

Metaheurystyki

Notatki do wykładu

Kazimierz Grygiel

Część I.

Zadania optymalizacyjne Wiele zagadnień technicznych, ekonomicznych, logistycznych i innych problemów z którymi można się spotkać w codziennej praktyce ma postać *zadania optymalizacyjnego*.

W zadaniu optymalizacyjnym mamy do czynienia z pewnym zbiorem S obiektów, zwanych *rozwiązaniami* oraz *funkcją oceny* $f : S \rightarrow R$, która każdemu rozwiązaniu przypisuje wartość, będącą miarą jego *jakości*. Zadanie optymalizacyjne polega na wyznaczeniu rozwiązania *optymalnego*, czyli *najlepszego* pod względem jakości. (Jeśli takich rozwiązań jest więcej, niż jedno, na ogół wystarczy znaleźć dowolne z nich.) W zależności od tego, jak określimy jakość, zadanie sprowadza się do *maksymalizacji* lub *minimalizacji* funkcji oceny.

Umowa w sprawie oznaczeń Pojęcie “rozwiązanie optymalne” można zrelatywizować do dowolnego podzbioru $X \subset S$; podobnie można mówić o rozwiązaniach optymalnych w ciągu X rozwiązań z S . Przyjmijmy następującą konwencję dotyczącą oznaczeń:

- Dla dowolnego ciągu $X = (x_1, x_2, \dots)$ rozwiązań (skończonego lub nie) wyrażenie $\text{opt}(X)$ będzie oznaczać *pierwsze* (tzn. o najmniejszym indeksie) spośród rozwiązań optymalnych w ciągu X .
- Dla dowolnego podzbioru X rozwiązań wyrażenie $\text{opt}(X)$ będzie oznaczać *jedno* spośród rozwiązań optymalnych w podzbiorku X . Rozumiemy przez to, że metoda wyboru konkretnego rozwiązania nie jest z góry określona i powinna być dodatkowo wyspecyfikowana.

Metody heurystyczne Istnieje wiele wyspecjalizowanych metod matematycznych i algorytmów, służących do efektywnego rozwiązywania pewnych typów zadań optymalizacyjnych. Mówiąc nieformalnie, algorytm nazywamy *efektywnym*, jeżeli koszt jego zastosowania (mierzony czasem wykonania i wielkością zajętej pamięci) nie rośnie zbyt szybko w zależności od *rozmiaru* zadania. Rozmiar zadania to umownie wielkość pamięci potrzebnej do zapisania danych wejściowych dla zadania w pewnej standardowej notacji.

Dla wielu ważnych z praktycznego punktu widzenia typów zadań optymalizacyjnych (np. dla zadania komiwojażera) nie znaleziono efektywnych algorytmów, a perspektywa ich odkrycia w przyszłości wydaje się wątpliwa. Oznacza to, że istnieje praktyczna “bariera rozmiaru” przy rozwiązywaniu takich zadań za pomocą tradycyjnych algorytmów.

Alternatywą tradycyjnych algorytmów są *metody heurystyczne*. W potocznym znaczeniu terminem *heurystyka* (z greckiego *heuriskein* — znaleźć, odkryć) określa się praktyczną, opartą na doświadczeniu, "inteligentną" regułę postępowania, która może drastycznie uprościć lub skrócić proces rozwiązywania problemu, gdy metoda rozwiązania nie jest znana lub jest zawiła i czasochłonna. W algorytmice heurystyką nazywa się "niepełnowartościowy" algorytm, który umożliwia znalezienie w akceptowalnym czasie przynajmniej "dostatecznie dobrego" przybliżonego rozwiązania problemu, choć nie gwarantuje tego *we wszystkich* przypadkach.

Podejście heurystyczne można stosować "piętrowo", tworząc *metaheurystyki*, czyli heurystyki nadrzędne, sterujące w procesie iteracyjnego przeszukiwania heurystykami niższego rzędu. W ostatnich kilkudziesięciu latach opracowano wiele niekonwencjonalnych metod heurystycznego przeszukiwania, inspirowanych często mechanizmami fizycznymi lub biologicznymi, zaliczanych obecnie do rzędu metaheurystyk.

Przeszukiwanie heurystyczne Metody heurystyczne, którymi się tutaj zajmujemy, wywodzą się w pewnym sensie z algorytmu *pełnego przeglądu*. Algorytm ten ma zastosowanie w zadaniach optymalizacyjnych o skończonym zbiorze rozwiązań i polega po prostu na sprawdzeniu wszystkich możliwych przypadków. Strukturalnie, algorytm pełnego przeglądu składa się z dwóch, współdziałających ze sobą modułów. Jeden to *generator rozwiązań*, służący do produkcji kolejnych elementów zbioru S , drugi to *ewaluator*, obliczający wartość funkcji oceny dla bieżącego rozwiązania. Jakość bieżącego rozwiązania jest porównywana z jakością najlepszego znalezionego do tej pory rozwiązania, i w razie potrzeby następuje uaktualnienie tego ostatniego. W zapisie symbolicznym

$$x_{\text{opt}}(k) = \text{opt}\{x_1, x_2, \dots, x_k\}$$

Algorytm pełnego przeglądu znajduje rozwiązanie optymalne w czasie liniowym ze względu na liczbę rozwiązań, gdyż dopiero dla $k = |S|$ można mieć pewność, że $x_{\text{opt}}(k)$ jest rozwiązaniem optymalnym. (Chociaż może się zdarzyć, że algorytm "natrafił" na rozwiązanie optymalne w niewielu początkowych krokach, na ogół nie potrafimy tego stwierdzić.) Jeśli liczba rozwiązań rośnie wykładniczo wraz z rozmiarem zadania, to algorytm pełnego przeglądu przestaje mieć praktyczne zastosowanie.

