Prace Naukowe
Wydziałowego Zakładu
Informatyki
Politechniki Wrocławskiej

SZTUCZNA INTELIGENCJA

Nr 1

ALGORYTMY EWOLUCYJNE – PRZYKŁADY ZASTOSOWAŃ

Redakcja naukowa

Halina Kwaśnicka

Oficyna Wydawnicza
Politechniki Wrocławskiej

Prace Naukowe
Wydziałowego Zakładu
Informatyki
Politechniki Wrocławskiej

SZTUCZNA INTELIGENCJA

Nr 1

ALGORYTMY EWOLUCYJNE – PRZYKŁADY ZASTOSOWAŃ

Redakcja naukowa

Halina Kwaśnicka

Oficyna Wydawnicza Politechniki Wrocławskiej
Wrocław 2002

Wydziałowy Zakład Informatyki

Wydział Informatyki i zarządzania
Politechniki Wrocławskiej

Wybrzeże Wyspiańskiego 27
50-370 Wrocław

Recenzet

Artur Chorążyczewski

Opracowanie redakcyjne materiałów
Halina Kwaśnicka

Projekt okładki
Halina Kwaśnicka

OFICYNA WYDAWNICZA POLITECHNIKI WROCŁAWSKIEJ
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław

ISBN xx-xxxx-xxx-x

Drukarnia Oficyny Wydawniczej Politechniki Wrocławskiej. Zam. nr xxxxxxx

SPIS TREŚCI

CONTENTS

Nie ma takiej przeciwności losu, z której nie można by uczynić błogosławieństwa, jeśli przywoła się dobór naturalny.

R.C.Lewontin, *Geny, środowisko, i organizmy* w *Ukryte teorie nauki*, O. Sacks, J. Miller i in., wyd. ZNAK, Kraków 1996.

WSTĘP

Koło Naukowe Sztucznej Inteligencji CJANT przy Wydziałowym Zakładzie Informatyki na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej ukonstytuowało się w czerwcu 2000 roku, a działalność rozpoczęło właściwie dopiero jesienią. Podstawowym celem Koła jest włączanie studentów Inżynierii Oprogramowania do pracy naukowej w dziedzinie szeroko rozumianej Sztucznej Inteligencji. Efekty ich prac powinny być prezentowane na konferencjach naukowych, na których bywają również doświadczeni pracownicy naukowi oraz publikowane w materiałach konferencyjnych. Niezależnie od tej formy prezentacji wyników planujemy wydawać cyklicznie (w odstępach rocznych) Zeszyty Naukowe. Ponieważ chcemy, by każdy numer zeszytu był w miarę możliwości monotematyczny, to kolejne numery będą przekazywane do druku, kiedy zgromadzimy kilka interesujących prac o podobnej tematyce. Zależy nam, by w Zeszytach prezentować głównie efekty własnych, oryginalnych prac. Jednocześnie chcemy, by wydawane Zeszyty były użyteczne dla naszych młodszych studentów, bądź studentów innych kierunków, dlatego też każdy numer Zeszytu będzie zawierał artykuł wprowadzający w specyfikę dziedziny, której dotyczy. W szczególności będziemy publikować też prace przeglądowe, pokazujące aktualny stan wybranych zagadnień (zastosowań), będziemy starać się to czynić dla takich dziedzin i/lub zastosowań, które są nowe, brakuje literatury polskojęzycznej na ten temat i – według naszej opinii – tematyka ta spotka się z zainteresowaniem młodych czytelników.

Inauguracyjny Zeszyt poświęcony jest zastosowaniom algorytmów genetycznych. Pierwszy artykuł jest wprowadzeniem w dziedzinę obliczeń ewolucyjnych. Daje pogląd na temat algorytmów ewolucyjnych, zapoznaje dokładnie z algorytmami genetycznymi. Intencją edytora jest, by potencjalny czytelnik mógł zrozumieć pozostałe artykuły bez konieczności wcześniejszego studiowania dziedziny – to właśnie lektura Zeszytu powinna zachęcić go do sięgnięcia po jedną z dostępnych książek.

Drugi artykuł pokazuje jak można zastosować algorytm genetyczny do uproszczonego zadania z silnymi ograniczeniami – szukanie bezkolizyjnych połączeń zadanych punktów na płaszczyźnie (planowanie ścieżek na płytkach drukowanych).

Autor pokazuje, jak duże znaczenie ma dobór właściwej funkcji oceniającej potencjalne rozwiązania.

Tematyce zadań z silnymi ograniczeniami poświęcony jest też trzeci artykuł. Autor przedstawia krótki przegląd metod rozwiązywania problemów układania planów (lekcji, egzaminów), po czym pokazuje efekty własnej pracy – prezentuje sposób układania harmonogramu dyżurów w szpitalu (dane dla rzeczywistego szpitala) z wykorzystaniem algorytmu genetycznego.

Następny, czwarty artykuł analizuje możliwość wykorzystania algorytmów ewolucyjnych (programowania genetycznego połączonego z algorytmem genetycznym) do wydobywania wiedzy z baz danych. Zaproponowane podejście testowane jest na specjalnie wygenerowanych testowych bazach danych, po weryfikacji podejście to zastosowane jest do baz zawierających dane historyczne o aktywności słonecznej. Zadanie polega na przewidywaniu liczby plam na słońcu.

Bardzo praktyczne zastosowanie algorytmów genetycznych pokazane jest w piątym artykule. Zadanie polega na znalezienia optymalnego (równomiernego) rozkładu jazdy tramwajów we Wrocławiu. Oczywiście, muszą być spełnione pewne dodatkowe ograniczenia, np. mijanie się tramwajów na pojedynczych torach. Autor wykorzystał algorytmy genetyczne jako narzędzie do rozwiązania tego problemu. Przetestował również, jak zaproponowany algorytm daje sobie radę z rozkładem jazdy w Poznaniu.

Autorzy kolejnych dwóch artykułów to hobbyści, którzy próbowali uczynić z algorytmu genetycznego wielkiego kompozytora. Nie byli oni pierwsi w swoich próbach, próbowali to – z różnym skutkiem – inni zwolennicy muzyki i algorytmów genetycznych. Efekty ich prac, a właściwie zaprojektowanego przez autorów algorytmu genetycznego, można tylko częściowo poznać czytając artykuł szósty i siódmy. Z niektórymi dziełami genetycznego kompozytora można zapoznać się po zajrzeniu na strony internetowe autorów.

Następne numery Zeszytów będą poruszać takie problemy jak:

- Systemy łączące sieci neuronowe i algorytmy genetyczne,
- Ekstrakcja reguł z sieci neuronowych,
- Połączenie algorytmów genetycznych i logiki rozmytej,
- Metody rozwiązywania problemów wzorowane na naturze,
- Przetwarzanie wiedzy niepewnej – zbiory przybliżone,
- Metody pozyskiwania wiedzy z baz danych,

Autorzy składają serdeczne podziękowania recenzentowi, dr inż. Arturowi Chorążyczewskiemu za wnikliwą recenzję i wszystkie przekazane uwagi. Zostały one wzięte pod uwagę podczas redakcji końcowej wersji artykułów.

Halina Kwaśnicka

Obliczenia ewolucyjne

Halina Kwaśnicka
Wydziałowy Zakład Informatyki, Politechnika Wrocławska
E-mail: kwasnicka@ci.pwr.wroc.pl, http://www.ci.pwr.wroc.pl/~kwasnick

Streszczenie

Praca wprowadza w tematykę algorytmów ewolucyjnych. Omówione są etapy stosowania algorytmów genetycznych do rozwiązywania problemów optymalizacyjnych. Prosty przykład ułatwi zrozumienie metody. W dalszej części przybliżone są inne rodzaje algorytmów ewolucyjnych – programowanie genetyczne, strategie ewolucyjne i programowanie ewolucyjne.

Wprowadzenie

Obliczenia ewolucyjne to dział sztucznej inteligencji [12],[13],[15] – dziedziny stosunkowo młodej, budzącej wiele kontrowersji. Spotykane w literaturze przedmiotu różne jej definicje sprowadzają się do określenia, że jest to próba modelowania aspektów ludzkiego rozumowania za pomocą komputerów, albo próba rozwiązywania za pomocą komputerów takich problemów, które są zwykle rozwiązywane przez (inteligentnego) człowieka [4],[13],[18].

Wydaje się jednak, że coraz bardziej upowszechnia się rozumienie sztucznej inteligencji w sposób sformułowany przez D.B. Fogla w przedmowie do jego książki [4]: sztuczna inteligencja to zdolność systemu do dostosowania swojego działania tak, aby osiągnąć założony cel w środowisku, w którym się znajduje. Inteligentne stworzenia powstały w wyniku ewolucji biologicznej. Obserwując ją i modelując możemy uzyskać wiele inteligentnych zachowań. Wszelkie metody symulacji ewolucji z wykorzystaniem komputera noszą nazwę *Obliczeń ewolucyjnych* (**EC**, ang. ***E**volutionary **C**omputation*), natomiast algorytmy stosowane w takich symulacjach, to *Algorytmy ewolucyjne* (**EA**, ang. ***E**volutionary **A**lgorithms*).

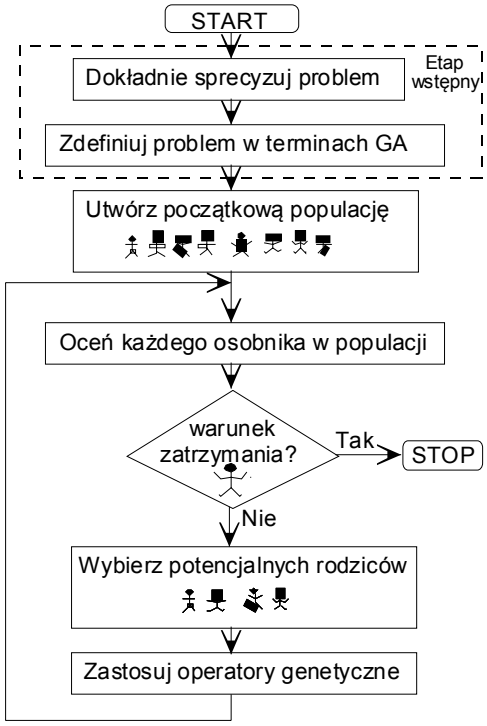
W ostatnich latach, paradygmat obliczeń ewolucyjnych stał się bardzo popularny. Obserwowany jest ogromny wzrost prac na ten temat, czasopism, konferencji, książek, powstają listy dyskusyjne, strony internetowe. Podobnie zwiększa się liczba różnych dziedzin, w których znajdują zastosowanie algorytmy ewolucyjne: od naturalnych dla nich zadań modelowania dynamiki populacji, poprzez zastosowania czysto techniczne (np. projektowanie samolotów, gazociągu), zadania szeregowania, gry logiczne, nauki chemiczne i fizyczne, po nauki ekonomiczne – popularny ostatnio paradygmat ekonomii ewolucyjnej.

Próbując naśladować ewolucję biologiczną opracowano kilka algorytmów, różniących się sposobem reprezentacji potencjalnego rozwiązania czy też

stosowanymi operatorami genetycznymi. Wszystkie one są stosowane jako metody optymalizacji, naśladujące główne czynniki ewolucji biologicznej, tj. selekcję naturalną i reprodukcję. Należy jednak pamiętać, że nie dają one gwarancji znalezienia optimum globalnego, są użyteczne tam, gdzie nie można zastosować innej metody (np. gradientowej) i gdzie wystarczy nam znalezienie rozwiązania satysfakcjonującego.

1. Algorytm Genetyczny

Algorytm genetyczny (**GA**, ang. ***G**enetic **A**lgorithm*) to ewolucja sztucznych „*osobników*", z których każdy jest zakodowanym potencjalnym rozwiązaniem rozpatrywanego problemu. Osobniki te ewoluują w sztucznym środowisku, ponieważ należą do tego samego gatunku (kodują rozwiązanie tego samego zadania) to konkurują o zasoby w tym środowisku. Podobnie jak w biologii, o szansach przeżycia i wydania potomstwa przez sztucznych osobników decyduje „*dobór naturalny*" – im lepiej jest przystosowany osobnik do danego środowiska, tym większe ma szanse przeżyć i wydać potomstwo. Potomstwo – podobnie jak w naturze – różni się od swoich rodziców dzięki działaniu specjalnie zaprojektowanych, wzorowanych na naturze, *operatorów genetycznych* (mutacji i krzyżowania). *Środowisko* odzwierciedla rozwiązywane przez algorytm genetyczny zadanie, zatem *ocena przystosowania* osobnika jest oceną jakości kodowanego przez niego rozwiązania. W ten sposób, w kolejnych pokoleniach populacji sztucznej osobników zaczynają dominować coraz to lepsze rozwiązania. Najlepszy osobnik, po rozkodowaniu, jest szukanym rozwiązaniem. W ostatnich latach algorytmy genetyczne stały się bardzo popularne jako narzędzie optymalizacyjne [7],[14].



Rysunek 1. Etapy w stosowaniu algorytmów genetycznych – ogólny schemat

Ewolucja populacji jest procesem przeszukiwania przestrzeni potencjalnych

rozwiązań. W procesach takich jest ważne zachowanie równowagi pomiędzy przekazywaniem najlepszych cech do następnego pokolenia, czyli wykorzystaniem dotychczas znalezionych „obiecujących" rozwiązań (ang. *exploiting*) z jednej strony, a szerokim przeszukiwaniem przestrzeni (ang. *exploring*) z drugiej strony. Algorytm genetyczny umożliwia zachowanie takiej równowagi.

Czynności niezbędne przy stosowaniu **GA** do rozwiązania rzeczywistego pokazane są na rysunku 1. Dwa pierwsze etapy stanowią etap wstępny i muszą być wykonane 'ręcznie'. Ich realizacja często stwarza duże trudności, zwłaszcza osobom niedoświadczonym. Pozostałe etapy algorytmu wykonuje program komputerowy, można do tego wykonać własną implementację **GA** lub też wykorzystać komercyjne, bądź dostępne w sieci programy.

Etap wstępny to dokładne sprecyzowanie problemu. Obejmuje takie czynności, jak ustalenie zakresu zmienności i dokładności rozwiązań, zdefiniowanie ograniczeń, sposobu kodowania, itp. Opis pojedynczego osobnika (zakodowane rozwiązanie) nazywany jest *chromosomem* lub *genotypem*, wartość *optymalizowanej funkcji* obliczona dla danego osobnika to jego *przystosowanie* (*fitness*). Od przystosowania osobnika zależy liczba jego potomków w następnym pokoleniu.

Początkową populację (chromosomy) można wybrać losowo, zwłaszcza jeśli brak jest jakichkolwiek przesłanek odnośnie dobrych rozwiązań. Jako warunek zatrzymania można ustalić:

- sprawdzenie, czy zadowalające rozwiązanie istnieje w aktualnej populacji – może to wykonać automatycznie program, lub analizujący prezentowane rozwiązania użytkownik,
- zadana liczba iteracji (wykonanych pokoleń),
- niemożność znalezienia lepszego rozwiązania przez zadaną liczbę pokoleń (np. jeśli minie 100 pokoleń bez poprawy najlepszego rozwiązania).

Tworzenie nowego pokolenia populacji jest procesem złożonym. Najpierw dokonuje się selekcji 'dobrych' rozwiązań i tworzy ich kopie (wybór rodziców). Podczas tworzenia kopii działają operatory genetyczne powodujące zróżnicowanie między osobnikami, najczęściej są to krzyżowanie i mutacja.

Niektóre etapy wyjaśnimy nieco szerzej i pokażemy działanie **GA** na prostym przykładzie.

1.1 Kodowanie rozwiązań (definiowanie chromosomu)

Potencjalne rozwiązanie jest kodowane do postaci tzw. genotypu (patrz rysunek 2). W klasycznym algorytmie genetycznym stosuje się kodowanie binarne (choć nie jest to konieczne i można stosować inny od binarnego alfabet). Aby zakodować binarnie wartości argumentów optymalizowanej funkcji musimy wiedzieć, ile należy przeznaczyć na nie bitów. Załóżmy, że optymalizowana funkcja ma m zmiennych (osobnik ma m fenów): $[x_1,...,x_m]$. Na poziomie genotypowym osobnik opisany jest przez łańcuch l bitów [011...01]. Liczba bitów (l_i) wymagana do reprezentacji (kodowania) pojedynczego fenu x_i może być wyliczona ze wzoru:

$$(b-a)\cdot 10^k \le 2^{l_i-1}, \tag{1}$$

gdzie: $[a, b]$ – przedział dopuszczalnych wartości dla i-tej zmiennej (x_i),
k – założona dokładność reprezentacji fenu (k miejsc po przecinku),
l_i – liczba wymaganych bitów, jest to najmniejsza liczba satysfakcjonująca powyższą nierówność.

Całkowita liczba bitów l potrzebnych do reprezentacji jednego osobnika jest sumą wymaganych długości dla wszystkich fenów:

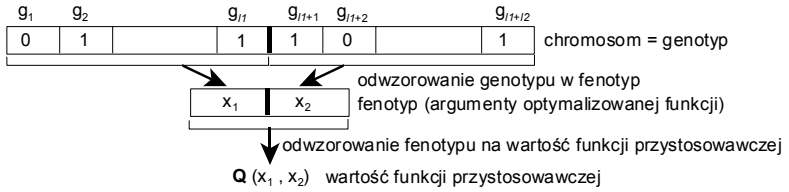
$$l = \sum_{i=1}^{m} l_i . \tag{2}$$

Aby policzyć przystosowanie osobnika, należy zdekodować jego genotyp. Wartość zakodowanej zmiennej x_i jest liczona ze wzoru:

$$x_i = a + (01011...11)_{10} \cdot \frac{b-a}{2^{l_i}-1}, \tag{3}$$

gdzie: $(0101...1)_{10}$ – dziesiętna wartość łańcucha binarnego reprezentującego zmienną x_i, traktowanego jako liczba zapisana w kodzie dwójkowym, pozostałe oznaczenia jak wyżej.[1]

Przykład reprezentacji osobnika (dla funkcji dwuwymiarowej) w prostym algorytmie genetycznym pokazany jest na rysunku 2.



Rysunek 2. Reprezentacja binarna osobnika w prostym algorytmie genetycznym

1.2 Przekształcanie funkcji celu na funkcję przystosowania

Czasami funkcję, którą chcemy optymalizować musimy przekształcić, aby można ją było wykorzystać jako funkcję przystosowania w **GA**. Dzieje się tak, gdy:

[1] Czasem lepsze efekty daje kod Graya – kolejne liczby całkowite różnią się od siebie wartością jednego bitu, np.: 0000 → 0; 0001 → 1; 0011 → 2; 0010 → 3; 0110 → 4; 0111 → 5; 0101 → 6; 0100 → 7; 1100 → 8, itd.

- Na argumenty optymalizowanej funkcji *f* nałożone są ograniczenia, tzn. że poszczególne x_i mogą przyjmować wartości z ograniczonego przedziału $[x_i^{\min}, x_i^{\max}]$.

GA może wyprodukować rozwiązanie nie mieszczące się w zadanych granicach. Prostą i skuteczną metodą jest nakładanie kar na przystosowanie osobnika za przekroczenie dopuszczalnych wartości. Kara może być różna dla poszczególnych x_i oraz może zależeć od stopnia przekroczenia zakresu. Postać funkcji przystosowania *Q* wyraża się wzorem:

$$Q(x_1^o,...,x_n^o) = f(x_1^o,...,x_n^o) - k_i \cdot \sum_{i=1}^{n} Kara(x_i^o), \quad \textbf{(4)}$$

gdzie: k_i – współczynnik określony dla *i*-tego fenu (argumentu),

$(x_1^o,...,x_n^o)$ – zdekodowany chromosom ocenianego osobnika (*o*),

$Kara(x_i^o)$ – kara przy wartości $x_i = x_i^o$ nie mieszczącej się w przedziale $[x_i^{\min}, x_i^{\max}]$

Zazwyczaj:

$$Kara(x_i^o) = \begin{cases} (x_i^o - x_i^{\max}) \cdot d_i, & dla \quad x_i^o > x_i^{\max} \\ 0, & dla \quad x_i^{\min} \le x_i^{o_l} \le x_i^{\max} \\ (x_i^{\min} - x_i^o) \cdot d_i, & dla \quad x_i^{\min} < x_i^o \end{cases}, \quad \textbf{(5)}$$

gdzie: d_i jest karą za przekroczenie *i*-tego argumentu o jednostkę.

- Algorytm genetyczny może produkować niepoprawne, nie dające się ocenić rozwiązania.

Przykładem takiego zadania jest automatyczne projektowanie sieci neuronowych, gdzie w trakcie ewolucji mogą powstawać wręcz rozwiązania nie mające sensu, np. sieci zawierające neurony, które nie mają połączeń wejściowych. Taka sieć nie jest w stanie przetwarzać żadnej informacji.[2] Możliwe są dwie strategie:

- *Usuwanie niepoprawnych osobników z populacji i generowanie na ich miejsce nowych*

Zwykle jest to zbyt czasochłonny sposób, niedopuszczalne osobniki mogą powstawać stosunkowo często i wtedy praca procesora w dużej mierze jest marnotrawiona.

- *Naprawianie powstających „degeneratów"*

Z reguły zadania, w których występują te problemy są na tyle skomplikowane, że opłaca się (w sensie czasu pracy procesora) naprawiać uszkodzone rozwiązania a nie eliminować je i generować na ich miejsce nowe [10],[11].

[2] Oczywiście, idealnym rozwiązaniem byłoby zastosowanie takich reprezentacji rozwiązań i/lub takich operatorów, które nie mogą wytwarzać niepoprawnych rozwiązań, np. specjalne operatory genetyczne dla zadań sekwencyjnych (zadanie komiwojażera), reprezentacja sieci neuronowych w postaci formuł gramatycznych [12].

- Funkcja celu ma być minimalizowana lub przyjmuje ujemne wartości

Ewolucja ‘dąży’ do powstawania coraz to bardziej przystosowanych osobników, zatem naturalna skłonność **GA** to maksymalizacja przystosowania. W wielu zadaniach należy minimalizować funkcję celu. W tej sytuacji za funkcję przystosowania Q możemy przyjąć różnicę pomiędzy ustaloną, dostatecznie dużą, stałą wartością MAX, a naszą funkcją celu $f(x_1,x_2,...,x_n)$:

$$Q = \begin{cases} MAX - f(x_1,...,x_n), & dla\, f(x_1,...,x_n) < MAX \\ 0, & dla\, f(x_1,...,x_n) \geq MAX \end{cases} . \qquad \textbf{(6)}$$

Jeśli funkcja celu $f(x)$ przyjmuje ujemne wartości, to należy znaleźć możliwie małą stałą MIN, która dodana do $f(x)$ da wynik większy od zera:

$$Q = \begin{cases} MIN + f(x_1,...,x_n), & dla\, f(x_1,...,x_n) + MIN > 0 \\ 0, & \text{w innych przypadkach} \end{cases} . \qquad \textbf{(7)}$$

1.3 Selekcja osobników do reprodukcji

Można stosować różne metody selekcji, niemniej wszystkie one muszą mieć jedną wspólną cechę: lepszy osobnik musi mieć większe szanse na posiadanie potomstwa, zgodnie z zasadą naturalnej selekcji ‘*przeżywa najlepszy*’. Jeśli metoda promuje mocno osobniki najlepsze nie dopuszczając słabszych do ‘rozrodu’, to mówimy, że jest to tzw. *twarda selekcja*. Metoda promująca lepsze osobniki, ale umożliwiająca rozród – z mniejszym prawdopodobieństwem – również osobnikom słabszym jest tzw. *miękką selekcją*. Najbardziej popularne metody selekcji, to:

- *Metoda ruletki* (nazywana również metodą *stochastyczną z powtórzeniami*)

Wartości przystosowania wszystkich osobników w populacji są sumowane, suma stanowi całe koło ruletki. Następnie każdemu osobnikowi przypisywany jest wycinek koła proporcjonalny do jego przystosowania. Koło ruletki jest ‘obracane’ (wybierana jest losowo liczba) i wybierany jest osobnik odpowiadający temu sektorowi na ruletce, w którym mieści się wylosowana liczba. Czynność losowania powtarza się N razy (N – rozmiar populacji).

- *Metoda próbkowania deterministycznego*

Wartość oczekiwana liczby potomków dla każdego osobnika liczona jest ze wzoru:

$$E_i = N \cdot pr_i, \qquad \textbf{(8)}$$

gdzie: E_i jest oczekiwaną liczbą potomków i-tego osobnika,

N – liczba osobników w populacji,

pr_i – prawdopodobieństwo wybrania i-tego osobnika do reprodukcji, wynosi:

$$pr_i = \frac{f_i}{\sum_{i=1}^{N} f_j}, \qquad \textbf{(9)}$$

f_j jest wartością funkcji przystosowania (fitness) j-tego osobnika.

Podstawiając równanie **(9)** do **(8)** otrzymujemy, że oczekiwana liczba liczby potomków i-tego osobnika jest równa stosunkowi przystosowania tego osobnika do średniego przystosowania osobników w populacji:

$$E_i = N \cdot pr_i = N \cdot \frac{f_i}{\sum_{j=1}^{N} f_j} = \frac{f_i}{\frac{1}{N}\sum_{j=1}^{N} f_j} = \frac{f_i}{f_{śr}}, \qquad \textbf{(10)}$$

Każdy osobnik z populacji ma tylu potomków, ile wynosi część całkowita wartości oczekiwanej E_i. Następnie osobniki są szeregowane według ułamkowych części E_i (malejąco). Do reprodukcji dobierane są osobniki z początku tej listy, w liczbie potrzebnej do zachowania stałego rozmiaru populacji.

- *Metoda stochastyczna według reszt z powtórzeniami*

Metoda ta jest podobna do poprzedniej, opiera się na liczeniu wartości oczekiwanej liczby potomków dla każdego osobnika z populacji i przydzieleniu mu całkowitej części E_i. Części ułamkowe E_i są wykorzystywane do stworzenia koła ruletki, za pomocą którego wybierane są pozostałe osobniki do reprodukcji.

- *Metoda stochastyczna według reszt bez powtórzeń*

Analogicznie jak w poprzednich dwóch metodach, część całkowita E_i stanowi liczbę potomków i-tego osobnika, natomiast część ułamkowa jest traktowana jako prawdopodobieństwo dla rozkładu Bernoulli'ego, brakujące osobniki są losowane zgodnie z tymi prawdopodobieństwami.

- *Metoda turniejowa*

Wybierane są dwa osobniki (można stosować turniej więcej niż dwóch osobników) i do reprodukcji wybierany jest najlepszy osobnik spośród biorących udział w turnieju. Czasami stosuje się wybór osobników do turnieju za pomocą metody ruletki, co powoduje silniejsze promwanie lepszych osobników (selekcja jest bardziej twarda).

- *Metoda rankingowa (nadawania rang)*

Osobniki w populacji są porządkowane malejąco według wartości funkcji celu. Liczba potomków osobnika zależy od jego *rangi*, przy czym ranga osobnika to jego miejsce w tym uszeregowaniu. Ustalana jest liczba potomków dla osobnika o najwyższej randze (max) i liczba potomków dla osobnika o najniższej randze (min), pozostałym osobnikom przydzielana jest liczba kopii proporcjonalnie do ich rangi. Metoda ta bywa krytykowana, ponieważ osłabia ona związek pomiędzy funkcją przystosowania (rangą) a funkcją celu. Mimo to, w niektórych zastosowaniach daje dobre wyniki.

1.4 Operatory genetyczne

Podstawowe operatory genetyczne to krzyżowanie (mieszanie materiału genetycznego różnych osobników) i mutacja (błąd reprodukcji osobnika).

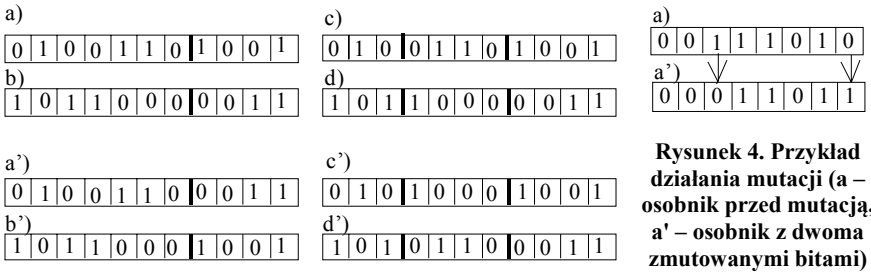
- *Krzyżowanie* (*crossover*)

W czasie krzyżowania dwa osobniki wymieniają między sobą części genotypu. Operator ten działa na reprodukowanym osobniku z zadanym prawdopodobieństwem,

drugi osobnik do krzyżowania wybierany jest losowo. Najprostsza postać to krzyżowanie *jednopunktowe* (osobniki *a* i *b* na rysunku 3., ich potomstwo to osobniki *a*' i *b*'). Może też być stosowane krzyżowanie *wielopunktowe*: osobniki *c* i *d*, powstałe po dwupunktowym krzyżowaniu potomstwo to *c*' i *d*'. Krzyżując reprodukowanego właśnie osobnika z innym można otrzymać dwóch potomków, składających się z części genotypów rodzicielskich. Do następnego pokolenia wybiera się losowo jednego z potomków, lepszego z nich, najbardziej podobnego do rodzica, itp., zgodnie z przyjętym założeniem. Uogólnieniem krzyżowania wielopunktowego jest krzyżowanie *jednorodne*. Dla każdego bitu pierwszego potomka podejmowana jest losowo (z zadanym prawdopodobieństwem, np. p=0,5) decyzja, od którego z rodziców ma dziedziczyć dany bit, drugi potomek otrzymuje bit od pozostałego rodzica.

- *Mutacja*

Powstały po krzyżowaniu potomek podlega mutacji. Mutacja zachodzi z zadanym prawdopodobieństwem, polega ona na losowej zmianie wartości bitu z *zera* na *jedynkę* lub odwrotnie (rysunek 4).



Rysunek 3. Krzyżowanie jednopunktowe (osobniki a i b, ich potomstwo a' i b') oraz dwupunktowe (c i d, potomstwo c', d')

Rysunek 4. Przykład działania mutacji (a – osobnik przed mutacją, a' – osobnik z dwoma zmutowanymi bitami)

1.5 Przykład (działanie prostego algorytmu genetycznego)

Zadanie: Znaleźć maksimum funkcji $F=x_1-x_2$, przy ograniczeniach: x_1, x_2 są liczbami całkowitymi z przedziału [0,31].

Decydujemy się na kodowanie binarne – kod dwójkowy, na każdą zmienną potrzeba 5 bitów (wynika to z dopuszczalnych wartości zmiennych x_1 i x_2), chromosom ma 10 bitów. Funkcja F przyjmuje wartości ujemne, co jest niedopuszczalne w **GA**, zatem musimy ją przekształcić w funkcję przystosowania. Jedną z możliwości jest funkcja Q zdefiniowana jako $Q = 32+F$, tę funkcję uwzględnimy przy ocenie osobników.

Zakładamy losową populację początkową, dla uproszczenia przyjmujemy, że populacja liczy 4 osobniki, mutacja zachodzi z prawdopodobieństwem 0,02 dla każdego bitu, rekombinacja reprodukowanego osobnika – jednopunktowa z prawdopodobieństwem 0,5, z losowo wybranym partnerem. Wybór osobnika do reprodukcji – np. metodą próbkowania deterministycznego. Warunek zatrzymania – zadana liczba pokoleń (np. 100).

Tabela 1. Populacja początkowa i generacja pierwszego pokolenia

Nr osob-nika	Chromosomy (losowe wartości)	Fenotyp (x_1, x_2)	Q_i	F_i	$E_i = Q_i/Q_{śr}$	Liczba potom-ków	Potomki (po działaniu operatorów genetycznych)
1	10001 01111	17, 15	34	02	1,133	1+0=1	10101 00101
2	**10101 00111**	**21, 07**	**46**	**14**	**1,533**	**1+1=2**	10111 00011, 10100 00001
3	00001 10101	01, 21	12	–20	0,400	0+0=0	brak
4	10101 11001	21, 25	28	–04	0,933	0+1=1	11110 11001
$Q_{śr}$	30,00						

Etapy 1 ÷ 4 (nie zachodzi warunek zatrzymania – nie wykonano zadanej liczby pokoleń) pokazuje tabela 1.

Analizując części całkowite wartości oczekiwanych E_1, E_2, E_3 i E_4 widzimy, że osobnik nr 1 i osobnik nr 2 będą mieć po jednym potomku. Dalej brakuje nam dwóch osobników do zachowania stałego rozmiaru populacji (założyliśmy 4 osobniki), dobieramy je zgodnie z malejącymi wartościami części ułamkowych E_1, E_2, E_3 i E_4: 0,933 (osobnik 4), 0,533 (osobnik 2), 0,400 (osobnik 3), 0,133 (osobnik 1) – czyli wybrane zostaną osobniki numer 4 i numer 2. Załóżmy dalej, że zgodnie z losowaniem z założonymi prawdopodobieństwami krzyżowania i mutacji wyszło nam, że:

Osobnik nr 1 (kopiowany raz) jest krzyżowany z osobnikiem nr 2, punkt krzyżowania – po szóstym bicie, mutowane bity – trzeci i dziewiąty.

Osobnik nr 2 (kopiowany dwukrotnie); raz bez krzyżowania, mutowane są geny: czwarty i ósmy; drugi raz – krzyżowanie z osobnikiem nr 4, po siódmym bicie, mutowane geny – piąty.

Osobnik nr 4 (raz kopiowany), brak krzyżowania, mutacja genów: drugi, czwarty, piąty.

Następuje teraz powrót do etapu 2: liczenie przystosowania osobników nowej populacji, sprawdzenie warunków zatrzymania i ew. generowanie kolejnego pokolenia. Pokazuje to tabela 2.

Tabela 2. Generacja kolejnego pokolenia

Nr osob-nika	Genotyp	Fenotyp	Q_i	F_i	$E_i = Q_i/Q_{śr}$	Liczba potom-ków	Potomki
1	10101 00101	21, 05	48	16	1,021	1+0=1	11101 00010
2	**10111 00011**	**23, 03**	**52**	**20**	**1,106**	**1+0=1**	11110 00011
3	10100 00001	20, 01	51	19	1,085	1+0=1	10101 10001
4	11110 11001	30, 25	37	05	0,787	0+1=1	11111 00001
$Q_{śr}$	47.00						

Poprawiła się średnia wartość przystosowawcza populacji. Generując kolejne pokolenia możemy po pewnym czasie uzyskać rozwiązanie optymalne [11111 00000], czyli [31,0], dla takiego osobnika F=31–0=31, a Q = 32+31=63.

Algorytm genetyczny szuka dobrego rozwiązania przesuwając populację w obiecujące rejony przestrzeni przystosowania (przestrzeni potencjalnych rozwiązań). Należy jednak pamiętać, że mechanizm ten bywa zawodny, np. jeśli mamy do czynienia z tzw. zwodniczymi funkcjami (ang. *deception function*). W takich sytuacjach **GA** może przesunąć populację w złym kierunku i ewolucja jest zbieżna do lokalnego optimum [14]. Algorytm genetyczny może być stosowany, jeśli dopuszczamy rozwiązanie bez gwarancji, że jest ono optymalne, wystarczy nam, że będzie satysfakcjonujące.

1.6 Specjalizowane operatory rekonfiguracyjne

W problemach sekwencyjnych (np. problem komiwojażera) geny w chromosomie są wartościami całkowitymi i stanowią numery miast, zaś kolejność genów na chromosomie mówi o tym, w jakiej kolejności miasta mają być odwiedzane. Zatem rozwiązaniem jest sekwencja genów w chromosomie. Silnym ograniczeniem nakładanym na chromosom jest wymóg, by każda wartość genu (z zakresu od 1 do liczby miast) wystąpiła na chromosomie dokładnie jeden raz. Dla tak zdefiniowanego chromosomu można stosować operator inwersji (jest to wylosowanie dwóch punktów w chromosomie i odwrócenie odcinka między tymi punktami) lub inne, specjalizowane operatory, które łączą w sobie cechy inwersji i krzyżowania. Poniżej omówione są przykładowe, podstawowe operatory rekonfiguracyjne.

Inwersja: Jest to stosunkowo prosty operator powoduje, że część genotypu (pomiędzy dwoma losowo wybranymi punktami) zostaje uporządkowana w odwrotnej kolejności.

```
I1 : 1 2 3 | 4 5 | 6 7 8
I2 : 4 3 2 | 7 6 | 5 8 1
-------------------------
I1': 1 2 3 | 7 6 | 4 5 8
I2': 7 3 2 | 4 5 | 6 8 1
```

Rysunek 5. Dwa osobniki przed PMX oraz dwa potomne osobniki uzyskane po PMX

Partially Matched Crossover (PMX): Na chromosomie wybierane są losowo dwa punkty. Część chromosomu pomiędzy tymi punktami stanowi tzw. sekcję dopasowania (ang. *matching section*) i ta część jest wymieniana pomiędzy reprodukowanymi osobnikami. Działanie operatora pokazane jest na rysunku 5. Osobniki I1' i I2' są potomkami osobników I1 i I2. W pierwszym kroku, po wylosowaniu sekcji dopasowania, I1' otrzymuje część chromosomu stanowiącą sekcję dopasowania osobnika I2 (tzn. wartości genów ***7*** i ***6*** są umieszczane u I1' na pozycjach odpowiadających sekcji dopasowania). Teraz geny o wartościach ***4*** i ***5*** (one były w sekcji dopasowania I1) zajmują u potomka I1' te miejsca, które u I1 były zajmowane odpowiednio przez geny o obecnych już wartościach ***7*** i ***6***. W podobny sposób jest tworzony drugi potomek (I2').

```
I1 : 1 2 3 | 4 5 | 6 7 8
I2 : 4 3 2 | 7 6 | 5 8 1
-------------------------
I1 : 1 2 3 | 4 5 | o o 8
I2 : o 3 2 | 7 6 | o 8 1
I1 : 3 4 5 | o o | 8 1 2
I2 : 2 7 6 | o o | 8 1 3
-------------------------
I1': 3 4 5 | 7 6 | 8 1 2
I2': 2 7 6 | 4 5 | 8 1 3

o – 'dziura'
```

przesuwanie dziur

Rysunek 6. Działanie operatora OX

Order Crossover (OX): operator zaczyna swoje działanie od losowego wyboru sekcji dopasowania, podobnie jak poprzednia metoda. Różni się jednak w

sposobie dopasowywania chromosomów. Geny, które powinny być przesunięte do sekcji dopasowania u pierwszego potomka, zostawiają „dziury". Powstałe „dziury" są przesuwane do sekcji dopasowania, a brakujące geny są dodawane na koniec chromosomu (rysunek 6). Sekcje dopasowania zostają wymienione pomiędzy dwoma reprodukowanymi osobnikami. Metoda OX, w odróżnieniu od PMX ma tendencję do zachowywania względnej pozycji genów.

Cycle Crossover (CX): Przy działaniu tego operatora, każdy gen u potomka jest wzięty od jednego z rodziców. Potomek jest tworzony według następującego algorytmu (patrz rysunek 7).

