodztwa k-tego rzędu zabrania łączenia w pary osobników z $k-1$ przodkami (to znaczy dla $k = 0$ nie ma ograniczeń, dla $k = 1$ osobnik nie może łączyć się ze sobą, dla $k = 2$ nie może łączyć się ze sobą, ze swoimi dziećmi, rodzicami ani ich rodzeństwem itp.). Wykonano kilka eksperymentów (na sześciu zadaniach testowych (ze wspomnianej wcześniej w tym rozdziale biblioteki softlib.rice.edu) z liczbą miast w zakresie 48-101). Wyniki wskazywały na mocną i ciekawą wzajemną zależność między zakazami kazirodztwa a współczynnikiem mutacji. Dla małych współczynników mutacji zakaz kazirodztwa poprawia wyniki. Jednak gdy współczynnik mutacji rośnie, znaczenie mechanizmu zakazu kazirodztwa spada, aż wreszcie (przy wysokich współczynnikach mutacji $\approx 0{,}1$) pogarsza on wyniki systemu. Także różne zakazy kazirodztwa nie wpływają istotnie na rozmaitość populacji (przy czym podobieństwo między dwoma osobnikami mierzy się stosunkiem różnicy między całkowitą liczbą krawędzi w rozwiązaniu a liczbą wspólnych krawędzi w obu rozwiązaniach oraz całkowitej liczby krawędzi w rozwiązaniu). Końcowym wnioskiem była negatywna odpowiedź na następujące pytanie: „czy lepszy jest mocniejszy zakaz?". Szerszą dyskusję wyników można znaleźć w [68].

Valenzuela i Jones [390] zaproponowali ciekawe podejście do zastosowania algorytmów ewolucyjnych w trudnych zadaniach kombinatorycznych. Ich metoda bazuje na pomyśle metody „dziel i zwyciężaj" w algorytmie Karpa-Steele'a dla zadania komiwojażera [219], [379]. Ich algorytm ewolucyjny „dziel i zwyciężaj" można zastosować do każdego zadania, w którym pewna wiedza o dobrych rozwiązaniach podproblemów jest użyteczna w tworzeniu rozwiązania globalnego. Jednak oni stosowali tę metodę do geometrycznego zadania komiwojażera. Można także rozważyć kilka metod bisekcji. W tych metodach

dzieli się prostokąt z n miastami na dwa mniejsze prostokąty (na przykład w jednej z metod dzieli się zadanie za pomocą dokładnego rozcięcia pola prostokąta równolegle do mniejszego boku, w innej metodzie odcina się $int(n/2)$ miast leżących najbliżej krótszego brzegu prostokąta, w ten sposób tworząc miasta „podzielone" między dwa podprostokąty. Końcowe podproblemy są całkiem małe (na ogół od 5 do 8 miast) i można je stosunkowo łatwo rozwiązać (do tego wybrano metodę 2-opt ze względu na jej szybkość i prostotę). Algorytm łączący zamienia niektóre krawędzie w dwóch odrębnych trasach, aby uzyskać jedną większą trasę. Główna rola algorytmu genetycznego polega na określeniu kierunku podziału (poziomy czy pionowy) w każdym etapie. Warto zauważyć, że struktura danych użyta do reprezentacji chromosomu osobnika była macierzą binarną M o wymiarach $p \times p$[1], skorelowaną z obszarami geometrycznymi kwadratu z zadania komiwojażera. Jeżeli w pewnym etapie algorytmu „dziel i zwyciężaj" należy podzielić prostokąt, to wybiera się bit z macierzy M, który jest najbliżej środka prostokąta. Wartość tego bitu określa kierunek bieżącego cięcia (poziome czy pionowe). Operatory genetyczne zastosowane w [390] były proste: krzyżowanie wymienia elementy binarne między dwiema macierzami, które są przecięte (w dwóch punktach) wzdłuż osi x lub y[2]. Mutacja rozrzuca wszystkie bity macierzy ze stałym prawdopodobieństwem (użyto wartości 0,1, to znaczy 10% bitów w macierzy podlega mutacji). Wyniki eksperymentalne i omówienie całości udziału algorytmów genetycznych w tym podejściu można znaleźć w [390].

Wygląda na to, że dobry program ewolucyjny dla zadania komiwojażera powinien zawierać operatory lokalnej poprawy (z grupy mutacji) oparte na algorytmach lokalnej optymalizacji oraz starannie zaprojektowane operatory binarne (z grupy krzyżowania), które zawierają heurystyczną informację o zadaniu. Kończymy ten rozdział prostym stwierdzeniem: poszukiwanie programu ewolucyjnego dla zadania komiwojażera z „najlepszą" reprezentacją i operatorami „genetycznymi", które na niej operują, ciągle trwa.

---

[1] Autorzy rozważali kwadrat, więc macierz ma równe wymiary.

[2] Autorzy przyjęli dodatkowe ograniczenia, że odległość tych dwóch punktów cięcia wynosi między jedną trzecią a dwoma trzecimi odległości wzdłuż osi, aby zapewnić rozsądną proporcję materiału genetycznego od każdego rodzica.

11

Programy ewolucyjne dla różnych zadań dyskretnych

Pewien rozczarowany gość zapytał:
– Dlaczego mój pobyt tutaj nie przyniósł
żadnych owoców?
– Być może dlatego, że zabrakło ci odwagi,
aby potrząsnąć drzewem?
łagodnie odpowiedział Mistrz.

Anthony de Mello, *Minuta mądrości*[1)]

Jak było napisane we wprowadzeniu, wydaje się, że większość badaczy modyfikowała swoje programy algorytmów genetycznych, używając do rozwiązywania zadań niestandardowych reprezentacji chromosomowych i ewentualnie projektując operatory genetyczne zależne od zadania (na przykład [141], [385], [65], [76] itd.), w ten sposób tworząc wydajne programy ewolucyjne. Takie modyfikacje omawialiśmy szczegółowo w poprzednich dwóch rozdziałach (rozdz. 9 i 10) dla zadania transportowego i zadania komiwojażera. W tym rozdziale zrobiliśmy trochę arbitralny wybór kilku innych programów ewolucyjnych opracowanych przez autora lub innych badaczy, opartych na niestandardowych reprezentacjach chromosomowych oraz operatorach z wiedzą specyficzną dla zadania. Omawiamy systemy dla zadania harmonogramowania (p. 11.1), zadania układania planu lekcji (p. 11.2), zadań podziału (p. 11.3) i zadania planowania drogi w środowisku ruchomych robotów (p. 11.4). Rozdział kończy się dodatkowym punktem 11.5, który zawiera kilka uwag o innych ciekawych zadaniach.

Opisane systemy i wyniki ich zastosowań są jeszcze jednym argumentem potwierdzającym słuszność podejścia programowania ewolucyjnego, które sprzyja tworzeniu struktur danych oraz operatorów dla poszczególnych klas zadań.

11.1. Harmonogramowanie

Rozważmy warsztat, w którym można zorganizować różne procesy wytwórcze. Jego główną cechą jest wielka różnorodność prac, które można tu wykonać [182]. Warsztat wytwarza dobra (części), które wymagają jednego lub więcej różnych programów obróbki. Każdy program obróbki składa się z ciągu ope-

[1)] Przekład Bogusława Steczka, wyd. Apostolstwa Modlitwy, 1991 (przyp. tłum.).

racji. Wymagają one zasobów i zajmują czas pracy maszyn. Warsztat pracuje na zamówienia, przy czym każde zamówienie dotyczy wykonania pewnej liczby tych samych części. Zadanie planowania, harmonogramowania i sterowania pracą takiego warsztatu jest bardzo skomplikowane i nie ma dużo wyników analitycznych pomocnych w jego rozwiązaniu [182], [132].

Zadanie harmonogramowania warsztatu polega na wybraniu ciągu operacji razem z przyporządkowaniem czasów rozpoczęcia i zakończenia oraz zasobów dla każdej operacji. Pod uwagę należy wziąć głównie koszt jałowej pracy maszyn oraz ich zdolności przetwórczych, koszt przetrzymywania (składowania) nie dokończonych części i konieczność dotrzymania terminów wykonania. Jak podano w [182]:

> Niestety te wymagania wykazują tendencje do sprzeczności. Można uzyskać niski koszt jałowego biegu maszyn i wykorzystania ich zdolności, przyporządkowując do tego minimalną liczbę maszyn i pracy ludzkiej. Jednak będzie to prowadziło do znacznego przetrzymywania części i wobec tego potrzeby znacznego ich składowania oraz trudności z dotrzymaniem terminów wykonania. Z drugiej strony zasadniczo można zagwarantować dotrzymanie terminów wykonania przez zapewnienie tak dużej liczby maszyn i pracy ludzkiej, że zamówienia nie będą musiały czekać na wykonanie. Jednak wtedy otrzyma się nadmiarowe koszty związane z jałową pracą maszyn i niewykorzystaniem ich zdolności przetwórczych. W związku z tym zachodzi konieczność dążenia do ekonomicznego kompromisu między tymi wymaganiami.

Są różne wersje zadania harmonogramowania prac na maszynach, każda z nich charakteryzująca się pewnymi dodatkowymi ograniczeniami (na przykład związanymi z konserwacjami, czasami zatrzymywania i uruchamiania maszyny itp.).

Aby zilustrować powyższy opis, rozważmy prosty przykład zadania rozdziału prac dla obrabiarek.

Przykład 11.1

Przypuśćmy, że mamy trzy zamówienia: o_1, o_2, o_3. W każdym zamówieniu trzeba wykonać następujące części i ich liczbę:

o_1 : 30 części a

o_2 : 45 części b

o_3 : 50 części a

Każda część wymaga wykonania jednego lub więcej programów obróbki:

a : program 1_a (opr_2, opr_7, opr_9)

a : program 2_a (opr_1, opr_3, opr_7, opr_8)

a : program 3_a (opr_5, opr_6)

b : program 1_b (opr_2, opr_6, opr_7)

b : program 2_b (opr_1, opr_9)

gdzie opr_i oznacza wymagane do wykonania operacje. Każda operacja wymaga pewnego czasu na jednej lub więcej maszynach. Oto ich wykaz:

opr_1 : (m_1 10) (m_3 20)

opr_2 : (m_2 20)

opr_3 : (m_2 20) (m_3 30)

opr_4 : (m_1 10) (m_2 30) (m_3 20)

opr_5 : (m_1 10) (m_3 30)

opr_6 : (m_1 40)

opr_7 : (m_3 20)

opr_8 : (m_1 50) (m_2 30) (m_3 10)

opr_9 : (m_2 20) (m_3 40)

Na koniec, zmiana operacji na maszynie wymaga czasu jej przezbrojenia:

m_1 : 3

m_2 : 5

m_3 : 7

Zadanie ustalania pracy maszyn wywołało pewne zainteresowanie w środowisku związanym z algorytmami genetycznymi. Jedną z pierwszych prób rozwiązania tego zagadnienia opisał Davis [72]. Główna idea jego podejścia polegała na zakodowaniu reprezentacji harmonogramu w taki sposób, aby: (1) operatory genetyczne działały w znaczący sposób, (2) dekoder zawsze dawał dopuszczalne rozwiązanie zadania. Ta strategia kodowania rozwiązań do operacji i dekodowania ich do oceny jest całkiem ogólna i może być zastosowana w różnych zadaniach z ograniczeniami. Ten sam pomysł występuje u Jonesa [211] w próbie rozwiązywania zadania podziału (patrz p. 11.5).

W ogólności chcielibyśmy reprezentować informacje o harmonogramach, na przykład „maszyna m_2 wykonuje operację o_1 na części a od chwili t_1 do t_2". Jednak większość operatorów (mutacje, krzyżowania), działając na takich informacjach, prowadziłoby do niedopuszczalnych harmonogramów. Dlatego Davis [72] użył strategii kodowania i dekodowania.

Popatrzmy, jak zastosowano strategie kodowania w zadaniu ustalania prac dla maszyn. System opracowany przez Davisa [72] utrzymywał listę preferencji dla każdej maszyny. Te preferencje powiązano z czasami. Początkowym elementem listy jest czas, w którym lista zaczęła być wykonywana, pozostałe elementy listy składają się z permutacji zamówień oraz dwóch dodatkowych

składników: „postój" i „praca jałowa". Procedura dekodująca symuluje operacje wykonywane na maszynach w taki sposób, że jeżeli tylko maszyna jest wolna, przyporządkowuje się jej pierwszą możliwą operację z listy preferencji. Jeżeli więc lista preferencji dla maszyny m_1 jest następująca:

m_1 : (40 o_3 o_1 o_2 „postój" „praca jałowa")

to procedura dekodująca w chwili 40 poszuka części z zamówienia o_3 do obróbki na maszynie m_1. Jeżeli to się nie uda, procedura dekodująca poszuka części z zamówień o_1 i o_2 (to znaczy z o_1, a w przypadku gdy jest to niemożliwe, z o_2). Ta reprezentacja gwarantuje, że otrzymany harmonogram będzie dopuszczalny.

Operatory były specyficzne dla zadania (wzięte z metod deterministycznych):

Praca jałowa: ten operator jest używany tylko z listami preferencji dla maszyn, które czekały dłużej niż godzinę. Wstawia on „praca jałowa" na drugim miejscu listy preferencji i ustawia pierwszy element listy (czas) na 60 (minut).

Mieszanie: ten operator „miesza" elementy na liście preferencji.

Krzyżowanie: ten operator wymienia listy preferencji dla wybranych maszyn.

Prawdopodobieństwa zastosowania tych operatorów, odpowiednio dla mieszania i krzyżowania, zmieniały się od 5% i 40% na początku obliczenia aż do 1% i 5%. Prawdopodobieństwo operatora pracy jałowej ustawiono jako procent czasu, jaki maszyna spędzała na czekaniu, dzielonym przez całkowity czas symulacji.

Jednak obliczenia robiono tylko dla małych przykładów z dwoma zamówieniami, sześcioma maszynami i trzema operacjami [72]. Trudno jest więc ocenić użyteczność tego podejścia.

Inna grupa badaczy zajęła się rozdziałem prac na poszczególne maszyny z punktu widzenia zadania komiwojażera [63], [383], [384], [402]. Ich motywacja polegała na tym, że większość operatorów określonych dla zadania komiwojażera jest „ślepa", to znaczy nie używają one żadnej informacji o rzeczywistych odległościach między miastami (patrz rozdz. 10). Oznacza to, że te operatory mogą być przydatne w innych zadaniach szeregowania, gdzie nie ma odległości między dwoma punktami (miastami, zamówieniami, pracami itp.). Jednak tak być nie musi. Chociaż oba zadania, komiwojażera i harmonogramowania, są zadaniami szeregowania, to mają one inne (uzależnione od zadania) cechy. Dla zadania komiwojażera ważną informacją jest przyległość miast, gdy tymczasem w zadaniu harmonogramowania głównym celem jest wzajemna kolejność prac. Informacja o przyległości jest bezużyteczna w zadaniu harmonogramowania, gdy tymczasem wzajemna kolejność nie jest ważna dla zadania komiwojażera z powodu cyklicznego charakteru tras. Trasy (1 2 3 4 5 6 7 8) i (4 5 6 7 8 1 2 3) są faktycznie jednakowe. Dlatego też w tych innych zastosowaniach potrzebujemy innych operatorów. Jak zauważono w [370]:

Gil Syswerda [383] wykonał badania, w których „kombinacja krawędzi" (operator genetyczny specjalnie opracowany dla zadania komiwojażera) spisywał się słabo w porównaniu z innymi operatorami w harmonogramowaniu sekwencji prac. Ponieważ liczebność populacji przyjęta przez Syswerdę była mała (30 łańcuchów) i uzyskał on dobre wyniki, stosując tylko mutację (bez operatorów rekombinacyjnych), więc dyskusja Syswerdy na temat wzajemnej istotności pozycji, uporządkowania i przyległości w różnych zadaniach szeregowania wywołuje pytania, które nie były dotąd dostatecznie podniesione. Badacze, łącznie z nami [402], [403], milcząco zakładają, że wszystkie zadania szeregowania są podobne i że jeden operator genetyczny powinien nadawać się do wszystkich zadań szeregowania.

Podobnie zauważyli rok wcześniej Fox i McMahon [131][1]:

Ważnym zagadnieniem jest możliwość użycia każdego operatora genetycznego w różnych zadaniach szeregowania. Na przykład w zadaniu komiwojażera wartość sekwencji jest równoważna wartości tej samej sekwencji ustawionej w odwrotnym porządku. Ta cecha nie jest prawdziwa we wszystkich zadaniach szeregowania. W zadaniu harmonogramowania byłby to duży błąd.

W [370] porównano działanie sześciu operatorów szeregujących (krzyżowania z uporządkowaniem, krzyżowania z częściowym odwzorowaniem, krzyżowania cyklicznego, krzyżowania z wydłużaniem krawędzi, krzyżowania bazującego na porządku oraz krzyżowania bazującego na pozycjach – wszystkie te operatory były omówione w poprzednim rozdziale) w dwóch zastosowaniach szeregowania: zadania komiwojażera (ślepego) z 30 miastami oraz harmonogramowania ze 195 pracami do uszeregowania. Tak jak się spodziewano, wyniki z optymalizacji harmonogramu (o ile chodzi o „przydatność" tych sześciu operatorów) były prawie odwrotne do wyników z zadania komiwojażera. W przypadku optymalizacji harmonogramu najlepszy był operator z wydłużaniem krawędzi, za nim blisko następowały krzyżowanie z uporządkowaniem, krzyżowanie bazujące na porządku i krzyżowanie bazujące na pozycjach, a najgorsze były PMX i krzyżowanie cykliczne. Natomiast w przypadku zadania komiwojażera najlepsze były krzyżowania bazujące na porządku i na pozycjach, za nimi następowały krzyżowanie cykliczne i PMX, a na końcu były krzyżowanie z uporządkowaniem i krzyżowanie z wydłużaniem krawędzi. Te różnice można wyjaśnić, badając, jak te operatory zachowują informację o przyległości (w zadaniu komiwojażera) i porządku (w zadaniu harmonogramowania).

Podobne uwagi można sformułować dla innych zadań szeregowania (porządkowania). W [78] Davis opisuje algorytm genetyczny bazujący na porządku dla następującego zadania kolorowania grafu:

---

[1] Przedrukowano za zgodą z Rawlins G. *Foundation of Genetic Algorithms*, 1991.

> Mając zadany graf z ważonymi węzłami i n kolorów, osiągnąć najlepszy wynik w przyporządkowaniu kolorów do węzłów, aby żadna para połączonych (bezpośrednio) węzłów nie miała tego samego koloru. Funkcją celu jest całkowita suma wag pokolorowanych węzłów.

Prosty algorytm zachłanny poukładałby zbiór węzłów w porządku malejących wag i przetwarzał węzły (przyporządkowując pierwszy dopuszczalny kolor węzłowi z listy kolorów) w tym porządku. Oczywiście jest to zadanie szeregowania – przynajmniej jedna permutacja węzłów daje maksymalny wynik, a więc szukamy optymalnego ustawienia węzłów. Jest także jasne, że prosty algorytm zachłanny nie gwarantuje optymalnego rozwiązania, należy użyć jakichś innych metod. Ponownie, na pierwszy rzut oka, zadanie jest podobne do zadania komiwojażera, w którym poszukujemy najlepszego uporządkowania miast odwiedzanych przez komiwojażera. Jednak natura tego zadania jest zupełnie inna. Na przykład w zadaniu kolorowania grafu ważone są węzły, gdy tymczasem w zadaniu komiwojażera wagi są rozłożone między węzłami (jako odległości). W [78] Davis reprezentował uporządkowanie za pomocą listy węzłów (na przykład (2 4 7 1 4 8 3 5 9) jako reprezentację ścieżki w zadaniu komiwojażera) i użył dwóch operatorów: krzyżowania bazującego na porządku (omawianego w poprzednim rozdziale jako operatora dla zadania komiwojażera) oraz mutacji z mieszaniem podlist. Nawet mutacja, która powinna przeprowadzać lokalną modyfikację chromosomu, wydaje się być uzależniona od zadania [78]:

> Kusi myślenie o mutacji jako o wymianie wartości dwóch pól w chromosomie. Próbowałem tego w kilku różnych zadaniach, jednak nie działają one tak dobrze, jak mutacja z mieszaniem podlist.

W mutacji z mieszaniem podlist wybiera się podlistę węzłów od rodzica i miesza się ją u potomka, to znaczy rodzic (z początkiem i końcem wybranej podlisty zaznaczonymi znakiem „|")

$$p = (2\ 4\ |\ 7\ 1\ 4\ 8\ |\ 3\ 5\ 9)$$

może utworzyć potomka

$$o = (2\ 4\ |\ 4\ 8\ 1\ 7\ |\ 3\ 5\ 9)$$

Jednak trzeba sprawdzić, jak ten operator działa w innych zadaniach porządkowania lub harmonogramowania. I znowu zacytujmy [78]:

> W zadaniach polegających na porządkowaniu można użyć dużo innych typów mutacji. Mutacja z mieszaniem podlist jest najbardziej ogólna z tych, których używałem. Jak dotąd nic nie opublikowano o operatorach tego typu, chociaż jest to obiecujący kierunek przyszłych prac.

Powróćmy do zadań harmonogramowania. Jak wspomniano wcześniej, Syswerda [383] opracował program ewolucyjny dla zadań harmonogramowania. Jednak wybrał on prostą reprezentację chromosomu:

> Wybierając reprezentację chromosomu dla [...] harmonogramu, mamy do wyboru dwa podstawowe elementy. Pierwszym jest lista prac do harmonogramowania. Ta lista jest bardzo podobna do listy miast do odwiedzenia w zadaniu komiwojażera. [...] Alternatywą dla sekwencji prac jest przyjęcie za chromosom po prostu harmonogramu. Może to wyglądać na wyjątkowo kłopotliwą reprezentację, dla której będą potrzebne skomplikowane operatory, ale ma ona zdecydowaną zaletę dla skomplikowanych rzeczywistych zadań, takich jak harmonogramowanie. [...] W naszym przypadku chęć czystej i prostej reprezentacji chromosomu wygrała z bardziej skomplikowanym zadaniem. Składnia chromosomu, jakiej użyliśmy w zadaniu harmonogramowania, jest taka, jaką opisalismy powyżej dla zadania komiwojażera, tylko zamiast miast użyliśmy uporządkowania prac.

Chromosom zinterpretowano jako procedurę do tworzenia harmonogramów – kawałek programu, który „rozumie" szczegóły ustalania harmonogramu. Ta reprezentacja była uzupełniona specjalizowanymi operatorami. Rozważono trzy mutacje: mutację bazującą na pozycjach (wybiera się losowo dwie prace i drugą z nich umieszcza się przed pierwszą), mutację bazującą na porządku (dwie prace wybrane losowo są zamieniane) i mutację mieszającą (taką samą jak mutacja z mieszaniem podlist Davisa opisana w poprzednim punkcie). Wszystkie trzy działały znacznie lepiej niż przeszukiwanie losowe, przy czym najlepiej spisywała się mutacja bazująca na porządku. Jak wspomniano wcześniej, najlepszymi operatorami krzyżowania dla zadania harmonogramowania były krzyżowania bazujące na porządku i bazujące na pozycjach.

Wydaje się jednak, że wybór prostej reprezentacji nie był najlepszy. Sądząc z innych (nie związanych z tymi) obliczeń, na przykład dla zadania transportowego (rozdz. 9), wydaje się, że wybrana reprezentacja chromosomu powinna być znacznie bliższa harmonogramu. To prawda, że w takim przypadku należy włożyć znaczny wysiłek w określenie specyficznych dla zadań operatorów „genetycznych". Jednak ten wysiłek się opłaci, gdyż system będzie się wykazywał większą szybkością i lepszą dokładnością. Ponadto niektóre operatory mogą być proste [383]:

> Prosty algorytm zachłanny, przechodząc po harmonogramach, mógłby znaleźć miejsce dla pracy o wysokim priorytecie, usuwając jedną lub dwie prace o niskim priorytecie i zamieniając je na pracę o wysokim priorytecie.

Wydaje się, że ogólnie, a dla zadania harmonogramowania w szczególności, jest to kierunek do naśladowania. Włączyć wiedzę specyficzną dla zadania nie tylko do operatorów (jak zrobiono w przypadku prostej reprezentacji chromosomu), ale także w strukturę chromosomu. Pierwsze próby zastosowania takiego podejścia już się pojawiły. W swoich badaniach Husbands, Mill i Warrington [198] reprezentowali chromosom przez ciąg

$$(opr_1\ m_1\ s_1)\ (opr_2\ m_2\ s_2)\ (opr_3\ m_3\ s_3)\ \ldots$$

gdzie opr_i, m_i i s_i oznaczają odpowiednio i-tą operację, maszynę i ustawienie.

W [21] autorzy porównali trzy reprezentacje, począwszy od najprostszej (reprezentacja 1)

$(o_1)\ (o_2)\ (o_3)\ \ldots$

przez pośrednią (reprezentacja 2)

$(o_1 \text{ program } 1_a)\ (o_2 \text{ program } 2_b)\ (o_3 \text{ program } 2_a)\ \ldots$

do najbardziej skomplikowanej (reprezentacja 3)

$(o_1\langle opr_2 : m_2, opr_7 : m_3, opr_9 : m_2\rangle)\ (o_2\langle opr_1 : m_3, opr_9 : m_2\rangle)$

$(o_3\langle opr_1 : m_1, opr_3 : m_2, opr_7 : m_3, opr_8 : m_1\rangle)\ \ldots$

Wyniki dla reprezentacji 3 były wyraźnie lepsze niż dla dwóch pozostałych reprezentacji. W zamykającej dyskusji autorzy zauważyli [21]:

> Same operatory należy dopasować do wymagań dziedziny. Reprezentacja chromosomu powinna zawierać wszystkie informacje, które są związane z zadaniem optymalizacji.

Podsumowując, można sklasyfikować podejścia oparte na algorytmach genetycznych zastosowane w różnych zadaniach harmonogramowania na podstawie reprezentacji chromosomów. Dzielą się one na dwie kategorie [21]:

- **Reprezentacje pośrednie**, gdzie transformacja chromosomu na dopuszczalny harmonogram jest wykonywana za pomocą specjalnego dekodera (procedury tworzenia harmonogramów). Dopiero wtedy można ocenić pojedyncze rozwiązanie. Te reprezentacje można dalej podzielić na [51] reprezentacje niezależne od dziedziny i specyficzne dla zadania. Widzieliśmy oba te przypadki we wcześniejszych punktach.
- **Reprezentacje bezpośrednie**, gdzie harmonogram produkcji jest sam chromosomem (np. [198]). Na ogół taka reprezentacja wymaga operatorów specyficznych dla zadania [53].

Pełny przegląd algorytmów ewolucyjnych dla zadania harmonogramowania można znaleźć na przykład w [54].

11.2. Układanie planu lekcji

Jednym z najbardziej interesujących zadań w badaniach operacyjnych jest zadanie układania planu lekcji. Zadanie to ma ważne praktyczne zastosowania. Było ono intensywnie badane i wiadomo, że jest ono NP-trudne [106].

W zadaniu układania planu lekcji występuje wiele nietrywialnych ograniczeń o różnym charakterze. Dlatego też chyba dopiero ostatnio (o ile autorowi wiadomo) pojawiły się pierwsze próby [63] zastosowania metod algorytmów genetycznych do tego zadania. Jest wiele wersji zadania układania planu lekcji. Jedną z nich można opisać przez

- listę nauczycieli $\{T_1, ..., T_m\}$,
- listę odcinków czasowych (godzin) $\{H_1, ..., H_n\}$,
- listę klas $\{C_1, ..., C_k\}$.

Zadanie polega na wyznaczeniu optymalnego planu (nauczyciele – godziny – klasy). Funkcja celu powinna spełniać pewne wymagania (ograniczenia miękkie). Obejmują one cele dydaktyczne (na przykład rozłożenie przedmiotów w tygodniu), cele personalne (na przykład pozostawienie wolnych popołudni dla nauczycieli pracujących na części etatu) i cele organizacyjne (na przykład każda godzina ma przyporządkowanego dodatkowego nauczyciela na zastępstwa).

Ograniczenia obejmują:

- zadaną liczbę godzin dla każdego nauczyciela i dla każdej klasy, dopuszczalny plan lekcji musi być „zgodny" z tymi liczbami,
- w każdej klasie i godzinie jest tylko jeden nauczyciel,
- nauczyciel nie może uczyć dwóch klas naraz,
- w każdej klasie z zaplanowanymi zajęciami w danej godzinie znajduje się nauczyciel.

Wygląda na to, że najbardziej naturalną reprezentacją dla chromosomu przedstawiającego potencjalne rozwiązanie zadania ustalania planu jest reprezentacja macierzowa: macierz $(R)_{ij}$ ($1 \leqslant i \leqslant m$ oraz $1 \leqslant j \leqslant n$), w której wiersz odpowiada nauczycielowi, a kolumna godzinie, elementami macierzy R są klasy ($r_{ij} \in \{C_1, ..., C_k\}$)[1].

W [63] uwzględniono ograniczenia głównie za pomocą operatorów genetycznych (autorzy zastosowali także algorytm naprawy, aby wyeliminować przypadki, w których więcej niż jeden nauczyciel jest przyporządkowany tej samej klasie w tym samym czasie). Były użyte następujące operatory genetyczne:

Mutacja rzędu k: ten operator wybiera dwa sąsiednie ciągi k-elementowe z tego samego wiersza macierzy R i zamienia je.

Mutacja dni: ten operator jest specjalnym przypadkiem poprzedniego; wybiera on dwie grupy kolumn (godzin) w macierzy R, które odpowiadają różnym dniom, i zamienia je.

Krzyżowanie: dla danych dwóch macierzy R_1 i R_2 operator ustawia wiersze pierwszej macierzy w porządku malejących wartości tak zwanej lokalnej funkcji dopasowania (część funkcji dopasowania związana tylko z cechami charakterystycznymi dla nauczyciela).

Najlepszych b wierszy (b jest parametrem ustalanym przez system na podstawie lokalnej funkcji dopasowania oraz obu rodziców) wybiera się jako bloki budujące, a pozostałych $m - b$ wierszy bierze się z macierzy R_2.

---

[1] Faktycznie w [63] elementami macierzy R były klasy z trzema indeksami, aby objąć podział klas w niektórych zajęciach, czasowo zatrudnionych nauczycieli itp.

Powstały program ewolucyjny był pomyślnie przetestowany na danych z dużej szkoły w Mediolanie we Włoszech [63].

Zadanie ustalania planu lekcji było także rozważane przez Paechtera, Luchiana i Petriuca [303], którzy porównali dwie metody ewolucyjne (metoda permutacji czasowej i przestrzennej oraz metoda umieszczania i szukania) na dużym zadaniu ustalania rzeczywistego planu lekcji. Ostatnio Burke i inni [57] opisali genetyczny algorytm hybrydowy dla zadania ustalania planu lekcji z wieloma ograniczeniami. W ich podejściu połączono bezpośrednią reprezentację planu lekcji z kilkoma heurystycznymi operatorami krzyżowania i heurystycznym operatorem mutacji. Tak więc algorytm zapewnia dopuszczalność rozwiązań przez specjalizowane struktury danych i operatory.

11.3. Podział obiektów i grafów

Jest ciekawa klasa zadań podziału[1], w której wymaga się podziału n obiektów na k kategorii. Ta klasa obejmuje wiele znanych zadań, jak zadanie pakowania kontenerów (przyporządkowania rzeczy do pojemników), zadanie kolorowania grafu (przyporządkowywanie kolorów wierzchołkom grafu) itd. Opracowano wiele różnych systemów dla różnych typów zadań podziału. W tym punkcie omówimy kilka z nich.

Jeden z rodzajów programów ewolucyjnych jest oparty na reprezentowaniu wszystkich obiektów (to znaczy przedmiotów w zadaniu pakowania pojemników lub wierzchołków w zadaniu kolorowania grafu) przez listę permutacji. Można w nich użyć specjalnych operatorów, a dekoder podejmuje decyzje o przyporządkowaniu. Na przykład w zadaniu kolorowania grafu Davis [78] reprezentował w chromosomie permutację wierzchołków i zastosował do tej struktury specjalizowane operatory (jednorodne krzyżowanie oparte na porządku, mutacja oparta na porządku). Jednocześnie użył dekodera zachłannego do interpretacji struktury w następujący sposób: rozważ szczególny kolor i pomaluj tym kolorem (jeżeli jest to możliwe) wszystkie wierzchołki (w kolejności podanej w chromosomie). Kiedy nie można już pomalować więcej wierzchołków, weź następny kolor. Davis [78] przedstawiał wyniki obliczeń dla zadania kolorowania grafu ze 100 wierzchołkami.

Ciekawe podejście do zadania podziału[2] przedstawiono w [235]. Von Laszewski zakodował podziały, używając kodowania grup za pomocą liczb, to znaczy podziały są reprezentowane jako łańcuchy całkowitoliczbowe o wymiarze n

$$(i_1, \ldots, i_n)$$

[1] Czasami te zadania nazywa się *zadaniami grupowania* [109].

[2] Dodatkowym ograniczeniem w tej wersji zadania podziału było to, że rozmiary podzielonych części mają być równe lub prawie równe.

gdzie j-ta liczba całkowita $i_j \in \{1, ..., k\}$ wskazuje grupę przyporządkowaną obiektowi j. Ta reprezentacja jest jednak wsparta „inteligentnymi operatorami strukturalnymi": krzyżowaniem strukturalnym i mutacją strukturalną. Omówimy je po kolei.

Krzyżowanie strukturalne: mechanizm krzyżowania strukturalnego wyjaśnimy na następującym przykładzie. Przypuśćmy, że wybrano dwóch rodziców (z łańcuchami 12-elementowymi)

$$p_1 = (112311232233) \text{ oraz } p_2 = (112123122333)$$

Te łańcuchy dekoduje się na następujące podziały:

$$p_1 : \{1, 2, 5, 6\}, \{3, 7, 9, 10\}, \{4, 8, 11, 12\} \quad \text{oraz}$$

$$p_2 : \{1, 2, 4, 7\}, \{3, 5, 8, 9\}, \{6, 10, 11, 12\}$$

Najpierw wybiera się losowo podział, powiedzmy podział 2, i kopiuje się go z p_1 do p_2

$$p_2' = (112123222233)$$

Proces kopiowania (jak widać z powyższego przykładu) zazwyczaj niszczy wymaganie równych rozmiarów podziałów, w związku z czym trzeba zastosować algorytm naprawy. Zauważmy, że w początkowym p_2 znajdowały się elementy przyporządkowane podziałowi 2, które nie były elementami skopiowanego podziału. Są to elementy 5 i 8. Te elementy wymazuje się

$$p_2'' = (1121 * 32 * 2233)$$

i zamienia (losowo) na liczby z innych podziałów, które nadpisano w czasie kopiowania. Tak więc końcowy potomek może mieć następującą postać:

$$p_2''' = (112133212233)$$

Jak wspomniano wcześniej, w kodowaniu grup za pomocą liczb dwa identyczne podziały mogły być reprezentowane przez inne łańcuchy ze względu na inne numery podziału. Aby temu zaradzić, zanim użyje się krzyżowania, poprawia się kodowanie przez minimalizację różnicy między dwoma rodzicami.

Mutacja strukturalna: zazwyczaj mutacja wymienia pojedynczą składową łańcucha na pewną liczbę losową. Jednak niszczy to wymaganie równego rozmiaru podziałów. Mutacja strukturalna wymienia dwie liczby w łańcuchu. Tak więc rodzic

$$p = (112133212233)$$

może utworzyć następującego potomka (zamienione liczby na pozycjach 4 i 6):

$$p' = (112331212233)$$

Algorytm zaprogramowano jako algorytm genetyczny równoległy powiększony o dodatkowe strategie (takie jak strategia zamiany rodzica). Dla losowo wybranych grafów o 900 wierzchołkach z największym stopniem wierzchołka 4 ten program ewolucyjny był wyraźnie lepszy od innych algorytmów heurystycznych [235]. Podobnego podejścia próbował Mühlenbein [287], który eksperymentował z tym samym kodowaniem grup za pomocą liczb i także używał „inteligentnego" krzyżowania, które przesyła całe podziały, a nie pojedyncze obiekty.

Kilka programów ewolucyjnych w tej klasie utworzyli Jones i Beltramo [211]. Programy te używają różnych reprezentacji i kilku operatorów do manipulacji nimi. Warto zwrócić uwagę na wpływ włączenia wiedzy specyficznej dla zadania na działanie opracowanych programów ewolucyjnych. Wybrano dwa zadania testowe:

- podział n liczb na k grup, aby zminimalizować różnicę między sumami z grup,
- podział 48 stanów kontynentalnej części Stanów Zjednoczonych na grupy o czterech kolorach tak, aby zminimalizować liczbę par stanów ze wspólną granicą, pomalowanych tymi samymi kolorami.

W pierwszej grupie programów ewolucyjnych kodowano podziały jako łańcuchy całkowitoliczbowe n-elementowe

$$(i_1, ..., i_n)$$

gdzie j-ta liczba całkowita $i_j \in \{1, ..., k\}$ oznacza numer grupy przyporządkowanej obiektowi j. To jest właśnie *kodowanie grup za pomocą liczb.*

Kodowanie grup za pomocą liczb tworzy możliwość używania standardowych operatorów. Mutacja zastępuje pojedynczy (losowo wybrany) gen i_j (losową) liczbą z zakresu $\{1, ..., k\}$. Krzyżowanie (jednopunktowe lub jednorodne) zawsze tworzy dopuszczalnego potomka. Jednak, jak zauważono w [211], potomek (po mutacji i krzyżowaniu) może zawierać mniej niż k grup. Ponadto potomek dwóch rodziców reprezentujących ten sam podział może reprezentować zupełnie inny podział ze względu na różne ponumerowanie grup. Użyto specjalnych algorytmów naprawy (metoda odrzucania, numerowanie rodziców), aby wyeliminować te kłopoty. Można także rozważyć zastosowanie krzyżowania operującego na krawędziach (określonego w poprzednim rozdziale). Zakładamy tu, że dwa obiekty są połączone krawędziami wtedy i tylko wtedy, gdy są w tej samej grupie. Krzyżowanie operujące na krawędziach tworzy potomka przez kombinację krawędzi rodziców.

Warto zauważyć, że kilka eksperymentów na dwóch zadaniach testowych wykazało wyższość krzyżowania operującego na krawędziach [211]. Jednak użyte reprezentacje nie wspomagały tego operatora. Jak podano, zajmowało to 2-5 razy więcej czasu komputerowego na iterację niż przy innych metodach krzyżowania. Jest to spowodowane nieodpowiednią reprezentacją. Na przykład dwóch rodziców

$p_1 = (11222233)$

$p_2 = (12222333)$

reprezentuje następujące krawędzie:

krawędzie dla p_1 : (12), (34), (35), (36), (45), (46), (56), (78)

krawędzie dla p_2 : (23), (24), (25), (34), (35), (45), (67), (68), (78)

Potomek powinien zawierać krawędzie znajdujące się przynajmniej u jednego rodzica, na przykład

(11222333)

reprezentuje następujące krawędzie:

(12), (34), (35), (45), (67), (68), (78)

Jednak proces wyboru krawędzi nie jest prosty. Wybór (56) i (67) – obie krawędzie są reprezentowane powyżej – powinien pociągnąć za sobą obecność krawędzi (57), której nie ma. Wydaje się, że jakaś inna reprezentacja mogłaby być bardziej odpowiednia dla tego zadania.

W drugiej grupie programów ewolucyjnych koduje się podziały jako łańcuchy o wymiarze $n + k - 1$ z różnymi liczbami całkowitymi

$(i_1, ..., i_{n+k-1})$

Liczby całkowite z zakresu $\{1, ..., n\}$ reprezentują obiekty, a liczby z zakresu $\{n + 1, ..., n + k - 1\}$ reprezentują separatory. Jest to *permutacja z kodowaniem separatorów*. Na przykład łańcuch 7-elementowy

(1122233)

jest reprezentowany przez łańcuch 9-elementowy

(128345967)

w którym 8 i 9 są separatorami.

Oczywiście trzeba użyć wszystkich $k - 1$ separatorów. Nie mogą się one także pojawiać na pierwszej lub ostatniej pozycji oraz nie mogą się pojawiać razem, jeden obok drugiego (inaczej łańcuch dekodowałby się na mniej niż k grup).

Jak zwykle, należy starannie zaprojektować operatory. Będą one podobne do pewnych operatorów używanych przy rozwiązywaniu zadania komiwojażera, kiedy jest ono reprezentowane przez permutację miast. Mutacja wymienia dwa obiekty (separatory są wyłączone). Rozważono dwa krzyżowania: krzyżowanie z uporządkowaniem (OX) oraz krzyżowanie z częściowym dopasowaniem (PMX) – omawialiśmy je w rozdziale 10. Krzyżowanie jest powtarzane tak długo, aż potomek[1] będzie się dekodował na podział z k grupami.

---

[1] Jones i Beltramo [211] tworzyli tylko jednego potomka w krzyżowaniu.

Ogólnie, wyniki programów ewolucyjnych działających na permutacjach z kodowaniem za pomocą separatorów były lepsze niż dla programów z kodowaniem grup za pomocą liczb. Jednak w żadnym kodowaniu nie używa się w istotny sposób wiedzy specyficznej dla zadania. Można utworzyć trzecią rodzinę programów ewolucyjnych zawierających tę wiedzę w funkcji celu. Tak właśnie zrobiono w [211] w następujący sposób.

Zastosowana reprezentacja była najprostsza: łańcuch n-elementowy reprezentuje n obiektów

$$(i_1, ..., i_n)$$

gdzie $i_j \in \{1, ..., n\}$ oznacza numer obiektu, a więc $i_j \neq i_p$ dla $j \neq p$. Do interpretacji tej reprezentacji użyto algorytmu zachłannego: pierwsze k obiektów w łańcuchu użyto do inicjalizacji k grup, to znaczy każdy z k pierwszych obiektów jest umieszczony w innej grupie. Pozostałe obiekty są dodawane na zasadzie „pierwszy przyszedł – pierwszy wyszedł", to znaczy są one dodawane w kolejności, w jakiej pojawiają się w łańcuchu, a umieszcza się je w grupie, która daje najlepszą wartość funkcji celu.

Ta heurystyka zachłanna upraszcza także operatory. Każda permutacja jest zakodowanym podziałem dopuszczalnym. Możemy więc użyć tych samych operatorów co w zadaniu komiwojażera. Oprócz tego „kodowanie zachłanne" daje wyraźnie lepsze wyniki niż programy ewolucyjne stosujące inne kodowania: grup za pomocą liczb i permutacji z separatorami [211].

Ostatnio Falkenauer [109] zaproponował do rozwiązywania różnych zadań grupowania (podziału) tak zwany *grupujący* algorytm genetyczny. Skupił się on na zaprojektowaniu odpowiedniej reprezentacji chromosomu obejmującej strukturę zadania. W podejściu tym chromosom składa się z dwóch części: *części obiektowej* i *części grupowej*. W części obiektowej korzysta się z kodowania grup za pomocą numerów (omawianego wcześniej w tym punkcie). Składa się ono z łańcuchów całkowitoliczbowych n-wymiarowych

$$(i_1, ..., i_n)$$

gdzie j-ta liczba całkowita $i_j \in \{1, ..., k\}$ oznacza numer grupy przyporządkowanej obiektowi j. Część grupowa chromosomu ma zmienną długość i reprezentuje tylko grupy. Na przykład chromosom

(1 2 1 3 3 3 1 1 2 : 1 2 3)

jest interpretowany następująco. Druga część chromosomu wskazuje, że są trzy grupy (1, 2 i 3). Pierwsza część chromosomu pozwala zinterpretować rozmieszczenie: do grupy 1 należą obiekty $\{1, 3, 7, 8\}$, do grupy 2 obiekty $\{2, 9\}$, a do grupy 3 obiekty $\{4, 5, 6\}$. Zauważmy, że możemy zamienić w powyższej reprezentacji cyfrę „3" na „5" (oczywiście w obu częściach) i rozmieszczenie pozostanie nie zmienione.

Kluczowym pomysłem w tej reprezentacji jest to, że operatory genetyczne pracują na części grupowej (to znaczy drugiej części) chromosomu, a pierwsza część chromosomu jest używana tylko do identyfikacji rozmieszczenia.

Na przykład w zadaniu pakowania pojemników (to znaczy upakowania n obiektów w minimalnej liczbie pojemników o ustalonej pojemności) zaproponowany operator krzyżowania do pakowania pojemników działa następująco. Załóżmy, że dwa chromosomy rodziców są następujące:

(1 2 1 3 3 3 1 1 2 : 1 2 3) oraz

(2 3 3 5 1 4 2 2 6 : 2 3 4 5 6)

Wybiera się (losowo) dwa punkty cięcia w każdej z części grupowych chromosomu, powiedzmy

(1 2 1 3 3 3 1 1 2 : 1 | 2 3 |) oraz

(2 3 3 5 1 4 2 2 6 : 2 | 3 4 | 5 6)

Następnie zawartość wyciętej części pierwszego rodzica wstawia się w pierwszym punkcie cięcia drugiego rodzica (przy tworzeniu drugiego potomka role pierwszego i drugiego rodzica zamienia się)

$(\ldots : 2_2\ 2_1\ 3_1\ 3_2\ 4_2\ 5_2\ 6_2)$

(indeksy oznaczają rodzica). Teraz eliminuje się konflikty. Zauważmy bowiem, że wymienione powyżej zawartości pojemników są następujące:

pojemnik 2_2 – obiekty 1, 7 i 8
pojemnik 2_1 – obiekty 2 i 9
pojemnik 3_1 – obiekty 4, 5 i 6
pojemnik 3_2 – obiekty 2 i 3
pojemnik 4_2 – obiekt 6
pojemnik 5_2 – obiekt 4
pojemnik 6_2 – obiekt 9

Ponieważ w „nowych" pojemnikach (od pierwszego rodzica) oraz „starych" pojemnikach – od drugiego rodzica – znajdują się wspólne obiekty usuwa się te „stare" pojemniki (odpowiedzialne za konflikty) z drugiej części chromosomu, który ma teraz postać

$(\ldots : 2_2\ 2_1\ 3_1)$

Zauważmy, że usunęliśmy pojemnik 6_2 (ponieważ obiekt 9 już jest w 2_1), pojemnik 5_2 (ponieważ obiekt 4 już jest w 3_1), pojemnik 4_2 (ponieważ obiekt

6 już jest w 3_1) i pojemnik 3_2 (ponieważ obiekt 2 już jest w 2_1). Na tym etapie chromosom potomka wygląda następująco:

$$(2_2\ 2_1\ ?\ 3_1\ 3_1\ 3_1\ 2_2\ 2_2\ 2_1 : 2_2\ 2_1\ 3_1)$$

Po zmianie nazwy pojemników 2_2, 2_1 i 3_1 na 1, 2 i 3 chromosom wygląda tak:

(1 2 ? 3 3 3 1 1 2 : 1 2 3)

Zauważmy, że z powodu powyższej metody rozwiązywania konfliktu trzeci obiekt stracił swoje przywiązanie (co zaznaczono znakiem zapytania „?" w pierwszej części chromosomu). Możemy więc użyć pewnego heurystycznego algorytmu naprawy, aby utworzyć osobnika dopuszczalnego. Falkenauer [109] użył heurystyki pierwszego poprawiającego do wstawienia brakującego obiektu. Jeżeli nie ma miejsca dla obiektu 3 w żadnym z trzech pojemników w powyższym chromosomie, to trzeba utworzyć dodatkowy pojemnik

(1 2 4 3 3 3 1 1 2 : 1 2 3 4)

Zauważmy, że takie krzyżowanie przesyła tak dużo znaczącej informacji od obu rodziców, jak to jest tylko możliwe.

W powyższym podejściu operator mutacji był prosty i użyteczny. Wybierał on i eliminował (losowo) kilka pojemników. Obiekty bez przydziału ponownie umieszczano w pojemnikach na zasadzie heurystyki pierwszego poprawiającego (w losowym porządku).

Wyniki eksperymentalne były bardzo dobre. Falkenauer [109] kończy swój artykuł następującą uwagą:

> Mamy także nadzieję, że stworzyliśmy przekonywujący przykład do oceny istotności odpowiedniego kodowania (a w konsekwencji i operatorów genetycznych) dla skutecznego zastosowania paradygmatu algorytmów genetycznych.

Trudno się z tym nie zgodzić.

11.4. Planowanie drogi w środowisku ruchomego robota

Nawigacja jest nauką (lub sztuką) zajmującą się kierowaniem ruchomego robota, gdy porusza się on w swoim środowisku[1]. W każdym schemacie nawigacji dąży się do osiągnięcia punktu przeznaczenia bez zagubienia się lub kolizji z jakimkolwiek obiektem.

Często drogę dla robota planuje się zawczasu. Prowadzi ona robota do celu przy założeniu, że środowisko jest dokładnie znane i nie zmienia się,

[1] Oczywiście jest to definicja zawężona do potrzeb tego punktu (przyp. tłum.).

a robot może dokładnie podążać po wyznaczonej drodze. Początkowe programy wyznaczania drogi wykonywały właśnie takie planowanie zawczasu lub mogły tylko korzystać z takich planów (na przykład [248], [218], [205]). Jednak ograniczone zastosowanie planowania zawczasu spowodowały, że zajęto się badaniami związanymi z planowaniem na bieżąco, w którym korzysta się z wiedzy zdobytej w trakcie badania lokalnego otoczenia [222], aby ominąć nieznane przeszkody w trakcie poruszania się w środowisku.

Program ewolucyjny, który tutaj opiszemy, to znaczy nawigator ewolucyjny, łączy planowanie zawczasu i planowanie na bieżąco z użyciem prostej mapy o wysokiej jakości i wydajnego algorytmu planowania [243], [244]. W pierwszej części algorytmu (planowania zawczasu) poszukuje się globalnie optymalnej drogi od punktu startowego do punktu przeznaczenia. Natomiast druga część (planowanie na bieżąco) jest odpowiedzialna za rozwiązywanie możliwych sytuacji kolizyjnych z nieznanymi wcześniej obiektami przez zamianę części początkowej drogi na optymalną poddrogę. Należy zaznaczyć, że obie części nawigatora ewolucyjnego używają tego samego algorytmu ewolucyjnego, tylko z innymi wartościami parametrów.

Przez ostatnich pięć lat różni badacze eksperymentowali z metodami obliczeń ewolucyjnych dla zadania planowania drogi. Davidor [69] użył dynamicznych struktur chromosomów i zmodyfikowanego operatora krzyżowania do optymalizacji pewnych rzeczywistych procesów (włącznie z zastosowaniem do wyznaczania drogi robotów). W [355] opisano algorytm genetyczny dla zadania planowania drogi, a w [356] algorytm genetyczny dla wyznaczenia strategii wieloheurystycznego poszukiwania w czasie rzeczywistym. W obu podejściach zakłada się, że istnieje mapa zawierająca węzły. Inni użyli do planowania drogi systemów klasyfikujących [414] lub paradygmatu programowania genetycznego [176]. Nasze podejście jest wyjątkowe w tym sensie, że nawigator ewolucyjny: (1) działa w całej wolnej przestrzeni i nie przyjmuje żadnych apriorycznych założeń o dopuszczalnych węzłach drogi, (2) łączy razem algorytmy planowania zawczasu i na bieżąco.

Zanim objaśnimy szczegółowo algorytm, zajmijmy się najpierw strukturą mapy. Aby wspomóc poszukiwanie drogi w całej wolnej przestrzeni ciągłej do reprezentacji obiektów występujących w środowisku, używa się grafów wierzchołkowych. Obecnie ograniczymy się do środowiska dwuwymiarowego z obiektami tylko wielokątnymi oraz ruchami robota tylko po liniach. Wobec tego robot ulega „ściśnięciu" do punktu, gdy tymczasem obiekty w środowisku odpowiednio „urosną" [248]. Zakłada się, że mamy do czynienia z ruchomym robotem wyposażonym w czujnik ultrasoniczny (na przykład z robotem Denninga). Znany obiekt jest reprezentowany przez uporządkowaną (zgodnie z ruchem wskazówek zegara) listę jego wierzchołków. Spotkane na bieżąco przeszkody są modelowane jako kawałki „ścian", przy czym każdy kawałek „ściany" jest odcinkiem linii prostej i jest reprezentowany przez jego dwa końcowe punkty. Ta reprezentacja jest zgodna z reprezentacją znanych obiektów, a jednocześnie uwzględnia fakt, że z badania ze szczególnego miejsca można uzys-

kać tylko częściową informację o nieznanej przeszkodzie. Na koniec, całe środowisko jest określone jako powierzchnia prostokąta.

Należy teraz określić drogi, jakie może utworzyć nawigator ewolucyjny. Droga składa się z jednego lub więcej odcinków linii prostej z punktem początkowym i punktem docelowym, a także (o ile jest to możliwe) miejsc przecięć dwóch sąsiednich odcinków, określających *węzły*. Droga dopuszczalna składa się z dopuszczalnych węzłów, droga niedopuszczalna zawiera przynajmniej jeden węzeł niedopuszczalny. Przypuśćmy, że mamy drogę $p = (m_1, m_2, ..., m_n)$, $n \geqslant 2$, gdzie m_1 i m_n oznaczają odpowiednio węzeł początkowy i węzeł docelowy. Węzeł m_i $(i = 1, ..., n-1)$ jest niedopuszczalny, jeżeli nie można go połączyć z następnym węzłem m_{i+1} z powodu występującej przeszkody albo jest umieszczony wewnątrz (lub zbyt blisko) pewnej przeszkody. Założymy, że węzły początkowy i docelowy znajdują się poza przeszkodami i nie za blisko nich. Zauważmy jednak, że punkt początkowy nie musi być dopuszczalny (może nie być możliwe połączenie go z następnym węzłem), natomiast węzeł docelowy jest zawsze dopuszczalny. Zauważmy także, że różne drogi mogą składać się z różnej liczby węzłów.

Jesteśmy teraz gotowi prześledzić procedurę nawigatora ewolucyjnego (rys. 11.1). Nawigator ewolucyjny najpierw wczytuje mapę i miejsca punktów początkowego i docelowego w zadaniu. Następnie algorytm ewolucyjny wyznaczania drogi zawczasu (WDZ) generuje drogę globalną bliską optymalnej. Jest to droga z kawałków linii prostych łączących dopuszczalne węzły. Na rysunku 11.2 pokazano taką drogę globalną wyznaczoną zawczasu przez nawigator ewolucyjny. (Zaczernione kółko oznacza robota).

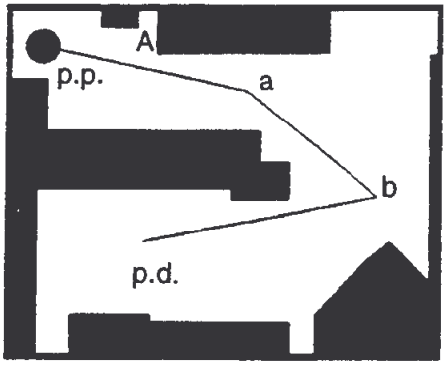
Kiedy robot zaczyna poruszać się po drodze w kierunku punktu docelowego, bada otoczenie do najbliższych obiektów i algorytm ewolucyjny pracujący na bieżąco (WDB) generuje lokalne drogi, aby uniknąć niespodziewanych kolizji i obiektów. Do symulacji efektu związanego z nieznanymi obiektami w środowisku utworzono dodatkowe pliki danych reprezentujące te przeszkody (jako kawałki „ścian", jak wyjaśniono wcześniej). Eksperymentowano z pięcioma różnymi zbiorami nieznanych obiektów. Na rysunku 11.3a-d przedstawiono ruchy robota dla jednego z takich zbiorów.