1. Bierzemy pierwszy gen od pierwszego rodzica I1 (w naszym przykładzie jest to wartość ***1***)
2. Potomek ma już pierwszy gen. Teraz należy zapewnić, by wartość z pierwszej pozycji I2 była obecna u potomka, dlatego też od I1 należy wziąć gen o wartości równej pierwszemu genowi u drugiego rodzica, w naszym przykładzie gen o wartości ***4***, kopiujemy go na tę pozycję, na której znajduje się u I1 (w przykładzie – na czwarte miejsce).
3. Osobnik I2 na czwartym miejscu (zajmowanym u I1 przez wartość ***4***) ma wartość ***7***, kopiujemy tę wartość (na jej miejsce u I1) do tworzonego potomka I1'.
4. Powtarzamy powyższe czynności (punkty 2 i 3) aż napotkamy gen, który już istnieje u tworzonego potomka (na miejscu zajmowanym przez ***8*** u I1 jest ***1***, która już wcześniej została skopiowana do potomka).
5. Brakujące pozycje u potomka wypełniamy kopiując geny od drugiego rodzica (I2).

```
I1 : 1 2 3 4 5 6 7 8
I2 : 4 3 2 7 6 5 8 1

I1': 1 - - - - - - -
I1': 1 - - 4 - - - -
I1': 1 - - 4 - - 7 -
I1': 1 - - 4 - - 7 8
I1': 1 3 2 4 6 5 7 8

I2': 4 - - - - - - -
I2': 4 - - - - - - 1
I2': 4 - - - - - 8 1
I2': 4 - - 7 - - 8 1
I2': 4 2 3 7 5 6 8 1
```

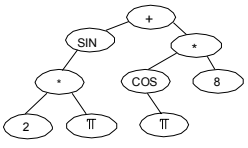
Rysunek 7. Tworzenie potomków stosując metodę CX

Analogicznie tworzony jest potomek I2'.

Można stosować inne (dodatkowe) techniki przyspieszające znalezienie rozwiązania problemów sekwencyjnych, na przykład różne heurystyki (wykorzystując pewną wiedzę o problemie).

2. Programowanie genetyczne

W programowaniu genetycznym (**GP** – ang. ***G****enetic Programming*) osobnik podlegający ewolucji nie jest binarnym łańcuchem, lecz złożoną strukturą drzewiastą [8],[12]. Zbiór możliwych struktur jest zbiorem wszystkich możliwych kombinacji funkcji, które mogą być rekurencyjnie wyprowadzone ze zbioru funkcji $F = \{f_1, f_2, ..., f_{N_F}\}$ o mocy N_F i zbioru symboli terminalnych $T = \{t_1, t_2, ..., t_{N_T}\}$ o mocy N_T. Przykłady takich zbiorów, to: F=(AND, OR, NOT, SIN, COS, +, –, ∗), T=(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, π), przykładowa funkcja w postaci drzewa pokazana jest na rysunku 8.



Rysunek 8. Przykład struktury drzewa w programowaniu genetycznym: *F*=SIN(2∗π)+8∗COS(π)

J. Koza [8] zaproponował programowanie genetyczne jako metodę automatycznego generowania programów. Węzły w drzewie są funkcjami (lub operatorami) jedno lub wieloargumentowymi, a liście są symbolami terminalnymi. Dla takich struktur należy inaczej niż w **GA** zdefiniować krzyżowanie i mutację. Krzyżowanie polega na wymianie poddrzew między dwoma osobnikami (strukturami), węzeł przecięcia jest losowany w każdym drzewie oddzielnie. Krzyżowanie odgrywa kluczową rolę w **GP**. Mutacja bywa różnie implementowana: losowa zmiana funkcji w węźle, losowa zmiana wartości liścia, zamiana wybranego poddrzewa innym, losowo wygenerowanym, zamiana dwóch poddrzew wychodzących z jednego węzła (permutacja). Dodatkowo, na drzewach można wykonywać:

- Edycję (upraszczanie) – nie zmienia się znaczenia wyrażenia reprezentowanego przez drzewo lecz upraszcza się formę (np. x OR x zastępuje się x),
- Enkapsulację (ADF – ang. *Automatically Defined Function*) – wyodrębnianie potencjalnie użytecznego poddrzewa i nadanie mu nazwy, tak by można go było stosować jak symbol terminalny. Enkapsulacja zapewnia niepodzielność wybranego poddrzewa w wyniku krzyżowania.

Kluczową rolę w **GP** pełni dobór odpowiednich zbiorów funkcji i symboli terminalnych. Wybór zbyt dużej ich liczby powoduje znaczny wzrost przestrzeni poszukiwań, jeśli jest ich za mało, lub są źle dobrane dla danego problemu, to satysfakcjonujące rozwiązanie nie może być znalezione.

3. Strategie ewolucyjne

Początek strategii ewolucyjnych (**ES** – ang. ***E**volutionary **S**trategy*) to lata sześćdziesiąte [14]. Zadaniem **ES**, podobnie jak **GA** i **GP**, jest rozwiązywanie problemów optymalizacyjnych metodą wzorowaną na ewolucji biologicznej. W **ES** nie stosuje się kodowania potencjalnych rozwiązań. Każdy chromosom jest łańcuchem liczb rzeczywistych: bezpośrednio wartości argumentów funkcji celu oraz prawdopodobieństw mutacji i rekombinacji. We wczesnych pracach ewoluowano populację składającą się z pojedynczego osobnika, a jako czynnik różnicujący potomka od rodzica wykorzystywano mutację. Małe zmiany mutacyjne były bardziej prawdopodobne niż duże. Potomek mógł zastąpić rodzica tylko wtedy, gdy był od niego lepszy. Takie podejście, w którym potomek konkuruje tylko ze swoim rodzicem nazywane jest *strategią dwuelementową* (ang. *two-membered evolution strategy*) i oznaczane jest przez symbol **(1+1)-ES**. Ewolucja populacji μ elementowej (gdzie μ>1) nazywana jest *strategią wieloelementową* (ang. *multi-membered evolution strategy*). W strategii wieloelementowej stosowane jest też krzyżowanie: dwa osobniki są losowo wybierane do reprodukcji, każdy z jednakowym prawdopodobieństwem. Podobnie jak w strategii **(1+1)-ES**, w jednym pokoleniu produkowany jest tylko jeden potomek, który zastępuje najgorszego osobnika w populacji. Jeśli wygenerowany potomek jest gorszy od wszystkich w populacji, jest on usuwany i populacja nie zmienia się w tym pokoleniu. Taka strategia (produkcja pojedynczego osobnika w jednym pokoleniu) jest nazywana **(μ+1)-ES**. W późniejszych aplikacjach **ES** zostały

rozwinięte do postaci znanych pod nazwą **(μ+λ)-ES** i **(μ,λ)-ES**. W strategii **(μ+λ)-ES** populacja liczy μ osobników, które w jednym pokoleniu produkują λ potomków. Do populacji tworzącej następne pokolenie wybierane są najlepsze osobniki zarówno spośród rodziców jak i potomków. Różnica pomiędzy strategiami **(μ+λ)-ES** i **(μ,λ)-ES** polega na tym, że w tej ostatniej, do następnego pokolenia wybierane są osobniki tylko spośród potomków (jest to populacja jednopokoleniowa).

ES były rozwijane jako metoda optymalizacji numerycznej, podczas gdy przed **GA** stawiano wymagania zdolności przeszukiwania. W obu podejściach wykorzystuje się darwinowską metodę selekcji oraz populację osobników będących potencjalnymi rozwiązaniami. Ważniejsze różnice pomiędzy **GA** a **ES** to:

- Reprezentacja osobnika: w klasycznym **GA** koduje się rozwiązania w postaci binarnego łańcucha, natomiast w **ES** ewoluują rozwiązania w postaci łańcucha liczb rzeczywistych, powiększone o prawdopodobieństwa mutacji i rekombinacji.
- Proces selekcji i reprodukcji: w **GA** populacja N-elementowa produkuje N potomków, które stanowią następne pokolenie, w **ES** – populacja μ-elementowa produkuje λ potomków, przy czym do następnego pokolenia wybieranych jest (w sposób deterministyczny) μ najlepszych osobników spośród rodziców i potomków razem, lub tylko spośród potomków. Selekcja, poza tym, że w **ES** jest deterministyczna, to zachodzi po procesie reprodukcji, natomiast w **GA** zachodzi przed reprodukcją (osobniki są wybierane do reprodukcji) i nie jest deterministyczna.
- W **GA**, prawdopodobieństwa mutacji i rekombinacji są z reguły stałe dla wszystkich osobników i w czasie całej ewolucji, natomiast w **ES** są one różne dla różnych osobników i ulegają zmianie w czasie ewolucji (same podlegają ewolucji stanowiąc część chromosomu).
- W strategiach ewolucyjnych potomki nie spełniające wymaganych warunków są eliminowane z populacji, nie ma mechanizmów typu nakładanie kar.

4. Programowanie ewolucyjne

Lawrence Fogel [5] zaproponował ewolucję populacji automatów skończonych w celu predykcji zmian środowiska i nazwał to podejście programowaniem ewolucyjnym (**EP** – ***E**volutionary **P**rogramming*). Opis środowiska to sekwencja symboli ze skończonego alfabetu. Zadanie polega na przewidywaniu kolejnego symbolu znając pewną ich sekwencję. Miarą przystosowania ewoluujących automatów skończonych jest dokładność przewidywania. Każdy osobnik w populacji produkuje jednego potomka. Potomek jest zmutowanym rodzicem (nie stosuje się krzyżowania). Mutacja jest to losowy proces, polegający na:

- zmianie symbolu wejściowego,
- dodaniu nowego stanu,
- usunięciu jednego z istniejących stanów, lub
- zmianie ścieżki pomiędzy stanami (tranzycji).

Decyzja o tym, która mutacja zajdzie podejmowana jest w trakcie tworzenia potomka, na podstawie zadanego rozkładu prawdopodobieństwa. Po reprodukcji każdego osobnika, populacja chwilowo składa się z podwojonej liczby osobników: rodziców i potomków. Jako następne pokolenie wybierane są najlepsze osobniki z podwojonej populacji, tak, że zachowany jest stały rozmiar populacji w kolejnych pokoleniach. Zwykle stosuje się turniejową metodę selekcji (turniej *n*-elementowy polega na wybraniu *n* elementów z populacji, zwycięzca – element z największym przystosowaniem – zostaje wybrany do następnego pokolenia). W pracach L. Fogel'a można znaleźć opisy wielu eksperymentów, w których automatom skończonym stawiano coraz trudniejsze zadania predykcji. Eksperymentowano z zastosowaniem programowania ewolucyjnego do gier. Eksperymenty z dwuosobowymi grami o sumie zerowej pokazały, że **EP** jest w stanie odnaleźć globalnie najlepszą strategię dla prostych gier i niewielkiej liczby graczy (czterech). Prace nad stosowaniem programowania ewolucyjnego są dość liczne, ich krótki przegląd można znaleźć w [5].

W **EP** nie stosuje się kodowania rozwiązań (podobnie jak w **ES**), reprezentacja wynika wprost z zadania – może to być sieć neuronowa w takiej postaci, jak jest implementowana. Mutacja często bywa zmniejszana w miarę zbliżania się do optimum globalnego. Zwykle jest to implementowane w ten sposób, że wariancja mutacji podlega zmianom (zgodnie z zadanym operatorem), czyli sama mutacja podlega ewolucji.

Literatura

[1] Bäck T, Hamel U, Schwefel HP. *Evolutionary Computataion: comments on the historyand current state*, IEEE Trans Evol Comput, 1997, pp. 3-17.

[2] Cantú-Paz E. (1997), *A Survey of Parallel Genetic Algorithms*, IlliGAL Report No. 97003.

[3] Davis L. (1987), *Genetic Algorithms and Simulated Annealing*, Morgan Kaufmann Publishers, Inc. San Mateo, California.

[4] Fogel DB. *Evolutionary Computation. Toward a New Philosophy of Machine Intelligence*, New York: IEEE Press: The Institute of Electrical and Electronics Engineers, Inc., 1995.

[5] Fogel LJ. *Artificial Intelligence Through Simulated Evolution*, Chichester: Wiley, 1996.

[6] Goldberg DE. *Algorytmy genetyczne i ich zastosowania*, WNT: Warszawa, 1995.

[7] Holland JH. *Adaptation in Natural and Artificial Systems*. Michigan, 1975.

[8] Koza J. *Genetic Programing*. London: MIT Press, 1996.

[9] Koza JR. *Future Work and Practical Applications of Genetic Programming*, 1996, Dla: *Handbook of Evolutionary Computation*, http://www-cs-faculty.stanford.edu/~koza.

[10] Kwaśnicka H. Evolutionary Approach to Artificial Neural Network Design – Good Way or Blind Alley. *MENDEL'97 III International Mendel Conference on Genetic, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets*. Brno, Czech Republic, 1997:342-47.

[11] Kwaśnicka H. Neural Network Design with Genetic Algorithm. In: *Computer Science, Proceedings of the Conference*. Czech Republic: Technical U. Brno, 1995:386-393.

[12] Kwaśnicka H. *Obliczenia ewolucyjne w sztucznej inteligencji*, Wrocław: Oficyna Wydawnicza PWr, 1999.

[13] Medsker LR. *Hybrid Intelligent Systems*, Boston: Kluwer Akademic Publishers, 1995.

[14] Michalewicz Z. *Algorytmy genetyczne + struktury danych = programy ewolucyjne*. WNT: Warszawa, 1996.
[15] Mitchell TM. *Machine Learning*, The McGraw-Hill Companies, Inc., 1997.
[16] Montana JD, Davis L. *Training Feedforward Neural Networks Using Algorithms*. Cambridge: BBN Systems and Technologies Corp., 1989.
[17] Rutkowska D, Piliński M, Rutkowski L. *Sieci neuronowe, algorytmy genetyczne i systemy rozmyte*, Warszawa: Wydawnictwo Naukowe PWN, 1996.
[18] Schalkoff RJ. *Artificial Intelligence: An Engineering Approach*. McGraw-Hill Publishing Comp., 1990.

Techniki Ewolucyjne w projektowaniu układu ścieżek na płytkach drukowanych

Rafał Pasek
Wydziałowy Zakład Informatyki
Politechnika Wrocławska

Streszczenie

Algorytmy genetyczne oraz pokrewne techniki bazujące na ewolucji cieszą się coraz większym zainteresowaniem tak teoretyków jak i praktyków. Podejmowane są próby ich wykorzystania w zagadnieniach inżynierii projektowej. W pracy tej rozważany jest problem prowadzenia połączeń na płytkach drukowanych (PCB) w kontekście możliwości zastosowania metod ewolucyjnych. Podstawowy nacisk położony jest na aspekt dostosowania zagadnienia projektowego do optymalizacyjnej natury przeszukiwania ewolucyjnego. Zaproponowane podejście obejmuje transformację problemu w zadanie wymagające satysfakcji szeregu ograniczeń. Przedstawiono i omówiono podstawowy algorytm ewolucyjny, sposób jego dostrojenia do specyfiki problemu, a także zastosowane metody obsługi ograniczeń obejmujące karanie, opracowane z wykorzystaniem wiedzy dziedzinowej reprezentację i operatory oraz mechanizm krokowej adaptacji wag.

Wstęp

Popularne w ostatnim czasie algorytmy genetyczne oraz inne techniki naśladujące mechanizm ewolucji znalazły wiele zastosowań w rozwiązywaniu problemów optymalizacyjnych. Ich niewątpliwą zaletą jest połączenie prostoty i ogólności. Umożliwiają optymalizację szerokiej klasy problemów: liniowych i nieliniowych, określonych zarówno na ciągłych, dyskretnych oraz mieszanych przestrzeniach poszukiwań, nieograniczonych i ograniczonych. W przypadku wielu złożonych problemów szczególnie interesującą cechą jest elastyczność pozwalającą na wykorzystanie opartych na wiedzy dziedzinowej heurystyk, co pozwala na dostrojenie algorytmów ewolucyjnych do charakterystyki konkretnego problemu. Dotyczy to w szczególności zagadnień związanych z inżynierią projektową, wykazujących dużą złożoność, co czyni je trudnymi do rozwiązania klasycznymi metodami algorytmicznymi.

Wyróżniającą tę klasę problemów cechą jest fakt występowanie zbioru ograniczeń, których liczba i powiązania stanowią o rzeczywistej trudności zagadnienia. Obecność ograniczeń wpływa znacząco na efektywność każdego algorytmu optymalizacyjnego, także technik bazujących na symulowanej ewolucji. W niniejszej pracy omówiono algorytm ewolucyjny rozwiązujący przykładowy problem związany z inżynierią projektową, wykorzystujący niektóre z szeroko stosowanych technik dotyczących

spełniania narzuconych na zadanie ograniczeń. Wykorzystano opracowaną w oparciu o wiedzę dziedzinową reprezentację oraz operatory genetyczne, a także mechanizm krokowej adaptacji wag.

1. Specyfikacja problemu

Algorytmy ewolucyjne znalazły zastosowanie w wielu aspektach projektowania i montażu płytek drukowanych (PCB), wliczając projektowanie strukturalne układu, selekcję komponentów oraz optymalizację procesu montażu elementów. W pracy tej skupiono się na możliwościach wykorzystania technik ewolucyjnych w celu rozwiązania problemu prowadzenia połączeń fizycznych na jednowarstwowych płytkach PCB, w kontekście zagadnienia optymalizacyjnego z silnymi ograniczeniami.
W problemie tym dane są:

- ograniczony, spójny obszar płaszczyzny zwany dalej płytką drukowaną,
- uporządkowany zbiór punktów lutowniczych P,
- funkcja przyporządkowująca każdemu punktowi ze zbioru P jego pozycję na płytce zadaną parą współrzędnych kartezjańskich $f: P \rightarrow R \times R$,
- uporządkowany zbiór połączeń strukturalnych (tj. planowanych) S między punktami ze zbioru P, zadanych jako pary punktów, które powinny zastać połączone fizycznie tj. $S = \{(p_1, p_2) \mid p_1, p_2 \in P, p_1 \text{ i } p_2 \text{ są strukturalnie połączone}\}$.

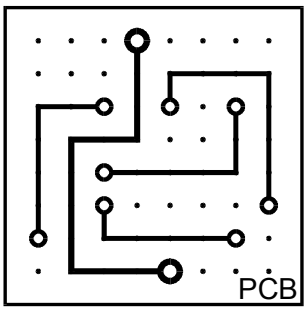
Problem polega na zaprojektowaniu sieci fizycznych połączeń w taki sposób, aby dowolne dwa punkty lutownicze zostały fizycznie połączone wtedy i tylko wtedy, jeżeli występowało między nimi połączenie strukturalne. Przyjmując, że dla danego rozwiązania zbiór F zawiera wszystkie i tylko te pary punktów lutowniczych, między którymi istnieje fizyczne połączenie, warunek sukcesu można zapisać następująco:

$$\forall p_1, p_2 \in P. \quad (p_1, p_2) \in F \Leftrightarrow (p_1, p_2) \in S \qquad \textbf{(1)}$$

Poniższe rozważania oparto o zredukowaną wersję problemu, w której *(warunek 0.)*:

- płytka składa się z jednej warstwy, w kształcie prostokąta o ustalonych rozmiarach,
- punkty leżą na przecięciach siatki, tj. ich współrzędne zadane są liczbami całkowitymi,
- fizyczne połączenia mogą być prowadzone tylko wzdłuż nałożonej na powierzchnię płytki siatki, zbudowanej z kwadratów o boku równym 1.

Przykład schematu połączeń fizycznych PCB odpowiadającego temu warunkowi przedstawia *rys.1*.



Rysunek 1: Przykładowy schemat PCB

2. Podejście ewolucyjne

Zastosowanie algorytmu ewolucyjnego do dowolnego zadania wymaga określenia sposobu obliczania funkcji celu, reprezentacji osobnika oraz opracowania zestawu operatorów genetycznych. Dla sformułowanego czysto projektowego problemu trudno spełnić te wymagania w sposób bezpośredni, konieczne jest jego dostosowanie do optymalizacyjnej natury przeszukiwania ewolucyjnego. Należy tego dokonać poprzez określenie zbioru optymalizowanych współczynników oraz ograniczeń, których spełnienie stanowi kryterium określenia projektu jako satysfakcjonujący.

Sposób kodowania osobnika, czyli jego reprezentacja oraz zestaw operatorów łącznie określają zestaw potencjalnych rozwiązań problemu możliwych do uzyskania w drodze symulowanej ewolucji, czyli przestrzeń rozwiązań. Funkcja oceniająca przystosowanie powstałych osobników, pozwala z kolei na ukierunkowanie ewolucji w kierunku rozwiązań, które uważamy za lepsze, a w konsekwencji kieruje poszukiwania w obszary, gdzie spodziewamy się znaleźć satysfakcjonujące rozwiązanie. Zagadnienia te należy rozpatrywać łącznie, szczególnie w przypadku konieczności stosowania skomplikowanej reprezentacji osobnika, a tym samym złożonego sposobu mapowania genotypu w fenotyp, a także konieczności obsługi nałożonych na problem ograniczeń.

2.1 Reprezentacja

Ze sposobem kodowania osobnika w bezpośredni sposób wiąże się określenie maksymalnej dopuszczalnej przestrzeni poszukiwań. Odpowiednio dobrana reprezentacja może znacząco zmniejszyć przestrzeń poszukiwań, wpływając tym samym na efektywność algorytmu optymalizacyjnego.

Rozważany problem można rozłożyć na dwie składowe, wynikające z warunku równoważności danego wzorem [1] stanowiące niezależne warunki ograniczeń, a mianowicie (*ograniczenia 1a. i 1b.*):

(1a) każde dwa punkty, które są połączone strukturalnie, muszą zostać połączone fizycznie,

(1b) żadne dwa punkty, które nie są połączone strukturalnie, nie mogą zostać połączone fizycznie.

Najprostszą z możliwości jest wybranie reprezentacji dopuszczającej dowolne, zgodne z *warunkiem 0.* rozwiązania. Osobnik, który spełniłby oba powyższe ograniczenia, mógłby zostać uznany za prawidłowy projekt układu połączeń na płytce drukowanej. W ten sposób problem został więc sprowadzony do zagadnienia wyewoluowania prawidłowych, tj. spełniających *ograniczenia 1a i 1b.* osobników.

Zagadnieniu obsługi ograniczeń w algorytmach ewolucyjnych poświęcono wiele prac. W pracy [2] sklasyfikowano i przedstawiono 11 mechanizmów pozwalających na powstanie w toku symulowanej ewolucji prawidłowych, tzn. spełniających ograniczenia, osobników. Zostały one zgrupowane w trzy podstawowe grupy obejmujące Zapobieganie, Korygowanie i Presję. Metody związane z Zapobieganiem i Korygowaniem pozwalają na stałe zapewnienie zgodności osobników z ograniczeniami, dotyczą więc tzw. ograniczeń twardych. Z Presją z kolei związana jest klasa ograniczeń miękkich, tj. takich, których naruszanie w czasie działania algorytmu nie jest zabronione.

Karanie osobników uważa się za najbardziej ogólną metodę obsługi ograniczeń. Mechanizm ten bazuje zazwyczaj na preferowaniu rozwiązań w mniejszym stopniu naruszających ograniczenia, co powinno spychać populację w kierunku rozwiązań w jak największym stopniu satysfakcjonujących poszczególne ograniczenia. W opracowaniu skutecznego mechanizmu karania pomocne mogą okazać się, sformułowane na podstawie badań hipotezy, które znaleźć można w pracy [4]:

(a) „kary, które są miarą odległości od poprawności są lepsze niż te, które są prostą funkcją liczby naruszonych ograniczeń”,

(b) „w przypadku problemów z kilkoma ograniczeniami oraz kilkoma pełnymi rozwiązaniami, kary które są jedynie funkcją liczby naruszonych ograniczeń nie prowadzą zazwyczaj do znalezienia rozwiązania”.

Dodatkowo, w [5] sformułowana jest następująca hipoteza:

(c) „algorytm genetyczny ze zmiennym współczynnikiem kary jest skuteczniejszy od algorytmu ze stałym współczynnikiem kary”.

Istnieją także inne metody obsługi ograniczeń, są one jednak w mniejszym lub większym stopniu zależne od problemu. Jedną z najbardziej interesujących technik jest wykorzystanie tzw. dekoderów, należących do klasy związanej z Zapobieganiem. Stanowią one specjalizowane schematy mapowania genotypu na fenotyp, określające sposób budowania prawidłowego osobnika na podstawie informacji zawartych w chromosomie. Za ich pomocą możliwe jest, wspomniane wcześniej, zredukowanie przestrzeni poszukiwań, a także zapewnienie spełnienia jednego bądź kilku narzuconych na problem ograniczeń. Przy wyborze odpowiedniej reprezentacji – dekodera, należy wziąć pod uwagę szereg czynników (za [1], *wymogi (1)-(5)*):

dla każdego prawidłowego rozwiązania, musi istnieć kodujący je osobnik,

(1) każdy zdekodowany osobnik, musi odpowiadać prawidłowemu rozwiązaniu,

(2) każde prawidłowe rozwiązanie musi być reprezentowane przez tę samą liczbę możliwych, różnych osobników.

Ponadto, sugeruje się aby:

(3) transformacja osobnika w potencjalne rozwiązanie była szybka obliczeniowo,
(4) dekoder miał cechę lokalności, tj. małe zmiany w chromosomie powodowały małe zmiany w rozwiązaniu.

Opracowanie dekodera spełniającego powyższe wymagania dla złożonych, rzeczywistych problemów jest zazwyczaj poważnym problemem. Wielu praktyków wykorzystuje więc fakt, że analogiczny do zastosowania dekodera efekt osiągnąć można poprzez wykorzystanie specjalnie opracowanej reprezentacji oraz zestawu operatorów. Założeniem jest tutaj zachowanie poprawności – zgodności z zadanym ograniczeniem – wszystkich osobników w populacji. Operatory powinny więc działać w taki sposób, aby niemożliwe było otrzymanie osobnika nieprawidłowego poprzez działanie na osobnikach prawidłowych. Należy dodatkowo pamiętać o takim wyborze populacji początkowej, aby składała się ona wyłącznie z poprawnych osobników.
Stosowanie specyficznej dla problemu reprezentacji oraz specjalizowanych operatorów często pozwala na uzyskanie algorytmów ewolucyjnych efektywniejszych niż w przypadku technik opartych na karaniu. Zakres istniejących zastosowań jest bardzo szeroki, począwszy od optymalizacji numerycznej i klasycznych problemów, jak problem komiwojażera, przez uczenie maszynowe, aż po przetwarzanie sygnałowe, robotykę, oraz zagadnienia związane z inżynierią projektową.

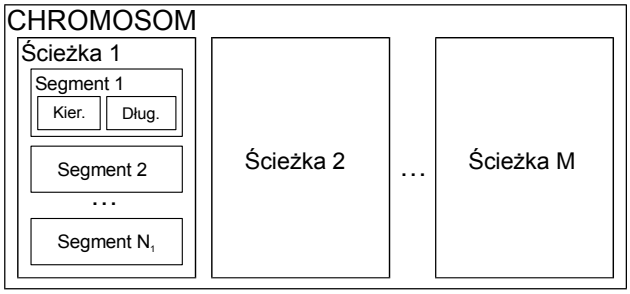
W tym momencie warto wrócić do określonego wcześniej rozkładu zadania na dwa podproblemy, z którymi skojarzone zostały *ograniczenia 1a. i 1b*. Istnieje możliwość eliminacji jednego z ograniczeń poprzez zastosowanie odpowiedniej reprezentacji oraz zestawu operatorów, gwarantujących jego utrzymanie. W tym przypadku wybrano *warunek 1a.* tj. konieczność fizycznego połączenia wszystkich strukturalnie połączonych punktów. Warunkiem podtrzymywanym jest więc istnienie w populacji jedynie osobników kodujących rozwiązania zawierające wszystkie niezbędne połączenia fizyczne, zaś zadanie sprowadza się do znalezienia takiego rozwiązania, które nie zawiera nadmiarowych połączeń fizycznych, między punktami niepołączonymi strukturalnie, co stanowi *warunek 1b*.

2.2 Sposób kodowania

Każdy osobnik powinien przedstawiać rozwiązanie zawierające wszystkie wymagane połączenia fizyczne – „ścieżki". Dla zadanego problemu rozwiązanie zawiera $n = card(S)$ ścieżek, kodowanych w chromosomie. Pojedyncza ścieżka stanowi połączenie pomiędzy dwoma punktami lutowniczymi. Składa się ona z szeregu „segmentów" stanowiących pionowe lub poziome odcinki o całkowitoliczbowej długości. Kształt ścieżki reprezentowany jest przez skrócony kod łańcuchowy, składający się z ciągu par *(kierunek, długość)*, gdzie kierunek określony jest jako zmienna wyliczeniowa o wartościach *(góra, dół, prawo, lewo)*, natomiast długość jest liczbą naturalną. Aby

możliwe było zakodowanie i zdekodowanie rozwiązania, dla każdego połączenia jeden z łączonych punktów lutowniczych ustalony jest jako początek ścieżki.

Przykładowo dla najdłuższej (pogrubionej) ścieżki z *rys.1*. przyjmując jako początkowy górny punkt otrzymujemy kod w postaci: *((dół,3), (lewo,2), (dół,4), (prawo,3))*.



Rysunek 2: Struktura chromosomu

Reprezentacja ma strukturę hierarchiczną i składa się z komponentów kodujących poszczególne ścieżki. Jest to szczególnie wygodne ze względu na intuicyjność opracowanych operatorów genetycznych. Dla ułatwienia, kody ścieżek przedstawione są w ustalonej kolejności, co pozwala na ich identyfikacją na podstawie zajmowanej pozycji. Warto jednak zaznaczyć, iż ze względu na nieustaloną liczbę segmentów tworzących ścieżkę, mamy do czynienia z chromosomem o zmiennej długości.

Rozpatrując cechy powyższej reprezentacji pod kątem wymagań stawianych przed klasycznym dekoderem, możemy stwierdzić, iż bezpośrednio spełnione są punkty:

(1) reprezentacja pozwala na przedstawienie każdego poprawnego rozwiązania,
(2) każdemu poprawnemu rozwiązaniu odpowiada dokładnie jeden kod,
(3) z kodu możemy w sposób bezpośredni odtworzyć kształt ścieżek, a tym samym całe rozwiązanie.

Wybrana reprezentacja nie wyklucza powstania niepoprawnych ze względu na *warunek 1a.* osobników, nie jest więc to klasyczny dekoder. Sposób kodowania i dekodowania zapewnia jedynie rozpoczynanie się ścieżki w miejscu jednego z łączonych punktów lutowniczych, nie ma natomiast bezpośredniego mechanizmu wymuszającego umiejscowienie końca ścieżki w drugim punkcie. Jak już jednak wspomniano możliwe jest wprowadzenie równoważnego mechanizmu, polegającego na utrzymywaniu poprawności wszystkich osobników w populacji.

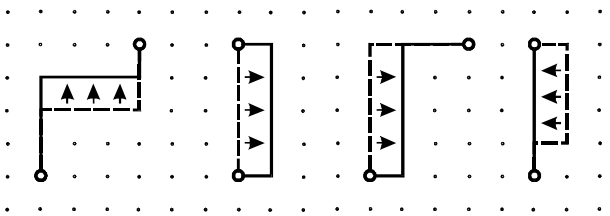
2.3 Operatory

Aby możliwe było spełnienie wymogu (2), zgodnie z którym każdy zdekodowany osobnik musi odpowiadać prawidłowemu (w tym przypadku spełniającemu *warunek 1a.*) rozwiązaniu, należy zapewnić takie funkcjonowanie operatorów, aby niemożliwe

było powstanie nieprawidłowych osobników w wyniku działania na osobnikach prawidłowych. Zakładając, że populacja początkowa składa się wyłącznie z osobników prawidłowych, tj. składających się ze ścieżek rozpoczynających i kończących się w łączonych punktach, pozwoli to na utrzymanie spełnienia *warunku 1a*. przez wszystkie wyewoluowane rozwiązania. Wynika z tego, że wszelkie modyfikacje wprowadzane przez operatory, nie mogą prowadzić do relokacji punktu końcowego poszczególnych ścieżek – stałość punktu początkowego zapewnia sposób kodowania. Opracowany zestaw operatorów obejmuje operator mutacji oraz rekombinacji.

2.4 Mutacja

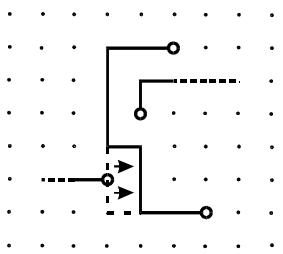
Operator mutacji działa na poziomie ścieżki tj. modyfikuje wybraną ścieżkę składową osobnika. Mechanizm działania wzorowany jest na heurystyce postępowania projektanta obwodów drukowanych. Wprowadzane w konfigurację ścieżki modyfikacje polegają na przesunięciu jednego z segmentów ścieżki w kierunku do niego prostopadłym. Wiąże się to zazwyczaj z modyfikacją długości segmentu poprzedniego i następnego, a czasem z ich utworzeniem lub usunięciem. Typowe sytuacje przedstawione są na *rys.3*., gdzie linią przerywaną pokazano wejściową konfigurację ścieżki, strzałkami kierunek przesunięcia segmentu, a linią ciągłą wynik mutacji.



Rysunek 3: Przykłady działania operatora mutacji „a”

Zaimplementowana operacja mutacji polega na losowym wybraniu dowolnego segmentu ścieżki, z jednakowym prawdopodobieństwem, i przesunięciu go o jedną pozycję w wylosowanym kierunku – góra lub dół w przypadku segmentu poziomego, lewo lub prawo w przypadku segmentu pionowego. Pociąga to za sobą konieczność zmiany długości, utworzenia, bądź usunięcia poprzedniego i następnego segmentu, bez zmiany ich położenia. Schemat ten daje się łatwo uogólnić poprzez wprowadzenie parametru określającego siłę mutacji, od którego uzależniona jest wartość przesunięcia. Operator spełnia warunek wykluczenia relokacji punktu końcowego ścieżki, a jednocześnie zapewnia zgodność z wymogiem (5) o lokalności zmian. Zapewnia przeszukanie całej przestrzeni rozwiązań, przez sekwencję mutacji można bowiem uzyskać każdą dozwoloną zmianę i tylko dozwoloną zmianę.

Niestety, niektóre, pożądane z punktu widzenia zadania projektowego, modyfikacje kształtu ścieżki wymagają sekwencji wielu mutacji, przeprowadzających osobnika przez niepożądane z punktu widzenia *warunku 1b.* obszary przestrzeni poszukiwań charakteryzujące się dużą liczbą przecinających, bądź nakładających się ścieżek. Jest to związane z koniecznością przemieszczenia tylko fragmentu jednego z segmentów ścieżki, przy pozostawieniu pozostałej jego części na miejscu – sytuację taką przedstawia *rys.4*. Operator mutacji „b" dzieli segment w losowo wybranym punkcie na dwie części, z których jedna (także losowo wybrana) podlega przesunięciu. Mimo, iż operator pierwotny „a" jest szczególną wersją operatora „b", zachowano obie wersje, wybierane z równym prawdopodobieństwem. Miało to na celu zmniejszenie udziału w populacji, osobników składających się z nieregularnych, wielosegmentowych ścieżek, generowanych przez ogólniejszą odmianę.



Rysunek 4: Operator mutacji „b"

W przypadku obu operatorów wielkość przesunięcia segmentu losowana jest z zakresu *1..s* (*s* zadane przez użytkownika), przy czym prawdopodobieństwo wylosowania kolejnych, coraz większych wartości jest liniowo malejące. Bezpośrednio preferowane są więc mniejsze przesunięcia.

2.5 Krzyżowanie

Opracowanie operatora rekombinacji utrudnia fakt zmiennej długości chromosomu. Jednak i w tym przypadku powstało wiele prac zajmujących się bezpośrednio tym zagadnieniem, z których można wymienić choćby [6]. Stworzenie operatora krzyżowania, działającego na poziomie pojedynczych ścieżek, byłoby niejasne z punktu widzenia sposobu kodowania, dopuszczającego zmienną długość chromosomu, jak i samego zagadnienia projektowania obwodów drukowanych. Krzyżowanie operuje więc na poziomie nadrzędnym, i polega na wymianie całych, niezmienionych ścieżek pomiędzy rodzicami. Przebiega analogicznie do krzyżowania jednopunktowego znanego z AG z ta różnica, że jednostką podstawową jest tutaj nie bit, lecz stanowiąca komponent tworzący ścieżka. Jak już wspomniano, ścieżki są uporządkowane, a ich liczba jest taka sama dla wszystkich osobników, tej samej pozycji w wektorach dwóch osobników odpowiada więc to samo połączenie. Ponieważ operator nie modyfikuje samych ścieżek, zachowany jest wymóg utrzymania *warunku 1a*.

3. Algorytm ewolucyjny

Zastosowany algorytm bazuje na klasycznym schemacie algorytmu genetycznego. Podstawowym krokiem jest utworzenie nowej populacji osobników, czyli potencjalnych rozwiązań, na podstawie wyselekcjonowanych, najlepiej dostosowanych przedstawicieli poprzedniej generacji. Pseudokod algorytmu przedstawia się następująco:

```
BEGIN
   Inicjuj populacje poprawnymi (spełniającymi warunek 1a.)
    osobnikami;
   Ocena;
   WHILE nie znaleziono satysfakcjonującego rozwiązania DO
      Selekcja i Krzyżowanie;
      Mutacja;
      Ocena;
   END WHILE
END
```

Podstawową cechą algorytmu jest utrzymywanie liczebności populacji na stałym, zadanym przez użytkownika poziomie. W każdym kroku algorytmu, obejmującym generację nowego pokolenia, należy więc utworzyć stałą liczbę osobników tworzących nową populację. Wymaga to zdefiniowania funkcji oceny przystosowania danego osobnika, która jest podstawowym kryterium określającym prawdopodobieństwo jego wyboru.

Proces tworzenia nowego pokolenia można podzielić na kilka etapów: selekcja osobników do reprodukcji, krzyżowanie osobników, mutacje osobników. W tym rozwiązaniu fazy selekcji osobników i krzyżowania zostały połączone. Na tym etapie wybierane są, metodą losowania z zastosowaniem ruletki, pary osobników. Tworzą one pulę przejściową o liczebności takiej jak cała populacja, poprzez bezpośrednie skopiowanie pary rodziców, bądź skopiowanie ich potomków powstałych na drodze krzyżowania. Prawdopodobieństwo wystąpienia krzyżowania jest określane przez użytkownika. Kolejny etap – Mutacja – przetwarza pulę przejściową w nową populację. Dla każdego osobnika z puli, z wybranym przez użytkownika prawdopodobieństwem wywoływany jest operator mutacji. Warto podkreślić dwa fakty: prawdopodobieństwo krzyżowania dotyczy każdej wylosowanej pary, a prawdopodobieństwo mutacji dotyczy każdego osobnika jako całości. Ponadto zdecydowano się na zastosowanie mechanizmu pozwalającego na zachowanie najlepszego osobnika w danym pokoleniu, poprzez przeniesienie go bez modyfikacji do nowej populacji. Dzięki temu najlepsze rozwiązania nie są gubione, jednocześnie mechanizm ten przez swój elitaryzm powoduje skupienie osobników w pobliżu najlepszego.

3.1 Funkcja oceny

Przystosowanie danego osobnika określa się analizując stopień spełnienia wymagań określonych przez cel zadania. W tym przypadku oceniamy rozwiązania pod kątem spełniania ograniczenia związanego z *warunkiem 1b*. Jako podstawową metodę osiągnięcia celu, tj. znalezienia rozwiązania satysfakcjonującego oba warunki, zastosowano Presję, a konkretnie – metodę związaną z karaniem osobników za naruszanie nałożonych ograniczeń.

Analiza sformułowania problemu wykazuje, że podstawowym źródłem naruszania ograniczeń jest przecinanie się ścieżek, co prowadzi w konsekwencji do powstawania niepożądanych połączeń fizycznych między punktami nie połączonymi strukturalnie. Karanie osobników w zależności od liczby przecięć powinno spowodować presję selekcyjną kierującą ewolucję w kierunku rozwiązań o coraz mniejszej ich liczbie, a w konsekwencji, w kierunku satysfakcjonującego rozwiązania. Liczba przecięć określa stopień naruszenia ograniczenia (*warunek 1b.*), czyli odległość osobnika od poprawności, zwaną także kosztem ukończenia, bądź kosztem naprawy. Zgodnie z wspomnianą hipotezą (a) wykorzystanie tego parametru daje lepsze rezultaty niż poprzestanie na samym fakcie naruszenia ograniczenia.