Kiedy robot zbliżył się zbytnio do dolnego lewego rogu pobliskiego obiektu „A", algorytm ewolucyjny działający na bieżąco wewnętrznie „powiększył" na miejscu „A" i wygenerował lokalną drogę, też składającą się z kawałków linii prostych, aby odsunąć się od „A". Następnie robot przebył bez przeszkód nową drogę do punktu „a". Kiedy poruszał się z „a" do „b", wykrył nieznany nowy obiekt „B". Teraz nawigator ewolucyjny zaktualizował mapę i znowu algorytm ewolucyjny działający na bieżąco wygenerował lokalną drogę z węzłem „d" (rys. 11.3a). Kiedy robot poruszał się z „d" w kierunku „b", znalazł się zbyt blisko obiektu „B". W rezultacie wygenerował inną lokalną drogę reprezentowaną przez węzeł „e" (rys. 11.3b). Robot przejechał wtedy z „d" do „e", a następnie osiągnął cel częściowy „b". W następnym kroku powinien poruszyć się z „b" do punktu docelowego. Jak pokazano na rys. 11.3c, ten

fragment trasy był zbyt bliski dolnego prawego rogu obiektu „C". Wobec tego wygenerowano inną lokalną drogę reprezentowaną węzłem „f", skąd robot dostał się do punktu docelowego. Na rysunku 11.3d pokazano początkową drogę globalną oraz drogę przebytą w rzeczywistości. Zauważmy, że proces

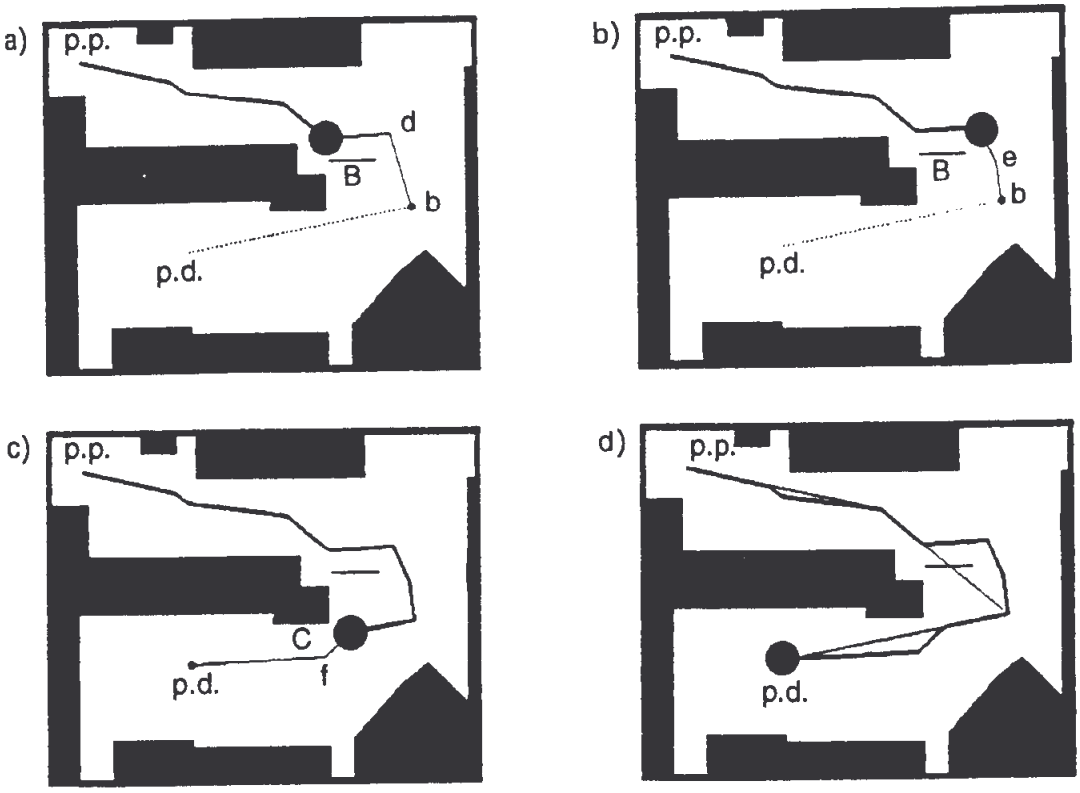
```
procedure nawigator ewolucyjny
begin
   begin (planowanie zawczasu)
      wczytaj mapę
      wczytaj zadanie
      wykonaj planowanie
         bieżąca droga := WDZ(punkt początkowy, punkt docelowy)
   end (planowanie zawczasu)
   if bieżąca droga jest dopuszczalna then
   begin (planowanie na bieżąco)
   repeat
      przesuwaj się po bieżącej drodze badając otoczenie
      if zbyt blisko obiektu then
      begin
         start_lokalny := obecne miejsce
         cel_lokalny := następny węzeł bieżącej drogi
         if obiekt jest nowy
            then nanieś obiekt na mapę
            else wewnętrznie powiększ obiekt w najbliższym punkcie
         wykonaj planowanie:
               droga_lokalna := WDB(start_lokalny, cel_lokalny)
         uaktualnij bieżącą drogę
      end
   until (w punkcie docelowym) or (warunek niepowodzenia)
   end (planowanie na bieżąco)
end
```

Rys. 11.1. Struktura nawigatora ewolucyjnego



Rys. 11.2. Środowisko i droga globalna

nawigacji kończy się, kiedy robot osiąga punkt docelowy lub zgłasza niepowodzenie, to znaczy kiedy nawigator ewolucyjny nie może znaleźć dopuszczalnej drogi w pewnym odcinku czasu (to znaczy w ramach zadanej liczby pokoleń algorytmu ewolucyjnego działającego na bieżąco).



Rys. 11.3. Drogi przebyte rzeczywiście

Jak wspomnieliśmy wcześniej, nawigator ewolucyjny łączy planowanie zawczasu i planowanie na bieżąco w ramach tych samych struktur danych i tego samego algorytmu planowania. To znaczy jedyną różnicą między algorytmem ewolucyjnym wyznaczania drogi zawczasu i wyznaczania na bieżąco są parametry, jakich one używają: liczebności populacji pop_size_g, liczby pokoleń T_g, maksymalnej długości chromosomu n_g itp. dla pierwszego oraz pop_size_l, T_l, n_l itp. dla drugiego. Zauważmy, że oba one wykonują *planowanie globalne*, nawet jeżeli algorytm na bieżąco zazwyczaj generuje drogę lokalną, to operuje on na uaktualnionej *mapie globalnej*. Ponadto, jeżeli nie jest znany na początku żaden obiekt w środowisku lub nie ma żadnego obiektu między punktem początkowym a docelowym, to algorytm wyznaczania drogi zawczasu wygeneruje drogę prostoliniową z tylko dwoma węzłami: punktem początkowym i punktem docelowym. Wtedy doprowadzenie robota do punktu docelowego z uniknięciem nieznanych przeszkód całkowicie zależy od algorytmu działającego na bieżąco.

Obecnie omówimy szczegółowo części składowe obu algorytmów.

Chromosomy są uporządkowaną listą węzłów drogi, jak pokazano na rys. 11.4. Każdy z węzłów drogi, poza wskaźnikiem następnego węzła, zawiera

w opisie współrzędne x i y pośredniego węzła wzdłuż drogi oraz zmienną binarną b, która wskazuje, czy dany węzeł jest dopuszczalny, czy nie.

Długość chromosomów (liczba węzłów drogi zapisanych w chromosomie) jest zmienna. Przy planowaniu zawczasu maksymalna długość chromosomu jest ustalona na n_g – liczbę wierzchołków reprezentujących znane obiekty w środowisku. Jest nieprawdopodobne, aby którakolwiek dopuszczalna droga wymagała większej liczby pośrednich węzłów. Nawet w skomplikowanym środowisku droga dopuszczalna może być całkiem prosta. Dlatego przyjmuje się, że długość chromosomów jest zmienna, aby elegancko wyjść z takich sytuacji.

Rys. 11.4. Chromosom reprezentujący węzeł

W czasie planowania na bieżąco droga lokalna naokoło przeszkody zazwyczaj będzie zawierała małą liczbę węzłów i w rezultacie parametr n_l, maksymalna długość chromosomu w tej fazie, jest stosunkowo mały na początku przeszukiwania. Jednak, jeżeli proces ewolucyjny nie znajdzie dopuszczalnej drogi po kilku liczbach pokoleń, to maksymalna długość chromosomu powinna zostać powiększona, gdyż jest wtedy całkiem prawdopodobne, że dopuszczalna droga ma bardziej skomplikowaną strukturę. W nawigatorze ewolucyjnym założono, że parametr n_l zależy od numeru bieżącego pokolenia, a dokładniej, że $n_l(t) = t$.

Początkowe populacje chromosomów (pop_size_g dla algorytmu ewolucyjnego wyznaczania drogi zawczasu i pop_size_l dla wyznaczania na bieżąco) generuje się losowo. Dla określenia długości każdego chromosomu generuje się liczbę losową z zakresu 2, ..., $max(2, n_g)$ (dla planowania zawczasu). Współrzędne x i y dla każdego węzła chromosomu tworzy się także losowo (wartości współrzędnych są oczywiście ograniczone wymiarami środowiska).

Dla każdego węzła w każdym chromosomie określa się wartość zmiennej binarnej b (sprawdzanie dopuszczalności). Jeżeli węzeł jest dopuszczalny, to wartość b przyjmuje się jako PRAWDA, jeżeli nie, to jako FAŁSZ. Metody sprawdzania dopuszczalności węzła (to znaczy dopuszczalności miejsca, dostatecznej odległości od najbliższych obiektów i możliwości połączeń) są stosunkowo proste i opierają się na algorytmach opisanych przez Pavlidisa [310].

Dopasowanie (koszt całej trasy) chromosomu $p = (m_1, m_2, ..., m_n)$ określa się za pomocą dwóch oddzielnych funkcji oceny (dla osobników dopuszczalnych i niedopuszczalnych):

- dla dopuszczalnych dróg p:

$$\text{Path_Cost}(p) = w_d\, dist(p) + w_s\, smooth(p) + w_c\, clear(p)$$

gdzie wagi w_d, w_s i w_c normalizują cały koszt drogi, a:

– $dist(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1})$, gdzie $d(m_i, m_{i+1})$ jest odległością między węzłami m_i oraz m_{i+1}, to znaczy funkcja $dist(p)$ podaje całkowitą długość drogi p,

– $smooth(p) = \max_{i=2}^{n-1} s(m_i)$, gdzie

$$s(m_i) = \frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}}$$

to znaczy funkcja $smooth(p)$ podaje największą krzywiznę p w węźle,

– $clear(p) = \max_{i=1}^{n-1} c_i$, gdzie

$$c_i = \begin{cases} d_i - \tau, & \text{jeżeli } d_i \geqslant \tau \\ a(\tau - d_i) & \text{w odwrotnym przypadku} \end{cases}$$

d_i jest najmniejszą odległością między odcinkiem (m_i, m_{i+1}) drogi a wszystkimi znanymi obiektami, τ określa odległość bezpieczną, a zaś jest współczynnikiem, to znaczy funkcja $clear(p)$ podaje największą z odległości między wszystkimi odcinkami p oraz obiektami,

- dla niedopuszczalnej drogi p:

$$\text{Path_Cost}(p) = \alpha + \beta + \gamma$$

gdzie α jest liczbą przecięć drogi p ze wszystkimi ścianami obiektów, β jest średnią liczbą przecięć w odcinku niedopuszczalnym, a γ podaje koszt najgorszej dopuszczalnej drogi w bieżącej populacji; ze względu na tę ostatnią zmienną każda dopuszczalna droga w populacji jest lepsza od każdej niedopuszczalnej (patrz także p. 15.3, część C).

Algorytmy wyznaczania drogi zawczasu i na bieżąco zawierają kilka operatorów (krzyżowanie, dwie mutacje, wstawianie, usuwanie, wygładzanie i wymiana). Omówimy je po kolei.

Krzyżowanie

Ten operator jest podobny do klasycznego krzyżowania jednopunktowego szeroko stosowanego w algorytmach genetycznych. Łączy on „dobre" części drogi od obu rodziców, aby ewentualnie utworzyć lepszą drogę w potomku. Dwa wybrane chromosomy są rozcinane w pewnym miejscu i łączone ze sobą: pierwsza część pierwszego chromosomu z drugą częścią drugiego chromosomu, a pierwsza część drugiego chromosomu z drugą częścią pierwszego chromosomu. Jednak punktów cięcia nie wybiera się losowo. Jeżeli chromosom zawiera niedopuszczalne węzły, to punkt cięcia wypada za jednym z nich.

Mutacja 1

Ta mutacja odpowiada za dokładne dostrojenie współrzędnych węzłów występujących w chromosomie. Jeżeli węzeł w chromosomie wybrano do tej mutacji, to modyfikuje się jego współrzędne. Na przykład współrzędną $x \in [a, b]$ (tak samo jak i współrzędną y) zmienia się w następujący sposób:

$$x' = \begin{cases} x - \delta(t, x - a), & \text{jeżeli } r = 0 \\ x + \delta(t, b - x), & \text{jeżeli } r = 1 \end{cases}$$

gdzie r jest losowym bitem, a funkcja $\delta(t, z)$ podaje wartość z zakresu $[0, z]$ taką, że prawdopodobieństwo, iż $\delta(t, z)$ jest bliskie zeru, wzrasta ze wzrostem t (t jest bieżącym numerem pokolenia w procesie ewolucyjnym). Ten operator jest podobny do mutacji niejednorodnej używanej w systemach ewolucyjnych do optymalizacji nieliniowej (rozdz. 7). Mutacja 1 odpowiada za „wygładzanie" kształtu drogi.

Mutacja 2

Jest użyteczna w przypadkach, gdy jest potrzebna większa zmiana wartości (taka sytuacja zdarza się często w fazie planowania na bieżąco, kiedy przeszkoda blokuje drogę). Jeżeli węzeł chromosomu wybrano do tej mutacji, to modyfikuje się jego współrzędne. Na przykład współrzędną $x \in [a, b]$ (tak samo jak i współrzędną y) zmienia się w następujący sposób:

$$x' = \begin{cases} x - \Delta(t, x - a), & \text{jeżeli } r = 0 \\ x + \Delta(t, b - x), & \text{jeżeli } r = 1 \end{cases}$$

gdzie r jest losowym bitem, a funkcja $\Delta(t, z)$ podaje wartość z zakresu $[0, z]$ taką, że prawdopodobieństwo, iż $\Delta(t, z)$ jest bliskie zeru, wzrasta ze wzrostem numeru pokolenia t.

Wstawianie

Ten operator wstawia nową krawędź do istniejącej drogi. Każde miejsce między dwoma węzłami ma jednakowe prawdopodobieństwo takiego wstawienia.

Usuwanie

Ten operator usuwa węzeł z drogi. Każdy węzeł ma jednakowe prawdopodobieństwo takiego usunięcia.

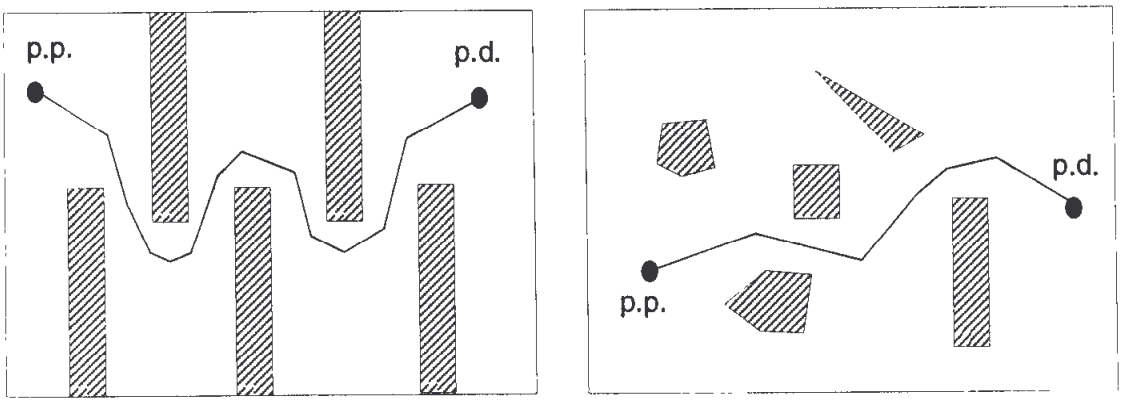
Wygładzanie

Ten operator wygładza część drogi, wycinając ostre zakręty. Dla wybranego węzła m_i (z dużą krzywizną) operator wybiera dwa nowe węzły k_1 i k_2 (odpowiednio z odcinków (m_{i-1}, m_i) oraz (m_i, m_{i+1})) i wstawia je do drogi oraz usuwa m_i, a więc tworzy nową drogę p':

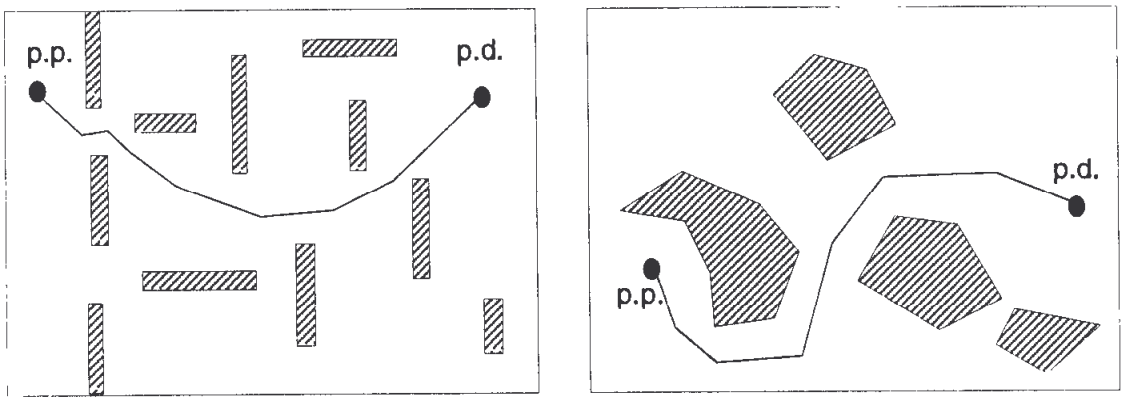
$$p' = (m_1, \ldots, m_{i-1}, k_1, k_2, m_{i+1}, \ldots, m_n)$$

Wymiana
Ten operator dzieli chromosom na dwie części (punkt podziału jest generowany losowo) i zamienia te części.

We wstępnych obliczeniach nawigator ewolucyjny wykazał się wydajnością i efektywnością w porównaniu z nawigatorami używającymi tradycyjnych sposobów (na przykład w [130]). Wyniki obecnej wersji systemu dla dwóch różnych środowisk przedstawiono na rys. 11.5 i 11.6.



Rys. 11.5. Wyniki uzyskane przez nawigatora ewolucyjnego dla dwóch środowisk



Rys. 11.6. Wyniki uzyskane przez nawigatora ewolucyjnego dla innych dwóch środowisk

Oczywiście należy badać możliwości nawigatora ewolucyjnego, prowadząc więcej testów z różnymi środowiskami, a co ważniejsze, z rzeczywistym robotem. Jednocześnie pozostało do rozwiązania kilka spraw związanych z ewolucyjnymi nawigatorami. Obejmują one: (1) zaprojektowanie sprytniejszych warunków zakończenia dla algorytmów wyznaczania drogi zawczasu i na bieżąco, aby mogły one lepiej realizować cele optymalizacji (obecnie algorytmy zatrzymują się albo wtedy, kiedy znajdą dopuszczalną drogę, albo po upływie ustalonej liczby pokoleń), (2) wprowadzenie adaptacyjnych częstości stosowania operatorów genetycznych, w przeciwieństwie do stałych w obecnej wersji systemu (ta modyfikacja powinna poprawić działanie systemu; jest ona oparta na prostej obserwacji, że różne operatory mogą odgrywać różną rolę w róż-

nych etapach procesu ewolucyjnego), (3) rozszerzenie nawigatora ewolucyjnego, aby działał w środowisku z obiektami niewielokątnymi, (4) włączenie wiedzy o bieżącym stanie przeszukiwania do pracy operatorów (na przykład może byłoby lepiej krzyżować dwie drogi w niedopuszczalnych węzłach), (5) badanie pewnych mechanizmów uczenia się, aby nawigator ewolucyjny mógł korzystać z przeszłych doświadczeń.

Nawigator ewolucyjny, mimo wydajności i efektywności w wielu zadaniach, ma jedno duże ograniczenie: zakłada się w nim, że dopuszczalna i dostatecznie dobra droga może być osiągnięta przez małe zakłócenie najlepszej obecnie drogi. System nie jest zaprojektowany tak, aby mógł na pewnym etapie drogi zamienić całkowicie bieżącą drogę globalną na inną (być może lepszą) drogę globalną. Może więc warto poeksperymentować z innymi rozwiązaniami, na przykład jest obecnie opracowywany nawigator adaptacyjny. Nawigator adaptacyjny nie będzie się składał z procedur planowania drogi zawczasu i na bieżąco, jak nawigator ewolucyjny, ale będzie miał tylko procedurę planowania drogi na bieżąco. Będzie ona ciągle poprawiała drogę od bieżącego miejsca, w którym znajduje się robot, do punktu docelowego, opierając się na nowo zebranej informacji z czujników.

11.5. Uwagi

W tym końcowym punkcie omówimy krótko kilka stosunkowo nowych zastosowań metod ewolucyjnych, które są, z różnych przyczyn, ciekawe (pod względem budowy programu ewolucyjnego). Omówimy je po kolei.

Są pewne zastosowania (na przykład zadanie projektowania sieci), w których rozwiązaniem jest graf. Zadanie reprezentowania grafu w algorytmie genetycznym jest zresztą całkiem interesujące samo w sobie. Ostatnio Palmer i Kershenbaum [304] opisali eksperymenty z różnymi sposobami reprezentowania drzew. Określili oni pożądane właściwości dobrej reprezentacji. Obejmują one:

1) kompletność: możliwość reprezentowania wszystkich występujących drzew,
2) brak nadmiarowości: wszystkie drzewa powinny być reprezentowane przez tę samą liczbę kodowań,
3) rozsądek: reprezentowanie tylko drzew,
4) wydajność: łatwość przejścia od zakodowanej reprezentacji drzewa do reprezentacji w bardziej konwencjonalnej formie, umożliwiającej obliczanie funkcji dopasowania i sprawdzanie ograniczeń,
5) lokalność: małe zmiany reprezentacji drzewa powodują małe zmiany drzewa.

Oczywiście idealna reprezentacja drzewa powinna spełniać wszystkie powyższe warunki. Jednak nie jest tak w większości reprezentacji. W [304] autorzy rozważyli kilka reprezentacji. Objęło to:

- reprezentację w postaci wektora charakterystycznego, kiedy drzewo jest reprezentowane za pomocą wektora binarnego o długości równej liczbie krawędzi związanego z nim grafu,
- reprezentację poprzedzającą, kiedy węzeł jest określany przez pozycję, a zapisuje się w niej węzeł poprzedzający, drzewo jest tutaj zakodowane w postaci wektora całkowitoliczbowego o długości równej liczbie węzłów,
- reprezentację za pomocą liczb Prüfera, kiedy drzewo jest zakodowane jako liczba $(n-2)$-cyfrowa (n jest liczbą węzłów w drzewie), przy czym każda cyfra jest liczbą całkowitą wyznaczaną przez specjalny algorytm (szczegóły można znaleźć w [304]).

Autorzy zaproponowali także nową reprezentację, która wywodzi się z prostego spostrzeżenia, że pewne węzły są węzłami wewnętrznymi, a inne liśćmi. W tej reprezentacji chromosom dla każdego węzła i możliwej krawędzi jest nadmiarowy (a więc drzewo jest reprezentowane jako wektor $n + n(n-1)/2$ liczb). Z powodu nadmiarowości trzeba zmodyfikować macierz kosztów grafu

$$C'_{ij} = C_{ij} + P_1(C_{max})b_{ij} + P_2(C_{max})(B_i + b_j)$$

gdzie P_1 i P_2 są parametrami metody, a C_{max} jest maksymalnym kosztem połączenia. Drzewo reprezentowane przez chromosom znajduje się za pomocą algorytmu Prima znajdowania minimalnego drzewa rozpiętego na danych węzłach na podstawie macierzy kosztu reprezentacji nadmiarowej. W tej reprezentacji można także zakodować dowolne drzewo, znając odpowiednie wartości b_i. Ma ona całkiem ciekawe właściwości. Pełne jej omówienie można znaleźć w [304].

W innym artykule Abuali i in. [3] badali nowy schemat kodowania dla reprezentacji drzew rozpiętych (dla stochastycznego zadania minimalnego drzewa rozpiętego). Ten schemat kodowania jest oparty na tak zwanych *kodach wyznacznikowych*, które są wektorami składającymi się z $n-1$ liczb całkowitych, a liczba k na pozycji i odpowiada krawędzi z wierzchołka k do $i+1$[1]. Wyniki eksperymentalne i porównanie tego kodowania z innymi metodami można znaleźć w [3].

Esbensen [101] omawia algorytm genetyczny do znajdowania optymalnych drzew Steinera. Zadanie drzew Steinera (jego wersja decyzyjna jest NP-kompletna) formułuje się jak następuje: dla zadanego grafu i podanego podzbioru wierzchołków wyznaczyć podgraf o najmniejszym koszcie rozpięty na podanych wierzchołkach. Autor użył deterministycznego dekodera, który interpretuje dowolny zbiór wybranych wierzchołków Steinera jako dopuszczalne drzewo Steinera. Tak więc każdy chromosom jest łańcuchem binarnym, w którym każdy bit odpowiada wierzchołkowi, czyli taki łańcuch binarny reprezentuje

[1] Schematy kodowania wyznacznikowego mogą tworzyć grafy, które nie są drzewami rozpiętymi na zadanych wierzchołkach. W pracy omawianej w [3] użyto algorytmów naprawy.

podzbiór wierzchołków Steinera[1]. Ten podzbiór wierzchołków łącznie z podanymi wierzchołkami jest punktem startowym dla dekodera, który: (1) tworzy podgraf związany z tymi wierzchołkami, (2) wyznacza minimalne drzewo rozpięte na tym podgrafie, (3) tworzy (na podstawie minimalnego rozpiętego drzewa) inny podgraf przez zamianę każdej krawędzi na odpowiednią najkrótszą ścieżkę w grafie początkowym, (4) wyznacza minimalne drzewo rozpięte na wynikłym z tego podgrafie, (5) wyznacza drzewo Steinera przez powtarzane wyrzucanie (z ostatniego minimalnego rozpiętego drzewa) wszystkich wierzchołków rzędu 1 nie znajdujących się na początkowej liście wierzchołków. Wyniki eksperymentalne (obejmujące przypadki grafów mających do 2500 wierzchołków) znajdują się w [101].

Zadanie pokrycia zbioru jest zadaniem polegającym na pokryciu wierszy macierzy binarnej $A = (a_{ij})$ o wymiarach $n \times k$ podzbiorem kolumn[2] o minimalnym koszcie. Zadanie to ma wiele praktycznych zastosowań (rozmieszczanie urządzeń, przydział zadań załodze itp.). Każdej kolumnie $1 \leqslant j \leqslant k$ jest przyporządkowany koszt c_j, wobec tego zadanie pokrycia zbioru można zapisać w postaci

$$\min \sum_{j=1}^{k} c_j x_j$$

gdzie x_j oznacza binarną zmienną decyzyjną ($x_j = 1$ wtedy i tylko wtedy, gdy wybiera się kolumnę j) przy ograniczeniu

$$\sum_{j=1}^{k} a_{ij} x_j \geqslant 1$$

dla wszystkich $1 \leqslant i \leqslant n$.

Beasley [31] eksperymentował ze zmodyfikowanym algorytmem genetycznym na wielu zadaniach pokrycia zbioru, począwszy od zadań o 200 wierszach i 1000 kolumn do zadań o 1000 wierszy i 10 000 kolumn. Wyniki były bardzo dobre. Algorytm znajdował optymalne rozwiązania dla zadań o mniejszych wymiarach oraz wysokiej jakości rozwiązania dla zadań o większych wymiarach.

Zauważmy, że zadanie pokrycia zbioru ma „idealną" reprezentację dla algorytmów genetycznych: łańcuch binarny z x-ów ($1 \leqslant j \leqslant k$) reprezentuje potencjalne rozwiązanie zadania. Nie ma potrzeby dodatkowego kodowania. Funkcja oceny (dla osobników dopuszczalnych) jest także prosta, jest to

$$f(\mathbf{x}) = \sum_{j=1}^{k} c_j x_j$$

---

[1] Wierzchołki Steinera składają się z dodatkowych wierzchołków, które należy dodać do podanego zbioru. A więc długość chromosomu jest określona przez różnicę między całkowitą liczbą węzłów a liczbą węzłów zadanych.

[2] To znaczy znalezieniu takiego zestawu kolumn, że w każdym wierszu znajduje się jedynka (przyp. tłum.).

Największym wyzwaniem w zadaniu pokrycia zbioru jest sprawa dopuszczalności. Jakikolwiek operator działający na takich łańcuchach binarnych może utworzyć potomka, który narusza ograniczenia zadania (na przykład pewne wiersze mogą nie mieć pokrycia). Beasley [31] do utrzymania dopuszczalności rozwiązań użył algorytmu naprawy (patrz rozdz. 15, gdzie znajduje się ogólne omówienie algorytmów naprawy). Ten algorytm naprawy miał za zadanie pokrycie wierszy, które jeszcze nie były pokryte. Poszukiwanie tych dodatkowych kolumn było oparte na stosunku kosztu kolumny do liczby wierszy bez pokrycia, które kolumna pokryje. Warto zauważyć, że kiedy proces naprawy jest zakończony i osobnik niedopuszczalny jest zamieniony na dopuszczalnego, wtedy wykonuje się krok lokalnej optymalizacji, w którym usuwa się zbędne kolumny (to znaczy kolumny, które można usunąć bez naruszenia ograniczeń).

Bui i Eppley [55] opisali hybrydowy algorytm genetyczny dla zadania maksymalnej kliki i podali wyniki obliczeń dla zadań testowych zawierających do 1500 wierzchołków i ponad pół miliona krawędzi. Zadanie maksymalnej kliki polega na wyznaczeniu zamkniętego podgrafu (czyli kliki) o maksymalnym wymiarze (mierzonym liczbą wierzchołków) w danym grafie. Wersja decyzyjna tego zadania jest NP-trudna [134] i wobec tego zaproponowano dla niej wiele różnych metod przybliżonych. Zauważmy, że zadanie polega na wyznaczeniu podzbioru wierzchołków, można więc użyć bezpośredniej reprezentacji binarnej: wektor binarny

$$(b_1, \ldots, b_n)$$

opisuje taki podzbiór (dla n wierzchołków ustawionych w dowolnym porządku[1)], przy czym $b_i = 1$ oznacza wybranie i-tego węzła). Zamiast określać skomplikowane operatory, które utrzymywałyby dopuszczalność rozwiązań (to znaczy, które gwarantowałyby, że potomek jest kliką), lub zamiast tego tworzyć algorytmy naprawy (które sprowadzałyby dowolne rozwiązanie do kliki), autorzy utworzyli sprytną funkcję celu, która rozróżnia nie-kliki od (prawie) klik. Ma ona postać

$$f(X) = \alpha|X| + \beta \frac{2e(X)}{|X|(|X|-1)}$$

gdzie α i β są liczbami całkowitymi (zwanymi odpowiednio wagami podstawowymi i wagami kończącymi), a $e(X)$ podaje liczbę krawędzi w podgrafie związanym z X. Warto zauważyć, że współczynnik β/α zmienia się. Na początku algorytmu jest on mały (faza badania, w której kładzie się nacisk na ogólność rozwiązania), a w trakcie jego zaawansowania wzrasta (nacisk na dokończenie

[1)] Jednak w [55] autorzy prowadzili obliczenia z dwoma różnymi etapami wstępnymi, które porządkowały wierzchołki albo w malejącym porządku ich rzędów, albo przyjmowały rząd wynikający z głębokości pierwszego przeszukiwania.

rozwiązania). Warto podkreślić, że ten algorytm rozszerzono przez dołączenie procedury lokalnej optymalizacji. Ta procedura heurystyczna: (1) określa, czy usunięcie pewnego wierzchołka (są one przeglądane w specjalnej kolejności) nie poprawia dopasowania rozwiązania (jeżeli tak, to wierzchołek jest usuwany) i (2) próbuje zwiększyć ogólność rozwiązania, badając, czy dodanie wierzchołka nie zwiększa wartości dopasowania rozwiązania (jeżeli tak, to wierzchołek jest dodawany). Pełne omówienie szczegółów systemu i wyników obliczeń można znaleźć w [55].

Ciekawe zastosowanie metod ewolucyjnych do ładowania kontenerów opisał Juliff [215]. Zadanie ładowania kontenerów jest specjalnym typem zadania szeregowania, w którym występuje: (1) pakowanie kartonów do kontenerów, (2) ustawianie kontenerów w ciężarówce. Wymagania są dosyć skomplikowane w związku z koniecznością utrzymania odpowiedniego rozłożenia ładunku (przez cały czas) podczas rozwożenia oraz maksymalizacji wydajności ładowania i rozładowywania kartonów. Tak jak dla zadań szeregowania (patrz p. 11.1) można zbudować system z reprezentacjami bezpośrednimi lub pośrednimi. W tej ostatniej zadanie wykona procedura szeregowania (a w tym przypadku ustawiania ładunku). Jednak oba te podejścia mają pewne wady: operatory są mocno uzależnione od zadania (reprezentacja bezpośrednia) lub występuje ograniczona możliwość przeszukiwania (reprezentacja pośrednia). W tej ostatniej chromosom zazwyczaj reprezentuje jedną cechę, a inne cechy nie są bezpośrednio reprezentowane (wiedza o tych dodatkowych cechach jest po prostu wbudowana w procedurę ustawiania ładunku), więc przeszukiwanie genetyczne nie korzysta z pełnej informacji.

Juliff opracował kilka systemów ładowania kontenerów [215] z reprezentacją bezpośrednią i inteligentnymi procedurami ustawiania ładunku. W jednym z nich przyjął strukturę wielochromosomową, aby objąć różne cechy zadania (dokładnie były to trzy chromosomy). I oczywiście system wielochromosomowy działał lepiej niż inne, jednochromosomowe systemy.

Ostatnio przedstawiono wiele interesujących zastosowań metod ewolucyjnych w różnych zadaniach zarządzania (łącznie z zadaniami szeregowania i układania planu lekcji). Całościowe opracowanie zagadnień związanych z takimi zastosowaniami zawiera niedawna praca Nissena [297].

Występuje także rosnące zainteresowanie metodami ewolucyjnymi w inżynierii przemysłowej. Wiele artykułów omawia poszczególne zagadnienia i podaje opisy szczegółowych rozwiązań w tej dziedzinie. Więcej informacji na ten temat można na przykład znaleźć w specjalnym wydaniu *Computers and Industrial Engineering Journal* [135] poświęconym algorytmom genetycznym i inżynierii przemysłowej.

Kończymy ten rozdział ogólną uwagą, że wciąż więcej i więcej różnych zastosowań metod ewolucyjnych w zadaniach optymalizacji kombinatorycznej (łącznie z zadaniem komiwojażera, patrz rozdz. 10) zawiera pewne heurystyczne algorytmy lokalnego przeszukiwania poprawiające ich działanie (wspaniały przegląd metod lokalnego przeszukiwania obejmujący najnowsze osiągnięcia

znajduje się w [2]). Jest to rozwiązywane albo w wydzielonych krokach algorytmu, albo przez tworzenie ich sprytnych (inteligentnych) mutacji. Wyniki opisane przez Yagiurę i Ibarakiego w ich ostatniej pracy [411] wykazują, że takie algorytmy genetyczne lokalnego przeszukiwania są silnymi metodami, które wypadają bardzo dobrze w porównaniach z innymi metodami (losowym przeszukiwaniem lokalnym z wielokrotnym punktem startowym lub zwykłymi algorytmami genetycznymi).

Wyniki te podkreślają także wagę operatorów mutacji, które nie powinny być tylko operatorami pomocniczymi, ale zasadniczymi składnikami każdego systemu ewolucyjnego. Jest to zgodne (w pewnym sensie) z ostatnimi wynikami Jonesa [213], który eksperymentował z makromutacjami i wykazał ich istotność w kodowaniu binarnym (w stosunku do operatora krzyżowania). Wygląda jednak na to, że im większa jest ogólność alfabetu użytego do kodowania osobników, tym bardziej rośnie rola operatora(-ów) mutacji. Tak było w algorytmach, w których osobniki były reprezentowane jako wektory zmiennopozycyjne, całkowitoliczbowe, macierze, automaty skończone itd. Nie będzie więc w tym nic dziwnego, jeżeli zobaczymy nowy trend w programowaniu genetycznym (patrz rozdz. 13), w którym podkreśla się rolę różnych operatorów mutacji. Może to także umożliwić istotne zmniejszenie liczebności populacji w metodach programowania genetycznego, co jeszcze bardziej zwiększy ich wydajność.

12

Uczenie maszynowe

Rzeczywistym problemem nie jest to,
czy myślą maszyny, ale czy myślą ludzie.

B.F. Skinner, *Contingencies of Reinforcement*

Uczenie maszynowe polega przede wszystkim na tworzeniu programów komputerowych potrafiących pozyskiwać, na podstawie wprowadzanej informacji, nową wiedzę lub poprawić wiedzę już posiadaną. W wielu badaniach używa się heurystycznych metod uczenia zamiast algorytmicznych. Najbardziej aktywnym obszarem badań w ostatnich latach [284] było symboliczne nauczanie empiryczne. Ten obszar jest związany z tworzeniem lub modyfikowaniem ogólnych opisów symbolicznych, których struktura nie jest znana *a priori*. Najbardziej popularnym tematem w symbolicznym nauczaniu empirycznym jest opracowywanie opisów pojęć na podstawie przykładów [234], [284]. W szczególności praktyczne znaczenie mają zadania z przestrzenią atrybutów. W wielu takich dziedzinach jest bowiem stosunkowo łatwo podać przykłady zdarzeń, trudno natomiast jest sformułować hipotezy. Celem systemu z tego typu nadzorowanym nauczaniem jest:

> *Utworzyć reguły klasyfikacji dla pojęć obecnych w zbiorze wejściowym, znając początkowy zbiór przykładowych zdarzeń oraz ich przyporządkowanie do pojęć.*

W zależności od języka wyjściowego można podzielić wszystkie podejścia do automatycznego pozyskiwania wiedzy na dwie kategorie: symboliczne i niesymboliczne. W systemach niesymbolicznych wiedza nie jest reprezentowana wprost. Na przykład w modelach statystycznych wiedza jest reprezentowana jako zbiór przykładów łącznie z pewnymi statystykami. W modelu neuronowym wiedza jest rozłożona w połączeniach sieci [355]. W systemach symbolicznych natomiast wiedzę tworzy się i utrzymuje w języku opisowym wysokiego poziomu. Najbardziej znanymi przykładami tego rodzaju systemów są systemy z rodzin AQ i ID [281], [314].

W tym rozdziale opiszemy dwie genetyczne metodyki uczenia maszynowego i omówimy program ewolucyjny GIL (*Genetic Inductive Learning* – gene-

tyczne uczenie indukcyjne) zaproponowany przez Janikowa [200]. Dwa pierwsze podejścia znajdują się gdzieś między systemami symbolicznymi i niesymbolicznymi. Używa się w nich (do pewnego stopnia) języków opisowych wysokiego poziomu. Natomiast stosowane operatory nie są określone w tym języku i operują na poziomie niesymbolicznym. Nawet w pewnych ostatnich reprezentacjach zorientowanych zadaniowo ([149], [363], [340]) operatory nadal operują na zachowawczym, tradycyjnym poziomie podsymbolicznym. Trzeci system – GIL – jest programem ewolucyjnym przystosowanym do „uczenia się na podstawie przykładów". Wiedza specyficzna dla zadania jest w nim zawarta w strukturach danych i operatorach.

Rozważmy przykład, z którego będziemy korzystali w dalszym ciągu rozdziału.

Przykład 12.1
Jest on wzięty ze świata robotów Emeralda (patrz [217] i [407]). Każdy robot jest opisany za pomocą sześciu atrybutów. Atrybuty te oraz ich dziedziny wyglądają następująco:

Atrybut	**Wartości atrybutu**
Kształt_Głowy	Okrągła, Kwadratowa, Wielokątna
Kształt_Ciała	Okrągły, Kwadratowy, Wielokątny
Czy_Uśmiechnięty	Tak, Nie
Trzyma	Miecz, Balon, Flagę
Kolor_Żakietu	Czerwony, Żółty, Zielony, Niebieski
Ma_Krawat	Tak, Nie

Litery wytłuszczone identyfikują atrybuty i ich wartości, na przykład (**Ż** = **Ż**) oznacza „Kolor_Żakietu jest Żółty". Przykładami opisów pojęć (gdzie każde pojęcie C_i jest opisane za pomocą powyższych sześciu atrybutów i ich wartości) są:

C_1 – Głowa jest okrągła i żakiet jest czerwony lub głowa jest kwadratowa i trzyma balon

C_2 – Uśmiecha się i trzyma balon lub głowa jest okrągła

C_3 – Uśmiecha się i nie trzyma miecza

C_4 – Żakiet jest czerwony i nie nosi krawatu lub głowa jest okrągła i uśmiecha się

C_5 – Uśmiecha się i trzyma balon lub miecz

Atrybuty są trzech rodzajów: nominalne (ich dziedziną są zbiory wartości), liniowe (ich dziedziny są liniowo uporządkowane) i strukturalne (ich dziedziny

są częściowo uporządkowane). Zdarzenia są podzielone na klasy decyzyjne: zdarzenia z danej klasy są dla niej przykładami pozytywnymi, a wszystkie inne przykładami negatywnymi. Przykłady do uczenia są podane w formie zdarzeń, a każde zdarzenie jest wektorem wartości atrybutów.

Opisy pojęć są zapisane w VL_1 (uproszczona wersja systemu logiki zmiennowartościowej – *Variable Valued Logic System*) [281] – szeroko zaakceptowanego języka do reprezentacji zdarzeń wejściowych dla jakiegokolwiek programu działającego w przestrzeni atrybutów.

Opis pojęcia C jest dysjunkcją kompleksów

$$c_1 \vee \ldots \vee c_k \Rightarrow C$$

każdy kompleks (c_i) jest wyrażony koniunkcją selektorów, które są trójkami (atrybut – relacja – zbiór_wartości) – na przykład $\langle \dot{Z} = C \rangle$[1] dla „Kolor_Żakietu jest Czerwony".

Pojęcia $C_1 - C_5$ można zapisać następująco:

$$\langle G = O \rangle \wedge \langle \dot{Z} = C \rangle \vee \langle G = K \rangle \wedge \langle T = B \rangle \Rightarrow C_1$$
$$\langle U = T \rangle \wedge \langle T = B \rangle \vee \langle G = O \rangle \Rightarrow C_2$$
$$\langle U = T \rangle \wedge \langle T \neq M \rangle \Rightarrow C_3$$
$$\langle \dot{Z} = C \rangle \wedge \langle K \neq T \rangle \vee \langle G = O \rangle \wedge \langle U = T \rangle \Rightarrow C_4$$
$$\langle U = T \rangle \wedge \langle T = \{B, M\} \rangle \Rightarrow C_5$$

Zauważmy, że selektor $\langle K \neq T \rangle$ można interpretować jako $\langle K = N \rangle$, gdyż atrybut K jest atrybutem logicznym (nominalnym). Selektor $\langle T = \{B, M\} \rangle$ interpretuje się jako „Trzyma Balon lub Miecz" (dysjunkcja wewnętrzna).

Zadanie polega na skonstruowaniu systemu, który uczy się pojęć, to znaczy tworzy reguły decyzyjne, które są spełnione dla wszystkich przykładów pozytywnych i żadnego negatywnego. Można ocenić i porównać systemy na wszystkich opisach robotów, biorąc pod uwagę częstość błędów i złożoność utworzonych reguł. System powinien potrafić sklasyfikować nowe zdarzenia lub zasugerować (być może więcej niż jedną) klasyfikację zdarzeń opisanych w sposób niekompletny.

W ostatnich dwóch dziesięcioleciach narastało zainteresowanie zastosowaniem metod programowania ewolucyjnego w nauczaniu maszynowym (systemy GBML, od „*genetic based machine learning systems*") [88]. Było to spowodowane atrakcyjnością interpretacji chromosomów reprezentujących wiedzę jednocześnie jako danych dla operatorów genetycznych oraz jako programów służących do wykonania pewnych zadań. Jednak wczesne zastosowania, chociaż częściowo zakończone sukcesem (na przykład [323], [340], [341]), natrafiły także na wiele problemów [86]. Ogólnie, w środowisku związanym z algorytmami genetycznymi istnieją dwa rywalizujące ze sobą podejścia do tego zadania. Jak przedstawił De Jong [86]:

---

[1] W przypadku jednego elementu w zbiorze wartości autor opuszcza znaki { } (przyp. tłum.).

Dla każdego, kto czytał Hollanda [191], naturalnym sposobem postępowania jest reprezentowanie całego zbioru reguł jako łańcucha (osobnika), utrzymanie populacji potencjalnych zbiorów reguł i stosowanie selekcji oraz operatorów genetycznych do utworzenia nowych pokoleń zbioru reguł. Historycznie był to sposób przyjęty przez De Jonga i jego studentów w Uniwersytecie w Pittsburghu (na przykład Smith [363], [364]), co dało podstawę nazwie „podejście Pitt".

Jednak w tym samym czasie Holland opracował model rozpoznawania (system klasyfikujący), w którym członkami populacji są indywidualne reguły, a zbiór reguł jest reprezentowany przez całą populację (na przykład Holland i Reitman [189] lub Booker [45]). Podejście to szybko stało się znane jako „podejście Michigan" i zapoczątkowało przyjacielską, ale prowokującą serię dyskusji dotyczących zalet i słabości obu sposobów.

Wydaje się, że trzeci sposób oparty na metodach programowania ewolucyjnego będzie najbardziej owocny. Pomysł uwzględnienia wiedzy specyficznej dla zadania jak zwykle przez: (1) staranne określenie odpowiednich struktur danych, (2) użycie operatorów „genetycznych" specyficznych dla zadania, musi dać wyniki w postaci poprawy dokładności systemu i jego działania. Jednak podejście Pitt jest bliższe naszej idei programowania ewolucyjnego, gdyż utrzymuje populację pełnych rozwiązań (zbioru reguł) zadania, gdy tymczasem w podejściu Michigan (system klasyfikujący) stworzono, przez licytacje, algorytmy brygad porządkujących oraz metody genetyczne w odniesieniu do zmodyfikowanych reguł, nową metodykę, bardzo różną od metod programowania ewolucyjnego. Natomiast programy oparte na podejściu Pitt, nawet jeżeli chromosom jest w nich reprezentowany w języku opisu wysokiego poziomu, nie stosują żadnej metodologii uczenia do modyfikacji swoich operatorów. Jest to podstawowa różnica między podejściem Pitt a podejściem programowania ewolucyjnego.

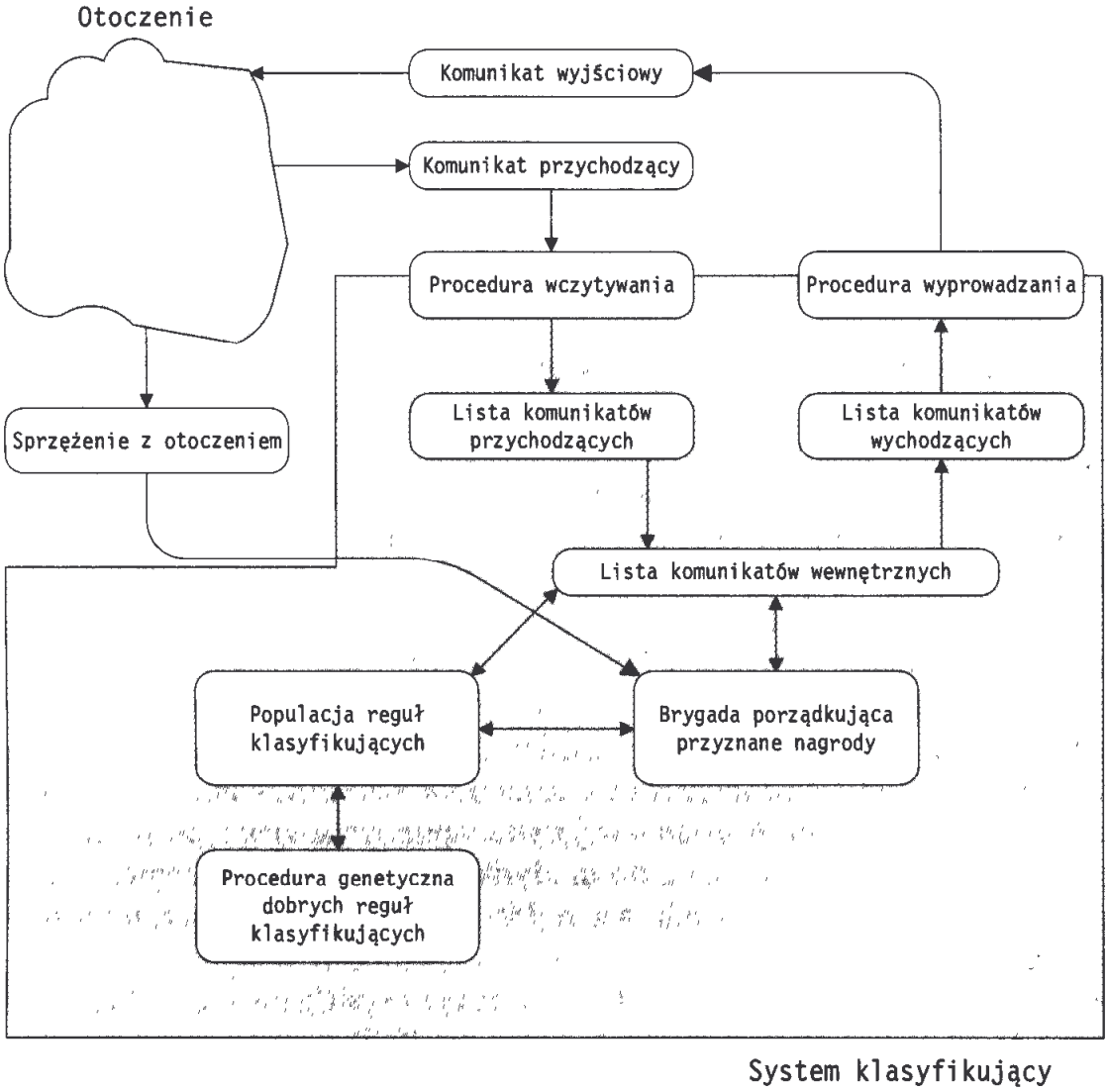
W dalszym ciągu omówimy podstawowe zasady systemów klasyfikujących (podejście Michigan w p. 12.1), podejście Pitt (p. 12.2) i program ewolucyjny Janikowa [200] (p. 12.3) uczenia indukcyjnego reguł decyzyjnych z przykładów opisanych za pomocą atrybutów[1].

12.1. Podejście Michigan

Systemy klasyfikujące są rodzajem systemów regułowych z ogólnym mechanizmem równoległego przetwarzania reguł w celu adaptacyjnego generowania nowych reguł oraz testowania efektywności reguł istniejących. Systemy klasyfi-

[1] Więcej informacji o systemach klasyfikujących można znaleźć w [107], ostatnim specjalnym wydaniu czasopisma *Evolutionary Computation*.

kujące tworzą ramy, w których populacja reguł zakodowanych w postaci łańcuchów binarnych ewoluuje przez sporadycznie podawane pobudzenia i wzmocnienia z otoczenia. System „uczy się", które odpowiedzi są odpowiednie dla zadanego pobudzenia. Reguły systemu klasyfikacyjnego tworzą populację osobników rozwijających się w czasie.



Rys. 12.1. System klasyfikujący i otoczenie

System klasyfikujący (rys. 12.1) składa się z następujących części:

- procedur wczytywania i wyprowadzania,
- systemu komunikatów (listy komunikatów przychodzących, wychodzących i wewnętrznych),
- systemu reguł (populacja reguł klasyfikujących),
- systemu przyznania nagrody (algorytmu brygady porządkującej),
- procedury genetycznej (reprodukcji reguł klasyfikujących).

Otoczenie wysyła komunikat (ruch na tablicy, przykład nowego zdarzenia itp.), który jest akceptowany przez procedurę wczytywania i umieszczany na

liście komunikatów przychodzących. Procedura wczytywania dekoduje komunikat na jeden lub kilka (rozkodowanych) komunikatów i umieszcza je na (wewnętrznej) liście komunikatów. Komunikaty aktywują reguły klasyfikujące. Mocne, uaktywnione reguły klasyfikujące umieszczają komunikaty na liście komunikatów. Nowe komunikaty mogą uaktywnić inne reguły klasyfikujące lub mogą wysyłać pewne komunikaty do listy komunikatów wyjściowych. W tym ostatnim przypadku procedury wyprowadzania kodują te komunikaty na komunikaty wyjściowe (przesunięcie w tablicy, decyzja itp.), które są przesyłane do otoczenia. Otoczenie ocenia akcje systemu (sprzężenie z otoczeniem), a brygada porządkująca uaktualnia moc reguł klasyfikujących.

Omówimy szczegółowo niektóre z powyższych akcji, używając jako przykładu świata robotów Emeralda. Omówimy najpierw podstawowe zagadnienia. Pojedyncza reguła klasyfikująca składa się z dwóch części: (1) warunku, (2) komunikatu. Część warunku jest łańcuchem o skończonej długości zapisanym w pewnym alfabecie, który zawiera symbol „nieistotne" – „*". Część komunikatu jest łańcuchem o skończonej długości zapisanym w tym samym alfabecie, jednak nie zawiera symbolu „nieistotne". Każda reguła klasyfikująca ma swoją moc. Moc jest istotna w procesie licytacji, w którym reguły rywalizują o wysłanie swojego komunikatu. Omówimy to dalej w tym punkcie. Teraz wrócimy do świata robotów Emeralda. Możemy wyrazić regułę decyzyjną przez jedną lub więcej reguł klasyfikujących. Każda reguła klasyfikująca ma następującą postać:

$$(p_1, p_2, p_3, p_4, p_5, p_6) : d$$

gdzie p_i oznacza wartość i-tego atrybutu ($1 \leqslant i \leqslant 6$) dla powyżej zapisanej dziedziny (na przykład $p_4 \in \{M, B, F, *\}$ (Trzyma) i $d \in \{C_1, C_2, C_3, C_4, C_5\}$.

Na przykład reguła klasyfikująca

$$(O\ *\ *\ *\ C\ *) : C_1$$

reprezentuje „jeżeli głowa jest okrągła i żakiet jest czerwony, to pojęcie C_1". Zbiór reguł klasyfikujących dla pięciu pojęć podanych w tym rozdziale to:

$$(O\ *\ *\ *\ C\ *) : C_1$$
$$(K\ *\ *\ B\ *\ *) : C_1$$
$$(*\ *\ T\ B\ *\ *) : C_2$$
$$(O\ *\ *\ *\ *\ *) : C_2$$
$$(*\ *\ T\ B\ *\ *) : C_3$$
$$(*\ *\ T\ F\ *\ *) : C_3$$
$$(*\ *\ *\ *\ C\ N) : C_4$$
$$(O\ *\ T\ *\ *\ *) : C_4$$
$$(*\ *\ T\ B\ *\ *) : C_5$$
$$(*\ *\ T\ M\ *\ *) : C_5$$

Aby uprościć przykład, założymy, że system uczy się jednego pojęcia C_1. Każdy system można łatwo uogólnić, aby obejmował wiele pojęć. W naszym przypadku każda reguła klasyfikująca ma postać

$$(p_1, p_2, p_3, p_4, p_5, p_6) : d$$

gdzie $d = 1$ (przynależność do pojęcia C_1) lub $d = 0$ (w odwrotnym przypadku).

Załóżmy, że na pewnym etapie procesu uczenia jest w systemie mała (losowa) populacja reguł klasyfikujących q (każda reguła jest podana ze swoją mocą):

$$q_1 = (* \; * \; * \; M \; C \; *) : 1, \quad s_1 = 12{,}3$$
$$q_2 = (* \; * \; T \; * \; * \; N) : 0, \quad s_2 = 10{,}1$$
$$q_3 = (K \; O \; * \; * \; * \; *) : 1, \quad s_3 = 8{,}7$$
$$q_4 = (* \; W \; * \; * \; * \; *) : 0, \quad s_4 = 2{,}3$$

Załóżmy dalej, że z otoczenia przychodzi komunikat wejściowy (nowe przykładowe zdarzenie)

$$(OOTMCN)$$

Jest to opis robota z okrągłą (O) głową, okrągłym (O) ciałem, uśmiechniętego (T), trzymającego miecz (M), w czerwonym (C) żakiecie i bez krawata (N). Oczywiście ten robot jest pozytywnym przykładem dla pojęcia C_1, ze wzgledu na okrągłą głowę i czerwony żakiet.

Ten komunikat uaktywnia trzy reguły klasyfikujące: q_1, q_2 i q_4. Te reguły licytują się, przy czym każda odzywka jest proporcjonalna do mocy reguły ($bid_i = bs_i$). Najmocniejsza reguła klasyfikująca q_1 wygrywa i wysyła swój komunikat. Ponieważ ten komunikat jest zdekodowany jako klasyfikacja właściwa, reguła klasyfikująca otrzymuje nagrodę $r > 0$ i jej moc staje się

$$s_1 := s_1 - bid_1 + r$$

(jeżeli komunikat byłby zdekodowany jako odpowiedź zła, to „nagroda" r byłaby ujemna). Dla współczynników $b = 0{,}2$ oraz $r = 4{,}0$ nowa moc reguły klasyfikującej q_1 wynosi $s_1 = 12{,}3 - 2{,}46 + 4{,}0 = 13{,}84$.

Jednym z parametrów systemu klasyfikacyjnego jest okres algorytmu genetycznego, t_{ga}, który podaje liczbę kroków czasowych (liczbę wyżej opisanych cykli) między przywołaniami algorytmu genetycznego. Oczywiście t_{ga} może być stałe, może być generowane losowo (ze średnią równą t_{ga}), a nawet może być w ogóle nie podane i wtedy decyzja o wywołaniu algorytmu genetycznego może być podjęta na podstawie działania systemu. Załóżmy jednak, że przyszedł czas na użycie algorytmu genetycznego do reguł klasyfikujących.

Przyjmiemy moce reguł klasyfikujących za ich dopasowania – proces selekcji można wykonać, używając selekcji ruletkowej (rozdz. 2). Jednak w sys-

temach klasyfikujących nie jesteśmy już zainteresowani najmocniejszymi (najbardziej dopasowanymi) regułami klasyfikacyjnymi, ale raczej całą populacją reguł klasyfikujących, która wykonuje zadanie klasyfikacji. Z tego wynika, że nie powinniśmy generować całej populacji i że musimy być ostrożni w wyborze osobników do zamiany. Zazwyczaj używa się *modelu ze współczynnikami zatłoczenia* (rozdz. 4), gdyż wymienia się w nim podobnych członków populacji.

Stosowanymi operatorami są znowu mutacja i krzyżowanie. Jednak konieczne są w nich pewne modyfikacje. Rozważmy pierwszy atrybut, Kształt_Głowy. Jego dziedziną jest zbiór $\{O, K, W, *\}$. Jeżeli stosujemy mutację, to zmieni ona mutowany znak na jeden z trzech pozostałych znaków (z równym prawdopodobieństwem):

$$O \rightarrow \{K, W, *\}$$
$$K \rightarrow \{O, W, *\}$$
$$W \rightarrow \{O, K, *\}$$
$$* \rightarrow \{O, K, W\}$$

Moc potomka jest na ogół taka sama jak jego rodzica.

Krzyżowanie nie wymaga żadnej modyfikacji. Korzystamy z faktu, że wszystkie reguły klasyfikujące mają tę samą długość. Aby skrzyżować dwoje wybranych rodziców, powiedzmy q_1 i q_2

$$(* * * \mid K\ O\ *) : 1 \qquad \text{oraz} \qquad (* * T \mid * * N) : 0$$

generujemy losowo punkt cięcia (powiedzmy po trzecim znaku, jak zaznaczono) i dostajemy potomków

$$(* * * * * N) : 0 \qquad \text{oraz} \qquad (* * T\ K\ O\ *) : 1$$

Moce potomków są (być może ważoną) średnią mocy rodziców.

System klasyfikujący jest teraz gotowy do kontynuowania procesu uczenia i rozpoczęcia nowego cyklu z t_{ga} krokami, rozpatrzenia dalszych przykładów pozytywnych i negatywnych oraz modyfikacji mocy reguł klasyfikujących. Spodziewamy się, że w rezultacie populacja reguł klasyfikujących zbiegnie się do pewnej liczby mocnych osobników, na przykład

$$(O * * * C *) : 1$$
$$(K * * B * *) : 1$$
$$(W * * * * *) : 0$$
$$(* * * M\ \dot{Z}\ *) : 0$$

Omówiony powyżej przykład był bardzo prosty. Miał on na celu objaśnienie podstawowych składników systemu klasyfikującego na podstawie paradygmatu uczenia dla świata robotów Emeralda. Zauważmy jednak, że zastosowa-

liśmy najprostszy system licytacji (na przykład nie użyliśmy zmiennej *efektywności licytacji* $e_bid = bid + N(0, \sigma_{bid})$, która wprowadza losowe zakłócenie z odchyleniem standardowym σ_{bid} i zerową wartością oczekiwaną), że nie zastosowaliśmy żadnego systemu podatkowego (zazwyczaj każda reguła klasyfikująca jest opodatkowana, aby zapobiec przesunięciu populacji w kierunku reguł produktywnych) i że wybraliśmy tylko jednego zwycięzcę (najmocniejszą regułę klasyfikującą) w każdym kroku. W ogólności zwycięzcą może być więcej niż jedna reguła klasyfikująca i wtedy wszystkie umieszczają swoje komunikaty na liście komunikatów. Ponadto algorytm brygady porządkowej nie był użyty, w tym sensie, że nagrodę przyznawano w każdym kroku (po każdym podanym przykładzie). Nie było żadnego powiązania między przykładami i nie było potrzeby śledzenia łańcucha nagród w celu przyznania nagrody regule, której komunikat uaktywnił obecnie wygrywającą regułę klasyfikującą. Dla pewnych zadań, jak zadanie planowania, długość części komunikatu jest taka sama jak długość części warunku. W takich przypadkach uaktywniona reguła klasyfikująca (poprzedni zwycięzca) umieszcza swój komunikat na (wewnętrznej) liście komunikatów, która z kolei może uaktywnić inne reguły klasyfikujące (nowych zwycięzców). Wtedy moc poprzedniego zwycięzcy jest powiększana o nagrodę – zapłatę od nowych zwycięzców.

Pełne omówienie różnych wersji systemów klasyfikujących i uwagi o historii rozwoju można znaleźć w [154].

12.2. Podejście Pitt

Podejście Michigan można rozumieć jako komputerowy model świadomości. Wiedza świadomej jednostki jest wyrażona przez zestaw reguł, które podlegają modyfikacji w czasie. Możemy oceniać całą świadomą jednostkę przez jej oddziaływanie na otoczenie. Ocena pojedynczej reguły (to znaczy jednego osobnika) jest nieistotna.

Natomiast w podejściu Pitt przyjmuje się, że każdy osobnik w populacji reprezentuje cały zestaw reguł (oddzielną świadomą jednostkę). Te osobniki rywalizują między sobą, słabe osobniki wymierają, mocne przeżywają i reprodukują się. Robi się to na podstawie naturalnej selekcji propocjonalnej do ich dopasowania, operatorów krzyżowania i mutacji. W skrócie, w podejściu Pitt używa się do uczenia algorytmu genetycznego. W ten sposób w podejściu Pitt unika się delikatnego problemu przydzielania nagród, w którym metoda heurystyczna (taka jak algorytm porządkujący) rozdziela (pozytywne lub negatywne) nagrody między regułami, które współpracowały w wypracowaniu (pożądanego lub niepożądanego) zachowania.

Pozostają jednak do rozważenia pewne interesujące sprawy. Pierwsza to reprezentacja. Czy powinniśmy używać wektorów binarnych o skończonej długości (z ustalonymi formatami pól) do reprezentacji reguł? Taka reprezentacja

jest idealna do generowania nowych reguł, gdyż możemy wtedy użyć do tego celu klasycznego krzyżowania i mutacji. Jednak wygląda to na zbyt ograniczone rozwiązanie, odpowiednie tylko dla systemów pracujących na niższym poziomie czułości. Może lepiej by było, aby geny w chromosomie reprezentowały wartości atrybutów (to znaczy liczba genów była równa liczbie atrybutów)? Taka decyzja (nie wsparta specjalizowanymi operatorami) nie wygląda obiecująco. Jeżeli wymiarowości pewnych dziedzin są duże, to prawdopodobieństwo, że system zbiegnie się zbyt szybko jest bardzo duże [86], nawet ze znacznie większym współczynnikiem mutacji niż zazwyczaj. Wygląda na to, że aby wstawić znaczniki między częściami chromosomu, niezbędna jest pewna wewnętrzna struktura reprezentująca oraz operatory, które znają reprezentację chromosomów. Takie podejście omówiono w [364] i [340].

Smith [363] poszedł nawet dalej i eksperymentował z osobnikami o zmiennej długości. Uogólnił on wiele wyników znanych wcześniej tylko dla algorytmów genetycznych ze stałą długością łańcuchów na algorytmy genetyczne ze zmienną długością łańcuchów.

Jednak wygląda na to, że są potrzebne dalsze śmiałe decyzje w zakresie skomplikowanych zagadnień reprezentacji i operatorów. Dopiero ostatnio [24] zauważono potrzebę takich decyzji:

> Rozwiązaniem tego problemu może być wybranie innych operatorów genetycznych, które są bardziej odpowiednie do „reprezentacji naturalnej". Tradycyjne operatory działające na łańcuchach nie są świętością. Analiza matematyczna algorytmów genetycznych wykazuje, że działają one najlepiej, kiedy wewnętrzna reprezentacja wspomaga powstawanie użytecznych bloków budujących, które mogą następnie podlegać kombinacjom z innymi i poprawiać działanie. Reprezentacje w postaci łańcuchów są tylko jednym z wielu sposobów osiągnięcia tego celu.

W następnym punkcie zaprezentujemy program ewolucyjny (z podejściem Pitt), który właśnie to zawiera. Reprezentacja jest bogata i naturalna ze specjalizowanymi operatorami uwzględniającymi reprezentację i przyjętymi bezpośrednio z metodyki uczenia maszynowego.

12.3. Program ewolucyjny: system GIL

Program ewolucyjny GIL [200] przenosi algorytm genetyczny (podejście Pitt) bliżej do poziomu symbolicznego, przede wszystkim przez określenie specjalizowanych operatorów działających na poziomie zadania. Ponownie podstawowymi składnikami GIL, jak każdego innego programu ewolucyjnego, są struktury danych i operatory genetyczne. System zaprojektowano do nauki tylko pojedynczego pojęcia. Jednak można go łatwo rozszerzyć na przypadek uczenia się wielu pojęć, wprowadzając wielokrotne populacje.

12.3.1. Struktury danych

Pojedynczy chromosom reprezentuje opis pojęcia, którego rozpoznawania należy się nauczyć. Zakłada się, że każde zdarzenie niezgodne z takim opisem jest przykładem negatywnym tego pojęcia. Każdy chromosom jest zbiorem (dysjunkcją) pewnej liczby kompleksów. Liczba kompleksów w chromosomie może się zmieniać – założenie, że wszystkie chromosomy mają jednakową długość (tak jak w algorytmach genetycznych utrzymujących populację łańcuchów o stałej długości) byłoby co najmniej sztuczne, a więc sprzeczne z metodą programowania ewolucyjnego. Takie podejście nie jest jednak niczym nowym w algorytmach genetycznych. Jak napisano w poprzednim punkcie, Smith [363] uogólnił wiele wyników formalnych dla algorytmów genetycznych na łańcuchy o zmiennej długości i zaprogramował system, który utrzymuje populację takich łańcuchów (o zmiennej długości).

Każdy kompleks, jak zdefiniowano w VL_1, jest koniunkcją pewnej liczby selektorów odpowiadających różnym atrybutom. Z kolei każdy selektor jest wewnętrzną dysjunkcją wartości należących do dziedziny swoich atrybutów. Na przykład następujący zapis mógłby być chromosomem (czyli opisem pojęcia C_1):

$$(\langle G = O\rangle \wedge \langle \dot{Z} = C\rangle) \vee (\langle G = K\rangle \wedge \langle T = B\rangle)$$

Głównie ze względu na wydajność przyjęto reprezentację binarną dla selektorów w wewnętrznej reprezentacji chromosmów. Binarne 1 na pozycji i oznacza wystąpienie w tym selektorze i-tej wartości z dziedziny. Oznacza to, że długość podłańcucha odpowiadającego temu atrybutowi jest równa liczebności dziedziny atrybutów. Zauważmy, że zestaw samych jedynek w selektorze odpowiada symbolowi *nieistotne* dla tego atrybutu. Tak więc chromosom opisujący pojęcie C_1 w notacji podobnej do tej, której używaliśmy w systemach klasyfikujących, jest reprezentowany przez

$$(O * * * C * \vee K * * B * *)$$

gdy tymczasem w wewnętrznej reprezentacji systemu GIL zapis ten wygląda następująco:

$$(100|111|11|111|1000|11 \vee 010|111|11|010|1111|11)$$

gdzie pionowe kreski rozdzielają selektory. Zauważmy, że taka reprezentacja umożliwia eleganckie rozwiązanie wewnętrznej dysjunkcji, na przykład pojęcie C_5

$$\langle U = T\rangle \wedge \langle T = \{B, M\}\rangle$$

można przedstawić jako

$$(111|111|10|110|1111|11)$$

12.3.2. Operatory genetyczne

Operatory systemu GIL są oparte na metodyce uczenia indukcyjnego wprowadzonej przez Michalskiego [280]. Opisuje ona różne operatory indukcyjne, które składają się na proces indukcyjnego wnioskowania. Obejmuje ona: *usuwanie warunku*, to znaczy usuwanie selektora z opisu pojęcia, *dodawanie nowej reguły i usuwanie reguły*, *rozszerzanie odnośników* – rozszerzanie wewnętrznej dysjunkcji, *domykanie przedziału* – w dziedzinach liniowych dodawanie brakujących wartości między dwiema występującymi wartościami oraz *uogólnienie wstępujące* – w dziedzinach strukturalnych wstępowanie na wyższy stopień drzewa uogólnień.

W systemie GIL określa się oddzielnie operatory indukcyjne na trzech poziomach abstrakcji: poziomie chromosomu, poziomie kompleksu i poziomie selektora. Omówimy po kolei niektóre z operatorów.

Poziom chromosomu: operatory działają na całych chromosomach.

- *RuleExchange*: ten operator jest podobny do krzyżowania w klasycznym algorytmie genetycznym, gdyż wymienia on wybrane kompleksy między dwoma chromosomami rodziców. Na przykład dwoje rodziców

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11) oraz

  (111|001|01|111|1111|01 ∨ 110|100|10|111|0010|01)

  może utworzyć następujących potomków:

  (100|111|11|111|1000|11 ∨ 111|001|01|111|1111|01) oraz

  (010|111|11|010|1111|11 ∨ 110|100|10|111|0010|01)

- *RuleCopy*: ten operator jest podobny do *RuleExchange*, z tym, że kopiuje on losowo kompleksy jednego z rodziców do drugiego. Na przykład dwoje rodziców

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11) oraz

  (111|001|01|111|1111|01 ∨ 110|100|10|111|0010|01)

  może utworzyć następujących potomków:

  (100|111|11|111|1000|11 ∨ 111|001|01|111|1111|01 ∨

  110|100|10|111|0010|01) oraz

  (010|111|11|010|1111|11)

- *NewPEvent*: ten operator jednoargumentowy dołącza opis pozytywnego przykładu do wybranego chromosomu. Na przykład z rodzica

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11)

oraz nie uwzględnionego zdarzenia

(100|010|10|010|0010|01)

tworzy się następującego potomka:

(100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨
100|010|10|010|0010|01)

- *RuleGeneralization*: ten operator jednoargumentowy uogólnia losowy podzbiór kompleksów. Na przykład z rodzica

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨
  100|010|10|010|0010|01)

  oraz wybranych do sumowania drugiego i trzeciego kompleksu tworzy się następującego potomka:

  (100|111|11|111|1000|11 ∨ 110|111|11|010|1111|11)

- *RuleDrop*: ten operator jednoargumentowy usuwa losowy podzbiór kompleksów. Na przykład z rodzica

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|11 ∨
  100|010|10|010|0010|01)

  może być utworzony następujący potomek:

  (100|111|11|111|1000|11)

- *RuleSpecialization*: ten operator jednoargumentowy specjalizuje losowy podzbiór kompleksów. Na przykład z rodzica

  (100|111|11|111|1000|11 ∨ 010|111|11|010|1111|10 ∨
  111|010|10|010|1111|11)

  oraz drugiego i trzeciego kompleksu wybranego do specjalizacji tworzy się następującego potomka:

  (100|111|11|111|1000|11 ∨ 010|010|10|010|1111|10)

Poziom kompleksu: operatory działają na kompleksach chromosomu.

- *RuleSplit*: ten operator działa na pojedynczym kompleksie, dzieląc go na kilka kompleksów. Na przykład z rodzica

  (100|111|11|111|1000|11)

  może być utworzony następujący potomek (operator dzieli drugi selektor):

  (100|011|11|111|1000|11 ∨ 100|100|11|111|1000|11)

- *SelectorDrop*: ten operator działa na pojedynczym kompleksie i „usuwa" jeden selektor, to znaczy wszystkie wartości wybranego selektora są zamieniane na łańcuch „11..1". Na przykład z rodzica

  (100|010|11|111|1000|11)

  może być utworzony następujący potomek (operator działa na piąty selektor):

  (100|010|11|111|1111|11)

- *IntroSelektor*: ten operator działa przez „dodanie" selektora, to znaczy modyfikuje on selektor o postaci „11...1". Na przykład z rodzica

  (100|010|11|111|1111|11)

  może być utworzony następujący potomek (operator działa na piąty selektor):

  (100|010|11|111|0001|11)

- *NewNEvent*: ten operator jednoargumentowy dołącza opis negatywnego zdarzenia do wybranego chromosomu. Na przykład z rodzica

  (110|010|11|111|1111|11)

  oraz zgodnego z nim zdarzenia negatywnego

  (100|010|10|010|0100|10)

  może być utworzony następujący potomek:

  (010|010|11|111|1111|11 ∨ 110|010|01|111|1111|11 ∨
  110|010|11|101|1111|11 ∨ 110|010|11|111|1011|11 ∨ 110|010|11|111|1111|01)

Poziom selektora: operatory działają na selektory.

- *ReferenceChange*: operator dodaje lub usuwa pojedynczą wartość (0 lub 1) w chromosomie, to znaczy w dziedzinie jednego z selektorów. Na przykład z rodzica

  (100|010|11|111|0001|11)

  może być utworzony następujący potomek (zauważ różnicę w czwartym selektorze):

  (100|010|11|110|0001|11)

- *ReferenceExtension*: operator rozszerza dziedzinę selektora, zezwalając na kilka dodatkowych wartości. Dla różnych typów atrybutów (nominalnych, liniowych, strukturalnych) stosuje on różne prawdopodobieństwa wyboru wartości. Na przykład z rodzica

  (100|010|11|111|1010|11)

może powstać następujący potomek (operator „domyka" dziedzinę piątego selektora):

(100|010|11|111|1110|11)

- *ReferenceRestriction*: ten operator usuwa pewne wartości z dziedziny selektora. Na przykład z rodzica

  (100|010|11|111|1011|11)

  może powstać następujący potomek:

  (100|010|11|111|1000|11)

System GIL jest dosyć skomplikowany i zawiera sporo parametrów (na przykład prawdopodobieństw użycia operatorów). Omówienie tych i innych kwestii związanych z zaprogramowaniem można znaleźć w [200]. Warto jednak zauważyć, że operatorom przydziela się na początku prawdopodobieństwa, ale faktyczne prawdopodobieństwa oblicza się jako funkcję tych prawdopodobieństw i dwóch innych parametrów: (zadanego) pożądanego rozdziału między specjalizacją a uogólnieniem oraz (dynamicznej) miary aktualnej poprawności opisu pozytywnych i negatywnych przykładów. Ten pomysł jest podobny do omawianego wcześniej (w ostatnim punkcie rozdz. 8).

W każdej iteracji wszystkie chromosomy są oceniane ze względu na ich pełność i zgodność (i ewentualnie koszt, jeżeli się tego żąda) i nową populację tworzy się tak, że wystąpienie w niej lepszych chromosomów jest bardziej prawdopodobne. Następnie operatory genetyczne są stosowane do nowej populacji i cykl się powtarza.

12.4. Porównanie

W ostatniej publikacji [407] przedstawiono ocenę kilku strategii uczenia, łącznie z systemem klasyfikującym (CFS), siecią neuronową (BpNet), programem uczenia za pomocą drzewa decyzyjnego (C4.5) i programu uczenia regułowego (AQ15). W systemach użyto przykładów ze świata robotów Emeralda. Systemy miały nauczyć się pięciu pojęć ($C_1 - C_5$) podanych na początku tego rozdziału, na podstawie różnej liczby pozytywnych i negatywnych przykładów (ogółem były 432 różne roboty, to znaczy użyto wszystkich możliwych kombinacji wartości atrybutów). Systemy porównywano na podstawie średniego błędu rozpoznawania wszystkich 432 (widzianych wcześniej lub nie) robotów. Wyniki (z [407]) podano w tabl. 12.1.

W tablicy 12.2 podano częstość rozpoznawania przez system GIL pojedynczych pojęć (z [200]). Jak można się było spodziewać, program ewolucyjny GIL działa znacznie lepiej niż system CFS z systemem klasyfikującym. Dosyć nie-

oczekiwanie GIL wypadł także lepiej niż inne systemy uczenia. Jego wyższość jest najbardziej widoczna w przypadku małej części widzianych i nie widzianych przykładów.

Tablica 12.1. Częstości błędów dla różnych systemów

System	Scenariusz uczenia (Pozytywne %/Negatywne %)				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	22,8%	5,0%	4,8%	1,2%	0,0%
BpNet	9,7%	6,3%	4,7%	7,8%	4,8%
C4.5	9,7%	8,3%	11,3%	2,5%	1,6%
CFS	21,3%	20,3%	21,5%	19,7%	23,0%

Tablica 12.2. Częstości błędów dla systemu GIL

Pojęcie	Scenariusz uczenia (Pozytywne %/Negatywne %)				
	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
C_1	11,1%	5,3%	0,0%	0,0%	0,0%
C_2	0,0%	0,0%	0,0%	0,0%	0,0%
C_3	0,0%	0,0%	0,0%	0,0%	0,0%
C_4	10,4%	0,0%	0,0%	0,0%	0,0%
C_5	0,0%	0,0%	0,0%	0,0%	0,0%

Dokładniejsze omówienie porównania tych systemów (na przykład złożoności generowanych reguł), zagadnienia związane z zaprogramowaniem systemu GIL oraz wyniki innych obliczeń (dla multiplekserów, raka piersi itp.) można znaleźć w [200].

12.5. REGAL

Ciekawe podejście do tworzenia opisu pojęć na podstawie przykładów podali ostatnio Giordana i Saitta [139]. Opracowany system REGAL uczy się zapisywania pojęć w dysjunktywnej postaci normalnej

$$c_1 \vee \ldots \vee c_k \Rightarrow C$$

Każdy kompleks (c_i) jest w nim wyrażony jako koniunkcja selektorów, które mogą zawierać wewnętrzne dysjunkcje, na przykład

$$\langle G = O \vee W \rangle \wedge \langle \dot{Z} = C \vee \dot{Z} \rangle \vee \langle G = K \rangle \wedge \langle T = B \vee F \rangle \Rightarrow C$$

Podstawową sprawą jest znowu reprezentacja. System REGAL pracuje z łańcuchami binarnymi o stałej długości, co wymaga odwzorowywania dys-

junktywnej postaci normalnej na takie łańcuchy. Uzyskuje się to, nakładając ograniczenie na złożoność formuły, co określa się za pomocą wzorcowego wyrażenia w języku Λ – reprezentuje ono najbardziej skomplikowaną formułę. Inne formuły można uzyskać, opuszczając pewne literały z Λ. W ten sposób można ustalić odpowiedniość między literałami w Λ i bitami łańcucha. System REGAL używa krzyżowania dwupunktowego i jednorodnego oraz krzyżowań uogólniających i specjalizujących określanych dla danego zadania.

System był przetestowany w trybach „uczenia jednego kompleksu na raz" oraz „uczenia wielu dysjunkcji na raz", gdzie stosowano funkcje rozdzielające. Bardziej szczegółowy opis systemu oraz wyniki obliczeń można znaleźć w [139], dalsze wyniki obliczeń opisano w [140].

13

Programowanie ewolucyjne a programowanie genetyczne

Przeszłość jest teraźniejszością, czyż nie tak?
Jest także przyszłością.

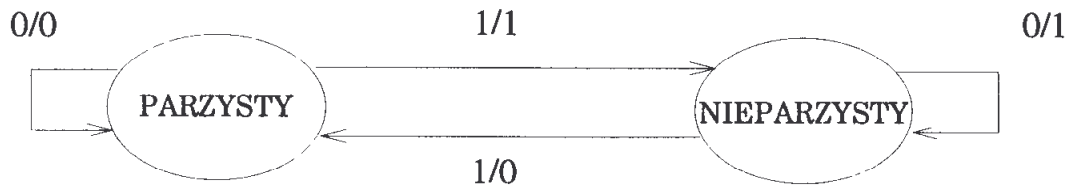
Eugene O'Neill, *Long Day's Journey Into Night*

W tym rozdziale dokonamy krótkiego przeglądu dwóch mocnych metod ewolucyjnych: programowania ewolucyjnego (p. 13.1) oraz programowania genetycznego (p. 13.2). Są to dwie metody, których czas pierwszego opracowania różni się od siebie o ćwierć wieku. Były one przeznaczone do rozwiązywania innych zadań, mają inne rozwiązania chromosomów dla osobników w populacji i kładą nacisk na inne operatory. Jednak są one bardzo podobne do siebie z naszej perspektywy „programów ewolucyjnych". Do wykonania zadań używają one specjalizowanych struktur danych (automatów ze skończoną liczbą stanów i programów komputerowych o strukturze drzewa) oraz specjalizowanych operatorów „genetycznych". Obie metody kontrolują także skomplikowanie swoich struktur (pewne miary skomplikowania automatów ze skończoną liczbą stanów lub drzewa mogą być dołączone do funkcji oceny). Omówimy je po kolei.

13.1. Programowanie ewolucyjne

Pierwsze metody programowania ewolucyjnego opracował Lawrence Fogel [126]. Miały one na celu rozwój sztucznej inteligencji, w sensie rozwoju umiejętności przewidywania zmian w otoczeniu. Otoczenie opisywano przez ciąg symboli (ze skończonego alfabetu), a algorytm ewolucyjny miał tworzyć, jako wyjście, nowy symbol. Symbol wyjściowy powinien maksymalizować funkcję wypłat, która była miarą dokładności przewidywania.

Możemy na przykład rozważyć ciąg zdarzeń oznaczonych symbolami a_1, a_2, ... Algorytm powinien przewidzieć nowy (nieznany) symbol, powiedzmy a_{n+1} na podstawie poprzednich (znanych) symboli a_1, a_2,..., a_n. Idea programowania ewolucyjnego polegała na opracowaniu takiego algorytmu.



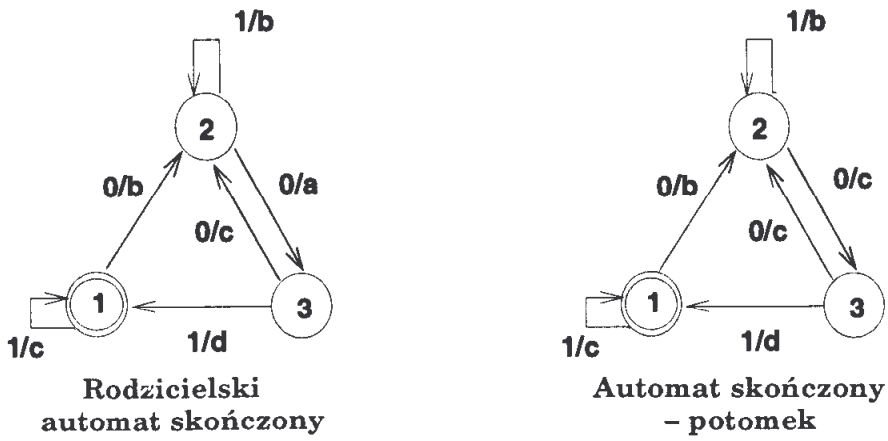
Rys. 13.1. Automat skończony do sprawdzania parzystości

Do reprezentacji chromosomów dla osobników wybrano automaty ze skończoną liczbą stanów (automaty skończone). Zresztą automaty skończone dają sensowną reprezentację zachowania przy interpretacji symboli. Na rysunku 13.1 przedstawiono schemat przepływowy prostego automatu skończonego do sprawdzania parzystości. Takie schematy przepływowe są grafami skierowanymi z wierzchołkiem w każdym stanie, krawędziami, które wskazują na możliwość przejść z jednego stanu do innego, oraz wartościami wejściowymi i wyjściowymi (zapis a/b przy krawędzi prowadzącej od stanu S_1 do stanu S_2 wskazuje, że wartość wejściowa a, gdy automat jest w stanie S_1, daje na wyjściu b oraz powoduje przejście do stanu S_2.

Są dwa stany „PARZYSTY" i „NIEPARZYSTY" (automat zaczyna od stanu „PARZYSTY"). Automat rozpoznaje parzystość łańcucha binarnego.

A więc w metodzie programowania ewolucyjnego utrzymuje się populację automatów skończonych. Każdy taki osobnik reprezentuje potencjalne rozwiązanie zadania (to znaczy reprezentuje szczególne zachowanie). Jak już wspomniano, każdy automat skończony jest oceniany, aby określić pewną miarę jego „dopasowania". Robi się to w następujący sposób. Każdy automat jest poddany działaniu otoczenia w tym sensie, że bada poprzednio podane symbole. Dla każdego ciągu, powiedzmy $a_1, a_2, \ldots, a_i$ podaje on na wyjściu a'_{i+1}, które jest porównywane z następnym przychodzącym symbolem a_{i+1}. Na przykład, jeżeli przyszło dotąd n symboli, to automat skończony robi n przewidywań (jedno dla każdego podłańcucha a_1, a_2, a_3 i tak dalej, aż do a_n). W funkcji oceny bierze się pod uwagę całościowy wynik (na przykład pewną ważoną średnią dokładności w n przewidywaniach).

Podobnie jak w strategiach ewolucyjnych (p. 8.1), w metodzie programowania ewolucyjnego najpierw tworzy się potomka, a następnie wybiera osobnika do następnego pokolenia. Każdy rodzic tworzy jednego potomka. Wobec tego rozmiar pośredniej populacji podwaja się (jak w (*pop_size*, *pop_size*)-SE). Potomków (nowe automaty skończone) tworzy się przez losową mutację populacji rodziców (patrz rys. 13.2). Jest pięć możliwych operatorów mutacji: zmiana symbolu wyjściowego, zmiana przejść między stanami, dodanie stanu, usunięcie stanu i zmiana stanu początkowego (są dodatkowe ograniczenia na minimalną i maksymalną liczbę stanów). Te mutacje wybiera się zgodnie z pewnymi rozkładami prawdopodobieństw (które mogą się zmieniać w procesie ewolucyjnym). Można także stosować więcej niż jedną mutację w tym samym rodzicu (decyzja o liczbie mutacji u poszczególnego osobnika odbywa się na podstawie innego rozkładu prawdopodobieństwa).



Rys. 13.2. Automat skończony i jego potomek. Automat rozpoczyna pracę w stanie 1

Najlepszych *pop_size* osobników zatrzymuje się do następnego pokolenia, to znaczy, aby się zakwalifikować do następnego pokolenia osobnik powinien mieścić się w górnych 50% pośredniej populacji. W oryginalnej wersji [126] ten proces iterowano kilka razy przed podaniem następnego symbolu wyjściowego. Kiedy poda się symbol wyjściowy, dodaje się go do listy znanych symboli i cały proces się powtarza.

Oczywiście powyższa procedura może być rozszerzana na wiele sposobów. Jak podano w [121]:

> Funkcja wypłat może być dowolnie złożona i może mieć chwilowe składowe. Nie ma żadnego wymogu, aby używać klasycznego wskaźnika z kwadratem błędu lub z inną gładką funkcją. Ponadto nie wymaga się, aby przewidywania wykonywać z wyprzedzeniem jednokrotnym. Przewidywanie może być wykonywane z dowolnym wyprzedzeniem wielokrotnym. Można przyjmować wielowymiarowe otoczenia, a procesy zachodzące w otoczeniu nie muszą być stacjonarne, ponieważ symulowana ewolucja zaadaptuje się do zmian w statystyce przejść.

Jak wspomniano w p. 8.2, metody programowania ewolucyjnego uogólniono, aby mogły rozwiązywać zadanie optymalizacji numerycznej. Szczegóły można znaleźć w [117] i [121]. Inne przykłady metod programowania ewolucyjnego znajdują sie także w [126] (podział ciągu liczb całkowitych na liczby pierwsze i złożone), [120] (zastosowanie metod programowania ewolucyjnego w iterowanym dylemacie więźnia), jak również w [123], [124], [378], [254] z wieloma innymi zastosowaniami.

13.2. Programowanie genetyczne

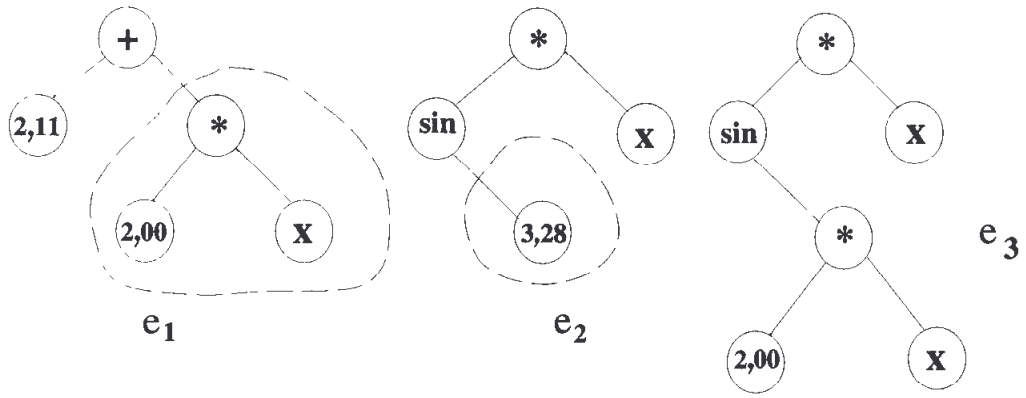
Inne interesujące podejście opracował stosunkowo niedawno Koza [228], [231]. Koza zasugerował, że pożądany program powinien sam rozwijać się w trakcie procesu ewolucyjnego. Inaczej mówiąc, zamiast rozwiązywać zadanie i zamiast

budować program ewolucyjny do rozwiązywania zadania, powinniśmy przeszukiwać przestrzeń możliwych programów komputerowych w celu znalezienia najlepszego (najbardziej dopasowanego). Koza opracował nową metodykę nazwaną *programowaniem genetycznym*, która podaje sposób wykonywania takiego przeszukiwania. Tworzy się w niej populację wykonywalnych programów komputerowych, w której pojedyncze programy rywalizują ze sobą, słabe programy wymierają, a mocne reprodukują się (za pomocą krzyżowania, mutacji)....

Zastosowanie programowania genetycznego do szczególnego zadania składa się z pięciu kroków. Są to:

- wybór końcówek,
- wybór funkcji,
- określenie funkcji oceny,
- wybór parametrów systemu,
- wybór warunku zakończenia.

Warto zauważyć, że struktura podlegająca ewolucji jest programem komputerowym o strukturze hierarchicznej[1]. Przestrzeń rozwiązań jest hiperprzestrzenią programów, którą można przedstawić jako przestrzeń drzew z korzeniami. Każde drzewo składa się z funkcji i końcówek odpowiednich dla każdej dziedziny zadań. Zbiór wszystkich funkcji i końcówek jest wybrany *a priori* w taki sposób, że niektóre ze złożonych drzew dają rozwiązanie.



Rys. 13.3. Wyrażenie e_3 jako potomek e_1 i e_2. Linią przerywaną oznaczono pola wymieniane w operacji krzyżowania

Na przykład dwie struktury e_1 i e_2 (rys. 13.3) reprezentują odpowiednio wyrażenia $2x + 2{,}11$ oraz $x \sin(3{,}28)$. Możliwy potomek e_3 (po krzyżowaniu e_1 i e_2) reprezentuje $x \sin(2x)$.

Populacja początkowa składa się z takich drzew. Konstrukcja (losowego) drzewa jest prosta. Funkcja oceny przyporządkowuje wartość dopasowania,

[1] W rzeczywistości Koza przyjął wyrażenie S w LISP-ie we wszystkich eksperymentach. Obecnie jednak są także programy programowania genetycznego w języku C oraz innych językach programowania.

która ocenia działanie drzewa (programu). Ocena odbywa się na podstawie wcześniej ustalonego zbioru przypadków testujących. W ogólności funkcja oceny podaje sumę odległości między prawidłowymi a uzyskanymi wynikami we wszystkich przypadkach testowych. Selekcja jest proporcjonalna. Prawdopodobieństwo wybrania drzewa do następnego pokolenia jest proporcjonalne do jego dopasowania. Podstawowym operatorem jest krzyżowanie, które prowadzi do dwóch potomków z dwóch wybranych rodziców. W krzyżowaniu tworzy się potomka przez wymianę poddrzew między dwoma rodzicami. Są także inne operatory: mutacja, permutacja, edycja oraz operacja definiowania bloku budującego [228]. Na przykład w typowej mutacji wybiera się wierzchołek drzewa i generuje nowe (losowe) poddrzewo, które zaczyna się w wybranym wierzchołku.

Na dodatek do pięciu głównych kroków tworzenia programu genetycznego dla szczególnego zadania Koza [232] ostatnio rozpatrzył zalety uwzględnienia jeszcze jednej cechy: zbioru procedur. Są one zwane *funkcjami definiowanymi automatycznie*. Wygląda na to, że jest to wyjątkowo użyteczne pojęcie w metodach programowania genetycznego, w szczególności w zakresie wielokrotnego wykorzystywania oprogramowania. Funkcje definiowane automatycznie wykrywają i wykorzystują regularności, symetrie, podobieństwa, wzorce i modułowości rozpatrywanego zadania. Końcowy program genetyczny może wywoływać te procedury w różnych etapach swojej pracy.

Prawdopodobnie byłoby błędem zaklasyfikowanie programowania genetycznego jako jeszcze jednej wersji programów ewolucyjnych ze specjalną reprezentacją osobników. Fakt, że programowanie genetyczne działa na programach komputerowych ma kilka interesujących aspektów. Na przykład operatory można rozumieć jako programy, które podlegają odrębnej ewolucji podczas działania systemu. Dodatkowo zbiór funkcji może składać się z kilku programów, które wykonują skomplikowane zadania. Takie funkcje mogą ewoluować dalej podczas obliczenia ewolucyjnego (na przykład funkcje definiowane automatycznie). Oczywiście jest to jedna z najbardziej ekscytujących dziedzin w obecnym rozwoju obliczeń ewolucyjnych z tylko niewielką liczbą wyników eksperymentalnych (patrz na przykład, oprócz [231] i [232], także [225] oraz [8]).

14

Hierarchia programów ewolucyjnych

Mądrzy powinni zachować się na jutro,
a nie stawiać wszystkiego na jedną kartę.

Cervantes, *Przemyślny szlachcic Don Kichote z Manczy*[1)]

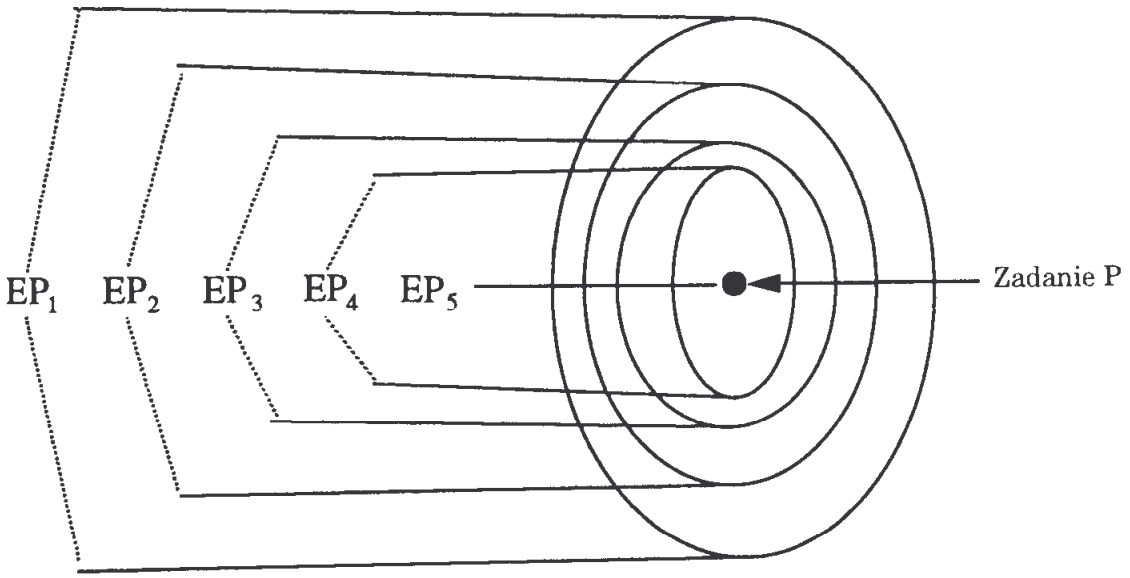
W tej książce omawialiśmy różne strategie zwane programami ewolucyjnymi, które można stosować w trudnych zadaniach optymalizacji i które są oparte na zasadzie ewolucji. Programy ewolucyjne zapożyczyły dużo z algorytmów genetycznych. Jednak zawierają one wiedzę specyficzną dla zadania w „naturalnych" strukturach danych i operatorach „genetycznych" wrażliwych na zadanie. Podstawowa różnica między algorytmami genetycznymi a programami ewolucyjnymi polega na tym, że te pierwsze uważa się za słabe metody niezależne od zadania, co nie zachodzi w wypadku tych drugich.

Granica między słabymi a mocnymi metodami nie jest dobrze zdefiniowana. Można zbudować różne programy ewolucyjne, które wykazują zmienny stopień uzależnienia od zadania. W ogólności dla konkretnego zadania P można skonstruować całą rodzinę programów ewolucyjnych EP_i, w której każdy program „rozwiąże" zadanie (rys. 14.1). Określenie „rozwiąże" oznacza „da rozsądne rozwiązanie", to jest rozwiązanie, które nie musi być oczywiście optymalne, ale jest dopuszczalne (spełnia ograniczenia zadania).

Program ewolucyjny EP_5 (rys. 14.1) jest najbardziej przystosowany do zadania i rozwiązuje tylko zadanie P. System EP_5 nie będzie działał dla żadnej zmodyfikowanej wersji zadania (na przykład po dodaniu nowego ograniczenia lub zmianie wymiaru zadania). Następny program ewolucyjny EP_4 można stosować w pewnej (stosunkowo małej) klasie zadań zawierających zadanie P. Dalsze programy ewolucyjne EP_3 i EP_2 działają w większych dziedzinach, podczas gdy EP_1 nie zależy od dziedziny i może być zastosowany do dowolnego zadania optymalizacji.

Widzieliśmy już częściowo taką hierarchię w różnych miejscach tej książki. Rozważmy w szczególności nieliniowe zadanie transportowe P o wymiarach 20×20.

[1)] Przekład Anny Ludwiki Czerny i Zygmunta Czerny, PIW, 1986 (przyp. tłum.).



Rys. 14.1. Hierarchia programów ewolucyjnych

Zawiera ono 400 zmiennych oraz $20 + 20 = 40$ równań (z których 39 jest niezależnych). Dodatkowe ograniczenia dotyczą nieujemności zmiennych. W zasadzie można skonstruować program ewolucyjny, powiedzmy EP_5, który rozwiąże to szczególne zadanie. Może to być algorytm genetyczny z 39 funkcjami kary dostrojonymi bardzo starannie do tych ograniczeń lub z dekoderami albo algorytmami naprawy. Jakakolwiek zmiana wymiaru zadania (przesunięcie z 20×20 na 20×21) lub zmiana kosztu transportu z jednego miejsca nadania do jednego miejsca odbioru spowoduje błędne działanie systemu EP_5.

Jest także program ewolucyjny GENETIC-2 (rozdz. 9), którego można użyć dla dowolnego zadania transportowego. Nazwijmy go systemem EP_4. Należy on nadal do klasy mocnych metod, gdyż można go stosować tylko w klasie zadań transportowych. Jednak jest on znacznie słabszy niż EP_5, gdyż może rozwiązywać wszystkie zadania transportowe.

Innym programem ewolucyjnym, powiedzmy EP_3, który można zastosować w zadaniu P, jest GENOCOP (rozdz. 7). Optymalizuje on dowolną funkcję z liniowymi ograniczeniami, które właśnie występują w zadaniu transportowym P. Oczywiście EP_3 jest słabszą metodą niż EP_4. Jednak nadal można ją uważać za stosunkowo mocną metodę, gdyż można ją stosować tylko do optymalizacji numerycznej z liniowymi ograniczeniami.

Jeszcze innym programem ewolucyjnym (nazwijmy go EP_2), który można zastosować do naszego zadania transportowego P o wymiarze 20×20, jest strategia ewolucyjna (rozdz. 8). Strategie ewolucyjne można stosować dla dowolnego zadania optymalizacji numerycznej z (niekoniecznie liniowymi) ograniczeniami nierównościowymi. Oczywiście zadanie P należy do dziedziny EP_2. EP_2 jest słabszą metodą niż EP_3, gdyż działa dla ograniczeń nierównościowych (w zadaniu P równości można łatwo zamienić na nierówności, używając metody opisanej w rozdz. 7).

Możemy także skonstruować program ewolucyjny ogólnego zastosowania EP_1, który może być po prostu klasycznym algorytmem genetycznym ze standardowym zbiorem funkcji kar. Każda funkcja kary odpowiada jednemu z ograniczeń zadania. System EP_1 nie zależy od dziedziny. Może on rozwiązywać dowolne zadanie optymalizacji z dowolnym zbiorem ograniczeń. Przy optymalizacji numerycznej mogą to być też równości nieliniowe, co powoduje, że jest on słabszy niż EP_2, który działa tylko dla nierówności. Ponadto EP_1 można stosować także do innych (nienumerycznych) zadań. Zakładamy tutaj, że program EP_1 zawsze daje rozwiązanie dopuszczalne. Możemy go łatwo wzmocnić, przyjmując populację początkową składającą się z rozwiązań dopuszczalnych oraz funkcje kary, dekodery lub algorytmy naprawy utrzymujące osobniki wewnątrz tej przestrzeni.

Oznaczmy przez $dom(EP_i)$ zbiór wszystkich zadań, jakie można rozwiązywać za pomocą programu ewolucyjnego EP_i, to znaczy dla których program daje rozwiązania dopuszczalne. Oczywiście

$$dom(EP_5) \subseteq dom(EP_4) \subseteq dom(EP_3) \subseteq dom(EP_2) \subseteq dom(EP_1)$$

Rzecz jasna, powyższy przykład nie jest w żaden sposób zamknięty. Można utworzyć inne programy ewolucyjne, które będą się znajdowały między EP_i a EP_{i+1} dla pewnego $1 \leqslant i \leqslant 4$. Oczywiście mogą być także programy ewolucyjne, które nakładają się na inne w powyższej hierarchii. Na przykład możemy zbudować system do optymalizacji zadań transportowych z kosztami wyrażonymi funkcjami wielomianowymi lub do optymalizacji zadań określonych na wypukłych przestrzeniach przeszukiwań, lub też do zadań z ograniczeniami w postaci równań nieliniowych. Inaczej mówiąc, zbiór programów ewolucyjnych jest częściowo uporządkowany. Oznaczymy relację częściowego uporządkowania przez $\prec$. Ma ona następujące znaczenie: jeżeli $EP_p \prec EP_q$, to program ewolucyjny EP_p jest metodą słabszą niż EP_q, to znaczy $dom(EP_q) \subseteq dom(EP_p)$. Wracając do naszego przykładu zadania transportowego P oraz hierarchii programów ewolucyjnych EP_i

$$EP_1 \prec EP_2 \prec EP_3 \prec EP_4 \prec EP_5$$

Istnieje hipoteza, że jeżeli $EP_p \prec EP_q$, to mocniejsza metoda EP_q powinna w ogólności działać lepiej niż słabsza EP_p. Nie mamy oczywiście dowodu tej hipotezy, gdyż jest ona oparta tylko na pewnej liczbie wyników obliczeń oraz prostej intuicji, że wiedza zorientowana na zadanie poprawia algorytm w zakresie jego działania (czasu i dokładności), a jednocześnie zawęża możliwość jego zastosowania. Widzieliśmy już, że GENETIC-2 działał lepiej niż GENOCOP i zastanawialiśmy się, dlaczego GENOCOP jest lepszy od klasycznego algorytmu genetycznego w szczególnej klasie zadań. Jeżeli ta hipoteza jest prawdziwa, to GENOCOP powinien dawać lepsze wyniki niż program ewolucyjny oparty na strategii ewolucyjnej dla zadań z liniowymi ograniczeniami,

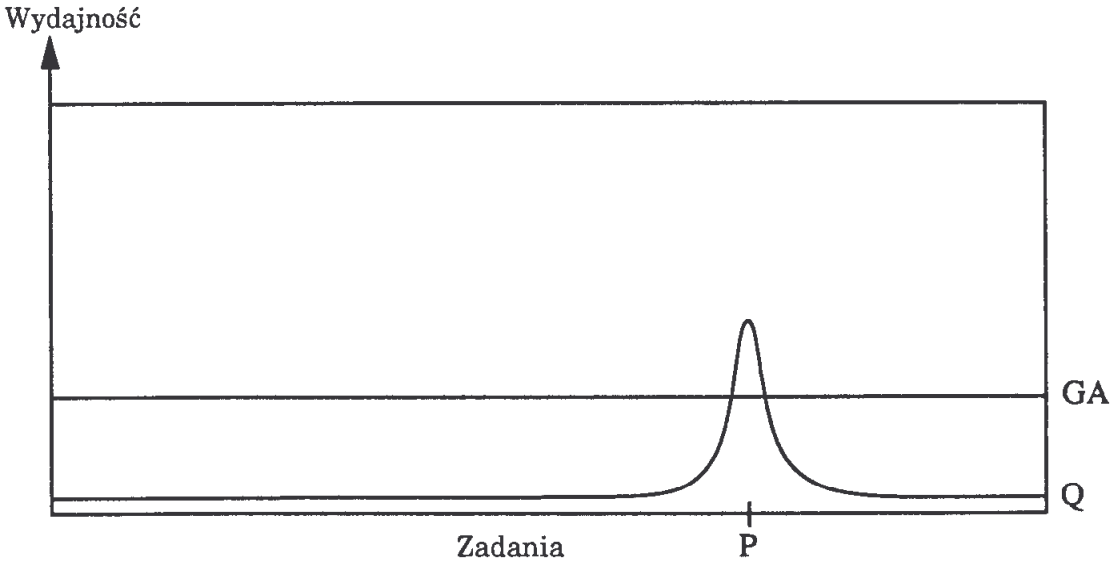
gdyż strategia ewolucyjna jest słabszą metodą niż GENOCOP. Także niektórzy inni badacze popierają tę hipotezę. Zacytujmy Davisa [77]:

> Jest truizmem w systemach ekspertowych, że uwzględnienie wiedzy z dziedziny prowadzi do lepszego działania przy optymalizacji. Ten truizm z pewnością narodził się z mojego doświadczenia z zastosowaniem algorytmów genetycznych w zagadnieniach przemysłowych. Krzyżowanie binarne i mutacja binarna są operatorami nie zawierającymi wiedzy. Wobec tego, jeżeli wzbraniamy się przed dodaniem wiedzy do naszych algorytmów genetycznych, to jest bardzo prawdopodobne, że będą one gorsze od każdego rozsądnego algorytmu optymalizacji, w którym bierze się pod uwagę taką wiedzę z dziedziny.

Goldberg [156], [154] przedstawia jeszcze inne spojrzenie. Zacytujmy z [156]:

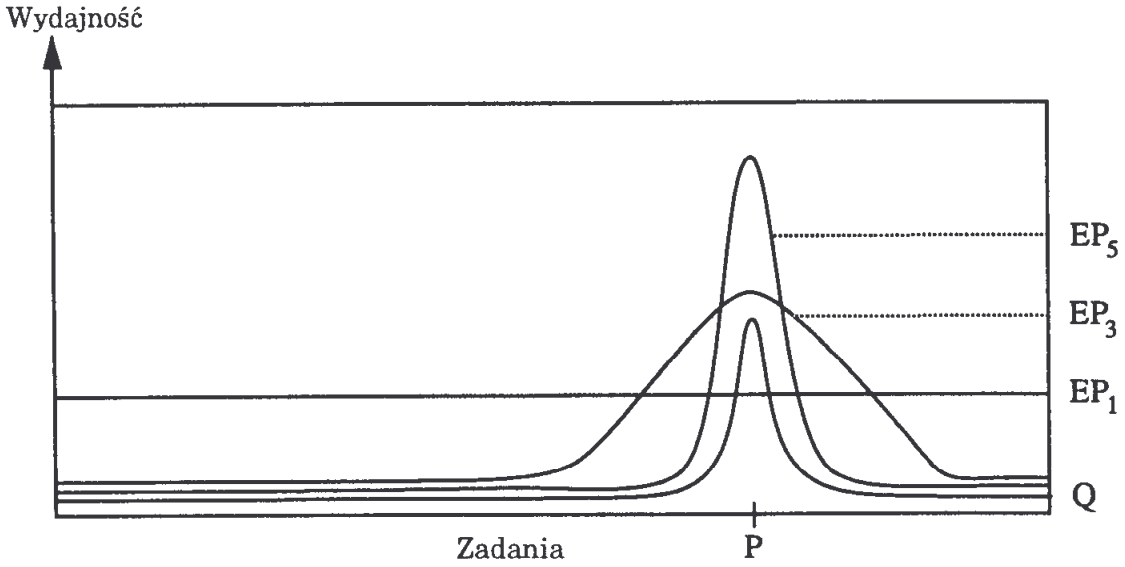
> Oczywiście ludzie opracowali bardzo wydajne procedury przeszukiwania dla wąskich klas zadań – algorytmy genetyczne mają małą szansę pobić metody kierunków sprzężonych lub gradientowe w zadaniach optymalizacji kwadratowej ciągłej. Ale to mija się z celem. [...] Szerokość połączona z umiarkowaną – nie szczytową – wydajnością określa podstawowy temat przeszukiwania genetycznego: stabilność.

Przedstawiliśmy tę uwagę na rys. 14.2, gdzie (klasyczna) metoda Q działa lepiej dla zadania P, ale nigdzie więcej. Tymczasem algorytmy genetyczne działają nieźle w całym spektrum zastosowań. (Rysunek 14.2 jest uproszczeniem podobnych rysunków podanych w [156] i [154]).



Rys. 14.2. Spektrum zastosowań i algorytmy genetyczne w układzie wydajność – zadania

Jednak przy obecności nietrywialnych, trudnych ograniczeń działanie algorytmów genetycznych często się pogarsza. Natomiast programy ewolucyjne, przez włączenie pewnej wiedzy specyficznej dla zadania, mogą pokonać nawet metody klasyczne (rys. 14.3).



Rys. 14.3. Spektrum zastosowań i programy ewolucyjne w układzie wydajność – zadania

Powinniśmy jednak znowu podkreślić, że większość programów ewolucyjnych przedstawionych w książce nie ma podstaw teoretycznych. Nie ma dla nich ani twierdzenia o schematach (jak dla klasycznych algorytmów genetycznych), ani twierdzeń o zbieżności (jak dla strategii ewolucyjnych). Należy także podkreślić, że programy ewolucyjne są na ogół znacznie wolniejsze niż inne metody optymalizacji. Jednakże ich czas działania często rośnie liniowo (lub jak $n \log n$) z wymiarowością zadania, co nie zachodzi dla większości innych metod. Także ostatnie prace Nicka Radcliffa [315], [316], [317] na temat analizy formy i właściwości operatorów genetycznych (na przykład odpowiednich rekombinacji) stosowanych do dowolnych struktur danych są ważnym krokiem w kierunku uzyskania pewnej podstawy teoretycznej dla programów ewolucyjnych.

Powyższą dyskusję potwierdzono ostatnio [261] serią eksperymentów. Pomysł mocniejszych i słabszych programów ewolucyjnych sprawdzono na pewnym szczególnym zadaniu P (nieliniowym zadaniu transportowym) i pięciu programach ewolucyjnych EP_i ($i = 1, ..., 5$). Omówimy te eksperymenty w następnych akapitach.

Określmy pewne szczególne zbilansowane zadanie transportowe P. Załóżmy, że są 3 punkty nadania i 4 punkty odbioru. Zapasy towarów są następujące:

$$source(1) = 10,\ source(2) = 15,\ source(3) = 20$$

a zapotrzebowanie:

$$dest(1) = 3,\ dest(2) = 20,\ dest(3) = 5,\ dest(4) = 17$$

Całkowity przewóz wynosi więc 45. Jak przedstawiono w rozdz. 9, rozwiązanie optymalne nieliniowego zadania transportowego nie może zawierać

ani zera, ani wartości całkowitych (jak to jest w liniowym zadaniu transportowym). Na przykład dla pewnych funkcji kosztów przewozu f_{ij} mogłoby być optymalne następujące rozwiązanie:

Przewożona ilość

	3,0	**20,0**	**5,0**	**17,0**
10,0	1,34	1,52	0,01	7,13
15,0	1,15	10,39	0,39	3,07
20,0	0,51	8,09	4,60	6,80

W naszym zadaniu testowym P użyto tych samych funkcji f dla każdego przewozu f_{ij}. Do zróżnicowania ich użyto macierzy kosztów. Macierz zawiera współczynniki c_{ij}, które skalują podstawowy kształt funkcji.

Przyjęto następującą funkcję f dla przewozu x_{ij}:

$$f(x_{ij}) = \begin{cases} 0, & \text{jeżeli } x_{ij} = 0 \\ d + c_{ij} \cdot \sqrt{x_{ij}}, & \text{w pozostałym przypadku} \end{cases}$$

dla $i = 1, 2, 3, j = 1, 2, 3, 4$, przy czym $d = 5{,}0$ oraz

$$\begin{array}{llll} c_{11} = 0{,}0 & c_{12} = 21{,}0 & c_{13} = 50{,}0 & c_{14} = 62{,}0 \\ c_{21} = 21{,}0 & c_{22} = 0{,}0 & c_{23} = 17{,}0 & c_{24} = 54{,}0 \\ c_{31} = 50{,}0 & c_{32} = 17{,}0 & c_{33} = 0{,}0 & c_{34} = 60{,}0 \end{array}$$

Tak więc zadanie P polega na minimalizacji

$$\sum_{i=1}^{3} \sum_{j=1}^{4} f(x_{ij})$$

przy następujących ograniczeniach:

$$x_{11} + x_{12} + x_{13} + x_{14} = 10$$
$$x_{21} + x_{22} + x_{23} + x_{24} = 15$$
$$x_{31} + x_{32} + x_{33} + x_{34} = 20$$
$$x_{11} + x_{21} + x_{31} = 3$$
$$x_{12} + x_{22} + x_{32} = 20$$
$$x_{13} + x_{23} + x_{33} = 5$$
$$x_{14} + x_{24} + x_{34} = 17$$

Zadanie P rozwiązano za pomocą systemu GAMS (patrz rozdz. 6). Najlepsze rozwiązanie znalezione przez GAMS to:

$$\sum_{i=1}^{3} \sum_{j=1}^{4} f(x_{ij}) = 430{,}64$$

co uzyskano dla

$$\begin{array}{llll} x_{11} = 3{,}0 & x_{12} = 0{,}0 & x_{13} = 0{,}0 & x_{14} = 7{,}0 \\ x_{21} = 0{,}0 & x_{22} = 5{,}0 & x_{23} = 0{,}0 & x_{24} = 10{,}0 \\ x_{31} = 0{,}0 & x_{32} = 15{,}0 & x_{33} = 5{,}0 & x_{34} = 0{,}0 \end{array}$$

Ten wynik posłużył jako wygodny punkt odniesienia do oceny przedstawionych tu programów ewolucyjnych. Powołujemy się na GAMS jako na klasyczną (gradientową) metodę Q dla zadania P (patrz rys. 14.3).