Jednocześnie w bezpośredni sposób minimalizacji podlegać mają także dwa dodatkowe parametry: sumaryczna długość wszystkich ścieżek oraz liczba segmentów. Ma to na celu preferowanie rozwiązań o krótkich i regularnych ścieżkach. Uzyskana w ten sposób większa ilość wolnego miejsca powinna wpłynąć na wzrost prawdopodobieństwa korzystnej mutacji, redukującej liczbę przecięć poprzez przesunięcie przecinającego się segmentu w kierunku obszaru nie zajętego przez żadną ścieżkę.

Aby uwzględnić wszystkie wymagania, zamiast bezpośredniego obliczania przystosowania osobnika, wyznaczany jest stopień jego nieprzystosowania, określony przez sumaryczną karę. Wartość ta określona jest przez sumę ważoną kar za naruszenie ograniczeń i optymalizowanych parametrów oznaczanych jako k_i, a dokładnie:

- k_1 - liczba przecięć,
- k_2 - sumaryczna długość ścieżek,
- k_3 - sumaryczna liczba segmentów tworzących ścieżki.

Zestaw ten rozszerzono o dodatkowe parametry związane z wymogiem nie wykraczania ścieżek poza dozwolony obszar, określony przez wymiary płytki:

- k_4 - liczba ścieżek poza płytką,
- k_5 - sumaryczna długość części ścieżek poza płytką.

Parametrom tym przypisano wagi w_i odpowiadające ich znaczeniu, tak aby najwyżej karane było umieszczanie ścieżek poza dozwolonym obszarem, kolejno mniej – przecinanie się ścieżek, długość ścieżek i najmniej liczba segmentów tworzących. Stopień nieprzystosowania określa *wzór 2*.:

$$f = \sum_i w_i k_i \quad \textbf{(2)}$$

Wyznaczona ocena ma wartość tym większą im gorzej oceniamy danego osobnika, zastosowanie ruletki jako mechanizmu selekcji wymaga więc odpowiedniego przeliczenia, tak aby prawdopodobieństwo wylosowania osobnika o większej wartości kary było mniejsze. Jest to realizowane poprzez wyznaczenie wartości przystosowania jako odwrotnie proporcjonalnej do wyznaczonej kary.

$$F_j = \frac{f_{\min}}{f_j} \quad \textbf{(3)}$$

gdzie: f_j, F_j – odpowiednio nieprzystosowanie (kara) i przystosowanie *j*-tego osobnika w populacji

f_{min} – najmniejsza wartość nieprzystosowania w populacji, pełni rolę współczynnika normalizacyjnego.

Cechą tego przekształcenia jest bezpośrednie przełożenie stosunku kar dwóch osobników na stosunek prawdopodobieństwa wybrania ich do następnego pokolenia: osobnik o *k*-krotnie większej karze ma *k*-krotnie mniejszą szansę wylosowania. Cechy tej nie posiada często stosowana metoda wyznaczania przystosowania poprzez odjęcie wartości kary od stałej, zgodnie z wzorem $F = c - f$. Ponadto, w przyjętej metodzie nie zachodzi potrzeba obsługi przypadków, dla których wyznaczona wartość przystosowania byłaby ujemna.

3.2 Krokowa adaptacja wag

Zastosowanie karania jako mechanizmu obsługi ograniczeń w bezpośredni sposób sprowadza problem do zadania optymalizacji. Związane są z tym jednak pewne wady, do których należą m.in.:

(a) utrata informacji związana z reprezentacją wiedzy o naruszonych ograniczeniach w postaci pojedynczej wartości,
(b) nieefektywność mechanizmu w przypadku problemów rzadkich.

Rozpatrywany problem należy zakwalifikować do klasy problemów rzadkich, gdyż pomimo pozbycia się jednego z ograniczeń, w zredukowanej przestrzeni przeszukiwań udział rozwiązań poprawnych tj. składających się z rozłącznych ścieżek jest bardzo mały. Przeprowadzone testy wykazały skłonność algorytmu do przedwczesnej zbieżności i utykania w jednym z lokalnych optimów. Jest to związane ze strukturą przestrzeni poszukiwań, w której głębokie, lokalne optima związane z ustaleniem się konfiguracji zawierającej niewielką liczbę przecinających się ścieżek otoczone są przez silnie karane rozwiązania zawierające wiele przecięć. Należałoby więc wprowadzić mechanizm pozwalający na wyprowadzenie populacji z takiego optimum w celu zwiększenia zdolności przeszukiwawczych algorytmu.

Opracowany mechanizm bazuje na zaproponowanym w przez Eibena i Hemerta w [3] mechanizmie krokowej adaptacji wag (ang. SAW – Stepwise Adaptation of Weights). Wspomniana praca skupia się na problemie określenia wag związanych z

twardością bądź priorytetem ograniczeń. Osiągnięcie satysfakcjonującej efektywności algorytmu wymaga takiego ustalenia wartości wag, aby odpowiadały rzeczywistej twardości ograniczeń, dzięki czemu spełnienie trudniejszego z ograniczeń będzie silniej premiowane. Wartości te są zazwyczaj ustalane na podstawie wiedzy dziedzinowej, ewentualnie dostrajane doświadczalnie. Przeprowadzone przez autorów badania wykazały jednak, że z faktu, iż przeszukiwanie ewolucyjne jest procesem dynamicznym składającym się z różnych faz, wynika, że optymalne wartości parametrów algorytmu podlegają zmianom w czasie jego działania. Przedstawiony pomysł polega więc na dostosowywaniu wartości wag w czasie rozwiązywania problemu przez algorytm ewolucyjny.

Rozpatrzmy następującą postać minimalizowanej funkcji celu (dla *n* ograniczeń):

$$f = \sum_{i=1}^{n} \alpha_i \chi_i \qquad \textbf{(4)}$$

gdzie χ_i oznacza wartość kary generowaną przez naruszenie *i*-tego ograniczenia, a α_i jest podlegającą krokowej adaptacji wagą związaną z *i*-tym ograniczeniem. Ideę algorytmu krokowej adaptacji wag przedstawia poniższy pseudokod:

```
Ustaw początkowe wartości wag α_i (a więc postać funkcji f);
WHILE nie znaleziono satysfakcjonującego rozwiązania DO
    FOR T_p-kolejnych obliczeń funkcji celu DO
        Obliczaj przystosowanie według f;
    END FOR
    Przedefiniuj f i przelicz przystosowanie osobników;
END WHILE
```

Zmiana funkcji przystosowania *f* polega na dodaniu wartości $\Delta\alpha$ do wag α_i tych ograniczeń, które są naruszone przez najlepszego osobnika po T_p krokach obejmujących wyznaczenie przystosowania. Mechanizm powoduje więc zwiększenie priorytetu tych ograniczeń, które są na danym etapie optymalizacji najtrudniejsze do spełnienia (najczęściej łamane), poprzez zwiększenie presji selekcyjnej nałożonej na wymóg poprawności osobnika względem tych ograniczeń. Efektywność tego podejścia potwierdza słuszność hipotezy (c) mówiącej, że zastosowanie zmiennych wartości wag może dać potencjalnie lepsze rezultaty niż w przypadku stałych ich wartości.

Aby rozwiązać problem związany z utykaniem w optimum lokalnym, *warunek 1b.* o rozłączności wszystkich ścieżek zastąpiono zestawem *n*-warunków (gdzie *n* – liczba połączeń) dotyczących osobno każdej ze ścieżek. Każdy warunek związany jest z wymogiem rozłączności pojedynczej ścieżki (nie przecinania się jej z żadną ścieżką). Spełnienie tych ograniczeń przez wszystkie ścieżki, a więc zapewnienie rozłączności każdej ze ścieżek, jest równoważne spełnieniu *warunku 1b*. Z ograniczeniami tymi związane są modyfikowane w czasie działania algorytmu wagi – mamy więc po jednym współczynniku α_i, $i=1..n$ dla każdej ze ścieżek, których wartości będą podlegać adaptacji.

Wprowadzenie współczynników α_i wymaga określenia nowego sposobu wyznaczania wartości kary za przecięcie ścieżek. W celu wyznaczenia nowej wartości kary dla danego osobnika określane są wszystkie punkty przecięć ścieżek, a dla każdego punktu wartość kary wyznacza się jako iloczyn współczynników α_i przecinających się w tym punkcie ścieżek. Kary za poszczególne przecięcia są sumowane, a ostateczna wartość zastępuje we *wzorze 2.* wartość współczynnika k_1 (oryginalnie równą liczbie przecięć). Określenie sumarycznej kary jako iloczynu współczynników α_i przecinających się ścieżek powoduje szczególne mocne karanie przecięć ścieżek o wysokich współczynnikach kar. Nakłada więc presję na wyeliminowanie przecięć między nimi na rzecz słabiej karanych przecięć ze ścieżkami o niskiej wartości współczynnika α.

Zgodnie z ideą Eibena i Hemerta wartości wag należy zmieniać co określoną liczbę kroków, w tym przypadku zdecydowano się na modyfikację po każdym pokoleniu. Wagi α_i podlegają modyfikacji pod warunkiem, że nowe pokolenie nie zawiera lepszego, tj. niżej karanego rozwiązania, co wskazuje na możliwe utknięcie w lokalnym optimum. Algorytm modyfikacji kar przedstawia się następująco:

```
Ustaw wartości wag αi dla wszystkich ścieżek na 1;
WHILE nie znaleziono satysfakcjonującego rozwiązania DO
    wyznacz nowe pokolenie;
    IF nie znaleziono lepszego rozwiązania THEN
        na podstawie najlepszego osobnika –
        - ustal listę przecinających się ścieżek;
        zwiększ wartości wag αi dla przecinających –
        - się ścieżek o 1;
    END IF
END WHILE
```

Mamy więc początkową wartość współczynników $\alpha=1$ co oznacza, że każdy z warunków rozłączności danej ścieżki jest tak samo ważny, oraz jednostkową modyfikację $\Delta\alpha=1$. Modyfikowanie wag można porównać do gromadzenia przez algorytm wiedzy o strukturze przestrzeni rozwiązań, wykorzystywanej do sprawniejszego rozwiązywania problemu. Należy jednak zaznaczyć, że nie następuje tutaj wyznaczenie zestawu najlepszych dla danego problemu wag, które czynią go łatwym do rozwiązania. Przeprowadzone przez Eibena i Hemerta badania polegające na porównaniu efektywności algorytmu wykorzystującego mechanizm SAW oraz algorytmu opartego o stałe wartości wag ustalone na podstawie ostatecznych wartości wag algorytmu SAW, wykazały dużo wyższą efektywność podejścia adaptacyjnego.

Efektywność mechanizmu krokowej adaptacji wag bazuje raczej na wymuszaniu ciągłej zmiany punktu skupienia poszukiwań, co prowadzi do niejawnej dekompozycji problemu. Algorytm ewolucyjny wzbogacony o ten mechanizm wykazuje właściwości eksploracyjne – ewolucja przeprowadza populację przez szereg optimów lokalnych, tj. konfiguracji o niewielkiej liczbie przecinających się ścieżek. Przez cały okres zastoju

związanego z utknięciem w lokalnym optimum następuje zwiększanie wartości współczynników kar α_i przecinających się ścieżek. Prowadzi to do jego „eksploatacji" poprzez wzrost sumarycznej wartości kary dla osobników z taką konfiguracją przecinających się ścieżek i związane z tym obniżenie ich przystosowania. Po osiągnięciu odpowiedniego poziomu wartości współczynników korzystniejsze stają się konfiguracje zawierające być może większą liczbę przecięć, lecz nie zawierające przecięć między dotychczas najczęściej karanymi ścieżkami. Następuje ponowna zbieżność algorytmu do kolejnego optimum lokalnego. Skojarzenie funkcjonowania tego mechanizmu z eksploatacją złóż związane jest z faktem obniżania stopnia atrakcyjności obszaru – określonego jako przystosowanie osobników – wskutek pozostawania populacji w jego obrębie. Po wyczerpaniu złoża populacja rozbiega się w poszukiwaniu nowego, w którym ponownie się skupia.

Wadą zastosowanego rozwiązania jest niedokładność określenia „eksploatowanego" obszaru przestrzeni rozwiązań. Jest on zdefiniowany jedynie na podstawie zbioru przecinających się ścieżek i podlega jednorodnej eksploatacji przez zwiększanie ich współczynników kar, niezależnie od faktycznego położenia zbioru osobników jak i samego optimum lokalnego. Aby lepiej określić eksploatowany obszar i umożliwić jego opuszczenie poprzez serię nieznacznych modyfikacji osobników w kierunku prawidłowego rozwiązania, wprowadzono dodatkowy zestaw adaptowanych współczynników β_i związanych z wszystkimi punktami płytki (leżącymi na przecięciach linii siatki). Po ich uwzględnieniu wyznaczanie kary dla *j*-tego przecięcia polega na obliczeniu iloczynu współczynników α_i przecinających się w danym punkcie ścieżek (tak jak w poprzednim przypadku) oraz współczynnika β_j tego punktu. Wartości współczynników β_j ustalane i zmieniane są analogicznie do współczynników α_i, tj. początkowo ich wartość wynosi jeden i zwiększana jest o jeden dla punktów, w których znajdują się przecięcia (określonych na podstawie najlepszego osobnika po każdym pokoleniu). Podsumowując, wartość kary za *j*-te przecięcie dana jest *wzorem 5*.:

$$P_j = \left(\prod_{s_j} \alpha_{s_j} \right) \cdot \beta_j \qquad \textbf{(5)}$$

gdzie s_j przyjmuje wartości indeksów ścieżek przecinających się w *j*-tym punkcie.
Sumaryczna wartość kary za wszystkie przecięcia zastępująca we *wzorze 2*. współczynnik k_1 określający liczbę przecięć, dana jest *wzorem 6*.:

$$k_1 = \sum_j P_j \qquad \textbf{(6)}$$

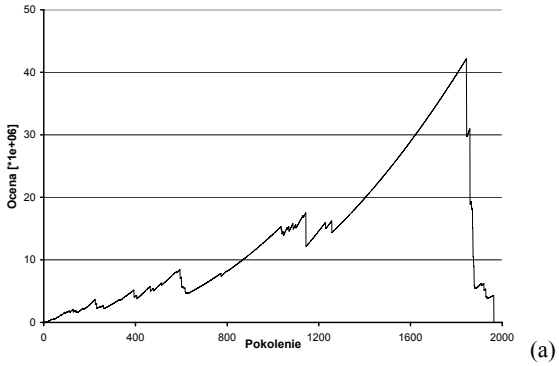
gdzie *j* określa kolejne punkty przecięcia dla konfiguracji danego osobnika.

Wyznaczoną w ten sposób ocenę (karę) określa się w dalszej części dokumentu jako adaptacyjną, natomiast wyznaczoną w sposób pierwotny (tj. dla k_1 równego liczbę przecięć) jako rzeczywistą. Ocena rzeczywista odpowiada ocenie adaptacyjnej przy ustaleniu wartości wag α_i oraz β_i na jeden.

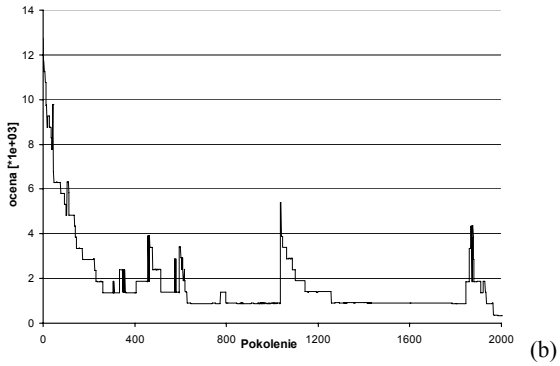
3.3 Mechanizm funkcjonowania krokowej adaptacji wag

Następujące po sobie etapy zbiegania do optimum lokalnego oraz jego eksploatacji najłatwiej prześledzić na wykresie przedstawiającym zmiany wartości nieprzystosowania najlepszego osobnika w kolejnych pokoleniach przedstawione na rysunku 5ab.

Analizując przebieg z rys.5a. można dostrzec kolejne etapy szybkiego, w przybliżeniu kwadratowego, wzrostu oceny najlepszego osobnika – a więc jego kary. Wzrosty te są wynikiem zwiększania wartości wag α_i i β_i kar za przecięcia wywołanego utknięciem w optimum lokalnym. Są to etapy „eksploatacji" optimum, trwającej aż do momentu przekroczenia krytycznej wartości wag pozwalającej na ustalenie się innej konfiguracji przecięć czemu towarzyszy gwałtowny spadek wartości kary adaptacyjnej.



(a)

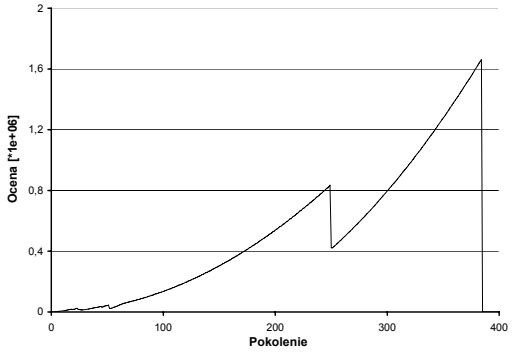


(b)

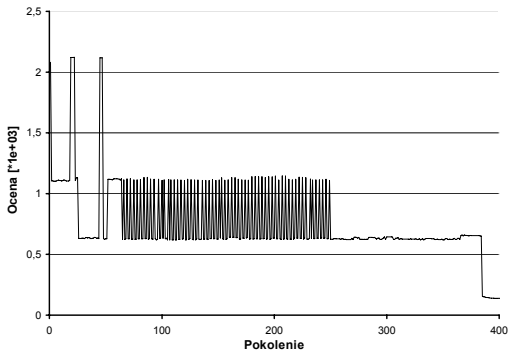
Rysunek 5: Wartość adaptacyjnej (a) oraz rzeczywistej (b) oceny najlepszego osobnika

Porównując przebiegi przedstawiające adaptacyjną i rzeczywistą ocenę można zauważyć, że wytrąceniu populacji z optimum lokalnego (spadek wartości kary adaptacyjnej) towarzyszy gwałtowny wzrost rzeczywistej kary najlepszego osobnika, zwiększa się więc liczba przecięć. Następnie liczba przecięć ulega redukcji i populacja zbiega w kierunku kolejnego optimum lokalnego – obserwujemy „schodkowy" spadek wartości rzeczywistej kary. Przebieg algorytmu składa się więc faktycznie z trzech powtarzających się naprzemiennie faz: zbiegania do optimum lokalnego (i), jego eksploatacji (ii) oraz ekspansji w poszukiwaniu nowego optimum (iii). Proces ten kończy się w momencie znalezienia optimum związanego z konfiguracja pozbawioną przecinających się ścieżek.

Jeszcze jeden typ zachowania pozostawił wyraźnie widoczny ślad na przebiegach z *rys.6ab*. Chodzi tutaj o oscylacje rzeczywistej oceny najlepszego osobnika, której towarzyszy wzrost kary adaptacyjnej. Odpowiada to sytuacji naprzemiennego ustalania się dwóch lub większej liczby różnych konfiguracji przecinających się ścieżek, które kolejno stają się korzystniejsze by następnie wskutek „eksploatacji" ustąpić miejsca następnym. Ostatecznie prowadzi to do ustalenia się innej konfiguracji związanej z nowym optimum lokalnym.



(a)



(b)

Rysunek 6: Wartość adaptacyjnej (a) oraz rzeczywistej (b) oceny najlepszego osobnika

4. Badania

W tej części pracy zawarte są wyniki badań i ich analiza pod kątem efektywności funkcjonowania wprowadzonych mechanizmów. Badania dla wszystkich czterech problemów przeprowadzono przy ustaleniu następujących wartości podstawowych parametrów algorytmu:

- Prawdopodobieństwo mutacji: 0,8
- Prawdopodobieństwo krzyżowania: 0,1
- Liczebność populacji: 100

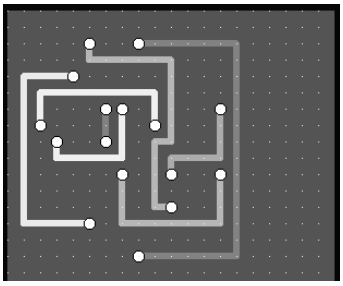
Oznacza to, że przeciętnie 80 na 100 osobników z puli przejściowej poddawane było pojedynczej mutacji, natomiast 5 na 50 wylosowanych par zastępowane było swoimi potomkami przy kopiowaniu do puli przejściowej. Jako moment znalezienia rozwiązania określono numer pokolenia, do którego należał pierwszy osobnik pozbawiony przecięć. W celu wyznaczenia ostatecznej oceny działanie algorytmu

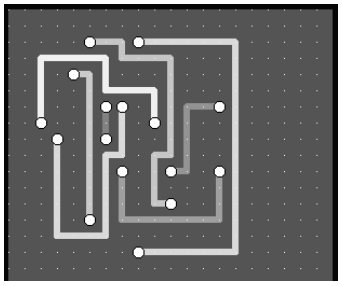
kontynuowano do momentu uzyskania osobnika o minimalnej długości i liczbie segmentów tworzących ścieżki, przy ustaleniu maksymalnej siły mutacji *s=1* w celu praktycznego wyeliminowania możliwości „przekroczenia" jednej ze ścieżek nad inną.

4.1 Problem 1

Liczba punktów lutowniczych	16
Liczba połączeń	8
Uwagi: • położone blisko siebie punkty lutownicze utrudniają prowadzenie ścieżek najkrótszą drogą • duża liczba różnych rozwiązań	

Pomiary przeprowadzono dla trzech wartości parametru *s* określającego maksymalną siłę mutacji. Dla każdej wartości siły przeprowadzono 30 pomiarów, notując moment rozwiązania problemu oraz ocenę znalezionego rozwiązania. Wartości uśredniono (śred.) i wyznaczono ich odchylenie standardowe (odch.).





Rysunek 7ab: Problem 1 - przykładowe rozwiązania, w tym najlepsze - 7a

L.p.	*s = 10*		*s = 5*		*s = 3*	
	l. pokoleń	ocena	l. pokoleń	ocena	l. pokoleń	ocena
1	77	106,6	140	112,5	412	112,4
2	59	108,3	271	106,5	324	106,7
3	195	104,2	99	106,6	184	106,5
4	408	116,8	280	106,7	248	106,6
5	220	114,3	95	112,4	904	110,9
6	107	104,2	186	106,6	174	112,4
7	308	114,8	133	106,6	127	106,7
8	37	100,6	141	106,7	209	106,6
9	49	104,2	53	100,5	270	106,6
10	105	116,5	161	100,6	312	106,7
11	140	106,7	310	110,8	265	106,7
12	102	106,6	165	106,6	201	102,6
13	62	128,5	275	106,6	160	106,5
14	368	108,3	36	100,5	842	106,6
15	248	100,6	40	104,2	576	114,7

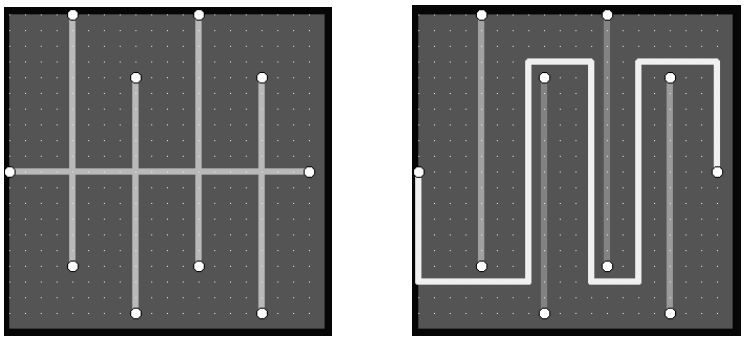
16	263	114,8	155	100,6	394	110,8
17	639	106,7	91	100,6	310	107
18	247	114,6	137	120,4	212	102,6
19	60	104,2	68	108,3	341	106,6
20	225	104,2	642	107	57	100,5
21	43	104,2	343	100,5	122	106,6
22	89	108,3	415	120,4	92	106,6
23	160	108,3	279	120,4	237	106,7
24	341	104,2	79	106,6	95	102,6
25	258	108,3	269	106,8	261	100,5
26	276	110,6	350	106,8	44	100,6
27	73	108,5	82	114,7	202	106,6
28	129	120,4	124	106,7	843	107
29	64	104,2	52	100,6	101	106,6
30	149	110,9	135	106,8	520	106,6
śred.	**183,3**	**109,1**	**186,9**	**107,4**	**301,3**	**106,6**
odch.	**136,5**	**6,2**	**134,6**	**5,8**	**228,0**	**3,4**

Tabela 1: Wyniki dla problemu 1

Można zauważyć, że dla większych *(5,10)* wartości maksymalnej siły mutacji rozwiązanie znajdowane jest przeciętnie prawie dwukrotnie szybciej niż dla małej wartości siły. Wartość *s=3* jest więc w tym przypadku zbyt mała aby umożliwić jednorazową mutację pozwalającą na pozbycie się niepożądanego przecięcia. Wymagana jest w tym przypadku sekwencja mutacji, przeprowadzająca przez silnie karane konfiguracje, a tym samym mniej prawdopodobna i wymagająca dłuższego działania mechanizmu eksploatacji. Dla tej wartości siły algorytm zbieżny jest z kolei do przeciętnie lepiej ocenianych rozwiązań i wykazuje się prawie dwukrotnie mniejszym odchyleniem standardowym ich oceny. Związane jest to z faktem bardziej prawdopodobnego, a przez to częstszego powstawania konfiguracji zawierającej prowadzone naokoło ścieżki w przypadku dużych wartości parametru *s*. Dla małych wartości siły częściej powstają rozwiązania zawierające ścieżki prowadzone pomiędzy punktami lutowniczymi, a więc krótsze. Mniejsza liczba oraz zbliżona ocena takich konfiguracji tłumaczy mniejszy rozrzut oceny rozwiązań. Analogiczny wpływ wartości parametru *s* widoczny jest także dla pozostałych problemów.

4.2 Problem 2

Liczba punktów lutowniczych	10
Liczba połączeń	4
Uwagi: • jedno najprostsze rozwiązanie wymagające prowadzenia ścieżki poziomej pomiędzy ścieżkami pionowymi • stosunkowo dużo miejsca pomiędzy ścieżkami	



Rysunek 8ab: Problem 2 – sytuacja wejściowa (8a) i rozwiązanie (8b)

Tabela 2. zawiera wyniki badań przeprowadzonych dla problemu 2 i czterech różnych wartości parametru *s*. Zestawiono wartości średnie oraz odchylenia standardowe liczby pokoleń potrzebnych do znalezienia rozwiązania. Poza jednym przypadkiem wszystkie znalezione rozwiązania odpowiadały wzorowi z *rys.8b*.

	s = 10	*s* = 5	*s* = 3	*s* = 1
śred.	1040,7	1856,4	2019,1	4398,7
odch.	711,0	1321,9	924,5	2111,2

Tabela 2: Wyniki dla problemu 2

Także i w tym przypadku widać tendencję do wzrostu liczby pokoleń koniecznych do znalezienia rozwiązania wraz ze zmniejszaniem siły mutacji. Jednak nawet dla najniższej możliwej wartości *s=1*, odpowiadającej dopuszczeniu przesunięć tylko o jedną pozycję, algorytm jest w stanie znaleźć prawidłowe rozwiązanie. Jest to możliwe dzięki wprowadzeniu kar pozycyjnych i związanych z nimi współczynników β_i zwielokrotniających karę za przecinanie się ścieżek w konkretnym miejscu. Wymusza to ciągłą zmianę punktu przecięcia dwóch ścieżek umożliwiając ich rozdzielenie poprzez sekwencję mutacji wyprowadzającą punkt przecięcia poza koniec jednej ze ścieżek.

4.3 Problem 3

Liczba punktów lutowniczych	38
Liczba połączeń	19
Uwagi: • duża liczba punktów i połączeń	

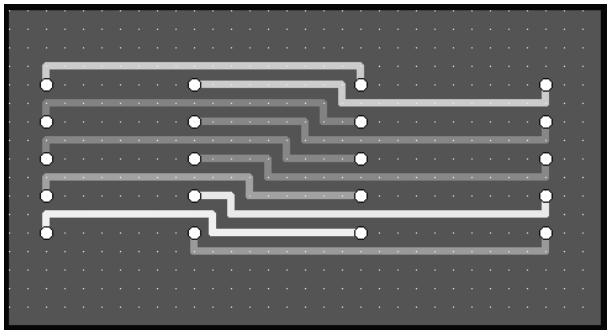
Lp.	*s = 5*		*s = 10*	
	l. pokoleń	ocena	l. pokoleń	ocena

1	3706	334,1	575	342,2
2	516	333,7	6073	344,2
3	4995	340,3	894	362,7
4	1447	325,6	915	333,8
5	1444	333,9	5095	358,9
6	2706	327,7	1445	348,6
7	8548	380,5	1305	345,5
8	1592	352,5	1307	356,9
9	3360	325,5	7425	378,3
10	4082	339,9	558	340,2
śred.	3239,6	339,4	2559,2	351,1
odch.	2334,7	16,6	2587,4	13,1

Tabela 3: Wyniki dla problemu 3

4.4 Problem 4

Liczba punktów lutowniczych	16
Liczba połączeń	6
Uwagi: • konieczność ciasnego, równoległego prowadzenia połączeń • jedno najlepsze rozwiązanie, oraz kilka gorszych zawierających ścieżki prowadzone naokoło	



Rysunek 9: Problem 4 - najlepsze rozwiązanie

s = 5		*s = 10*	
l.pokoleń	ocena	l.pokoleń	ocena

śred.	4506,4	228,7	3177,4	234,3
odch.	2162,5	4,4	2275,9	20,1

Tabela 4: Wyniki dla problemu 4

Konieczność ciasnego prowadzenia ścieżek sprawia, że rozwiązanie tego problemu bez mechanizmu kar pozycyjnych okazało się praktycznie niemożliwe. Jest to związane z faktem istnienia tylko jednego najlepszego rozwiązania, pozbawionego nadmiarowego miejsca dla wewnętrznych ścieżek. W wyniku większości mutacji związanych z przemieszczeniem któregoś z poziomych segmentów powstają konfiguracje zawierające nakładające się fragmenty ścieżek, silnie karane ze względu na sumowanie się kar z każdego punktu przecięcia. Mamy więc do czynienia z głębokim optimum globalnym otoczonym wysoką i rozległą barierą silnie karanych rozwiązań. Podobny charakter mają także inne lokalne optima przestrzeni poszukiwań, co sprawia, że jest to problem szczególnie trudny do rozwiązania tradycyjnymi metodami ewolucyjnymi. Uzyskane wyniki wykazują względnie dużą liczbę pokoleń potrzebnych na znalezienie rozwiązania pozbawionego przecięć. Ponadto należy zwrócić uwagę na fakt, iż większość ze znalezionych rozwiązań zawiera jedną lub więcej ścieżek prowadzonych naokoło, a rozwiązanie optymalne (pokazane na *rys.9.*) znajdowane jest zaledwie w kilku procentach przypadków.

Porównując problemy 3 i 4 można zauważyć, że rzeczywista trudność zadania nie jest prostą funkcją liczby punktów i połączeń, lecz zależy w dużym stopniu od gęstości ich położenia. Określa ona nadmiar miejsca dla prowadzenia ścieżek, który wpływa na charakter rozkładu funkcji celu. Konieczność gęstego prowadzenie ścieżek odpowiada przestrzeń poszukiwań o głębokich i ciasnych optimach lokalnych, z czym związany jest dłuższy czas „błądzenia" populacji oraz konieczność dłuższej eksploatacji znalezionych optimów lokalnych.

5. Podsumowanie

Przedstawiony w tej pracy algorytm ewolucyjny dostrojono do rozwiązywania jednego szczególnego problemu związanego z szeroko pojętą inżynierią projektową. Podstawowym zagadnieniem było takie sformułowanie problemu, z którym w sposób naturalny związany jest zestaw ograniczeń, aby sprowadzić go do zadania optymalizacyjnego możliwego do rozwiązania za pomocą technik ewolucyjnych. Zakres modyfikacji obejmuje wprowadzenie opartej na wiedzy dziedzinowej strukturalnej reprezentacji oraz zestawu operatorów genetycznych, ich współdziałanie w celu ograniczenia przestrzeni poszukiwań oraz redukcji liczby twardych ograniczeń, a także wykorzystanie mechanizmu krokowej adaptacji wag w celu zwiększenia możliwości eksploracyjnych. Źródłem tych modyfikacji były sformułowane w wielu pracach wskazówki i hipotezy odnośnie problemu spełniania ograniczeń, które zostały zaadaptowane do tego zadania. Wykorzystano zarówno metody polegające na zapewnianiu spełnienia ograniczeń w oparciu o specjalnie opracowaną reprezentację i zestaw operatorów, jak i metody związane z nałożeniem presji selekcyjnej spychającej

populację w kierunku obszaru poprawnych rozwiązań. Nie są to oczywiście wszystkie możliwości, jednak wybrano je ze względu na ich ogólność (karanie), jak i możliwość wykorzystania wiedzy dziedzinowej (reprezentacja, operatory, kary pozycyjne). Warto także zwrócić uwagę na sposób dostosowania mechanizmu krokowej adaptacji wag do tego problemu. Widać tutaj możliwość analogicznego zastosowania wszędzie tam, gdzie jedno z ograniczeń daje się rozbić na zbiór słabszych ograniczeń, z którymi możemy związać podlegające adaptacji wagi. Wyniki badań świadczą o potencjalnej opłacalności takiego postępowania. Pozwoli ono być może na efektywne rozwiązywanie zadań tradycyjnie określanych jako trudne bądź niemożliwe do rozwiązania za pomocą technik ewolucyjnych.

Literatura

[1] Z.Michalewicz, D.Dasgupta, R.G.Le Riche, M.Schoenauer, Evolutionary Algorithms for Constrained Engineering Problems (1996), International Symposium on Methodologies for Intelligent Systems

[2] T.Yu, P.Bentley, Methods to Evolve Legal Phenotypes (1998), Fifth International Conference on Parallel Problem Solving from Nature

[3] A.E.Eiben, J.I.van Hemert, SAW-ing Eas: adapting the fitness function for solving constrained problems (1999), New Ideas in Optimization

[4] J.T. Richardson, M.R.Palmer, G.Liepins, M.Hilliard, Some Guidelines for Genetic Algorithms with Penalty Functions (1989), Proc. of the 3rd Int. Conference on GA, pp. 191-197

[5] W.Siedlecki, J.Sklanski, Constrained Genetic Optimization via Dynamic Reward Penalty Balancing and Its Use in Pattern Recognition (1989), Proc. of the 3rd Int. Conference on GA, pp. 141-150.

[6] P.J.Bentley, J.P.Wakefield, Hierarchical Crossover in Genetic Algorithms (1996), Proceedings of the 1st On-line Workshop on Soft Computing

Przegląd metod automatycznego planowania – przykład wykorzystania algorytmu genetycznego w rozwiązaniu prostego problemu planowania

Maciej Norberciak
Wydziałowy Zakład Informatyki, Politechnika Wrocławska
E-mail: norber@box43.pl

Streszczenie

Praca poświęcona jest tematyce automatycznego planowania i harmonogramowania. Dokonano uporządkowania stosowanych terminów oraz przeglądu stosowanych metod. Na zakończenie omówiono przykład zastosowania algorytmu genetycznego do zadania planowania dyżurów personelu w rzeczywistym szpitalu.

1. Wstęp

Niniejszy rozdział stanowi wprowadzenie do problematyki automatycznego planowania i harmonogramowania. Zawiera definicje tych pojęć oraz krótkie opisy najczęściej badanych problemów.

1.1 Co to jest planowanie?

W języku angielskim istnieje kilka słów oznaczających „plan" i „planowanie". Rzeczownik *plan* odpowiada polskiemu rzeczownikowi „plan", oznaczającemu „program zadań i prac (...), które mają być wykonane w określonym czasie; porządek, rozkład zajęć lub czynności przewidzianych do wykonania" (wg Małego Słownika Języka Polskiego, PWN 1993). Analogicznie angielski czasownik *plan* odpowiada polskiemu „planować" – układać plany. Oprócz tego w języku angielskim występuje wyraz *schedule* (zarówno w znaczeniu czasownikowym jak i rzeczownikowym), który jest najbliższy polskiemu rzeczownikowi „harmonogram" – „(...) opis obrazujący kolejność i czas trwania poszczególnych czynności w ogólnym planie pracy" (ibid.) – oraz czasownikowi „harmonogramować" – układać harmonogramy. Oprócz tego język angielski dysponuje „specjalnym" rzeczownikiem *timetable*, który oznacza plan zajęć lub rozkład jazdy.

W najbardziej ogólnym przypadku harmonogramowanie (*scheduling*) jest problemem grupowania zasobów (lub po prostu *zdarzeń*) w pewnych punktach czasowych (nazywanych w tej pracy *terminami*) w danym okresie czasowym, aby osiągnąć pewien określony cel (cele) i/lub spełnić określone założenia (*ograniczenia*). Układanie planów, czy też krócej planowanie (*timetabling*), jest szczególnym przypadkiem harmonogramowania (Newall 1999).

1.2 Układanie planów szkolnych i jego warianty

Jednym z najpopularniejszych problemów planowania jest układanie planów dla szkół i uczelni. Układanie planu szkolnego jest problemem takiego ustalenia sekwencji spotkań studentów z nauczycielami w określonym przedziale czasowym, aby były spełnione różnego typu ograniczenia (Schaerf 1995).

Człowiekowi ułożenie planu, w zależności od skali jego złożoności, zajmuje zwykle od kilku godzin do kilku dni. Co więcej, tak skonstruowany plan może być niedoskonały pod pewnymi względami, np. uczeń ma między kolejnymi zajęciami wiele długich przerw. Z tych powodów zwrócono uwagę na możliwość automatyzacji procesu układania planów. Pierwsze prace z tej dziedziny powstały w latach sześćdziesiątych ubiegłego stulecia (za Schaerf 1995). Od tego czasu powstało i zostało wdrożonych wiele aplikacji, efektywnie rozwiązujących problem układania planów.

W literaturze pojawia się wiele wariantów problemu układania planu, różniących się zarówno typem szkoły (podstawowa, średnia, wyższa), jak i typem ograniczeń. A. Schaerf (1995) dzieli problemy układania planów na trzy klasy:

- Układanie planu lekcji (*school timetabling*) – ułożenie tygodniowego planu dla wszystkich zajęć w szkole tak, aby żaden nauczyciel nie prowadził dwóch zajęć w tym samym czasie i na odwrót,
- Układanie planu zajęć na uczelni (*course timetabling*) – ułożenie tygodniowego planu dla wszystkich kursów na uczelni tak, aby uniknąć nakładania się zajęć, na które uczęszcza ten sam podzbiór studentów,
- Układanie planu egzaminów (*examination timetabling*) – ułożenie planu egzaminów dla zbioru kursów na wyższej uczelni tak, aby uniknąć nakładania się egzaminów z przedmiotów, na które uczęszcza ten sam podzbiór studentów.