W celu uczciwego porównania programów ewolucyjnych EP_i ($i = 1, ..., 5$) ustalono w obliczeniach liczebność populacji na 70 oraz liczbę pokoleń na 5 000. Każde obliczenie powtarzano 20 razy. Wszystkie średnie dla poszczególnych obliczeń podane w dalszych punktach odnoszą się do średnich obliczonych z tych 20 przebiegów. Należy także podkreślić, że przedstawione programy ewolucyjne obejmują różne sposoby rozpoczęcia obliczeń. Jednak to omówimy trochę dalej.

Program ewolucyjny EP_1

Najsłabszym programem ewolucyjnym EP_1 użytym w obliczeniach był system GENESIS 1.2ucsd[1)] opracowany przez Nicola Schraudolpha z Uniwersytetu Kalifornijskiego w San Diego (system jest oparty na GENESIS 4.5, pakiecie algorytmów genetycznych napisanym przez Johna Grefenstette'a). W zasadzie można używać takiego podstawowego narzędzia do optymalizacji różnorodnych zadań i dziedzina $dom(EP_1)$ jest właściwie nieograniczona.

Sprawdzimy użyteczność tego programu ewolucyjnego na naszym zadaniu testowym P. Jest jasne, że system nie da żadnego użytecznego rozwiązania, jeżeli nie uwzględnimy ograniczeń za pomocą funkcji kar. Na przykład zrobiono kilka obliczeń EP_1, podając tylko dziedzinę dla każdej z dwunastu zmiennych. Nie mieliśmy tu dużego wyboru – dziedzinę dla każdej zmiennej określono od zera do najmniejszej sumy danego wiersza lub kolumny

$$\begin{array}{llll} 0{,}0 \leqslant x_{11} \leqslant 3{,}0 & 0{,}0 \leqslant x_{12} \leqslant 10{,}0 & 0{,}0 \leqslant x_{13} \leqslant 5{,}0 & 0{,}0 \leqslant x_{14} \leqslant 10{,}0 \\ 0{,}0 \leqslant x_{21} \leqslant 3{,}0 & 0{,}0 \leqslant x_{22} \leqslant 15{,}0 & 0{,}0 \leqslant x_{23} \leqslant 5{,}0 & 0{,}0 \leqslant x_{24} \leqslant 15{,}0 \\ 0{,}0 \leqslant x_{31} \leqslant 3{,}0 & 0{,}0 \leqslant x_{32} \leqslant 20{,}0 & 0{,}0 \leqslant x_{33} \leqslant 5{,}0 & 0{,}0 \leqslant x_{34} \leqslant 17{,}0 \end{array}$$

[1)] Ten system liczono z kodowaniem parametrów dynamicznych (Schraudolph, Belew, 1992). Jednak nie poprawia to oceny działania systemu, gdyż nie bierze się tu pod uwagę dokładności. Ta sama uwaga dotyczy programu ewolucyjnego EP_5 omawianego dalej.

Oczywiście żadne ze znalezionych przez program rozwiązań nie spełniało ograniczeń zadania. Typowy wynik podajemy poniżej:

$$x_{11} = 2{,}05 \quad x_{12} = 0{,}00 \quad x_{13} = 0{,}00 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 10{,}65 \quad x_{23} = 0{,}00 \quad x_{24} = 0{,}00$$
$$x_{31} = 0{,}00 \quad x_{32} = 0{,}00 \quad x_{33} = 0{,}00 \quad x_{34} = 0{,}00$$

Tak jak się spodziewano, powyższe rozwiązanie niedopuszczalne nie ma żadnej wartości dla użytkownika. Może ono nawet być jeszcze bardziej „poprawione". Rozwiązanie $x_{ij} = 0{,}0$ dla wszystkich $1 \leqslant i \leqslant 3$, $1 \leqslant j \leqslant 4$ daje optymalny koszt transportu (zero)!

Oczywiście należy nałożyć pewne kary związane z ograniczeniami. Ponieważ program ewolucyjny EP_1 nie powinien zależeć od rozwiązywanego zadania, próbowano tylko pewnych standardowych funkcji kar. Rozważono dwa zbiory takich funkcji. Pierwszy (umiarkowane kary p_i), to kary w postaci liniowych funkcji od naruszenia ograniczeń. Drugi (wysokie kary q_i), to kary kwadratowe za naruszenia ograniczeń. Dla naszego zadania P z siedmioma równościami liniowymi te funkcje wyglądały następująco:

$$p_1 = c \cdot |x_{11} + x_{12} + x_{13} + x_{14} - 10|$$
$$p_2 = c \cdot |x_{21} + x_{22} + x_{23} + x_{24} - 15|$$
$$p_3 = c \cdot |x_{31} + x_{32} + x_{33} + x_{34} - 20|$$
$$p_4 = c \cdot |x_{11} + x_{21} + x_{31} - 3|$$
$$p_5 = c \cdot |x_{12} + x_{22} + x_{32} - 20|$$
$$p_6 = c \cdot |x_{13} + x_{23} + x_{33} - 5|$$
$$p_7 = c \cdot |x_{14} + x_{24} + x_{34} - 17|$$

oraz $q_i = p_i^2/c$ ($i = 1, ..., 7$). We wszystkich obliczeniach przyjęto $c = 10{,}0$. Dla tej wartości stałej kary stanowią znaczący procent całkowitego kosztu (który, jak wynikało z wyników uzyskanych za pomocą systemu GAMS, wynosi około 400). Wyniki obliczeń są całkiem interesujące.

Następujący punkt reprezentuje *typowe rozwiązanie* dla obliczenia z karami p_i:

$$x_{11} = 3{,}00 \quad x_{12} = 3{,}77 \quad x_{13} = 0{,}00 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 1{,}23 \quad x_{23} = 0{,}00 \quad x_{24} = 13{,}77$$
$$x_{31} = 0{,}00 \quad x_{32} = 15{,}00 \quad x_{33} = 5{,}00 \quad x_{34} = 0{,}00$$

Powyższe rozwiązanie jest po prostu „typowe". Nie możemy podać *najlepszego* rozwiązania, gdyż jest stosunkowo trudno ocenić poprawność rozwiązania niedopuszczalnego. Aby dostać rozwiązanie dopuszczalne z niedopuszczalnego, musimy zrobić kilka poprawek, co rzutuje na końcowy koszt przewozów. Na

przykład powyższe rozwiązanie można skorygować do następującego rozwiązania dopuszczalnego:

$$x_{11} = 3{,}00 \quad x_{12} = 3{,}77 \quad x_{13} = 0{,}00 \quad x_{14} = 3{,}23$$
$$x_{21} = 0{,}00 \quad x_{22} = 1{,}23 \quad x_{23} = 0{,}00 \quad x_{24} = 13{,}77$$
$$x_{31} = 0{,}00 \quad x_{32} = 15{,}00 \quad x_{33} = 5{,}00 \quad x_{34} = 0{,}00$$

które daje całkowity koszt przewozów 453,43. Oczywiście inne korekcje dadzą lepszy lub gorszy koszt przewozów. (Powyższą korekcję zrobiono ręcznie. Polega ona na prostym zauważeniu, że całkowity koszt w pierwszym wierszu i czwartej kolumnie są mniejsze niż powinny być o 3,23. Wobec tego dodano 3,23 do x_{14}.)

W powyższym przykładzie ręczna korekta rozwiązania niedopuszczalnego dała niezłą wartość 453,43. Trzeba jednak podkreślić, że było to możliwe tylko ze względu na niską wymiarowość zadania. Proces wyznaczania „dobrej" korekcji rozwiązania niedopuszczalnego dla zadania transportowego o wymiarach 20×20 może być tak samo trudny jak rozwiązanie całego zadania. Wygląda na to, że trzeba użyć mocniejszych kar, aby wymusić przesunięcie rozwiązania do obszaru dopuszczalnego.

Rzeczywiście przyjęcie mocniejszych kar dało rozwiązanie, które było „prawie" dopuszczalne. Następujący punkt reprezentuje najlepsze rozwiązanie dla obliczeń z karami q_i:

$$x_{11} = 3{,}00 \quad x_{12} = 6{,}98 \quad x_{13} = 0{,}00 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 0{,}00 \quad x_{23} = 3{,}06 \quad x_{24} = 11{,}93$$
$$x_{31} = 0{,}00 \quad x_{32} = 13{,}02 \quad x_{33} = 1{,}93 \quad x_{34} = 5{,}03$$

Powyższe rozwiązanie można łatwo (przez ręczne zaokrąglenie) doprowadzić do rozwiązania dopuszczalnego

$$x_{11} = 3{,}00 \quad x_{12} = 7{,}00 \quad x_{13} = 0{,}00 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 0{,}00 \quad x_{23} = 3{,}00 \quad x_{24} = 12{,}00$$
$$x_{31} = 0{,}00 \quad x_{32} = 13{,}00 \quad x_{33} = 2{,}00 \quad x_{34} = 5{,}00$$

które daje całkowity koszt przewozów równy 502,53. Jest on gorszy niż koszt 453,43 uzyskany z umiarkowanymi karami. Jednak należy ponownie podkreślić, że proces znajdowania „dobrej" korekty dla rozwiązania uzyskanego przy umiarkowanych karach może być dosyć skomplikowany dla zadań o większych wymiarach. Możemy myśleć o tym kroku jako o procesie rozwiązywania nowego zadania transportowego ze zmodyfikowanymi sumami wierszy i kolumn (będącymi różnicą między otrzymanymi a prawidłowymi sumami), w których zmienne, powiedzmy δ_{ij}, reprezentują odpowiednie korekcje początkowych zmiennych x_{ij}. Tak więc, w ogólności mocniejsze kary dają lepsze wyniki. Jednocześnie te wyniki są nadal gorsze niż wyniki uzyskane z oprog-

ramowania komercyjnego GAMS (system Q na rys. 14.3). Należy także dodać, że „bardzo mocne" kary nie poprawiają działania programu. W krańcowym przypadku, gdy przyporządkujemy osobnikom naruszającym ograniczenia dopasowanie zerowe, wtedy system często stanie na pierwszym znalezionym rozwiązaniu dopuszczalnym.

Końcowy (i do przewidzenia) wniosek z obliczeń z EP_1 jest taki, że użycie funkcji kar nie gwarantuje rozwiązań dopuszczalnych i że „dobra" naprawa może być kosztowna.

Program ewolucyjny EP_2

Jak przedstawiono w rozdz. 8, w strategiach ewolucyjnych przyjmuje się, że w skład zadania optymalizacji wchodzi zbiór $q \leqslant 0$ nierówności

$$g_1(\mathbf{x}) \geqslant 0, \ldots, g_q(\mathbf{x}) \geqslant 0$$

Jeżeli w jakiejś iteracji potomek nie spełnia tych ograniczeń, to jest on dyskwalifikowany, to znaczy nie umieszcza się go w nowej populacji. Jeżeli częstość pojawiania się takich niedopuszczalnych potomków jest duża, to zmienia się parametry sterujące, na przykład zmniejszając składowe wektora $\boldsymbol{\sigma}$.

Użyliśmy KORR 2.1, programu Hansa-Paula Schwefela i Franka Hoffmeistera dla $(\mu + \lambda)$-SE i (μ, λ)-SE, jako naszego następnego programu ewolucyjnego EP_2. Oczywiście strategie ewolucyjne można stosować do zadań optymalizacji parametrycznej, więc $dom(EP_2) \subseteq dom(EP_1)$, a w konsekwencji $EP_1 \prec EP_2$.

Jak przedstawiono wcześniej, EP_2 obejmuje tylko ograniczenia nierównościowe. Z tego powodu zadanie P przepisano, aby wyeliminować równości. W wyniku tego funkcja celu ma tylko sześć zmiennych: $y_1, y_2, y_3, y_4, y_5, y_6$ i zadanie transportowe wygląda następująco:

$$\begin{aligned}\min f(y_1) + f(y_2) + f(y_3) + f(10{,}0 - y_1 - y_2 - y_3) + f(y_4) + f(y_5) + \\ + f(y_6) + f(15{,}0 - y_4 - y_5 - y_6) + f(3{,}0 - y_1 - y_4) + \\ + f(20{,}0 - y_2 - y_5) + f(5{,}0 - y_3 - y_6) + \\ + f(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8{,}0)\end{aligned}$$

gdzie

$$y_1 = x_{11}, \quad y_2 = x_{12}, \quad y_3 = x_{13}, \quad y_4 = x_{21}, \quad y_5 = x_{22}, \quad y_6 = x_{23}$$

i zachodzi osiemnaście ograniczeń:

g_1 : $y_1 \geqslant 0$ (to znaczy $x_{11} \geqslant 0$)
g_2 : $y_2 \geqslant 0$ (to znaczy $x_{12} \geqslant 0$)
g_3 : $y_3 \geqslant 0$ (to znaczy $x_{13} \geqslant 0$)
g_4 : $y_4 \geqslant 0$ (to znaczy $x_{21} \geqslant 0$)

g_5 : $y_5 \geqslant 0$ (to znaczy $x_{22} \geqslant 0$)
g_6 : $y_6 \geqslant 0$ (to znaczy $x_{23} \geqslant 0$)
g_7 : $10{,}0 - y_1 - y_2 - y_3 \geqslant 0$ (to znaczy $x_{14} \geqslant 0$)
g_8 : $15{,}0 - y_4 - y_5 - y_6 \geqslant 0$ (to znaczy $x_{24} \geqslant 0$)
g_9 : $3{,}0 - y_1 - y_4 \geqslant 0$ (to znaczy $x_{31} \geqslant 0$)
g_{10}: $20{,}0 - y_2 - y_5 \geqslant 0$ (to znaczy $x_{32} \geqslant 0$)
g_{11}: $5{,}0 - y_3 - y_6 \geqslant 0$ (to znaczy $x_{33} \geqslant 0$)
g_{12}: $y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8{,}0 \geqslant 0$ (to znaczy $x_{34} \geqslant 0$)
g_{13}: $3{,}0 - y_1 \geqslant 0$ (to znaczy $x_{11} \leqslant 3$)
g_{14}: $10{,}0 - y_2 \geqslant 0$ (to znaczy $x_{12} \leqslant 10$)
g_{15}: $5{,}0 - y_3 \geqslant 0$ (to znaczy $x_{13} \leqslant 5$)
g_{16}: $3{,}0 - y_4 \geqslant 0$ (to znaczy $x_{21} \leqslant 3$)
g_{17}: $15{,}0 - y_5 \geqslant 0$ (to znaczy $x_{22} \leqslant 15$)
g_{18}: $5{,}0 - y_6 \geqslant 0$ (to znaczy $x_{23} \leqslant 5$)

Średnia wartość najlepszego znalezionego przez EP_2 kosztu przewozu (z 20 niezależnych obliczeń) wynosiła 460,75, a najlepsze wyznaczone rozwiązanie (które dało całkowitą wartość kosztu 420,74) wyglądało następująco:

$$\begin{array}{llll} x_{11} = 3{,}00 & x_{12} = 2{,}00 & x_{13} = 5{,}00 & x_{14} = 0{,}00 \\ x_{21} = 0{,}00 & x_{22} = 0{,}00 & x_{23} = 0{,}00 & x_{24} = 15{,}00 \\ x_{31} = 0{,}00 & x_{32} = 18{,}00 & x_{33} = 0{,}00 & x_{34} = 2{,}00 \end{array}$$

Jak się spodziewano, wyniki EP_2 są lepsze niż wyniki z poprzedniego programu ewolucyjnego EP_1. Dodatkowym punktem dla EP_2 jest to, że nie potrzeba korygować wyników, aby przesunąć je do obszaru dopuszczalnego. Wygląda jednak na to, że działanie EP_2 zależy od punktu początkowego w przestrzeni rozwiązań (który jest podany przez użytkownika). Z tego powodu jest dosyć trudno przedstawić całkowitą analizę systemu.

Program ewolucyjny EP_3

Trzecim opisanym tu programem jest GENOCOP (rozdz. 7). Ponieważ GENOCOP (jako nasz program ewolucyjny EP_3) może tylko uwzględnić ograniczenia liniowe, więc jest jasne, że $dom(EP_3) \subseteq dom(EP_2)$ i w konsekwencji $EP_2 \prec EP_3$.

Zadanie transportowe P jest zadaniem z $m = 12$ zmiennymi. Każdy chromosom jest zakodowany jako wektor o dwunastu liczbach zmiennopozycyjnych $(y_1, ..., y_{12})$. Tak więc zadanie P ma postać

$$\min \sum_{i=1}^{12} f(y_i)$$

gdzie

$$y_1 = x_{11}, \quad y_2 = x_{12}, \quad y_3 = x_{13}, \quad y_4 = x_{14},$$
$$y_5 = x_{21}, \quad y_6 = x_{22}, \quad y_7 = x_{23}, \quad y_8 = x_{24},$$
$$y_9 = x_{31}, \quad y_{10} = x_{32}, \quad y_{11} = x_{33}, \quad y_{12} = x_{34},$$

z sześcioma niezależnymi ograniczeniami liniowymi:

$$y_1 + y_2 + y_3 + y_4 = 10$$
$$y_5 + y_6 + y_7 + y_8 = 15$$
$$y_9 + y_{10} + y_{11} + y_{12} = 20$$
$$y_1 + y_5 + y_9 = 3$$
$$y_2 + y_6 + y_{10} = 20$$
$$y_3 + y_7 + y_{11} = 5$$

(Siódme równanie $y_4 + y_8 + y_{12} = 10$ jest niepotrzebne, gdyż jest ono liniowo zależne od podanych wyżej sześciu równań). Dodatkowymi ograniczeniami liniowymi są

$$y_i \geqslant 0 \qquad \text{dla} \qquad i = 1, ..., 12$$

Wykonano 20 obliczeń GENOCOP-u. Wartości całkowitego kosztu przewozów wahały się od 420,74 (najgorszy przypadek) dla następującego rozwiązania (zaokrąglonego do drugiej cyfry po przecinku):

$$x_{11} = 3{,}00 \quad x_{12} = 4{,}38 \quad x_{13} = 2{,}62 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 15{,}00 \quad x_{23} = 0{,}00 \quad x_{24} = 0{,}00$$
$$x_{31} = 0{,}00 \quad x_{32} = 0{,}62 \quad x_{33} = 2{,}38 \quad x_{34} = 17{,}00$$

do wartości 356,98 (najlepszy przypadek) dla rozwiązania

$$x_{11} = 3{,}00 \quad x_{12} = 7{,}00 \quad x_{13} = 0{,}00 \quad x_{14} = 0{,}00$$
$$x_{21} = 0{,}00 \quad x_{22} = 13{,}00 \quad x_{23} = 2{,}00 \quad x_{24} = 0{,}00$$
$$x_{31} = 0{,}00 \quad x_{32} = 0{,}00 \quad x_{33} = 3{,}00 \quad x_{34} = 17{,}00$$

Średni (z 20 obliczeń) koszt przewozów dla systemu GENOCOP wyniósł 405,45. Oczywiście wszystkie uzyskane rozwiązania były dopuszczalne. GENOCOP jako system bardziej dostosowany do zadania działał wyraźnie dużo lepiej niż strategie ewolucyjne EP_2.

Program ewolucyjny EP_4

Następnym opisanym tu programem ewolucyjnym EP_4 jest GENETIC-2. Jak przedstawiono w rozdz. 9, system ten był zbudowany do optymalizacji jakiegokolwiek nieliniowego zadania transportowego, więc jest jasne, że $dom(EP_4) \subseteq dom(EP_3)$ i w konsekwencji $EP_3 \prec EP_4$. W GENETIC-2 potencjalne rozwiązanie jest reprezentowane przez macierz i dla tej reprezentacji określono odpowiednie operatory.

Wykonano 20 obliczeń GENETIC-2. Wartości całkowitego kosztu przewozów wahały się od 397,02 (najgorszy przypadek) dla rozwiązania (zaokrąglonego do drugiego miejsca po przecinku):

$$\begin{array}{llll} x_{11} = 3{,}00 & x_{12} = 5{,}00 & x_{13} = 0{,}00 & x_{14} = 2{,}00 \\ x_{21} = 0{,}00 & x_{22} = 15{,}00 & x_{23} = 0{,}00 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}00 & x_{33} = 5{,}00 & x_{34} = 15{,}00 \end{array}$$

do wartości 356,98 (najlepszy przypadek) dla rozwiązania

$$\begin{array}{llll} x_{11} = 3{,}00 & x_{12} = 7{,}00 & x_{13} = 0{,}00 & x_{14} = 0{,}00 \\ x_{21} = 0{,}00 & x_{22} = 13{,}00 & x_{23} = 2{,}00 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}00 & x_{33} = 3{,}00 & x_{34} = 17{,}00 \end{array}$$

(tego samego rozwiązania, które znalazł GENOCOP). Jednak średnia (znowu z 20 obliczeń) kosztu przewozów z systemu GENETIC-2 wyniosła 391,65, a więc była dużo lepsza niż 405,45 uzyskane z GENOCOP-u. Ponownie wszystkie rozwiązania były dopuszczalne. GENETIC-2 (EP_4), jako system bardziej dopasowany do zadania działał wyraźnie lepiej niż GENOCOP (EP_3).

Program ewolucyjny EP_5

Ostatni opisany tutaj program ewolucyjny EP_5 jest znowu oparty na systemie GENESIS 1.2ucsd, tym samym, którego użyliśmy do obliczeń wcześniej. Tym razem jednak spróbowano „dostroić" zbiór funkcji kar, aby ukierunkować system dokładnie na rozwiązanie zadania P. Dodatkowo wyeliminowano wszystkie równości, gdyż intuicyjnie wydaje się, że łatwiej jest uwzględnić nierówności niż równości.

Tak więc znowu zadanie P przepisano jako

$$\begin{aligned} \min f(y_1) + f(y_2) + f(y_3) + f(10{,}0 - y_1 - y_2 - y_3) + f(y_4) + f(y_5) + \\ + f(y_6) + f(15{,}0 - y_4 - y_5 - y_6) + f(3{,}0 - y_1 - y_4) + \\ + f(20{,}0 - y_2 - y_5) + f(5{,}0 - y_3 - y_6) + \\ + f(y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8{,}0) \end{aligned}$$

gdzie $y_1 = x_{11}$, $y_2 = x_{12}$, $y_3 = x_{13}$, $y_4 = x_{21}$, $y_5 = x_{22}$, $y_6 = x_{23}$ oraz

$$\begin{aligned} &0{,}0 \leqslant y_1 \leqslant 3{,}0 \\ &0{,}0 \leqslant y_2 \leqslant 10{,}0 \\ &0{,}0 \leqslant y_3 \leqslant 5{,}0 \\ &0{,}0 \leqslant y_4 \leqslant 3{,}0 \\ &0{,}0 \leqslant y_5 \leqslant 15{,}0 \\ &0{,}0 \leqslant y_6 \leqslant 5{,}0 \end{aligned}$$

Próbowano dostroić sześć funkcji kar:

$$p_1 = \begin{cases} w_1 \cdot (c_1 + y_1 + y_2 + y_3 - 10{,}0)^2, & \text{jeżeli } 10{,}0 - y_1 - y_2 - y_3 < 0{,}0 \\ 0{,}0 & \text{w pozostałym przypadku} \end{cases}$$

$$p_2 = \begin{cases} w_2 \cdot (c_2 + y_4 + y_5 + y_6 - 15{,}0)^2, & \text{jeżeli } 15{,}0 - y_4 - y_5 - y_6 < 0{,}0 \\ 0{,}0 & \text{w pozostałym przypadku} \end{cases}$$

$$p_3 = \begin{cases} w_3 \cdot (c_3 + y_1 + y_4 + \ - 3{,}0)^2, & \text{jeżeli } 3{,}0 - y_1 - y_4 < 0{,}0 \\ 0{,}0 & \text{w pozostałym przypadku} \end{cases}$$

$$p_4 = \begin{cases} w_4 \cdot (c_4 + y_2 + y_5 - 20{,}0)^2, & \text{jeżeli } 20{,}0 - y_2 - y_5 < 0{,}0 \\ 0{,}0 & \text{w pozostałym przypadku} \end{cases}$$

$$p_5 = \begin{cases} w_5 \cdot (c_5 + y_3 + y_5 - 5{,}0)^2, & \text{jeżeli } 5{,}0 - y_3 - y_6 < 0{,}0 \\ 0{,}0 & \text{w pozostałym przypadku} \end{cases}$$

$$p_6 = \begin{cases} w_6 \cdot (c_6 + 8{,}0 - y_1 - y_2 - y_3 - y_4 - y_5 - y_6)^2, \\ \qquad \text{jeżeli } y_1 + y_2 + y_3 + y_4 + y_5 + y_6 - 8{,}0 < 0{,}0 \\ 0{,}0 \qquad \text{w pozostałym przypadku} \end{cases}$$

gdzie w_i oraz c_i są dodatkowymi wagami.

Jak zwykle wszystkie kary dodawano do funkcji celu. Po wielu eksperymentach (w których, odpowiednio, zwiększano lub zmniejszano odpowiednie wagi dla ograniczeń, które były naruszane lub spełniane), doszliśmy do następującego zbioru:

$c_1 = 2{,}5 \quad w_1 = 2{,}0$
$c_2 = 0{,}3 \quad w_2 = 1{,}3$
$c_3 = 5{,}0 \quad w_3 = 2{,}5$
$c_4 = 5{,}0 \quad w_4 = 2{,}0$
$c_5 = 0{,}2 \quad w_5 = 1{,}3$
$c_6 = 0{,}1 \quad w_6 = 2{,}0$

Nie twierdzimy, oczywiście, że powyższy zbiór wag jest optymalną konfiguracją. Dostrojenie było wykonane po prostu „ręcznie". Jeżeli pewne ograniczenie nie było spełnione, to stopniowo zwiększaliśmy odpowiednie wagi. Jednak możemy zrobić następujące dwie uwagi:

- system EP_5 z powyższymi wagami działa całkiem dobrze dla zadania P,
- jeżeli zmienimy zadanie P, dodając jeszcze jeden punkt nadania lub odbioru lub po prostu zmieniając wagi zadania c_{ij}, to program ewolucyjny EP_5 nie daje dobrych wyników.

Jest więc jasne, że $dom(EP_5) \subseteq dom(EP_4)$ i w konsekwencji $EP_4 \prec EP_5$.

Jedno z obliczeń za pomocą systemu EP_5 dało następujące rozwiązanie:

$$\begin{array}{llll} x_{11} = 2{,}93 & x_{12} = 6{,}91 & x_{13} = 0{,}16 & x_{14} = 0{,}00 \\ x_{21} = 0{,}07 & x_{22} = 13{,}09 & x_{23} = 1{,}84 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}00 & x_{33} = 3{,}00 & x_{34} = 17{,}00 \end{array}$$

Zauważmy, że wszystkie ograniczenia są spełnione i że wartość funkcji celu wynosi 391,2. Powyższe rozwiązanie można ręcznie skorygować do najlepszego rozwiązania znalezionego przez GENOCOP i GENETIC-2

$$\begin{array}{llll} x_{11} = 3{,}00 & x_{12} = 7{,}00 & x_{13} = 0{,}00 & x_{14} = 0{,}00 \\ x_{21} = 0{,}00 & x_{22} = 13{,}00 & x_{23} = 2{,}00 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}00 & x_{33} = 3{,}00 & x_{34} = 17{,}00 \end{array}$$

Jednak system EP_5 znalazł także rozwiązanie z lepszą wartością niż 391,2, a mianowicie 378,25 dla następującego planu przewozów:

$$\begin{array}{llll} x_{11} = 2{,}53 & x_{12} = 7{,}47 & x_{13} = 0{,}00 & x_{14} = 0{,}00 \\ x_{21} = 0{,}47 & x_{22} = 12{,}53 & x_{23} = 2{,}00 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}00 & x_{33} = 3{,}00 & x_{34} = 17{,}00 \end{array}$$

które jest znacznie trudniej skorygować (pamiętamy, że optymalne rozwiązanie nie musi się składać z liczb całkowitych, na przykład jedno z rozwiązań otrzymanych z GENETIC-2 wynosiło

$$\begin{array}{llll} x_{11} = 3{,}00 & x_{12} = 7{,}00 & x_{13} = 0{,}00 & x_{14} = 0{,}00 \\ x_{21} = 0{,}00 & x_{22} = 12{,}25 & x_{23} = 2{,}75 & x_{24} = 0{,}00 \\ x_{31} = 0{,}00 & x_{32} = 0{,}75 & x_{33} = 2{,}25 & x_{34} = 17{,}00 \end{array}$$

z całkowitym kosztem przewozów równym 380,86).

W ogólności powinno być możliwe skonstruowanie „doskonałego" programu ewolucyjnego, który jest przykrojony do zadania P. Możemy dołożyć dodatkową wiedzę do takiego systemu przez dołączenie kosztów przewozu c_{ij}, cech sześciu niezależnych ograniczeń, być może z jakimiś dodatkowymi heurystykami, aby zmodyfikować rozwiązania dopuszczalne. Można dodać dodatkowe ograniczenia, aby „naprowadzić" system na pożądany kierunek. Jednak należy dodać, że trudność skonstruowania takiego systemu rośnie z wymiarowością zadania i jego użyteczność byłaby całkiem ograniczona (tylko do zadania P).

Przedstawione wcześniej wyniki obliczeń potwierdziły intuicyjną hipotezę, że wiedza specyficzna dla zadania poprawia działanie algorytmu, zawężając jego możliwości zastosowania.

Jak wspomniano wcześniej, dla uczciwego porównania programów ewolucyjnych ustaliliśmy liczebność populacji na 70 i liczbę pokoleń na 5 000 we wszystkich obliczeniach, a obliczenia były powtarzane 20 razy. Jednak w omawianych programach ewolucyjnych stosuje się inne sposoby rozpoczęcia obliczeń. W pierwszym programie ewolucyjnym EP_1 generuje się populację w taki sposób, że osobniki nie muszą być dopuszczalne (ponieważ ograniczenia

obejmują równości, byłoby nawet dziwne, gdyby chociaż jeden z wygenerowanych osobników był dopuszczalny). W drugim programie ewolucyjnym EP_2 przyjmuje się jednego (dopuszczalnego) osobnika jako punkt początkowy. W powyższych obliczeniach wygenerowano dwadzieścia różnych dopuszczalnych punktów początkowych. W trzecim programie EP_3 dokonuje się kilku (co jest parametrem systemu) prób znalezienia początkowego osobnika dopuszczalnego w przestrzeni przeszukiwań. Jeżeli się to uda, to początkowa populacja składa się z *pop_size* jednakowych kopii znalezionego osobnika. Jeżeli się nie uda, to system zwraca się do użytkownika o podanie dopuszczalnego punktu początkowego. Zbiór dopuszczalnych punktów początkowych w powyższych obliczeniach był taki sam jak dla EP_2. W czwartym programie EP_4 generuje się i utrzymuje populację osobników dopuszczalnych, gdy tymczasem w EP_5 (podobnie jak w EP_4) generuje się początkową populację (być może) niedopuszczalnych osobników.

Przy porównywaniu naszych programów ewolucyjnych warto wiedzieć o tych różnicach w sposobach rozpoczęcia. Jednak wyniki obliczeń wykazują, że wpływ poszczególnych sposobów rozpoczynania na działanie systemu był nieznaczny. Nie jest to dziwne. Dla mocno ograniczonych zadań w ogólności (a dla zadania transportowego w szczególności) punkty „dopuszczalne" w przestrzeni przeszukiwań nie oznaczają punktów „dobrych". Heurystyczne sposoby rozpoczynania są dobre tylko w przypadkach, kiedy użytkownik zna dobrą heurystykę, którą można włączyć do systemu (a nawet wtedy trzeba to zrobić ostrożnie, aby uniknąć przedwczesnej zbieżności!). Nie było poprawy w EP_1 czy EP_5, kiedy rozpoczynano je od osobników dopuszczalnych, o ile jeden z nich nie był rzeczywiście dobry. W „sprytnym" rozpoczęciu dla programu ewolucyjnego EP_4 wygenerowano zbiór punktów dopuszczalnych ze średnią oceną 456. Nie poprawił on działania algorytmu. Po prostu trzeba rozpocząć od populacji dopuszczalnej, gdyż operatory w EP_4 utrzymują dopuszczalność. W innych programach (EP_2 i EP_3) użyto zbioru stosunkowo słabych punktów dopuszczalnych z wartościami dopasowania z zakresu [493, 610] (ze średnią 562).

Podsumowujemy więc stwierdzając, że proces rozpoczynania obliczeń nie miał wpływu na przedstawione wyniki.

Po tych wstępnych uwagach jesteśmy gotowi zająć sie dwiema praktycznymi sprawami związanymi z programowaniem ewolucyjnym. Dla danego zadania P

1) na ile słaby (lub mocny) powinien być program ewolucyjny?
2) jak powinniśmy postępować przy budowie programu ewolucyjnego?

Nie ma prostych odpowiedzi na te pytania. W dalszym ciągu przedstawimy kilka ogólnych uwag i zasad intuicyjnych utworzonych na podstawie obliczeń, pomieszanych z pewną dozą chęci spełnienia pragnień.

Pierwsza sprawa jest związana z optymalizacją wyboru tworzonego programu ewolucyjnego. Dla danego zadania P na ile słaby (lub mocny) powinien być program ewolucyjny? Inaczej mówiąc, czy dla danego zadania P powin-

niśmy raczej tworzyć EP_2 czy EP_4? Nasza hipoteza sugeruje, że włączenie wiedzy specyficznej dla zadania daje lepsze wyniki, jeżeli chodzi o dokładność. Jednak, jak stwierdzono we wprowadzeniu, opracowanie systemu mocniejszego, o lepszym działaniu, może zająć dużo czasu, jeżeli potrzebna jest do tego szersza analiza zadania w celu zaprojektowania specjalizowanej reprezentacji, operatorów i poprawienia działania. Natomiast mamy już pewne standardowe pakiety, jak GENESIS Grefenstette'a, GENITOR Whitleya, OOGA Davisa, GENESIS 1.2ucsd Schraudolpha, lub któryś z systemów strategii ewolucyjnych Schwefela. A jeżeli spóbujemy znaleźć efektywną reprezentację binarną dla danego zadania, to może to wymagać małej lub żadnej adaptacji oprogramowania!

Niekiedy tak, a niekiedy nie. Jeżeli ktoś rozwiązuje zadanie transportowe z mocnymi ograniczeniami (to znaczy ograniczeniami, które muszą być spełnione), to jest mała szansa, że jakiś pakiet standardowy da rozwiązania dopuszczalne, lub jeżeli zaczniemy od populacji rozwiązań dopuszczalnych i zmusimy system do ich utrzymywania, to możemy nie uzyskać jakiegokolwiek postępu – w takich przypadkach system nie działa lepiej niż program przypadkowego przeszukiwania. Natomiast dla pewnych zadań takie pakiety standardowe mogą dawać całkiem zadowalające wyniki. Krótko, odpowiedzialność za podjęcie decyzji w sprawie (1) spada na użytkownika. Decyzja zależy od wielu czynników, włączając w to wymaganie dokładności żądanego rozwiązania, skomplikowanie czasowe algorytmu, koszt opracowania nowego systemu, dopuszczalność wyznaczonego rozwiązania (to znaczy ważność ograniczeń zadania), częstość używania opracowanego systemu i inne.

Przypuśćmy więc, że (z pewnego powodu) musimy (lub chcemy) zbudować nowy system do rozwiązania nietrywialnego zadania optymalizacji. Może to być spowodowane tym, że standardowe pakiety algorytmów genetycznych nie dają akceptowanych rozwiązań dopuszczalnych, a nie ma pakietów komputerowych odpowiednich dla tego zadania. Wtedy musimy dokonać wyboru: albo próbujemy zbudować program ewolucyjny, albo staramy się rozwiązać zadanie, stosując metody tradycyjne (heurystyczne). Warto zauważyć, że w tradycyjnych metodach rozwiązywanie zadania optymalizacji zazwyczaj odbywa się w trzech krokach:

1) zrozumienie zadania,
2) rozwiązanie zadania,
3) oprogramowanie algorytmu opracowanego w poprzednim kroku.

W tradycyjnych metodach programista powinien rozwiązać zadanie – tylko wtedy można utworzyć prawidłowy program. Jednak bardzo często rozwiązanie algorytmiczne zadania nie jest możliwe, albo przynajmniej jest bardzo trudne. Ponadto w niektórych zastosowaniach nie jest ważne, aby znaleźć rozwiązanie optymalne – wystarczy jakiekolwiek rozwiązanie z rozsądnym marginesem błędu (względną odległością od wartości optymalnej). Na przykład w pewnych zadaniach transportowych można szukać *dobrego* planu przewozów – znalezienie optymalnego nie jest wymagane. W naszych obliczeniach

z niektórych systemów ewolucyjnych otrzymywano (stosunkowo szybko) rozwiązanie z wartością, powiedzmy, 1109, gdy tymczasem wartość optymalna wynosiła 1102. W takim przypadku (błąd mniejszy niż 1%) rozwiązanie przybliżone może być bardziej pożądane.

W metodach programowania ewolucyjnego zazwyczaj eliminuje się drugi, najtrudniejszy krok. Zaraz po tym, jak zrozumieliśmy zadanie, możemy zabrać się za sprawy oprogramowania. Głównym zadaniem programisty przy tworzeniu programu ewolucyjnego jest wybór odpowiednich struktur danych oraz operatorów „genetycznych", które będą na nich operowały (resztę pozostawia się procesowi ewolucyjnemu). To zadanie nie musi być trywialne, gdyż prócz różnorodności struktur danych, których można użyć do reprezentacji chromosomu, każda struktura danych może pozwalać na szeroki wybór operatorów genetycznych. Wymaga to zrozumienia natury zadania przez programistę. Jednak nie potrzeba rozwiązywać wcześniej zadania. Aby utworzyć program ewolucyjny, programista powinien wykonać pięć podstawowych kroków:

1. Po pierwsze wybrać reprezentację genetyczną dla rozwiązań zadania. Jak zauważono już wcześniej, wymaga to pewnego zrozumienia natury zadania. Jednak nie ma potrzeby rozwiązywania zadania. Wybrana reprezentacja rozwiązań zadania powinna być „naturalna" i ta decyzja jest pozostawiona programiście (zauważmy, że w obecnych środowiskach programowych programiści wybierają odpowiednie struktury danych na własną rękę). Wygląda na to, że jest to najbardziej istotny krok, który wpływa na pozostałe składniki programu ewolucyjnego. Reprezentacja powinna umożliwiać uwzględnienie w niej wszystkich ważnych informacji o rozwiązaniu. Niestety, nie ma gotowych wskazówek dla takiego wyboru. Zauważmy, że na tym koncentrują się podstawowe różnice między różnymi wzorcami obliczeń ewolucyjnych (łańcuchy binarne, wektory liczb zmiennopozycyjnych, automaty skończone, programy komputerowe itp.).

2. Drugim zadaniem programisty jest utworzenie początkowej populacji rozwiązań. Można to zrobić na wiele sposobów (losowo, za pomocą pewnego algorytmu heurystycznego itp.). Należy uważać w przypadkach, kiedy trzeba spełnić zbiór ograniczeń zadania. Często przyjmuje się populację *dopuszczalnych* rozwiązań jako dobry punkt początkowy programu ewolucyjnego (patrz następny rozdział). Jednak w wielu przypadkach nie znamy żadnego rozwiązania dopuszczalnego zadania. Niekiedy naprawa rozwiązania niedopuszczalnego jest tak trudna, jak znalezienie dopuszczalnego rozwiązania (na przykład w mocno ograniczonym zadaniu układania planu lekcji).

3. Wybór funkcji oceny (która ocenia rozwiązania w kategoriach ich dopasowania) nie powinien stanowić większej trudności dla wielu zadań optymalizacji. Jednak czasami to zadanie jest dalekie od trywialnego (patrz następny rozdział).

4. Operatory „genetyczne" powinny być zaprojektowane starannie – projekt powinien brać pod uwagę samo zadanie oraz jego ograniczenia. Jest tu

istotne zbadanie *znaczenia* informacji przekazywanej przez operatory. Jeżeli przeglądamy tylko dopuszczalną część przestrzeni rozwiązań, to operatory powinny zmieniać rozwiązanie(a) dopuszczalne w inne rozwiązania dopuszczalne. Można także użyć algorytmów naprawy, funkcji kar lub innych metod (patrz następny rozdział), aby zapewnić spełnienie ograniczeń zadania. Dla wielu rzeczywistych zadań jest więcej niż pomocne uwzględnienie heurystyk przeszukiwania lokalnego w operatorach.

5. Wartości różnych parametrów używanych przez program mogą być zadawane przez programistę. Jednak w bardziej zaawansowanych wersjach środowiska programowania ewolucyjnego mogą one być sterowane przez nadzorujące metaprocesy (jak omawiany w [169]), których jedynym zadaniem jest dostrojenie wszystkich parametrów. Coraz więcej badań jest skierowanych na samoadaptację parametrów programów ewolucyjnych.

Jak widać z powyższego przepisu na zbudowanie programu ewolucyjnego, ogólna idea leżąca u jego podstaw polega na użyciu „naturalnych" struktur danych oraz wrażliwych na zadanie „operatorów genetycznych". Jednak nadal może być trudne zbudowanie takiego systemu (dla poszczególnych zadań) od początku. Doświadczony programista powinien dać sobie z tym radę. Jednak otrzymany program może być całkiem niewydajny. Aby pomóc użytkownikowi w tym zadaniu, może warto stworzyć nową metodykę postępowania łącznie z oprogramowaniem (specjalizowanym językiem programowania) oraz sprzętem (komputery równoległe). To tutaj zaczynają się nasze chęci spełnienia pragnień.

Obecnie jest sporo różnych metodyk programowania w informatyce: programowanie strukturalne, programowanie logiczne, programowanie zorientowane obiektowo, programowanie funkcjonalne. Żadne z nich nie pasuje w pełni do budowy programu ewolucyjnego. Celem nowej metodyki byłoby utworzenie odpowiednich narzędzi do uczenia (tutaj optymalizację rozumie się jako proces nauczania), używając architektury procesów równoległych.

Mamy nadzieję rozwinąć ten pomysł przy zaprojektowaniu języka programowego PROBIOL (*PROgraming in BIOLogy*) w środowisku programowym EVA (*EVolution progrAmming*). Ważną sprawą w tej metodyce jest opracowanie programów do sterowania rozwojem procesów ewolucyjnych zachodzących w „maszynie ewolucyjnej" – maszyna ewolucyjna jest reprezentowana przez „towarzystwo mikroprocesorów" [296]. Niektóre z tych kwestii omówiono krótko w [204]. Kluczową motywacją dla nowego środowiska programowego jest opracowanie narzędzia programowego opartego na architekturze równoległej. Jest to ważne. Jak napisano w [6]:

> Równoległość z pewnością zmieni sposób, w jaki myślimy o użyciu komputerów. Obiecuje to, że w naszym zasięgu znajdzie się rozwiązywanie zadań i granice wiedzy, o których nigdy nie marzyliśmy wcześniej. Bogata różnorodność architektur doprowadzi do odkrycia nowych i bardziej wydajnych rozwiązań zarówno dla starych, jak i dla nowych problemów.

15

Programy ewolucyjne i heurystyki

„Oto nadszedł czas – rzecze Mors –
Pomówić o wielu sprawach:
O butach – laku – o kapuście –
O królach – i okrętach.
I czemu woda w morzu kipi –
I o skrzydlatych prosiętach".

Lewis Caroll, *Po drugiej stronie lustra*[1)]

Jak przedstawiono w poprzednich rozdziałach, najbardziej znane programy ewolucyjne obejmują algorytmy genetyczne, programowanie ewolucyjne, strategie ewolucyjne i programowanie genetyczne. Jest także wiele systemów hybrydowych, które zawierają różne cechy powyższych programów i w rezultacie są trudne do zaklasyfikowania. Nazywamy je po prostu programami ewolucyjnymi (lub algorytmami ewolucyjnymi albo metodami obliczeń ewolucyjnych). Jak wspomnieliśmy już kilka razy w tekście, jest ogólnie przyjęte, że każdy algorytm ewolucyjny do rozwiązywania zadań musi zawierać pięć podstawowych składników:

- reprezentację genetyczną rozwiązań zadania,
- sposób tworzenia początkowej populacji rozwiązań,
- funkcję oceny (to znaczy otoczenie), klasyfikującą rozwiązania według ich „dopasowania",
- operatory „genetyczne", które zmieniają układ genetyczny dzieci w czasie reprodukcji,
- wartości parametrów (liczebność populacji, prawdopodobieństwa użycia operatorów genetycznych itp.).

Jest powszechnie wiadomo, że w celu udanego zaprogramowania metod ewolucyjnych do konkretnego rzeczywistego zadania są potrzebne pewne dodatkowe reguły heurystyczne. Odnoszą się one do reprezentacji genetycznej rozwiązań, do operatorów „genetycznych", które zmieniają te rozwiązania, do wartości różnych parametrów, do metod tworzenia początkowej populacji. Wygląda na to, że tylko jeden punkt z powyżej wymienionych pięciu podstawowych składników algorytmu genetycznego – funkcja oceny – jest zazwyczaj zadana i nie wymaga żadnych modyfikacji heurystycznych. Rzeczywiście

[1)] Przekład Roberta Stillera, wyd. „Alfa", Warszawa 1986 (przyp. tłum.).

w wielu wypadkach proces wyboru funkcji oceny jest oczywisty (na przykład w klasycznych zadaniach numerycznych lub w optymalizacji kombinatorycznej). W rezultacie w ostatnich dwóch dziesięcioleciach przebadano wiele trudnych funkcji. Często służyły one za podstawę testowania różnych metod selekcji, różnych operatorów, różnych reprezentacji i tak dalej. Jednak proces wyboru funkcji oceny może być dosyć skomplikowany, szczególnie wtedy, kiedy mamy do czynienia z rozwiązaniami dopuszczalnymi i niedopuszczalnymi. Wtedy właśnie używa się reguł heurystycznych. W tym punkcie prześledzimy niektóre z nich i omówimy ich zalety i wady.

Jak napisano we wprowadzeniu, wszystkie programy ewolucyjne mają tę samą strukturę (rys. 0.1 we wprowadzeniu), ale występuje też między nimi wiele różnic (często ukrytych na niższych poziomach). Używają one innych struktur danych dla reprezentacji chromosomów i w rezultacie inne są też operatory „genetyczne". Mogą one, lub nie, zawierać inną informację (do sterowania procesem przeszukiwania) w genach. Są także inne różnice. Na przykład dwa wiersze na rys. 0.1 (we wprowadzeniu)

wybierz $P(t)$ z $P(t-1)$

zmień $P(t)$

mogą pojawić się w odwrotnej kolejności – w strategiach ewolucyjnych najpierw zmienia się populację, a potem tworzy się nową populację w procesie selekcji. Nawet w jednej metodzie, powiedzmy w ramach algorytmów genetycznych, jest wiele odmian. Na przykład jest wiele metod wyboru osobników do przeżycia i reprodukcji. Jak omówiono w rozdz. 4, obejmują one: (1) selekcję proporcjonalną, w której prawdopodobieństwo wyboru jest proporcjonalne do dopasowania osobnika, (2) metodę porządkową, w której wszystkie osobniki z populacji są ustawiane od najlepszego do najgorszego, a prawdopodobieństwa ich wyboru są stałe w całym procesie ewolucyjnym[1], (3) selekcję turniejową, w której pewna liczba osobników (zazwyczaj dwóch) rywalizuje o wybór do następnego pokolenia, ten krok rywalizacji (turniej) powtarza się tyle razy, ile wynosi liczebność populacji. Każda z tych kategorii zawiera dalsze istotne szczegóły. W selekcji proporcjonalnej można użyć metod okna skalującego lub zaokrąglania, istnieją różne sposoby przyporządkowywania prawdopodobieństw w metodzie porządkowej (rozkład liniowy lub nieliniowy), istotną rolę w metodach selekcji turniejowej gra rozmiar turnieju. Ważne są też decyzje dotyczące strategii pokoleniowej. Można na przykład zamienić całą populację na populację potomków lub można wybrać najlepszych osobników z dwóch populacji (populacji rodziców i populacji potomków). Wybór ten może nastąpić w sposób deterministyczny lub niedeterministyczny. Można

[1] Na przykład prawdopodobieństwo wyboru najlepszego osobnika wynosi zawsze 0,15 bez względu na jego dokładną ocenę, prawdopodobieństwo wyboru drugiego z najlepszych wynosi 0,14 itd. Jedynym wymaganiem jest, aby lepsze osobniki miały większe prawdopodobieństwa, a suma tych prawdopodobieństw była równa jeden.

także tworzyć kilku (w szczególności jednego) potomków, którzy zamieniają pewnych (najgorszych?) osobników (systemy oparte na na takiej strategii pokoleniowej nazywa się „ustalonymi"). Można także użyć modelu „elitarnego", w którym przenosi się najlepszych osobników z bieżącego pokolenia do pokolenia następnego[1]. Taki model jest bardzo użyteczny przy rozwiązywaniu różnego rodzaju zadań optymalizacyjnych. Ostatnio Ronald [333] eksperymentował z rozszerzeniem procesu selekcji przez zezwolenie dopasowanym osobnikom na dobór partnerów (tak zwane *podejście selekcji-uwodzenia*). W tym podejściu osobniki są nadal wybierane na podstawie ich dopasowania, jednak w czasie rozmnażania się osobnik może wybrać swojego partnera na podstawie własnych preferencji (są one sformułowane w postaci cech fenotypowych i mogą tworzyć część fenotypu).

Do wybranej reprezentacji chromosomu można dobrać różne operatory genetyczne. W tej książce rozważaliśmy różne typy mutacji. W niektórych z tych typów prawdopodobieństwo mutacji zależy od numeru pokolenia i/lub pozycji bitu. Jak przedstawiono w rozdz. 10 i 11, wiele operatorów „mutacji" zawiera jakiś heurystyczny algorytm lokalnego przeszukiwania w celu poprawiania działania algorytmu ewolucyjnego. Prócz krzyżowania jednopunktowego mamy także krzyżowania dwupunktowe, trójpunktowe itp., krzyżowania, w których wymienia się odpowiednią liczbę odcinków między chromosomami rodziców, a także „krzyżowanie jednorodne", w którym wymienia się pojedyncze geny od obu rodziców. Kiedy chromosom jest permutacją liczb całkowitych 1, ..., n, wtedy istnieje wiele sposobów mutacji takiego chromosomu oraz krzyżowania dwóch chromosomów (na przykład krzyżowania PMX, OX, CX, ER – z rekombinacją krawędzi, ERR – rozszerzone z rekombinacją krawędzi)[2]. Ostatnio Bui i Moon [56] formalnie uogólnili krzyżowania liniowych łańcuchów na kodowania binarne n-wymiarowe.

Różnorodność struktur, operatorów, metod selekcji itp. jasno wskazuje na to, że dla niektórych zadań pewne wersje algorytmów ewolucyjnych działają lepiej niż inne. W literaturze znajduje się wiele porównań różnego rodzaju (na przykład strategii ewolucyjnych z algorytmami genetycznymi, krzyżowania jednopunktowego z krzyżowaniem dwupunktowym i z krzyżowaniem jednorodnym itp.). W rezultacie, tworząc udany algorytm ewolucyjny dla konkretnego zadania (lub klasy zadań), użytkownik używa „wspólnej wiedzy": zbioru reguł heurystycznych, które pojawiły się w czasie ostatnich dwóch dekad jako podsumowanie niezliczonych eksperymentów z różnymi systemami i różnymi zadaniami. W następnym punkcie opiszemy krótko pewne reguły heurystyczne wyboru odpowiednich składników algorytmów ewolucyjnych, a w p. 15.3 opiszemy szczegółowo algorytmy heurystyczne używane do oceny osobnika w populacji.

---

[1] Oznacza to, że jeżeli najlepszy osobnik z bieżącego pokolenia ginie w trakcie selekcji lub przy zastosowaniu operatorów genetycznych, to system wymusza i tak jego obecność w następnym pokoleniu.

[2] W większości przypadków w krzyżowaniu bierze udział dwóch rodziców, jednak tak być nie musi, patrz rozdz. 4.

15.1. Metody i heurystyki: podsumowanie

Struktura danych dla szczególnego typu zadania i zbiór operatorów „genetycznych" są najbardziej podstawowymi składnikami algorytmu genetycznego. Na przykład oryginalny algorytm genetyczny przeznaczony do modelowania *procesów adaptacyjnych* działa głównie na łańcuchach binarnych z operatorami kombinacyjnymi oraz mutacją jako operatorem pomocniczym. Mutacja przestawia bit w chromosomie, a krzyżowanie wymienia materiał genetyczny między dwoma rodzicami. Jeżeli rodzice są reprezentowani przez łańcuchy pięciobitowe, powiedzmy (0, 0, 0, 0, 0) i (1, 1, 1, 1, 1), to krzyżowanie wektorów po drugiej składowej utworzy potomków (0, 0, 1, 1, 1) i (1, 1, 0, 0, 0).

Strategie ewolucyjne rozwinięto jako metodę rozwiązywania zadań optymalizacji parametrycznej. W rezultacie chromosom reprezentujący osobnika jest parą wektorów zmiennopozycyjnych, to znaczy $\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma})$. Wektor $\mathbf{x}$ reprezentuje punkt w przeszukiwanej przestrzeni, a drugi wektor $\boldsymbol{\sigma}$ jest wektorem odchyleń standardowych. Mutacja zamienia $\mathbf{v}$ na $(\mathbf{x}', \boldsymbol{\sigma}')$, przy czym

$$\boldsymbol{\sigma}' = \boldsymbol{\sigma} \cdot e^{N(0,\Delta\sigma)} \qquad \text{oraz}$$

$$\mathbf{x}' = \mathbf{x} + N(0, \boldsymbol{\sigma}')$$

gdzie $N(0, \boldsymbol{\sigma})$ jest wektorem niezależnych liczb losowych o rozkładzie normalnym z zerową średnią i odchyleniem standardowym $\boldsymbol{\sigma}$, a $\Delta\boldsymbol{\sigma}$ jest parametrem metody.

Początkowo metody programowania ewolucyjnego miały na celu rozwój sztucznej inteligencji, a do reprezentacji chromosomowej osobników przyjęto automaty skończone. Potomków (nowe automaty skończone) tworzy się przez losową mutację populacji rodziców. Jest pięć możliwych operatorów mutacji: zmiana symbolu wyjściowego, zmiana stanu przesyłania, dodanie stanu, usunięcie stanu oraz zmiana stanu początkowego (z pewnymi dodatkowymi ograniczeniami na minimalną i maksymalną liczbę stanów).

Metody programowania genetycznego umożliwiają przeszukiwanie przestrzeni możliwych programów komputerowych w celu znalezienia najlepszego (najbardziej dopasowanego).

Wielu badaczy modyfikowało następnie algorytmy ewolucyjne, „dodając" do nich wiedzę specyficzną dla zadania. W kilku pracach zajmowano się metodami rozpoczynania ewolucji, różnymi reprezentacjami, metodami dekodowania (odwzorowania reprezentacji genetycznych na reprezentacje „fenotypowe") oraz zastosowaniem heurystyk w operatorach genetycznych. Takie niestandardowe systemy hybrydowe cieszą się znacznym zainteresowaniem w środowisku związanym z obliczeniami ewolucyjnymi. Bardzo często te systemy, rozszerzone o wiedzę specyficzną dla zadania, działają lepiej niż klasyczne metody ewolucyjne czy metody standardowe (jak opisano w poprzednim rozdziale).

Jest kilka reguł heurystycznych pomagających użytkownikowi w wyborze odpowiednich struktur danych i operatorów dla konkretnego zadania. Jest powszechnie wiadomo, że w zadaniach optymalizacji numerycznej należy użyć strategii ewolucyjnej[1] lub algorytmu genetycznego z reprezentacją zmiennopozycyjną, gdy tymczasem pewne wersje algorytmu genetycznego są najlepsze dla zadań optymalizacji kombinatorycznej. Programy genetyczne są wspaniałe w odkrywaniu reguł zadanych programami komputerowymi, a metod programowania ewolucyjnego można z powodzeniem użyć do modelowania zachowania się systemu (na przykład w zadaniu dylematu więźnia, patrz [120]). Dodatkowa popularna reguła heurystyczna przy stosowaniu algorytmów ewolucyjnych w rzeczywistych zadaniach polega na modyfikacji algorytmu za pomocą wiedzy specyficznej dla zadania. Ta wiedza jest uwzględniana w strukturach danych chromosomu i specjalizowanych operatorach genetycznych. Na przykład w systemie GENETIC-2 (rozdz. 9) zbudowanym dla nieliniowego zadania transportowego użyto dla chromosomów reprezentacji macierzowej, mutacji specyficznej dla zadania (jako głównego operatora używanego z prawdopodobieństwem 0,4) oraz krzyżowania arytmetycznego (jako operatora pomocniczego używanego z prawdopodobieństwem 0,05). Trudno jest zaklasyfikować ten system. Nie jest on w pełni algorytmem genetycznym, gdyż może on działać tylko z operatorem mutacji, co nie prowadzi do znaczącego obniżenia jakości wyników. Ponadto wszystkie elementy macierzy są liczbami zmiennopozycyjnymi. Nie jest to strategia ewolucyjna, gdyż nie koduje się w strukturach chromosomów żadnych parametrów sterujących. I oczywiście nie ma on nic wspólnego z programowaniem genetycznym lub programowaniem ewolucyjnym. Jest to po prostu metoda ewolucyjna dla konkretnego zadania.

Inna możliwość polega na hybrydyzacji. W takim rozwiązaniu [78] uwzględnia się znane algorytmy w celu poprawienia wyników systemu ewolucyjnego. Można to zrobić, używając wyników z innych algorytmów do utworzenia populacji początkowej systemu ewolucyjnego, dołączając pewne operatory przeszukiwania lokalnego do operatorów „genetycznych" lub „zapożyczając" pewne strategie kodowania.

Niektóre z tych pomysłów były wcześniej uwzględnione w procedurach ewolucyjnych zwanych *przeszukiwaniem rozproszonym* (rozdz. 8). W procesie tym tworzy się populację początkową, przeglądając dobre rozwiązania uzyskane za pomocą heurystyk. Punkty przyjęte za rodziców są dalej przetwarzane jako kombinacje liniowe z wagami zależącymi od kontekstu, przy czym w takich kombinacjach może brać udział jednocześnie wielu rodziców. Operatory kombinacji liniowej są następnie modyfikowane za pomocą adaptacyjnego procesu zaokrąglania, aby uzyskać, tam gdzie jest to wymagane, składowe z wartościami dyskretnymi. (Wektory, na których dokonuje się operacji, mogą zawierać zarówno składowe rzeczywiste, jak i całkowite, w przeciwieństwie do

[1] Także metody programowania ewolucyjnego uogólniono tak, aby obejmowały zadania optymalizacji numerycznej, patrz [117].

składowych czysto binarnych). Na koniec preferowane rezultaty są ponownie poddawane heurystykom i proces powtarza się. To podejście okazało się użyteczne przy mieszanej optymalizacji całkowitoliczbowej i kombinatorycznej.

Jest tylko kilka reguł heurystycznych dla tworzenia populacji początkowej. Można rozpoczynać od losowo utworzonej populacji lub użyć do jej utworzenia jakiegoś algorytmu deterministycznego (z wieloma innymi możliwościami leżącymi pomiędzy tymi ekstremami). Są także pewne ogólne reguły heurystyczne dla określania wartości różnych parametrów. W wielu zastosowaniach algorytmów genetycznych liczebność populacji wynosi od 50 do 100, prawdopodobieństwo krzyżowania między 0,65 a 1,00, a prawdopodobieństwo mutacji między 0,001 a 0,01. Dodatkowe reguły heurystyczne są często używane do zmieniania liczebności populacji lub prawdopodobieństw w czasie procesu ewolucyjnego.

Wydaje się, że żadna z metod ewolucyjnych nie jest idealna (czy nawet niezawodna) w całym spectrum zadań. Tylko cała rodzina algorytmów opartych na pojęciu obliczeń ewolucyjnych (to znaczy algorytmy ewolucyjne) ma właściwość niezawodności. Ale główny klucz do udanych zastosowań znajduje się w metodach heurystycznych, umiejętnie połączonych z metodami ewolucyjnymi.

15.2. Rozwiązania dopuszczalne i niedopuszczalne

W metodach obliczeń ewolucyjnych funkcja oceny jest jedynym połączeniem zadania z algorytmem. Funkcja oceny szereguje osobników w populacji. Lepsze osobniki mają większe szanse przeżycia i reprodukcji. Dlatego podstawową rzeczą jest określenie funkcji oceny, która charakteryzuje zadanie w „doskonały sposób". W szczególności należy bardzo starannie rozpatrzyć postępowanie z osobnikami dopuszczalnymi i niedopuszczalnymi. Bardzo często populacja zawiera osobniki niedopuszczalne, gdy tymczasem szukamy osobników optymalnych *dopuszczalnych*. Określenie odpowiedniej miary oceny osobników dopuszczalnych i niedopuszczalnych ma dużą wagę – wpływa ono bezpośrednio na wynik (sukces lub porażkę) algorytmu.

Sprawa postępowania z osobnikami niedopuszczalnymi jest bardzo ważna przy rozwiązywaniu za pomocą metod ewolucyjnych zadań optymalizacji z ograniczeniami. Na przykład w dziedzinie ciągłej ogólne zadanie programowania nieliniowego[1] polega na wyznaczeniu takiego $\mathbf{x}$, które spełnia warunek

$$\text{optymalizuj } f(\mathbf{x}),\ \mathbf{x} = (x_1, \ldots, x_n) \in R^n$$

przy czym $\mathbf{x} \in \mathscr{F} \subseteq \mathscr{S}$. Zbiór $\mathscr{S} \subseteq R^n$ jest przestrzenią rozwiązań, a zbiór $\mathscr{F} \subseteq \mathscr{S}$ jest przestrzenią rozwiązań *dopuszczalnych*. Zazwyczaj przestrzeń roz-

[1] Rozważamy tu tylko zmienne ciągłe.

wiązań $\mathscr{S}$ jest n-wymiarowym prostopadłościanem w R^n (dziedzina zmienności określona przez ograniczenia dolne i górne)

$$l(i) \leqslant x_i \leqslant u(i), \qquad 1 \leqslant i \leqslant n$$

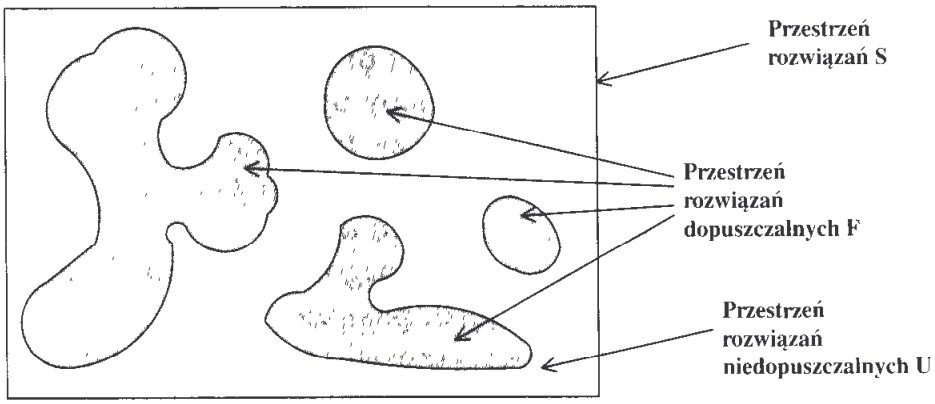
a zbiór rozwiązań dopuszczalnych $\mathscr{F}$ jest przecięciem $\mathscr{S}$ i zbioru wyznaczonego przez dodatkowe $m \geqslant 0$ ograniczeń

$$g_j(\mathrm{x}) \leqslant 0, \quad \text{dla } j = 1, \ldots, q \qquad \text{oraz} \quad h_j(\mathrm{x}) = 0, \quad \text{dla } j = q + 1, \ldots, m$$

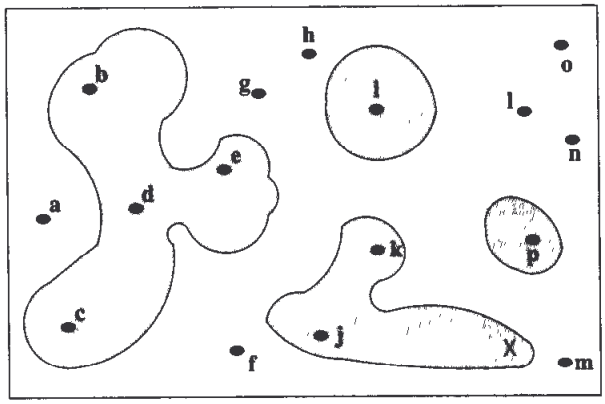
W większości badań nad zastosowaniem metod obliczeń ewolucyjnych w zadaniach programowania nieliniowego rozważano złożone funkcje celu przy $\mathscr{F} = \mathscr{S}$. Wiele funkcji testowych używanych w ciągu ostatnich 20 lat było określonych na całej dziedzinie n zmiennych. Tak było w przypadku pięciu funkcji F1 – F5 zaproponowanych przez De Jonga [82], jak również z wieloma innymi zadaniami testowymi zaproponowanymi później.

W dziedzinie dyskretnej problemy związane z ograniczeniami zauważono dużo wcześniej. Zadanie pakowania, zadanie pokrycia zbioru, wszystkie typy zadań szeregowania i układania planu lekcji zawierają ograniczenia. Powstało kilka metod heurystycznych do postępowania z ograniczeniami, jednak nie były one przebadane w sposób systematyczny.

W ogólności przestrzeń przeszukiwań $\mathscr{S}$ składa się z dwóch rozłącznych podzbiorów, odpowiednio z podprzestrzeni dopuszczalnej i niedopuszczalnej $\mathscr{F}$ i $\mathscr{U}$ (patrz rys. 15.1). Nie robimy żadnych założeń o tych podprzestrzeniach. W szczególności nie muszą one być wypukłe ani spójne (taki przypadek przedstawiono przykładowo na rys. 15.1, gdzie część dopuszczalna przestrzeni rozwiązań składa się z podzbiorów rozłącznych). Przy rozwiązywaniu zadań optymalizacji szukamy optimum *dopuszczalnego*. W procesie przeszukiwania mamy do czynienia z różnymi osobnikami dopuszczalnymi i niedopuszczalnymi. Na przykład (patrz rys. 15.2) na pewnym etapie procesu ewolucyjnego populacja może się składać z osobników dopuszczalnych (b, c, d, e, i, j, k, p) i niedopuszczalnych (a, f, g, h, l, m, n, o). Rozwiązanie optymalne (globalne) zaznaczono przez „X".



Rys. 15.1. Przestrzeń rozwiązań oraz jej części: dopuszczalna i niedopuszczalna



Rys. 15.2. Populacja 16 osobników: a - o

Obecność osobników dopuszczalnych i niedopuszczalnych w populacji wpływa na inne elementy algorytmu ewolucyjnego. Na przykład, czy w metodzie selekcji elitarnej należy wybierać najlepszego osobnika *dopuszczalnego*, czy po prostu osobnika najlepszego w ogóle? Zresztą niektóre operatory mogą być zastosowane tylko do osobników dopuszczalnych. Jednak główną cechą takiego scenariusza jest potrzeba oceny osobników dopuszczalnych i niedopuszczalnych. Zagadnienie, jak oceniać osobniki w populacji, nie jest całkiem łatwe. W ogólności musimy określić dwie funkcje oceny: $eval_f$ i $eval_u$, odpowiednio dla obszarów dopuszczalnego i niedopuszczalnego. Należy przy tym rozważyć wiele istotnych kwestii (omawiamy je szczegółowo w następnym punkcie):

A. Jak porównywać dwa osobniki dopuszczalne, na przykład „c" i „j" z rys. 15.2? Inaczej mówiąc, jak określić funkcję $eval_f$?
B. Jak porównywać dwa osobniki niedopuszczalne, na przykład „a" i „n" z rys. 15.2? Inaczej mówiąc, jak określić funkcję $eval_u$?
C. Jak powinny odnosić się do siebie funkcje $eval_f$ i $eval_u$? Czy powinniśmy na przykład przyjąć, że $eval_f(s) \succ eval_u(r)$ dla każdego $s \in \mathscr{F}$ i każdego $r \in \mathscr{U}$ (symbol $\succ$ oznacza tutaj „lepszy niż", to znaczy „większy niż" dla maksymalizacji i „mniejszy niż" dla minimalizacji)?
D. Czy powinniśmy uważać osobniki niedopuszczalne za szkodliwe i eliminować je z populacji?
E. Czy powinniśmy „naprawiać" rozwiązania niedopuszczalne, przesuwając je do najbliższego punktu w przestrzeni rozwiązań dopuszczalnych (na przykład naprawiona wersja „m" na rys. 15.2 mogłaby być w optimum „X")?
F. Jeżeli naprawiamy osobniki niedopuszczalne, czy powinniśmy zamieniać w populacji osobniki niedopuszczalne na ich wersje naprawione, czy też powinniśmy użyć procedury naprawy tylko w celu oceny?
G. Ponieważ naszym celem jest wyznaczenie optymalnego rozwiązania dopuszczalnego, czy powinniśmy karać osobniki niedopuszczalne?
H. Czy powinniśmy zaczynać od populacji początkowej złożonej z osobników dopuszczalnych i utrzymywać dopuszczalność potomków za pomocą specjalizowanych operatorów?

I. Czy powinniśmy zmieniać topologię przestrzeni rozwiązań, stosując dekodery?
J. Czy powinniśmy wybrać zbiór ograniczeń, które określają przestrzeń rozwiązań dopuszczalnych i przetwarzać oddzielnie osobniki i ograniczenia?
K. Czy powinniśmy skoncentrować się na przeszukiwaniu granicy między dopuszczalną a niedopuszczalną częścią przestrzeni rozwiązań?
L. Jak wyznaczyć rozwiązanie dopuszczalne?

W obliczeniach ewolucyjnych pojawiło się kilka kierunków postępowania z rozwiązaniami niedopuszczalnymi. Omówiliśmy kilka z nich w rozdz. 7 przy optymalizacji numerycznej. Teraz omówimy je, używając przykładów z dziedziny dyskretnej i ciągłej.

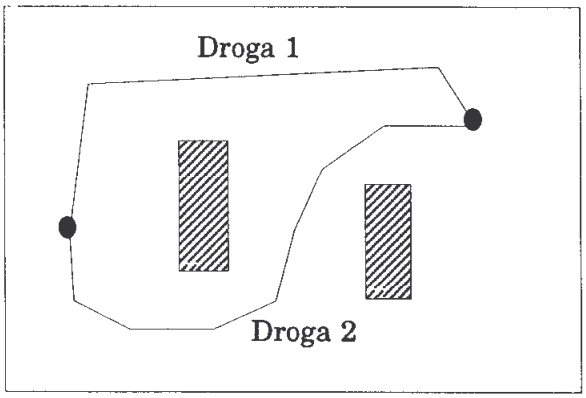
15.3. Heurystyki do oceny osobników

W tym punkcie omówimy kilka metod postępowania z rozwiązaniami dopuszczalnymi i niedopuszczalnymi w populacji. Większość z nich pojawiła się całkiem niedawno. Dopiero kilka lat temu Richardson i inni [332] twierdzili: „Próby zastosowania algorytmów genetycznych do zadań optymalizacji z ograniczeniami odbywają się według dwóch różnych wzorców: (1) modyfikacji operatorów genetycznych, (2) karania łańcuchów, które nie spełniają ograniczeń". Nie jest to już słuszne, gdyż zaproponowano do tego celu różnorodne heurystyki. Nawet kategoria funkcji kar obejmuje kilka metod, które różnią się wielu istotnymi szczegółami w zakresie określenia funkcji kary i jej zastosowania do rozwiązań niedopuszczalnych. W innych metodach utrzymuje się dopuszczalność osobników w populacji za pomocą specjalizowanych operatorów lub dekoderów, nakłada się ograniczenia, że każde rozwiązanie dopuszczalne jest „lepsze" niż jakiekolwiek rozwiązanie niedopuszczalne, rozważa się po jednym ograniczeniu na raz ze szczególnym liniowym uporządkowaniem, naprawia rozwiązania niedopuszczalne, używa się metod optymalizacji wielokryterialnej, przyjmuje się algorytmy kulturowe lub szereguje się rozwiązania stosując szczególne modele współewoluujące. Omówimy te metody po kolei, rozważając kwestie A–L z poprzedniego punktu.

A. Określenie $eval_f$

Jest to zazwyczaj najprostsza sprawa. W większości zadań optymalizacji funkcja oceny f dla rozwiązań dopuszczalnych jest zadana. Tak jest w przypadku zadań optymalizacji numerycznej i w większości zagadnień z badań operacyjnych (zadania załadunku, zadania komiwojażera, zadania pokrycia zbioru itp.). Jednak w niektórych zadaniach wybór funkcji oceny może nie być wcale łatwy. Na przykład przy tworzeniu systemu ewolucyjnego do sterowania ruchomym robotem (rozdz. 11) należy ocenić drogę robota. Nie jest jasne, czy

lepszą ocenę powinno się przydzielić drodze 1, czy 2 (rys. 15.3), jeżeli bierzemy pod uwagę ich całkowitą długość, dostateczną odległość od przeszkód i gładkość. Droga 1 jest krótsza, ale droga 2 jest gładsza. W takich zadaniach trzeba dołączyć do funkcji oceny jakąś miarę heurystyczną. Zauważmy, że nawet fragment funkcji oceny dotyczący mierzenia gładkości lub dostatecznej odległości od przeszkód dla drogi nie jest prosty.



Rys. 15.3. Droga w środowisku

Podobnie jest w wielu zadaniach projektowania, gdzie nie ma prostych wyrażeń do porównania dwóch dopuszczalnych projektów. Widać, że w takich przypadkach są potrzebne pewne heurystyki zależne od zadania, pozwalające określić numeryczną miarę $eval_f$ dopuszczalnego osobnika x.

Jednym z najlepszych przykładów zilustrowania kłopotu związanego z koniecznością oceny osobników dopuszczalnych jest zadanie spełniania logicznego. Dla zadanego wyrażenia w normalnej postaci koniunkcyjnej, powiedzmy

$$F(x) = (x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3) \wedge (x_2 \vee x_3)$$

jest trudno porównać dwa osobniki dopuszczalne $p = (0, 0, 0)$ i $q = (1, 0, 0)$ (w obu przypadkach $F(p) = F(q) = 0$). De Jong i Spears [90] przebadali kilka możliwości. Na przykład można określić $eval_u$ jako ułamek liczby czynników koniunkcji przyjmujących wartość prawda, w powyższym przypadku

$$eval_f(p) = 0{,}666 \qquad \text{i} \qquad eval_f(q) = 0{,}333$$

Można także [305] zamienić zmienne logiczne x_i na liczby zmiennopozycyjne y_i i przyporządkować

$$eval_f(y) = |y_1 - 1||y_2 + 1||y_3 - 1| + |y_1 + 1||y_3 + 1| + |y_2 - 1||y_3 - 1|$$

lub

$$eval'_f(y) = (y_1 - 1)^2(y_2 + 1)^2(y_3 - 1)^2 + (y_1 + 1)^2(y_3 + 1)^2 + \\ + (y_2 - 1)^2(y_3 - 1)^2$$

W powyższych przypadkach rozwiązanie zadania spełniania logicznego odpowiada zbiorowi globalnych punktów minimalnych funkcji celu: wartość *prawda* dla $F(x)$ jest równoważna wartości minimum globalnego 0 funkcji $eval_u(y)$.

Zacytujmy także z [109], gdzie autor odrzucił pomysł bezpośredniego utworzenia $eval_f$ dla zadania pakowania pojemników:

> Określmy odpowiednią funkcję kosztów dla zadania pakowania pojemników. Celem jest wyznaczenie minimalnej potrzebnej liczby pojemników, więc pierwsza funkcja kosztu, jaka przychodzi na myśl, to po prostu liczba pojemników potrzebnych do „zapakowania" wszystkich obiektów. Jest to prawidłowe pod względem matematycznym, ale jest nieużyteczne praktycznie. Rzeczywiście, taka funkcja kosztu prowadzi do wyjątkowo nieprzyjaznego pejzażu przestrzeni przeszukiwań: bardzo mała liczba punktów optymalnych w przestrzeni ukryta między wykładniczą liczbą punktów, w których ta rzekoma funkcja kosztu jest po prostu większa o jednostkę od optimum. Co gorsze, te trochę suboptymalne punkty mają ten sam koszt. Trudność polega na tym, że taka funkcja kosztu nie daje możliwości kierowania algorytmem w trakcie przeszukiwań, prowadząc do zadania „poszukiwania igły w stogu siana".

Dlatego ustaliliśmy następującą funkcję kosztu dla zadania pakowania pojemników [...]:

$$\max f_{BPP} = \frac{\sum_{i=1}^{N} (F_i/C)^k}{N}$$

gdzie N jest liczbą pojemników dla ocenianego rozwiązania, F_i jest sumą wymiarów obiektów wypełniających pojemnik i, C jest pojemnością pojemnika, a k jest stałą, $k > 1$.

Stała k wyraża nasze zainteresowanie pojemnikami „ekstremalnymi" w porównaniu z mniej załadowanymi. Im większe jest k, tym bardziej preferujemy dobrze upakowaną grupę „elitarną" w porównaniu ze zbiorem jednakowo upakowanych pojemników. Faktycznie wartość k daje nam możliwość zmiany „nieregularności" optymalizowanej funkcji, od „igły w stogu siana" ($k = 1$, $f_{BPP} = 1/N$) do „najlepiej upakowanego pojemnika" ($k \to \infty$, $f_{BPP} \to \max_i[(F_i/C)^k]$)".

Jak widać, zagadnienie wyboru „idealnej" funkcji $eval_f$ nie jest łatwe.

Jest też inna możliwość: w pewnych przypadkach nie potrzebujemy w ogóle określać funkcji $eval_f$! Jest ona tylko potrzebna wtedy, kiedy w algorytmie ewolucyjnym stosuje się selekcję proporcjonalną (patrz rozdz. 4). W innych typach selekcji można ustalić tylko relację liniowego uporządkowania ρ osobników w populacji. Jeżeli relacja liniowego uporządkowania pozwala odpowiedzieć na pytanie: „czy osobnik dopuszczalny x jest lepszy niż osobnik dopuszczalny y?"[1], to taka relacja ρ jest wystarczająca dla metod selekcji turniejowej

[1] Oznaczenie $\rho(x, y)$ należy interpretować jako x jest lepsze niż y dla dopuszczalnych x i y.

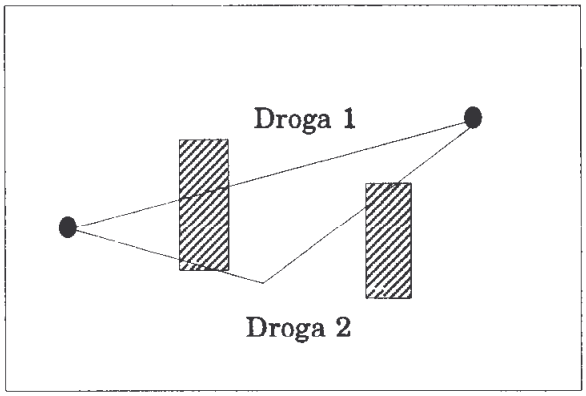
lub porządkowej, w których wymaga się, odpowiednio, albo wyboru najlepszego osobnika spośród pewnej liczby osobników, albo liniowego uporządkowania wszystkich osobników.

Oczywiście może być niezbędne użycie pewnych heurystyk do stworzenia takiej relacji liniowego uporządkowania ρ. Na przykład w zadaniach optymalizacji wielokryterialnej jest stosunkowo łatwo ustalić częściowe uporządkowanie między poszczególnymi rozwiązaniami. Dodatkowe heurystyki mogą być niezbędne do uporządkowania osobników, które nie są porównywalne za pomocą częściowego uporządkowania.

Podsumowując, wygląda na to, że selekcja turniejowa i porządkowa umożliwiają użytkownikowi większą elastyczność. Czasami jest łatwiej porównać dwa rozwiązania, niż podać ich wartości oceny w postaci liczb. Jednak w tych metodach trzeba rozwiązać inne problemy związane z porównywaniem dwóch osobników niedopuszczalnych (patrz część B) oraz osobników dopuszczalnych i niedopuszczalnych (patrz część C).

B. Określenie $eval_u$

Jest to dosyć trudne zadanie. Możemy go całkowicie uniknąć, odrzucając osobniki niedopuszczalne (patrz część D). Niekiedy jest możliwe rozszerzenie dziedziny funkcji $eval_f$, aby objąć osobniki niedopuszczalne, to znaczy przyjąć $eval_u(x) = eval_f(x) \pm Q(x)$, gdzie $Q(x)$ jest albo karą dla osobnika niedopuszczalnego x, albo kosztem naprawy takiego osobnika (patrz część G). Inna możliwość to określenie odrębnej funkcji oceny $eval_u$, niezależnej od $eval_f$. Jednak w takim przypadku trzeba ustalić relacje między tymi dwoma funkcjami (patrz część C).



Rys. 15.4. Droga niedopuszczalna w środowisku

Oceniać osobniki niedopuszczalne jest trudno. Tak jest w przypadku zadania plecakowego, gdzie wielkość naruszenia pojemności nie musi być dobrą miarą „dopasowania" osobnika (patrz część G). Jest tak także w przypadku wielu zadań harmonogramowania i ustalania planu lekcji oraz w zadaniu planowania drogi. Nie jest jasne, czy lepsza jest droga 1, czy 2 (rys. 15.4), gdyż droga 2 ma więcej punktów przecięć z przeszkodami i jest dłuższa niż droga 1.

Natomiast większość niedopuszczalnych dróg jest „gorszych" przy powyższych kryteriach niż linia prosta (droga 1).

Tak jak w przypadku rozwiązań dopuszczalnych (część A), można opracować relację porządkującą dla osobników niedopuszczalnych (w odróżnieniu od określenia $eval_u$). W obu przypadkach jest konieczne ustalenie relacji między ocenami osobników dopuszczalnych i niedopuszczalnych (część C).

C. Relacja między $eval_f$ i $eval_u$

Załóżmy, że przetwarzamy w populacji zarówno osobniki dopuszczalne, jak i niedopuszczalne i że oceniamy je, używając dwóch funkcji oceny: $eval_f$ i $eval_u$. Inaczej mówiąc, oceny osobników dopuszczalnego x i niedopuszczalnego y są odpowiednio równe $eval_f(x)$ i $eval_u(y)$. Bardzo ważne jest ustalenie relacji między tymi funkcjami.

Jedną z możliwości (o której wspomniano już w części B) jest określenie $eval_u$ za pomocą $eval_f$, to znaczy $eval_u(y) = eval_f(y) + Q(y)$, przy czym $Q(y)$ reprezentuje albo karę nałożoną na osobnika niedopuszczalnego y, albo koszt naprawy takiego osobnika (przedstawiamy tę możliwość w części G).

Jest też inny sposób. Można utworzyć globalną funkcję oceny $eval$:

$$eval(p) = \begin{cases} q_1 \cdot eval_f(p), & \text{jeżeli } p \in \mathscr{F} \\ q_2 \cdot eval_u(p), & \text{jeżeli } p \in \mathscr{U} \end{cases}$$

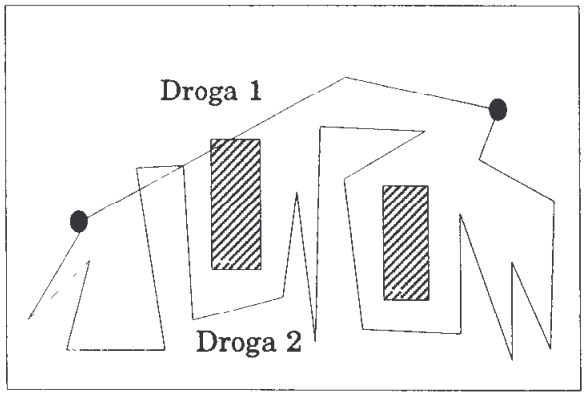
Inaczej mówiąc, do ustalenia wzajemnej istotności funkcji $eval_f$ i $eval_u$ użyto tu dwóch wag q_1 i q_2.

W obu powyższych metodach osobniki niedopuszczalne mogą być „lepsze" niż osobniki dopuszczalne. W ogólności może być tak, że dla osobnika dopuszczalnego x i osobnika niedopuszczalnego y zachodzi $eval(y) \succ eval(x)$[1]. Może to doprowadzić do tego, że algorytm zbiegnie się do rozwiązania niedopuszczalnego. Dlatego też kilka osób badało kary dynamiczne Q (patrz część G), które wzmacniają nacisk na osobniki niedopuszczalne przy aktualnym stanie przeszukiwania. Słabością tych metod jest ich zależność od zadania. Często wybór $Q(x)$ (lub wag q_1 i q_2) jest prawie tak trudny, jak rozwiązanie pierwotnego zadania.

Natomiast niektórzy autorzy [319], [279] przedstawili dobre wyniki z algorytmów genetycznych, w których założono, że każdy osobnik dopuszczalny jest lepszy niż jakikolwiek osobnik niedopuszczalny. Powell i Skolnick [312] zastosowali tę heurystyczną regułę w zadaniach optymalizacji numerycznej (patrz rozdz. 7). Oceny rozwiązań dopuszczalnych znajdowały się w przedziale $(-\infty, 1)$, a rozwiązań niedopuszczalnych w przedziale $(1, \infty)$ (przy minimalizacji). Michalewicz i Xiao [279] eksperymentowali z zadaniem planowania drogi (rozdz. 11) i używali dwóch różnych funkcji ocen dla osobników dopuszczal-

[1] Symbol $\succ$ oznacza „jest lepszy niż", to znaczy „większy niż" dla maksymalizacji i „mniejszy niż" dla minimalizacji.

nych i niedopuszczalnych. Wartości $eval_u$ zwiększano (czyniąc je mniej atrakcyjnymi), dodając stałą tak, że najlepsze osobniki niedopuszczalne były gorsze niż najgorsze osobniki dopuszczalne. Jednak nie jest jasne, czy tak powinno być zawsze. Na przykład jest wątpliwe, czy osobnik dopuszczalny „b" (rys. 15.2) powinien być oceniony lepiej niż osobnik niedopuszczalny „m", który jest „tuż" przy rozwiązaniu dopuszczalnym. Podobny przykład można podać dla planowania drogi. Nie jest jasne, czy droga dopuszczalna 2 (patrz rys. 15.5) zasługuje na lepszą ocenę niż niedopuszczalna droga 1.



Rys. 15.5. Drogi dopuszczalna i niedopuszczalna w środowisku robota

Sprawa ustalenia relacji między funkcjami oceny dla osobników dopuszczalnych i niedopuszczalnych jest jedną z trudniejszych do rozwiązania przy stosowaniu algorytmów ewolucyjnych w konkretnym zadaniu.

D. Pomijanie osobników niedopuszczalnych

Ta heurystyka „kary śmierci" jest popularnym sposobem postępowania w wielu metodach ewolucyjnych (na przykład w strategiach ewolucyjnych). Zauważmy, że pominięcie osobników niedopuszczalnych prowadzi do kilku uproszczeń algorytmu. Na przykład nie trzeba wtedy określać $eval_u$ i porównywać jej z $eval_f$.

Metoda eliminacji rozwiązań niedopuszczalnych z populacji może być dobra, gdy przestrzeń rozwiązań dopuszczalnych jest wypukła i stanowi dostatecznie dużą część całej przestrzeni przeszukiwań (na przykład w strategiach ewolucyjnych nie można przyjmować ograniczeń równościowych, gdyż przy takich ograniczeniach stosunek objętości przestrzeni rozwiązań dopuszczalnych do przestrzeni rozwiązań niedopuszczalnych wynosi zero). Zresztą takie podejście natrafia na poważne ograniczenia. Na przykład w wielu zadaniach przeszukiwania, w których populacja początkowa składa się tylko z osobników niedopuszczalnych, należy je poprawić, a nie pominąć. Ponadto całkiem często system może łatwiej osiągnąć rozwiązanie optymalne, jeżeli można „przekroczyć" obszar niedopuszczalny (szczególnie przy niewypukłych przestrzeniach rozwiązań dopuszczalnych).

E. Naprawa osobników niedopuszczalnych

Algorytmy naprawy cieszą się szczególną popularnością w środowisku związanym z obliczeniami ewolucyjnymi. W wielu zadaniach optymalizacji kombinatorycznej (na przykład w zadaniu komiwojażera, zadaniu plecakowym, zadaniu pokrycia zbioru itp.) jest stosunkowo łatwo „naprawić" osobnika niedopuszczalnego. Takiej naprawionej wersji można użyć albo tylko do oceny, na przykład przyjmując

$$eval_u(y) = eval_f(x)$$

gdzie x jest naprawioną (to znaczy dopuszczalną) wersją y, albo może ona zamieniać (z pewnym prawdopodobieństwem) oryginalnego osobnika w populacji (patrz część F). Zauważmy, że naprawiona wersja rozwiązania „m" (rys. 15.2) może trafić w optimum „X".

Proces naprawy osobników niedopuszczalnych wiąże się z połączeniem uczenia i ewolucji (tak zwany *efekt Baldwina* [399]). Uczenie (jak na przykład lokalne przeszukiwanie w ogólności, a w szczególności lokalne przeszukiwanie w poszukiwaniu lokalnego rozwiązania dopuszczalnego) i ewolucja zazębiają się ze sobą. Osobnikowi przypisuje się wartość dopasowania po poprawie. W ten sposób przeszukiwanie lokalne jest podobne do uczenia się konkretnego łańcucha w czasie jednego pokolenia.

Słabością tych metod jest ich zależność od zadania. Dla każdego konkretnego zadania należy opracować specyficzny dla niego algorytm naprawy. Ponadto nie ma ustalonych heurystyk do projektowania takich algorytmów. Zazwyczaj można użyć naprawy zachłannej, naprawy losowej lub jakiejkolwiek innej heurystyki, która przeprowadzi proces naprawy. W pewnych zadaniach proces naprawy osobników niedopuszczalnych może być tak samo trudny, jak rozwiązanie początkowego zadania. Tak jest w nieliniowym zadaniu transportowym, większości zadań harmonogramowania i układania planów lekcji, i wielu innych. Natomiast system GENPCOP III (rozdz. 7) do optymalizacji numerycznej z ograniczeniami (dla nieliniowych ograniczeń) pracuje z algorytmami naprawy.

F. Zamiana osobników na ich naprawione wersje

Zagadnienie zamiany naprawionych osobników jest związane z tak zwaną *ewolucją Lamarckiana* [399], gdzie zakłada się, że osobnik poprawia się w czasie swego życia i że poprawa ta jest kodowana w chromosomie. Jak podano w [399]:

> Nasze wyniki analityczne i empiryczne wskazują na to, że strategie Lamarckiana są często wyjątkowo szybką formą przeszukiwania. Jednak istnieją funkcje, dla których zarówno proste algorytmy genetyczne bez uczenia, jak i bez strategii Lamarckiana [...] zbiegają się do optimum lokalnego, gdy tymczasem prosty algorytm genetyczny z efektem Baldwina zbiega się do optimum globalnego.

Dlatego też należy bardzo ostrożnie używać strategii zamiany.

Jak opisano w p. 4.5.2, Orvosh i Davis opracowali tak zwaną regułę 5%, która mówi, że w wielu zadaniach optymalizacji kombinatorycznej metoda obliczeń ewolucyjnych z algorytmem naprawy daje najlepsze wyniki wtedy, kiedy 5% naprawionych osobników zamienia swoje niedopuszczalne oryginały. W dziedzinach ciągłych powstaje nowa reguła zamiany. System GENOCOP III (rozdz. 7) do optymalizacji numerycznej z ograniczeniami nieliniowymi jest oparty na naprawach. Pierwsze doświadczenia (z 10 testów z różną liczbą zmiennych, ograniczeń, typów ograniczeń, liczbą ograniczeń aktywnych w optimum itp.) wskazują na to, że reguła zamiany 15% jest wyraźnie najlepsza. Wyniki systemu są znacznie lepsze niż przy mniejszych lub większych stosunkach zamiany.

Obecnie wydaje się, że „optymalne" prawdopodobieństwo zamiany zależy od zadania i może zmieniać się także w trakcie procesu ewolucyjnego. Potrzebne są dalsze badania porównujące różne reguły heurystyczne ustalania tego parametru, mającego ogromne znaczenie dla wszystkich metod opartych na naprawie.

G. Karanie osobników niedopuszczalnych

Jest to najczęściej stosowany sposób w środowisku związanym z algorytmami genetycznymi. Rozszerza się w nim dziedzinę funkcji $eval_f$ i przyjmuje się, że

$$eval_u(p) = eval_f(p) \pm Q(p)$$

gdzie $Q(p)$ reprezentuje albo karę dla osobnika niedopuszczalnego p, albo koszt naprawy takiego osobnika. Podstawowym pytaniem tutaj jest, jak określić taką funkcję kary $Q(p)$? Intuicyjnie jest to proste: kara powinna być tak mała, jak to jest możliwe, tuż ponad granicą, poniżej której takie rozwiązania niedopuszczalne są optymalne (tak zwana *reguła najmniejszej kary* [239]). Jednak trudno jest efektywnie zastosować tę regułę.

Relacja między osobnikiem niedopuszczalnym „p" i obszarem dopuszczalnym $\mathscr{F}$ w przestrzeni rozwiązań $\mathscr{S}$ gra istotną rolę w karaniu osobników niedopuszczalnych. Osobnik może być karany za to, że jest niedopuszczalny, przy czym w celu określenia wartości kary mierzy się „wielkość" niedopuszczalności, lub można wziąć pod uwagę wysiłek włożony w „naprawę" tego osobnika. Na przykład w zadaniu plecakowym z pojemnością 99 możemy mieć dwa rozwiązania niedopuszczalne dające ten sam zysk, przy czym całkowita waga wszystkich zapakowanych obiektów wynosi 100 lub 105. Jednak trudno jest argumentować, że pierwszy osobnik z całkowitą wagą 100 jest „lepszy" niż drugi z całkowitą wagą 105, mimo że dla pierwszego osobnika naruszenie ograniczenia na pojemność jest znacznie mniejsze niż dla drugiego. Powodem jest to, że w pierwszym rozwiązaniu może wystąpić 5 obiektów o wadze 20, w drugim (między innymi) może znajdować się obiekt o wadze 6. Usunięcie tego obiektu utworzyłoby rozwiązanie dopuszczalne, być może znacznie lepsze niż jakakolwiek naprawiona wersja pierwszego osobnika. Dlatego w takich przypadkach funkcja kary powinna uwzględniać „łatwość naprawy" osobnika oraz jakość naprawionej wersji. Określenie takiej funkcji kary jest zależne od zadania i w ogólności dosyć trudne.

W rozdziale 7 omówiono wiele metod z funkcjami kary. Były to: metody statyczne [195], metody dynamiczne [210], [267] i metody adaptacyjne [360], [27]. W rozdziale tym opisano także rozdzielny algorytm genetyczny [239], w którym w dwóch populacjach ewoluujących „równolegle" zastosowano niskie i wysokie kary.

Wygląda na to, że odpowiedni wybór metody kary zależy od: (1) stosunku objętości obszaru dopuszczalnego do objętości całej przestrzeni rozwiązań, (2) właściwości topologicznych obszaru dopuszczalnego, (3) typu funkcji celu, (4) liczby zmiennych, (5) liczby ograniczeń, (6) typu ograniczeń, (7) liczby ograniczeń aktywnych w optimum. Wobec tego użycie funkcji kary nie jest łatwe i możliwa jest tylko częściowa analiza właściwości tych metod. Obiecującym kierunkiem zastosowania funkcji kary jest użycie kar samoadaptujących się. Współczynniki kar można włączyć do struktur chromosomów w podobny sposób, jak parametry sterujące włączono do struktur w strategiach ewolucyjnych i programowaniu ewolucyjnym.

H. Utrzymywanie populacji dopuszczalnej za pomocą specjalnych reprezentacji i operatorów genetycznych

Prawdopodobnie jedną z najrozsądniejszych heurystyk w celu zapewnienia dopuszczalności jest użycie specjalizowanych reprezentacji i operatorów utrzymujących dopuszczalność osobników w populacji. Takie było początkowe zamierzenie w tej książce, co odbiło się na jej tytule.

W czasie ostatniego dziesięciolecia opracowano kilka specjalizowanych systemów dla konkretnych zadań optymalizacji. Użyto w nich szczególnych reprezentacji chromosomów i specjalizowanych operatorów „genetycznych" do zmiany ich zawartości. Niektóre z tych systemów są opisane w [78], wiele innych przykładów podano w książce. Na przykład w systemie GENOCOP (rozdz. 7) przyjmuje się tylko liniowe ograniczenia oraz dopuszczalny punkt początkowy (czyli dopuszczalną populację początkową). Zamknięty zbiór operatorów utrzymuje dopuszczalność rozwiązań. Gdy na przykład dochodzi do mutacji jakiegoś składnika x_i wektora rozwiązań $\mathbf{x}$, wówczas system określa bieżącą dziedzinę $dom(x_i)$ (która zależy od liniowych ograniczeń i pozostałych wartości składowych wektora rozwiązań $\mathbf{x}$) i nową wartość x_i bierze się z tej dziedziny (albo z płaskim rozkładem prawdopodobieństwa dla mutacji jednorodnej, albo z innym rozkładem prawdopodobieństwa dla mutacji niejednorodnej lub brzegowej). W każdym przypadku wektor rozwiązania dla potomka jest zawsze dopuszczalny. Podobnie w krzyżowaniu arytmetycznym

$$a\mathbf{x} + (1 - a)\bar{Y}$$

dla dwóch wektorów rozwiązań dopuszczalnych $\mathbf{x}$ i $\mathbf{y}$ uzyskuje się zawsze rozwiązanie dopuszczalne (dla $0 \leqslant a \leqslant 1$) w wypukłej przestrzeni rozwiązań (w systemie przyjmuje się tylko liniowe ograniczenia, co powoduje, że przestrzeń rozwiązań dopuszczalnych $\mathscr{F}$ jest wypukła). W rezultacie nie trzeba określać funkcji $eval_u$. Natomiast funkcja $eval_f$ jest (jak zwykle) funkcją celu f.

Bardzo często takie systemy są bardziej niezawodne niż inne metody ewolucyjne z karami (patrz na przykład rozdz. 14). W rezultacie jest to dosyć popularny kierunek. W wielu dziedzinach praktycy do opracowania bardzo skutecznych algorytmów ewolucyjnych stosują reprezentacje specyficzne dla zadania i specjalizowane operatory. Dotyczy to optymalizacji numerycznej, uczenia maszynowego, sterowania optymalnego, modelowania poznawczego, klasycznych badań operacyjnych (zadania komiwojażera, zadań załadunku, zadań transportowych, zadań przydziału, pakowania pojemników, harmonogramowania, podziału itp.), projektowania inżynierskiego, integracji systemowej, gier iteracyjnych, robotyki, przetwarzania sygnałów i wielu innych.

Warto także zauważyć, że wczesne metody programowania ewolucyjnego [126] oraz programowania genetycznego [231] podchodzą pod kategorię algorytmów ewolucyjnych. Zachowuje się w nich dopuszczalność automatów skończonych lub programów o strukturze hierarchicznej za pomocą specjalizowanej reprezentacji i operatorów.

I. Użycie dekoderów

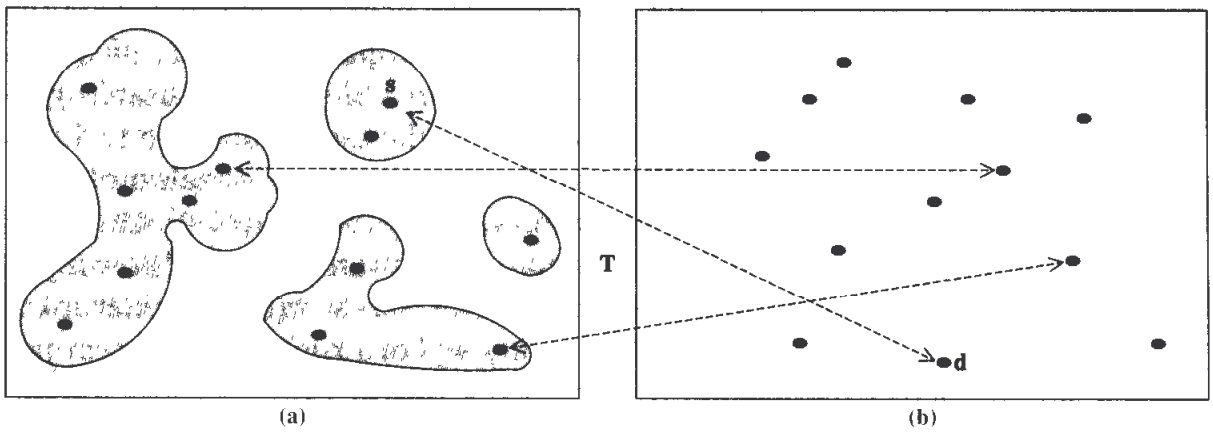
Dekodery są interesującą możliwością dla wszystkich stosujących praktycznie metody ewolucyjne. W tych metodach chromosom „zawiera instrukcję", jak utworzyć rozwiązanie dopuszczalne. Na przykład ciąg obiektów dla zadania pakowania można interpretować jako: „weź obiekt, jeżeli jest to możliwe". Taka interpretacja zawsze doprowadzi do rozwiązania dopuszczalnego. Rozważmy następujący scenariusz: próbujemy rozwiązać zero-jedynkowe zadanie załadunku z n obiektami. Wartość i waga i-tego obiektu są odpowiednio równe p_i oraz w_i. Możemy poukładać obiekty według malejących wartości p_i/w_i i zinterpretować łańcuch binarny

(1100110001001110101001010111010101...0010)

w następujący sposób: weź pierwszy obiekt z listy (to znaczy obiekt o największym stosunku wartości do wagi), jeżeli tylko zmieści się do plecaka. Powtórz to dla drugiego, piątego, szóstego, dziesiątego itd. obiektu z listy, tak długo, aż plecak będzie pełny, albo nie będzie już więcej obiektów do rozważenia. Zauważmy, że ciąg samych jedynek odpowiada rozwiązaniu zachłannemu. Jakikolwiek ciąg bitów odpowiada rozwiązaniu dopuszczalnemu, każde rozwiązanie dopuszczalne może być zakodowane na wiele sposobów. Możemy zastosować klasyczne operatory binarne (krzyżowania i mutacji). Każdy potomek jest oczywiście dopuszczalny.

Należy jednak zauważyć, że przy zastosowaniu dekoderów trzeba wziąć pod uwagę kilka czynników. Każdy dekoder narzuca zależność T między rozwiązaniem dopuszczalnym i zakodowanym (patrz rys. 15.6). Należy przy tym spełnić kilka warunków: (1) dla każdego rozwiązania $s \in \mathscr{F}$ istnieje rozwiązanie zakodowane d, (2) każde rozwiązanie zakodowane d odpowiada pewnemu rozwiązaniu dopuszczalnemu s, (3) wszystkie rozwiązania w $\mathscr{F}$ są reprezentowane przez tę samą liczbę rozwiązań zakodowanych d. Dodatkowo można wyma-

gać, aby przekształcenie T: (4) było szybkie do liczenia oraz (5) miało cechy lokalności, w tym sensie, że małe zmiany w rozwiązaniach zakodowanych powodują małe zmiany samego rozwiązania. Interesujące badania nad kodowaniem drzew w algorytmach genetycznych przedstawili Palmer i Kershenbaum [304], formułując jednocześnie powyższe warunki.



Rys. 15.6. Przekształcenie T rozwiązań (a) oryginalnych i (b) zakodowanych

J. Rozdzielanie osobników i ograniczeń

Jest to ogólna i ciekawa heurystyka. Pierwsza możliwość to użycie metod optymalizacji wielokryterialnej, gdzie funkcja celu f oraz miary przekroczenia ograniczeń f_j (dla m ograniczeń) tworzą wektor $\mathbf{v}$ o wymiarze $(m + 1)$:

$$\mathbf{v} = (f, f_1, ..., f_m)$$

Stosując pewną metodę optymalizacji wielokryterialnej, możemy spróbować zminimalizować jego składowe. Dla idealnego rozwiązania x powinno zachodzić $f_j(x) = 0$ dla $1 \leqslant i \leqslant m$ oraz $f(x) \leqslant f(y)$ dla wszystkich dopuszczalnych y (dla zadania minimalizacji). Jak podano w rozdz. 7, udane zaprogramowanie tego podejścia opisano ostatnio w [376].

Inna heurystyka opiera się na pomyśle rozpatrywania ograniczeń w szczególnym porządku. Schoenauer i Xanthakis [346] nazwali tę metodę podejściem z „*pamięcią behawioralną*". Początkowe kroki w tej metodzie są poświęcone badaniu obszaru dopuszczalnego. Funkcja celu f jest optymalizowana dopiero w końcowym kroku (patrz rozdz. 7).

W rozdziale 7 omówiono także krótko możliwość włączenia wiedzy o rozwiązaniach do przestrzeni wiary algorytmów kulturowych [325]. Takie algorytmy pozwalają na efektywne przeszukiwanie przestrzeni rozwiązań dopuszczalnych [329].

K. Badanie granicy między dopuszczalną i niedopuszczalną częścią przestrzeni rozwiązań

Jednym z ostatnio opracowanych podejść dla optymalizacji z ograniczeniami jest *oscylacja strategiczna*. Oscylację strategiczną zaproponowano początkowo

dla strategii ewolucyjnej oraz przeszukiwania rozproszonego, a ostatnio stosowano ją w różnych sformułowanych zadaniach optymalizacji kombinatorycznej i nieliniowej (patrz na przykład przegląd Glovera [147]). To podejście opiera się na określaniu *poziomów krytycznych*, które dla naszego celu będą reprezentowały granicę między dopuszczalnością a niedopuszczalnością, ale które mogą także obejmować takie elementy, jak etap budowy lub wybrane wartości przedziału dla funkcjonału. W kontekście dopuszczalności i niedopuszczalności podstawowa strategia polega na zbliżaniu się i przekraczaniu granicy dopuszczalności na podstawie algorytmu z adaptacyjnymi karami i wymuszeniami (które są stopniowo rozluźniane lub zawężane w zależności od tego, czy bieżący kierunek poszukiwania ma objąć głębszą część obszaru, czy też tylko bliższą granicy) lub po prostu na zastosowaniu zmodyfikowanych gradientów albo podgradientów do przesunięcia się w żądanym kierunku. W kontekście przeszukiwania lokalnego reguły wyboru ruchów zazwyczaj poprawia się, aby lepiej uwzględnić rozpatrywany obszar oraz kierunek ruchu. W procesie powtarzanego zbliżania się i przekraczania granicy dopuszczalności z różnych kierunków unika się wejścia na poprzednią trajektorię za pomocą mechanizmów z pamięcią oraz prawdopodobieństwem.

Zastosowanie różnych reguł (zależnych od obszaru i kierunku) łączy się na ogół z przekraczaniem granicy po obu stronach na różną głębokość. Inna możliwość, to zbliżanie się do granicy i powrót po jednej stronie, bez jej przekraczania. Oscylacje jednostronne są szczególnie użyteczne w różnych sformułowaniach zadań z zakresu harmonogramowania i teorii grafów, gdzie użyteczną strukturę można zachować tylko do pewnego punktu, po którym się ją traci (jak pozostanie bez prac do przydzielenia lub wyjście poza warunki określające drzewo czy drogę itp.). W tych przypadkach z konstruktywnym procesem tworzenia poziomu krytycznego jest związany destrukcyjny proces niszczenia struktury.

W oscylacjach strategicznych jest często istotne poświęcenie dodatkowego czasu na przeszukanie obszarów w pobliżu ograniczeń. Można to zrobić, zapoczątkowując ciąg ciasnych oscylacji wokół ograniczenia jako wstęp do każdej większej oscylacji, z głębszymi przekroczeniami. Jeżeli można włożyć więcej wysiłku w wykonanie ruchu, to metoda może uwzględniać bardziej pracochłonne ruchy (takie jak różnego rodzaju „wymiany"), aby pozostać przez dłuższy czas na ograniczeniu. Na przykład można użyć takich ruchów do przejścia do lokalnego optimum za każdym razem, gdy doszło do krytycznego zbliżenia się do ograniczeń. Strategia stosowania ruchów na dodatkowym poziomie wynika z *zasady przybliżonej optymalności*, która z grubsza mówi, że dobre konstrukcje na jednym poziomie będą najprawdopodobniej bliskie dobrym konstrukcjom na innym poziomie.

Jedna z użytecznych rodzajów oscylacji strategicznych polega na zwężaniu i rozszerzaniu ograniczeń dla funkcji $g(x)$. Takie podejście sprawdziło się w wielu zastosowaniach, gdzie $g(x)$ reprezentowało takie czynniki, jak przyporządkowanie siły roboczej i wartości funkcji (a także poziomów dopasowania

i niedopasowania), aby ukierunkować poszukiwania na badanie odpowiedniego obszaru na różnej głębokości. W przypadku stopni dopuszczalności i niedopuszczalności $g(x)$ może reprezentować funkcję wektorową związaną ze zbiorem ograniczeń występujących w zadaniu (które mogą być na przykład wyrażone jako $g(x) \leqslant b$). W takim wypadku sterowanie przeszukiwaniem za pomocą ograniczania $g(x)$ można interpretować jako zmiany parametrów w wybranych ograniczeniach. Często stosowanym innym wyjściem jest utworzenie funkcji Lagrange'a zawierającej $g(x)$ lub funkcji kary z zastępczymi ograniczeniami, unika się w ten sposób funkcji wektorowej i pozwala na wyważenie istotności stopni przekraczania różnych ograniczeń. Podejścia z zastępczymi ograniczeniami są szczególnie użyteczne przy wykrywaniu takiego zbioru ograniczeń. Jest to połączone ze specjalną pamięcią do przechowywania poprzednich sytuacji, co umożliwia ocenę wzajemnego wpływu ograniczeń. Podejścia uwzględniające takie pomysły można znaleźć na przykład w [133], [136], [221], [393], [410], [148].

L. Wyznaczanie rozwiązań dopuszczalnych

Są zadania, w których ważne jest wyznaczenie jakiegokolwiek rozwiązania dopuszczalnego. W takich zadaniach nie interesuje nas właściwie zagadnienie optymalizacji (wyznaczenie *najlepszego* dopuszczalnego rozwiązania), ale znalezienie *jakiegokolwiek* punktu w podprzestrzeni rozwiązań dopuszczalnych $\mathscr{F}$. Takie zadania nazywa się *zadaniami spełniania ograniczeń*. Klasycznym przykładem zadania tego typu jest znane zadanie N hetmanów, polegające na ustawieniu N hetmanów na szachownicy o N rzędach i N kolumnach w taki sposób, aby żadne dwa z nich nie biły się wzajemnie.

Kilka osób eksperymentowało z zastosowaniem systemów ewolucyjnych w zadaniach spełniania ograniczeń. Bowen i Dozier [48] opracowali hybrydowy algorytm genetyczny ze skomplikowaną strukturą chromosomów. Każdy chromosom, oprócz pamiętania wartości v_i wszystkich zmiennych, przechowuje także: (1) wskaźniki, które wskazują mutacji heurystycznej, które geny mają być zmutowane, (2) „numer rodziny" osobnika, (3) wektor, zawierający po jednej liczbie dla każdego genu, reprezentującej liczbę naruszonych ograniczeń, gdy wartość zmiennej wynosi v_i, (4) pewną heurystyczną wartość, która służy do określenia, który z genów ma być wskazany do mutacji. System był z powodzeniem przetestowany na wielu losowo wygenerowanych zadaniach spełniania ograniczeń [48] i porównany z innymi metodami programowania z ograniczeniami.

Eiben i inni [99] próbowali określić dla tego zadania specjalizowane operatory. Autorzy badali kilka operatorów heurystycznych: mutację i krzyżowania (łącznie z krzyżowaniami z wielu rodzicami). W operatorach mutacji wybierano pozycje w chromosomie rodzica oraz nowe wartości tych pozycji, przy czym liczba modyfikowanych wartości, kryteria wyboru pozycji i wartości do modyfikacji oraz kryteria określenia nowych wartości były parametrami operatora. Krzyżowania z wielu rodzicami oparto na przeglądaniu, w którym bada się po

kolei pozycje rodziców i wybiera się jedną z wartości do umieszczenia na zaznaczonej pozycji dziecka. System z takimi operatorami heurystycznymi był zaprogramowany i przetestowany na zadaniu N hetmanów i zadaniu kolorowania grafu.

Paredis badał dwa różne podejścia do zadania spełniania ograniczeń. Pierwsze z nich [306], [307] oparł na sprytnej reprezentacji osobników. Każdy gen może przyjmować, zamiast wartości zmiennej, wartość „?", która reprezentuje otwartą możliwość. Populacja początkowa składa się z łańcuchów z samymi „?". W cyklu selekcja-przyporządkowanie-propagacja zamienia się pewne „?" na wartości z odpowiedniej dziedziny (robi się to, wybierając zmienną, której dziedzina zawiera więcej niż jeden element), wybiera wartość z tej dziedziny i przyporządkowuje się ją tej zmiennej, a na koniec wykonuje krok propagacji, który zapewnia, że dokonane dotąd przyporządkowania zgadzają się z dziedzinami. Dopasowanie takich częściowo określonych osobników definiuje się przez wartość funkcji celu najlepszego znalezionego pełnego rozwiązania (kiedy zaczynamy poszukiwanie od danego osobnika częściowego). Operatory rozszerzono, aby uwzględnić proces naprawy (krok sprawdzania ograniczeń). System zaprogramowano i przeprowadzono obliczenia dla kilku zadań N hetmanów [307] i pewnych zadań szeregowania [306].