Blisko powiązany z układaniem planu zajęć jest tzw. *podproblem grupowania*. Na niektórych uczelniach wyższych pewne zajęcia są powtarzane częściej niż raz w tygodniu. W szczególności są to zajęcia wspólne dla dużej liczby studentów różnych wydziałów (kierunków, specjalności) podzielonych na grupy. Załóżmy, że specjalność S_1 obejmuje wykłady W_1 i W_2, a specjalność S_2 kursy W_1 i W_3. Dodatkowo załóżmy, że wykład kursu W_2 odbywa się w punkcie czasowym p, a wykład W_3 w punkcie q. W tym przypadku wykład W_1 nie może się odbywać ani o czasie p ani q. Jeżeli jednak studenci specjalności S_1 i S_2 zostaną podzieleni na dwie grupy, to jedna będzie mogła uczęszczać na wykład W_1 o czasie p a druga o czasie q. Problem podziału studentów na grupy przy ustalonym planie zajęć w celu minimalizacji liczby konfliktów nazywany jest właśnie podproblemem grupowania (*grouping subproblem*, *student sectioning*) (Schaerf 1995).

1.3 Planowanie a służba zdrowia

Planowanie personelu (*employee scheduling*) jest problemem bardzo podobnym do układania planów szkolnych. Popularne są tu modele związane ze służbą zdrowia – planowanie dyżurów lekarzy różnych specjalności, pracy pielęgniarek i personelu

technicznego. W tych przypadkach chodzi o ułożenie planu tak, żeby wszystkie terminy w pewnym okresie czasowym (najczęściej w ciągu miesiąca) były obsadzone pracownikami w określony sposób, przy jednoczesnym zapewnieniu równomiernego obciążenia pracowników pracą. Opisywany w tej pracy problem jest wariantem problemu planowania personelu w służbie zdrowia. Oprócz planowania personelu, w służbie zdrowia występują również problemy harmonogramowania zadań dla oddziałów intensywnej opieki medycznej, planu operacji chirurgicznych oraz działań specjalistycznych jednostek diagnostycznych (laboratoriów, ultrasonografów itp.). Przykłady zadań planowania i harmonogramowania w służbie zdrowia można znaleźć w (Spyropoulos 2000; Marinagi, Spyropoulos, Papatheodorou i Kokkotos 2000; Oddi i Cesta 2000; Valouxis i Housos 2000).

1.4 Wybrane problemy harmonogramowania

Artykuł ten dotyczy głównie problemów układania planów, lecz nie należy zapominać, że analogiczne metody rozwiązań stosuje się także w zadaniach harmonogramowania. Poniżej opisano kilka popularnych problemów harmonogramowania.

Harmonogramowanie zadań w środowiskach przetwarzania równoległego

Problem ten dotyczy przydziału zadań (procesów) do procesorów w środowisku przetwarzania równoległego. System multiprocesorowy jest reprezentowany przez nieskierowany, nieważony graf, zwany *grafem systemu* (*system graph*). Węzeł grafu systemu reprezentuje procesor komputera równoległego w architekturze MIMD, wraz z lokalną pamięcią. Krawędzie oznaczają dwukierunkowe kanały komunikacyjne pomiędzy procesorami i opisują topologię systemu. Zwykle zakłada się, że procesory mają tę samą moc obliczeniową, a komunikacja przez kanały nie zużywa czasu procesora (procesorów).

Program równoległy jest reprezentowany przez skierowany acykliczny graf ważony, zwany *grafem sekwencji zadań* (*precedence task graph*) lub *grafem programu*. Węzły w tym grafie reprezentują zadania (procesy) elementarne, które są wykonywane w kolejności opisanej przez graf. Wagi węzłów oznaczają czas przetwarzania każdego z zadań na jednym procesorze systemu multiprocesorowego, zaś wagi krawędzi – czas komunikacji między zadaniami, jeśli zadania wykonują się w sąsiednich procesorach (jeśli zadania wykonują się w tym samym procesorze, to czas komunikacji między nimi wynosi 0). Celem harmonogramowania jest takie przydzielenie zadań do procesorów, aby została zachowana właściwa kolejność wykonywania zadań oraz całkowity czas wykonania programu był jak najmniejszy (Święcicka, Seredyński i Jażdżyk 2001).

Problemy typu job-shop scheduling

Problemy typu job-schop scheduling (JSS) są ważne z praktycznego punktu widzenia. W ogólnym przypadku dana jest pewna liczba *prac* (*jobs*) do wykonania na pewnej liczbie maszyn (często podawany jest przykład obróbki części w fabryce samochodów). Każda z prac składa się z *czynności* (*tasks*) i poszczególne czynności

muszą być wykonywane przez odpowiednie maszyny. W danej chwili każda maszyna może obrabiać tylko jedną część i żadna część nie może być obrabiana przez dwie maszyny jednocześnie. Zadanie polega na znalezieniu harmonogramu czynności minimalizującego zadane kryterium (np. całkowity czas pracy maszyn). W ogólnej postaci problemu JSS kolejność wykonywanych czynności jest istotna, w odróżnieniu od problemu *open-schop scheduling* (OSS), gdzie może być dowolna. Najbardziej złożonym przypadkiem jest *harmonogramowanie dynamiczne* (*dynamic scheduling*), zwane też często *reharmonogramowaniem* (*rescheduling*), gdzie w czasie pracy maszyn mogą pojawiać się dodatkowe czynności do wykonania oraz zadany jest czas, w którym należy zrealizować daną czynność. W zależności od tego, czy zmiany pojawiają się w ustalonych chwilach, czy też losowo, mamy do czynienia z reharmonogramowaniem deterministycznym bądź stochastycznym. Jako, że teoretycznie proces reharmonogramowania może trwać w nieskończoność (gdy ciągle pojawiają się nowe czynności do wykonania), stosuje się inne niż w przypadku problemów OSS i JSS kryteria oceny jakości rozwiązania. Wszystkie opisane wyżej problemy są NP-trudne (Fang, Ross i Corne 1993; Fang, Ross i Corne 1994).

Harmonogramowanie łapania kurczaków

Zagadnienie harmonogramowania łapania kurczaków (*chicken catching scheduling*) zostało opracowane na podstawie rzeczywistego problemu szkockiego przedsiębiorstwa drobiarskiego. Przedsiębiorstwo ma kilka fabryk (o danej wydajności), z których każda dysponuje pewną liczbą ciężarówek określonej (często różnej) pojemności wraz z kierowcami oraz pewną liczbą drużyn „łapaczy". Drużyny łapaczy mogą pracować na pół lub pełny etat na jednej z trzech zmian i mają określoną wydajność łapania. Podobnie określany jest maksymalny czas pracy kierowców. Zadanie polega na takim zaplanowaniu pracy łapaczy i kierowców, poruszających się między fabrykami a położonymi w różnych miejscach farmami, aby zapewnić ciągłą pracę fabryk. Jednocześnie muszą być spełnione różnego typu ograniczenia (kierowcy nie mogą zbyt długo siedzieć za kierownicą bez przerwy, ptaki nie mogą zbyt długo przebywać w klatkach transportowych, należy unikać bezczynności łapaczy i kierowców itp.) (Hart, Ross i Nelson 1998; Hart, Ross i Nelson 1998).

1.5 Podejścia do problemu i do rozwiązania

Jeżeli układanie ma na celu znalezienie jakiegokolwiek planu spełniającego wszystkie zdefiniowane ograniczenia, problem jest wyrażony jako *problem poszukiwania* (*search problem*). Jeżeli jednak poszukujemy planu, który spełnia wszystkie *silne* ograniczenia (czyli takie, które *muszą* być spełnione, aby plan był akceptowalny) oraz minimalizuje (lub maksymalizuje) funkcję celu, zawierającą ograniczenia *słabe*, problem wyrażony jest jako *problem optymalizacji* (*optimalization problem*). W obu przypadkach definiuje się *problem podstawowy* (*underlying problem*), który jest problemem zawyrokowania, czy istnieje rozwiązanie (w przypadku problemu poszukiwania) lub czy istnieje rozwiązanie, dla którego funkcja celu osiąga pewną

założoną wartość (w przypadku problemu optymalizacji). Zwykle problem podstawowy jest NP-zupełny, tak więc idealne rozwiązanie może być znalezione tylko w przypadkach o niewielkich rozmiarach (Schaerf 1995).
Wszystkie podejścia do rozwiązania opierają się na pomyśle, aby najpierw umieścić na planie zajęcia obłożone najsilniejszymi ograniczeniami, jednak różnią się rozumieniem pojęcia „najsilniejsze ograniczenia". Najprostsze podejście do rozwiązania imituje sposób rozumowania człowieka – *bezpośrednią heurystykę* (*direct heuristics*), opartą na *kolejnych przyrostach* (*successive augmentation*). Człowiek układa plan kolejno dodając jedne zajęcia po drugich, aż wypełni nimi cały plan. Historycznie, rozwiązania oparte na tym podejściu powstały najwcześniej. Potem zaczęto stosować bardziej ogólne metody, takie jak redukcja do dobrze poznanego problemu *kolorowania grafu*. Najpóźniej zaczęto stosować metaheurezy, np. przeszukiwanie tabu, symulowane wyżarzanie czy algorytmy genetyczne (Schaerf 1995).

1.6 Układanie automatyczne a interaktywne

Opinie w kwestii, czy układanie planów może być w pełni automatyczne, różnią się z dwóch powodów z dwóch powodów: po pierwsze, czasem trudno wyjaśnić w sposób zrozumiały dla programu komputerowego, dlaczego jeden plan jest lepszy od innego; po drugie, ponieważ zwykle przestrzeń poszukiwań jest bardzo duża, interwencja człowieka może pchnąć poszukiwania we właściwym (lub choćby tylko obiecującym) kierunku, którego system mógłby w ogóle nie zbadać, lub mógłby go zbadać po długim czasie. Powyższe argumenty przemawiają za budową systemów, które przynajmniej pozwalają manipulować swoim wyjściem. Niektóre systemy wymagają częstszej i dalej idącej interwencji człowieka, i nazywane są *interaktywnymi* (lub *półautomatycznymi*). Ich przeciwieństwem są systemy *automatyczne* (*batch*), które działają bez ingerencji ze strony człowieka (Schaerf 1995).

2. Modele stosowane w planowaniu

Poniższy rozdział zawiera opis ograniczeń i mierników jakości stosowanych w problemach planowania oraz matematyczną postać przykładowego problemu typu klasa-nauczyciel.

2.1 Możliwe ograniczenia i mierniki jakości w problemach planowania

Układanie planu zajęć na uczelni polega na przypisaniu zbioru wykładów i innych form zajęć do pewnej liczby miejsc (sal) i przedziałów czasowych. Podstawową różnicą między planem zajęć na uczelni a planem lekcji w szkołach podstawowych i średnich jest fakt, że zajęcia na uczelni mają często przypisane te same zbiory studentów, podczas gdy zbiory uczniów, przypisanych do lekcji w szkole są prawie zawsze rozłączne. Jeżeli dwa wykłady (lub inne formy zajęć) dzielą ten sam zbiór studentów, to nie mogą odbywać się w tym samym czasie. Jest to tzw. *ograniczenie silne*, czyli takie, które *musi* być spełnione aby plan był akceptowalny, w odróżnieniu

od *ograniczeń słabych* (*soft constraints*), takich jak np. zapewnienie odpowiednio długiej przerwy między zajęciami na posiłek lub dojazdy, których złamanie nie pociąga za sobą takich konsekwencji. Plan nazywany jest *osiągalnym* (*feasible*), gdy spełnione są dla niego wszystkie silne ograniczenia. Ważną rolę w przypadku zajęć na uczelni odgrywa wielkość, dostępność i wyposażenie sal wykładowych (laboratoriów itp.). Często też muszą być wzięte pod uwagę wzajemne relacje między kursami. Warianty tego problemu uwzględniają również zajęcia, w których uczestniczy więcej niż jedna grupa studentów, niedostępność określonych sal w pewnych terminach, przypisanie z góry zajęć do sal i (lub) godzin, uwzględnienie w planie przerw (na dojazdy, posiłki itp.) oraz układanie planów dla zajęć o różnym czasie trwania. Miarą jakości planu może być jego *zwartość* (*compactness*). Plan jest zwarty z punktu widzenia nauczyciela, jeżeli poszczególne jego zajęcia są pogrupowane razem tak bardzo, jak to jest możliwe. Podobnie można zdefiniować zwartość z punktu widzenia ucznia (studenta). Odwrotną do zwartości miarą jest *rozrzucenie* (*distribution*). Jeżeli jakieś zajęcia odbywają się pięć razy w tygodniu (np. języki, matematyka), to niepożądane jest, aby wszystkie zajęcia danego przedmiotu odbywały się w tym samym dniu. Odległość między zajęciami z danego przedmiotu to właśnie rozrzucenie. Model zwartości i rozrzucenia wraz z rozszerzonym modelem ograniczeń silnych można znaleźć w (Drexl i Salewski 1997). Zarówno zakres problemów branych pod uwagę podczas układania planu, jak rozróżnienie, które z wynikających z nich ograniczeń są silne, a które słabe zależy od specyfiki uczelni, dla której plan jest układany.

2.2 Postać matematyczna problemu układania planu lekcji

Poniższy model dotyczy „klasycznego" modelu klasa-nauczyciel w układaniu planów lekcji (słowo „klasa" występuje tu w znaczeniu grupy ludzi, a nie sali). Mamy dane m klas $c_1,..., c_m$, n nauczycieli $t_1,..., t_n$ oraz p terminów 1,..., p. Dodatkowo daną mamy macierz nieujemnych liczb całkowitych $R_{m\times n}$, zwaną *macierzą wymagań* (*requriements matrix*), gdzie r_{ij} jest ilością lekcji, jakie daje nauczyciel t_j klasie c_i. Problem polega na przypisaniu lekcji do terminów tak, aby żadna klasa i żaden nauczyciel nie uczestniczyli jednocześnie w więcej niż jednej lekcji. Postać matematyczna tego problemu przedstawia się następująco (de Werra 1997a):

$$\textit{znaleźć } x_{ijk} \qquad (i = 1,..., m; j = 1,..., n; k = 1,..., p)$$

$$\textit{takie, że } \sum_{k=1}^{p} x_{ijk} = r_{ij} \qquad (i = 1,..., m; j = 1,..., n), \tag{1}$$

$$\sum_{j=1}^{n} x_{ijk} \le 1 \qquad (i = 1,..., m; k = 1,..., p), \tag{2}$$

$$\sum_{i=1}^{m} x_{ijk} \le 1 \qquad (j = 1,..., n; k = 1,..., p), \tag{3}$$

$$x_{ijk} = 0 \text{ lub } 1 \qquad (i = 1,..., m; j = 1,..., n; k = 1,..., p), \tag{4}$$

gdzie $x_{ijk} = 1$ gdy klasa c_i i nauczyciel t_j spotykają się w terminie k, w przeciwnym przypadku $x_{ijk} = 0$.

Założenie (1) daje pewność, że każdy nauczyciel ma z każdą klasą pewną określoną liczbę lekcji. Założenie (2) daje pewność, że każdy nauczyciel ma co najwyżej jedną lekcję w każdym terminie. Założenie (3) zapewnia to samo w stosunku do klasy.

Dowiedziono, że istnieje rozwiązanie powyższego problemu, pod warunkiem, że żaden nauczyciel ani żadna klasa nie mają więcej niż p lekcji.

$$\sum_{i=1}^{m} r_{ij} \le p \qquad (j = 1,..., n), \tag{5}$$

$$\sum_{j=1}^{n} r_{ij} \le p \qquad (i = 1,..., m). \tag{6}$$

Większość teoretycznych rozwiązań i twierdzeń dotyczących powyższego problemu wyprowadzono, korzystając z redukcji instancji problemu do grafu. Szczegóły tego podejścia można znaleźć w rozdziale 3.4.

2.3 Problem układania planów lekcji jako zadanie optymalizacji

Problem opisany w rozdziale 2.2 jest problemem poszukiwania, którego rozwiązaniem jest każdy osiągalny plan. W rzeczywistości pewien osiągalny plan może być lepszy niż inne i celem jest znalezienie optymalnego. Takie podejście zmusza nas do sformułowania problemu planowania jako zadania optymalizacji z funkcją celu, którą będziemy minimalizować (lub maksymalizować).

Najprostsze i najstarsze historycznie jest podejście Jungingera. Postuluje on dodanie do problemu poszukiwania następującej funkcji celu

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=1}^{p} d_{ijk} x_{ijk}\,, \tag{7}$$

gdzie d_{ijk} jest przypisane do terminów k, w których lekcja nauczyciela t_j z klasą c_i jest mniej pożądana (Schaerf 1995).

W (Colorni, Dorigo i Maniezzo 1990a) zaproponowano bardziej złożoną funkcję celu, uwzględniającą kilka aspektów ułożonego planu. Funkcja taka brała pod uwagę następujące wielkości (ułożone według zmniejszającej się wagi): koszt dydaktyczny (np. rozrzucenie wykładów po różnych dniach tygodnia), koszt organizacyjny (np. posiadania nauczycieli zastępczych „w zapasie") i koszt personalny (np. konkretny dzień wolny dla danego nauczyciela). Jeszcze inne podejście opisane jest w (Ross, Corne i Fang 1994). Autorzy z każdym złamanym ograniczeniem wiążą określoną karę (np. za to, że nauczyciel musi nauczać w terminie, w którym jest niedostępny).

3. Przegląd metod automatycznego planowania

Rozdział ten zawiera krótki przegląd metod automatycznego planowania opisanych w ciągu ostatnich dziesięciu lat.

3.1 Metody heurystyczne

Bezpośrednie heurystyki wypełniają plan, wstawiając do niego po jednym zdarzeniu (wykładzie, grupie wykładów, egzaminie itd. – w zależności od wariantu problemu) tak długo, aż nie pojawi się żaden konflikt. Od tego momentu zamienia się zdarzenia miejscami (lub usuwa się je z planu) tak, aby znalazło się miejsce dla innych (Schaerf 1995).

Najprostszym podejściem heurystycznym w układaniu planów jest *metoda porządkowa* (*sequential method*), której podstawą jest pewna określona *strategia porządkowania* zdarzeń (*event sequencing strategy*). Metoda ta używa heurezy do określenia, jak trudne do zaplanowania byłoby rozważane zdarzenie, aby można było ułożyć zdarzenia w kolejności malejącej trudności. Oprócz tego podobnej strategii można użyć do ułożenia w kolejności dostępnych terminów.

Heurystyczna metoda z nawrotami (*heuristic backtracking method*) przedstawiona np. w (Burke, Newall i Weare 1998) jest typową metodą porządkową, zastosowaną do testowego problemu układania planu egzaminów. Autorzy proponują trzy strategie porządkowania zdarzeń: *largest degree first*, gdzie jako pierwsze są umieszczane na planie zdarzenia z największą liczbą zdarzeń kolidujących, *largest colour degree first*, w której w pierwszej kolejności są umieszczane na planie zdarzenia kolidujące z największą liczbą zdarzeń już zaplanowanych, oraz *least saturation degree firs*), gdzie na początku planowane są zdarzenia z najmniejszą liczbą dostępnych w danym momencie możliwych terminów. Po wybraniu strategii porządkowania w każdej iteracji algorytm umieszcza pierwsze w kolejności zdarzenie na planie w takim terminie, aby wartość funkcji celu była jak najmniejsza. W ostatnich dwóch strategiach w przypadku, gdy takich terminów jest wiele, strategia largest degree first jest używana do rozstrzygnięcia konfliktów. W przypadku, gdy konflikty nadal występują, wybierany jest najwcześniejszy termin. (Corne i Ross 1995) nazywają to strategią *„first-fit”*, natomiast w strategii *„best-fit”* wybierany jest najwcześniejszy termin, do którego nie przypisano jeszcze żadnych zdarzeń. W tej samej pracy proponowany jest także losowy wybór terminu ze zbioru konfliktów.

Jeżeli nie ma możliwych terminów, należy usunąć kilka (w szczególności jedno) zdarzeń z planu. Każdy z terminów (nie tylko tych z przypisanymi zdarzeniami) jest badany pod kątem przynależności do jednej z dwóch grup:

a. Zdarzenie nie może być zaplanowane w tym terminie, nawet jeżeli nie byłoby kolizji z innymi zdarzeniami w tym lub innych terminach.
b. Zdarzenie nie może być zaplanowane w tym terminie, o ile z planu nie zostanie usunięta pewna liczba zdarzeń.

W przypadku a) termin jest odrzucany. Z terminów, należących do grupy b) wybierany jest ten, który ma najmniejszy koszt (czyli zaplanowanie zdarzenia w tym

terminie wiąże się z usunięciem z planu najmniejszej liczby innych zdarzeń). Dodatkowo istnieje mechanizm zapobiegający tworzeniu się pętli bez końca. Polega on na tym, że jeżeli zaplanowanie zdarzenia e_1 spowodowało usunięcie zdarzenia e_2 z terminu p, to taka operacja nie może zostać wykonana po raz drugi (ten termin dla zaplanowania zdarzenia e_1 nie jest dłużej brany pod uwagę).

Metody porządkowe mają pewną wadę – dają tylko jedno rozwiązanie, co może oznaczać, że istnieje inne, równie dobre lub nawet lepsze. Aby je znaleźć można zmieniać strategie porządkowania w kolejnych przebiegach algorytmu lub wprowadzić do czysto heurystycznej metody elementy niedeterministyczne. W (Burke, Newall, Weare 1998) zaproponowano dwie metody wyboru zdarzenia do zaplanowania: *selekcję turniejową* (*tournament selection*), gdzie generowany jest losowy podzbiór o określonej wielkości zbioru jeszcze nie zaplanowanych zdarzeń, a następnie stosuje się heurystykę do wybrania „najlepszego" (pierwszego w kolejności) zdarzenia z tego podzbioru, oraz *selekcję progową* (*bias selection*), gdzie zdarzenia są układane zgodnie ze strategią porządkowania, a następnie losowo wybierane jest jedno z pewnej liczby najlepszych. Dalsza część algorytmu pozostaje bez zmian. Autorzy wykazali wyższość metod łączących heurystyki z elementami losowymi nad metodami czysto heurystycznymi oraz podejściem z całkowicie losową strategią porządkowania. Chociaż heurystyczne metody niedeterministyczne powodują relatywnie małą poprawę funkcji celu (do 25%) w stosunku do metod czysto heurystycznych i są bardziej czasochłonne, to autorzy podkreślają, że rozwiązania były szeroko rozrzucone po dużej przestrzeni i metody te mogą stanowić dobry kompromis między technikami porządkowymi a bardziej skomplikowanymi i czasochłonnymi podejściami, takimi jak algorytmy genetyczne czy metody przeszukiwania lokalnego.

We wspomnianej wyżej pracy (Corne i Ross 1995) opisują nieco inne podejście. Rozróżniają oni dwa rodzaje algorytmów, różniących się strategią wyboru terminu, do którego zostanie przypisane zdarzenie, spośród terminów spełniających ograniczenia silne i/lub słabe (rozumiane tak, jak to opisano w rozdziale 2.1). W algorytmach *„zachłannych"* (*„greedy"*) wybór terminu odbywa się zgodnie z którąś z wymienionych strategii (first-fit, next-fit bądź losowo) oraz *„głodnawe"* (*„peckish"*), podobne do opisanej wyżej selekcji turniejowej, gdzie ze zbioru *wszystkich* terminów losowana jest pewna ich liczba k i wybierany jest ten, w którym zaplanowanie zdarzenia spowoduje najmniej konfliktów. Liczba ta została nazwana *„zachłannością"* (*„greedness"*) algorytmu. Dla k=1 wybór jest całkowicie losowy, gdy k zbliża się do liczby wszystkich możliwych terminów, algorytm staje się zachłanny. Dodatkowo algorytmy podzielono na *jednorodne* (*uniform*), gdzie brane są pod uwagę zarówno silne, jak i słabe ograniczenia, i *osiągalne* (*feasible*), gdzie słabe ograniczenia są pomijane. Autorzy proponują wymienione algorytmy jako sposoby inicjalizacji populacji dla metod poszukiwania opartych o populacje, takich jak algorytmy genetyczne czy niektóre metody przeszukiwania lokalnego.

Inną metodą w której wykorzystywane są heurystyki jest *metoda badania skupisk* (*cluster method*). Polega ona na grupowaniu zdarzeń w skupiska, w których zdarzenia

nie wchodzą w konflikty z innymi zdarzeniami w ramach skupiska. Głównymi problemami w tym podejściu jest właściwe przypisanie zdarzeń do skupisk i skupisk do terminów (najczęściej dąży się do minimalizacji liczby konfliktów między sąsiednimi w sensie terminów skupiskami). Podejście to ma poważne ograniczenia, gdyż po ustaleniu zawartości skupisk ułożenie dobrego jakościowo planu może być niemożliwe. Przegląd metod opartych o skupiska można znaleźć w (Newall 1999).

3.2 Metody przeszukiwania lokalnego (local search techniques)

Jest to grupa metod ogólnego zastosowania do rozwiązywania zadań optymalizacji. Wszystkie bazują na pojęciu *sąsiada*. Jeżeli rozważymy zadanie optymalizacji z przestrzenią rozwiązań *S* oraz funkcją celu *f*, którą będziemy minimalizować, to funkcja *N*, której postać jest zależna od struktury problemu, przypisuje do każdego rozwiązania $s \in S$ jego *sąsiedztwo* $N(s) \subseteq S$. Każde rozwiązanie $s' \in N(s)$ jest nazywane sąsiadem *s*.

Algorytmy przeszukiwania lokalnego rozpoczynają działanie od pewnego początkowego rozwiązania s_0, które jest ustalane losowo, lub otrzymywane w wyniku zastosowania innych metod. Następnie wchodzą w pętlę, w której poruszają się po przestrzeni rozwiązań, przechodząc od rozwiązania do jednego z jego sąsiadów. Modyfikację transformującą rozwiązanie w jeden z jego sąsiadów nazywamy *ruchem* (*move*). Jedną z metod przeszukiwania lokalnego jest metoda *steepest descendent*, która analizuje wszystkie możliwe ruchy i wybiera ten, dla którego wartość funkcji celu jest najmniejsza. Kandydat jest akceptowany tylko w przypadku, gdy wartość funkcji celu jest mniejsza niż poprzednia. Algorytm zatrzymuje się, gdy funkcja celu osiągnie minimum lokalne. Metoda ta wymaga przeszukania całego sąsiedztwa bieżącego rozwiązania, w odróżnieniu od metody *randomized descendent*, która wybiera losowo sąsiada i przyjmuje go za nowe rozwiązanie, jeśli zmniejsza on wartość funkcji celu. Jeśli tak nie jest, losowany jest kolejny sąsiad. Algorytm kończy się po ustalonej liczbie iteracji bez zmiany wartości funkcji celu. Podobnie jak technika steepest descendent zatrzymuje się po osiągnięciu lokalnego minimum. Modyfikacją metody randomized descendent jest algorytm *randomized non-ascendent,* (*RNA*), który akceptuje losowego sąsiada, jeśli wartość funkcji celu jest mniejsza lub równa bieżącej. Ten algorytm również kończy się po ustalonej liczbie iteracji bez zmniejszenia wartości funkcji celu. Metoda ta może być ulepszona poprzez wprowadzenie *ruchów bocznych* (*sideways moves*) (Selman, Levesque i Mitchell 1992, za Schaerf 1996), dzięki czemu może poruszać się w kierunku minimum przez *plaetau* (płaskie obszary).

Metoda *steepest non-ascendent* łączy elementy metody steepest descendent z ruchami bocznymi. Wybór między dwoma ruchami jednakowo minimalizującymi funkcję celu jest losowy.

W p*rzeszukiwanie tabu* (*tabu search*) algorytm przegląda cały zbiór sąsiadów bieżącego rozwiązania. Następnie przyjmuje za bieżące rozwiązanie ten element zbioru, dla którego wartość funkcji celu jest najmniejsza, niezależnie od tego, czy wartość ta jest lepsza czy gorsza od poprzedniej. Aby uniknąć cykli istnieje tzw. *lista*

tabu (*tabu list*), która jest listą ruchów, których nie wolno wykonać. Lista ta jest zbudowana na zasadzie tonącego stosu (zawiera określoną liczbę poprzednich ruchów; kiedy kolejny ruch jest dodawany do listy, najstarszy jest usuwany). Oprócz tego istnieje mechanizm, który pozwala ominąć tabu – jeżeli ruch niesie za sobą *znaczną* poprawę funkcji celu, to pomija się jego status tabu i nowe rozwiązanie jest akceptowane. Precyzyjniej rzecz ujmując, definiowana jest *funkcja aspiracji* (*aspiration function*) A, która dla każdej wartości v funkcji celu wyznacza wartość v', która reprezentuje wartość, którą algorytm chciałby osiągnąć po v. Jeżeli przyjmiemy bieżące rozwiązanie s, funkcję celu f i rozwiązanie sąsiednie s', uzyskane przez wykonanie ruchu m, to jeżeli $f(s') \leq A(f(s))$, to s' zostanie zaakceptowane, nawet jeżeli m jest na liście tabu. Algorytm kończy się po określonej liczbie iteracji bez poprawy wartości funkcji celu lub gdy bieżące rozwiązanie osiągnie określoną wartość funkcji celu (Schaerf 1996; Schaerf 1995; Weare 1995; Newall 1999).
Symulowane wyżarzanie (*simulated annealing*) zawsze akceptuje wybranego losowo sąsiada, jeżeli wartość funkcji celu jest dla niego równa lub lepsza od bieżącej. Jeżeli tak nie jest, to nowe rozwiązanie jest akceptowane z prawdopodobieństwem równym $e^{-\Delta/T}$, gdzie Δ jest różnicą między wartością funkcji celu dla nowego i bieżącego rozwiązania, a T parametrem, zwanym temperaturą. Na początku działania algorytmu temperatura jest ustawiana na odpowiednio wysoką wartość T_o. Po określonej liczbie iteracji temperatura jest obniżana o *stopień schładzania* (*cooling rate*) a, taki że $T_n = a * T_{n-1}$, gdzie $0 \leq a \leq 1$. Algorytm kończy się, gdy temperatura osiągnie wartość bliską 0 i nie będą już właściwie akceptowane żadne zmiany (system jest „zamrożony) (Schaerf 1996; Schaerf 1995; Ross i Corne 1995; Michalewicz 1999; Weare 1995; Newall 1999; Thompson i Dowsland 1998). W (Thompson i Dowsland 1998) autorzy zastosowali symulowane wyżarzanie do rozwiązania problemu układania planu egzaminów. Jako że zarówno stopień schładzania, jak i postać funkcji sąsiedztwa mają duży wpływ na efektywność i jakość rozwiązania w metodach przeszukiwania lokalnego, zaproponowano rozwiązanie problemu w dwóch fazach – pierwsza daje plan osiągalny, w drugiej z przestrzeni rozwiązań usuwane są wszystkie nieosiągalne rozwiązania (co pozwala na jej zawężenie), a pozostała przestrzeń jest przeszukiwana w celu znalezienia rozwiązania optymalizującego funkcję ograniczeń słabych. Dodatkowo postać funkcji sąsiedztwa w fazie drugiej była dużo bardziej złożona, niż w fazie pierwszej (tzw. *Kempe chain neighbourhood*). Podejście to okazało się słuszne, autorzy wykazali ponadto, że przydatność sąsiedztwa typu Kempe chain nie ogranicza się tylko do optymalizacji ograniczeń słabych, ale może także dawać dobre rozwiązania w przypadkach, gdzie znalezienie osiągalnego planu jest bardzo trudne.

3.3 Algorytmy genetyczne

Algorytmy genetyczne (AG) są narzędziem optymalizacyjnym ogólnego zastosowania. Przy ich zastosowaniu można uzyskać dobre jakościowo rozwiązania nawet przy zwiększającym się rozmiarze problemu i z tego powodu znalazły wiele

różnorodnych zastosowań. Dobre wprowadzenie do problematyki algorytmów genetycznych można znaleźć w (Michalewicz 1999).

Typowy AG rozpoczyna działanie od losowego wygenerowania początkowego zbioru rozwiązań $\{s_1^0, \ldots, s_n^0\}$, zwanego *populacją* w chwili 0. Następnie powtarzana jest w pętli procedura, tworząca populację $\{s_1^{t+1}, \ldots, s_n^{t+1}\}$ w chwili t+1 z populacji w chwili t. Aby to uczynić, obliczana jest wartość funkcji celu dla każdego rozwiązania, a następnie losowo wybierane jest n (niekoniecznie różnych) rozwiązań w czasie t. Prawdopodobieństwo wylosowania danego elementu populacji jest uzależnione od wartości funkcji celu dla tego rozwiązania tak, że rozwiązania o wyższej wartości funkcji (przy założeniu, że staramy się ją zmaksymalizować) mają większe prawdopodobieństwo wylosowania. Następnie wybiera się z określonym prawdopodobieństwem rozwiązania do *krzyżowania*, polegającego na przemieszaniu dwóch rozwiązań przez zamianę miejscami odpowiadających sobie fragmentów ich reprezentacji. Dodatkowo rozwiązania mogą być (z określonym prawdopodobieństwem) poddane *mutacji*. Mutacja zmienia losowo pewną część rozwiązania. Algorytm kończy działanie gdy wykonana zostanie określona liczba iteracji lub najlepsze rozwiązanie osiągnie (bądź przekroczy) założoną wartość funkcji celu, albo też przez określoną liczbę iteracji wartość funkcji celu nie ulegnie poprawie. Modyfikacją algorytmów genetycznych są *algorytmy mimetyczne*. W algorytmach mimetycznych po rekombinacji (krzyżowaniu i mutacji) następuje faza przeszukiwania lokalnego – rozwiązania są kolejno poddawane działaniu wybranego algorytmu, aby poprawić ich jakość przed kolejną iteracją właściwego AG. Opis takiego podejścia wraz z danymi eksperymentalnymi można znaleźć w (Newall 1999). Autor, oprócz przeszukiwania lokalnego, traktowanego jak operator genetyczny, wprowadza pojęcia „lekkiej" (*light*) i „ciężkiej" (*heavy*) mutacji (mutacja lekka mutuje zdarzenie w ramach terminu, ciężka zamienia całe terminy). W pracy można również znaleźć porównanie różnych heurystyk przeszukiwania, prosty algorytm heurystyczny oraz opis wielofazowego algorytmu ewolucyjnego.

Od początku lat dziewięćdziesiątych prowadzone są intensywne badania nad zastosowaniem algorytmów genetycznych w rozwiązywaniu problemów planowania. Opis pierwszych prób wykorzystania AG do układania planu lekcji można znaleźć w (Colorni, Dorigio i Maniezzo 1990a) oraz (w rozszerzonej wersji) w (Colorni, Dorigio i Maniezzo 1990b). Autorzy zastosowali macierzową reprezentację osobnika (rozwiązania). Wiersz takiej macierzy odpowiada nauczycielowi, kolumna terminowi, a jej elementami są klasy. Użyto trzech operatorów genetycznych: mutacji rzędu k (operator wybiera dwa sąsiednie ciągi k-elementowe z tego samego wiersza macierzy i zamienia je miejscami), mutacji dni (zamienia miejscami dwie grupy kolumn macierzy) oraz krzyżowania (dla danych dwóch macierzy operator ustawia wiersze pierwszej macierzy w porządku malejących wartości tzw. *lokalnej funkcji celu*, która jest składową funkcji celu związaną tylko z charakterystyką danego nauczyciela). Uwzględniono wyłącznie silne ograniczenia. Powstały program został z powodzeniem zastosowany na dużej uczelni we Włoszech. Dwa lata później w (Colorni, Dorigio i Maniezzo 1992) przedstawiono analizę porównawczą różnych sposobów układania

zastosowanych do tego samego problemu w terminach funkcji celu – układania ręcznego, algorytmu genetycznego, mimetycznego, symulowanego wyżarzania, symulowanego wyżarzania z reinicjacją (temperatura „zamrożonego" systemu była ponownie ustawiana na wartość początkową) oraz przeszukiwania tabu (również w wersji z ponowną inicjacją, tym razem listy tabu). Algorytm genetyczny, a zwłaszcza mimetyczny, okazał się dobrą alternatywą dla przeszukiwania tabu, tylko trochę pozostając w tyle zarówno pod względem jakości najlepszego rozwiązania, jak i średniej jego jakości. Autorzy podkreślają jednak niezwykłą elastyczność AG. Pozostałe metody były zdecydowanie gorsze. We wspomnianej pracy wprowadzono również specjalny operator, zdolny przekształcać nieosiągalne plany w osiągalne.

Podobną analizę porównawczą AG, symulowanego wyżarzania oraz wariantu algorytmu RNA przeprowadzili autorzy (Ross i Corne 1995). W terminach funkcji celu AG okazał się gorszy od metod „klasycznych", jednak jego elastyczność (którą autorzy mierzą jako odległość Hamminga między różnymi rozwiązaniami tego samego problemu proponowanymi przez algorytm) okazała się dużo większa. W pracy tej podobnie jak we wcześniejszej (Ross, Corne i Fang 1994) autorzy zastosowali zupełnie inną reprezentację osobnika, nazwaną przez nich *bezpośrednią* (*direct*). Osobnik jest wektorem symboli o długości $3v$ (gdzie v jest liczbą zdarzeń), podzielonym na trójki. Poszczególne elementy w trójce oznaczają kolejno czas, miejsce i nauczyciela. Operatory genetyczne zostały dostosowane do takiej reprezentacji, autorzy zaproponowali również operator mutacji, który wybiera losowo kilka (a nie jedną, jak proponowano wcześniej) wartości mutowanej części rozwiązania, i przyjmuje za nowe rozwiązanie tę, która daje największą poprawę funkcji celu. Uwzględniono zarówno ograniczenia silne, jak i słabe. Powstały program zastosowano na jednej z angielskich uczelni z dobrym rezultatem.

3.4 Redukcja do kolorowania grafu

Podstawy teorii grafów można znaleźć w (Ross i Wright 1996), (Wilson 2000) oraz w (Thulasiraman i Swamy 1992). Problem kolorowania węzłów grafu jest jednym z klasycznych NP-zupełnych problemów dotyczących grafów. Dla danego grafu $G=(V, E)$, gdzie V jest zbiorem węzłów, a E zbiorem krawędzi, problem polega na znalezieniu podziału V na jak najmniejszą liczbę klas kolorów takich, że żadne dwa wierzchołki nie będą należały do tej samej klasy, jeśli istnieje krawędź między nimi (Wilson 2000).

Metody oparte o redukcję do kolorowania grafu są szeroko stosowane w rozwiązywaniu problemów układania planów i harmonogramowania. Takie podejścia są nazywane *harmonogramowaniem chromatycznym* (*chromatic scheduling*) (de Werra 1997). W typowych problemach układania planów szkolnych poszczególnym zdarzeniom odpowiadają węzły, istnienie krawędzi między dwoma węzłami oznacza, że przypisane tym węzłom zdarzenia nie mogą być zaplanowane w tym samym terminie. Kolor węzła reprezentuje termin. Taki model jest dobry dla prostych zadań układania planów, może być jednak rozszerzony tak, aby obejmował także ograniczenia słabe. Rozszerzenia takie zostały zaproponowane między innymi w (de

Werra 1996) i rozwinięte w (de Werra i Mahadev 1997) oraz (de Werra 1997a) – przedstawienie wymagań w postaci multigrafu pozwala uwzględnić zdarzenia z góry przypisane do zbioru możliwych terminów (w szczególności do jednego) i niedostępność określonych terminów dla pewnych zdarzeń. W (de Werra 1997a) przedstawiono również sposoby na zapewnienie, że plan będzie zwarty, przez zastosowanie kolorowania krawędzi (problem kolorowania krawędzi polega na znalezieniu podziału E na jak najmniejsza liczbę klas kolorów tak, aby żadne dwie krawędzie nie należały do tej samej klasy, jeżeli dzielą wspólny wierzchołek) (Wilson 2000). W (de Werra 1997b), jak również w (de Werra i Mahadev 1997) opisana jest koncepcja zastosowania *kolorowania ograniczonego* (czyli takiego, gdzie każdy z węzłów lub krawędzi ma przypisany zbiór kolorów, które może przyjąć) do wyrażenia niedostępności terminów i przypisania z góry. Ostatnia praca zawiera również propozycję bardziej ogólnego sposobu na radzenie sobie z przypisaniem z góry – kolorowanie krawędzi z uwzględnieniem kosztu. Praca (de Werra 1996), podobnie jak (de Werra, Hoffman, Mahadev i Peled 1996) dotyczy głównie problemu job-schop scheduling, znajdują się tam jednak (pierwsza praca) uniwersalne modele, pozwalające wyrazić wymagania co do jednoczesności zdarzeń oraz ograniczenia dotyczące zasobów (np. wielkości sal) oraz (druga praca) opis reprezentacji za pomocą drzew oraz nieco prostszej, ale mniej uniwersalnej metody uwidaczniania przypisania z góry za pomocą wcześniejszego pokolorowania określonych wierzchołków (krawędzi). Praca (de Werra 1999) opisuje złożoną reprezentację za pomocą grafów, która może być zastosowana w problemach układania planów z wieloma ograniczeniami. Sam autor przyznaje jednak, że opisywane przez niego przypadki są wciąż dalekie od rzeczywistości i zbyt mało ogólne. Dlatego w dalszej części niniejszego opracowania wymienione są tylko podejścia oparte na kolorowaniu węzłów w grafach prostych. Wspomnieć warto jednak wcześniej pracę (Hilton, Slivnik i Stirling 2001), gdzie opisana jest reprezentacja za pomocą multigrafu z pętlami i rozwiązanie z zastosowaniem kolorowania krawędzi problemu układania planu w szkole z wieloma ograniczeniami słabymi (przypisanie z góry i niedostępność w terminie zarówno dla nauczycieli, jak i zajęć) i miernikami jakości (zarówno zwartość, jak i miara rozrzucenia zadań po planie). Autorzy zaproponowali tam również skomplikowaną koncepcję *conference scheduling* (układania planu konferencji).

Istnieje kilkanaście algorytmów i heurystyk stosowanych do rozwiązywania problemów kolorowania grafów. Część z nich została opisana w rozdziale dotyczącym metod heurystycznych, jako że kolejność kolorowania wierzchołków odpowiadają w zadaniach układania planów strategiom porządkowania zdarzeń. Przegląd metod stosowanych w kolorowaniu grafów znajduje się w (Weare 1995). Do kolorowania grafów mogą być również stosowane metody przeszukiwania lokalnego oraz metody hybrydowe. W (Burke, Elliman i Weare 1994) autorzy zastosowali kompilację dwóch zaawansowanych metod heurystycznych do rozwiązania problemów układania planu egzaminów oraz układania planu zajęć na uczelni wraz z rozszerzeniami obejmującymi słabe ograniczenia. Autorzy (Mausser i Magazine 1996) proponują

własną metodę heurystyczną do rozwiązania jednego z wariantów problemu układania planu egzaminów. W pracy (Čangalović, Kovačević-Vujčić, Ivanović i Dražić 1998) wykorzystano połączenie metody heurystycznej z przeszukiwaniem tabu, co dało bardzo dobre efekty.

3.5 Inne metody

Wśród innych podejść do rozwiązania problemu planowania należy wymienić metody oparte o przepływ w sieciach (*network flow techniques*). Sieć zdefiniowana jest jako skierowany graf ważony, w którym wagi są nieujemnymi liczbami rzeczywistymi (waga krawędzi nazywana jest jej *przepustowością*). Oprócz tego w sieci wyróżnione są dwa wierzchołki: *źródło* -wierzchołek, do którego nie wchodzi żadna krawędź (ma tylko krawędzie wychodzące), oraz *ujście* – wierzchołek, z którego nie wychodzą żadne krawędzie. Wiele problemów planowania i harmonogramowania zostało zredukowane do sekwencji przepływów w sieciach. Jednak ze względu na istnienie mniej skomplikowanych i bardziej uniwersalnych modeli, nie prowadzi się obecnie zbyt wielu badań w tym kierunku. Kompendium wiedzy o sieciach i ich zastosowaniach można znaleźć w (Dolan i Aldous 1993).

Innym interesującym pomysłem jest zastosowanie sieci neuronowych w rozwiązywaniu problemów planowania. Podstawy teorii sieci neuronowych można znaleźć np. w (Osowski 1994) lub (Tadeusiewicz 1993). Sieci neuronowe są uważane za dobre narzędzie do rozwiązywania problemów optymalizacji. W latach dziewięćdziesiątych pojawiły się propozycje wykorzystania ich na polu planowania. Najczęściej proponowano, aby sieć była narzędziem optymalizującym wynik działania innego algorytmu, np. kolorowania grafu. Przykład takiego podejścia można znaleźć w (Mausser i Magazine 1996), wraz z porównaniem z metodą czysto heurystyczną. Co prawda jakość planów produkowanych przez sieć neuronową była lepsza od tych uzyskanych metodami heurystycznymi, jednak jak przyznają autorzy, w praktyce podejście to jest nieefektywne ze względu na duże zapotrzebowanie sieci neuronowych na moc obliczeniową. Metody oparte na sieciach neuronowych są w planowaniu i harmonogramowaniu uważana jest za skomplikowane i stosunkowo mało efektywne, dlatego prace na ten temat są niezbyt liczne.

Warto wspomnieć również o metodach opartych o systemy ekspertowe. W podejściach tych buduje się systemy z bazą wiedzy w postaci regułowej. Reguły opisują zasady układania planów – w jakiej kolejności umieszczać zdarzenia na planie, jak rozwiązywać konflikty itp. Na podstawie zadanych parametrów (zdarzeń, terminów, zasobów i dotyczących ich ograniczeń) system układa plan, korzystając z bazy reguł. Opis takiego rozwiązania można znaleźć w (Lee i Wu 1995) – przedstawiony tam system CLXPERT pomaga układać plany dla uczelni na podstawie danych o nauczycielach, salach i kursach oraz bazy ponad 500 reguł pozyskanych od ekspertów. System działa efektywnie na jednym z tajwańskich uniwersytetów. Nieco inne podejście zaproponowano w (Burke, MacCarthy, Petrovic i Qu 2000). Opisany tam system zawiera bazę wcześniejszych przypadków i stara się znaleźć w niej przypadek najbliższy zadanym założeniom, a następnie zaadaptować go do bieżących

potrzeb stosując algorytm wnioskowania na podstawie przypadków – *case based reasoning (CBR*). Duży nacisk położono na ponowne użycie istniejących przypadków. Do przechowywania przypadków w bazie wiedzy zastosowano rozszerzoną reprezentację grafową (tzw. *grafy atrybutów*). Dzięki temu można było wykorzystać własności dotyczące izomorfizmu w grafach. System nie został jeszcze zastosowany do rozwiązania realnego problemu, ani jego efektywność nie została porównana z innymi metodami. Należy podkreślić, że podejścia oparte o systemy ekspertowe są historycznie najmłodsze wśród metod rozwiązywania problemów harmonogramowania i planowania (pierwsze prace na ten temat pojawiły się na początku lat dziewięćdziesiątych) i z pewnością wymagają jeszcze wielu badań. Jednak już teraz można stwierdzić, że wyniki prób są obiecujące, chociażby ze względu na stosunkowo krótki czas oczekiwania na rozwiązanie.

4. Układanie planu dyżurów na oddziale szpitalnym przy zastosowaniu algorytmu genetycznego

Poniższy rozdział zawiera opis rozwiązania prostego problemu planowania dyżurów personelu w służbie zdrowia przy pomocy algorytmu genetycznego.

4.1 Opis rzeczywistości

Na oddziale w szpitalu pracuje na stałe kilkunastu lekarzy. Dodatkowo czasem pomaga im kilku anestezjologów z innego szpitala. Plan dyżurów układa się w cyklu miesięcznym. Każdego dnia dyżur musi pełnić trzech lekarzy – anestezjolog i dwóch chirurgów. Co najmniej jeden z chirurgów na dyżurze musi mieć drugi stopień specjalizacji (w praktyce musi być oprócz tego doświadczony i odpowiedzialny). Według przepisów nie można brać kilku dyżurów pod rząd – przy dwóch lekarz spędza w szpitalu 56 godzin bez przerwy. W praktyce jednak takie przypadki są dość powszechne – układający dyżury starają się unikać sytuacji, gdy lekarz miałby trzy dyżury pod rząd (78 godzin w pracy !), ale takie przypadki również się zdarzają. Przed rozpoczęciem układania planu przeprowadza się rozmowy ze wszystkimi lekarzami. Określają oni ile mniej więcej chcą mieć dyżurów, kiedy je chcą lub nie chcą mieć, a także z kim chcieliby lub nie chcieliby dyżurować. Życzenia lekarzy najczęściej kolidują ze sobą, więc konieczna jest pewna swoboda w odbieraniu i dodawaniu dyżurów. Po pierwsze dyżury powinny być rozdzielone mniej więcej sprawiedliwie – nie można na przykład dać komuś wszystkich dyżurów weekendowych, bo są one cenne (płaci się za nie podwójnie, podobnie jak za dyżury we wszystkie dni ustawowo wolne). Po drugie, według przepisów lekarza nie można zmusić do odbycia więcej niż ośmiu dyżurów w miesiącu (a według zwyczaju kierujący zespołem raczej nie bierze więcej niż czterech). Po trzecie trzeba zostawić parę wolnych dyżurów dla lekarzy z innego szpitala, jednak ich termin może być ustalany bardziej elastycznie w stosunku do lekarzy „miejscowych". Po czwarte, układaniem dyżurów w Święta, Sylwestra itp. rządzą specjalne prawa, które nie są tutaj opisane, ze względu na ich marginalne znaczenie.

4.2 Używane dane i reprezentacja rozwiązania

Każdy lekarz ma przypisany odpowiadający mu następujący zestaw danych:

- *preferencje towarzyskie* – dla każdego innego lekarza określona jest jedna z trzech możliwych preferencji towarzyskich („tak", „nie" i „obojętna"),
- *preferencje czasowe* – dla każdego dnia miesiąca, na który plan jest układany, określona jest jedna z pięciu możliwych preferencji czasowych (w kolejności od najbardziej pożądanej: „koniecznie", „tak", „obojętnie", „nie" i „wykluczone"),
- minimalna i maksymalna liczba dyżurów w miesiącu, które chciałby mieć lekarz
- znaczniki oznaczające, czy lekarz jest anestezjologiem lub czy może pełnić dyżur samodzielnie.

Reprezentacja rozwiązania jest bezpośrednia. Genotyp osobnika ma długość $4d$, gdzie d jest liczbą dni w miesiącu, na który plan jest układany. Jest to powtórzona d razy krotka $<n, s_n^1, s_n^2, a_n>$, gdzie n jest identyfikatorem kolejnego terminu (dnia w miesiącu), , s_n^1 i s_n^2 są identyfikatorami odpowiednio pierwszego i drugiego chirurga, a a_n jest identyfikatorem anestezjologa.

4.3 Funkcja celu i użyty algorytm

Funkcja celu oparta jest na karze. Za każde złamane ograniczenie na termin, w którym ograniczenie zostało naruszone, nakładana jest kara. Kara nakładana jest tylko na ograniczenia dotyczące czasu i liczby dyżurów – dyżur w nieodpowiednim czasie (negatywna preferencja czasowa), przekroczona liczba dyżurów pod rząd lub przekroczona maksymalna liczba dyżurów – wtedy kara nakładana jest na pierwszy i każdy następny termin, w którym lekarz przekracza maksymalną liczbę dyżurów, pod warunkiem, że lekarz nie ma na ten termin ustawionej preferencji „koniecznie". Jeżeli tak jest, to algorytm nakłada karę, o ile jest to możliwe, na poprzedni cyklicznie termin z tym lekarzem. Podobnie jest w przypadku dyżurów pod rząd - dzięki sprawdzeniu poprzedniego i następnego terminu unikamy negatywnego oceniania sytuacji, gdy lekarz chce mieć 3 dyżury pod rząd na własne życzenie. Zsumowana ocena poszczególnych terminów oznacza ocenę planu jako całości. Algorytm ma za zadanie minimalizację funkcji celu.

Osobniki inicjowane są losowo, ale tak, aby zostały spełnione wszystkie silne ograniczenia:

- lekarze, których wzajemne preferencje towarzyskie będą ustawione na „tak", zawsze będą mieli dyżur ze sobą,
- lekarze, , których wzajemne preferencje towarzyskie będą ustawione na „nie", nigdy nie będą mieli dyżuru ze sobą,
- lekarz, który ma w danym terminie ma ustawioną preferencję czasową „koniecznie" będzie miał dyżur w tym terminie,
- lekarz, który ma w danym terminie ma ustawioną preferencję czasową „wykluczone" nie będzie miał dyżuru w tym terminie,

- anestezjolog nie będzie nigdy pełnił dyżuru na miejscu chirurga i na odwrót,
- wśród dwóch chirurgów na dyżurze przynajmniej jeden będzie miał prawo do pełnienia dyżuru samodzielnie.

Podejście takie można łatwo zastosować, gdyż otrzymywane z wywiadu z lekarzami dane są w większości spójne, lub też można względnie łatwo usunąć napotkane niespójności w trybie półautomatycznym. Należy też wspomnieć o stosunkowo niewielkim rozmiarze problemu, co ułatwia zadanie – problem jest prawdopodobnie NP-zupełny (wymaga to oczywiście formalnego dowodu, ale przyjmijmy to za roboczą hipotezę), więc możemy już na początku zapewnić osiągalność każdego planu.

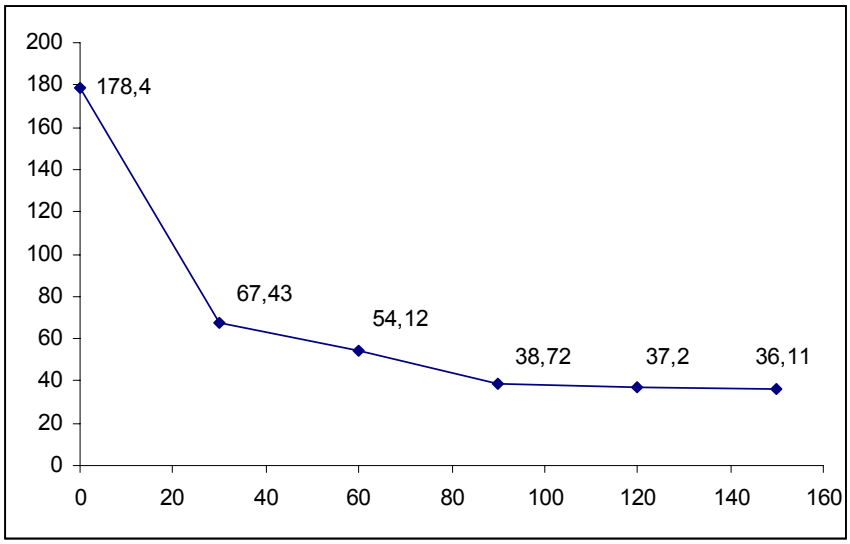
W każdej iteracji algorytmu poszczególne rozwiązania są oceniane i wybierane do następnego kroku zgodnie z zasadami działania algorytmów genetycznych. Następnie wybierane są osobniki do mutacji. Mutacja jest kompromisem pomiędzy wersją „lekką" a „ciężką" – mutuje jeden termin (z najgorszą wartością funkcji celu), ale wewnątrz terminu są mutowane tylko te elementy (lekarze), które tę wartość obniżają. Mutacja nie może pogorszyć wartości funkcji celu – dzięki temu w każdej iteracji algorytmu istnieje szansa na „naprawienie" jednego terminu. Dodatkowo, cały czas utrzymywana jest osiągalność planu, ponieważ elementy niezmienne (np. dyżury w terminie, na który dany lekarz ma ustawioną preferencję „koniecznie") nie podlegają mutacji. Na końcu wybierane są osobniki do krzyżowania – miejsce krzyżowania jest wybierane losowo i wybrane osobniki zamieniają się wszystkimi terminami od tego miejsca. Algorytm kończy się po określonej liczbie kroków.

4.4 Wyniki eksperymentów

Ze względu na brak ogólnie dostępnych danych testowych (benchmarkowych) dla tego specyficznego problemu oraz dużą trudność układania przykładowych zestawów danych, eksperymenty były prowadzone na danych rzeczywistych. Wynika z tego pewna niedogodność, gdyż co miesiąc powstaje tylko jeden zestaw danych. Jednak już eksperymenty z trzema zestawami pokazują pewne prawidłowości.

Podczas badań liczbę iteracji AG określono na 150. Podane wyniki są średnimi z trzech różnych zestawów danych, po pięć przebiegów dla każdego zestawu. Każdy zestaw zawierał dane dotyczące dwunastu chirurgów (w tym ośmiu samodzielnych) i trzech anestezjologów. Wielkość populacji ustalono na 200 osobników. Wykresy 1 i 2 pokazują wartości funkcji celu dla najlepszego planu w funkcji wykonanych iteracji algorytmu. W obu przypadkach prawdopodobieństwo krzyżowania ustawione było na 0.2 a prawdopodobieństwo mutacji na 1 (wykres 1) i 0.33 (wykres 2). Mutacja w każdym przebiegu algorytmu potencjalnie poprawia funkcję celu związaną z jednym terminem. Wyraźnie widać to na pierwszym wykresie – po trzydziestu iteracjach od rozpoczęcia wykonywania algorytmu wartość funkcji celu poprawia się niemal trzykrotnie. Przy trzykrotnie rzadszej mutacji taka poprawa zajmuje mniej więcej trzykrotnie więcej czasu. Należy jednak zauważyć, że jakość planów po 120 przebiegach jest praktycznie taka sama, więc z punktu widzenia efektywnego użycia

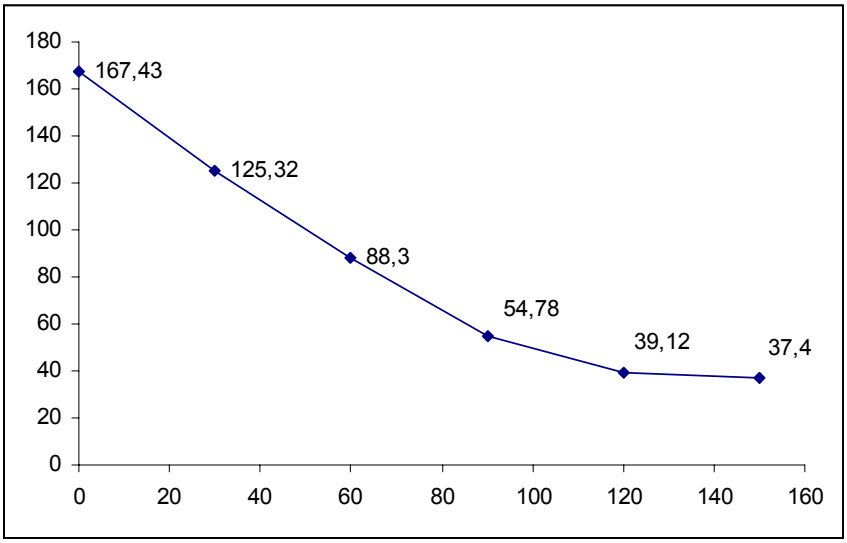
mocy obliczeniowej, korzystniejsze jest użycie mniejszego prawdopodobieństwa mutacji.



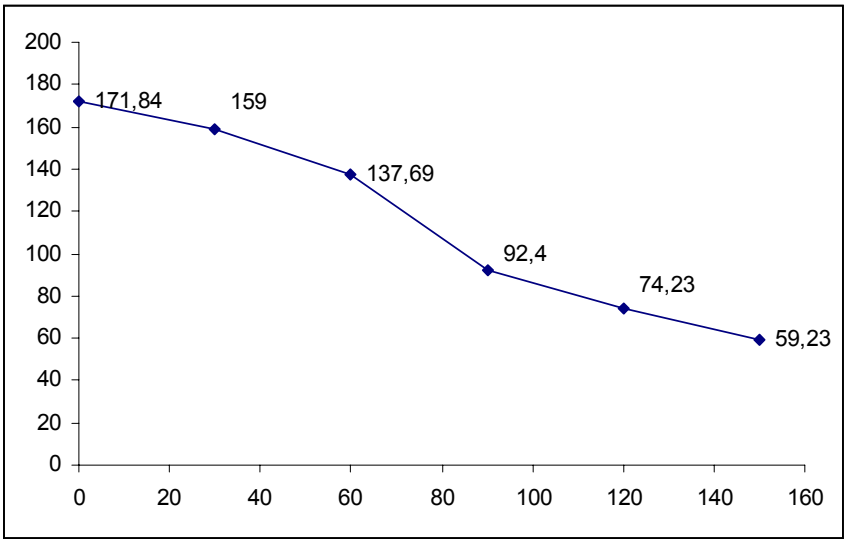
Wykres 1 Wartości funkcji celu dla najlepszego planu w funkcji wykonanych iteracji algorytmu; prawdopodobieństwo mutacji równe 1; prawdopodobieństwo krzyżowania równe 0,2

Wykres 3 pokazuje wartości funkcji celu dla prawdopodobieństwa krzyżowania równego 0,5 i prawdopodobieństwa mutacji równego 0,33. Krzyżowanie jest operacją o złożoności obliczeniowej (zwłaszcza pesymistycznej) dużo mniejszej, niż mutacja, jednak wprowadza do planu większe fluktuacje. Mutacja stara się poprawić wartość funkcji celu dla jednego terminu w kontekście innych terminów, gdy tymczasem krzyżowanie zamienia części planów miejscami, bez rozważania planu jako całości. Wyraźnie widać, że zbyt częste krzyżowanie niszczy powolną poprawę, dawaną przez mutację.

Na podstawie użytych danych zbudowano również plany metodą tradycyjną (ręcznie). Średnia wartość funkcji celu dla tych planów wynosiła 46,2, jednak uważane były przez użytkowników za lepsze od ułożonych automatycznie ze względu na lepszy (bardziej sprawiedliwy) rozdział dyżurów weekendowych między lekarzy.



Wykres 2. Wartości funkcji celu dla najlepszego planu w funkcji wykonanych iteracji algorytmu; prawdopodobieństwo mutacji: 0,33; prawdopodobieństwo krzyżowania: 0,2



Wykres 3. Wartości funkcji celu dla najlepszego planu w funkcji wykonanych iteracji algorytmu; prawdopodobieństwo mutacji: 0,33; prawdopodobieństwo krzyżowania: 0,5

4.5 Możliwe modyfikacje algorytmu

Algorytm wymaga pewnych modyfikacji. W pierwszej kolejności należy zmienić warunek zakończenia algorytmu tak, aby przerywał swoje działanie po określonej liczbie iteracji bez poprawy funkcji celu. Warta rozważenia jest likwidacja krzyżowania, a raczej zastąpienie go ciężką mutacją (mutującą np. wszystkie w kolejności terminy, w których przekroczona jest maksymalna liczba dyżurów lub maksymalna liczba dyżurów pod rząd). Konieczne jest też rozbudowanie algorytmu o możliwość układania dyżurów „pod telefonem" oraz rozszerzenie zakresu możliwych preferencji towarzyskich.

5. Podsumowanie

Z pewnością badania nad zastosowaniami metod poszukiwawczych i optymalizacyjnych będą kontynuowane. Wiele ze wspomnianych metod wymaga dokładniejszej analizy, aby możliwe było uzyskanie rezultatów jeszcze lepszych niż dotychczasowe. Szczególnie interesujące wydają się algorytmy genetyczne i ich warianty, ze względu na ich niezwykłą elastyczność, niewielkie skomplikowanie i umiarkowany koszt obliczeniowy. Przedstawione tutaj rozwiązanie prostego, rzeczywistego problemu za pomocą AG jest tego dobrym przykładem.

Literatura

Burke E. K., Elliman D. G. i Weare R. F. (1994). A University Timetabling System based on Graph Colouring and Constraint Manipulation. Journal of Research on Computing in Education Volume 27 Issue 1

Burke E.K., MacCarthy B., Petrovic S. i Qu R. (2000). Structured cases in case-based reasoning — re-using and adapting cases for time-tabling problems. Knowledge-Based Systems 13

Burke E. K., Newall J. P. i Weare R. F. (1998). A Simple Heuristically Guided Search for the Timetable Problem. Proceedings of the International ICSC Symposium on Engineering of Intelligent Systems. ICSC Academic Press.

Corne D. i Ross P. (1995). Peckish Initialisation Strategies for Evolutionary Timetabling. Proceedings of the First International Conference on the Theory and Practice of Automated Timetabling, Napier University, Edinburgh

Čangalović M., Kovačević-Vujčić V., Ivanović L. i Dražić M. (1998) Modeling and solving a real-life assignment problem at universities. European Journal of Operational Research 110

Colorni A., Dorigo M. i Maniezzo V. (1990a). Genetic Algorithms and Highly Constrained Problems: the Time-Table Case. Proceedings of the First International Workshop na Parallel Problem Solving from Nature, Lecture Notes in Computer Science 496

Colorni A., Dorigo M. i Maniezzo V. (1990a). Genetic Algorithms: a New Approach to the Time-Table Problem. Lecture Notes in Computer Science – NATO ASI Series, Vol. F 82, Combinatorial Optimalization

Colorni A., Dorigo M. i Maniezzo V. (1992). A Genetic Algorithm to Solve the Timetable Problem. Tech. rep. 90-060, Politecnico di Milano, Włochy. Dostępne na http://www.asap.cs.nott.ac.uk/ASAP/publications/

Dolan A. i Aldous J. (1993). Networks and Algorithms: an Introductory Approach. John Wiley & Sons Ltd.

Drexl A. i Salewski F. (1997). Distribution requirements and compactness constraints in school timetabling. European Journal of Operational Research 102

Hart E., Ross P. i Nelson J. (1998). Solving a Real-World Problem using a Heuristically Driven Evolving Schedule Builder. Journal of Evolutionary Computing, vol. 6, no.1

Hart E., Ross P. i Nelson J. (1999). Scheduling Chicken Catching - An Investigation into the Success of a GA on a Real World Scheduling Problem. Annals of Operations Research 92

Hilton A.J.W., Slivnik T. i Stirling D.S.G. (2001). Aspects of edge list-colourings. Discrete Mathematics 231

Hsiao-Lan Fang, Ross P. i Corne D. (1993). A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems. Proceedings of the Fifth International Conference on Genetic Algorithms, San Mateo: Morgan Kaufmann

Hsiao-Lan Fang, Ross P. i Corne D. (1994). A Promising Hybrid GA/Heuristic Approach for Open-Schop Scheduling Problems. 11th European Conference on Artificial Inteligence, John Wiley & Sons

Shie-Jue Lee i Chih-Hung Wu (1995). CLXPERT: A Rule-Based Scheduling System. Expert Systems With Applications, Vol. 9, No. 2

Marinagi C.C., Spyropoulos C.D., Papatheodorou C. i Kokkotos (2000). Continual planning and scheduling for managing patients test in hospital laboratories. Artificial Intelligence in Medicine 20

Mausser H.E. i Magazine M.J. (1996). Comparision of neural and heuristic methods for a time-tabling problem. European Journal of Operational Research 93

Michalewicz Z. (1999). Algorytmu genetyczne + struktury danych = programy ewolucyjne, Wydawnictwa Naukowo-Techniczne

Newall J. P. (1999). Hybrid Methods for Automated Timetabling, PhD Thesis, Department of Computer Science, University of Nottingham, UK

Oddi A. i Cesta A. (2000). Toward interactive scheduling system for managing medical resources. Artificial Intelligence in Medicine 20

Osowski S. (1994). Sieci neuronowe. Oficyna Wydawnicza Politechniki Warszawskiej

Ross P. i Corne D. (1995). Comparing Genetic Algorithms, Simulated Annealing, and Stochastic Hillclimbing on Timetabling Problems. Evolutionary Computing; AISB Workshop, Sheffield 1995, Selected Papers", wyd. T. Fogarty, Springer-Verlag Lecture Notes in Computer Science 993

Ross P., Corne D. i Hsiao-Lan Fang (1994). Successful Lecture Timetabling with Evolutionary Algorithms. Workshop Notes, ECAI'94 Workshop

Ross K. A. i Wright C. R. B. (1996). Matematyka dyskretna. Wydawnictwo Naukowe PWN

Schaerf A. (1995). A Survey of Automated Timetabling. Tech. rep. CS-R9567, CWI, Amsterdam, Holandia. Dostępne na http://www.cwi.nl/ftp/CWIreports/AP

Schaerf A. (1996). Tabu Search Techniques for Large School Timetabling Problems Tech. rep. CS-R9611, CWI, Amsterdam, Holandia. Dostępne na http://www.cwi.nl/ftp/CWIreports/AP

Spyropoulos C.D. (2000). AI planning and scheduling in the medical hospital enviroment. Artificial Intelligence in Medicine 20

Święcicka A., Seredyński F. i Jażdżyk M. (2001). Cellular Automata Approach to Scheduling Problem in Case of Modifications of a Program Graph. Proceedings of the International Symposium „Intelligent Information Systems 2001" June 18-22, 2001, Zakopane, Poland. Springer-Verlag

Tadeusiewicz R. (1993). Sieci neuronowe. Oficyna Wydawnicza Politechniki Warszawskiej

Thompson J. M. i Dowsland K. A. (1998). A Robust Simulated Annealing Based Examination Timetabling System. Computers Ops Research, Vol. 25, No. 7/8

Thulasiraman K. i Swamy M. N. S. (1992). Graphs: theory and algorithms. John Wiley & Sons

Valouxis C. i Housos E. (2000). Hybrid optimalization techniques for workshift and rest assignment of nursing personnel. Artificial Intelligence in Medicine 20

Weare R. F. (1995) Automated Examination Timetabling, PhD Thesis, Department of Computer Science, University of Nottingham, UK

de Werra D. (1996). Extensions of coloring models for scheduling purposes. European Journal of Operational Research 92

de Werra D. (1997a). The combinatorics of timetabling. European Journal of Operational Research 96

de Werra D. (1997b). Restricted coloring models for timetabling. Discrete Mathematics 165/166

de Werra D. (1999). On a multiconstrained model for chromatic scheduling. Discrete Applied Mathematics 94

de Werra D., Hoffman A.J., Mahadev N.V.R. i Peled U.N. (1996). Restrictions and preassignment in preempitve open shop scheduling. Discrete Applied Mathematics 68

de Werra D. i Mahadev N.V.R. (1997) Preassignment requirements in chromatic scheduling. Discrete Applied Mathematics 76

Wilson R. J. (2000). Wprowadzenie do teorii grafów. Wydawnictwo Naukowe PWN

Algorytmy genetyczne w szukaniu zależności funkcyjnych pomiędzy danymi liczbowymi

Ewa Szpunar-Huk
Wydziałowy Zakład Informatyki, Politechnika Wrocławska
E-mail: eszpunar@sun10.ci.pwr.wroc.pl

Streszczenie

Artykuł prezentuje hybrydowy algorytm, oparty o klasyczny algorytm genetyczny i o programowanie genetyczne, umożliwiający wyszukiwanie zależności pomiędzy danymi liczbowymi zawartymi w relacyjnych bazach danych. W artykule przedstawione są wyniki testów algorytmu wykonanych na danych sztucznie wygenerowanych, jego cechy, a także opisany jest przykład zastosowania go do danych rzeczywistych, dotyczących plam na Słońcu.

Wprowadzenie

Wraz z powstawaniem coraz to nowszych baz danych o wciąż zwiększających się rozmiarach rosną potrzeby analizy zgromadzonych informacji, ich opisu, klasyfikacji oraz znajdowania związków między nimi. Doprowadziło to do rozwoju dziedziny *Data Mining* , mającej na celu dostarczenie skutecznych mechanizmów pozyskiwania wiedzy z baz danych. W celu znalezienia zależności funkcyjnych pomiędzy wartościami liczbowymi atrybutów w relacyjnych bazach danych, klasycznie stosuje się metody regresji oparte o np. konkretyzację wzorców lub wykrywanie trendów w danych. Metody te są czasochłonne, nieskuteczne przy niedokładnych danych lub małej ich ilości i często nakładają duże ograniczenia na postać otrzymanego wzoru [7]. Dlatego konieczne jest szukanie nowych i bardziej uniwersalnych metod. Pewne możliości w tej dziedzinie stwarza stosowanie programowania genetycznego – odmiany algorytmów genetycznych – opisanej przez J. Kozę [10].

W niniejszym artykule opisany jest hybrydowy algorytm genetyczny, powstały z połączenia programowania genetycznego z klasycznym algorytmem genetycznym. Przedstawione są jego właściwości oraz przykład zastosowania do znajdowania zależności pomiędzy danymi liczbowymi. Testy przeprowadzone zostały na danych zarówno sztucznie wygenerowanych, jak i rzeczywistych – dotyczących stopnia aktywności słonecznej.

1. Programowanie genetyczne

Programowanie genetyczne to odmiana algorytmu genetycznego, w którym osobnik to zakodowany program, o zmieniającym się w czasie kształcie i rozmiarze [10],[14]. Program taki zbudowany jest z funkcji i operatorów należących do predefiniowanego zbioru **F** oraz symboli terminalnych z ustalonego zbioru **T**. Poprawne określenie tych

zbiorów jest o tyle istotne, że w przypadku niepełnego lub błędnego ich zdefiniowania niemożliwe może się okazać odnalezienie rozwiązania [10].

Podobnie jak w klasycznym algorytmie genetycznym, pierwszym krokiem w programowaniu genetycznym jest utworzenie populacji początkowej. Osobnika – czyli program – można zakodować w postaci drzewa, którego węzły to funkcje należące do zbioru **F**, natomiast liście to symbole terminalne ze zbioru **T**. Dla każdego węzła istnieje tyle poddrzew, ile argumentów wymaga funkcja znajdująca się w węźle. Istotną rolę odgrywa tu określenie maksymalnej głębokości tworzonych drzew. Przy zbyt płytkim drzewie może okazać się niemożliwe zakodowanie rozwiązania, zbyt głębokie powoduje nadmierny wzrost przestrzeni poszukiwań i utrudnia znalezienie rozwiązania [10],[14].

Funkcja celu w tym przypadku jest ściśle zależna od rezultatu wykonania programu, który dany osobnik koduje, na określonym zbiorze danych. Na podstawie wartości funkcji celu następuje selekcja, czyli określane jest, które osobniki i w jakiej liczbie przejdą do fazy krzyżowania [10],[14].

Krzyżowanie osobników w programowaniu genetycznym polega na losowym wyborze węzła – punktu krzyżowania u dwóch osobników, a następnie wymianie pomiędzy tymi osobnikami poddrzew, których korzeniem są wybrane punkty krzyżowania. Aby zapobiegać powstawaniu bardzo dużych drzew zakłada się maksymalną głębokość drzewa-potomka. Jeżeli głębokość potomka jest większa niż zakładana, to kopiuje się jednego z rodziców do nowej populacji [10],[14].

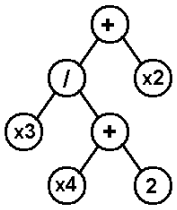
W algorytmie tym mogą występować dodatkowo inne operacje, takie jak mutacja – zmiana losowego poddrzewa na inne, permutacja – zamiana poddrzew wychodzących z jednego węzła, czy upraszczanie – niezmienianie znaczenia programu, ale jego uproszczenie [10],[14].

2. Algorytm hybrydowy do odnajdowania zależności funkcyjnych w danych liczbowych

Prezentowany algorytm zbudowany jest z dwóch algorytmów. Pierwszy z nich (zwany dalej PG) oparty jest o programowanie genetyczne, natomiast drugi bazuje na algorytmie genetycznym w klasycznym ujęciu (nazywany jest dalej AG). Hybrydowy algorytm ma zastosowanie w relacyjnych bazach danych składających się z n kolumn danych liczbowych z określonego zakresu i o określonej dokładności, oznaczonych $x_1, ..., x_n$. Jego zadaniem jest znalezienie postaci funkcji $x_1=f(x_2, ..., x_n)$, najlepiej odzwierciedlającej potencjalnie istniejącą zależność między kolumnami, przy założeniu, że szukana funkcja nie musi być funkcją wszystkich, lecz tylko wybranych kolumn, należących do bazy danych [12],[16].

2.1 Algorytm PG

Funkcję wielu zmiennych można zakodować w postaci drzewa, w którego węzłach znajdują się identyfikatory podstawowych operatorów arytmetycznych, natomiast liście to identyfikatory kolumn bądź stałych. Taką też postać ma osobnik w algorytmie PG. Zbiorem symboli nieterminalnych w tym przypadku jest zbiór **F** ={+,-,*,/}, natomiast zbiór symboli terminalnych składa się z dwóch podzbiorów: *W* i *I*, gdzie $I=\{x_2, ..., x_n\}$ to zbiór identyfikatorów kolumn natomiast *W* to zbiór współczynników od *0* do *100*. Przykład osobnika kodującego zależność $x_1=x_3/(x_4+2)+x_2$ przedstawiony jest na rysunku 1 [12],[16].



Rysunek 1. Przykład osobnika

Osobnik oceniany jest na dwa sposoby. Pierwszy z nich polega na tym, że dla każdej krotki w bazie danych obliczana jest wartość ze wzoru, jaki koduje dany osobnik, a następnie obliczana jest wartość bezwzględna różnicy otrzymanego wyniku i wartości w kolumnie wynikowej x_1 – czyli błąd wartości zwróconej przez osobnika, po czym błędy dla wszystkich krotek są sumowane. Funkcja ta jest wyrażona wzorem:

$$F = \sum_{i=1}^{N} \left| x_1^i - y^i \right| \quad \textbf{(1)}$$

gdzie:
N– liczba krotek w bazie danych
x_1^i – wartość w pierwszej kolumnie dla i-tej krotki
y^i – wartość dla i-tej krotki zwracana przez osobnika

Drugi sposób jest podobny, z tym że po obliczeniu błędu liczone jest, jaki procent wartości w kolumnie wynikowej stanowi błąd, a następnie wartości procentowe dla wszystkich krotek są sumowane i liczona jest ich wartość średnia:

$$F = \frac{1}{N} \sum_{i=1}^{N} \frac{100 * \left| x_1^i - y^i \right|}{x_1^i} \quad \textbf{(2)}$$

gdzie:
N– liczba krotek w bazie danych
x_1^i – wartość w pierwszej kolumnie dla i-tej krotki
y^i – wartość dla i-tej krotki zwracana przez osobnika

Podczas wyliczania wartości zwracanej przez danego osobnika dla danej krotki może się okazać, że występuje dzielenie przez zero, wtedy wartość funkcji oceniającej dla danego osobnika przyjmuje wartość –1, bez względu na to, jaki wynik otrzymano dla pozostałych krotek [12],[16].

Stosowaną metodą selekcji jest metoda elitarna z zachowaniem najlepszych osobników bez zmian. Natomiast mutacja polega na losowej zmianie, zgodnie z określonym, niewielkim prawdopodobieństwem, wartości w węzłach i liściach drzew, przy czym typ wartości pozostaje zawsze bez zmian, tzn. operator jest zastępowany operatorem, kolumna – kolumną, a współczynnik – współczynnikiem [12][16].

Opisany powyżej algorytm testowany był na danych sztucznie wygenerowanych, przy wykorzystaniu funkcji oceniającej liczonej ze wzoru (1). Okazało się, że algorytm ten pozwala na dokładne odnalezienie wielu zależności istniejących pomiędzy danymi w bazie. Ich przykłady, dla populacji równej 300 osobników i liczby kolumn w bazie równej liczbie kolumn występującej we wzorze, liczbie krotek w bazie równej 100, wraz z potrzebną do odnalezienia zależności liczbą pokoleń pokazane są w tabeli 1 [12],[16].