W drugim podejściu [308] Paredis badał model współewoluujący, w którym populacja potencjalnych rozwiązań ewoluuje wspólnie z populacją ograniczeń. Bardziej dopasowane rozwiązania spełniają więcej ograniczeń, a bardziej dopasowane ograniczenia są przekraczane przez więcej rozwiązań. Oznacza to, że osobniki z populacji rozwiązań są w całej przestrzeni rozwiązań i że nie rozróżnia się osobników dopuszczalnych i niedopuszczalnych. Ocena osobnika jest określana na podstawie miar naruszania ograniczeń f_i. Jednak lepsze f_i (na przykład ograniczenia aktywne) wpływają bardziej na ocenę ograniczeń. To podejście przetestowano na zadaniu N hetmanów i porównano z innymi podejściami z pojedynczymi populacjami [309], [308].

16
Zakończenie

Nie słyszeliście jeszcze niczego, chłopy.

Al Jonson, *The Jazz Singer*

Dziedzina obliczeń ewolucyjnych rozwinęła się raptownie w ciągu ostatnich kilku lat. Jednak nadal jest w niej wiele białych plam do wypełnienia, wiele badań do wykonania, wiele pytań czeka na odpowiedzi. W końcowym rozdziale tej książki przejrzymy kilka ważnych kierunków rozwoju, w których spodziewamy się dużej aktywności i istotnych wyników. Omówimy je po kolei.

Podstawy teoretyczne

Jak napisano wcześniej w tej książce, niektóre programy ewolucyjne mają pewne podstawy teoretyczne. Można wykazać, że strategie ewolucyjne stosowane w zadaniach regularnych (rozdz. 8) są zbieżne. Natomiast algorytmy genetyczne mają twierdzenie o schematach (rozdz. 3), które wyjaśnia, dlaczego one działają. Jednak wiele metod modyfikuje się, stosując je w rzeczywistych zadaniach. Na przykład, aby zaadaptować algorytmy genetyczne do optymalizacji funkcji, niezbędne było rozszerzenie ich o dodatkowe możliwości (na przykład dynamicznie skalowane dopasowania, selekcję na podstawie uszeregowania, dołączenie strategii elitarnej, adaptację różnych parametrów przeszukiwania, różne reprezentacje, nowe operatory itp.). Strategie ewolucyjne stosowane w zadaniach optymalizacji numerycznej z ograniczeniami zazwyczaj obejmują pewne metody heurystyczne do uwzględniania ograniczeń. Większość z tych modyfikacji powoduje, że nie spełniają one założeń wymaganych w twierdzeniach. Natomiast zazwyczaj powodują one istotne poprawienie działania systemów. W kontekście algorytmów genetycznych modyfikacje te [89]:

> ... wyprowadziły zastosowania prostych algorytmów genetycznych daleko poza nasze początkowe teorie i oczekiwania, wprowadzając potrzebę powrotu do nich i rozszerzenia ich.

Jak napisano we wprowadzeniu do obecnego trzeciego wydania, może to być jedno z bardziej wyzywających zadań dla badaczy zajmujących się obliczeniami ewolucyjnymi.

Ważne jest także kontynuowanie badań nad czynnikami wpływającymi na możliwości rozwiązywania różnych zadań (zazwyczaj optymalizacyjnych) za pomocą systemów ewolucyjnych. Co jest powodem, że zadanie jest trudne lub łatwe do rozwiązania za pomocą metody ewolucyjnej? Jest to podstawowa sprawa dla obliczeń ewolucyjnych. Pewne wyniki związane z zadaniami zawodnymi, pejzażami chropowatych dopasowań, epistazą czy funkcjami z królewską drogą są krokami w kierunku przybliżonych odpowiedzi na to pytanie.

Optymalizacja funkcji

Przez wiele lat większość metod ewolucyjnych używanych do optymalizacji funkcji oceniano i porównywano między sobą. Wygląda też na to, że dziedzina optymalizacji funkcji pozostanie podstawą wielu nowych porównań i badania nowych możliwości różnych algorytmów. Oczekuje się, że powstaną nowe teorie dotyczące metod ewolucyjnych w optymalizacji funkcji (na przykład płodzące algorytmy genetyczne [289]). Dodatkowo spodziewamy się postępu w:

- Rozwoju metod postępowania z ograniczeniami. Jest to w ogóle ważne zagadnienie, tym bardziej w optymalizacji funkcji. Większość rzeczywistych zadań optymalizacji funkcji zawiera ograniczenia. Jednak jak dotąd, zaproponowano, przeanalizowano i porównano ze sobą stosunkowo mało metod.
- Opracowaniu systemów dla zadań o wielkiej skali. Jak dotąd, w większości eksperymentów, przyjmuje się stosunkowo małe liczby zmiennych. Byłoby ciekawe przeanalizować, jak skalować metody ewolucyjne do wymiaru zadania dla zadań z tysiącami zmiennych.
- Opracowaniu systemów dla zadań programowania matematycznego. W tej dziedzinie wykonano bardzo mało prac. Istnieje potrzeba przebadania systemów ewolucyjnych dla zmiennych całkowitoliczbowych czy logicznych, dla eksperymentów z zadaniami programowania mieszanego oraz całkowitoliczbowego.

Reprezentacja i operatory

Tradycyjnie algorytmy genetyczne działają na łańcuchach binarnych, strategie ewolucyjne na wektorach zmiennopozycyjnych, programowanie ewolucyjne na automatach skończonych (reprezentowanych przez macierze), w metodach zaś programowania genetycznego używa się drzew jako struktur dla osobników. Jednak istnieje potrzeba systematycznych badań związanych z:

- reprezentacją złożonych, nieliniowych obiektów o zmiennych wymiarach, a w szczególności reprezentacją „kopii" złożonych obiektów,

- rozwojem operatorów ewolucyjnych na poziomie genotypów dla takich obiektów.

Te badania można rozumieć jako krok w kierunku zbudowania złożonego hybrydowego systemu ewolucyjnego, który obejmie dodatkowe metody przeszukiwania. Na przykład wydaje się, że warto jest lepiej przebadać operatory Lamarckiana, które prowadzą do poprawy osobników w czasie ich życia. W rezultacie poprawione, „nauczone" cechy takiego osobnika są przekazywane następnemu pokoleniu.

Dodatkowe zagadnienia do rozwiązania to zrozumienie wpływu różnych czynników (w tym dwóch najważniejszych: reprezentacji i operatorów) na działanie metod ewolucyjnych. W szczególności należałoby się zbliżyć do odpowiedzi na podstawowe pytania:

- które zadania są łatwe dla metod ewolucyjnych?
- które zadania są trudne?
- dlaczego?
- jak wybrać najlepszą reprezentację i operatory dla konkretnego zadania?

Badania nad zawodem [152], funkcjami z królewską drogą [192], [212], właściwościami operatorów [315], [316], [317] lub korelacją odległości dopasowań [214] są krokami w tym kierunku.

Nielosowy dobór

W większości obecnych metod zawierających operator krzyżowania przyjmuje się losowy dobór osobników. Wygląda jednak na to, że przy obecnym trendzie odchodzenia od systemów prostych w kierunku bardziej złożonych zagadnienie nielosowego doboru będzie coraz bardziej istotne. Jest wiele możliwości do przebadania. Obejmują one wprowadzenie płci lub połączeń „rodzinnych" między osobnikami albo ustalenie pewnych preferencji (na przykład uwodzenia [333]). Pewne proste schematy były już rozważone przez kilku badaczy (na przykład metoda Eshelmana zapobiegania kazirodztwu [105]). Jednak podstawowym celem powinno być ustalenie reguł dla nielosowego doboru. Zbadano już (patrz p. 8.3.1) kilka możliwości w ramach optymalizacji wielokryterialnej. Obejmują one *wspólne funkcje*, które pomagają w uzyskaniu stabilnych populacji, oraz *etykietowanie*, w którym osobniki otrzymują etykiety. Jednak bardzo mało zrobiono w tym kierunku dla złożonych struktur chromosomów.

Systemy samoadaptacyjne

Ponieważ w algorytmach ewolucyjnych używa się idei ewolucji, więc jest bardziej niż naturalne oczekiwać, że w tych metodach wystąpią cechy samoadap-

tacji. Prócz strategii ewolucyjnych, które zawierają pewne parametry sterujące w wektorach rozwiązań, w większości innych metod używa się stałych reprezentacji, operatorów i parametrów sterujących. Najbardziej obiecujące badania w tej dziedzinie obejmują mechanizmy samoadaptacyjne w systemach dla:

- Reprezentacji osobników (jak zaproponowana przez Shaefera [354]; do tej kategorii także należy metoda dynamicznego kodowania parametrów [347] oraz nieporządny algorytm genetyczny [155]).
- Operatorów. Jest oczywiste, że różne operatory grają różną rolę na różnych etapach procesu ewolucyjnego. Operatory powinny się dostosowywać (jak na przykład krzyżowanie adaptacyjne [341], [367]). Jest to w szczególności prawdziwe dla zmiennych w czasie dopasowań.
- Parametrów sterujących. Przeprowadzono już badania w tym kierunku: z adaptacyjnymi liczebnościami populacji [12] lub adaptacyjnymi prawdopodobieństwami operatorów [77], [216], [368]. Jednak dużo pozostaje do zrobienia.

Wydaje się, że jest to jeden z najbardziej obiecujących kierunków badań. Zresztą moc algorytmów ewolucyjnych polega na ich adaptacyjności.

Systemy współewoluujące

Wzrasta zainteresowanie systemami współewoluującymi, w których zachodzi więcej niż jeden proces ewolucji. Zazwyczaj występują w nim różne populacje (na przykład dodatkowe populacje pasożytów lub drapieżników), które działają wzajemnie na siebie. W takich systemach funkcja oceny dla jednej populacji może zależeć od stanu procesu ewolucyjnego w innej(ych) populacji(ach). Jest to ważne zagadnienie przy modelowaniu sztucznego życia, zastosowań w przedsiębiorczości itp.

Systemy współewoluujące mogą być istotne w zadaniach o wielkiej skali [311], w których (wielkie) zadania są dekomponowane na mniejsze podzadania. Dla każdego z podzadań można utworzyć oddzielny proces ewolucyjny, zależny od innych procesów. Zazwyczaj także ocena osobników w jednej populacji zależy od rozwoju innych populacji.

Opisywaliśmy już (rozdz. 7) algorytm współewoluujący (GENOCOP III), w którym współistnieją ze sobą dwie populacje (punktów niekoniecznie dopuszczalnych i punktów odniesienia, w pełni dopuszczalnych). W tym systemie ocena punktów zależy od bieżącej populacji punktów odniesienia. W tym samym rozdziale przedstawiono także podejście zaproponowane przez Le Richea i in. [239], w którym dwie populacje osobników współdziałają ze sobą przy zbliżaniu się do dopuszczalnego rozwiązania optymalnego z dwóch kierunków (z obszarów dopuszczalnego i niedopuszczalnego w przestrzeni przeszukiwań). Także Paredis [307] eksperymentował z systemami współewoluującymi w zadaniach spełniania ograniczeń (patrz p. 7.4 oraz 15.3 L).

Ostatnio użyto systemu współewoluującego [295] do modelowania strategii postępowania dwóch rywalizujących ze sobą zakładów (autobusowego i kolejowego), które rywalizują o pasażerów na tej samej trasie. Rzecz jasna zysk przedsiębiorstwa zależy od obranej strategii (objętości przewozów i cen) drugiego przedsiębiorstwa. Badania objęły zależności między różnymi strategiami w czasie.

Struktury diploidalne/poliploidalne a struktury haploidalne

Zastosowanie diploidów (lub poliploidów) można rozumieć jako sposób wprowadzenia pamięci do struktur osobników. Zamiast pojedynczego chromosomu (struktury haploidalnej) zawierającego dokładną informację o osobniku, struktura diploidalna składa się z pary chromosomów. Wybór między dwoma wartościami odbywa się za pomocą funkcji dominacji. Struktury diploidalne (poliploidalne) mają szczególne znaczenie w środowisku niestacjonarnym (to znaczy dla funkcji celu zmiennych w czasie) i przy modelowaniu złożonych systemów (ewentualnie z modelami współewoluującymi). Jednak nie istnieje teoria potwierdzająca zalety włączenia funkcji dominacji do systemu. Są tylko niezbyt obszerne dane eksperymentalne w tej dziedzinie.

Modele równoległe

Można się spodziewać, że obliczenia równoległe umożliwią uzyskanie rozwiązań zadań obecnie niemożliwych do rozwiązania. Jest to niewątpliwie jedna z najważniejszych dziedzin informatyki. Algorytmy ewolucyjne dobrze nadają się do programowania równoległego. Jak stwierdził Goldberg [154]:

> W świecie, w którym algorytmy działające seryjnie są zazwyczaj przerabiane na równoległe za pomocą niezliczonych sztuczek i wykrętów, jest niemałą ironią, że algorytmy genetyczne (algorytmy w dużej mierze równoległe) są przerabiane na seryjne za pomocą równie nienaturalnych sztuczek i wykrętów.

Jednak nie ma ustalonej metodyki wprowadzania równoległości do algorytmów genetycznych. Istniejące rozwiązania równoległe można zaklasyfikować do następujących kategorii:

- *Masowe algorytmy genetyczne równoległe.* W takich algorytmach używa się dużej liczby procesorów (zazwyczaj 2^{10} i więcej). Często jeden procesor odpowiada jednemu osobnikowi w populacji. W tym modelu jest wiele możliwości wyboru metod i doboru (łączenia łańcuchów do krzyżowania). Pewne prace eksperymentalne w tej dziedzinie opisał Mühlenbein [286].
- *Modele równoległych wysp.* W takich algorytmach przyjmuje się, że ewoluuje równolegle kilka populacji. Modele te obejmują migrację (ruch osobników z jednej populacji do innej) i krzyżowania między osobnikami z różnych

podpopulacji. Opisano wiele eksperymentów z modelami równoległymi. Pełne ich omówienie można znaleźć w pracy Whitleya [396].

- *Hybrydowe algorytmy genetyczne równoległe.* Są one podobne do pierwszego modelu (masowych algorytmów genetycznych równoległych) przez to, że jest w nich jednoznaczna odpowiedniość między procesorami a osobnikami. Jednak używa się tylko niedużej liczby procesorów. Dodatkowo w tych algorytmach uwzględnia się inne (heurystyczne) algorytmy (na przykład wzrostu), aby poprawić działanie systemu. Zazwyczaj opisuje się tu wyniki eksperymentalne [286], [164], gdyż analiza takich systemów wcale nie jest łatwa.

Modele równoległe mogą także być naturalnym podłożem dla innych pojęć z zakresu obliczeń równoległych, takich jak nielosowy dobór, pewne aspekty samoadaptacji lub systemy współewoluujące.

Inne osiągnięcia

Jest także wiele innych ciekawych osiągnięć w dziedzinie obliczeń ewolucyjnych. Jedno z nich to pojawienie się tak zwanych *algorytmów kulturowych* (patrz także p. 7.4). Jak napisano w [330]:

> Algorytmy kulturowe podtrzymują dwa modele dziedziczenia, jeden na poziomie mikroewolucji pojęć cech, a drugi na poziomie makroewolucji pojęć wiary. Te dwa modele oddziałują na siebie za pomocą kanału komunikacyjnego, który pozwala, aby zachowanie osobników zmieniało strukturę wiary i aby struktura wiary ograniczała sposób zachowania się osobnika. Tak więc algorytmy kulturowe można opisać za pomocą trzech podstawowych składowych: struktury wiary, struktury populacji i kanału komunikacji.

Podstawowa struktura algorytmu kulturowego jest przedstawiona na rys. 16.1.

Wydaje się dodatkowo, że algorytmy kulturowe są odpowiednie do sterowania „ewolucją w ewolucji". Określenie „ewolucja w ewolucji" oznacza, że pewne elementy przestrzeni ewolucji (jak człowiek w ewolucji socjalnej lub gen w ewolucji biologicznej albo pewne jednostki w ewolucji materii) poprawiają swoje możliwości uczenia się przez doświadczenie wyniesione z procesu ewolucji. Jest to uogólnienie idei analogii między ewolucją genetyczną gatunków biologicznych a ewolucją kulturalną społeczeństw ludzkich na abstrakcyjny proces ewolucji adaptacyjnej. Te analogie były wspaniale przedstawione przez Richarda Dawkinsa w jego książce [81]. Jeden z argumentów potwierdzających powyższe idee można obejrzeć na rys. 16.2. Pierwsza kolumna symbolizuje na nim czas liczony od początku wszechświata do teraz. Każda następna kolumna jest powiększeniem ostatnich 20% poprzedniej kolumny. Punkty oznaczone gwiazdką reprezentują najważniejsze wydarzenia w procesie ewolucji, które doprowadziły do obecnej cywilizacji. Zauważmy, że krzywa wyznaczona przez gwiazdki rośnie dużo szybciej niż funkcja wykładnicza, powiedzmy 5^x, gdyż gdyby wzrost wynosił 5^x, to gwiazdki byłyby na tym samym poziomie.

```
Procedure algorytm kulturowy
begin
  t ← 0
  utwórz populację początkową P(t)
  utwórz początkową sieć wierzeń B(t)
  utwórz kanał komunikacji
  oceń P(t)
  while (not warunek zakończenia) do
  begin
    otwórz kanał komunikacji między P(t) a B(t)
    dopasuj B(t)
    otwórz kanał komunikacji między P(t) a B(t)
    zmień dopasowanie P(t)
    t ← t + 1
    wybierz P(t) z P(t − 1)
    przeprowadź ewolucję P(t)
    oceń P(t)
  end
end
```

Rys. 16.1. Struktura algorytmu kulturowego

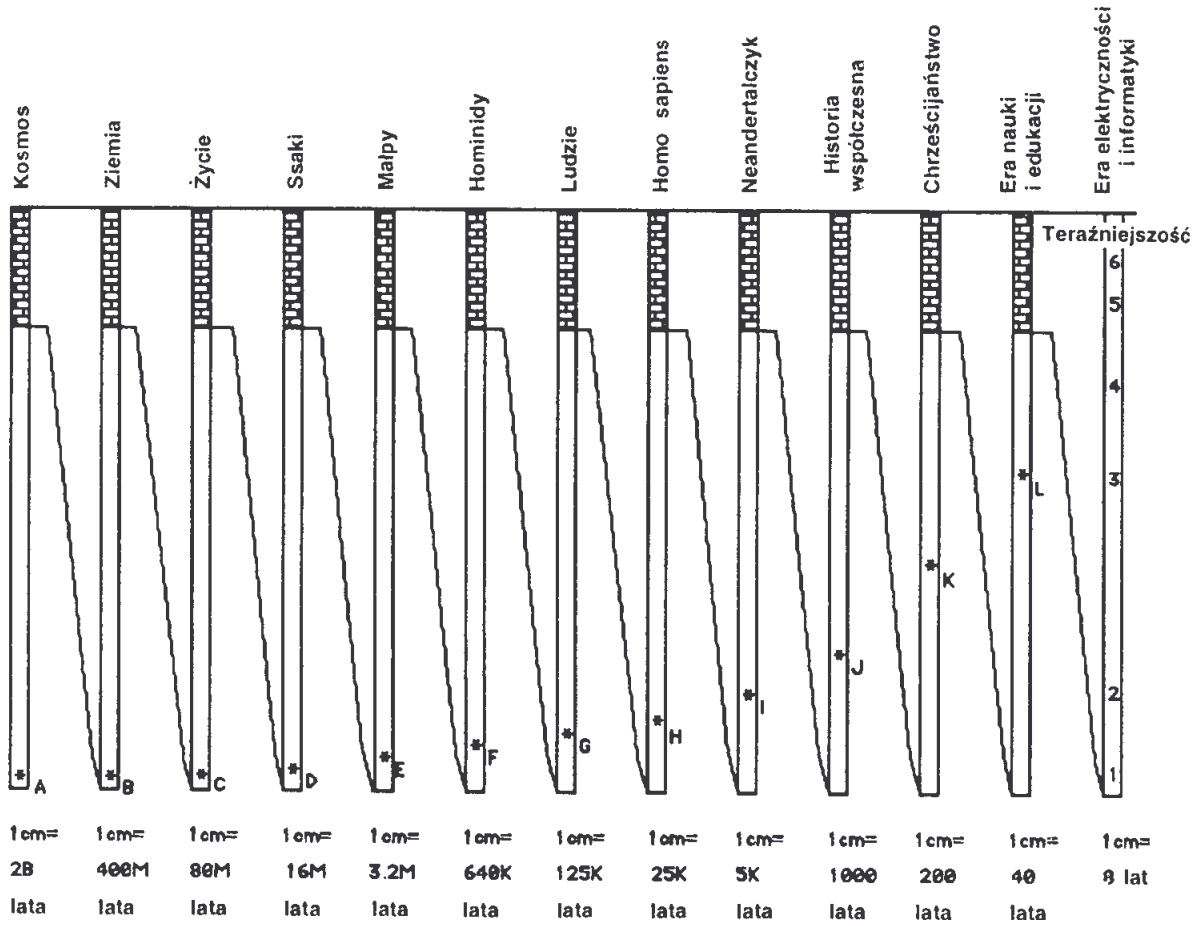
Zacytujmy z [371]:

> Większość osób zajmujących się rozwojem fizycznym, chemicznym i biologicznym zgadza się, że rozwój cząstek organicznych rozpoczął się około 4 bilionów lat temu. Pierwsze żyjące komórki pojawiły się okolo 3,5 biliona lat temu, a pierwsze proste zwierzęta wielokomórkowe pojawiły się mniej więcej 6000 milionów lat temu. Wspólny przodek małp naczelnych i ludzi pojawił się jakieś 6 milionów lat temu, a początek historii pisanej zaczął się około 6000 lat temu.

Oczywiście ta dziedzina wymaga dalszych badań. Kilka interesujących wyników związanych z zastosowaniem algorytmów kulturowych do różnych zadań opisano w [60], [326], [327], [328], [330], [331], [377].

Warto zauważyć, że istnieje wiele innych podejść do uczenia, optymalizacji i rozwiązywania zadań opartych na innych naturalnych metaforach z natury. Najbardziej znane przykłady to sieci neuronowe i symulowane wyżarzanie. Wzrasta zainteresowanie wszystkimi tymi dziedzinami. Najbardziej owocnym i wyzywającym kierunkiem wydaje się być „kombinacja" pewnych pomysłów rozrzuconych obecnie po różnych dziedzinach. Schwefel i Männer napisali we wprowadzeniu do materiałów z pierwszego sympozjum na temat rozwiązywania zadań równoległych z natury [351]:

> Jest faktem, że strategie ewolucyjne w Europie i algorytmy genetyczne w USA przetrwały ponad dziesięciolecie nieakceptacji i lekceważenia. Jest jednak tak-



*A początek wszechświata i pierwsze elementy chemiczne
*B początki skał na Ziemi
*C początki zwierząt wielokomórkowych
*D początki łożyskowców
*E początki Propliopithecus
*F początki Australopithecinae
*G początki Homo erectus
*H początki Homo sapiens
*I początki człowieka z Cro-Magnon
*J początki pierwszych cywilizacji
*K początek Renesansu
*L początek współczesnej nauki i technologii produkcji

1 – pierwszy samolot
2 – teoria mechaniki kwantowej
3 – pierwszy wybuch atomowy, pierwsze komputery
4 – pierwszy człowiek w kosmosie
5 – elementy scalone VLSI
6 – nadprzewodzenie w wysokich temperaturach

Rys. 16.2. Ponadwykładniczy wzrost szybkości rozwoju. Lewa kolumna na tym rysunku reprezentuje czas od początku powstania wszechświata do teraz. Druga kolumna jest powiększeniem tej części pierwszej kolumny, która znajduje się ponad pochyłą linią łączącą obie kolumny. Podobnie jest z następnymi kolumnami

że prawdą, że dużo więcej warstw pomysłów rozwinęło się w izolacji geograficznej i wobec tego nie było wprowadzonych do utworzonego potomka. Jest teraz czas na nowe generacje algorytmów, które zrobią użytek z bogatej puli pomysłów znajdującej się po obu stronach Atlantyku, a także zrobią użytek ze sprzyjającego otoczenia przejawiającego się w postaci masowo pojawiających się systemów z procesorami równoległymi.

Pisaniu tej książki przyświecały dwa podstawowe cele. Jeden to przekonanie Czytelnika, że ewolucja jest potężnym i ogólnym pojęciem, które powinno znaleźć miejsce w wielu metodach rozwiązywania zadań. W szczególności cała dziedzina sztucznej inteligencji powinna przesunąć się w kierunku zastosowania metod ewolucyjnych. Lawrence Fogel [125] powiedział w swoim wykładzie plenarnym na światowym kongresie na temat inteligencji obliczeniowej (Orlando, 27 czerwca – 2 lipca 1994 r.):

> Jeżeli celem jest generacja sztucznej inteligencji, to znaczy rozwiązywanie nowych problemów nowymi sposobami, to nie jest właściwe użycie skończonego zbioru reguł. Reguły potrzebne do rozwiązywania każdego zadania powinny się po prostu rozwijać... .

Drugim celem było podkreślenie podobieństw między różnymi metodami ewolucyjnymi. Metody te omówiono w książce pod względem budowy programów ewolucyjnych dla poszczególnych klas zadań. W ten sposób wszystkie różnice między tymi metodami (na przykład algorytmami genetycznymi, strategiami ewolucyjnymi, programowaniem ewolucyjnym, programowaniem genetycznym itp.) ukryto na niższym poziomie. W różnych metodach używa się po prostu różnych reprezentacji chromosomów i odpowiednich zbiorów (bardziej lub mniej genetycznych) operatorów.

Dodatek A

Dodatek ten zawiera bardzo prosty program algorytmu genetycznego napisany przez Denisa Cormiera z North Carolina State University i zmodyfikowany przez Sitę S. Raghavana z University of North Carolina w Charlotte. Program napisano w najprostszy sposób, bez zasadniczego sprawdzenia błędów. W wielu przypadkach zrezygnowano z krótszego zapisu na korzyść jasności. Do modyfikacji programu w celu użycia go do konkretnego zadania należy zmienić definicje stałych oraz określaną przez użytkownika funkcję oceny „evaluation function''. Program jest napisany dla zadania maksymalizacji z dodatnią funkcją celu. Nie ma rozróżnienia między wartościami funkcji celu oraz dopasowaniem osobników. W systemie używa się selekcji proporcjonalnej, modelu elitarnego, krzyżowania jednopunktowego oraz mutacji jednorodnej (dużo lepsze wyniki można uzyskać, zastępując mutację jednorodną przez mutację gaussowską; zachęcamy Czytelnika do wprowadzenia odpowiednich zmian do systemu – patrz ćwiczenie 6 z dodatku D).

W programie nie używa się grafiki, a nawet wysyłania informacji na ekran. Powinien on więc być łatwy do przeniesienia na różne komputery. Program ten można też znaleźć w ftp.uncc.edu w katalogu coe/evol, plik prog.c.

Żądany przez program plik wejściowy powinien mieć nazwę „gadata.txt''; system zapisuje wyniki w pliku „galog.txt''. Plik wejściowy zawiera kilka wierszy: liczba wierszy odpowiada liczbie zmiennych. Każdy wiersz zawiera ograniczenie dolne i górne dla zmiennej o kolejnym numerze (to znaczy pierwszy wiersz zawiera ograniczenie dolne i górne dla pierwszej zmiennej, drugi wiersz dla drugiej zmiennej itd.).

```
/*************************************************************/
/* Jest to prosty program algorytmu genetycznego             */
/* z dodatnią funkcją oceny, przy czym dopasowanie           */
/* osobnika jest takie samo jak wartość funkcji celu.        */
/*************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Zmienić poniższe parametry stosownie do potrzeb */

#define POPSIZE 50                          /* liczebność populacji */
#define MAXGENS 1000                    /* maksymalna liczba pokoleń */
#define NVARS 3                        /* liczba zmiennych w zadaniu */
#define PXOVER 0.8               /* prawdopodobieństwo krzyżowania */
#define PMUTATION 0.15               /* prawdopodobieństwo mutacji */
#define TRUE 1
#define FALSE 0

int generation;                     /* numer bieżącego pokolenia */
int cur_best;                                /* najlepszy osobnik */
FILE *galog;                                    /* plik wyjściowy */

struct genotype                /* genotyp (GT), członek populacji */
{
double gene[NVARS];                          /* łańcuch zmiennych */
   double fitness;                              /* dopasowanie GT */
   double upper[NVARS];      /* górne ograniczenia na zmienne GT */
   double lower[NVARS];      /* dolne ograniczenia na zmienne GT */
   double rfitness;                        /* dopasowanie względne */
   double cfitness;                          /* dopasowanie łączne */
};

struct genotype population[POPSIZE+1];                /* populacja */
struct genotype newpopulation[POPSIZE+1];     /* nowa populacja, */
                                               /* która zamienia */
                                              /* stare pokolenie */

/* Deklaracje procedur używanych w tym algorytmie genetycznym */

void initialize(void);
double randval(double, double);
void evaluate(void);
void keep_the_best(void);
void elitist(void);
void select(void);
void crossover(void);
void Xover(int,int);
void swap(double *, double *);
void mutate(void);
void report(void);
```

```
/*************************************************************/
/* Procedura inicjująca. Nadaje ona genom wartości wewnątrz    */
/* ograniczeń. Ustawia także (na zero) wartości dopasowań dla  */
/* wszystkich członków populacji. Wczytuje ograniczenia dolne  */
/* i górne zmiennych z pliku wejściowego "gadata.txt",         */
/* a następnie losowo generuje wartości wewnątrz tych          */
/* ograniczeń dla każdego genu w każdym genotypie w populacji. */
/* Format pliku wejściowego "gadata.txt" wygląda następująco: */
/*ograniczenie_dolne_zmiennej1 ograniczenie_górne_zmiennej1*/
/*ograniczenie_dolne_zmiennej2 ograniczenie_górne_zmiennej2 ...*/
/*************************************************************/

void initialize(void)
{
FILE *infile;
int i, j;
double lbound, ubound;

if ((infile=fopen("gadata.txt","r"))==NULL)
    {
    fprintf(galog,"/nNie mogę otworzyć pliku wejściowego!/n");
    exit(1);
    }

/* ustalanie wartości wewnątrz ograniczeń */

for (i=0; i < NVARS; i++)
    {
    fscanf(infile, "%lf",&lbound);
    fscanf(infile, "%lf",&ubound);

    for (j=0; j < POPSIZE; j++)
        {
        population[j].fitness=0;
        population[j].rfitness=0;
        population[j].cfitness=0;
        population[j].lower[i]=lbound;
        population[j].upper[i]= ubound;
        population[j].gene[i]=randval(population[j].lower[i],
                                  population[j].upper[i]);
        }
    }

fclose(infile);
}
```

```
/*************************************************************/
/* Generator liczb losowych, generuje wartości wewnątrz      */
/* ograniczeń                                                */
/*************************************************************/

double randval(double low, double high)
{
double val;
val=((double)(rand()%1000)/1000.0)*(high-low)+low;
return(val);
}

/*************************************************************/
/* Funkcja oceny. Jest to funkcja ustalana przez użytkownika. */
/* Po każdej jej zmianie należy skompilować program.         */
/* Obecnie jest to funkcja: x[1]^2-x[1]*x[2]+x[3]            */
/*************************************************************/

void evaluate(void)
{
int mem;
int i;
double x[NVARS+1];

for (mem=0; mem < POPSIZE; mem++)
    {
    for (i=0; i < NVARS; i++)
        x[i+1]=population[mem].gene[i];

    population[mem].fitness=(x[1]*x[1])-(x[1]*x[2])+x[3];
    }
}

/*************************************************************/
/* Procedura keep_the_best. Zapamiętuje ona najlepszego      */
/* dotychczasowego członka populacji. Warto zwrócić uwagę,   */
/* że w ostatnim elemencie macierzy population znajduje      */
/* się kopia najlepszego osobnika.                           */
/*************************************************************/

void keep_the_best()
{
int mem;
int i;
cur_best=0;     /* zapamiętanie wskaźnika najlepszego osobnika */
```

```
for (mem=0; mem < POPSIZE; mem++)
    {
    if (population[mem].fitness > population[POPSIZE].fitness)
        {
        cur_best=mem;
        population[POPSIZE].fitness=population[mem].fitness;
        }
    }
/* po znalezieniu najlepszego członka populacji skopiuj geny */
for (i=0; i < NVARS; i++)
    population[POPSIZE].gene[i]=population[cur_best].gene[i];
}

/*************************************************************
/
/* Procedura elitist. Najlepszy osobnik z poprzedniego        */
/* pokolenia jest zapamiętywany jako ostatni w macierzy.      */
/* Jeżeli najlepszy osobnik z bieżącego pokolenia jest        */
/* gorszy niż najlepszy osobnik z poprzednich pokoleń, to ten  */
/* ostatni zastępuje najgorszego osobnika bieżącej populacji. */
/*************************************************************/

void elitist()
{
int i;
double best, worst;/* najlepsza i najgorsza wartość dopasowania */
int best_mem, worst_mem;                /* wskaźniki najlepszego */
                                        /* i najgorszego osobnika */

best=population[0].fitness;
worst=population[0].fitness;
for (i=0; i < POPSIZE-1; ++i)
    {
    if(population[i].fitness > population[i+1].fitness)
        {
        if (population[i].fitness > = best)
            {
            best=population[i].fitness;
            best_mem=i;
            }
        if (population[i+1].fitness < = worst)
            {
            worst=population[i+1].fitness;
            worst_mem=i+1;
```

```
            }
        }
    else
        {
        if (population[i].fitness < = worst)
            {
            worst=population[i].fitness;
            worst_mem=i;
            }
        if (population[i+1].fitness > = best)
            {
            best=population[i+1].fitness;
            best_mem=i+1;
            }
        }
    }
/* Jeżeli najlepszy osobnik z nowej populacji jest lepszy niż    */
/* najlepszy osobnik z poprzednich populacji, to skopiuj         */
/* najlepszego z nowej populacji; jeżeli nie, to zastąp          */
/* najgorszego osobnika z bieżącej populacji przez               */
/* najlepszego z poprzednich pokoleń.                            */

if (best > = population[POPSIZE].fitness)
    {
    for (i=0; i < NVARS; i++)
        population[POPSIZE].gene[i]=population[best_mem].gene[i];
    population[POPSIZE].fitness=population[best_mem].fitness;
    }
else
    {
    for (i=0; i < NVARS; i++)
        population[worst_mem].gene[i]=population[POPSIZE].gene[i];
    population[worst_mem].fitness=population[POPSIZE].fitness;
    }
}

/*************************************************************/
/* Procedura wyboru. Standardardowa selekcja proporcjonalna dla */
/* zadania maksymalizacji z modelem elitarnym - zapewnia, że    */
/* przeżywa najlepszy osobnik populacji.                         */
/*************************************************************/

void select(void)
{
```

```
int mem, i, j, k;
double sum=0;
double p;

/* wyznaczanie całkowitego dopasowania populacji */
for (mem=0; mem < POPSIZE; mem++)
    {
    sum += population[mem].fitness;
    }

/* obliczanie dopasowania względnego */
for (mem=0; mem < POPSIZE; mem++)
    {
    population[mem].rfitness=population[mem].fitness/sum;
    }
population[0].cfitness=population[0].rfitness;

/* obliczanie dopasowania łącznego */
for (mem=1; mem < POPSIZE; mem++)
    {
    population[mem].cfitness=population[mem-1].cfitness+
                                population[mem].rfitness;
    }

/* końcowy wybór przeżywających na podstawie dopasowań łącznych */
for (i=0; i < POPSIZE; i++)
    {
    p=rand()%1000/1000.0;
    if (p < population[0].cfitness)
        newpopulation[i]=population[0];
    else
        {
        for (j=0; j < POPSIZE;j++)
            if (p > = population[j].cfitness &&
                p < population[j+1].cfitness)
            newpopulation[i]=population[j+1];
        }
    }

/* kopiowanie populacji po jej utworzeniu */
for (i=0; i < POPSIZE; i++)
    population[i]=newpopulation[i];
}
```

```
/*************************************************************/
/* Wybór do krzyżowania. Wybór dwóch rodziców, którzy wezmą udział */
/* w krzyżowaniu. Przyjęto krzyżowanie jednopunktowe.          */
/*************************************************************/

void crossover(void)
{
int i, mem, one;
int first=0;          /* obliczanie liczby wybranych osobników */
double x;

for (mem=0; mem < POPSIZE; ++mem)
    {
    x=rand()%1000/1000.0;
    if (x < PXOVER)
        {
        ++first;
        if (first % 2 == 0)
            Xover(one, mem);
        else
            one=mem;
        }
    }
}

/*************************************************************/
/* Krzyżowanie: wykonanie krzyżowania dwóch wybranych rodziców. */
/*************************************************************/

void Xover(int one, int two)
{
int i;
int point;                                  /* punkt krzyżowania */

/* wybór punktu krzyżowania */
if(NVARS > 1)
    {
    if(NVARS == 2)
        point=1;
    else
        point=(rand() % (NVARS-1))+1;

    for (i=0; i < point; i++)
        swap(&population[one].gene[i], &population[two].gene[i]);
    }
}
```

```
/*************************************************************/
/* Wymiana. Procedura wymiany, która pomaga wymienić 2 zmienne. */
/*************************************************************/

void swap(double *x, double *y)
{
double temp;

temp=*x;
*x=*y;
*y=temp;

}

/*************************************************************/
/* Mutacja. Losowa mutacja jednorodna. Zmienna wybrana do mutacji */
/* jest zamieniana na wartość losową zawartą wewnątrz          */
/* ograniczenia dolnego i górnego tej zmiennej.                */
/*************************************************************/

void mutate(void)
{
int i, j;
double lbound, hbound;
double x;

for (i=0; i < POPSIZE; i++)
    for (j=0; j < NVARS; j++)
        {
        x=rand()%1000/1000.0;
        if (x < PMUTATION)
            {
            /* wyznaczanie ograniczeń dla mutowanej zmiennej */
            lbound=population[i].lower[j];
            hbound=population[i].upper[j];
            population[i].gene[j]=randval(lbound, hbound);
            }
        }
}
```

```
/*************************************************************/
/* Procedura zapisu wyników. Zapisuje się w niej postępy     */
/* symulacji. Dane zapisywane do pliku wyjściowego są        */
/* rozdzielone przecinkami.                                  */
/*************************************************************/

void report(void)
{
int i;
double best_val;            /* najlepsze dopasowanie populacji */
double avg;                   /* średnie dopasowanie populacji */
double stddev; /* odchylenie standardowe dopasowania populacji */
double sum_square;   /* suma kwadratów do obliczania odchylenia*/
                                                /* standardowego */
double square_sum;     /* kwadrat sumy do obliczania odchylenia*/
                                                /* standardowego */
double sum;                 /* całkowite dopasowanie populacji */

sum=0.0;
sum_square=0.0;

for (i=0; i < POPSIZE; i++)
   {
   sum += population[i].fitness;
   sum_square += population[i].fitness * population[i].fitness;
   }

avg=sum/(double)POPSIZE;
square_sum=avg * avg * (double)POPSIZE;
stddev=sqrt((sum_square-square_sum)/(POPSIZE-1));
best_val=population[POPSIZE].fitness;

fprintf(galog, "\n%5d,   %6.3f, %6.3f, %6.3f \n\n", populacja,
        best_val, avg, stddev);
}
```

```
/*************************************************************/
/* Główna procedura. W każdym pokoleniu następuje wybór      */
/* najlepszych jej osobników, krzyżowanie i mutacja, a następnie */
/* ocena wynikłej populacji, aż do spełnienia warunku końcowego. */
/*************************************************************/

void main(void)
{
int i;

if ((galog=fopen("galog.txt","w"))==NULL)
    {
    exit(1);
    }
generation=0;

fprintf(galog, "\n numer   najlepsza średnie    odchylenie \n");
fprintf(galog, " populacji wartość   dopasowanie standardowe \n");

initialize();
evaluate();
keep_the_best();
while(generation < MAXGENS)
        {
        generation++;
        select();
        crossover();
        mutate();
        report();
        evaluate();
        elitist();
        }
fprintf(galog,"\n\n Symulacja zakończona \n");
fprintf(galog,"\n Najlepszy osobnik: \n");

for (i=0; i < NVARS; i++)
    {
    fprintf (galog,"\n var(%d)=%3.3f",i,
             population[POPSIZE].gene[i]);
    }
fprintf(galog,"\n\n Najlepsze dopasowanie=%3.3f",
population[POPSIZE].fitness);
fclose(galog);
printf("Sukces\n");
}
/*************************************************************/
```

Dodatek B

Jest kilka funkcji testowych, których można użyć do różnych eksperymentów (dodatkowe zadania można znaleźć w dodatku A w [350], [186] i [113])[1].

1. Funkcja De Jonga F1:

$$\sum_{i=1}^{3} x_i^2, \qquad \text{gdzie } -5{,}12 \leqslant x_i \leqslant 5{,}12.$$

   Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2, x_3) = (0, 0, 0)$.
2. Funkcja De Jonga F2:

$$100(x_1^2 - x_2)^2 + (1 - x_1)^2, \qquad \text{gdzie } -2{,}048 \leqslant x_i \leqslant 2{,}048.$$

   Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (1, 1)$.
3. Funkcja De Jonga F3:

$$\sum_{i=1}^{5} \text{integer}(x_i), \qquad \text{gdzie } -5{,}12 \leqslant x_i \leqslant 5{,}12.$$

   Ta funkcja przyjmuje globalną wartość minimalną -30 dla wszystkich $-5{,}12 \leqslant x_i \leqslant -5{,}0$.
4. Funkcja De Jonga F4:

$$\sum_{i=1}^{30} i x_i^4 + \text{Gauss}(0, 1), \qquad \text{gdzie } -1{,}28 \leqslant x_i \leqslant 1{,}28.$$

   Ta funkcja (bez zakłócania gaussowskiego) przyjmuje globalną wartość minimalną 0 w $(x_1, x_2, \ldots, x_{30}) = (0, 0, \ldots, 0)$.

[1] Patrz także [401], gdzie omawia się tworzenie funkcji testowych.

5. Funkcja De Jonga F5:

$$\frac{1}{1/K + \sum_{j=1}^{25} f_j^{-1}(x_1, x_2)}, \qquad \text{gdzie } f_j(x_1, x_2) = c_j + \sum_{i=1}^{2} (x_i - a_{ij})^6,$$

przy czym $-65{,}536 \leqslant x_i \leqslant 65{,}536$, $K = 500$, $c_j = j$, oraz

$$[a_{ij}] = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32 & -16 & \dots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \dots & 32 & 32 & 32 \end{bmatrix}$$

Ta funkcja przyjmuje globalną wartość minimalną 0,998 w $(x_1, x_2) = (-32, -32)$.

6. Funkcja Schaffera F6:

$$0{,}5 + \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0{,}5}{[1{,}0 + 0{,}001(x_1^2 + x_2^2)]^2}, \qquad \text{gdzie } -100 \leqslant x_i \leqslant 100.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (0, 0)$.

7. Funkcja Schaffera F7:

$$(x_1^2 + x_2^2)^{0{,}25}[\sin^2(50(x_1^2 + x_2^2)^{0{,}1}) + 1{,}0], \qquad \text{gdzie } -100 \leqslant x_i \leqslant 100.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (0, 0)$.

8. Funkcja Goldsteina-Price'a:

$$[1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \cdot$$
$$\cdot [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)],$$
$$\text{gdzie } -2 \leqslant x_i \leqslant 2$$

Ta funkcja przyjmuje globalną wartość minimalną 3 w $(x_1, x_2) = (0, -1)$.

9. Funkcja Branina RCOS:

$$a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e,$$

gdzie $-5 \leqslant x_1 \leqslant 10{,}0 \leqslant x_2 \leqslant 15$, oraz $a = 1$, $b = 5{,}1/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $e = 10$, $f = 1/(8\pi)$.

Ta funkcja przyjmuje globalną wartość minimalną 0,397887 w trzech różnych punktach $(x_1, x_2) = (-\pi, 12{,}275)$, $(\pi, 2{,}275)$ oraz $(9{,}42478, 2{,}475)$.

10. Rodzina funkcji czterowymiarowych Shekela SQRN5, SQRN7, SQRN10:

$$s3(x_1, x_2, x_3, x_4) = -\sum_{j=1}^{5} \frac{1}{\sum_{i=1}^{4} (x_i - a_{ij})^2 + c_j}$$

$$s4(x_1, x_2, x_3, x_4) = -\sum_{j=1}^{7} \frac{1}{\sum_{i=1}^{4} (x_i - a_{ij})^2 + c_j}$$

$$s5(x_1, x_2, x_3, x_4) = -\sum_{j=1}^{10} \frac{1}{\sum_{i=1}^{4} (x_i - a_{ij})^2 + c_j}$$

gdzie $0 \leqslant x_i \leqslant 10$ dla $i = 1, 2, 3, 4$, a a_{ij} oraz c_{ij} są podane w tabl. B.1.

Te funkcje przyjmują globalne wartości minimalne w punktach $(x_1, x_2, x_3, x_4) = (4, 4, 4, 4)$ z następującymi wartościami $s3_{min} = -10{,}15320$, $s4_{min} = -10{,}402820$, $s5_{min} = -10{,}53628$.

Tablica B.1. Dane dla funkcji $s3$, $s4$, $s5$

j	a_{1j}	a_{2j}	a_{3j}	a_{4j}	c_j
1	4,0	4,0	4,0	4,0	0,1
2	1,0	1,0	1,0	1,0	0,2
3	8,0	8,0	8,0	8,0	0,2
4	6,0	6,0	6,0	6,0	0,4
5	3,0	7,0	3,0	7,0	0,6
6	2,0	9,0	2,0	9,0	0,6
7	5,0	5,0	3,0	3,0	0,3
8	8,0	1,0	8,0	1,0	0,7
9	6,0	2,0	6,0	2,0	0,5
10	7,0	3,6	7,0	3,6	0,5

11. Funkcja grzbietu wielbłąda sześciogarbnego:

$$\left(4 - 2{,}1x_1^2 + \frac{x_1^4}{3}\right)x_1^2 + x_1x_2 + (-4 + 4x_2^2)x_2^2,$$

gdzie $-3 \leqslant x_1 \leqslant 3$ oraz $-2 \leqslant x_2 \leqslant 2$.

Ta funkcja przyjmuje globalną wartość minimalną $-1{,}0316$ w dwóch różnych punktach $(x_1, x_2) = (-0{,}0898, 0{,}7126)$ oraz $(0{,}0898, -0{,}7126)$.

12. Funkcja Shuberta:

$$\sum_{i=1}^{5} i\cos[(i+1)x_1 + i] \cdot \sum_{i=1}^{5} i\cos[(i+1)x_2 + i],$$

gdzie $-10 \leqslant x_i \leqslant 10$ dla $i = 1, 2$.

Ta funkcja ma 760 lokalnych minimów, przy czym w 18 z nich przyjmuje wartość $-186{,}73$.

13. Funkcja Stuckmana:

$$f8(x_1, x_2) = \begin{cases} \lfloor(\lfloor m_1\rfloor + \frac{1}{2})\sin(a_1)/a_1\rfloor, & \text{jeśli } 0 \leqslant x_1 \leqslant b \\ \lfloor(\lfloor m_2\rfloor + \frac{1}{2})\sin(a_2)/a_2\rfloor, & \text{jeśli } b < x_1 \leqslant 10, \end{cases}$$

gdzie $0 \leqslant x_i \leqslant 10$ dla $i = 1, 2$, a m_i jest zmienną losową z zakresu (0, 100), b jest zmienną losową z zakresu (0, 10) oraz

$$a_i = \lfloor|x_1 - r_{1i}|\rfloor + \lfloor|x_2 - r_{2i}|\rfloor,$$

gdzie r_{11} jest zmienną losową z zakresu $(0, b)$, r_{12} jest zmienną losową z zakresu $(b, 10)$, r_{21} jest zmienną losową z zakresu (0, 10), a r_{22} jest zmienną losową z zakresu (0, 10) (wszystkie zmienne losowe mają rozkład równomierny).

Maksimum globalne znajduje się w

$$(x_1, x_2) = \begin{cases} (r_{11}, r_{21}), & \text{jeśli } m_1 > m_2 \\ (r_{12}, r_{22}), & \text{w pozostałym przypadku} \end{cases}$$

14. Funkcja Easoma:

$$-\cos(x_1)\cos(x_2)e^{-(x_1-\pi)^2-(x_2-\pi)^2}, \quad \text{gdzie } -100 \leqslant x_i \leqslant 100 \text{ dla } i = 1, 2.$$

Ta funkcja przyjmuje globalną wartość minimalną -1 w $(x_1, x_2) = (\pi, \pi)$.

15. Funkcja 1 Bohachevsky'ego:

$$x_1^2 + 2x_2^2 - 0{,}3\cos(3\pi x_1) - 0{,}4\cos(4\pi x_2) + 0{,}7, \quad \text{gdzie } -50 \leqslant x_i \leqslant 50.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (0, 0)$.

16. Funkcja 2 Bohachevsky'ego:

$$x_1^2 + 2x_2^2 - 0{,}3(\cos(3\pi x_1)\cos(4\pi x_2)) + 0{,}3, \quad \text{gdzie } -50 \leqslant x_1 \leqslant 50.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (0, 0)$.

17. Funkcja 3 Bohachevsky'ego:

$$x_1^2 + 2x_2^2 - 0{,}3\cos(3\pi x_1) + \cos(4\pi x_2) + 0{,}3, \quad \text{gdzie } -50 \leqslant x_i \leqslant 50.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2) = (0, 0)$.

18. Funkcja Coldville'a:

$$100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 +$$
$$+ 10{,}1((x_2 - 1)^2 + (x_4 - 1)^2) + 19{,}8(x_2 - 1)(x_4 - 1),$$
$$\text{gdzie } -10 \leqslant x_i \leqslant 10.$$

Ta funkcja przyjmuje globalną wartość minimalną 0 w $(x_1, x_2, x_3, x_4) = (1, 1, 1, 1)$.

Dodatek C

Jest kilka funkcji, których można użyć do różnych eksperymentów z optymalizacją z ograniczeniami (dodatkowe zadania można znaleźć w dodatku A w [350], [186] i [113]).

1. Zadanie ([114]) polega na minimalizacji

$$G1(\mathbf{x}, \mathbf{y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i$$

przy ograniczeniach

$$\begin{aligned}
&2x_1 + 2x_2 + y_6 + y_7 \leqslant 10, && 2x_1 + 2x_3 + y_6 + y_8 \leqslant 10,\\
&2x_2 + 2x_3 + y_7 + y_8 \leqslant 10, && -8x_1 + y_6 \leqslant 0,\\
&-8x_2 + y_7 \leqslant 0, && -8x_3 + y_8 \leqslant 0,\\
&-2x_4 - y_1 + y_6 \leqslant 0, && -2y_2 - y_3 + y_7 \leqslant 0,\\
&-2y_4 - y_5 + y_8 \leqslant 0, && 0 \leqslant x_i \leqslant 1,\ i = 1, 2, 3, 4,\\
&0 \leqslant y_i \leqslant 1,\ i = 1, 2, 3, 4, 5, 9, && 0 \leqslant y_i,\ i = 6, 7, 8.
\end{aligned}$$

Globalnym rozwiązaniem jest $(\mathbf{x}^*, \mathbf{y}^*) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$ z wartością $G1(\mathbf{x}^*, \mathbf{y}^*) = -15$.

2. Zadanie ([186]) polega na minimalizacji funkcji

$$G2(\bar{X}) = x_1 + x_2 + x_3$$

przy ograniczeniach

$1 - 0{,}0025(x_4 + x_6) \geqslant 0,$ $\quad x_1x_6 - 833{,}33252x_4 - 100x_1 + 83333{,}333 \geqslant 0,$

$1 - 0{,}0025(x_5 + x_7 - x_4) \geqslant 0,$ $\quad x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geqslant 0,$

$1 - 0{,}01(x_8 - x_5) \geqslant 0,$ $\quad x_3x_8 - 1250000 - x_3x_5 + 2500x_5 \geqslant 0,$

$100 \leqslant x_1 \leqslant 10000,$ $\quad 100 \leqslant x_i \leqslant 10000,\ i = 2,\ 3,$

$10 \leqslant x_i \leqslant 1000,\ i = 4,\ \ldots,\ 8.$

Zadanie ma 3 liniowe i 3 nieliniowe ograniczenia. Funkcja $G2$ jest liniowa i ma globalne minimum w

$$\bar{X}^* = (579{,}3167,\ 1359{,}943,\ 5110{,}071,\ 182{,}0174,\ 295{,}5985,\ 217{,}9799,\ 286{,}4162,\ 395{,}5979),$$

przy czym $G2(\bar{X}^*) = 7049{,}330923$. Wszystkich sześć ograniczeń jest aktywnych w optimum.

3. Zadanie ([186]) polega na minimalizacji funkcji

$$G3(\bar{X}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,$$

przy ograniczeniach

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geqslant 0,$$

$$282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geqslant 0,$$

$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geqslant 0,$$

$$-4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geqslant 0$$

$$-10{,}0 \leqslant x_i \leqslant 10{,}0, \quad i = 1,\ \ldots,\ 7.$$

Zadanie ma 4 nieliniowe ograniczenia, funkcja $G3$ jest nieliniowa i ma globalne minimum w

$$(\bar{X}^*) = (2{,}330499,\ 1{,}951372,\ -0{,}4775414,\ 4{,}365726,\ -0{,}6244870,\ 1{,}038131,\ 1{,}594227),$$

przy czym $G3(\bar{X}^*) = 680{,}6300573$. Dwa (z czterech) ograniczenia są aktywne w optimum globalnym (pierwsze i ostatnie).

4. Zadanie ([186]) polega na minimalizacji funkcji

$$G4(\bar{X}) = e^{x_1x_2x_3x_4x_5},$$

przy ograniczeniach

$x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10,$ $\quad x_2x_3 - 5x_4x_5 = 0,$ $\quad x_1^3 + x_2^3 = -1,$

$-2{,}3 \leqslant x_i \leqslant 2{,}3,$ $\quad i = 1,\ 2,$ $\quad -3{,}2 \leqslant x_i \leqslant 3{,}2,$ $\quad i = 3,\ 4,\ 5.$

Zadanie ma 3 nieliniowe ograniczenia. Funkcja $G4$ jest nieliniowa i ma globalne minimum w

$$\bar{X}^* = (-1{,}717143,\ 1{,}595709,\ 1{,}827247,\ -0{,}7636413,\ -0{,}7636450),$$

przy czym $G4(\bar{X}^*) = 0{,}0539498478$.

5. Zadanie ([186]) polega na minimalizacji funkcji

$$\begin{aligned} G5(\bar{X}) = {} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + \\ & + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + \\ & + (x_{10} - 7)^2 + 45, \end{aligned}$$

przy ograniczeniach

$$\begin{aligned} & 105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geqslant 0, \\ & -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geqslant 0, \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \geqslant 0, \\ & -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geqslant 0, \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geqslant 0, \\ & -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geqslant 0, \\ & 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geqslant 0, \\ & -0{,}5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geqslant 0, \\ & -10{,}0 \leqslant x_i \leqslant 10{,}0, \qquad i = 1, \ldots, 10. \end{aligned}$$

Zadanie ma 3 liniowe i 5 nieliniowych ograniczeń. Funkcja $G5$ jest kwadratowa i ma globalne minimum w

$$\begin{aligned} \bar{X}^* = {} & (2{,}171996,\ 2{,}363683,\ 8{,}773926,\ 5{,}095984,\ 0{,}9906548,\ 1{,}430574, \\ & 1{,}321644,\ 9{,}828726,\ 8{,}280092,\ 8{,}375927), \end{aligned}$$

przy czym $G5(\bar{X}^*) = 24{,}3062091$. Sześć (z ośmiu) ograniczeń jest aktywnych w optimum (oprócz ostatnich dwóch).

6. Zadanie ([220]) polega na maksymalizacji funkcji

$$G6(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|,$$

przy ograniczeniach

$$\prod_{i=1}^{n} x_i > 0{,}75, \quad \sum_{i=1}^{n} x_1 < 7{,}5n \text{ oraz } 0 < x_i < 10 \text{ dla } 1 \leqslant i \leqslant n.$$

Zadanie ma 2 nieliniowe ograniczenia. Funkcja $G6$ jest nieliniowa i jej globalne maksimum nie jest znane. Dobre rozwiązania (wyznaczone przez GENOCOP III, patrz rozdz. 7) są następujące. Dla $n = 20$:

$$\begin{aligned}\mathbf{x} = (&3{,}16311359,\ 3{,}13150430,\ 3{,}09515858,\ 3{,}06016588,\ 3{,}03103566,\\ &2{,}99158549,\ 2{,}95802593,\ 2{,}92285895,\ 0{,}48684388,\ 0{,}47732279,\\ &0{,}48044473,\ 0{,}48790911,\ 0{,}48450437,\ 0{,}44807032,\ 0{,}46877760,\\ &0{,}45648506,\ 0{,}44762608,\ 0{,}44913986,\ 0{,}44390863,\ 0{,}45149332)\end{aligned}$$

przy czym $G6(\mathbf{x}) = 0{,}80351067$. Dla $n = 50$:

$$\begin{aligned}\mathbf{x} = (&6{,}28006029,\ 3{,}16155291,\ 3{,}15453815,\ 3{,}14085174,\ 3{,}12882447,\\ &3{,}11211085,\ 3{,}10170507,\ 3{,}08703685,\ 3{,}07571769,\ 3{,}06122732,\\ &3{,}05010581,\ 3{,}03667951,\ 3{,}02333045,\ 3{,}00721049,\ 2{,}99492717,\\ &2{,}97988462,\ 2{,}96637058,\ 2{,}95589066,\ 2{,}94427204,\ 2{,}92796040,\\ &0{,}40970641,\ 2{,}90670991,\ 0{,}46131119,\ 0{,}48193336,\ 0{,}46776962,\\ &0{,}43887550,\ 0{,}45181099,\ 0{,}44652876,\ 0{,}43348753,\ 0{,}44577143,\\ &0{,}42379948,\ 0{,}45858049,\ 0{,}42931050,\ 0{,}42928645,\ 0{,}42943302,\\ &0{,}43294361,\ 0{,}42663351,\ 0{,}43437257,\ 0{,}42542559,\ 0{,}41594154,\\ &0{,}43248957,\ 0{,}39134723,\ 0{,}42628688,\ 0{,}42774364,\ 0{,}41886297,\\ &0{,}42107263,\ 0{,}41215360,\ 0{,}41809589,\ 0{,}41626775,\ 0{,}42316407)\end{aligned}$$

przy czym $G6(\mathbf{x}) = 0{,}83319378$.