Tabela 1. Przykładowe odnalezione zależności

Szukany wzór zależności	liczba pokoleń
$x_1= x_2 \cdot x_2 \cdot x_2$	3
$x_1= x_4 + x_3 \cdot x_2 - x_5$	8
$x_1= x_2/(x_3 + x_4)$	16
$x_1= x_2 + x_2 \cdot x_2 - x_3/ x_2$	22
$x_1= x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9$	31
$x_1= x_2 + x_3/ x_4 - x_4/ x_2$	45
$x_1= x_2 \cdot x_3/(x_4 + x_5) - x_5$	103
$x_1= x_2 + x_3/ x_4 + x_4 \cdot x_5 - (x_2 + x_4)/ x_3$	135
$x_1=10 \cdot x_2+5 \cdot x_3$	137
$x_1= x_2 + x_3 + x_4 \cdot x_4 - x_2 \cdot x_2 \cdot x_2/(x_3 \cdot x_3)$	207
$x_1= x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_2 \cdot x_3 \cdot x_4$	375
$x_1= x_2 \cdot x_2 \cdot x_2 + x_3 \cdot x_3 \cdot x_3 + x_4 \cdot x_4 \cdot x_4 + x_5 \cdot x_5 \cdot x_5$	579

2.2 Cechy algorytmu PG

Algorytm okazywał się skuteczny również w przypadku nadmiarowości kolumn, tzn. gdy liczba identyfikatorów kolumn występujących we wzorze, na podstawie którego generowane były dane testowe, była mniejsza niż zdefiniowano w bazie danych. Liczba pokoleń potrzebnych do znalezienia rozwiązania przy pewnej liczbie nadmiarowych kolumn jest większa niż w przypadku, gdy kolumn w bazie jest dokładnie tyle, co we wzorze. Przykład tej zależności dla wzoru $x_1=x_2+x_3-x_2 \cdot x_3$, 400 osobników w populacji i 100 krotek w bazie danych pokazany jest w tabeli 2 [12],[16].

Algorytm można stosować również wtedy, gdy dane w bazie są niedokładne (zaszumione). Przeprowadzone zostały testy, w których generowane dane według zadanego wzoru zmieniane były o pewną losową liczbę, nie większą niż 10% wartości. Porównywana była następnie suma błędów zawarta w bazie danych z wartością funkcji oceniającej dla najlepszego osobnika (liczonej wg wzoru 1), która

Tabela 2. Pomiary przy nadmiarowości kolumn

Liczba nadmiarowych kolumn	0	1	2	3	4	5	6
Liczba pokoleń dla pomiaru 1	9	16	15	17	45	34	107
Liczba pokoleń dla pomiaru 2	7	30	18	16	47	45	23
Liczba pokoleń dla pomiaru 3	8	11	16	25	20	20	51
Liczba pokoleń dla pomiaru 4	9	17	27	20	22	26	38
Liczba pokoleń dla pomiaru 5	6	17	27	11	33	37	32
Liczba pokoleń dla pomiaru 6	12	15	18	31	15	64	26
Liczba pokoleń dla pomiaru 7	10	16	17	25	26	36	18
Wartość średnia	8,71	17,4	19,7	20,7	29,7	37,4	42,1
odchylenie standardowe	1,97	5,91	5,08	6,75	12,4	14,2	30,6

jest sumą błędu po wszystkich danych. Okazało się, że odnalezione zależności dają mniejszy błąd niż suma błędów w bazie, czyli odnalezione zostały wzory lepiej odzwierciedlające zależności w bazie niż wzór, na podstawie którego dane były generowane i następnie zaszumiane, np. dla wzoru $x_1=x_2 \cdot x_3 \cdot x_4$ suma błędów zawartych w bazie wynosiła 761861,39, natomiast błąd zwracany przez najlepszego osobnika, mającego postać $x_1=x_2(1/x_4-x_3)+x_2 \cdot x_3 \cdot x_4-8 \cdot x_3$, po 50 pokoleniach wynosił mniej, bo 715182,477 [12],[16].

Trudności natomiast pojawiały się, gdy we wzorze reprezentującym zależność pomiędzy kolumnami występowały współczynniki. Niemożliwe było odnalezienie niektórych nawet dość prostych zależności np. $x_1=15 \cdot x_2+4 \cdot x_3-23{,}4 \cdot x_2/x_3$ (dla 500 osobników w populacji). Okazało się, że częściowo problem ten można rozwiązać poprzez zwiększenie rozmiarów populacji do 5000, np. zależność: $x_1=100 \cdot x_2 \cdot x_3 \cdot x_4+17{,}5 \cdot x_4/x_3+ 2 \cdot x_2$, której nie udało się znaleźć przy 500 osobnikach w populacji, przy 5000 odnaleziona została w 457 pokoleń. Dla wielu zależności zwiększenie rozmiarów populacji było niewystarczające. Aby uczynić algorytm bardziej skutecznym, wprowadzono dodatkowy algorytm genetyczny (AG) [12],[16].

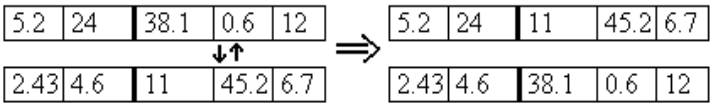
2.3 Algorytm dostrajający AG

Celem algorytmu opartego na klasycznym algorytmie genetycznym jest dobranie dla określonej parametrycznie liczby najlepszych osobników PG, co określoną liczbę pokoleń PG, jak najlepszych wartości współczynników, występujących w tych osobnikach, bez zmiany struktury drzewa. Ze względu na swój cel, algorytm ten nazwany został algorytmem dostrajającym.

Przebieg algorytmu dostrajającego współczynniki jest następujący: dla danego osobnika, przekazanego przez PG określa się liczbę współczynników w nim występujących. Jest to długość osobnika w AG. Osobnikiem w AG jest ciąg liczb z pewnego zakresu o określonej dokładności (liczbie miejsc po przecinku). Populacja początkowa zawiera ciąg utworzony na podstawie dostrajanego osobnika oraz określoną liczbę ciągów losowych. Jest ona nieduża, od 100 do 200 osobników.

Następnie tak utworzona populacja podlega cyklicznej selekcji, krzyżowaniu i mutacji przez ok. 70 pokoleń (wielkość populacji oraz liczba pokoleń określane są parametrycznie) [12],[16].

Osobniki oceniane są na podstawie tej samej funkcji oceniającej, jaka jest w PG. Liczona jest dla danego dostrajanego osobnika ze zmienianymi kolejno współczynnikami. Również analogiczny do PG jest wybór osobników do krzyżowania, ale samo krzyżowanie jest mniej skomplikowane. Dla pary osobników wybierany jest punkt krzyżowania, będący losowym miejscem w ciągu współczynników, po czym wymieniane są między osobnikami podciągi. Przykład krzyżowania dwóch osobników o długości 5 w AG przedstawiony jest na rysunku 2.



Rysunek 2. Przykład krzyżowania osobników w algorytmie dostrajającym AG

W przeciwieństwie do algorytmu PG, większą role odgrywa tu mutacja. Zaproponowana bowiem została specyficzna postać mutacji, której celem jest umożliwienie dobrego dostrojenia również współczynników z pewną liczbą miejsc po przecinku. Polega ona na zwiększeniu prawdopodobieństwa małych zmian wartości współczynników, tzn. jeżeli dana liczba ma podlegać mutacji, to dla każdej cyfry, z której jest zbudowana, określane jest prawdopodobieństwo mutacji pojedynczej cyfry, przy czym dla cyfr przed przecinkiem dziesiętnym, prawdopodobieństwo jest mniejsze niż dla cyfr po przecinku. Poza tym, na im dalszym miejscu po przecinku znajduje się cyfra, z tym większym prawdopodobieństwem będzie zmutowana [12],[16].

Po połączeniu algorytmów okazało się możliwe odnalezienie zależności, z którymi samo programowanie genetyczne nie radziło sobie , np. dla funkcji $x_1=\sin(x_2)$, dla $x_2 \in (0..180)$ wartość funkcji oceniającej liczonej wg wzoru (2), dla najlepszego osobnika po 250 pokoleniach, przy dostrajaniu co 20 pokoleń 20 osobników przez 40 pokoleń wynosiła 1,52. Gdy natomiast dostrajanie nie było stosowane, wartość funkcji oceniającej, po 2000 pokoleń, dla najlepszego osobnika była większa od 50. Poza tym dla zależności, do znalezienia których dostrajanie nie było konieczne, połączenie algorytmów przyspieszało ewolucję. Przykładem mogą być badania przeprowadzone dla danych wygenerowanych wg wzoru:

$$x_1=11,3/x_2+10\cdot x^2_3-5,2\cdot x_2/(x_3+x_5)+33,8+x^3_3\cdot x_5\cdot 26+ 20(x_4+x_2)/x_5+8x^3_5+ 12(x_3+x_5)/x_2+x^3_3.$$

Bez dostrajania po 165 pokoleniach wartość funkcji oceniającej (wg wzoru (2)) wynosiła 9,25, natomiast w ewolucji, startującej z tego samego pokolenia początkowego, w której dostrajanych było 15 osobników co 40 pokoleń PG przez 50

pokoleń AG**,** najlepszy osobnik już po 100 pokoleniach miał przystosowanie równe 1,2 [12],[16].

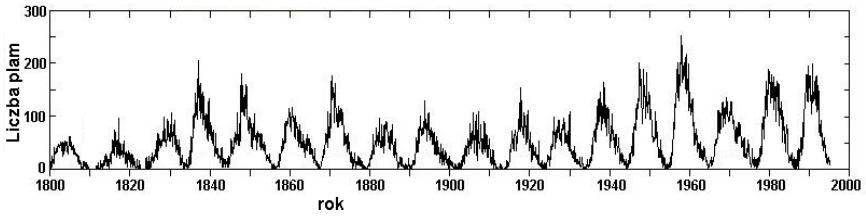
3. Testy na danych rzeczywistych

Dane rzeczywiste do badań generowane były na podstawie plików pochodzących z obserwatorium astronomicznego w Greenwich (Royal Greenwich Observatory), dotyczących liczby plam występujących na Słońcu w kolejnych dniach począwszy od 1874 roku. Algorytm testowany był zarówno pod kątem dokładności przybliżeń zależności pomiędzy sumami plam występujących na Słońcu w kolejnych okresach czasu, jak i możliwością zastosowania otrzymanych zależności do przewidywania liczby plam w kolejnych odcinkach czasu [16].

3.1 Plamy na Słońcu

Plamy na Słońcu są głównym przejawem i wyznacznikiem stopnia aktywności słonecznej, która ma duży wpływ na życie na Ziemi. Informacja o przyszłej liczbie plam najistotniejsza jest podczas określania orbit satelitów i przy planowaniu misji załogowych prowadzonych w przestrzeni kosmicznej, gdyż aktywność słoneczna wpływa na zmiany gęstości atmosfery ziemskiej wraz z wysokością, a poza tym wysokoenergetyczne cząstki wysyłane przez Słońce są groźne dla człowieka i aparatury elektronicznej. Promieniowanie słoneczne może wyrządzić wiele szkód również na Ziemi, gdyż może np. wytworzyć prąd w procesie indukcji w liniach energetycznych i w konsekwencji zaburzyć proces przesyłania prądu, a nawet wyrządzić szkody w sieci energetycznej, może także zakłócić pracę radarów i uniemożliwić pomiary geologiczne przy użyciu magnetografów oraz powoduje powstawanie zórz polarnych. Liczbę plam na Słońcu wiąże się także np. z klimatem czy wybuchami epidemii, ale są to powiązania na razie niepotwierdzone [11],[8], [13],[17].

Aktywność słoneczna ma charakter mniej więcej cykliczny, tzn. okresowo, co około 10,5 roku, na tarczy słonecznej najpierw nie ma plam, potem ich liczba wzrasta do pewnego poziomu, po czym znów spada do zera. Wykres miesięcznych sum plam w latach 1800-1997 przedstawiony jest na rysunku 3. Na wykresie tym można zauważyć charakterystyczne maksima i minima sumy liczby plam [11],[13].



Rysunek 3. Zmiana miesięcznej liczb plam w cyklu jedenastoletnim

Przewidywanie wielkości maksimum kolejnych cykli jest trudnym zadaniem, nad którym pracuje wielu naukowców, z czego największą grupą jest grupa 12-stu ekspertów z USA, Wielkiej Brytanii, Australii i Niemiec powołana przy wsparciu NASA przez Centrum Badań Przestrzeni Kosmicznej (Space Environment Center) [9]. Nie znana jest do tej pory zależność pomiędzy liczbą plam w poszczególnych odcinkach czasu i nie wiadomo w jakim stopniu jest to zjawisko losowe.

3.2 Sposób generowania bazy danych

Na rysunku 4. przedstawiony jest ogólny sposób generowania krotek wchodzących w skład bazy danych. Dane podzielone są na krotki uczące i testowe. Krotki uczące to krotki zawierające dane, wśród których hybrydowy algorytm genetyczny próbuje znaleźć zależność funkcyjną. Na krotkach testowych sprawdzane jest, na ile odnaleziony wzór sprawdza się dla nowych danych [16].

W celu wygenerowania krotek uczących najpierw wybierany jest okres czasu (t_0,t_1), nazywany dalej długością okresu uczącego. Potem ustalana jest liczba atrybutów w krotkach – n. Następnie określana jest długość odcinka czasu oznaczona na rysunku 4. jako d_1. Wartościami atrybutów od 2 do n są sumy plam występujących w odcinkach czasu tej długości. Liczba plam, będąca wartością dla atrybutu pierwszego, obliczana jest dla odcinków takiej samej bądź innej długości co d_1, oznaczonej d_2. Dodatkowo wyznaczana jest długość odcinka p (przesunięcie), która określa odstęp czasu pomiędzy okresem, dla którego generowana jest wartość atrybutu n, a okresem dla którego generowana jest wartość atrybutu pierwszego [16].

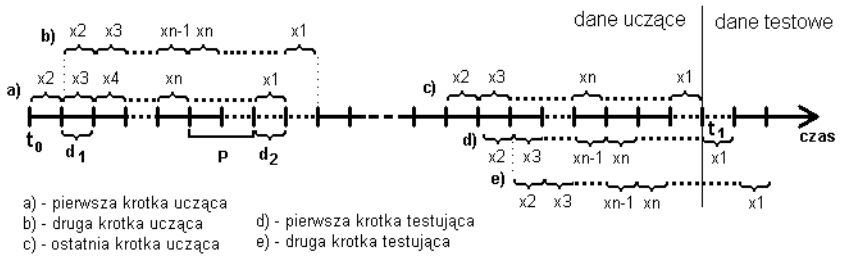
Zgodnie z powyższym, jeśli $s(t_p, t_k)$ oznacza liczbę plam w okresie (t_p,t_k), gdzie t_p to data początkowa, a t_k – data końcowa, to dla okresu uczącego (t_0,t_1) wartości atrybutów od 2 do n dla pierwszej krotki w bazie danych będą następujące:

$\{ x_2 = s(t_0,(t_0+d_1)), x_3 = s((t_0+d_1),(t_0+2d_1)),..., x_n = s((t_0+(n-2)d_1),(t_0+(n-1)d_1))\}$,

natomiast wartość w pierwszej kolumnie będzie wynosiła:

$x_1 = s((t_0+(n-1)d_1+p) , (t_0+(n-1)d_1+p+d_2))$.

Druga krotka ucząca będzie generowana analogicznie, przy czym t_0 przyjmie wartość t_0+d_1. Ostatnią (k-tą) krotką uczącą będzie krotka, dla której wartość pierwszego atrybutu liczona jest dla okresu (t_1, t_1+d_2). [16]



Rysunek 4. Generowanie krotek

Dla tak utworzonych krotek uczących, algorytm **PG** + **AG** szuka zależności $x_1=f(x_2,...,x_n)$ pomiędzy danymi zawartymi w bazie danych. Odpowiada to szukaniu zależności pomiędzy sumą plam w odcinku czasu długości d_2, a sumami plam w n kolejnych, poprzedzających dany okres odcinkach czasu długości d_1.

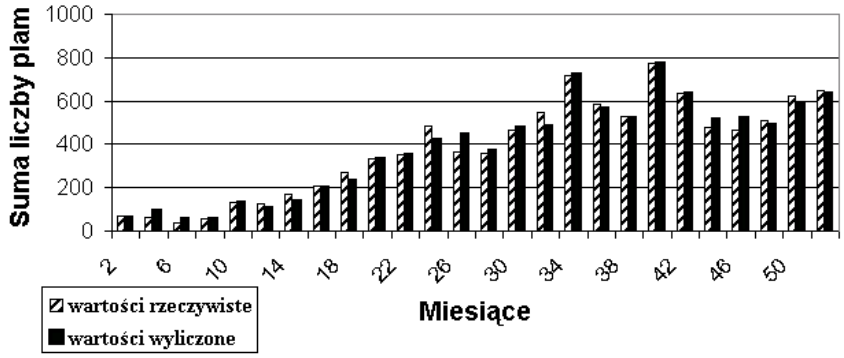
Sprawdzone zostaje, na ile dokładną zależność pomiędzy danymi występującymi w tak wygenerowanej bazie jest w stanie znaleźć proponowany algorytm, a następnie czy za pomocą uzyskanego wzoru można przewidywać liczbę plam w kolejnych okresach, tzn. w okresie (t_1, t_1+m*d_2), gdzie m=1,2,.. .W tym celu generowane są, analogicznie do krotek uczących, krotki testowe, co również przedstawia rysunek 4. Przy generowaniu krotek testowych parametry d_1, d_2, n oraz p pozostają bez zmian. Pierwszą krotką testową jest krotka, dla której wartość pierwszego atrybutu liczona jest dla okresu (t_1,(t_1+d_2)), drugą krotką ta, dla której wartość pierwszego atrybutu wynosi s (t_1+d_2,(t_1+2*d_2)), itd. [16].

3.3 Przeprowadzone testy

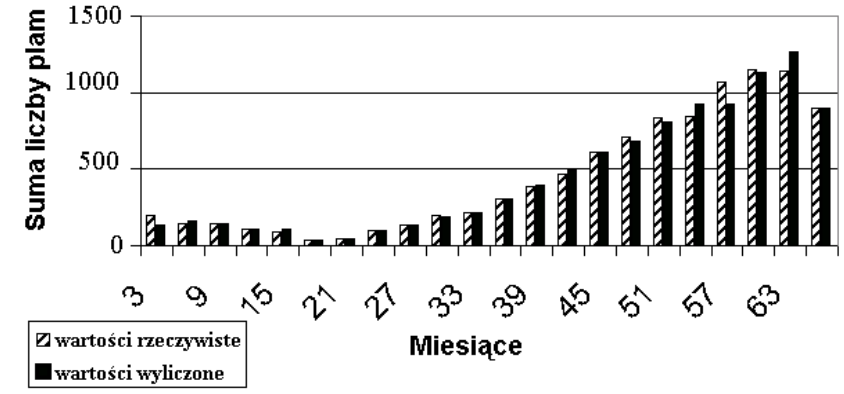
Przeprowadzone testy podzielone zostały na dwie zasadnicze grupy: testy krótkoterminowe i testy długookresowe. Testy krótkoterminowe to testy skuteczności przybliżania i przewidywania przy wartościach parametrów d_1 i d_2 zmieniających się od 30 do 120 dni, dla przesunięcia p od 0 do 180 dni oraz długości okresu uczącego nie większej niż 10 lat, w zależności od fazy cyklu jedenastoletniego. W testach długoterminowych dane uczące generowane były na podstawie okresu obejmującego kilka cykli jedenastoletnich, czyli więcej niż 10 lat. W eksperymentach tych długości okresów d_1, d_2 były większe niż przy testach krótkoterminowych (do 360 dni), a także większe było przesunięcie p (do 1800 dni). [16]

W testach krótkookresowych bardzo dobre okazało się przybliżenie danych uczących. Na rysunku 5. przedstawione jest porównanie wartości w kolumnie x_1, wyliczonej na podstawie odszukanego przez algorytm wzoru, z wartościami rzeczywistymi, przy dwóch różnych wartościach parametrów. Rysunek 5a) dotyczy okresu uczącego 1933-1939, wartości parametrów d_1 i d_2 równych 60 i parametru p równego 0. Ewolucja prowadzona była przez 900 pokoleń. Na rysunku 5b) przedstawione są wyniki dla okresu uczącego 1941-1949, wartości parametrów d_1 i d_2 równych 90 i parametru p równego również 0, po 900 pokoleniach. W testach długoterminowych przybliżenie było różne dla różnych okresów uczących, ale nadal zaskakująco dobre. Najlepsze przybliżenie uzyskano dla liczby kolumn w bazie równej 21 i parametrów d_1 oraz d_2, równych 180 dni, dla przesunięcia – równego 900 dni i okresu uczącego – lata 1914-1980. Funkcją oceniającą w tym przypadku była funkcja liczona wg wzoru (1), a dane zostały przybliżone ze średnim błędem procentowym (odpowiadającym przystosowaniu wyliczonemu przy pomocy drugiej funkcji) równym 16,9%. Przykład ten ilustruje rysunek 6. [16]

W testach dłogoterminowych uzyskanie dobrego przybliżenia danych było trudniejsze i wyniki zależały od wyboru okresu uczącego. Najlepsze przybliżenie danych uczących uzyskano dla liczby kolumn 21 i parametrów d_1 oraz d_2, równych 180 dni, dla przesunięcia równego 900 dni i okresu uczącego obejmującego lata 1914-



a) 1933-1939, 9 kolumn, $d_1 = d_2 = 60$, $p = 0$, 900 pokoleń

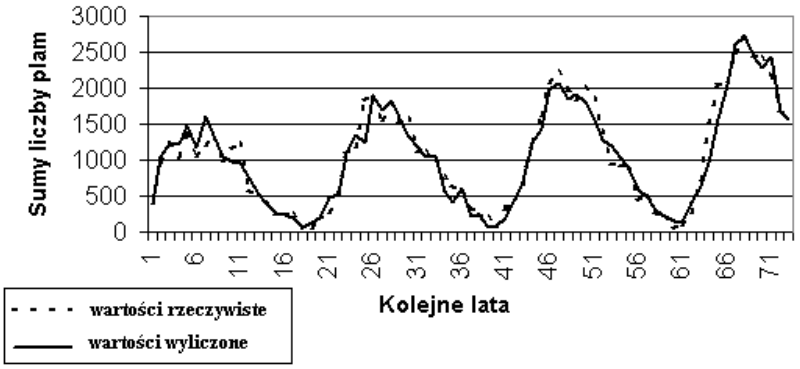


b) 1941-1947, 9 kolumn, $d_1 = d_2 = 90$, $p = 0$, 900 pokoleń

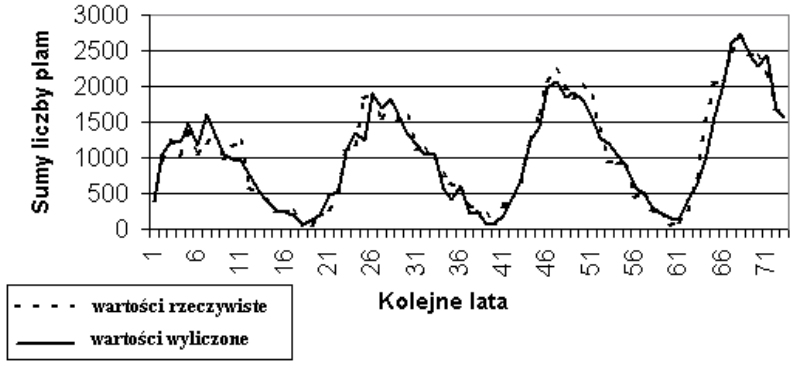
Rysunek 5. Przykład przybliżania wartości dla dwóch różnych okresów uczących

1980. Funkcją oceniającą w tym przypadku była funkcja liczona ze wzoru (1), a dane zostały przybliżone ze średnim błędem procentowym (odpowiadającym przystosowaniu wyliczonemu przy pomocy wzoru (2)) równym 16,9%. Uzyskane przybliżenie pokazane jest na rysunku 7. [16]

O ile uzyskane wzory, przybliżające dane uczące dobrze odzwierciedlały zależności pomiędzy danymi, o tyle nie nadawały się do przewidywania przyszłych wartości cyklu jedenastoletniego zarówno jeśli chodzi o przewidywanie z dużym, jak i krótkim wyprzedzeniem. W [16] przedstawione zostały szczegółowe wyniki testów i wskazane zostały możliwości rozbudowy algorytmu w celu poprawy jego przydatności nie tylko



Rysunek 6. Przykład przewidywania dla testu długookresowego



Rysunek 7. Przykładowe przybliżenie w teście długookresowym

do odnajdowania zależności, ale również do przewidywania. Najprawdopodobniej, dzięki możliwości zwiększenia liczby kolumn w bazie danych (do 30 lub 40) oraz wydłużenia okresu czasu, na podstawie którego generowane są krotki uczące, algorytm mógłby znaleźć taką zależność między danymi, która dobrze by przybliżała dane historyczne i jednocześnie mogła być stosowana do określania przyszłych wartości. W przypadku danych dotyczących plam na Słońcu nie jest jednak możliwe zwiększenie okresu uczącego ze względu na brak zapisów o prowadzonych regularnie obserwacjach przed drugą połową dziewiętnastego wieku [16].

4. Podsumowanie

Przeprowadzone eksperymenty pokazały dużą skuteczność przedstawionego algorytmu Data Mining w znajdowaniu zależności między danymi liczbowymi w bazach danych. Potwierdziły to badania przeprowadzone zarówno na danych wygenerowanych sztucznie, jak i rzeczywistych, dotyczących sum liczb plam na Słońcu. Algorytm znajduje istniejące zależności zarówno w bazach z dużą ilością informacji nadmiarowych, jak i przy danych niedokładnych – zaszumionych. Dzięki temu bardzo dobrze radzi sobie z przybliżaniem szeregów czasowych, reprezentujących przejawy aktywności słonecznej, mimo iż są one w dużej mierze zjawiskami chaotycznymi. Trafność uzyskiwanych rezultatów okazuje się niezależna tak od liczby krotek, jak i kolumn w bazie.

Zdolność odnajdowania zależności pomiędzy danymi może być poprawiona, poprzez zwiększenie zbioru wykorzystywanych przez algorytm funkcji (np. o funkcje trygonometryczne) i jednoczesne zwiększenie rozmiarów ewoluujących populacji. Podczas eksperymentów zauważono bowiem, że w przypadku uwzględnienia funkcji sinus jako potencjalnej operacji, uzyskiwane prognozy były bardziej trafne.

Niewątpliwie możliwe jest skuteczne zastosowanie proponowanej metody do innych problemów – wszędzie tam, gdzie wymagana jest analiza związków pomiędzy danymi liczbowymi, zgromadzonymi w bazach danych. Algorytm można także łatwo rozszerzać o dodatkowe funkcje i symbole terminalne, specyficzne dla określonego zagadnienia. Czyni to przedstawiony system elastycznym i skutecznym mechanizmem pozyskiwania wiedzy z baz danych.

Literatura

[7] Paweł Cichosz, *Systemy uczące się*, WNT, Warszawa 2000.

[8] David H. Hathaway, *Skylab*, dostępne w sieci: http:// science.msfc.nasa.gov\skylab.htm.

[9] J.A. Joselyn, J. Anderson, H. Coffey, K. Harvey, D. Hathaway,G. Heckman, E. Hildner, W. Mende, K. Schatten, R. Thompson, A.W.P. Thomson, and O.R. White, *Solar Cycle 23 Project: Summary of Panel Findings*, dostępne w sieci: http://www.sec.noaa.gov/info/ Cycle23.html.

[10] J.R. Koza (1996), *Future Work and Practical Applications of Genetic Programming*, Version 3, June 25, 1996, for "Handbook of Evolutionary Computation", dostępne w sieci: www-cs-faculty.stanford.edu/~koza.

[11] Rudolf Kippenhahn, *Na tropie tajemnic słońca*, Prószyński i S-ka, Warszawa 1997.

[12] H. Kwaśnicka, E. Szpunar, *Zastosowanie algorytmów genetycznych w pozyskiwaniu wiedzy z baz danych*, materiały konferencyjne, Pozyskiwanie Wiedzy z Baz Danych, Akademia Ekonomiczna, Wrocław, 2001, s.131-141.

[13] Ludwig Oster, *Astronomia współczesna*, Państwowe Wydawnictwo Naukowe, Warszawa 1978.

[14] Mitchell T.M., Machine learning, McGraw-Hill.

[15] D. Justin Schove, Sunspot cycles, *Hutchinson Ross Publishing Company*, Stroudsburg 1983.

[16] E. Szpunar, Zastosowanie metod Data Mining w przewidywaniu zmian stopnia aktywności słonecznej, praca magisterska, Politechnika Wrocławska, Wrocław 2001.
[17] Richard Thompson, *The Diverse Effect of Solar Events, Sydney*, Australia, 1995, dostępne w sieci: http://www.ips.gov.au/papers/richard/effects01.html.

Wykorzystanie algorytmów genetycznych do opracowywania rozkładów jazdy komunikacji miejskiej

Bogusław Molecki
Zakład Kolei Instytutu Inżynierii Lądowej, Politechnika Wrocławska
e-mail: molecki@alice.ci.pwr.wroc.pl

Streszczenie

W artykule omówiono możliwość zastosowania algorytmu genetycznego do poszukiwania optymalnej synchronizacji rozkładów jazdy transportu miejskiego. Przedstawiono szczegóły konstrukcji algorytmu i realizującego go oprogramowania. Omówiono otrzymane wyniki. Porównano możliwości wykorzystania algorytmu genetycznego i przeglądu zupełnego oraz podejścia hybrydowego do realizacji zadania.

1. Problem synchronizacji rozkładów jazdy

Obecnie w większości dużych ośrodków miejskich rozkłady jazdy komunikacji zbiorowej tworzone są w oparciu o zasadę jednego taktu podstawowego. Oznacza to, że wszystkie główne linie tramwajowe, trolejbusowe i autobusowe kursują w jednakowych odstępach czasu. W zależności od obciążenia na danej trasie różnicowana jest natomiast pojemność taboru [Wys 97].

Takie rozwiązanie ułatwia projektowanie rozkładów jazdy i zapewnianie przesiadek na węzłach dla pasażerów – analizie poddać można bowiem jedynie wycinek czasu długości jednego taktu (w pozostałym czasie sytuacja będzie analogiczna). Wszystkie przyjazdy i odjazdy tramwajów, czy autobusów określane są poprzez przesunięcie względem początku taktu.

Jak łatwo dostrzec, zadanie synchronizacji sprowadza się do znalezienia optymalnego zestawu odjazdów poszczególnych linii z punktów początkowych. Ocena rozwiązania polega zaś na sprawdzeniu synchronizacji odjazdów w wybranych węzłach sieci. I tak, jeżeli takt wynosi np. 15 minut (co oznacza, że wszystkie linie kursują z częstotliwością co 15 minut), a w danym punkcie pojawiają się pojazdy 3 linii, to ich odjazdy powinny następować w pięciominutowych odstępach. Oceniana jest zatem regularność kursowania komunikacji. Przy niskich wartościach taktu (do 20 minut) i obsłudze tras przez kilka linii (odstępy między pojazdami około 5 minut) traci bowiem znaczenie czas oczekiwania na przesiadkę (pasażer albo na przystanku początkowym oczekuje na połączenie bezpośrednie, albo przesiada się nawet kilkukrotnie w razie potrzeby, wykorzystując wielość linii między węzłami).

Trzeba przy tym zwrócić uwagę na fakt, iż poszczególne węzły sieci mają różną wagę – istotniejsze są np. długie odcinki na przedmieściach, na których znajduje się nawet kilkanaście przystanków obsługiwanych przez jeden zestaw linii, niż krótkie odcinki w centrum miasta. Waga danego punktu zmienia się również w trakcie dnia – rano istotniejszy jest kierunek do centrum, dzielnic przemysłowych, czy akademickich, po południu natomiast – centrów handlowych i dzielnic mieszkaniowych.
Rozwiązanie problemu synchronizacji rozkładu jazdy wymagać może jednak uwzględnienia dodatkowych ograniczeń:

- w ciągu dnia zmianie może ulegać wartość taktu (np. we Wrocławiu tramwaje w szczycie kursują co 12 minut, poza nim co 15 min., wieczorem – co 20 min.). W tym przypadku każda pora tworzy praktycznie odrębne zadanie synchronizacji, które może być rozwiązywane osobno.
- podobnie, w ciągu dnia zmianie mogą ulegać czasy przejazdu między węzłami (dzieje się tak głównie w ośrodkach, w których komunikacja zbiorowa nie ma wydzielonych pasów ruchu). Rozwiązanie jest analogiczne – doba dzielona jest na pory, w których obowiązują określone zestawy czasów – dla każdego z nich zadanie rozwiązywane jest oddzielnie.
- zmienione mogą być rozkłady jazdy tylko niektórych linii (często w przypadku remontów rekonstrukcji poddaje się jedynie rozkłady linii o zmienionej trasie). Rozwiązaniem jest sztywne uzależnienie odjazdu z pętli początkowej jednej linii od startu innej.
- niektóre linie wykorzystują na trasie odcinki jednotorowe, gdzie wyznaczone są stałe punkty mijania. Podobnie jak w poprzednim przypadku, rozwiązaniem jest sztywne powiązanie startów – w tym przypadku dwóch kierunków jazdy jednej linii.

2. Algorytm genetyczny

Zastosowanie algorytmu genetycznego [Gol 1995] do tak postawionego problemu nie nastręcza większych trudności. Opisujący osobnika genotyp powinien zawierać zestaw startów poszczególnych linii. Funkcja oceniająca natomiast – oceniać równomierność pojawiania się pojazdów komunikacji zbiorowej na wskazanych węzłach, z odpowiednimi wagami.
Po uwzględnieniu ograniczeń wymienionych w punkcie 1, przyjęto następującą postać genotypu:

- liczba genów będzie równa liczbie niezależnych startów linii – najczęściej jednej linii odpowiadać będą dwa geny (pierwszy dla kierunku *tam*, drugi – *z powrotem*), jednak za odjazdy linii o narzuconej z góry wzajemnej synchronizacji odpowiadać będzie tylko jeden gen, podobnie dla dwóch kierunków linii przebiegającej przez odcinki z wyznaczonymi sztywnie mijankami.
- jeden, wybrany gen powinien mieć zawsze wartość zero (istotne są nie same wartości startów linii, ale wzajemne przesunięcia między nimi).

- wykorzystując powyższy fakt, przyjęto że wartość zero będzie miał gen „zerowy” – będą z nim związane starty linii o rozkładach nie podlegających zmianom (narzuconej synchronizacji).

W trakcie działania algorytmu genetycznego, dane z genotypu przeliczane są na momenty pojawiania się linii na węzłach według wzoru:

$$t_{lw} = (g_{G(l)} + p_{1w}) \bmod T \quad \textbf{(1)}$$

gdzie:

t_{lw} – moment (w takcie) pojawienia się linii l na węźle w;

$g_{G(l)}$ – gen określający start grupy linii;

$G(l)$ – funkcja wskazująca, do której grupy linii zalicza się linia l;

p_{1w} – czas przejazdu linii l od punktu początkowego do węzła w;

T – długość taktu.

Następnie, przyjazdy wszystkich linii branych pod uwagę na danym węźle oceniane są pod względem regularności:

$$O_w = R(t_{l1w}, t_{l2w}, ..., t_{lnw}, T), \quad \textbf{(2)}$$

gdzie:

O_w – ocena synchronizacji na węźle *w*;

$R(t_1, t_2, ..., t_n, T)$ – funkcja oceniająca regularność wystąpień $t_1 \div t_n$ w takcie o długości *T*;

t_{liw} – moment (w takcie) pojawienia się linii l_i na węźle *w*;

n – liczba linii branych pod uwagę w ocenie synchronizacji na węźle.

Funkcja oceniająca regularność może być przy tym oparta np. na odchyleniu standardowym odstępów pomiędzy momentami pojawienia się linii na węźle.
Ogólna ocena rozwiązania sprowadza się do sumy ważonej wyników ocen cząstkowych:

$$O = \sum_{i=1}^{m} (w_{wi} \cdot O_{wi}), \quad \textbf{(3)}$$

gdzie:

O – ogólna ocena rozwiązania;

O_{wi} – ocena synchronizacji na węźle *wi*;

w_{wi} – waga synchronizacji na węźle *wi*;

m – liczba branych pod uwagę węzłów.

Jak widać, algorytm będzie mógł zdecydować się na poświęcenie kilku mniej istotnych węzłów sieci, za cenę dobrego rozwiązania na jednym z węzłów

podstawowych. Z tego też względu bardzo istotny jest rozważny dobór wag synchronizacji na poszczególnych węzłach.

W algorytmie założono wykorzystanie krzyżowania jednopunktowego. Sposób konstrukcji genotypu zapewnił, iż w procesie krzyżowania nie mogły powstać genotypy błędne. Dopuszczono również mutację – losowo wybrany gen mógł zmienić wartość o maksymalnie ±12,5%.

3. Realizacja Programowa

Podczas implementacji algorytmu oparto się na fikcyjnym takcie 256, co pozwoliło na wykorzystanie sprzętowej arytmetyki bajtowej (modulo 256). W związku z tym, przed rozpoczęciem obliczeń, wszystkie wartości czasów przejazdu były przeskalowywane (przemnożenie przez wartość 256/T).

Każdy gen mógł być więc przechowywany w jednym bajcie, co wpłynęło pozytywnie na szybkość obliczeń. Zgodnie z założeniami omówionymi w punkcie drugim, w każdym pokoleniu ewolucji genotyp był normalizowany (po operacji krzyżowania czy mutacji wartości wszystkich genów zmniejszane były o wartość genu zerowego).

Funkcję oceniającą oparto na sumowaniu odchyleń odstępów między momentami pojawienia się kolejnych linii na węźle od wartości optymalnej (tak np. dla T=12 min. i 4 linii optymalny odstęp wynosi 3 minuty).

W trakcie badań testom poddano samo zachowanie się algorytmu genetycznego. Na ich podstawie przyjęto, że:

- rozmiar populacji wynosić będzie 1024 osobniki;
- populacja początkowa będzie tworzona losowo;
- przez pierwsze 30 pokoleń nie będzie wykorzystywana mutacja;
- przez kolejne 30 pokoleń prawdopodobieństwo mutacji będzie wynosić 1%;
- począwszy od 61 pokolenia prawdopodobieństwo mutacji będzie 10-procentowe;
- wykorzystywana będzie elitarna metoda selekcji – w każdym pokoleniu gorsza połowa populacji zostanie zastąpiona krzyżówkami organizmów lepszych;
- moment zakończenia ewolucji będzie dobierany w zależności od otrzymywanych w danym przypadku wyników.

4. Badania na rzeczywistych sieciach tramwajowych

Powstałe oprogramowanie posłużyło do testów na danych pochodzących z dwóch dużych polskich ośrodków miejskich – Wrocławia i Poznania. W obu przypadkach „kręgosłup” sieci komunikacyjnej stanowią linie tramwajowe. W momencie wykonywania badań we Wrocławiu kursowały 23 linie (co 12, 15 lub 20 minut – zależnie od pory dnia), w Poznaniu 15 linii (co 10, 15 lub 20 minut – zależnie od rodzaju dnia).

Pierwsze badania wykonano dla Wrocławia, pory III dnia roboczego (od godziny 12.00 do 17.00). Linie tramwajowe kursują wówczas co 12 minut. W wyborze pory kierowano się opinią pracowników MPK Wrocław, zajmujących się ręcznym

opracowywaniem synchronizacji – właśnie ta pora została określona jako najtrudniejsza.

W chromosomie znalazło się 45 genów (23 linie tramwajowe, z których jedna (nr 17, relacji Klecina – Sępolno) miała narzucone mijanki). Na funkcję oceniającą składały się 44 oceny cząstkowe (26 związanych z węzłami podstawowymi i 18 – uzupełniającymi). Węzły podstawowe (waga 2) zostały wybrane na liniach dojazdowych do centrum, uzupełniające (waga 1) na krótszych odcinkach, w obrębie centrum.

Uzyskane wyniki oceniono wysoko – na 13 węzłów podstawowych, tylko w jednym (Pilczyce) równomierność odjazdów nie była dobra. W przypadku węzłów uzupełniających, regularność była dobra, lub zadowalająca.
Badanie ponowiono, podnosząc wagę węzła Pilczyce do 4, bez zmian pozostałych parametrów. Otrzymano praktycznie optymalny rozkład odjazdów na węźle Pilczyce, z nieznacznym pogorszeniem synchronizacji na niektórych węzłach uzupełniających.

Po zakończeniu podstawowej fazy badań, wykonano szereg optymalizacji dla celów kontrolnych i porównawczych. Sprawdzono m.in. działanie algorytmu dla innych pór czasowych Wrocławia i sieci tramwajowej Poznania. Uzyskane wyniki były dobre lub zadowalające (szczegółowe omówienie znajduje się w [Kwa 99]).

5. Porównanie metod synchronizacji rozkładów jazdy

W trakcie przeprowadzania testów działania algorytmu genetycznego, nawiązana została współpraca z działem rozkładów jazdy Miejskiego Przedsiębiorstwa Komunikacyjnego we Wrocławiu. W efekcie prac powstało nowe oprogramowanie komputerowe, wykorzystujące przegląd zupełny. Rozwiązanie takie było możliwe, ze względu na bardzo ostre warunki stawiane przez koordynatorów rozkładów jazdy w MPK Wrocław. W rezultacie przestrzeń poszukiwań rzędu 10^{40} zawężana jest maksymalnie do 10^6 przypadków, przy czym dla parzystych wartości taktu (12 i 20 minut) nie przekracza zazwyczaj nawet 30.000 kombinacji.

Nowe podejście oparte jest o operacje macierzowe, zbliżone do rozwiązywania układu równań z wieloma niewiadomymi. Ograniczenia narzucane synchronizacji są bowiem bardzo ostre – nie tylko zawierają dopuszczalne odstępy między pojawieniami się linii na węźle, ale nawet określają kolejność ich wystąpienia. O trudności prowadzenia takiego zadania optymalizacji świadczyć może fakt, iż w około połowie przypadków wykazywana jest niemożliwość realizacji pełnego zestawu warunków.

Na podstawie przeprowadzonych badań i kilkuletniego okresu współpracy z użytkownikiem rzeczywistego systemu, należy stwierdzić, iż zarówno algorytmy genetyczne, jak i przegląd zupełny mogą być z powodzeniem stosowane w poszukiwaniu synchronizacji rozkładów jazdy miejskiej komunikacji zbiorowej.
Największe pole zastosowań algorytmów genetycznych mieści się przy tym w przypadku sieci o niezbyt dobrze rozpoznanej specyfice – gdzie warunki stawiane

synchronizacji sprowadzają się do wskazania punktów, w których ma zostać dokonana optymalizacja. Ze wzrostem skomplikowania narzucanych warunków maleje rozmiar przestrzeni poszukiwań – możliwe więc staje się stosowanie technik opartych na przeglądzie zupełnym. Należy również wspomnieć o podejściu hybrydowym – możliwości wykonania synchronizacji kilku linii w oparciu o przegląd zupełny, a następnie wykorzystania wyników jako danych wejściowych algorytmu genetycznego (optymalizacja całości systemu).

Literatura

[Gol 95] **Goldberg D.E.**: Algorytmy genetyczne i ich zastosowania, Wydawnictwa Naukowo Techniczne, Warszawa 1995.

[Kwa 99] **Kwaśnicka H.**, **Molecki B.**: Timetabling of the city tram service using a genetic algorithm, w: „Inteligent Information Systems VIII", proceedings of the Workshop, Instytut Podstaw Informatyki PAN, Ustroń 1999.

[Wys 97] **Wyszomirski O. (red.)**: Komunikacja miejska w gospodarce rynkowej, Wydawnictwo Uniwersytetu Gdańskiego, Gdańsk 1997.

Muzyka komponowana przez komputery? System MGen

Marcin Bober
Wydziałowy Zakład Informatyki, Politechnika Wrocławska
E-mail: mbober@sun10.ci.pwr.wroc.pl.

Streszczenie

Artykuł podejmuje zagadnienie generowania muzyki przez program komputerowy wykorzystujący algorytm genetyczny jako mechanizm generujący. Autor artykułu musi w pewnym stopniu zdefiniować pojęcie muzyki, oceniać w obiektywny sposób to, co nam wszystkim wydaje się jak najbardziej subiektywne. Implementacja algorytmu genetycznego prezentowanego w artykule nie różni się niczym na poziomie ideowym od powszechnie przyjętej formy: oceń pokolenie, wybierz najlepszych, wykorzystaj ich do stworzenia nowego pokolenia, jednak tematyka artykułu jest już w dużym stopniu niestandardowa. Do oceny potrzebne są kryteria – jeśli chodzi o ocenę muzyki, musimy przyjąć duże założenia upraszczające, aby takie kryteria zdefiniować.

Wstęp

Ludzkość zawsze próbowała ułatwić sobie życie zaprzęgając do pracy maszyny. Początkowo do najprostszych czynności, potem do coraz bardziej złożonych. Na pewnym etapie tych ułatwień ktoś wpadł na pomysł, aby udostępnić dla komputerów – czyli maszyn obliczeniowych – dziedziny do tej pory zarezerwowane tylko dla człowieka. Sztuka, bo o niej mowa, z natury swojej rzeczy jest nieobliczalna, oddaje wnętrze człowieka, jego spontaniczność, intuicję, geniusz, specyficzną wiedzę opartą na niematematycznych regułach. Jak z tym wszystkim pogodzić imperatywny charakter tego co dzieje się wewnątrz maszyny? Dla wielu – absurd. Podjęte zostały już jednak pewne nieśmiałe kroki, aby nauczyć komputer „kultury", bo to właśnie jej przejawem jest sztuka. Bardzo często wykorzystuje się do tego algorytmy genetyczne (GA) lub inne metody wzorujące się na naturze. Do tej pory wyniki badań nie są najbardziej zachęcające, nie oznacza to jednak, że kierunek badań jest zły. Wszystko zależy, jak to zwykle w przypadku GA, od pomysłowości w dobieraniu kształtu genotypu oraz operatorów genetycznych. W większości przypadków nie znamy i nie chcemy z góry znać przebiegu działania GA. Interesuje nas jedynie efekt – czyli zoptymalizowane rozwiązanie.

Naprawdę trudnym zadaniem jest optymalizowanie muzyki. W tym wypadku efektem działania GA powinno być oczywiście wygenerowanie muzyki takiej, do jakiej przyzwyczajone jest ucho ludzkie, tzn. muzyki, w której panuje pewien porządek, jest ona złożeniem porządku w rytmice oraz harmonii. Pomijam tu zupełnie kwestię aranżacji, ekspresji czy artykulacji, które też są nieodłączną częścią muzyki,

ale większy wpływ na nie ma jednak wykonawca a nie kompozytor. Muzyka jest akceptowalna wtedy, gdy podoba się, z tego czy innego powodu, słuchaczowi. Owo podobanie się zależy tylko od jego indywidualnych preferencji – gustu muzycznego, od nastroju chwili, kontekstu w jakim się znajduje i na pewno również od pogody za oknem. Ponieważ uwzględnienie wszystkich tych czynników jest rzeczą niemożliwą, są to czynniki w większości niemierzalne, musimy podjąć pewne założenia upraszczające. W próbie generowania muzyki, pierwszym poczynionym założeniem mogłoby być ograniczenie się do gustu jednej i tylko jednej osoby. Jednak i wtedy pozostaje do uwzględnienia wiele nie dających się zdefiniować w ścisły sposób pojęć. Ograniczmy się więc do osiągnięcia następującego efektu: muzyka pochodząca z pierwszych pokoleń (nuty generowane są niemalże losowo) musi być odróżnialna od muzyki pochodzącej z pokoleń późniejszych - oczywiście ta druga powinna być „muzycznie" doskonalsza.

1. Wprowadzenie w tematykę, czyli jak robią to inni

Z pomysłami innych komputerowych kompozytorów najłatwiej zapoznać się w Internecie. Problem generowania muzyki przez maszyny okazuje się bardzo popularnym tematem w sieci. Bez żadnych trudności można przyjrzeć się pracom i wynikom prac innych autorów. Najbardziej interesujące wydają się być prace wykorzystujące algorytmy genetyczne. Autorzy prac przedstawiają struktury chromosomów, użyte operatory genetyczne, specyfikację swoich systemów. Dla przykładu i rozjaśnienia tematyki przedstawiony zostanie system zaimplementowany przez Bruce Jacob'a z uniwersytetu w Michigan.
Bruce Jacob[4] we wstępie do swojego artykułu „Komponowanie z algorytmami genetycznymi" próbuje zdefiniować sam proces powstawania muzyki, czyli twórczość. Według niego istnieją dwa, nie mające z sobą nic wspólnego, sposoby na komponowanie muzyki: błysk (flash) czyli geniusz, impresja, komponowanie niejako bez uwzględniania żadnych reguł, sztuka. Drugi sposób to proces iteracyjny, polegający na szukaniu najlepszego rozwiązania dzięki ciężkiej i wytrwałej pracy – w tym Bruce Jacob widzi szansę dla algorytmu genetycznego. Musimy polegać na ciężkiej pracy (hard work) z tego powodu, że nie tylko nie potrafimy modelować muzyki, ale nie potrafimy jej w żaden sposób zrozumieć. Ciężka praca Jacob'a polegała na wykorzystaniu istniejących już wzorców (motywów) muzycznych do generowania wariacji na ich temat. Ogólny zamysł jego projektu przedstawia się więc w następujący sposób:

1. zdefiniuj zbiór głównych motywów, które wykorzystasz w procesie kompozycji,
2. komponuj frazy układając motywy w pewne sekwencje,
3. twórz nowe motywy wybierając spośród motywów głównych i tych już wykorzystanych we frazach, produkując wariacje na ich temat,
4. połącz frazy w większe części.

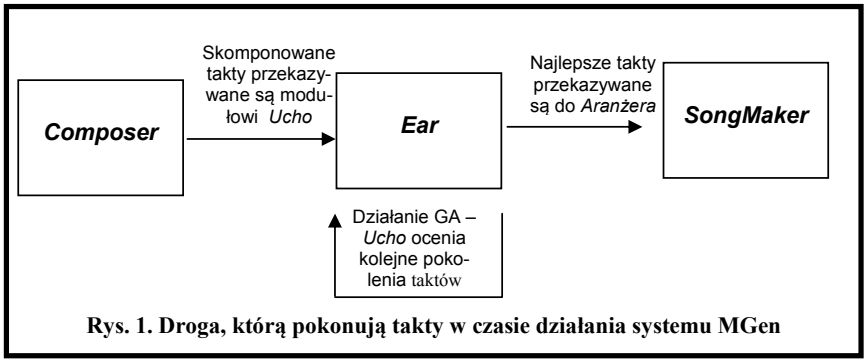
Aby ograniczyć domenę poszukiwań Jacob nie pracuje na poziomie pojedynczych nut, a wykorzystuje motywy, czyli struktury wyższego poziomu – mogą być nimi np.

takty. Cały system wykorzystuje i komponuje więc małe motywy, następnie aranżuje je w większe frazy. Wtedy kompozycja staje się o wiele prostszym procesem. W systemie Jacoba występują moduły: kompozytor (composer), ucho (ear) i aranżer (arranger). Użytkownik systemu definiuje zbiór głównych motywów. Kompozytor dokonuje wariacji na ich temat i używa ich do produkcji większych fraz. W momencie, gdy motyw dodawany jest do frazy, do akcji wkracza ucho. Jeśli ucho stwierdzi, że motyw nie jest odpowiednim składnikiem harmonicznym tej frazy, motyw jest usuwany. Gdy istnieje już odpowiednia liczba fraz do wykorzystania, aranżer produkuje i ocenia porządki fraz, które są następnie wykorzystywane do produkcji nowych i lepszych porządków. Algorytmy genetyczne są użyte w każdym z podanych tu komponentów. Parametry tych komponentów zapisane są w chromosomach, które to ewoluują właśnie dzięki algorytmom genetycznym.

2. Opis systemu MGen

System do generowania muzyki MGen napisany został w języku programowania Java i składa się, podobnie jak system B. Jacob'a, z trzech głównych części (rysunek 1):

- *Kompozytor* (moduł *Composer*)
- *Ucho* (moduł *Ear*)
- *Aranżer* (moduł *SongMaker*)



Rys. 1. Droga, którą pokonują takty w czasie działania systemu MGen

Moduły te są odpowiedzialne za produkowanie materiału muzycznego. Sama muzyka przechowywana jest w specjalnie stworzonej strukturze danych. Struktura ta jest czymś, co w terminologii algorytmów genetycznych nazywamy po prostu pokoleniem (*Generation*). Jest ona zbiorem taktów, takt to pojedynczy osobnik pokolenia, reprezentujący najmniejsze rozwiązanie problemu generowania muzyki. Wszystkie moduły działają w sposób od siebie niezależny (w innym czasie), każdy polega na wynikach swego poprzednika w podanej wyżej kolejności (najpierw pracuje *Kompozytor*, potem *Ucho*, którego wyniki pracy wykorzystuje *Aranżer* – rysunek 1).

Algorytm genetyczny wykorzystywany jest jedynie w module Ucho. *Ucho* ocenia występujące w pokoleniu takty, a ocena ta jest podstawą poprawnego działania algorytmu genetycznego.

2.1 Opis modułów oraz zastosowanego algorytmu genetycznego

- *Moduł Kompozytor*

Moduł ten jest odpowiedzialny za przygotowanie początkowej postaci pokolenia zawierającego wybraną przez użytkownika liczbę taktów. Komponowanie polega na losowym doborze nut (ich wartości rytmicznych jak i wysokości dźwięku), tak aby komponowały się one w pojedynczy takt w metrum 4/4. Na moduł *Kompozytor* nałożone są pewne ograniczenia, aby komponowane takty nie przyjmowały zanadto chaotycznej postaci. Komponowanie przeprowadzane jest z rozdzielczością do 16 części nuty z uwzględnieniem triol ósemkowych i czwórkowych (łącznie 7 rodzajów nut ze względu na rytmikę). Wysokość każdej nuty dobiera się losowo na przestrzeni trzech kolejnych skal chromatycznych (3 * 12 = 36 różnych dźwięków). Takty przyjmują uporządkowaną strukturę rytmiczną dzięki zastosowaniu następującego mechanizmu: **w danej chwili prawdopodobieństwo wylosowania ostatnio wylosowanej wartości rytmicznej jest równe ½, z założeniem, że początkowo wylosowanie ćwierćnuty jest równe ½.** Efektem tego jest powtarzanie się w taktach jednej po drugiej tych samych wartości rytmicznych, tak jak to się często dzieje w kompozycjach dokonanych przez człowieka (autor nie przeprowadzał żadnych badań statystycznych materiału nutowego). Pokolenie stworzone przy pomocy *Kompozytora* przekazywane jest do pracy z algorytmem genetycznym.

- *Moduł Ucho*

Ucho to zasadnicza część systemu MGen. Dokonuje oceny wszystkich taktów z pokolenia, aby algorytm genetyczny mógł prawidłowo działać z wykorzystaniem tych ocen jako wartości funkcji celu. Dlatego na poziomie opisu ucha dokonany zostanie opis zastosowanego algorytmu genetycznego.

Czym jest ocena taktu?

Ocena rozbita została na trzy części odpowiadające trzem punktom widzenia, według których oceniany jest pojedynczy takt:

1. **Takt oceniany jest pod względem przynależności jego nut do skali, w której chcemy komponować**.

W przyjętym w muzyce klasycznej (i każdej późniejszej, oprócz egzotyki typu muzyka orientalna) systemie tonalnym istnieje 12 różnych dźwięków: C Cis D Dis E F Fis G Gis A B H. W skład skal muzycznych wchodzi najczęściej tylko 7 spośród tych dźwięków. Istnieje wiele skal, przykładem skali jońskiej jest gama C dur, w skład której wchodzą dźwięki: C D E F G A H. Aby komponować w gamie C dur możemy wykorzystywać jedynie te dźwięki spośród wszystkich 12 na przestrzeni dowolnej liczby oktaw. Są to założenia trochę upraszczające, ale w gruncie rzeczy tak właśnie jest. Ocena skali jest więc odsetkiem należących do danej skali nut w danym takcie.

Każdy takt może otrzymać maksymalnie 100 punktów. Ucho ocenia takty według aktualnej skali wybranej przez użytkownika, np. skala D dorycka (dorian D).

2. **Ocena pod względem współbrzmienia ze sobą nut z każdego taktu**.

Problem ten rozwiązano bez sformalizowania zasad rządzących kompozycją harmonii utworu. Chociaż jest to w pewnym stopniu możliwe (na Akademiach Muzycznych wykłada się przedmioty uczące właściwej kompozycji), autor uznał, że jest to obszar wiedzy za mało przez niego poznany i gdyby miał w tej materii wyciągać wnioski, nie byłyby one do końca słuszne i prawdziwe. Na przykład, wcale nie jest prawdą, że najczęściej występują w bezpośrednim sąsiedztwie ze sobą nuty różniące się tylko o jeden stopień skali (bo gdzie mieściłaby się ekspresja budowana przez chociażby zaskoczenie?). Muzyka, jak każdy inny rodzaj sztuki może wyrażać ludzkie uczucia, a czy człowiek zawsze jest tylko w trochę innym stanie emocjonalnym niż był przed chwilą? Dlatego MGen proponuje inne rozwiązanie. Tą propozycją jest stworzenie przez użytkownika systemu banku brzmień. Bank ten byłby zbiorem taktów wyrażających indywidualne preferencje słuchacza i tym samym stanowiłby niepełną funkcję oceniającą wszystkie możliwe takty. Im większy jest rozmiar banku, tym dokładniejsza jest ocena muzyki. Gdyby bank zwierał wszystkie możliwe takty z ich ocenami, byłby kompletną funkcją oceniającą muzykę, uwzględniającą preferencje muzyczne słuchacza. Jest to niewykonalne chociażby dlatego, że liczba wszystkich możliwych taktów w systemie MGen jest z równa:

$$C = A^B,$$ gdzie:

C – liczba różnych taktów, jest to liczba permutacji 48 elementowych z powtórzeniami ze zbioru 37 elementowego,

A – liczba różnych możliwych nut, A = 3 * 12 + jedna nuta pusta = 37 nut,

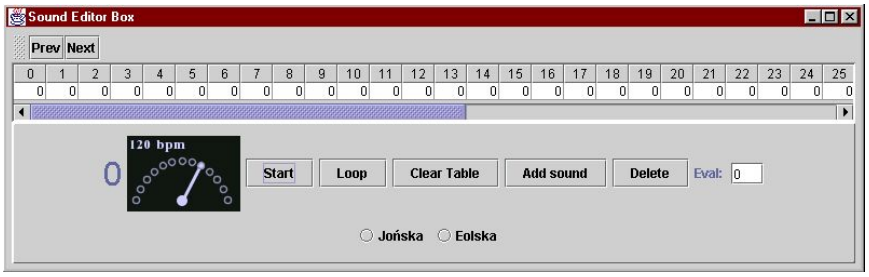
B – maksymalna liczba nut w takcie = 48, zatem:

$$\mathbf{C = 37^{48}.}$$

Poza tym, gdybyśmy dysponowali takim bankiem moglibyśmy po prostu wybrać z niego najlepsze takty (gdybyśmy mieli tylko czas na przeglądanie tak dużego zbioru ocen taktów).

W ostatecznej wersji systemu, takty z każdego pokolenia porównywane są z taktami pochodzącymi z banku. Jeśli w takcie z pokolenia wystąpi sekwencja nut z banku, jego ocenę zwiększa się o ocenę sekwencji z banku. Oceny taktów w banku pochodzą od samego autora tego banku. Każdy takt może otrzymać ocenę z zakresu 0..100.

Same takty banku jawią się dla użytkownika jako ciągi liczb z zakresu 1..21 (Rysunek 2), które odzwierciedlają sekwencje dźwięków wyróżnione przez autora banku. Przykładowo, ciąg 1,3,5 dla wybranej przez użytkownika skali C-dur oznacza ciąg dźwięków C, E, G. Jeżeli te dźwięki pojawią się w jakimkolwiek wygenerowanym takcie w tej właśnie kolejności, to ocena tego taktu wzrośnie o ocenę ciągu 1,3,5. Godnym zauważenia jest fakt, że sekwencje z banku (np.: 1,3,5) nie odzwierciedlają bezpośrednio konkretnych nut, a stopnie skali w jakiej aktualnie pracuje kompozytor oraz tej, której Ucho używa do oceny taktów.



Rysunek 2. Okno edytora banku brzmień

3. **Ocena rytmiki zrealizowana jest, podobnie jak wyżej ocena współbrzmienia nut, przy pomocy banku rytmów**.

W tym wypadku takt jest tablicą 48-elementową. Wypełnione pole tablicy oznacza nutę w takcie umiejscowioną w wybranym przez użytkownika miejscu taktu. W banku rytmów ostatnia zaznaczona w takcie nuta nie jest brana pod uwagę, stanowi ona jedynie ograniczenie długości trwania poprzedniej nuty (w przeciwnym wypadku ostatnia nuta dopełniałaby każdą sekwencję wartości rytmicznych do długości trwania taktu w metrum 4/4, patrz opis chromosomu w dalszej części pracy).

2.2 Opis algorytmu genetycznego, struktury chromosomu i użytych operatorów

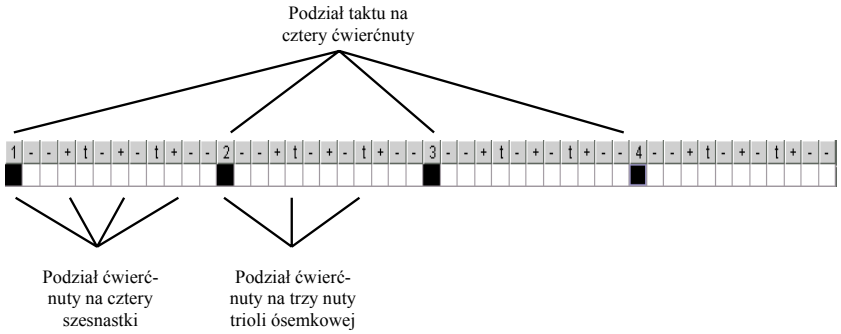
Zastosowany w systemie algorytm genetyczny właściwie nie różni się od standardowej postaci algorytmu genetycznego podanej w podręczniku Goldberga[1] czy Michalewicza[2]. Pierwsze pokolenie jest pokoleniem niemalże losowym, o liczbie osobników zadeklarowanych przez użytkownika. Każde następne pokolenie jest reprodukowane z poprzedniego po dokonaniu selekcji **metodą ruletki**. Użytkownik ma możliwość wyboru dowolnej liczby pokoleń, na którą uruchomiony zostaje algorytm genetyczny.

- *Reprezentacja taktu – reprezentacja chromosomu*

Chromosom (takt) to tablica złożona z 48 elementów 16 bitowych (typ *short* w Javie). Każdy element to niezależna nuta. Wartość elementu decyduje o wysokości nuty. Położenie elementu w chromosomie decyduje o długości trwania nuty. Każda kolejna nuta ogranicza długość trwania poprzedniej nuty. Długość ostatniej nuty ogranicza koniec taktu. Zastosowano więc kodowanie całkowitoliczbowe.

Przykład:

Gdybyśmy chcieli zapisać takt w metrum 4/4 składający się z 4 ćwierćnut o wysokości dźwięku C, to wyglądałby on tak jak na Rysunku 3.



Legenda:
‘1’,’2’,’3’,’4’ - odległość między kolejnymi liczbami to długość trwania ćwierćnuty
‘+’ - odległość między kolejnymi plusami to długość trwania szesnastki, w charakterze ‘+’ występują również liczby ‘1’,’2’,’3’,’4’
‘t’ - odległość między kolejnymi literkami ‘t’ to długość trwania trioli ósemkowych, w charakterze literki ‘t’ występują również liczby ‘1’,’2’,’3’,’4’
‘-‘ - pozostałe wartości rytmiczne

Rysunek 3. Przykładowy takt w metrum 4/4

Dokładnie widać, że elementy zaczernione reprezentują nuty, ich położenie na początku taktu, w ¼, ½, i ¾ części taktu odzwierciedla długość trwania nuty, natomiast nie prezentowana tutaj wartość każdego czarnego elementu, decyduje o wysokości brzmienia nuty.

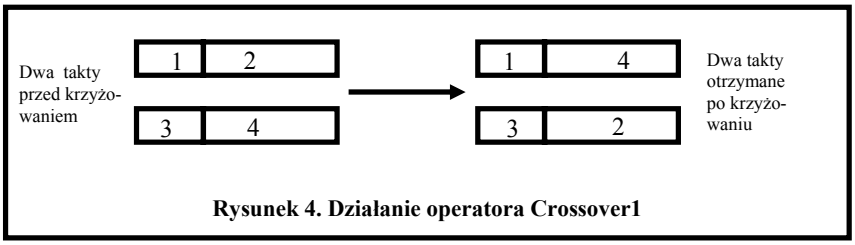
- *Operatory genetyczne. (Crossover1, Crossover2, Crossover3, Mutation1, Mutation2, My1)*

Crossover1 jest standardowym operatorem krzyżowania zaczerpniętym z Goldberga [1] (Rysunek 4). Losowo wybierany jest punkt przecięcia chromosomu (wartość z przedziału 0..47). Cięcie decyduje o wyglądzie dwóch taktów – potomków uzyskanych z dwóch taktów-rodziców. Należy przy tym zaznaczyć, że przy tej operacji możliwa jest zmiana długości trwania nut znajdujących się na końcu pierwszej części przecinanego taktu-rodzica. Nowa długość trwania nuty wcale nie musi mieć długości standardowej, tzn. takiej, jaką wykorzystuje w swym działaniu Kompozytor (ćwierćnuta, ósemka, ...). W ten sposób w takcie może zaplątać się nuta o

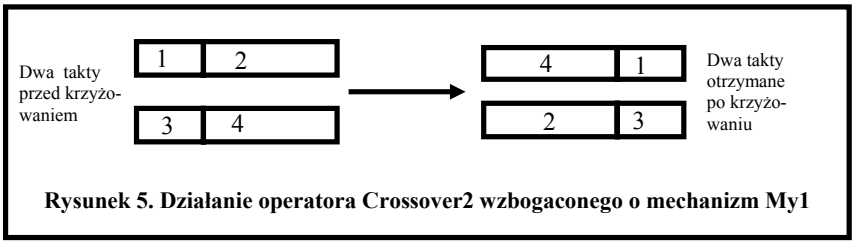
wartości rytmicznej ósemki z kropką, bądź każda inna nuta, której czas trwania jest wielokrotnością trzydziestkidwójki z kropką (każdy takt podzielony jest na maksymalnie 48 równych części – patrz opis genotypu). Autor projektu celowo nie eliminował tego zjawiska. Doświadczenie pokazało, że nowo powstałe nuty w niewielkim stopniu wpływają na nieregularność taktów w pokoleniu. Crossover1 sprzężony jest z mutacją **Mutation1**. W tym operatorze mutacji prawdopodobieństwo mutowania odnosi się do każdej, niepustej nuty w takcie. Nuta jest mutowana standardowo tylko pod względem wysokości brzmienia nuty (tak aby nowa wartość nuty należała do aktualnej skali), opcjonalnie można włączyć jej przemieszczenie się w takcie po dokonaniu mutacji (moving in mutation). Nowe położenie nuty wybierane jest tu w sposób zupełnie losowy.

Crossover2 jest operatorem bardzo podobnym do operatora Crossover1, ale wzbogacony jest o mechanizm **My1** (rys. 5). Mechanizm ten polega na zmianie sposobu tworzenia nowych taktów z taktów rodziców. My1 działa z pewnym ustalonym prawdopodobieństwem, dotyczącym każdego taktu i polega na zamianie kolejności dwóch sklejanych części tworzących nowy takt.

Crossover2 sprzęgnięty jest z innym operatorem mutacji niż Crossover1. **Mutation2** polega na mutowaniu całego taktu (prawdopodobieństwo mutowania przypada na pojedynczy takt) w pewien określony sposób. Losowo wybierana jest jedna nuta z taktu, a następnie mutowana jest jej wysokość brzmienia. Zmieniony został, włączany opcjonalnie, sposób przemieszczania się nuty w takcie. Nie zachodzi on całkowicie losowo, a w następujący sposób: znajdowana jest najdłuższa nuta w takcie, jej długość skracana jest do połowy poprzez wstawienie za nią nowej, zmutowanej nuty. Takie podejście pozwala na regularne rozmieszczenie się nut w takcie (zapobiega grupowaniu się nut koło siebie w takcie).



Rysunek 4. Działanie operatora Crossover1



Rysunek 5. Działanie operatora Crossover2 wzbogaconego o mechanizm My1

Crossover3 jest najbardziej wyspecjalizowanym, skomplikowanym i nowatorskim operatorem ze wszystkich wykorzystanych w systemie MGen. Jego działanie jest w pewien sposób dwupoziomowe. Jak już wiemy, takty oceniane są z trzech niezależnych punktów widzenia. Punkty te można podzielić na te oceniające melodię (przynależność nut do skali, współbrzmienie nut z sobą) oraz te oceniające rytmikę. Idea operatora jest więc taka aby w czasie selekcji stworzyć dwa pokolenia rodzicielskie, każde o liczbie osobników równej liczności populacji, gdzie w pierwszej puli znalazłyby się takty o najładniejszym brzmieniu, a w drugiej puli takty o najlepszej rytmice. Wtedy takty przeznaczone do kolejnego pokolenia byłyby krzyżówką taktów z obu pokoleń rodzicielskich.
Crossover3 wykorzystuje więc inny mechanizm selekcji – **Selection2**. Selection2 metodą ruletki tworzy dwa pokolenia rodzicielskie: *RythmParents* i *SoundParents*. Obie pule rodzicielskie nie muszą być zbiorami rozłącznymi. Działanie Crossover3 jest następujące: w pierwszej fazie działanie operatora jest identyczne z działaniem operatora Crossover2, z tą różnicą, że pulą rodzicielską są takty z puli *RythmParents*. Tworzone są takty będące wynikiem krzyżowania taktów o najlepszej rytmice w pokoleniu. Następnie, w drugiej fazie, wartości melodyczne nut z taktów są nadpisywane wartościami melodycznymi taktów pochodzących z puli rodzicielskiej *SoundParents*. Operator Crossover3 sprzężony jest z wersją mutacji Mutation2.

- *Moduł Aranżer*(*SongMaker*)

Jego działanie polega na grupowaniu taktów w większe formy muzyczne (frazy). Takty do tworzenia fraz pobiera on z przygotowanego wcześniej przez użytkownika banku taktów. Użytkownik, obserwując ewolucję taktów, może wybierać sobie takty odpowiadające jego preferencjom i podsyłać je aranżerowi (przycisk Add na ekranie głównym aplikacji). Aranżer składa frazy według podanej przez użytkownika formy. Forma może mieć przykładową postać:

Forma1: abcddabcdd,

gdzie każdy znak odpowiada jednemu taktowi (niekoniecznie różnemu od innych znaków). Fraza w tym przypadku budowana jest z czterech części *a*, *b*, *c* oraz *d* składanych przez aranżer w podanej w Formie1 kolejności. Przyporządkowanie taktów do znaków odbywa się losowo w przestrzeni wybranych przez użytkownika taktów (losowanie ze zwracaniem). Aranżer pozwala na zapisanie powstałych fraz do pliku Midi oraz odczyt wcześniej powstałych fraz zapisanych w pliku Midi.

3. System MGen w praktyce

Program uruchamia się wykonując plik wsadowy *run.bat* z katalogu *classes* po uprzednim upewnieniu się, że w systemie zainstalowany jest JDK w wersji 1.3. Może zdarzyć się tak, że nie są ustawione ścieżki do JDK i wtedy warto przekopiować do katalogu *classes* plik *java.exe*.

Czynności wykonywane po uruchomieniu programu:

- Przy pomocy *Kompozytora* należy wygenerować początkowe pokolenie taktów. Zalecana wielkość pokolenia to 200 taktów. Po wygenerowaniu pokolenia pojawi się jego reprezentacja widoczna w oknie aplikacji. Otrzymane takty można odsłuchiwać klikając na ocenę danego taktu i w ten sposób zaznaczając go, a następnie naciskając przycisk *Start* na panelu po lewej stronie okna aplikacji.
- Następnie należy utworzyć banki brzmień i rytmów. Można to zrobić ręcznie (menu *Bank*|*EditXXXBank*), ale można też skorzystać z banków przygotowanych przez autora wybierając z menu opcje *Bank*|*OpenXXXBank,* otworzyć banki *rbank2.rbk* oraz *bank2.sbk* z katalogu *Bank*.
- Aby banki te zostały wykorzystane do ocenienia zerowego pokolenia należy wybrać opcję *Evaluate* (żółta ikonka z paska narzędzi lub opcja z menu *Tools*|*Evaluate*). Po wybraniu tej opcji oceny taktów widoczne na ekranie powinny się zwiększyć. Dowodzi to prawidłowego działania banków.
- Możemy teraz przejść do generowania następnego pokolenia klikając niebieską ikonkę z paska narzędzi lub dowolną liczbę pokoleń wprzód wybierając opcję z menu *Tools*|*Next n Generations*. W czasie podróży po pokoleniach możemy dowolnie zmieniać parametry algorytmu genetycznego. Dotyczy to zarówno zmiany prawdopodobieństwa krzyżowania i mutacji (opcja z menu *File*|*Options*) jak również możemy włączać i wyłączać mutację taktów (*Tools*|*Mutation*) oraz przemieszczanie się nut w takcie w czasie mutacji (*Tools*|*Moving in Mutation*). Dowolnie mogą być również zmieniane banki w czasie działania algorytmu.
- W czasie ewolucji pokoleń należy również dostarczać wybrane przez nas takty aranżerowi (*SongMaker*) do komponowania większych fraz. Dokonujemy tego klikając na ocenę dowolnego taktu a następnie na przycisk *Add* na panelu po lewej stronie okna aplikacji. Aranżer *SongMaker* można w dowolnej chwili resetować, tzn. usuwać wszystkie podesłane mu wcześniej takty (*Tools*|*ResetSoundMaker*).
- W celu uruchomienia *SongMaker'a* wybieramy opcję z menu *Tools*|*SongMaker*. Wpisujemy formułę i klikamy przycisk *Create*. Po tym możemy odsłuchać stworzoną frazę (*Play*) lub zapisać ja do pliku .midi (*Save*).

4. Analiza wyników, wnioski dotyczące systemu MGen

W trakcie powstawania systemu autor spotkał się z wieloma zjawiskami charakterystycznymi dla AG. Dla wybranej zbyt małej populacji początkowej, takty bardzo szybko się do siebie upodobniały. Aby temu zaradzić włączono pierwszą wersję mutacji (Mutation1). Nie był to jednak operator właściwy dla przyjętej reprezentacji taktów. Zbyt chaotyczne jego działanie prowadziło do grupowania się nut koło siebie w jednym miejscu taktu, przez co takty sprawiały wrażenie nieskładnych i mocno przypadkowych. Również pierwsza wersja krzyżowania (Crossover1) nie dopuszczała do przepływu nutek z jednej strony taktu na drugą.

Dopiero wprowadzenie drugiej wersji krzyżowania doprowadziło do ciągłego wzrostu ocen w populacji przy zachowaniu różnorodności taktów. Operator My1 pozwolił na przepływ nut z początku na koniec taktu i odwrotnie (operator My1 wywoływany jest z prawdopodobieństwem 0.01 dla każdego zachodzącego krzyżowania). Druga wersja mutacji, oprócz zmiany wysokości nut, odpowiedzialna stała się za losowe rozbijanie leżących zbyt blisko siebie nut.

Pierwsza wersja krzyżowania dawała mało ciekawe wyniki. Najciekawsze wyniki osiągnięto stosując drugą i trzecią wersję krzyżowania.

Proponowane prawdopodobieństwa dla operatorów oraz inne parametry algorytmu:

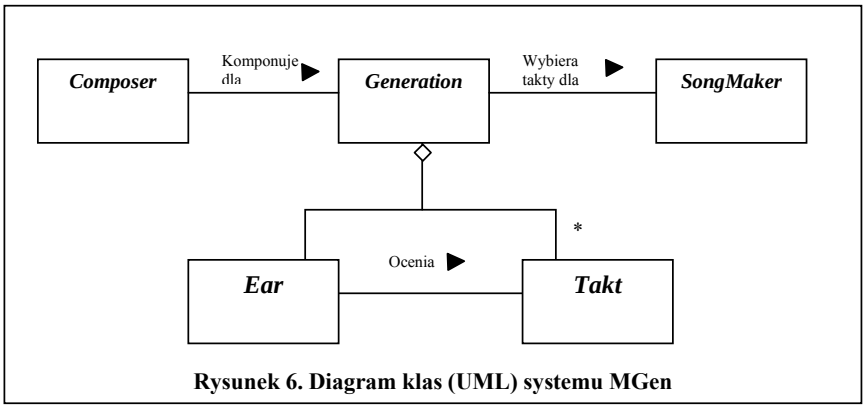
- Prawdopodobieństwo krzyżowania, pcross = 0.99
- Prawdopodobieństwo pmut = 0.05
- Liczebność osobników w pokoleniu, size_of_generation = 200 (ciekawe efekty uzyskiwano również dla mniejszych pokoleń - liczba osobników rzędu 40).
- Liczba generacji – proponowane jest obserwowanie kolejnych pokoleń i wybór z nich taktów do aranżera na bieżąco (liczba taktów w aranżerze może być dowolna, optymalnie 5 – 10). W razie upodobnienia się osobników proponowane jest zwiększenie prawdopodobieństwa mutacji i włączenie algorytmu na około 10 pokoleń.

5. Implementacja systemu MGen

System MGen napisany został w języku programowania Java, który jest językiem bardzo mocno zorientowanym na programowanie obiektowe. Dlatego naturalnym stało się zaimplementowanie wszystkich modułów systemu jako klas języka Java. Oto krótki opis głównych klas systemu wraz z diagramem UML (rys. 6):

Klasa *Takt* – klasa modelująca takt w metrum 4/4. Jej instancje to tablice 48 elementów typu *Short.*

Klasa *Generation* – klasa, której obiekt to tablica obiektów klasy Takt. Jedna ze zmiennych stanu obiektu klasy *Generation*, *No_Generation*, to zmienna określająca numer pokolenia, które zawiera sam obiekt. Jedna z metod obiektu, metoda *NextGeneration()*, dokonuje przejścia obiektu w stan nowego pokolenia



Rysunek 6. Diagram klas (UML) systemu MGen

Klasa *Kompozytor* – odpowiedzialna jest za generowanie materiału muzycznego. Jedna z jej metod jako argument przyjmuje obiekt klasy Takt i wypełnia go wygenerowanymi nutami

Klasa *Ear* – klasa będąca częścią klasy *Generation*. Odpowiedzialna jest za ocenę wszystkich taktów wchodzących w skład klasy *Generation*.