7. Zadanie ([114]) polega na minimalizacji

$$G7(\mathbf{x}, y) = -10{,}5x_1 - 7{,}5x_2 - 3{,}5x_3 - 2{,}5x_4 - 1{,}5x_5 - 10y + \\ - 0{,}5 \sum_{i=1}^{5} x_i^2,$$

przy ograniczeniach

$$6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leqslant 6{,}5, \qquad 10x_1 + 10x_3 + y \leqslant 20,$$

$$0 \leqslant x_i \leqslant 1, \qquad 0 \leqslant y.$$

Globalnym rozwiązaniem jest $(\mathbf{x}^*, y^*) = (0, 1, 0, 1, 1, 20)$, przy czym $G7(\mathbf{x}^*, y^*) = -213$.

8. Zadanie ([186]) polega na minimalizacji

$$G8(\mathbf{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \ldots + x_{10}} \right),$$

przy ograniczeniach

$$x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2, \qquad x_4 + 2x_5 + x_6 + x_7 = 1,$$

$$x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1, \qquad x_i \geqslant 0{,}000001, \ (i = 1, ..., 10),$$

gdzie

$$c_1 = -6{,}089; \quad c_2 = -17{,}164; \quad c_3 = -34{,}054; \quad c_4 = -5{,}914;$$

$$c_5 = -24{,}721; \quad c_6 = -14{,}986; \quad c_7 = -24{,}100; \quad c_8 = -10{,}708;$$

$$c_9 = -26{,}662; \quad c_{10} = -22{,}179.$$

Najlepsze rozwiązanie wyznaczone przez GENOCOP (rozdz. 7) to

$$\mathbf{x}^* = (0{,}04034785, 0{,}15386976, 0{,}77497089, 0{,}00167479, 0{,}48468539, 0{,}00068965, 0{,}02826479, 0{,}01849179, 0{,}03849563, 0{,}10128126)$$

przy czym wartość funkcji celu jest w nim równa $-47{,}760765$.

9. Zadanie ([113]) polega na maksymalizacji

$$G9(\mathbf{x}) = \frac{3x_1 + x_2 - 2x_3 + 0{,}8}{2x_1 - x_2 + x_3} + \frac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3},$$

przy ograniczeniach

$$x_1 + x_2 - x_3 \leqslant 1, \qquad -x_1 + x_2 - x_3 \leqslant -1,$$

$$12x_1 + 5x_2 + 12x_3 \leqslant 34{,}8, \qquad 12x_1 + 12x_2 + 7x_3 \leqslant 29{,}1,$$

$$-6x_1 + x_2 + x_3 \leqslant -4{,}1, \qquad 0 \leqslant x_i, \ i = 1, 2, 3.$$

Rozwiązaniem globalnym jest $\mathbf{x}^* = (1, 0, 0)$, przy czym $G9(\mathbf{x}^*) = 2{,}471428$.

10. Zadanie ([114]) polega na minimalizacji

$$G10(\mathbf{x}) = x_1^{0{,}6} + x_2^{0{,}6} - 6x_1 - 4x_3 + 3x_4,$$

przy ograniczeniach

$$-3x_1 + x_2 - 3x_3 = 0, \qquad x_1 + 2x_3 \leqslant 4,$$

$$x_2 + 2x_4 \leqslant 4, \qquad x_1 \leqslant 3,$$

$$x_4 \leqslant 1, \qquad 0 \leqslant x_i, \ i = 1, 2, 3, 4.$$

Najlepsze znane rozwiązanie globalne to $\mathbf{x}^* = (4/3, 4, 0, 0)$, przy czym $G10(\mathbf{x}^*) = -4{,}5142$.

11. Zadanie ([114]) polega na minimalizacji

$$G11(x, \mathbf{y}) = 6{,}5x - 0{,}5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5,$$

przy ograniczeniach

$$x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leqslant 16,$$
$$-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leqslant -1,$$
$$2x + 0{,}5y_1 + 0{,}2y_2 - 3y_3 - y_4 - 4y_5 \leqslant 24,$$
$$0{,}2x + 2y_1 + 0{,}1y_2 - 4y_3 + 2y_4 + 2y_5 \leqslant 12,$$
$$-0{,}1x - 0{,}5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leqslant 3,$$
$$y_3 \leqslant 1,\ y_4 \leqslant 1 \text{ oraz } y_5 \leqslant 2,$$
$$x \geqslant 0,\ y_i \geqslant 0, \text{ dla } 1 \leqslant i \leqslant 5.$$

Rozwiązaniem globalnym jest $(x, \mathbf{y}^*) = (0, 6, 0, 1, 1, 0)$, przy czym $G11(x, \mathbf{y}^*) = -11$.

12. To zadanie utworzono z trzech różnych zadań ([186]) w następujący sposób. Zminimalizuj

$$G12(\mathbf{x}) = \begin{cases} f_1 = x_2 + 10^{-5}(x_2 - x_1)^2 - 1{,}0, & \text{jeśli } 0 \leqslant x_1 < 2 \\ f_2 = \dfrac{1}{27\sqrt{3}}((x_1 - 3)^2 - 9)x_2^3, & \text{jeśli } 2 \leqslant x_1 < 4 \\ f_3 = \dfrac{1}{3}(x_1 - 2)^3 + x_2 - \dfrac{11}{3}, & \text{jeśli } 4 \leqslant x_1 \leqslant 6 \end{cases}$$

przy ograniczeniach

$$x_1/\sqrt{3} - x_2 \geqslant 0,$$
$$-x_1 - \sqrt{3}\,x_2 + 6 \geqslant 0,$$
$$0 \leqslant x_1 \leqslant 6 \text{ oraz } x_2 \geqslant 0.$$

Funkcja $G12$ ma trzy globalne rozwiązania:

$$\mathbf{x}_1^* = (0, 0), \qquad \mathbf{x}_2^* = (3, \sqrt{3}) \qquad \text{oraz} \quad \mathbf{x}_3^* = (4, 0).$$

We wszystkich przypadkach $G12(\mathbf{x}_i^*) = -1, \quad i = 1, 2, 3.$

Dodatek D

Prawdopodobnie najlepszym sposobem prowadzenia zajęć z metod obliczeń ewolucyjnych jest zorganizowanie ich jako zajęcia laboratoryjne. Po kilku wstępnych ćwiczeniach (na przykład zaprogramowania prostego algorytmu genetycznego), które pozwolą studentowi zorientować się w podstawowych pojęciach metod ewolucyjnych (można to zrobić w kilku pierwszych tygodniach semestru), grupa będzie gotowa na zagadnienia bardziej zaawansowane.

Jest kilka możliwości ćwiczeń eksperymentalnych. Zapewne ten tekst wywoła więcej pytań niż odpowiedzi. W ostatnim rozdziale, zakończeniu, podano listę bieżących zagadnień naukowych. Można ich użyć do ułożenia kilku ćwiczeń. Oczywiście ćwiczenia mogą się różnić złożonością i czasem potrzebnym do ich wykonania. Niektóre będą całkiem proste, inne mogą wymagać grupy kilku studentów pracujących wspólnie. W każdym bądź razie jest ważne, aby pamiętać, że wybór ćwiczeń jest arbitralny i może (a nawet musi) wyzwalać nowe pomysły.

Tak więc w tym dodatku podajemy wykaz różnych ćwiczeń wraz z kilkoma uwagami o opisie eksperymentów komputerowych. Często się zdarza, że niektóre wyniki są na tyle interesujące, że warto je opublikować!

Kilka możliwych ćwiczeń

1. Porównać na kilku funkcjach testowych działanie niektórych algorytmów: wzrostu, stochastycznego wzrostu, symulowanego wyżarzania, algorytmów genetycznych, strategii ewolucyjnych.
2. Porównać na kilku funkcjach testowych działanie metod algorytmów genetycznych z różnymi selekcjami (proporcjonalną, z uporządkowaniem, turniejową).
3. Porównać na kilku funkcjach testowych 3 wersje algorytmów genetycznych z różnymi kodowaniami chromosomów (binarnym, Graya, z liczbami całkowitoliczbowymi).

4. Dla typowego zadania z ograniczeniami (na przykład zadania upakowania) przeprowadzić obliczenia z różnymi metodami uwzględniania ograniczeń (dekodery, algorytmy naprawy, funkcje kary itp.).
5. Porównać różne operatory dla reprezentacji zmiennopozycyjnych (na przykład mutację gaussowską z mutacją niejednorodną, krzyżowanie heurystyczne itp.).
6. Zmodyfikować prosty program z dodatku A tak, aby lepiej nadawał się do rozwiązywania zadań optymalizacji numerycznej. Można to zrobić na wiele sposobów:

   (a) lepiej zorganizować plik wejściowy, aby użytkownik mógł podawać dane dla wielu zmiennych, a także inne parametry metody;
   (b) zmodyfikować system, aby obejmował zadania minimalizacji;
   (c) zmodyfikować system, aby uwzględniał wszystkie wartości funkcji celu (nie tylko dodatnie);
   (d) zamienić mutację równomierną na gaussowską (do przybliżenia liczby losowej Q o rozkładzie normalnym z wartością oczekiwaną μ i wariancją σ^2 można wygenerować 12 liczb losowych R_i, $i = 1, 2, ..., 12$, z zakresu [0, 1] z rozkładem równomiernym, wtedy

   $$Q = \mu + \sigma\left(\sum_{i=1}^{12} R_i - 6\right);$$

   (e) zamienić krzyżowanie jednopunktowe na arytmetyczne;
   (f) wprowadzić dodatkowe operatory, w tym krzyżowanie z wieloma rodzicami (na przykład obliczając „środek" dla kilku rodziców jako średnie z ich współrzędnych i przesuwając najsłabszego osobnika ku środkowi);
   (g) wprowadzić zmienne różnego typu (logiczne, całkowite);
   (h) itd.

7. Porównać strategie ewolucyjne i algorytmy genetyczne z reprezentacją zmiennopozycyjną oraz odpowiednimi operatorami.
8. Opracować mechanizm adaptacyjny dla parametrów jednej z metod ewolucyjnych (prawdopodobieństw użycia operatorów, liczebności populacji, zakresu mutacji, typu krzyżowania itp.) i przeprowadzić obliczenia.
9. Zaopatrzyć się w jedno z obecnie bezpłatnie dostępnych oprogramowań (jak na przykład GENOCOP) i zaadaptować je do zmiennych całkowitoliczbowych i logicznych. Przeprowadzić obliczenia dla zadań programowania całkowitoliczbowego.
10. Wprowadzić niestandardowe cechy do algorytmów genetycznych (płeć, powiązania rodzinne, współpracę między osobnikami, pary rodzące wieloraczki) i przeprowadzić dla nich obliczenia. Opracować uzasadnienie dla takich dodatkowych heurystyk i sprawdzić je na kilku zadaniach testowych. Należy wziąć pod uwagę zyski i straty wynikające z możliwej poprawy działania oraz większego skomplikowania systemu.

11. Rozważyć zadanie z funkcją celu zmienną w czasie. Przeprowadzić eksperymenty obliczeniowe porównujące haploidalne i diploidalne struktury chromosomów.
12. Opracować grafikę dla pewnego systemu ewolucyjnego do wykreślania bieżących danych statystycznych o stanie przeszukiwania.
13. Wziąć jakiekolwiek nietrywialne zadanie optymalizacyjne ze znanej Ci dziedziny (bazy danych, badania operacyjne, przetwarzanie obrazów, projektowanie inżynierskie, sterowanie rozmyte, sztuczne sieci neuronowe, gry, robotyka itp.). Opracować program ewolucyjny dla tej klasy zadań. Porównać go z innymi znanymi metodami rozwiązywania tego typu zadań (patrz następny punkt).

Kilka uwag o opisywaniu eksperymentów z metodami heurystycznymi

Wiele metod ewolucyjnych ocenia się na podstawie obliczeń przeprowadzonych na kilku zadaniach testowych (takich, jak wymienione w dodatkach B i C). Bardzo często jest trudno wyprowadzić na podstawie wyników z tych obliczeń uogólnienia prowadzące do pewnych „globalnych" stwierdzeń o konkretnej metodzie. Można jednak wykazać w ten sposób użyteczność nowej metody, porównując ją na kilku starannie wybranych przypadkach z inną, dobrze ugruntowaną metodą. Zgodnie z [26], użyteczność nowej metody heurystycznej może polegać na:

- tworzeniu wysokiej jakości rozwiązań szybciej niż inne metody,
- wyznaczaniu rozwiązań o lepszej jakości niż inne metody,
- mniejszej wrażliwości na zmiany w cechach zadania, na jakość danych lub zmiany dostrajanych parametrów niż inne metody,
- łatwości użycia,
- możliwości zastosowania w szerszej klasie zadań.

Dodatkowo [26]:

> opracowania badawcze na temat heurystyk są wartościowe, jeżeli są one *odkrywcze* – umożliwiające zrozumienie ogólnego tworzenia heurystyk lub struktury zadania przez podanie przyczyn uzyskania takich, a nie innych wyników z algorytmu, oraz wyjaśnienie jego zachowania się, lub *teoretyczne* – podające uzasadnienia teoretyczne, na przykład dla ograniczeń na jakość rozwiązania.

Artykuł Barra i in. [26] podaje bardzo dobry przegląd zagadnień projektowania i opisywania wyników numerycznych dla metod heurystycznych. Na przykład przygotowując i opisując wyniki, powinno się postępować według następujących pięciu punktów (wymienionych w [26]):

- określić cele eksperymentu,
- wybrać sposoby mierzenia wyników i badanych czynników,

- zaprojektować i wykonać obliczenia,
- przeanalizować wyniki i wyciągnąć wnioski,
- opisać wyniki eksperymentu.

Jest ważne, aby uwzględnić wszystkie te punkty. Na przykład cel eksperymentu może być różny, jak podano w [26]:

> Eksperymenty obliczeniowe dla algorytmów zazwyczaj podejmuje się, aby: (a) porównać działanie różnych algorytmów na tej samej klasie zadań lub (b) scharakteryzować albo opisać oddzielnie działanie algorytmu. Chociaż te cele nieco się zazębiają, badacz powinien określić, co właściwie chce osiągnąć za pomocą testowania (na przykład, na jakie pytania chce odpowiedzieć, jakie hipotezy sprawdzić).

Za miarę jakości działania można przyjąć na przykład jakość najlepszego wyznaczonego rozwiązania lub czas jego wyznaczenia, lub czas, w jakim algorytm osiągnie rozwiązanie „akceptowalne", lub też stabilność metody. W większości przypadków podstawową sprawą jest porównanie nowej metody z ustalonymi metodami dla danej klasy zadań. Ważne też jest, aby pamiętać o przeanalizowaniu kluczowych czynników (jak wpływu rozmiaru zadania na jakość uzyskanego rozwiązania i nakłady obliczeniowe). Końcowe opracowanie powinno też zawierać wszystkie informacje pozwalające czytelnikowi na odtworzenie wyników.

Jest wiele bibliotek ze standardowymi zadaniami testowymi w *World Wide Web*. Powinny one być często używane do badań eksperymentalnych (na przykład zadania testowe z badań operacyjnych można znaleźć w OR-Library (o.rlibrary@ic.ac.uk), jest ona dostępna przez ftp w mscmga.ms.ic.ac.uk lub przez WWW w http://mscmga.ma.ac.uk).

Literatura

[1] Aarts E.H.L., Korst J.: *Simulated Annealing and Boltzmann Machines.* John Wiley, Chichester, UK, 1989.

[2] Aarts E.H.L., Lenstra J.K. (Editors): *Local Search in Combinatorial Optimization.* John Wiley, Chichester, UK, 1995.

[3] Abuali F.N., Wainwright R.L., Schoenefeld D.A.: *Determinant Factorization: A New Encoding Scheme for Spanning Trees Applied to the Probabilistic Minimum Spanning Tree Problem,* w [103], s. 470-477.

[4] Ackley D.H.: *An Empirical Study of Bit Vector Function Optimization,* w [73], s. 170-204.

[5] Adler D.: *Genetic Algorithms and Simulated Annealing: A Marriage Proposal.* Proceedings of the IEEE International Conference on Neural Networks, March 28-April 1, 1993, s. 1104-1109.

[6] Akl S.G.: *The Design and Analysis of Parallel Algorithms.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[7] Alander J.T.: *Proceedings of the First Finnish Workshop on Genetic Algorithms and their Applications.* Helsinki University of Technology, Finland, November 4-5, 1992.

[8] Angeline P.J., Kinnear K.E. (Editors): *Advances in Genetic Programming II.* MIT Press, Cambridge, MA, 1996.

[9] Antonisse H.J.: *A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint,* w [344], s. 86-91.

[10] Antonisse H.J., Keller K.S.: *Genetic Operators for High Level Knowledge Representation,* w [171], s. 69-76.

[11] Arabas J., Mulawka J., Pokraśniewicz J.: *A New Class of the Crossover Operators for the Numerical Optimization,* w [103], s. 42-48.

[12] Arabas J., Michalewicz Z., Mulawka J.: *GAVaPS – a Genetic Algorithm with Varying Population Size,* w [275], s. 73-78.

[13] Attia N.F.: *New Methods of Constrained Optimization using Penalty Functions.* Ph.D. Thesis, Essex University, England, 1985.

[14] Axelrod R.: *Genetic Algorithm for the Prisoner Dilemma Problem,* w [73], s. 32-41.

[15] Bäck T.: *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1995.

[16] Bäck T., Hoffmeister F.: *Extended Selection Mechanisms in Genetic Algorithms,* w [32], s. 92-99.

[17] Bäck T., Fogel D.B., Michalewicz Z.: *Handbook of Evolutionary Computation.* University Oxford Press, New York, 1996.

[18] Bäck T., Hoffmeister F., Schwefel H.-P.: *A Survey of Evolution Strategies*, w [32], s. 2-9.
[19] Bäck T., Rudolph G., Schwefel H.-P.: *Evolutionary Programming and Evolution Strategies: Similarities and Differences*, w [124], s. 11-22.
[20] Bäck T., Schwefel H.-P.: *An Overview of Evolutionary Algorithms for Parameter Optimization*. Evolutionary Computation, Vol.1, No.1, s. 1-23, 1993.
[21] Bagchi S., Uckun S., Miyabe Y., Kawamura K.: *Exploring Problem-Specific Recombination Operators for Job Shop Scheduling*, w [32], s. 10-17.
[22] Baker J.E.: *Adaptive Selection Methods for Genetic Algorithms*, w [167], s. 101-111.
[23] Baker J.E.: *Reducing Bias and Inefficiency in the Selection Algorithm*, w [171], s. 14-21.
[24] Bala J., De Jong K.A., Pachowicz P.: *Using Genetic Algorithms to Improve the Performance of Classification Rules Produced by Symbolic Inductive Method*, Proceedings of the 6th International Symposium on Methodologies of Intelligent Systems, Lecture Notes in Artificial Intelligence, Vol.542, s. 286-295, Springer-Verlag, 1991.
[25] Banach S.: *Sur les opérations dans les ensembles abstraits et leurs applications aux équations intégrales*. Fundamenta Mathematica, Vol.3, s. 133-181, 1922.
[26] Barr R.S., Golden B.L., Kelly J.P., Resende M.G.C., Stewart W.R.: *Designing and Reporting on Computational Experiments with Heuristic Methods*. Proceedings of the International Conference on Metaheuristics for Optimization, Kluwer Publishing, s. 1-17, 1995.
[27] Bean J.C., Hadj-Alouane A.B.: *A Dual Genetic Algorithm for Bounded Integer Programs*. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53, 1992.
[28] Beasley D., Bull D.R., Martin R.R.: *An Overview of Genetic Algorithms: Part 1, Foundations*. University Computing, Vol.15, No.2, s. 58-69, 1993.
[29] Beasley D., Bull D.R., Martin R.R.: *An Overview of Genetic Algorithms: Part 2, Research Topics*. University Computing, Vol.15, No.4, s. 170-181, 1993.
[30] Beasley D., Bull D.R., Martin R.R.: *A Sequential Niche Technique for Multimodal Function Optimization*. Evolutionary Computation, Vol.1, No.2, s. 101-125, 1993.
[31] Beasley J.E., Chu P.C.: *A Genetic Algorithm for the Set Covering Problem*. Technical Report, The Management School, Imperial College, July 1994.
[32] Belew R., Booker L. (Editors): *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann Publishers, San Mateo, CA, 1991.
[33] Bellman R.: *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.
[34] Bennett K., Ferris M.C., Ioannidis Y.E.: *A Genetic Algorithm for Database Query Optimization*, w [32], s. 400-407.
[35] Bersini H., Varela F.J.: *The Immune Recruitment Mechanism: A Selective Evolutionary Strategy*, w [32], s. 520-526.
[36] Bertoni A., Dorigo M.: *Implicit Parallelism in genetic Algorithms*. Artificial Intelligence, Vol.61, no.2, s. 307-314, 1993.
[37] Bertsekas D.P.: *Dynamic Programming. Deterministic and Stochastic Models*. Prentice-Hall, Englewood Cliffs, NJ, 1987.
[38] Bethke A.D.: *Genetic Algorithms as Function Optimizers*. Doctoral Dissertation, University of Michigan, 1980.
[39] Betts J.T.: *An Accelarated Multiplier Method for Nonlinear Programming*, Journal of Optimization Theory and Applications, Vol.21, No.2, s. 137-174, 1977.
[40] Biggs M.C.: *Constrained Minimization Using Recursive Quadratic Programming: Some Alternative Subproblem Formulations*. Towards Global Optimization (Eds. L.C.W. Dixon and G.P. Szego), North-Holland, 1975.
[41] Biggs M.C.: *A Numerical Comparison Between Two Approaches to the Nonlinear Programming Problem*. Numerical Optimization Centre Technical Report No. 77, The Hatfield Polytechnic, 1976.
[42] Bilchev G., Parmee I.C.: *Ant Colony Search vs. Genetic Algorithms*. Technical Report, Plymouth Engineering Design Centre, University of Plymouth, 1995.
[43] Bilchev G.: Informacja ustna, April 1995.

[44] Bland R.G., Shallcross D.F.: *Large Traveling Salesman Problems Arising From Experiments in X-Ray Crystallography: A Preliminary Report on Computation*, Operations Research Letters, Vol.8, s. 125-128, 1989.

[45] Booker L.B.: *Intelligent Behavior as an Adaptation to the Task Environment*, Doctoral Dissertation, University of Michigan, 1982.

[46] Booker L.B.: *Improving Search in Genetic Algorithms*, w [73], s. 61-73.

[47] Bosworth J., Foo N., Zeigler B.P.: *Comparison of Genetic Algorithms with Conjugate Gradient Methods*. Washington, DC, NASA (CR-2093), 1972.

[48] Bowen J., Dozier G.: *Solving Constraint Satisfaction Problems Using a Genetic/Systematic Search Hybrid that Realizes when to Quit*, w [103], s. 122-129.

[49] Brindle A.: *Genetic Algorithms for Function Optimization*. Doctoral Dissertation, University of Alberta, Edmonton, 1981.

[50] Brooke A., Kendrick D., Meeraus A.: *GAMS: A User's Guide*, The Scientific Press, 1988.

[51] Broyden C.G., Attia N.F.: *A Smooth Sequential Penalty Function Method for Solving Nonlinear Programming Problem*, Lecture Notes in Control and Information Sciences, Vol.59, System Modelling and Optimization, Proceedings of the 11th IFIP Conference, July 1983. Springer-Verlag, 1983.

[52] Broyden C.G., Attia N.F.: *Penalty Functions, Newton's Method, and Quadratic Programming*. Journal of Optimization Theory and Applications, Vol.58, No.3., 1988.

[53] Bruns R.: *Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling*, w [129], s. 352-359.

[54] Bruns R.: *Scheduling*: w [17], section F1.5.

[55] Bui T.N., Eppley P.H.: *A Hybrid Genetic Algorithm for the Maximum Clique Problem*, w [103], s. 478-483.

[56] Bui T.N., Moon B.R.: *On Multi-Dimensional Encoding/Crossover*, w [103], s. 49-56.

[57] Burke E.K., Elliman D.G., Weare R.F.: *A Hybrid Genetic Algorithm for Highly Constrained Timetabling Problems*, w [103], s. 605-610.

[58] Carey M.R., Johnson D.S.: *Computers and Intractibility – A Guide to the Theory of NP--Completeness*. W.H. Freeman, San Francisco, 1979.

[59] Cartwright H.M., Mott G.F.: *Looking Around: Using Clues from the Data Space to Guide Genetic Algorithm Searches*, w [32], s. 108-114.

[60] Cavaretta M.J.: *Using Cultural Algorithm to Control Genetic Operators*, w [378], s. 158-166.

[61] Četverikov S.S.: On Some Aspects of the Evolutionary Process From the Viewpoint of Modern Genetics (in Russian). *Journal Exper. Biol.*, Vol.2, No.1, s. 3-54, 1926.

[62] Cleveland G.A., Smith S.F.: *Using Genetic Algorithms to Schedule Flow Shop Releases*, w [344], s. 160-169.

[63] Colorni A., Dorigo M., Maniezzo V.: *Genetic Algorithms and Highly Constrained Problems: The Time-Table Case*, w [351], s. 55-59.

[64] Colville A.R.: *A Comparative Study on Nonlinear Programming Codes*. IBM Scientific Center Report 320-2949, New York, 1968.

[65] Coombs S., Davis L.: *Genetic Algorithms and Communication Link Speed Design: Theoretical Considerations*, w [171], s. 252-256.

[66] Cooper L., Steinberg D.: *Introduction to Methods of Optimization*. W.B. Saunders, London, 1970.

[67] Cormier D.R.: *Pluto: A General Purpose Evolution Programming Shell*. Technical Report, August 17, 1993, Department of Industrial Engineering, North Carolina State University.

[68] Craighurst R., Martin W.: *Enhancing GA Performance through Crossover Prohibitions Based on Ancestry*, w [103], s. 130-135.

[69] Davidor Y.: *Genetic Algorithms and Robotics*. World Scientific, 1991.

[70] Davidor Y., Schwefel H.-P., Männer R. (Editors): Proceedings of the Third International Conference on Parallel Problem Solving from Nature (PPSN), Lecture Notes in Computer Science, Vol. 866, Springer-Verlag, New York, 1994.

[71] Davis L.: *Applying Adaptive Algorithms to Epistatic Domains*. Proceedings of the International Joint Conference on Artificial Intelligence, s. 162-164, 1985.

[72] Davis L.: *Job Shop Scheduling with Genetic Algorithms*, w [167], s. 136-140.
[73] Davis L. (Editor): *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, San Mateo, CA, 1987.
[74] Davis L., Steenstrup M.: *Genetic Algorithms and Simulated Annealing: An Overview*, w [73], s. 1-11.
[75] Davis L., Ritter F.: *Schedule Optimization with Probabilistic Search*. Proceedings of the Third Conference on Artificial Intelligence Applications, Computer Society Press, s. 231--236, 1987.
[76] Davis L., Coombs S.: *Genetic Algorithms and Communication Link Speed Design: Constraints and Operators*, w [171], s. 257-260.
[77] Davis L.: *Adapting Operator Probabilities in Genetic Algorithms*, w [344], s. 61-69.
[78] Davis L. (Editor): *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.
[79] Davis L.: *Bit-Climbing, Representational Bias, and Test Suite Design*, w [32], s. 18-23.
[80] Davis T.E., Principe J.C.: *A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm*, w [32], s. 174-181.
[81] Dawkins R.: *The Selfish Gene*. Oxford University Press, New York, 1976.
[82] De Jong K.A.: *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, (Doctoral dissertation, University of Michigan), Dissertation Abstract International, 36(10), 5140B. (University Microfilms No 76-9381).
[83] De Jong K.A.: *Adaptive System Design: A Genetic Approach*. IEEE Transactions on Systems, Man, and Cybernetics, Vol.10, No.3, s. 556-574, 1980.
[84] De Jong K.A.: *Genetic Algorithms: A 10 Year Perspective*, w [167], s. 169-177.
[85] De Jong K.A.: *On Using Genetic Algorithms to Search Program Spaces*, w [171], s. 210-216.
[86] De Jong K.A.: *Learning with Genetic Algorithm: An Overview, Machine Learning*. Vol.3, s. 121-138, 1988.
[87] De Jong K.A. (Editor): *Evolutionary Computation*. MIT Press, 1993.
[88] De Jong K.A.: *Genetic-Algorithm-Based Learning*, w [226], s. 611-638.
[89] De Jong K.: *Genetic Algorithms: A 25 Year Perspective*, w [415], s. 125-134.
[90] De Jong K.A., Spears W.M.: *Using Genetic Algorithms to Solve NP-Complete Problems*, w [344], s. 124-132.
[91] De Jong K.A., Spears W.M.: *Using Genetic Algorithms for Supervised Concept Learning*. Proceedings of the Second International Conference on Tools for AI, s. 335-341, 1990.
[92] De La Maza M., Yuret D.: *Dynamic Hill Climbing*. AI Expert, March 1994, s. 26-31.
[93] Dhar V., Ranganathan N.: *Integer Programming vs. Expert Systems: An Experimental Comparison*. Communications of the ACM, Vol.33, No.3, s. 323-336, 1990.
[94] Dixmier J.: *General Topology*. Springer-Verlag, 1984.
[95] Dozier G., Bahler D., Bowen J.: *Solving Small and Large Scale Constraint Satisfaction Problems Using a Heuristic-Based Microgenetic Algorithm*, w [275], s. 306-311.
[96] Eades P., Lin X.: *How to Draw a Directed Graph*, Technical Report, Department of Computer Science, University of Queensland, Australia, 1989.
[97] Eades P., Tamassia R.: *Algorithms for Drawing Graphs: An Annotated Bibliography*. Technical Report No. CS-89-09, Brown University, Department of Computer Science, October, 1989.
[98] Eiben A.E., Aarts E.H.L., Van Hee K.M.: *Global Convergence of Genetic Algorithms: On Infinite Markov Chain Analysis*, w [351], s. 4-12.
[99] Eiben A.E., Raue P.-E., Ruttkay Zs.: *Solving Constraint Satisfaction Problems Using Genetic Algorithms*, w [276], s. 542-547.
[100] Eiben A.E., Raue P.-E., Ruttkay Zs.: *Genetic Algorithms with Multi-parent Recombination*, w [70], s. 78-87.
[101] Esbensen H.: *Finding (Near-)Optimal Steiner Trees in Large Graphs*, w [103], s. 485-491.
[102] Eshelman L.J.: *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*, w [318], s. 265-283.
[103] Eshelman L.J. (Editor): *Proceedings of the Sixth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1995.

[104] Eshelman L.J., Caruana R.A., Schaffer J.D.: *Biases in the Crossover Landscape*, w [344], s. 10-19.

[105] Eshelman L.J., Schaffer J.D.: *Preventing Premature Convergence in Genetic Algorithms by Preventing Incest*, w [32], s. 115-122.

[106] Even S., Itai A., Shamir A.: *On the Complexity of Timetable and Multicommodity Flow Problems*. SIAM Journal on Computing, Vol.5, No.4, 1976, s. 691-703.

[107] *Evolutionary Computation*. Vol.2, No.1, Spring 1994; specjalny numer na temat systemów klasyfikujących.

[108] Falkenauer E.: *The Grouping Genetic Algorithms – Widening the Scope of the GAs*. Belgian Journal of Operations Research, Statistics and Computer Science, Vol.33, 1993, s. 79-102.

[109] Falkenauer E.: *A New Representation and Operators for GAs Applied to Grouping Problems*. Evolutionary Computation, Vol.2, No.2, s. 123-144.

[110] Falkenauer E.: *Solving Equal Piles with a Grouping Genetic Algorithm*, w [103], s. 492-497.

[111] Fiacco A.V., McCormick G.P.: *Nonlinear Programming*. John Wiley, Chichester, UK, 1968.

[112] Fletcher R.: *Practical Methods of Optimization*, Vol.2, *of Constrained Optimization*. John Wiley, Chichester, UK, 1981.

[113] Floudas C.A., Pardalos P.M.: *A Collection of Test Problems for Constrained Global Optimization Algorithms*. Lecture Notes in Computer Science, Vol.455, Springer-Verlag, 1987.

[114] Floudas C.A., Pardalos P.M.: *Recent Advances in Global Optimization*. Princeton Series in Computer Science, Princeton University Press, Princeton, NJ, 1992.

[115] Fogarty T.C.: *Varying the Probability of Mutation in the Genetic Algorithm*, w [344], s. 104-109.

[116] Fogel D.B.: *An Evolutionary Approach to the Traveling Salesman Problem*. Biol. Cybern., Vol.60, s. 139-144, 1988.

[117] Fogel D.B.: *Evolving Artificial Intelligence*. PhD Thesis, University of California, San Diego, 1992.

[118] Fogel D.B. (Editor): IEEE Transactions on Neural Networks, special issue on Evolutionary Computation, Vol.5, No.1, 1994.

[119] Fogel D.B.: *An Introduction to Simulated Evolutionary Optimization*. IEEE Transactions on Neural Networks, special issue on EP, Vol.5, No.1, 1994.

[120] Fogel D.B.: *Evolving Behaviours in the Iterated Prisoner's Dilemma*. Evolutionary Computation, Vol.1, No.1, s. 77-97, 1993.

[121] Fogel D.B.: *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press, Piscataway, NJ, 1995.

[122] Fogel D.B., Atmar J.W.: *Comparing Genetic Operators with Gaussian Mutation in Simulated Evolutionary Process Using Linear Systems*. Biol. Cybern., Vol.63, s. 111-114, 1990.

[123] Fogel D.B., Atmar W.: *Proceedings of the First Annual Conference on Evolutionary Programming*. La Jolla, CA, 1992, Evolutionary Programming Society.

[124] Fogel D.B., Atmar W.: *Proceedings of the Second Annual Conference on Evolutionary Programming*, La Jolla, CA, 1993, Evolutionary Programming Society.

[125] Fogel L.J.: *Evolutionary Programming in Perspective: The Top-Down View*, w [415], s. 135--146.

[126] Fogel L.J., Owens A.J., Walsh M.J.: *Artificial Intelligence Through Simulated Evolution*. John Wiley, Chichester, UK, 1966.

[127] Fonseca C.M., Fleming P.J.: *An Overview of Evolutionary Algorithms in Multiobjective Optimization*. Evolutionary Computation, Vol.3, No.1, 1995, s. 165-180.

[128] Forrest S.: *Implementing Semantic Networks Structures Using the Classifier System*, w [167], s. 24-44.

[129] Forrest S. (Editor): *Proceedings of the Fifth International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA, 1993.

[130] Foux G., Heymann M., Bruckstein A.: *Two-Dimensional Robot Navigation Among Unknown Stationary Polygonal Obstacles*. IEEE Transactions on Robotics and Automation, Vol.9, s. 96-102, 1993.

[131] Fox B.R., McMahon M.B.: *Genetic Operators for Sequencing Problems*, w [318], s. 284-300.
[132] Fox M.S.: *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann Publishers, San Mateo, CA, 1987.
[133] Freville A., Plateau G.: *Heuristics and Reduction Methods for Multiple Constraint 0-1 Linear Programming Problems*. European Journal of Operational Research, Vol. 24, s. 206-215.
[134] Garey M., Johnson D.: *Computers and Intractability*. W.H. Freeman, San Francisco, 1979.
[135] Gen M., Wasserman G.S., Smith A.E. (Guest Editors): *Computers and Industrial Engineering Journal*, special issue on genetic algorithms and industrial engineering, Vol.30, No.2, 1996.
[136] Gendreau M., Hertz A., Laporte G.: *A Tabu Search Heuristic for Vehicle Routing*. CRT--7777, Centre de Recherche sur les transports, Universite de Montreal, 1991.
[137] Gill P.E., Murray W., Wright M.H.: *Practical Optimization*. Academic Press, London, 1978.
[138] Gillies A.M.: *Machine Learning Procedures for Generating Image Domain Feature Detectors*. Doctoral Dissertation, University of Michigan, 1985.
[139] Giordana A., Saitta L.: *REGAL: An Integrated System for Learning Relations Using Genetic Algorithms*. Proceedings of the International Workshop on Multistrategy Learning, MSL--93, Harpers Ferry, VA, s. 234-249, 1993.
[140] Giordana A., Saitta L., Zini F.: *Learning Disjunctive Concepts with Distributed Genetic Algorithms*, w [275], s. 115-119.
[141] Glover D.E.: *Solving a Complex Keyboard Configuration Problem Through Generalized Adaptive Search*, w [73], s. 12-27.
[142] Glover F.: *Heuristics for Integer Programming Using Surrogate Constraints*. Decision Sciences, Vol.8, No.1, s. 156-166, 1977.
[143] Glover F.: *Tabu Search – Part I*. ORSA Journal on Computing, Vol.1, No.3, s. 190-206, 1989.
[144] Glover F.: *Tabu Search – Part II*. ORSA Journal on Computing, Vol.2, No.1, s. 4-32, 1990.
[145] Glover F.: *Tabu Search for Nonlinear and Parametric Optimization*. Technical Report, Graduate School of Business, University of Colorado at Boulder; wstępna wersja przedstawiona na EPFL Seminar on OR and AI Search Methods for Optimization Problems, Lausanne, Switzerland, November 1990.
[146] Glover F.: *Genetic Algorithms and Scatter Search: Unsuspected Potentials, Statistics and Computing*. Vol.4, s. 131-140.
[147] Glover F.: *Tabu Search Fundamentals and Uses*. Graduate School of Business, University of Colorado, 1995.
[148] Glover F., Kochenberger G.: *Critical Event Tabu Search for Multidimensional Knapsack Problems*. Proceedings of the International Conference on Metaheuristics for Optimization, Kluwer Publishing, s. 113-133, 1995.
[149] Goldberg D.E.: *Genetic Algorithm and Rule Learning in Dynamic Control System*, w [167], s. 8-15.
[150] Goldberg D.E.: *Dynamic System Control Using Rule Learning and Genetic Algorithms*, w Proceedings of the International Joint Conference on Artificial Intelligence, 9, s. 588-592, 1985.
[151] Goldberg D.E.: *Optimal Initial Population Size for Binary-Coded Genetic Algorithms*. TCGA Report No.85001, Tuscaloosa, University of Alabama, 1985.
[152] Goldberg D.E.: *Simple Genetic Algorithma and the Minimal, Deceptive Problem*, w [73], s. 74-88.
[153] Goldberg D.E.: *Sizing Populations for Serial and Parallel Genetic Algorithms*, w [344], s. 70-79.
[154] Goldberg D.E.: *Algorytmy genetyczne i ich zastosowania*. WNT, Warszawa 1995.
[155] Goldberg D.E.: *Messy Genetic Algorithms: Motivation, Analysis, and First Results*. Complex Systems, Vol.3, s. 493-530, 1989.
[156] Goldberg D.E.: *Zen and the Art of Genetic Algorithms*, w [344], s. 80-85.
[157] Goldberg D.E.: *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking*. University of Illinois at Urbana-Champaign, Technical Report No. 90001, September 1990.

[158] Goldberg D.E.: *Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale*. Complex Systems, Vol.4, s. 415-444, 1990.
[159] Goldberg D.E., Deb K., Korb B.: *Do not Worry, Be Messy*, w [32], s. 24-30.
[160] Goldberg D.E., Lingle R.: *Alleles, Loci, and the TSP*, w [167], s. 154-159.
[161] Goldberg D.E., Milman K., Tidd C.: *Genetic Algorithms: A Bibliography*. IlliGAL Technical Report 92008, 1992.
[162] Goldberg D.E., Richardson J.: *Genetic Algorithms with Sharing for Multimodal Function Optimization*, w [171], s. 41-49.
[163] Goldberg D.E., Segrest P.: *Finite Markov Chain Analysis of Genetic Algorithms*, w [171], s. 1-8.
[164] Gorges-Schleuter M.: *ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy*, w [344], s. 422-427.
[165] Greene F.: *A Method for Utilizing Diploid/Dominance in Genetic Search*, w [275], s. 439-444.
[166] Grefenstette J.J.: *GENESIS: A System for Using Genetic Search Procedures*. Proceedings of the 1984 Conference on Intelligent Systems and Machines, s. 161-165.
[167] Grefenstette J.J. (Editor): Proceedings of the First International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ, 1985.
[168] Grefenstette J.J., Gopal R., Rosmaita B., Van Gucht D.: *Genetic Algorithm for the TSP*, w [167], s. 160-168.
[169] Grefenstette J.J.: *Optimization of Control Parameters for Genetic Algorithms*. IEEE Transactions on Systems, Man, and Cybernetics, Vol. 16, No.1, s. 122-128, 1986.
[170] Grefenstette J.J.: *Incorporating Problem Specific Knowledge into Genetic Algorithms*, w [73], s. 42-60.
[171] Grefenstette J.J. (Editor): Proceedings of the Second International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.
[172] Gregory J.W.: *Nonlinear Programming FAQ*. 1995, Usenet sci.answers; dostępne przez anonymous ftp z rtfm.mit.edu w /pub/usenet/sci.answers/nonlinear-programming-faq.
[173] Groves L., Michalewicz Z., Elia P., Janikow C.: *Genetic Algorithms for Drawing Directed Graphs*. Proceedings of the Fifth International Symposium on Methodologies of Intelligent Systems, North-Holland, Amsterdam, s. 268-276, 1990.
[174] Hadj-Alouane A.B., Bean J.C.: *A Genetic Algorithm for the Multiple-Choice Integer Program*. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-50, 1992.
[175] Han S.P.: *Superlinearly Convergent Variable Metric Algorithm for General Nonlinear Programming Problems*. Mathematical Programming, Vol.11, s. 263-282, 1976.
[176] Handley S.: *The Genetic Planner: The Automatic Generation of Plans for a Mobile Robot via Genetic Programming*. Proceedings of 8th IEEE International Symposium on Intelligent Control, August 25-27, 1993.
[177] Heitkötter J. (Editor): *The Hitch-Hiker's Guide to Evolutionary Computation*. FAQ w comp.ai.genetic, issue 1.10, 20 December 1993.
[178] Herdy M.: *Application of the Evolution Strategy to Discrete Optimization Problems*, w [351], s. 188-192.
[179] Held M., Karp R.M.: *The Traveling Salesman Problem and Minimum Spanning Trees*. Operations Research, Vol.18, s. 1138-1162.
[180] Held M., Karp R.M.: *The Traveling Salesman Problem and Minimum Spanning Trees: Part II*. Mathematical Programming, Vol.1, s. 6-25.
[181] Hesser J., Männer R., Stucky O.: *Optimization of Steiner Trees Using Genetic Algorithms*, w [344], s. 231-236.
[182] Hillier F.S., Lieberman G.J.: *Introduction to Operations Research*. Holden-Day, San Francisco, CA, 1967.
[183] Hinterding R.: *Mapping, Order-independent Genes and the Knapsack Problem*, w [275], s. 13-17.
[184] Hinterding R., Gielewski H., Peachey T.C.: *The Nature of Mutation in Genetic Algorithms*, w [103], s. 65-72.

[185] Hobbs M.F.: *Genetic Algorithms, Annealing, and Dimension Alleles.* MSc. Thesis, Victoria University of Wellington, 1991.
[186] Hock W., Schittkowski K.: *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems, Vol.187, Springer-Verlag, 1981.
[187] Hoffmeister F., Bäck T.: *Genetic Algorithms and Evolution Strategies*, w [351], s. 455-470.
[188] Holland J.H.: *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.
[189] Holland J.H., Reitman J.S.: *Cognitive Systems Based on Adaptive Algorithms*, w D.A. Waterman, F. Hayes-Roth (Editors), *Pattern-Directed Inference Systems.* Academic Press, New York, 1978.
[190] Holland J.H.: *Properties of the Bucket Brigade*, w [167], s. 1-7.
[191] Holland J.H.: *Escaping Brittleness*, w R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Editors): *Machine Learning II.* Morgan Kaufmann Publishers, San Mateo, CA, 1986.
[192] Holland J.H.: *Royal Road Functions*, Genetic Algorithm Digest. Vol.7, No.22, 12 August 1993.
[193] Holland J.H., Holyoak K.J., Nisbett R.E., Thagard P.R.: *Induction.* MIT Press, Cambridge, MA, 1986.
[194] Homaifar A., Guan S.: *A New Approach on the Traveling Salesman Problem by Genetic Algorithm.* Technical Report, North Carolina A & T State University, 1991.
[195] Homaifar A., Lai S. H.-Y., Qi X.: *Constrained Optimization via Genetic Algorithms.* Simulation, Vol.62, 1994, s. 242-254.
[196] Horn J., Nafpliotis N.: *Multiobjective Optimization Using the Niched Pareto Genetic Algorithm.* Department of Computer Science, University of Illinois at Urbana-Champaign, IlliGAL Report 93005, 1993.
[197] Horowitz E., Sahni S.: *Fundamentals of Computer Algorithms.* Computer Science Press, Potomac, MD, 1978.
[198] Husbands P., Mill F., Warrington S.: *Genetic Algorithms, Production Plan Optimization, and Scheduling*, w [351], s. 80-84.
[199] Ingber L., Rosen B.: *Genetic Algorithms and Very Fast Simulated Reannealing: A Comparison.* Mathematical and Computer Modelling, Vol.16, No.11, s. 87-100, 1992.
[200] Janikow C.: *Inductive Learning of Decision Rules in Attribute-Based Examples: a Knowledge--Intensive Genetic Algorithm Approach.* PhD Dissertation, University of North Carolina at Chapel Hill, July 1991.
[201] Janikow C., Michalewicz Z.: *Specialized Genetic Algorithms for Numerical Optimization Problems.* Proceedings of the International Conference on Tools for AI, s. 798-804, 1990.
[202] Janikow C., Michalewicz Z.: *On the Convergence Problem in Genetic Algorithms.* UNCC Technical Report, 1990.
[203] Janikow C., Michalewicz Z.: *An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms*, w [32], s. 31-36.
[204] Jankowski A., Michalewicz Z., Ras Z., Shoff D.: *Issues on Evolution Programming*, in Computing and Information, (R. Janicki, W.W. Koczkodaj, Editors), North-Holland, Amsterdam, 1989, s. 459-463.
[205] Jarvis R.A., Byrne J.C.: *Robot Navigation: Touching, Seeing, and Knowing.* Proceedings of the 1st Australian Conference on AI, 1986.
[206] Jog P., Suh J.Y., Gucht D.V.: *The Effects of Population Size, Heuristic Crossover, and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem*, w [344], s. 110-115.
[207] Johnson D.S.: *Local Optimization and the Traveling Salesman Problem*, w M.S. Paterson (Editor), Proceedings of the 17th Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science, Vol.443, s. 446-461, Springer-Verlag, 1990.
[208] Johnson D.S.: *The Traveling Salesman Problem: A Case Study in Local Search*, przedstawione na Metaheuristics International Conference, Breckenridge, Colorado, July 22-26, 1995.
[209] Johnson D.S.: Informacja ustna, October 1995.
[210] Joines J.A., Houck C.R.: *On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs*, w [276], s. 579-584.

[211] Jones D.R., Beltramo M.A.: *Solving Partitioning Problems with Genetic Algorithms*, w [32], s. 442-449.
[212] Jones T.: *A Description of Holland's Royal Road Function* Evolutionary Computation, Vol.2, No.4, 1994, s. 409-415.
[213] Jones T.: *Crossover, Macromutation, and Population-based Search*, w [103], s. 73-80.
[214] Jones T., Forrest S.: *Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms*, w [103], s. 184-192.
[215] Juliff K.: *A Multi-chromosome Genetic Algorithm for Pallet Loading*, w [129], s. 467-473.
[216] Julstrom B.A.: *What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm*, w [103], s. 81-87.
[217] Kaufman K.A., Michalski R.S., Scultz A.C.: *EMERALD 1: An Integrated System for Machine Learning and Discovery Programs for Education and Research*. Center for AI, George Mason University, User's Guide, No. MLI-89-12, 1989.
[218] Kambhampati S.K., Davis L.S.: *Multi-resolution Path Planning for Mobile Robots*. IEEE Journal of Robotics and Automation, RA-2, s. 135-145, 1986.
[219] Karp R.M.: *Probabilistic Analysis of Partitioning Algorithm for the Traveling Salesman Problem in the Plane*. Mathematics of Operations Research, Vol.2, No.3, 1977, s. 209-224.
[220] Keane A.: *Genetic Algorithms Digest*. Thursday, May 19, 1994, Volume 8, Issue 16.
[221] Kelly J.P., Golden B.L., Assad A.A.: *Large Scale Controlled Rounding Using Tabu Search with Strategic Oscillation*, Annals of Operations Research, Vol.41, s. 69-84, 1993.
[222] Khatib O.: *Real-Time Obstacles Avoidance for Manipulators and Mobile Robots*. International Journal of Robotics Research, Vol.5, s. 90-98, 1986.
[223] Khuri S., Bäck T., Heitkötter J.: *The Zero/One Multiple Knapsack Problem and Genetic Algorithms*. Proceedings of the ACM Symposium of Applied Computation (SAC '94).
[224] Kingdon J.: *Genetic Algorithms: Deception, Convergence and Starting Conditions*. Technical Report, Dept. of Computer Science, University College, London, 1992.
[225] Kinnear K.E. (Editor): *Advances in Genetic Programming*. MIT Press, Cambridge, MA, 1994.
[226] Kodratoff Y., Michalski R.: *Machine Learning: An Artificial Intelligence Approach*. Vol.3, Morgan Kaufmann Publishers, San Mateo, CA, 1990.
[227] Korte B.: *Applications of Combinatorial Optimization*. Talk at the 13th International Mathematical Programming Symposium, Tokyo, 1988.
[228] Koza J.R.: *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Report No. STAN-CS-90-1314, Stanford University, 1990.
[229] Koza J.R.: *A Hierarchical Approach to Learning the Boolean Multiplexer Function*, w [318], s. 171-192.
[230] Koza J.R.: *Evolving a Computer Program to Generate Random Numbers Using the Genetic Programming Paradigm*, w [32], s. 37-44.
[231] Koza J.R.: *Genetic Programming*. MIT Press, Cambridge, MA, 1992.
[232] Koza J.R.: *Genetic Programming – 2*. MIT Press, Cambridge, MA, 1994.
[233] Laarhoven P.J.M. van, Aarts E.H.L.: *Simulated Annealing: Theory and Applications*. D. Reidel, Dordrecht, Holland, 1987.
[234] Langley P.: *On Machine Learning*. Machine Learning, Vol.1, No.1, s. 5-10, 1986.
[235] von Laszewski G.: *Intelligent Structural Operators for the k-Way Graph Partitioning Problem*, w [32], s. 45-52.
[236] Lawer E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B.: *The Traveling Salesman Problem*. John Wiley, Chichester, UK, 1985.
[237] Lee F.N.: *The Application of Commitment Utilization Factor (CUF) to Thermal Unit Commitment*. IEEE Transactions on Power Systems, Vol.6, No.2, s. 691-698, May 1991.
[238] Le Riche R., Haftka R.T.: *Improved Genetic Algorithm for Minimum Thickness Composite Laminate Design*. Composites Engineering, Vol.3, No.1, s. 121-139, 1995.
[239] Le Riche R., Knopf-Lenoir C., Haftka R.T.: *A Segregated Genetic Algorithm for Constrained Structural Optimization*, w [103], s. 558-565.

[240] Leler W.: *Constraint Programming Languages: Their Specification and Generation*. Addison--Wesley, Reading, MA, 1988.

[241] Lidd M.L.: *Traveling Salesman Problem Domain Application of a Fundamentally New Approach to Utilizing Genetic Algorithms*. Technical Report, MITRE Corporation, 1991.

[242] Liepins G.E., Vose M.D.: *Representational Issues in Genetic Optimization*. Journal of Experimental and Theoretical Artificial Intelligence, Vol.2, No.2, s. 4-30, 1990.

[243] Lin H.-S., Xiao J., Michalewicz Z.: *Evolutionary Algorithm for Path Planning in Mobile Robot Environment*, w [275], s. 211-216.

[244] Lin H.-S., Xiao J., Michalewicz Z.: *Evolutionary Navigator for a Mobile Robot*, Proceedings of the IEEE Conference on Robotics and Automation, IEEE Computer Society Press, New York, 1994, s. 2199-2204.

[245] Lin S., Kernighan B.W.: *An Effective Heuristic Algorithm for the Traveling Salesman Problem*. Operations Research, s. 498-516, 1972.

[246] Litke J.D.: *An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes*. Communications of the ACM, Vol.27, No.12, s. 1227-1236, 1984.

[247] Logan T.D.: *GENOCOP: An Evolution Program for Optimization Problems with Linear Constraints*, Master Thesis. Department of Computer Science, University of North Carolina at Charlotte, 1993.

[248] Lozano-Perez T., Wesley M.A.: *An Algorithm for Planning Collision Free Paths among Polyhedral Obstacles*. Communications of the ACM, Vol.22, s. 560-570, 1979.

[249] Mahfoud S.W.: *A Comparison of Parallel and Sequential Niching Methods*, w [103], s. 136--143.

[250] Maniezzo V.: *Granularity Evolution*. Dipartimento di Elettronica e Informazione, Politecnico di Milano, Technical Report, 1993.

[251] Männer R., Manderick B. (Editors): Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN), North-Holland, Elsevier Science Publishers, Amsterdam, 1992.

[252] Martello S., Toth P.: *Knapsack Problems*. John Wiley, Chichester, UK, 1990.

[253] McCormick G.P.: *Computability of Global Solutions to Factorable Nonconvex Programs, Part I: Convex Underestimating Problems*. Mathematical Programming, Vol.10, No.2 (1976), s. 147-175.

[254] McDonnell J.R., Reynolds R.G., Fogel D.B. (Editors): Proceedings of the Fourth Annual Conference on Evolutionary Programming, The MIT Press, 1995.

[255] Messa K., Lybanon M.: *A New Technique for Curve Fitting*. Naval Oceanographic and Atmospheric Research Report JA 321:021:91, 1991.

[256] Michalewicz Z. (Editor): Proceedings of the 5th International Conference on Statistical and Scientific Databases, Lecture Notes in Computer Science, Vol.420, Springer-Verlag, 1990.

[257] Michalewicz Z.: *A Genetic Algorithm for Statistical Database Security*, IEEE Bulletin on Database Engineering, Vol.13, No.3, September 1990, s. 19-26.

[258] Michalewicz Z.: *EVA Programming Environment*. UNCC Technical Report, 1990.

[259] Michalewicz Z.: *Optimization of Communication Networks*. Proceedings of the SPIE International Symposium on Optical Engineering and Photonics in Aerospace Sensing, SPIE, Bellingham, WA, 1991, s. 112-122.

[260] Michalewicz Z. (Editor): *Statistical and Scientific Databases*. Ellis Horwood, London, 1991.

[261] Michalewicz Z.: *A Hierarchy of Evolution Programs: An Experimental Study*. Evolutionary Computation, Vol.1, No.1, 1993, s. 51-76.

[262] Michalewicz Z.: *Evolutionary Computation Techniques for Nonlinear Programming Problems*. International Transactions in Operational Research, 1994.

[263] Michalewicz Z. (Ed.): Statistics & Computing, special issue on evolutionary computation, 1994.

[264] Michalewicz Z.: *A Hierarchy of Evolution Programs: An Experimental Study*, Evolutionary Computation, Vol.1, No.1, 1993, s. 51-76.

[265] Michalewicz Z. (Editor): Statistics & Computing, special issue on evolutionary computation, 1994.

[266] Michalewicz Z.: *Genetic Algorithms, Numerical Optimization and Constraints*, w [103], s. 151-158.

[267] Michalewicz Z., Attia N.: *Evolutionary Optimization of Constrained Problems*, w [378], s. 98-108.

[268] Michalewicz Z., Janikow C.: *Genetic Algorithms for Numerical Optimization*. Statistics and Computing, Vol.1, No.1, 1991.

[269] Michalewicz Z., Janikow C.: *Handling Constraints in Genetic Algorithms*, w [32], s. 151-157.

[270] Michalewicz Z., Janikow C.: *GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints*, accepted for publication, Communications of the ACM, 1992.

[271] Michalewicz Z., Janikow C., Krawczyk J.: *A Modified Genetic Algorithm for Optimal Control Problems*, Computers & Mathematics with Applications, Vol.23, No.12, s. 83-94, 1992.