Klasa *SongMaker* – klasa odpowiedzialna za tworzenie większych fraz z taktów pochodzących z ewolucji, a przekazanych do *SongMaker'a* z klasy *Generation*.

6. Podsumowanie

Ocena muzyki musi mieć charakter wyłącznie jakościowy i taka jest również postać tego artykułu. Wyniki działania algorytmu dostępne są w plikach .midi i bynajmniej nie służą one do słuchania a jedynie do dalszej ich obróbki (brak interpretacji ze strony odtwórcy a jedynie suchy zapis materiału nutowego). Pliki .midi są pełnowartościowym zapisem materiału muzycznego, dlatego stworzona przez generator muzyka może być w dowolny sposób wykorzystywana i aranżowana w dalszym procesie obróbki. Prawda jest taka, że jej wykorzystanie zależy tylko i wyłącznie od pomysłowości i umiejętności ludzi zajmujących się muzyką na stałe i w sposób profesjonalny.

Literatura

[1] Zbigniew Michalewicz „Algorytmy genetyczne + struktury danych = programy ewolucyjne", WNT, Warszawa, 1999r.

[2] David E. Goldberg „Algorytmy genetyczne i ich zastosowania", Warszawa, 1996r.

[3] Bruce Jacob „Algorithmic Composition for „AcousticInstruments" http://www.ee.umd.edu/~blj/algorithmic_composition/

[4] Bruce Jacob „COMPOSING WITH GENETIC ALGORITHMS" http://www.ee.umd.edu/~blj/algorithmic_composition/icmc.95.html

[5] Bruce Jacob „Algorithmic Composition As A Model Of Creativity" http://www.ee.umd.edu/~blj/algorithmic_composition/ algorithmicmodel.html

[6] Bruce Jacob „THEME DEVELOPMENT IN ALGORITHMIC COMPOSITION" http://www.ee.umd.edu/~blj/algorithmic_composition/nebula.html

[7] ThinkQuest Library of Entries "Threads: Computers, Music, and a Little Theory" http://library.thinkquest.org/3343/web-docs/page2.html

[8] Al Biles**,** opis systemu GenJam, http://www.it.rit.edu/~jab/GenJam.html

[9] Peter M. Todd, Gregory M. Werner **„**Frankensteinian Methods for Evolutionary Music Composition" http://citeseer.nj.nec.com/242222.html

[10] Uses of GAs in the Real World **–** Music Composition http://www.cs.qub.ac.uk/~M. Sullivan/ga/ga8.html

[11] Eduardo Reck Miranda „Composing Music with computers" http://mapage.noos.fr/press.release/compmusic.htm

ALGORYTMY GENETYCZNE W ROZWIĄZYWANIU PROBLEMÓW – GENEROWANIE MUZYKI

Błażej Budzyński
Wydziałowy Zakład Informatyki, Politechnika Wrocławska
E-mail: budzynski@poczta.visnet.pl

Streszczenie

W pracy omówiono zastosowanie algorytmów genetycznych w procesie komponowania muzyki. W punkcie pierwszym postaramy się znaleźć odpowiedzieć na pytanie, czym właściwie jest proces komponowania. Rozdział drugi jest krótkim wstępem do algorytmów genetycznych. Punkty trzy, cztery, pięć oraz sześć omawiają dokładnie istotę działania algorytmu komponującego opartego na AG. W końcowym rozdziale przypatrzymy się bliżej otrzymanym wynikom.

Wstęp

Komponować – to słowo słyszeli wszyscy ale co ono dokładnie oznacza? Cytując za [6]: „komponować – tworzyć dzieło sztuki (zwłaszcza muzycznej); rozmieszczać odpowiednio elementy (dzieła), układać z nich harmonijną całość". „Tworzyć dzieło sztuki" – taki termin zadowoli każdego ale jak go przenieść do świata algorytmów genetycznych. Pojęcie to (dzieło sztuki) nie dość, że jest niejednoznaczne, to na dodatek „przeogromne" – łączy w sobie elementy socjologii, psychologii oraz filozofii. Dlatego też w dalszych rozważaniach skupimy się na drugiej części definicji.

Zastanówmy się przez chwilę co pcha człowieka, aby „rozmieszczać odpowiednio elementy" oraz „układać z nich harmonijną całość". Ciśnie się od razu na usta, iż ma on inwencję (twórczą). No dobrze, ale, drążąc dalej temat, czymże jest inwencja? Nasuwają się dwie odpowiedzi: jest to przebłysk geniuszu (lub przypływ inspiracji) bądź też jest to proces stopniowego, krok po kroku, budowania dzieła (ciężka praca). O ile tego pierwszego nie umiemy zamodelować i nie uczynimy tego tak długo dopóki nie pojmiemy czym jest geniusz (bądź inspiracja), o tyle ten drugi, z samej swojej istoty, daje się łatwo zalgorytmizować.

W wydaniu elektronicznym, algorytmy próbujące naśladować działania podejmowane przez człowieka podczas komponowania muzyki, obejmują swym zasięgiem wiele technik: od prostych metod stochastycznych jak w *M and Jam Factory,* aż do skomplikowanych systemów ekspertowych, takich jak *EMI* Davida Cope'a czy *Cypher* Roberta Rowe'a ([8], [10]).

Systemy *EMI* oraz *Cypher* działają na podobnej zasadzie – generują muzykę analizując utwory stworzone przez człowieka. O ile jednak *EMI* analizuje dane wejściowe z gotowych kompozycji, o tyle system Rowe'a otrzymuje na wejściu dane

od muzyka grającego na żywo. Rowe przy tym odróżnia transformacyjne komponowanie muzyki od komponowania opartego na generowaniu. Chociaż jego *Cypher* zawiera oba elementy, swoje działanie opiera głównie na pierwszym sposobie – pobiera dane wejściowe od użytkownika, poddaje je serii transformacji, otrzymując na wyjściu coś pochodnego (ale niekoniecznie podobnego).

1. Algorytmy genetyczne oraz komponowanie muzyki – gdzie związek ?

Popularną metodą rozwiązywania problemu, szukania odpowiedzi na pytanie, czy - w ogólności - uzyskania zbioru spełniającego postawione wymagania, jest konwersja tegoż problemu (pytania) do postaci „problemu przeszukiwania". Idea jest następująca: przeszukanie całego zbioru możliwych rozwiązań tak, aby znaleźć jedno (rozwiązanie) najlepiej odpowiadające zadanym kryteriom. Jednakże, aby proces poszukiwania nie trwał niepomiernie długo, co mogłoby mieć miejsce przy sprawdzaniu wszystkich elementów zbioru potencjalnych rozwiązań, należy ten zbiór ograniczyć. Zazwyczaj czynność ta jest najtrudniejszym etapem przy stosowaniu techniki przeszukiwania.

Komponowanie muzyki można właśnie rozważać jako taki problem: bierzemy pod uwagę zbiór wszystkich możliwych kompozycji (który jest nieskończenie wielki) jako przestrzeń poszukiwań, z kryterium postawionym jako: „znaleźć kompozycję (lub prościej frazę[3]), która brzmi dobrze". Niestety, przestrzeń ta jest kompletnie nieuporządkowana co, mówiąc w przenośni, powoduje, że dobre rozwiązania mogą leżeć obok kompletnie niedobrych (w sensie: brzydkich, złych). Zmiana tylko kilku nut we frazie może ją uczynić o wiele mniej interesującą pomimo, iż obie „wyglądają" praktycznie identycznie. Właściwości te implikują, że proces poszukiwania rozwiązania jest trudny oraz nie dający się łatwo przewidzieć.

Do rozwiązania problemów przeszukiwania dobrze nadają się algorytmy genetyczne ([3], [2]), technika oparta na mechanizmach doboru naturalnego oraz dziedziczności. Algorytm rozpoczyna działanie z losowo wygenerowanymi rozwiązaniami danego problemu i używając odpowiednika biologicznej rekombinacji szuka coraz to lepszych rozwiązań. Potencjalne rozwiązania przedstawiane są jako chromosomy składające się z alleli, które to zbudowane są z liczb (ciągów liczb, bitów – w zależności od problemu). W takim przypadku rekombinacja jest po prostu procesem tworzenia nowego chromosomu na podstawie alleli zawartych w chromosomach rodziców. Ewolucja rozwiązań (w kierunku „lepszych") odbywa się poprzez wybieranie ciągów, które najlepiej odpowiadają postawionemu kryterium (kryteriom) i krzyżowanie ich. Pomimo elementu losowości, algorytmy genetyczne nie sprowadzają się do zwykłego błądzenia przypadkowego. Wykorzystują one efektywnie przeszłe doświadczenia do określenia nowego obszaru poszukiwań o spodziewanej podwyższonej wydajności.

Komentarz [V1]:

[3] **fraza** - odcinek melodii, obejmujący kilka taktów, stanowiący wyodrębnioną całość.

Jedną z pierwszych prób wykorzystania algorytmów genetycznych do komponowania muzyki były działania podjęte przez Hornera i Goldberga ([4], [7], [9]). Stworzyli oni AG służący do transformacji tematów muzycznych. Sekwencja prostych operacji, w z góry określonej liczbie kroków, miała za zadanie przeprowadzić początkową frazę do frazy pożądanej (również zadanej). Operacje obejmowały wstawianie, usuwanie oraz rotację (całego tematu) nut. Każdy osobnik w populacji był sekwencją tychże operacji. W celu obliczenia wskaźnika przystosowania, dana sekwencja operacji była wykonywana na początkowej („wejściowej") frazie w celu wygenerowania frazy wynikowej („przetransformowanej"). Następnie, funkcja celu brała pod uwagę dwa czynniki, oceniając osobnika tym wyżej im bardziej efekt jego działania (fraza przetransformowana) był podobny do frazy pożądanej oraz im bardziej liczba kroków transformacji była bliższa do zadanej.

Inny algorytm zaprezentował Horowitz ([5], [7], [9]). W przeciwieństwie jednak do poprzedników nie zajmował się on melodiami lecz rytmami. Ocena osobników odbywała się dwustopniowo. Najpierw podlegały one działaniu algorytmu genetycznego z określoną funkcją celu, a następnie otrzymane w ten sposób wyniki były oceniane przez człowieka i to ta ocena wpływała na dalszą ewolucję. Funkcja celu użyta w AG była ważoną sumą tego, jak bardzo dany osobnik (rytm) różni się od podanych (pożądanych) wartości takich jak synkopa[4], stopień akcentowania taktu (ang. *downbeat* – akcentowana miara taktu), stopień powtórzenia oraz kilku innych czynników.

Najgłośniejszym i najbardziej znanym przykładem połączenia AG z muzyką jest praca autorstwa Johna Bilesa [1]. Jego program, *GenJam*, generujący jazzowe „solówki", jest bazującym na algorytmach genetycznych modelem tego jak początkujący muzyk jazzowy uczy się improwizacji. Działanie algorytmu opiera się na kilku (różnych) hierarchicznie „poukładanych" populacjach, zawierających potencjalne, nowe muzyczne pomysły. Wszystkie te populacje służą do zbudowania jednej partii solowej. Każda melodia („solówka") odgrywana przez program jest na bieżąco oceniana przez człowieka (określanego mianem mentora). Wskaźnik przystosowania tejże melodii (osobnika) jest zwiększany za każdym razem gdy użytkownik wciśnie klawisz 'g' (ang. *good*), a zmniejszany gdy użytkownik naciśnie 'b' (ang. *bad*). Używając różnorodnych operatorów genetycznych oraz danych podanych przez mentora (m.in. użyte akordy, progresja[5]), algorytm tworzy nowe populacje, które z dużym prawdopodobieństwem zawierają „lepsze" (bardziej obiecujące) pomysły.

---

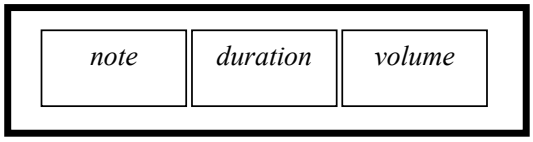
[4] **synkopa** - termin muzyczny oznaczający przeniesienie akcentu z mocnej części taktu na jego słabą część.

[5] **progresja -** termin muzyczny dotyczący przeniesienia danej melodii lub struktury harmonicznej o określoną odległość w górę lub w dół skali muzycznej. Progresja jest często stosowanym przez kompozytorów środkiem konstrukcyjnym.

Dalsza część pracy prezentuje algorytm zastosowany w systemie MUZG (od MUZyczne Geny).

2. Budowa osobnika w programie *MUZG*

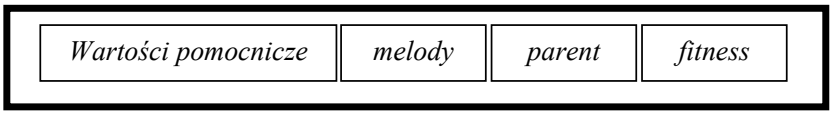
Strukturę genotypu, który odpowiada pojedynczej nucie, przedstawia rysunek 1. Pole *note* (*nuta*) oznacza wysokość dźwięku, pole *duration* (*okres*, *czas trwania*) określa długość dźwięku, a pole *volume* (*głośność*) odpowiada za głośność danej nuty.



Rysunek 1. Budowa genotypu

Wszystkie trzy pola mogą przyjmować tylko wartości całkowitoliczbowe.

Na rysunku 2 zaprezentowano budowę osobnika. Chromosomem (ciągiem kodowym) w tym osobniku jest (tablica) *melody*. Pole *parent* to dwuelementowa tablica w której zapisywane są numery rodziców (przykładowo dla populacji liczącej 250 osobników, numer rodzica jest liczbą z przedziału [0,249]). W polu *fitness*



Rysunek 2. Budowa osobnika

zapamiętywane jest przystosowanie danego osobnika. W polu *wartości pomocnicze* przechowywane są takie informacje jak: tonacja, ilość oktaw z których osobnik może losować dźwięki itp.

3. Operatory genetyczne - krzyżowanie i mutacja

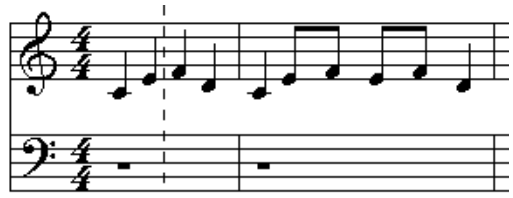
Aby lepiej zrozumieć ideę krzyżowania i mutacji, prześledźmy działanie obu tych operatorów krok po kroku. Zacznijmy od operatora krzyżowania.

Załóżmy, że mamy dwa „osobniki" (tak naprawdę mamy tylko odpowiednik pola *melody* z rysunku 2) takie jak na rysunkach 3 i 4. Przerywaną linią zaznaczono punkt krzyżowania. W wyniku otrzymamy osobnika z rysunku 5a.

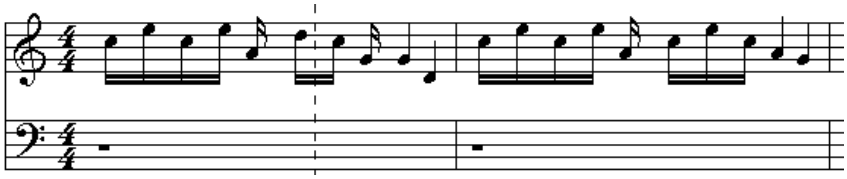
Patrząc uważnie na powstałego po krzyżowaniu osobnika, łatwo spostrzec błąd w pierwszym takcie. Ma tu miejsce następująca sytuacja – w wyniku krzyżowania w nowo powstałym osobniku otrzymaliśmy niewłaściwe (niedozwolone) wartości czasu trwania dźwięku (*duration*). W takiej sytuacji należy zastosować procedurę, która podzieli wartość *duration* na kilka mniejszych, dozwolonych wartości, dodając przy

okazji (jeśli zachodzi taka potrzeba) nowe pozycje w tablicy *melody*. Operacja ta w pewnym stopniu, ingeruje w ewolucję („psuje") osobnika, ale przy przyjętym sposobie kodowania ciężko byłoby znaleźć inne rozwiązanie. Ostateczny wynik krzyżowania przedstawia rysunek 5b. W tym przypadku wystarczyło zamienić ćwierćnutę na ósemkę.

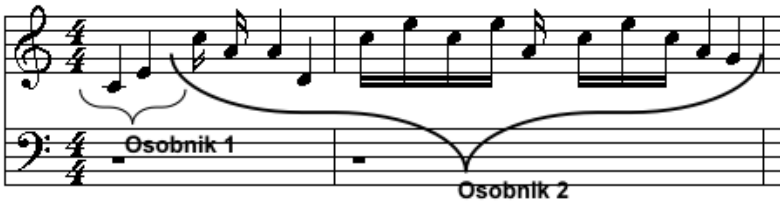
Po powyższym przykładzie widać, iż działanie operatora krzyżowania opiera się głównie na „kopiowaniu" melodii. Inaczej mówiąc, z zadanym prawdopodobieństwem sprawdzamy czy krzyżowanie w ogóle zachodzi, wybieramy losowo punkt krzyżowania (przerywana linia na rysunkach 3 i 4) i ostatecznie właśnie kopiujemy



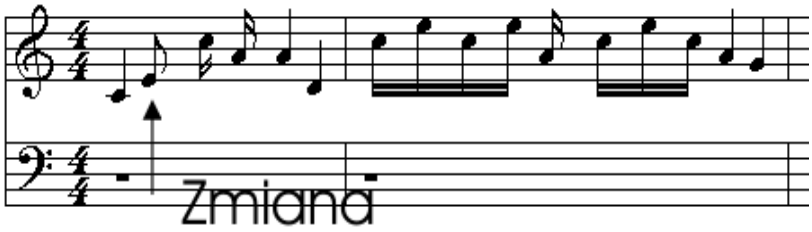
Rysunek 3. Osobnik nr 1



Rysunek 4. Osobnik nr 2



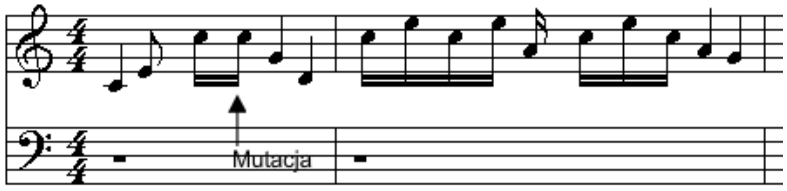
Rysunek 5a. Wynik krzyżowania



Rysunek 5b. Wynik krzyżowania (poprawiony)

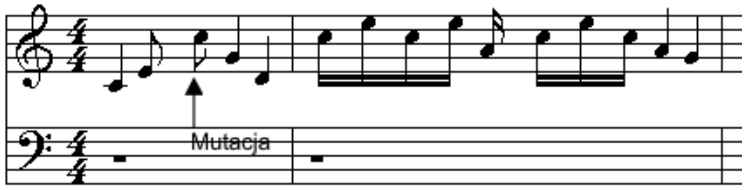
melodię.

Przejdźmy teraz do omówienia operatora mutacji. Zasadniczym elementem tego operatora jest zmiana wysokości dźwięku. Operacja ta ma podstawowe znaczenie dla algorytmu z opcją „wariacje" lub „schemat" (patrz niżej). Rysunek 6a prezentuje działanie tego kroku, przy założeniu, że osobnik poddany działaniu operatora jest taki jak na rysunku 5b.



Rysunek 6a. Operator mutacji – zmiana wysokości dźwięku

Drugim elementem procedury mutacji jest zmniejszanie głośności oraz łączenie (scalanie) nut. Ma to zapobiegać powstawaniu efektu objawiającego się „nadmiernym wypełnieniem" taktu nutami. Chodzi o to, iż wraz z kolejnymi iteracjami algorytmu genetycznego, ulega zapełnieniu tablica *melody*, co przy odsłuchu bardziej przypomina szybką zmianę częstotliwości dźwięku (oczywiście wszystko zależy od tempa) niż melodię (inaczej: użytkownik „nie widzi" w takim osobniku melodii co wpływa oczywiście na jego ocenę). Zjawisko to wynika, jak się łatwo domyślić, z dużej liczby losowo wygenerowanych osobników początkowych. Rysunek 6b przedstawia osobnika po wykonaniu tego kroku mutacji.



Rysunek 6b. Operator mutacji – „łączenie" nut

4. Funkcja celu

Funkcja celu jest „wielostopniowa". Pod pojęciem „wielostopniowa" należy tutaj rozumieć nie tylko ilość kryteriów oceniających ,ale także to iż niektóre z kryteriów nie oceniają osobnika (jako całości), a jedynie sąsiednie nuty (nie są to więc tak elementy oceniające w sensie algorytmów genetycznych). Funkcja celu zostanie omówiona krok po kroku dokładnie w takiej kolejności w jakiej oceniane są osobniki.

Zaczynamy od sprawdzenia nie osobników, ale poszczególnych nut (i ich „sąsiadów"). Po pierwsze, sprawdzamy jakie jest „prawdopodobieństwo" danej

długości trwania dźwięku. W celu uzyskania dobrych (czytaj: prostych) melodii, przyjęto, że najczęściej występują ósemki oraz ćwierćnuty.
Następnie, porównujemy dwie nuty pod kątem wysokości dźwięków (im mniejsza różnica między wysokościami tym lepiej). W tym miejscu należy zwrócić uwagę na „wyjątek": dźwięki o tej samej wysokości są oceniane gorzej od dźwięków różniących się co do wysokości.

Po trzecie, funkcja celu bada parametr *duration* dwóch sąsiadujących nut. Im większa jest różnica tym gorzej (przykładowo: trzydziestkadwójka sąsiadująca z całą nutą jest mało prawdopodobna i przez to źle oceniana).

Teraz przystępujemy do oceny całego osobnika.
Zaczynamy od sprawdzenia „wypełnienia" frazy (tablicy *melody*). Tutaj przyjęto, że bardziej równomierne rozłożenie oceniane jest wyżej. Oznacza to, że np. fraza zbudowana z dwóch taktów, jeden składający się z samych trzydziestekdwójek, drugi z całej nuty jest oceniany niżej niż dwutaktowa fraza zbudowana z samych ósemek. Na marginesie: z całą pewnością istnieją tysiące kompozycji, w których występują frazy podobne do pierwszej, ale każde poczynione tutaj założenie jest pewnym (często dość sporym) uproszczeniem.

Następnie znajdujemy najczęściej powtarzającą się wartość (dominantę) *duration*.
Kolejnym krokiem jest sprawdzenie odchylenia standardowego długości. Czynność ta łączy się z poprzednią. W zależności od parametrów możemy otrzymywać melodie od prostych (co, na przykład, oznacza cztery takty po dwie półnuty każdy) do bardziej skomplikowanych.

Ostatecznie badamy odchylenie standardowe wysokości. W tym miejscu możemy wpływać na melodię, dążąc do uzyskania coraz prostszych (np. dwa dźwięki grane naprzemiennie) lub coraz bardziej różnorodnych (w zależności od parametru programu).

5. Działanie algorytmu genetycznego w programie MUZG

Mając powyższe struktury danych oraz procedury możemy przejść do omówienia istoty działania algorytmu. Postępowanie tutaj przedstawione jest zbliżone do prostego algorytmu genetycznego zaprezentowanego w książce [2].
Pierwszą czynnością jaką należy wykonać, jest ustawienie parametrów algorytmu genetycznego (takich jak prawdopodobieństwo krzyżowania czy mutacji). W tym miejscu należy zaznaczyć, iż podobnie jak ma to miejsce w innych przypadkach algorytmów ewolucyjnych w sztuce, prawdopodobieństwo mutacji powinno być znacznie większe od standardowego (jeśli za standardowe przyjmiemy przedział między <0,01; 0,02>). Jest to spowodowane tym iż to właśnie mutacja jest operacją, posuwającą nasze muzyczne poszukiwania naprzód.

Następnie musimy wybrać metodę selekcji. Również i tutaj musimy się zatrzymać na chwilę. W programie zostały zaprogramowane trzy metody selekcji: według reguły ruletki, turniejowa oraz losowa według reszt bez powtórzeń. Po przeprowadzeniu

szeregu eksperymentów okazało się, że najlepsze rezultaty daje metoda turniejowa (patrz wykres 2).

Po wykonaniu przez użytkownika dwóch powyższych kroków do działania przystępuje komputer. Generuje on populację początkową, a następnie powtarza operacje selekcji, krzyżowania i mutacji (dla całej populacji) *n* razy, gdzie *n* jest parametrem użytkownika.

Ostatnią fazą algorytmu jest ocena (zatwierdzanie) wyniku. Jeśli otrzymany rezultat działania AG zadowala użytkownika, jest on zatrzymywany. W przeciwnym przypadku należy powtórzyć powyższe kroki.

Warto zaznaczyć, iż algorytm genetyczny nie generuje „całej muzyki" (całej kompozycji) na raz. Pojedyncze wywołanie algorytmu genetycznego produkuje jedynie melodie, z których to dopiero użytkownik składa cały utwór. Inspiracją do takiego podejścia była (oczywiście oprócz prostoty funkcji celu) znana w informatyce (i nie tylko) metoda „dziel i rządź". Ale uwaga: jest to spore ograniczenie (i uproszczenie) i należy wątpić, aby taką metodą udało się zbudować bardziej rozbudowany utwór.

Ponadto, zastosowano dwa dodatkowe mechanizmy mające na celu poprawienie uzyskiwanych wyników.

Pierwszy to wariacje[6]. Przez wariacje rozumiemy tutaj tylko zmianę tematu. Istotę działania oddaje następujący algorytm:

- Przeprowadź algorytm genetyczny (z zadanymi parametrami) w celu znalezienia tematu (fraza, która będzie zmieniana).
- Ze znalezionego tematu (który, w kategoriach AG, jest najlepszym osobnikiem uzyskanym w powyższym kroku) utwórz nową populację.
- Przeprowadź jeszcze raz algorytm genetyczny dla tak utworzonej populacji. Podczas działania algorytmu czynnikiem zmieniającym osobniki jest mutacja, która zmienia wysokość dźwięku nut.

Metoda ta daje zdecydowanie najlepsze rezultaty. Powody takiego stanu rzeczy wydają się być intuicyjnie jasne.

Po pierwsze, fraza służąca za temat wariacji jest już oceniona („dobra"; nie wychodzimy od populacji początkowej wygenerowanej losowo).

Po drugie zaś, mutacja nie zmienia wszystkich nut. Uzyskujemy w ten sposób frazę zbudowaną z kilkukrotnie powtórzonego tematu podstawowego (przez co łatwiej „wpada w ucho", a co za tym idzie, fraza taka jest wyżej oceniana), w której mutacja dokonała niewielkich zmian.

---

[6] **wariacje** - forma muzyczna oparta na zmianach tematu, reprezentatywna dla muzyki klasyczno-romantycznej (temat z wariacjami), także technika kompozytorska. Temat wariacji mógł być tworzony przez kompozytora lub przejęty z dzieła innego twórcy. Liczba wariacji nie była ściśle określona. Każda z kolejnych wariacji była przekształceniem tematu polegającym m.in. na figuracji partii melodycznej lub partii lewej ręki, zmianach harmonicznych, wprowadzeniu techniki polifonicznej, zmianach rytmu i metrum.

Drugi mechanizm, bardzo podobny do wariacji, polega na odpowiednim sposobie tworzenia populacji początkowej. Metoda ta sprowadza się do wygenerowania jednego osobnika całkowicie losowo, następnie na zapisaniu jego budowy (mówiąc obrazowo – zapamiętujemy „gdzie we frazie są nuty") i na tej podstawie generowania reszty populacji początkowej, losując już tylko wysokość dźwięków dla poszczególnych osobników. Dla tak uzyskanej populacji wykonujemy algorytm genetyczny. Wielką zaletą tej techniki jest większa różnorodność uzyskiwanych melodii. Z początku może się to wydawać dziwne. Wytłumaczenie jest proste – w momencie gdy mamy całą populację wygenerowaną losowo, mutacja, wraz z postępem działania algorytmu, sprowadza osobniki do prostych, „podobnych" melodii (w przypadku doświadczeń z „progiem" równym „ćwierćnucie", większość melodii - około 90% - była zbudowana wyłącznie z ćwierćnut oraz z ósemek). W momencie zastosowania tej metody, podobnie jak przy wariacjach, mutacja „pcha" algorytm w poszukiwaniu jak najlepszej melodii, w ramach przyjętego układu.

6. Wyniki

Na wykresie 1. przedstawiono zmianę wskaźnika przystosowania w zależności od liczby osobników. Widać wyraźnie, że wraz ze wzrostem liczby osobników, algorytm szybciej znajduje dobre rozwiązania. Niestety, koszt tego jest znaczny (patrz tabela 2). Parametry, dla których uzyskano omawiane wyniki zostały zebrane w tabeli 1. W tym miejscu należą się również dwa zdania wyjaśnienia na temat zbieżności wszystkich średnich wartości przystosowania (avg_50, avg_250 i avg_500) do (mniej więcej) jednej wartości. Jak można się domyślać, za taką sytuację odpowiada długość chromosomu. Przy tak niewielkiej liczbie taktów (długości chromosomu), dość „rygorystyczna" funkcja celu, ostatecznie zawsze znajdzie „podobne" osobniki.

Parametr	Wartość
długość chromosomu	4 (czyli cztery takty)
ilość iteracji	250
prawdopodobieństwo krzyżowania	0,95
prawdopodobieństwo mutacji	0,2
metoda selekcji	losowa według reszt bez powtórzeń

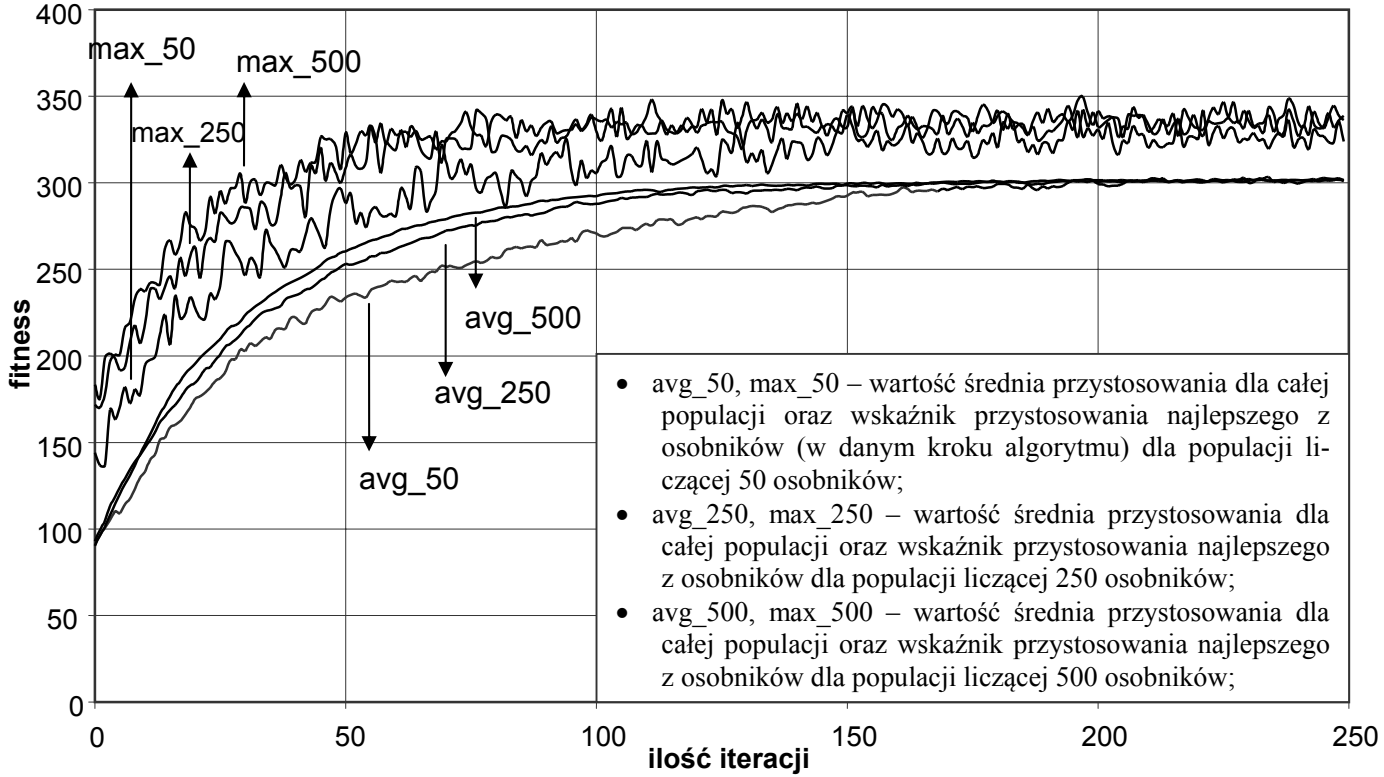
Tabela 1. Parametry algorytmu genetycznego dla omawianych wyników

Liczba osobników	Średni czas działania
50	≈ 2,1 s
250	≈ 8,04 s
500	≈ 14,98 s

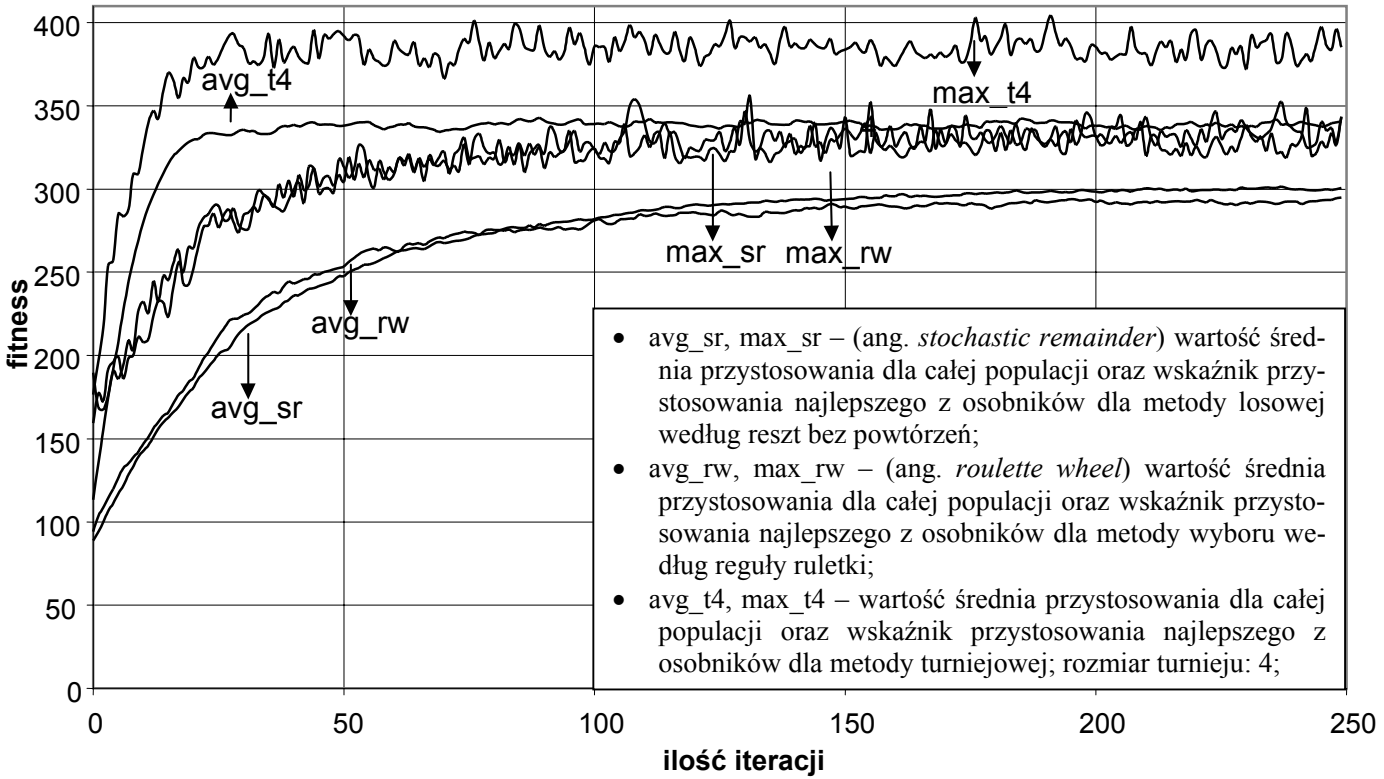
Tabela 2. Średni czas działania algorytmu w zależności od liczby osobników

Na wykresie numer dwa zaprezentowano wpływ metody selekcji na otrzymywane rezultaty. Potwierdza się to co zostało powiedziane wcześniej – najlepsze rezultaty otrzymuje się metodą turniejową. Powód jest prosty: metoda ta „pozbywa" się osobników o małym wskaźniku przystosowania, przez co wraz z kolejnymi iteracjami, osobniki podlegają już praktycznie tylko mutacji (jest to skrót myślowy: ponieważ większość populacji jest podobna (efekt turnieju), krzyżowanie prawie nic nie wnosi do poszukiwań). Jak wiemy z paragrafu o wariacjach, takie poszukiwania, zazwyczaj osiągają najwyższy wskaźnik przystosowania. Wszystkie pozostałe parametry są takie same jak dla algorytmu z wykresu numer jeden.

Wykres 1. Zmiana wskaźnika przystosowania w zależności od ilości osobników



Wykres 2. Zmiana wskaźnika przystosowania w zależności od metody wyboru



Literatura

[1] Biles J. A. „GenJam: A Genetic Algorithm for Generating Jazz Solos". *Proceedings of the 1994 International Computer Music Conference*, Aarhus, Denmark, 1994. International Computer Music Association. Praca dostępna pod adresem: ftp.cs.rit.edu/pub/jab/GenJam94.ps

[2] Goldberg D. E. „Algorytmy genetyczne i ich zastosowania". Wydawnictwa Naukowo-Techniczne, Warszawa 1998

[3] Holland, J. „Adaptation in Natural and Artificial Systems". Ann Arbor, Michigan: University of Michigan Press, 1975

[4] Horner A., Goldberg D. E. „Genetic Algorithms and Computer-Assisted Music Composition". *Proceedings of the Fourth International Conference on Genetic Algorithms*, Urbana-Champaign, Illinois, 1991.

[5] Horowitz D. „Generating Rhythms with Genetic Algorithms". *Proceedings of the 1994 International Computer Music Conference*, Aarhus, Denmark, 1994. International Computer Music Association

[6] Kopaliński W. „Słownik wyrazów obcych i zwrotów obcojęzycznych". Wiedza Powszechna, Warszawa 1983

[7] Papadopoulos G., Wiggins G. „A Genetic Algorithm for the Generation of Jazz Melodies". *Department of Artificial Intelligence, University of Edinburgh*. Praca dostępna pod adresem: www.dai.ed.ac.uk/daidb/people/homes/geraint/papers/STeP98.ps

[8] Rowe R. „Feature classification and related response in a real-time interactive music system". *The Journal of the Acoustical Society of America*, Suplement, 15 kwiecień, 1990. Abstrakt dostępny pod adresem: http://link.aip.org

[9] Todd P. M., Werner G. M. „Frankensteinian Methods for Evolutionary Music Composition". *Musical networks: Parallel distributed perception and performanc*e, Cambridge, MA: MIT Press/Bradford Books 1998. Praca dostępna pod adresem: www-abc.mpib-berlin.mpg.de/users/ptodd/publications/99evmus/99evmus.ps

[10] Yu C. „Computer Generated Music Composition". Praca napisana na wydziale Elektroniki i Informatyki na *Massachusetts Institute of Technology* w celu uzyskania stopni *Bachelor of Science in Computer Science and Engineering* oraz *Master of Engineering in Electrical Engineering and Computer Science*, 1996. Dokument dostępny pod adresem: http://lethe.media.mit.edu/online/net-music/net-instrument/LabeledSharle.htm