[272] Michalewicz Z., Jankowski A., Vignaux G.A.: *The Constraints Problem in Genetic Algorithms*, w *Methodologies of Intelligent Systems: Selected Papers*, M.L. Emrich, M.S. Phifer, B. Huber, M. Zemankova, Z. Ras (Editors), ICAIT, Knoxville, TN, s. 142-157, 1990.

[273] Michalewicz Z., Krawczyk J., Kazemi M., Janikow C.: *Genetic Algorithms and Optimal Control Problems*. Proceedings of the 29th IEEE Conference on Decision and Control, Honolulu, s. 1664-1666, 1990.

[274] Michalewicz Z., Logan T.D., Swaminathan S.: *Evolutionary Operators for Continuous Convex Parameter Spaces*, w [378], s. 84-97.

[275] Michalewicz Z., Schaffer D., Schwefel H.-P., Fogel D., Kitano H. (Editors): Proceedings of the First IEEE International Conference on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, Volume 1, Orlando, 27-29 June, 1994.

[276] Michalewicz Z., Schaffer D., Schwefel H.-P., Fogel D., Kitano H. (Editors): Proceedings of the First IEEE International Conference on Evolutionary Computation, IEEE Service Center, Piscataway, NJ, Volume 2, Orlando, 27-29 June, 1994.

[277] Michalewicz Z., Vignaux G.A., Groves L.: *Genetic Algorithms for Optimization Problems*, Proceedings of the 11th NZ Computer Conference, Wellington, New Zealand, s. 211-223, 1989.

[278] Michalewicz Z., Vignaux G.A., Hobbs M.: *A Non-Standard Genetic Algorithm for the Nonlinear Transportation Problem*. ORSA Journal on Computing, Vol.3, No.4, s. 307-316, 1991.

[279] Michalewicz Z., Xiao J.: *Evaluation of Paths in Evolutionary Planner/Navigator*. Proceedings of the 1995 International Workshop on Biologically Inspired Evolutionary Systems, Tokyo, Japan, May 30-31, 1995, s. 45-52.

[280] Michalski R.: *A Theory and Methodology of Inductive Learning*, in R. Michalski, J. Carbonell, T. Mitchell (Editors), *Machine Learning: An Artificial Intelligence Approach*. Vol.1, Tioga Publishing Co., Palo Alto, CA, 1983, i Springer-Verlag, 1994, s. 83-134.

[281] Michalski R., Mozetic I., Hong J., Lavrac N.: *The AQ15 Inductive Learning System: An Overview and Experiments*. Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.

[282] Michalski R., Watanabe L.: *Constructive Closed-Loop Learning: Fundamental Ideas and Examples*. MLI-88, George Mason University, 1988.

[283] Michalski R.: *A Methodological Framework for Multistrategy Task-Adaptive Learning*. Methodologies for Intelligent Systems, Vol.5, w Z. Ras, M. Zemankowa, M. Emrich (Editors), North-Holland, Amsterdam, 1990.

[284] Michalski R., Kodratoff Y.: *Research in Machine Learning: Recent Progress, Classification of Methods, and Future Directions*, w [226], s. 3-30.

[285] Montana D.J., Davis L.: *Training Feedforward Neural Networks Using Genetic Algorithms*. Proceedings of the 1989 International Joint Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[286] Mühlenbein H.: *Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization*, w [344], s. 416-421.

[287] Mühlenbein H.: *Parallel Genetic Algorithms in Combinatorial Optimization*, w O. Balci, R. Sharda, S.A. Zenios (Editors), *Computer Science and Operations Research – New Developments in Their Interfaces*. Pergamon Press, New York, 1992, s. 441-453.

[288] Mühlenbein H., Gorges-Schleuter M., Krämer O.: *Evolution Algorithms in Combinatorial Optimization*. Parallel Computing, Vol.7, s. 65-85, 1988.

[289] Mühlenbein H., Schlierkamp-Vosen D.: *Predictive Models for the Breeder Genetic Algorithm*. Evolutionary Computation, Vol.1, No.1, s. 25-49, 1993.

[290] Mühlenbein H., Voigt H.-M.: *Gene Poool Recombination for the Breeder Genetic Algorithm*. Proceedings of the Metaheuristics International Conference, Breckenridge, Colorado, July 22-26, 1995, s. 19-25.

[291] Murray W.: *An Algorithm for Constrained Minimization*. Optimization, (Ed: R. Fletcher), s. 247-258, Academic Press, London, 1969.

[292] Murtagh B.A., Saunders M.A.: *MINOS 5.1 User's Guide*. Report SOL 83-20R, December 1983, revised January 1987, Stanford University.

[293] Muselli M., Ridella S.: *Global Optimization of Functions with the Interval Genetic Algorithm*, Complex Systems, Vol.6, s. 193-212, 1992.

[294] Myung H., Kim J.-H., Fogel D.B.: *Preliminary Investigations into a Two-Stage Method of Evolutionary Optimization on Constrained Problems*, w [254], s. 449-463.

[295] Nadhamuni P.V.R.: *Application of Co-evolutionary Genetic Algorithm to a Game*. Master Thesis, Department of Computer Science, University of North Carolina, Charlotte, 1995.

[296] von Neumann J.: *Theory of Self-Reproducing Automata*, edited by Burks, University of Illinois Press, 1966.

[297] Nissen V.: *Evolutionary Algorithms in Management Science: An Overview and List of References*. European Study Group for Evolutionary Economics, 1993.

[298] Ng K.P, Wong K.C.: *A New Diploid Scheme and Dominance Change Mechanism for Non--Stationary Function Optimization*, w [103], s. 159-166.

[299] Oliver I.M., Smith D.J., Holland J.R.C.: *A Study of Permutation Crossover Operators on the Traveling Salesman Problem*, w [171], s. 224-230.

[300] Olsen A.: *Penalty Functions and the Knapsack Problem*, w [276], s. 554-558.

[301] Orvosh D., Davis L.: *Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints*, w [129], s. 650.

[302] Padberg M., Rinaldi G.: *Optimization of a 532-City Symmetric Travelling Salesman Problem*. Technical Report IASI-CNR, Italy, 1986.

[303] Paechter B., Luchian H., Petruic M.: *Two Solutions to the General Timetable Problem Using Evolutionary Methods*, w [275], s. 300-305.

[304] Palmer C.C., Kershenbaum A.: *Representing Trees in Genetic Algorithms*, w [275], s. 379-384.

[305] Pardalos P.: *On the Passage from Local to Global in Optimization*, w „Mathematical Programming", J.R. Birge, K.G. Murty (Editors), The University of Michigan, 1994.

[306] Paredis J.: *Exploiting Constraints as Background Knowledge for Genetic Algorithms: a Case--study for Scheduling*, w [251], s. 229-238.

[307] Paredis J.: *Genetic State-Space Search for Constrained Optimization Problems*. Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, 1993.

[308] Paredis J.: *Co-evolutionary Constraint Satisfaction*, w [70], s. 46-55.

[309] Paredis J.: *The Symbiotic Evolution of Solutions and their Representations*, w [103], s. 359-365.

[310] Pavlidid T.: *Algorithms for Graphics and Image Processing*. Computer Science Press, 1982.

[311] Potter M., De Jong K.: *A Cooperative Coevolutionary Approach to Function Optimization*, w [70], s. 249-257.

[312] Powell D., Skolnick M.M.: *Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints*, w [129], s. 424-430.

[313] Powell M.J.D.: *Variable Metric Methods for Constrained Optimization*. Mathematical Programming: The State of the Art, (Eds: A.Bachem, M. Grötschel, B. Korte), s. 288-311, Springer-Verlag, 1983.

[314] Quinlann J.R.: *Induction of Decision Trees*. Machine Learning, Vol.1, No.1, 1986.
[315] Radcliffe N.J.: *Formal Analysis and Random Respectful Recombination*, w [32], s. 222-229.
[316] Radcliffe N.J.: *Genetic Set Recombination*, w [398], s. 203-219.
[317] Radcliffe N.J., George F.A.W.: *A Study in Set Recombination*, w [129], s. 23-30.
[318] Rawlins G.: *Foundations of Genetic Algorithms*. First Workshop on the Foundations of Genetic Algorithms and Classifier Systems. Morgan Kaufmann Publishers, San Mateo, CA, 1991.
[319] Rechenberg I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, 1973.
[320] Reeves C.R.: *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, London, 1993.
[321] Reinelt G.: *TSPLIB – A Traveling Salesman Problem Library*. ORSA Journal on Computing, Vol.3, No.4, s. 376-384, 1991.
[322] Reinke R.E., Michalski R.: *Incremental Learning of Concept Descriptions*, in D. Michie (Editor), Machine Intelligence XI, 1985.
[323] Rendell Larry A.: *Genetic Plans and the Probabilistic Learning System: Synthesis and Results*, w [167], s. 60-73.
[324] Renders J.-M., Bersini H.: *Hybridizing Genetic Algorithms with Hill-climbing Methods for Global Optimization: Two Possible Ways*, w [275], s. 312-317.
[325] Reynolds R.G.: *An Introduction to Cultural Algorithms*, w [378], s. 131-139.
[326] Reynolds R.G., Brown W., Abinoja E.O.: *Guiding Parallel Bi-Directional Search Using Cultural Algorithms*, w [378], s. 167-174.
[327] Reynolds R.G., Maletic J.I.: *The Use of Version Space Controlled Genetic Algorithms to Solve the Boole Problem*. International Journal on Artificial Intelligence Tools, Vol.2, No.2, s. 219-234, 1993.
[328] Reynolds R.G., Maletic J.I.: *Learning to Cooperate Using Cultural Algorithms*, w [378], s. 140-149.
[329] Reynolds R.G., Michalewicz Z., Cavaretta M.: *Using Cultural Algorithms for Constraint Handling in Genocop*, w [254], s. 289-305.
[330] Reynolds R.G., Sverdlik W.: *Solving Problems in Hierarchically Structured Systems Using Cultural Algorithms*, w [124], s. 144-153.
[331] Reynolds R.G., Zannoni E., Posner R.M.: *Learning to Understand Software Using Cultural Algorithms*, w [378], s. 150-157.
[332] Richardson J.T., Palmer M.R., Liepins G., Hilliard M.: *Some Guidelines for Genetic Algorithms with Penalty Functions*, w [344], s. 191-197.
[333] Ronald E.: *When Selection Meets Seduction*, w [103], s. 167-173.
[334] Rudolph G.: *Convergence Analysis of Canonical Genetic Algorithms*. IEEE Transactions on Neural Networks, special issue on evolutionary computation, Vol.5, No.1, 1994.
[335] Rumelhart D., McClelland J.: *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, Vol.1. MIT Press, Cambridge, MA, 1986.
[336] Saravanan N., Fogel D.B.: *A Bibliography of Evolutionary Computation & Applications*. Department of Mechanical Engineering, Florida Atlantic University, Technical Report No. FAU-ME-93-100, 1993.
[337] Sarle W.: *Kangaroos*, artykuł wysłany na comp.ai.neural-nets 1 września 1993.
[338] Sasieni M., Yaspan A., Friedman L.: *Operations Research Methods and Problems*. John Wiley, Chichester, UK, 1959.
[339] Schaffer J.D.: *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms*. PhD Dissertation, Vanderbilt University, Nashville, 1984.
[340] Schaffer J.D.: *Learning Multiclass Pattern Discrimination*, w [167], s. 74-79.
[341] Schaffer J.D., Morishima A.: *An Adaptive Crossover Distribution Mechanism for Genetic Algorithms*, w [171], s. 36-40.
[342] Schaffer J.D.: *Some Effects of Selection Procedures on Hyperplane Sampling by Genetic Algorithms*, w [73], s. 89-103.

[343] Schaffer J., Caruana R., Eshelman L., Das R.: *A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization*, w [344], s. 51-60.

[344] Schaffer J. (Editor): Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA, 1989.

[345] Schaffer J.D., Whitley D., Eshelman L.J.: *Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art*. Proceedings of the International Workshop on Combinations of Genetic Algorithms and Neural Networks, Baltimore, MD, June 6, 1992, s. 1-37.

[346] Schoenauer M., Xanthakis S.: *Constrained GA Optimization*, w [129], s. 573-580.

[347] Schraudolph N., Belew R.: *Dynamic Parameter Encoding for Genetic Algorithms*. Machine Learning, Vol.9, No.1, s. 9-21, 1992.

[348] Schwefel H.-P.: *Numerical Optimization for Computer Models*, John Wiley, Chichester, UK, 1981.

[349] Schwefel H.-P.: *Evolution Strategies: A Family of Non-Linear Optimization Techniques Based on Imitating Some Principles of Organic Evolution*. Annals of Operations Research, Vol.1, s. 165-167, 1984.

[350] Schwefel H.-P.: *Evolution and Optimum Seeking*. John Wiley, Chichester, UK, 1995.

[351] Schwefel H.-P., Manner, R. (Editors): Proceedings of the First International Conference on Parallel Problem Solving from Nature (PPSN). Lecture Notes in Computer Science, Vol.496, Springer-Verlag, 1991.

[352] Schwefel H.-P.: Informacja ustna, July 1991.

[353] Seniw D.: *A Genetic Algorithm for the Traveling Salesman Problem*. MSc Thesis, University of North Carolina at Charlotte, 1991.

[354] Shaefer C.G.: *The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique*, w [171], s. 50-55.

[355] Shibata T., Fukuda T.: *Robot Motion Planning by Genetic Algorithm with Fuzzy Critic*. Proceedings of the 8th IEEE International Symposium on Intelligent Control, August 25-27, 1993.

[356] Shing M.T., Parker G.B.: *Genetic Algorithms for the Development of Real-Time Multi-Heuristic Search Strategies*, w [129], s. 565-570, 1993.

[357] Shonkwiler R., Van Vleck E.: Parallel Speed-up of Monte Carlo Methods for Global Optimization, nieopublikowany rękopis, 1991.

[358] Siedlecki W., Sklanski J.: *Constrained Genetic Optimization via Dynamic Reward-Penalty Balancing and Its Use in Pattern Recognition*, w [344], s. 141-150.

[359] Sirag D.J., Weisser P.T.: *Toward a Unified Thermodynamic Genetic Operator*, w [171], s. 116-122.

[360] Smith A., Tate D.: *Genetic Optimization Using A Penalty Function*, w [129], s. 499-503.

[361] Smith D.: *Bin Packing with Adaptive Search*, w [167], s. 202-207.

[362] Smith R.E.: *Adaptively Resizing Populations: An Algorithm and Analysis*, w [129], s. 653.

[363] Smith S.F.: *A Learning System Based on Genetic Algorithms*. PhD Dissertation, University of Pittsburgh, 1980.

[364] Smith S.F.: *Flexible Learning of Problem Solving Heuristics through Adaptive Search*. Proceedings of the Eighth International Conference on Artificial Intelligence, Morgan Kaufmann Publishers, San Mateo, CA, 1983.

[365] Spears W.M.: *Simple Subpopulation Schemes*, w [378], s. 296-307.

[366] Spears W.M., De Jong K.A.: *On the Virtues of Parametrized Uniform Crossover*, w [32], s. 230-236.

[367] Spears W.M.: *Adapting Crossover in Evolutionary Algorithms*, w [254], s. 367-384.

[368] Srinivas M., Patnaik L.M.: *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, Vol.24, No.4, 1994, s. 17-26.

[369] Srinivas N., Deb K.: *Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms*. Evolutionary Computation, Vol.2, No.3, 1994, s. 221-248.

[370] Starkweather T., McDaniel S., Mathias K., Whitley C., Whitley D.: *A Comparison of Genetic Sequencing Operators*, w [32], s. 69-76.

[371] Stebbins G.L.: *Darwin to DNA, Molecules to Humanity*. W.H. Freeman: New York, 1982.

[372] Stein D.: *Scheduling Dial a Ride Transportation Systems: An Asymptotic Approach*. PhD Dissertation, Harvard University, 1977.

[373] Stevens J.: *A Genetic Algorithm for the Minimum Spanning Tree Problem*. MSc Thesis, University of North Carolina at Charlotte, 1991.

[374] Stuckman B.E., Easom E.E.: *A Comparison of Bayesian/Sampling Global Optimization Techniques*. IEEE Transactions on Systems, Man, and Cybernetics, Vol.22, No.5, s. 1024--1032, 1992.

[375] Suh J.-Y., Gucht Van D.: *Incorporating Heuristic Information into Genetic Search*, w [171], s. 100-107.

[376] Surry P.D., Radcliffe N.J., Boyd I.D.: *A Multi-objective Approach to Constrained Optimization of Gas Supply Networks*, przedstawione na AISB-95 Workshop on Evolutionary Computing, Sheffield, UK, April 3-4, 1995.

[377] Sverdlik W., Reynolds R.G.: *Incorporating Domain Specific Knowledge into Version Space Search*. Proceedings of the 1993 IEEE International Conference on Tools with AI, Boston, MA, November 1993, s. 216-223.

[378] Sebald A.V., Fogel L.J.: Proceedings of the Third Annual Conference on Evolutionary Programming. San Diego, CA, 1994, World Scientific.

[379] Steele J.M.: *Probabilistic Algorithm for the Directed Traveling Salesman Problem*. Mathematics of Operations Research, Vol.11, No.2, 1986, s. 343-350.

[380] Svirezhev Yu.M., Passekov V.P.: *Fundamentals of Mathematical Evolutionary Genetics*. Kluwer Academic Publishers, Mathematics and Its Applications (Soviet Series), Vol.22, London, 1989.

[381] Sysło M.M., Deo N., Kowalik J.S.: *Discrete Optimization Algorithms*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[382] Syswerda G.: *Uniform Crossover in Genetic Algorithms*, w [344], s. 2-9.

[383] Syswerda G.: *Schedule Optimization Using Genetic Algorithms*, w [78], s. 332-349.

[384] Syswerda G., Palmucci J.: *The Application of Genetic Algorithms to Resource Scheduling*, w [32], s. 502-508.

[385] Suh J.-Y., Lee C.-D.: *Operator-Oriented Genetic Algorithm and Its Application to Sliding Block Puzzle Problem*, w [351], s. 98-103.

[386] Szałas A., Michalewicz Z.: *Contractive Mapping Genetic Algorithms and Their Convergence*. Department of Computer Science, University of North Carolina at Charlotte, Technical Report 006-1993.

[387] Taha H.A.: *Operations Research: An Introduction*. 4th ed., Collier Macmillan, London, 1987.

[388] Tamassia R., Di Battista G., Batini C.: *Automatic Graph Drawing and Readability of Diagrams*. IEEE Transactions Systems, Man, and Cybernetics, Vol.18, No.1, s. 61-79, 1988.

[389] Ulder N.L.J., Aarts E.H.L., Bandelt H.-J., van Laarhoven P.J.M., Pesch E.: *Genetic Local Search Algorithms for the Traveling Salesman Problem*, w [351], s. 109-116.

[390] Valenzuela C.L., Jones A.J.: *Evolutionary Divide and Conquer (I): A Novel Genetic Approach to the TSP*. Evolutionary Computation, Vol.1, No.4, 1994, s. 313-333.

[391] Vignaux G.A., Michalewicz Z.: *Genetic Algorithms for the Transportation Problem*, Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems, North-Holland, Amsterdam, s. 252-259, 1989.

[392] Vignaux G.A., Michalewicz Z.: *A Genetic Algorithm for the Linear Transportation Problem*. IEEE Transactions on Systems, Man, and Cybernetics, Vol.21, No.2, s. 445-452, 1991.

[393] Voss S.: *Tabu Search: Applications and Prospects*. Technical Report, Technische Hochshule Darmstadt, 1993.

[394] Whitley D.: *GENITOR: A Different Genetic Algorithm*. Proceedings of the Rocky Mountain Conference on Artificial Intelligence, Denver, 1988.

[395] Whitley D.: *The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best*, w [344], s. 116-121.

[396] Whitley D.: *GENITOR II: A Distributed Genetic Algorithm*. Journal of Experimental and Theoretical Artificial Intelligence, Vol.2, s. 189-214.

[397] Whitley D.: *Genetic Algorithms: A Tutorial*, w [263].

[398] Whitley D. (Editor): *Foundations of Genetic Algorithms-2*. Second Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[399] Whitley D., Gordon V.S., Mathias K.: *Lamarckian Evolution, the Baldwin Effect and Function Optimization*, w [70], s. 6-15.

[400] Whitley D., Mathias K., Fitzhorn P.: *Delta Coding: An Iterative Search Strategy for Genetic Algorithms*, w [32], s. 77-84.

[401] Whitley D., Mathias K., Rana S., Dzubera J.: *Building Better Test Functions*, w [103], s. 239-246.

[402] Whitley D., Starkweather T., Fuquay D'A.: *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator*, w [344], s. 133-140.

[403] Whitley D., Starkweather T., Shaner D.: *Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination*, w [78], s. 350-372.

[404] Wilson R.B.: *Some Theory and Methods of Mathematical Programming*. Ph.D. Dissertation, Harvard University, Graduate School of Business Administration, 1963.

[405] Winston W.L.: *Operations Research: Applications and Algorithms*. Duxbury, Boston, 1987.

[406] Wirth N.: *Algorithms + Data Structures = Programs*. Prentice-Hall, Englewood Cliffs, NJ, 1976.

[407] Wnęk J., Sarma J., Wahab A.A., Michalski R.S.: *Comparing Learning Paradigms via Diagrammatic Visualization*. Proceedings of the 5th International Symposium on Methodologies for Intelligent Systems, North-Holland, Amsterdam, s. 428-437, 1990.

[408] Wright A.H.: *Genetic Algorithms for Real Parameter Optimization*, w [318], s. 205-218.

[409] Xiao J.: *Evolutionary Planner/Navigator in a Mobile Robot Environment*, w [17], section G3.11.

[410] Xu J., Kelly J.P.: *A Robust Network Flow-Based Tabu Search Approach for the Vehicle Routing Problem*. Graduate School of Business, University of Colorado, Boulder, 1995.

[411] Yagiura Y., Ibaraki T.: *GA and Local Search Algorithms as Robust and Simple Optimization Tools*. Proceedings of the Metaheuristics International Conference, Breckenridge, Colorado, July 22-26, 1995, s. 129-134.

[412] Yao X.: *A Review of Evolutionary Artificial Neural Networks*. Technical Report, CSIRO, Highett, Victoria, Australia, 1993.

[413] Yao X., Darwen P.: *An Experimental Study of N-person Prisoner's Dilemma Games*. Proceedings of the Workshop on Evolutionary Computation, University of New England, November 21-22, 1994, s. 94-113.

[414] Zhou H.H., Grefenstette J.J.: *Learning by Analogy in Genetic Classifier Systems*, w [344], s. 291-297, 1989.

[415] Zurada J., Marks R., Robinson C. (Editors): *Computational Intelligence: Imitating Life*. IEEE Press, 1994.

Słownik angielsko-polski

Adaptive control – sterowanie adaptacyjne
adjacency representation – reprezentacja przyległościowa
affinity test – test podobieństwa
allele – allele
ARGOT strategy – strategia ARGOT
arithmetical crossover – krzyżowanie arytmetyczne
automatically defined functions – funkcje definiowane automatycznie

Baldwin effect – efekt Baldwina
Banach fixpoint theorem – twierdzenie Banacha o punkcie stałym
behavioral memory – pamięć behawioralna
bin packing crossover operator – operator krzyżowania dla pakowania pojemników
binary matrix representation – reprezentacja z macierzą binarną
Boltzmann distribution – rozkład Boltzmanna
Boolean satisfiability problem (SAT) – zadanie spełniania logicznego
boundary mutation – mutacja brzegowa
breeder genetic algorithms – algorytmy genetyczne płodzące
building block hypothesis – hipoteza o blokach budujących

Cauchy sequence – ciąg Cauchy'ego
chromosome lifetime parameter – parametr czasu życia chromosomu
chromosome variable length – zmienna długość chromosomu
classifier system – system klasyfikujący
CNF-satisfiability problem – zadanie spełniania postaci normalnej koniunkcyjnej
co-evolutionary systems – systemy współewoluujące
cognitive modeling – modelowanie poznawcze
communication network – sieć komunikacyjna
constraint satisfaction problem – zadanie spełniania ograniczeń
constraints – ograniczenia
crossover – krzyżowanie
crossover, alternating-edges – krzyżowanie z wymianą krawędzi
crossover, cycle (CX) – krzyżowanie cykliczne
crossover, edge recombination enhanced – rozszerzone krzyżowanie z rekombinacją krawędzi
crossover, guaranteed average – krzyżowanie gwarantowane średnie
crossover, heuristic – krzyżowanie heurystyczne
crossover, intermediate – krzyżowanie pośrednie
crossover, multi-point – krzyżowanie wielopunktowe
crossover, one point – krzyżowanie jednopunktowe
crossover, order (OX) – krzyżowanie z porządkowaniem
crossover, order-based – krzyżowanie bazujące na porządku

crossover, partially mapped – krzyżowanie z częściowym odwzorowaniem
crossover, position based – krzyżowanie bazujące na pozycjach
crossover, probability of – prawdopodobieństwo krzyżowania
crossover, segmented – krzyżowanie odcinkowe
crossover, shuffle – krzyżowanie tasowane
crossover, simple – krzyżowanie proste
crossover, simplex – krzyżowanie simpleksowe
crossover, subtour-chunks – krzyżowanie z wydzielaniem podtras
crossover, two-point – krzyżowanie dwupunktowe
crossover, uniform – krzyżowanie jednorodne
crowding factor model – model ze współczynnikiem zatłoczenia
cultural algorithms – algorytmy kulturowe
cycle crossover (CX) – krzyżowanie cykliczne

Database – baza danych
database query optimization – optymalizacja obsługi zapytań bazy danych
death penalty – kara śmierci
deception – zawód
decoders – dekodery
deletion – usunięcie
delta coding algorithm – algorytm delta-kodowania
deterministic sampling – próbkowanie deterministyczne
diploidy – diploidalność
displacement – przemieszczenie
divide and conquer – dziel i zwyciężaj
dominance – dominacja
don't care symbol – symbol nieistotne
dynamic control – sterowanie dynamiczne
dynamic parameter encoding – dynamiczne kodowanie parametrów
dynamic selection – selekcja dynamiczna

Edge recombination crossover – krzyżowanie z rekombinacją krawędzi
elitist model – model elitarny
elitist expected value model – elitarny model wartości oczekiwanej
environment – środowisko
epistasis – epistaza
EVA programming environment – środowisko programowe EVA
evaluation function – funkcja oceny
evolution program – program ewolucyjny
evolution strategies – strategie ewolucyjne
evolutionary programming – programowanie ewolucyjne
exploration – badanie
extinctive selection – selekcja wygaszająca

Finite state machine – automat skończony
fitness distance correlation – korelacja odległości dopasowań
fitness – dopasowanie, przystosowanie
function optimization – optymalizacja funkcji

Game playing – rozgrywanie gier
gene – gen
gene pool recombination – rekombinacja puli genów
gene scanning – przeglądanie genów
general problem solver – program rozwiązywania zadań
generational selection – selekcja pokoleniowa
genetic algorithm – algorytm genetyczny
genetic algorithm, contractive mapping – algorytm genetyczny z odwzorowaniem zwężającym
genetic algorithm, interval – algorytm genetyczny odcinkowy
genetic algorithm, messy (mGA) – algorytm genetyczny nieporządny
genetic algorithm, modified – algorytm genetyczny zmodyfikowany
genetic algorithm, representation direct – reprezentacja bezpośrednia algorytmu genetycznego
genetic algorithm, simple (SGA) – algorytm genetyczny prosty
genetic algorithm, steady state – algorytm genetyczny z ustalonym stanem
genetic algorithm with varying population size – algorytm genetyczny ze zmienną liczebnością populacji
genetic inductive learning – genetyczne uczenie indukcyjne
genetic programming – programowanie genetyczne
granularity evolution – ewolucja ziarnista
Gray codes – kody Graya

Hamming cliff – klif Hamminga
haploidy – haploidalność
harvest problem – zadanie zbierania plonów
heredity – dziedziczność
heuristic methods – metody heurystyczne
hillclimbing – metody wzrostu

hybrid genetic algorithms – hybrydowe algorytmy genetyczne
hyperplane – hiperpłaszczyzna

Immune recruitment mechanism – mechanizm odpornej rekrutacji
implicit parallelism – wewnętrzna równoległość
incest prevention – zapobieganie kazirodztwu
industrial engineering – inżynieria przemysłowa
insertion – wstawienie
intersection – przecięcie
inversion – inwersja

Jenkins nightmare – zmora Jenkinsa

Lagrangian relaxation – relaksacja Lagrange'a
Lamarckian evolution – ewolucja Lamarckiana
Lin-Kernighan algorithm – algorytm Lina-Kernighana
linear scaling – skalowanie liniowe
locus – locus

Machine learning – uczenie maszynowe
maintaining feasibility – utrzymywanie dopuszczalności
management science – zarządzanie
massively parallel GAs – masowe algorytmy genetyczne równoległe
maximum clique problem – zadanie maksymalnej kliki
meta genetic algorithm – algorytm metagenetyczny
Michigan approach – podejście Michigan
multi-membered strategies – strategie wieloelementowe
multi-parent operator – operator wielorodzicielski
multimodal optimization – optymalizacja wielomodalna
multiobjective optimization – optymalizacja wielokryterialna
mutation, non-uniform – mutacja nierównomierna
mutation of order k – mutacja rzędu k
mutation rate – częstość mutacji
mutation, scramble sublist – mutacja z mieszaniem podlist
mutation, uniform – mutacja równomierna

Natural selection – selekcja naturalna
navigation – nawigacja
neural network – sieci neuronowe
niche methods – metody niszy
non random mating – dobór nielosowy

Operator, cut – operator cięcia
operator, splice – operator łączenia
optimal control – sterowanie optymalne
order crossover (OX) – krzyżowanie z porządkowaniem
order of the schema – rząd schematu
ordinal representation – reprezentacja porządkowa
orgies – orgie

Pallet loading problem – zadanie ładowania kontenerów
paralell genetic algorithms – algorytmy genetyczne równoległe
parallel hybrid GAs – algorytmy genetyczne równoległe hybrydowe
parallel island models – modele równoległych wysp
Pareto-optimal points – punkty paretooptymalne
parity check – sprawdzanie parzystości
partially mapped crossover (PMX) – krzyżowanie z częściowym odwzorowaniem
partitioning – podział
path planning – planowanie drogi
path representation – reprezentacja ścieżkowa
penalty functions – funkcje kary
phenotype – fenotyp
Pitt approach – podejście Pitt
population average fitness – średnie dopasowanie populacji
population diversity – różnorodność populacji
population initialization – populacja początkowa
population reinitialization – ponowne zapoczątkowanie populacji
population size – liczebność populacji
power law scaling – skalowanie zgodne z prawem potęgowym
premature convergence – zbieżność przedwczesna
preservative selection – selekcja zachowująca
prisoner's dilemma – dylemat więźnia
Prüfer numbers – liczby Prüfera
push-cart problem – zadanie pchania wózka

Random keys – klucze losowe
random search – przeszukiwanie losowe
ranking linear selection – selekcja porządkowa liniowa
ranking nonlinear selection – selekcja porządkowa nieliniowa
ranking selection – selekcja porządkowa

repair algorithms – algorytmy naprawy
remainder stochastic sampling without replacement – stochastyczne próbkowanie na podstawie reszty bez zamiany
representation, binary – reprezentacja binarna
representation, floating point – reprezentacja zmiennopozycyjna
reproductive scheme growth equation – równanie wzrostu schematu
roulette wheel selection – selekcja ruletkowa
royal road functions – funkcje królewskiej drogi

Scaling – skalowanie
scaling window – okno skalujące
scatter search – przeszukiwanie rozproszone
scheduling – szeregowanie, harmonogramowanie
schema defining length – długość definiująca schematu
schema fitness – dopasowanie schematu
search space – przestrzeń rozwiązań
selection seduction – selekcja z uwodzeniem
selective pressure – napór selekcyjny
self-adapting systems – systemy samoadaptacyjne
serial selection – selekcja seryjna
set covering problem – zadanie pokrycia zbioru
sigma truncation scaling – skalowanie z obcinaniem na poziomie odchylenia standardowego
simulated annealing – symulowane wyżarzanie
smoothing – wygładzanie
solutions, feasible – rozwiązania dopuszczalne
solutions, infeasible – rozwiązania niedopuszczalne
species formation – tworzenie gatunków
static selection – selekcja statyczna
Steiner tree – drzewo Steinera
stochastic tournament – turniej stochastyczny
Strategic oscillation – oscylacje strategiczne
strategy operators – operatory strategii
swap – wymiana
symbolic empirical learning – nauczanie empiryczne symboliczne

Tabu search – przeszukiwanie tabu
termination condition – warunek zakończenia
timetable problem – zadanie układania planu lekcji
tournament selection – selekcja turniejowa
transportation balanced problem – zadanie transportowe zbilansowane
transportation, linear problem – zadanie transportowe liniowe
transportation, nonlinear problem – zadanie transportowe nieliniowe
traveling salesman problem – zadanie komiwojażera
two-membered strategies – strategie dwuelementowe

Union – zespolenie

Variable valued logic system – system logiki zmiennowartościowej
VLSI design – projektowanie VLSI

Wire routing – prowadzenie kabla

Słownik polsko-angielski

Algorytm delta-kodowania – delta coding algorithm
algorytm genetyczny – genetic algorithm
algorytm genetyczny nieporządny – messy genetic algorithm (mGA)
algorytm genetyczny odcinkowy – genetic algorithm, interval
algorytm genetyczny prosty – simple genetic algorithm (SGA)
algorytm genetyczny, reprezentacja bezpośrednia – genetic algorithm, representation direct
algorytm genetyczny z odwzorowaniem zwężającym – contractive mapping genetic algorithm
algorytm genetyczny z ustalonym stanem – steady state genetic algorithm
algorytm genetyczny ze zmienną liczebnością populacji – genetic algorithm with varying population size
algorytm genetyczny zmodyfikowany – modified genetic algorithm
algorytm Lina-Kernighana – Lin-Kernighan algorithm
algorytmy genetyczne płodzące – breeder genetic algorithms
algorytmy genetyczne równoległe – paralell genetic algorithms
algorytmy genetyczne równoległe hybrydowe – parallel hybrid GAs
algorytmy genetyczne równoległe masowe – massively parallel GAs
algorytmy kulturowe – cultural algorithms
algorytmy naprawy – repair algorithms
allele – allele
automaty skończone – finite state machine

Badanie – exploration
baza danych – database
baza danych, optymalizacja obsługi zapytań – database query optimization

Ciąg Cauchy'ego – Cauchy sequence
częstość mutacji – mutation rate

Dekodery – decoders
diploidalność – diploidy
długość definiująca schematu – schema defining length
dobór nielosowy – non random mating
dominacja – dominance
dopasowanie, przystosowanie – fitness
dopasowanie schematu – schema fitness
drzewo Steinera – Steiner tree
dylemat więźnia – prisoner's dilemma
dynamiczne kodowanie parametrów – dynamic parameter encoding
dziedziczność – heredity
dziel i zwyciężaj – divide and conquer

Efekt Baldwina – Baldwin effect
epistaza – epistasis
ewolucja Lamarckiana – Lamarckian evolution
ewolucja ziarnista – granularity evolution

Fenotyp – phenotype
funkcja oceny – evaluation function

funkcje definiowane automatycznie – automatically defined functions
funkcje kary – penalty functions
funkcje królewskiej drogi – royal road functions

Gen – gene
genetyczne uczenie indukcyjne – genetic inductive learning

Haploidalność – haploidy
harmonogramowanie – scheduling
hiperpłaszczyzna – hyperplane
hipoteza o blokach budujących – building block hypothesis
hybrydowe algorytmy genetyczne – hybrid genetic algorithms

Inwersja – inversion
inżynieria przemysłowa – industrial engineering

Kara śmierci – death penalty
klif Hamminga – Hamming cliff
klucze losowe – random keys
kody Graya – Gray codes
korelacja odległości dopasowań – fitness distance correlation
krzyżowanie – crossover
krzyżowanie arytmetyczne – arithmetical crossover
krzyżowanie bazujące na porządku – order-based crossover
krzyżowanie bazujące na pozycjach – position based crossover
krzyżowanie cykliczne – cycle crossover (CX)
krzyżowanie dwupunktowe – two-point crossover
krzyżowanie gwarantowane średnie – guaranteed average crossover
krzyżowanie heurystyczne – heuristic crossover
krzyżowanie jednopunktowe – one point crossover
krzyżowanie jednorodne – uniform crossover
krzyżowanie odcinkowe – segmented crossover
krzyżowanie pośrednie – intermediate crossover
krzyżowanie, prawdopodobieństwo – probability of crossover
krzyżowanie simpleksowe – simplex crossover
krzyżowanie tasowane – shuffle crossover
krzyżowanie wielopunktowe – multi-point crossover
krzyżowanie z częściowym odwzorowaniem – partially mapped crossover (PMX)
krzyżowanie z rekombinacją krawędzi, rozszerzone – edge recombination enhanced crossover
krzyżowanie z porządkowaniem – order crossover (OX)
krzyżowanie z wydzielaniem podtras – subtour-chunks crossover
krzyżowanie z wymianą krawędzi – alternating-edges crossover
krzyżowanie proste – simple crossover

Liczby Prüfera – Prüfer numbers
liczebność populacji – population size
locus – locus

Mechanizm odpornej rekrutacji – immune recruitment mechanism
metagenetyczny algorytm – meta genetic algorithm
metody heurystyczne – heuristic methods
metody niszy – niche methods
metody wzrostu – hillclimbing
model elitarny – elitist model
model wartości oczekiwanej elitarny – elitist expected value model
model ze współczynnikiem zatłoczenia – crowding factor model
modele równoległych wysp – parallel island models
modelowanie poznawcze – cognitive modeling
mutacja brzegowa – boundary mutation
mutacja równomierna – uniform mutation
mutacja nierównomierna – non-uniform mutation
mutacja rzędu k – mutation of order k
mutacja z mieszaniem podlist – scramble sublist mutation

Napór selekcyjny – selective pressure
nauczanie empiryczne symboliczne – symbolic empirical learning
nawigacja – navigation

Ograniczenia – constraints
okno skalujące – scaling window
operator cięcia – cut operator
operator krzyżowania dla pakowania pojemników – bin packing crossover operator
operator łączenia – splice operator
operator wielorodzicielski – multi-parent operator
operatory strategii – strategy operators
optymalizacja funkcji – function optimization

optymalizacja wielokryterialna – multiobjective optimization
optymalizacja wielomodalna – multimodal optimization
orgie – orgies
oscylacje strategiczne – strategic oscillation

Pamięć behawioralna – behavioral memory
parametr czasu życia chromosomu – chromosome lifetime parameter
planowanie drogi – path planning
podejście Michigan – Michigan approach
podejście Pitt – Pitt approach
podział – partitioning
ponowne zapoczątkowanie populacji – population reinitialization
populacja początkowa – initial population
program ewolucyjny – evolution program
program rozwiązywania zadań – general problem solver
programowanie ewolucyjne – evolutionary programming
programowanie genetyczne – genetic programming
próbkowanie deterministyczne – deterministic sampling
przecięcie – intersection
przeglądanie genów – gene scanning
przemieszczanie – displacement
przestrzeń rozwiązań – search space
przeszukiwanie losowe – random search
przeszukiwanie rozproszone – scatter search
przeszukiwanie tabu – tabu search
punkty paretooptymalne – Pareto-optimal points

Rekombinacja puli genów – gene pool recombination
relaksacja Lagrange'a – Lagrangian relaxation
reprezentacja ścieżkowa – path representation
reprezentacja porządkowa – ordinal representation
reprezentacja przyległościowa – adjacency representation
reprezentacja z macierzą binarną – binary matrix representation
reprezentacja binarna – binary representation
reprezentacja zmiennopozycyjna – floating point representation
rozgrywanie gier – game playing
rozkład Boltzmanna – Boltzmann distribution
rozwiązania dopuszczalne – feasible solutions
rozwiązania niedopuszczalne – infeasible solutions
równanie wzrostu schematu – reproductive scheme growth equation
różnorodność populacji – population diversity
rząd schematu – order of the schema

Selekcja dynamiczna – dynamic selection
selekcja naturalna – natural selection
selekcja pokoleniowa – generational selection
selekcja porządkowa – ranking selection
selekcja porządkowa liniowa – ranking linear selection
selekcja porządkowa nieliniowa – ranking nonlinear selection
selekcja ruletkowa – roulette wheel selection
selekcja seryjna – serial selection
selekcja statyczna – static selection
selekcja turniejowa – tournament selection
selekcja wygaszająca – extinctive selection
selekcja z uwodzeniem – seduction selection
selekcja zachowująca – preservative selection
sieci neuronowe – neural network
sieć komunikacyjna – communication network
skalowanie – scaling
skalowanie liniowe – linear scaling
skalowanie z obcinaniem na poziomie odchylenia standardowego – sigma truncation scaling
skalowanie zgodne z prawem potęgowym – power law scaling
sprawdzanie parzystości – parity check
sterowanie adaptacyjne – adaptive control
sterowanie dynamiczne – dynamic control
sterowanie optymalne – optimal control
stochastyczne próbkowanie na podstawie reszty bez zamiany – remainder stochastic sampling without replacement
strategia ARGOT – ARGOT strategy
strategie dwuelementowe – two-membered strategies
strategie ewolucyjne – evolution strategies
strategie wieloelementowe – multi-membered strategies
symbol nieistotne – don't care symbol
symulowane wyżarzanie – simulated annealing
system klasyfikujący – classifier system
system logiki zmiennowartościowej – variable valued logic system
systemy samoadaptacyjne – self-adapting systems

systemy współewoluujące – co-evolutionary systems
szeregowanie, harmonogramowanie – scheduling

Średnie dopasowanie populacji – population average fitness
środowisko – environment
środowisko programowe EVA – EVA programming environment

Test podobieństwa – affinity test
turniej stochastyczny – stochastic tournament
twierdzenie Banacha o punkcie stałym – Banach fixpoint theorem
tworzenie gatunków – species formation

Uczenie maszynowe – machine learning
usunięcie – deletion
utrzymywanie dopuszczalności – maintaining feasibility

Warunek zakończenia – termination condition
wewnętrzna równoległość – implicit parallelism
wstawienie – insertion
wygładzanie – smoothing
wymiana – swap

Zadanie komiwojażera – traveling salesman problem
zadanie ładowania kontenerów – pallet loading problem
zadanie maksymalnej kliki – maximum clique problem
zadanie pchania wózka – push-cart problem
zadanie pokrycia zbioru – set covering problem
zadanie spełniania postaci normalnej koniunkcyjnej – CNF-satisfiability problem
zadanie spełniania logicznego – Boolean satisfiability problem (SAT)
zadanie spełniania ograniczeń – constraint satisfaction problem
zadanie transportowe liniowe – linear transportation problem
zadanie transportowe nieliniowe – nonlinear transportation problem
zadanie transportowe zbilansowane – balanced transportation problem
zadanie układania planu lekcji – timetable problem
zadanie zbierania plonów – harvest problem
zapobieganie kazirodztwu – incest prevention
zarządzanie – management science
zawód – deception
zbieżność przedwczesna – premature convergence
zespolenie – union
zmienna długość chromosomu – chromosome variable length
zmora Jenkinsa – Jenkins nightmare

Skorowidz nazwisk

Aarts E.H.L. 95
Abuali F.N. 299
Arabas J. 150
Attia N.F. 168
Axelrod R. 49, 50

Baker J.E. 86, 87
Bäck T. 11, 88
Bean J.C. 185
Beasley D. 203
Beasley J. 300, 301
Belew R. 123, 208, 332
Beltramo M.A. 283, 285, 286
Bersini H. 121
Bertoni A. 81
Bilchev G. 191
Booker L.B. 307
Bowen J. 365
Brindle A. 86
Broyden C.G. 168
Bui T.N. 301, 347
Bull D.R. 203
Burke E.K. 283

Christofides N. 246
Coombs S. 28
Correns K. 40
Craighurst R. 272
Četverikov S.S. 41

Darvin C. 40
Davidor Y. 290
Davis L. 28, 32, 33, 34, 253, 276, 278, 279, 280, 329, 342
Davis T.E. 95
Dawkins R. 372
De Jong K.A. 9, 27, 28, 31, 42, 86, 120, 306, 307
de Vries H. 40
Deb K. 206
Dorigo M. 81
Dozier G. 365

Eiben A.E. 95, 121, 365
Eppley P.H. 301
Esbensen H. 299
Eshelman L.J. 85, 120, 369

Falkenauer E. 287, 289
Fiacco A.V. 168
Fleming P.J. 206
Fogel D.B. 11, 211
Fogel L.J. 25, 321, 375
Fonseca C.M. 206
Fox B.R. 260
Fox M.S. 278
Fuquay D'A. 258

Giordana A. 319
Glover D.E. 31
Glover F. 25, 210, 211, 364
Goldberg D.E. 28, 93, 95, 101, 104, 121, 127, 202, 208, 253, 329, 371
Gorges-Schleuter M. 271
Greene F. 41
Grefenstette J.J. 101, 121, 210, 257, 332, 342
Guan S. 257, 260, 267

Hadj-Alouane A.B. 185
Heitkötter J. 13
Hinterding R. 150
Hoffmeister F. 88, 335
Holland J.H. 25, 28, 29, 81, 138, 256, 307
Homaifar A. 174, 257, 260, 267
Houck C.R. 175
Husbands P. 280

Ibaraki T. 303

Janikow C. 18, 305, 307
Jenkins F. 40
Joines J.A. 175
Jones A.J. 272
Jones D.R. 276, 283, 285, 286
Jones T. 303
Juliff K. 302

Karp R.M. 246, 272
Keane A. 190
Kershenbaum A. 298, 363
Kingdon J. 96
Koza J.R. 25, 29, 323, 324, 325
Krämer O. 271

Le Riche R. 187, 370
Lidd M.L. 248

Lingle R. 253
Litke J.D. 246, 267
Luchian H. 283

Mahfoud S.W. 205
Mahomet 32
Maniezzo V. 211
Martin R.R. 177
Martin W. 272
Männer R. 373
McCormick G.P. 168
McMahon M.B. 260, 278
Mendel G. 40, 41
Mill F. 280
Moon B.R. 347
Morgan T. 40
Morishima A. 119
Murray W. 168
Musseli M. 211
Mühlenbein H. 92, 120, 271, 285, 371

Ng K.P. 40
Nissen V. 302

Oliver I.M. 254

Padberg M. 272
Paechter B. 283
Palmer C.C. 298, 363
Paredis J. 187, 366, 370
Patnaik L.M. 151
Petriuc M. 283
Powell D. 177, 357
Principe J.C. 95

Radcliffe N.J. 330
Rawlins G. 278
Rechenberg I. 18, 25, 192, 194
Reitman J.S. 307
Renders J.-M. 121
Reynolds R.G. 188
Richardson J. 202, 353
Ridella S. 211
Rinaldi G. 272
Ronald E. 347
Rudolph G. 96

Saitta L. 319
Schaffer J.D. 85, 119, 206
Schoenauer M. 175, 363
Schraudolph N. 123, 208, 332, 342
Schwefel H.-P. 25, 42, 192, 194, 196, 335, 373
Segrest P. 95
Seniw D. 260, 263
Shaefer C.G. 209, 370
Skolnick M.M. 177, 357
Smith A. 185
Smith R.E. 101
Smith S.F. 307, 313, 314
Spears W.M. 119, 120, 204
Srinivas M. 151
Srinivas N. 206
Starkweather T. 258
Steele J.M. 272
Steenstrup M. 29
Surry P.D. 187
Syswerda G. 120, 255, 278, 279

Tate D. 185

Valenzuela C.L. 272
Van Hee K.M. 95
Voigt H.-M. 120
von Laszewski 283
von Tschermak K. 40

Warrington S. 280
Whitley D. 85, 123, 207, 258, 372
Wong K.C. 41

Xanthakis S. 175, 363
Xiao J. 357

Yagiura Y. 303

Skorowidz rzeczowy

Algorytm delta-kodowania 23, 150, 207
– genetyczny 39
– – a strategia ewolucyjna 197
– –, funkcja oceny 46, 63
– –, krzyżowanie 47
– –, liczebność populacji 59, 100
– –, mutacja 47
– –, nieporządny 122, 370
– –, odcinkowy 150, 211
– –, operatory 43, 47, 121
– –, parametry 44, 48, 121
– –, prosty 104
– –, przykład 53
– –, reprezentacja 29, 44, 45, 59, 127
– – – bezpośrednia 281
– – – binarna 127
– – – pośrednia 281
– – – zmiennopozycyjna 127
– –, struktura 43
– – ze zmienną liczebnością populacji 100
– – zmodyfikowany 89
– – z odwzorowaniem zwężającym 95
– – z ustalonym stanem 92
– Lina-Kernighana 246
algorytmy genetyczne płodzące 92, 368
– – równoległe 371
– – – hybrydowe 372
– kulturowe 187, 372
– naprawy 52, 359
alele 41
ARGOT, strategia 150, 209
arytmetyczne krzyżowanie 144, 160, 200, 361
automaty skończone 322

Badanie 41
Baldwina efekt 359
Banacha twierdzenie o punkcie stałym 96, 97
baza danych 41
– –, optymalizacja obsługi zapytań 41
Boltzmanna rozkład 55
– selekcja 88

Ciąg Cauchy'ego 98
chromosom 41
–, parametr czasu życia 102
–, wiek 102
–, zmienna długość 313

Dekodery 41, 362
delta-kodowania algorytm 123, 150, 207
diploidalność 41, 371
dominacja 41
dopasowanie 44
dylemat więźnia 49
dynamiczne kodowanie parametrów 123, 150, 208, 370
drzewo Steinera 299
dziedziczność 40
„dziel i zwyciężaj" 273

Efekt Baldwina 359
epistaza 82
EVA, środowisko programowe 35, 344
ewolucja Lamarckiana 359
– ziarnista 150, 211

Fenotyp 41
funkcja oceny 46, 350
funkcje definiowane automatycznie 325
– kary 360
– królewskiej drogi 369

GAMS 140, 332
gen 41
GENESIS 95, 137, 206, 332, 342
GENETIC-1 223
–, funkcja oceny 222
–, operatory 222
–, rozpoczęcie 221
GENETIC-2 227, 249
–, funkcja oceny 224, 232
–, operatory 224, 232
–, reprezentacja 223, 232
–, rozpoczęcie 224, 232
genetyczne programowanie 25, 323, 349
– uczenie indukcyjne 304, 315
GENITOR 342
GENOCOP 155, 201
– II 167
– III 188
genotyp 41
Graya kody 128

Hamminga klif 137
haploidalność 41
harmonogramowanie 41, 274
hiperpłaszczyzna 71
hipoteza o blokach budujących 80
historia genetyki 40
hybrydowe algorytmy genetyczne 33, 372

Inwersja 82, 257
inżynieria przemysłowa 302

Kara śmierci 177, 358
klasyfikujący system 92, 307
klif Hamminga 137
klucze losowe 186
kody Graya 128
korelacja odległości dopasowań 369
korzystanie z wcześniejszych rozwiązań 41
krzyżowanie 43, 65, 76
– arytmetyczne 144, 160, 200, 361
– bazujące na porządku 255
– – – pozycjach 256
– CX 252, 254
– dwupunktowe 119
– gwarantowane średnie 160, 200
– heurystyczne 162, 249
– jednopunktowe 104, 119
– jednorodne 120
– macierzowe (MX) 267
– odcinkowe 119
– OX 252, 253, 347
– PMX 223, 253, 286, 347
– pośrednie 160
– prawdopodobieństwo 60
– proste 144, 161
– przecięcie 261
– simpleksowe 121
– strukturalne 284
– tasowane 119
– wielopunktowe 119
–, zespolenie 261
– z rekombinacją krawędzi 258, 347
– – – – rozszerzone 259, 347
– z wydzielaniem podtras 249
– z wymianą krawędzi 249
kulturowe algorytmy 187, 372

Lagrange'a relaksacja 185
liczby Prüfera 299
Lina-Kernighana algorytm 246
locus 41

Łańcuchy Markowa 95

Mechanizm odpornej rekrutacji 150, 209
metaalgorytm genetyczny 121
metody heurystyczne 345
– niszy 202
– wzrostu 41, 42, 53
Michigan, podejście 307
modelowanie poznawcze 41
modele równoległych wysp 371
mutacja 44, 67, 78
– brzegowa 159
– częstość 44, 47
– dni 282
– nierównomierna 133, 143, 160
–, prawdopodobieństwo 60
– równomierna 143, 159
– rzędu *k* 282
– strukturalna 284
–, usunięcie 296
–, wstawienie 257, 296
–, wygładzanie 296
–, wymiana 297
– z mieszaniem podlist 279

Napór selekcyjny 85
naprawa, algorytmy 52, 359

nawigacja 289
nielosowy dobór 369
nisze, metody 203
NSGA 206

Ograniczenia 30
–, algorymy naprawy 113, 189
–, funkcje kary 112, 174
–, liniowe 155
–, nieliniowe 167, 174, 188
OOGA 342
operator cięcia 122
– krzyżowania dla pakowania pojemników 288
– łączenia 122
– wielorodzicielski 121
optymalizacja funkcji 44, 368
– wielomodalna 202
– wielokryterialna 205
orgie 121
oscylacje strategiczne 363

Pamięć behawioralna 363
Pitt, podejście 307, 312
planowanie drogi 289
podejście Michigan 307
– Pitt 307, 312
podział 283
populacja, liczebność 59, 100
– początkowa 44, 46, 59, 341
–, ponowne zapoczątkowanie 280
–, różnorodność 85
–, średnie dopasowanie 74
PROBIOL 344
program ewolucyjny 25, 29, 326
–, rozwiązywania zadań 31
programowanie ewolucyjne 25, 150, 211, 321, 348, 362
– genetyczne 25, 323
projektowanie VLSI 39
prowadzenie kabla 41
Prüfera liczby 299
przedwczesna zbieżność 84
przeglądanie genów 121
przestrzeń rozwiązań 350
przeszukiwanie losowe 41
– rozproszone 25, 121, 210, 349
– tabu 210
punkty paretooptymalne 205, 206

REGAL 319
reguła sukcesu 1/5
– zamiany 15% 360
– – 5% 112, 115
rekombinacja puli genów 121
relaksacja Lagrange'a 185
rozgrywanie gier 41
rozkład Boltzmanna 55
rozwiązania dopuszczalne 350
– niedopuszczalne 350
równanie wzrostu schematu 75, 78, 79
równoległe masowe algorytmy genetyczne 371

Schemat 71
–, długość definiująca 72
–, dopasowanie 74
–, rząd 72
selekcja 59
– Boltzmanna 88
–, deterministyczne próbkowanie 86
– dynamiczna 88
–, elitarny model 86, 89
–, – – wartości oczekiwanej 57
–, – – ze współczynnikiem zatłoczenia 57
– naturalna 40
– pokoleniowa 89
– porządkowa 87
– – liniowa 87
– – nieliniowa 87
– ruletkowa 59
– seryjna 209
– statyczna 88
–, stochastyczne próbkowanie na podstawie reszty 86
–, – – uniwersalne 86, 130
– turniejowa 88
–, uwodzenie 347
– wygaszająca 89
– zachowująca 89
sieci neuronowe 28
sieć komunikacyjna 28
skalowanie liniowe 93
–, okno 94
– z obcinaniem na poziomie odchylenia standardowego 93
– zgodne z prawem potęgowym 93
sprawdzanie parzystości 322
Steinera drzewo 299
sterowanie adaptacyjne 41
– dynamiczne 130
– optymalne 41
– –, zadanie zbierania plonów 140, 145, 147
– –, – liniowo-kwadratowe 140, 145, 146
– –, – pchania wózka 141, 145, 147
strategia ARGOT 150, 209
strategie ewolucyjne 25, 193, 349
– – a algorytmy genetyczne 197

strategie dwuelementowe 194
, operatory 196
wieloelementowe 195
– $\mu + \lambda$ 195
– (μ, λ) 195
symboliczne nauczanie empiryczne 304
symbol nieistotne 71
symulowane wyżarzanie 43, 53, 55, 95
system klasyfikujący 92, 307
– logiki zmiennowartościowej 306
systemy samoadaptacyjne 369
– współewoluujące 370

Środowisko 44
– programowe EVA 35, 344

Test podobieństwa 209
twierdzenie Banacha o punkcie stałym 96, 97
– o schematach 80
tworzenie gatunków 202

Uczenie maszynowe 304
utrzymywanie dopuszczalności 361

VEGA 206

Warunek zakończenia 94
wewnętrzna równoległość 81, 127
współewoluujące systemy 370

Zadanie komiwojażera 31, 39, 41, 51, 245
– ładowania kontenerów 302
– maksymalnej kliki 301
– pokrycia zbioru 300
–, reprezentacja porządkowa 250
–, – przyległościowa 248
–, – ścieżkowa 252
– spełniania formy normalnej koniunkcyjnej 246
– – logicznego 57, 353
– – ograniczeń 365
– transportowe 41, 215, 234, 330
– – liniowe 215
– – nieliniowe 231
– – zbilansowane 216
– układania planu lekcji 281
– z macierzą binarną 260, 263, 267
zapobieganie kazirodztwu 85, 272
zarządzanie 302
zawód 81, 369
zmora Jenkinsa 40

WNT Warszawa 1999
Wydanie II
Ark. wyd. 30,0. Ark. druk. 27,0
Symbol Et/83460/MEN
Druk i oprawa: Łódzkie Zakłady Graficzne