Osobliwą cechą algorytmu pełnego przeglądu jest to, że można go przerwać po dowolnym kroku, a mimo to uzyskać *jakiś* rozwiązanie. Gdybyśmy zrezygnowali z pewności na rzecz efektywności, moglibyśmy wprowadzić do tego algorytmu *warunek zatrzymania*, po spełnieniu którego przerwamy dalsze poszukiwania i jako wynik podajemy najlepsze znalezione rozwiązanie ($x_{\text{opt}}(n)$, gdzie n — liczba wykonanych kroków). W ten sposób otrzymalibyśmy pewną prymitywną heurystykę, którą można uznać za prototyp bardziej zaawansowanych metod przeszukiwania heurystycznego.

Zasadnicza idea przeszukiwania heurystycznego polega na tym, aby w powyższym schemacie zastąpić jednolity, globalny mechanizm pełnego przeglądu dwustopniowym mechanizmem iterowanego przeszukiwania lokalnego. Proces przeszukiwania jest sterowany nie przez globalny algorytm indeksujący rozwiązania, lecz przez (ograniczoną) pamięć, gromadzącą informację o wcześniejszym przebiegu tegoż procesu. Rodzaj i zakres gromadzonej informacji zależy od wybranej metody, w każdym jednak przypadku obejmuje ona pewien podzbiór (co najmniej jednoelementowy) rozwiązań *bieżących*. W każdym cyklu iteracji heurystyczny generator rozwiązań

na podstawie tej informacji wytwarza pewną, stosunkowo niewielką liczbę nowych rozwiązań-kandydatów. Ewaluator oblicza wartości funkcji oceny dla kandydatów. Na tej podstawie inny mechanizm heurystyczny uaktualnia informację przechowywaną w pamięci (w szczególności podzbiór rozwiązań bieżących), po czym przechodzi się do następnego cyklu. Każdy cykl można interpretować jako przeszukiwanie lokalne w “sąsiedztwie” zbioru rozwiązań bieżących. Heurystyki są tak dobrane, aby faworyzować jak najlepsze rozwiązania bądź na etapie generowania kandydatów, bądź na etapie uaktualniania. Dzięki temu można mieć nadzieję, że proces przeszukiwania będzie — mówiąc obrazowo — szybko zmierzał w kierunku obszaru przestrzeni rozwiązań zawierającego rozwiązanie o wysokiej jakości. Proces zostaje przerwany po spełnieniu warunku zatrzymania (najczęściej jest nim przekroczenie limitu iteracji), a za wynik końcowy przyjmuje się najlepsze (jedno z najlepszych) rozwiązanie bieżące. Ten sposób działania eliminuje badanie znakomitej większości rozwiązań, istnieje jednak ryzyko, że pominięte zostaną przy tym także rozwiązania optymalne. Przeszukiwanie heurystyczne opiera się na istnieniu pewnych wewnętrznych, trudnych do precyzyjnego zdefiniowania regularności funkcji oceny, a jego skuteczność zależy od stopnia, w jakim dana funkcja spełnia owe ukryte założenia.

Podsumowując: w procesie przeszukiwania heurystycznego tworzony jest ciąg podzbiorów rozwiązań bieżących $\{S_k, k \geq 0\}$, któremu odpowiada ciąg najlepszych rozwiązań bieżących $\{x_{\text{opt}}(k) = \text{opt}(S_k), k \geq 0\}$.

Klasyfikacja heurystyk przeszukiwania Heurystyki przeszukiwania można klasyfikować na różne sposoby. Jeden z podziałów opiera się na liczebności podzbioru rozwiązań bieżących. Heurystyki z jednym rozwiązaniem bieżącym będziemy nazywać *heurystykami samotnego poszukiwacza*, a heurystyki z wieloma rozwiązaniami bieżącymi — *heurystykami populacyjnymi*. Należy tu wyjaśnić, że pojęcie populacji nie jest tożsame z teoriomnogościowym pojęciem zbioru i oznacza “zbiór z powtórzeniami”, tj. kolekcję, której elementy mogą się powtarzać (występować wielokrotnie). Nie zakłada się przy tym, że elementy populacji są uporządkowane, jak w przypadku ciągu. W większości heurystyk populacyjnych wielkość populacji pozostaje stała (choć liczebność odpowiadających im podzbiorów może się zmieniać). Podział ten jest istotny gdyż w heurystykach populacyjnych istnieje możliwość użycia dodatkowych mechanizmów generowania rozwiązań, które nie są dostępne w przypadku heurystyk samotnego poszukiwacza.

Innym ważnym kryterium klasyfikacyjnym jest stosowanie generatora liczb pseudolosowych do generowania lub uaktualniania rozwiązań. W tym przypadku heurystyki dzielimy na *deterministyczne* i *zrandomizowane (losowe)*.

Przeszukiwanie lokalne Istnieje wiele schematów przeszukiwania lokalnego. Wszystkie opierają się na koncepcji “odległości” lub “podobieństwa” rozwiązań, wynikającej bądź z naturalnej, bądź ze sztucznie nałożonej struktury tychże rozwiązań. Na przykład, w zadaniach optymalizacji kombinatorycznej rozwiązania są obiektami kombinatorycznymi i jako takie mają określoną strukturę, którą można wykorzystać w przeszukiwaniu lokalnym. Natomiast w przypadku optymalizacji numerycznej można otrzymać odpowiednią strukturę poprzez *dyskretyzację* rozwiązań, co wiąże się zastosowaniem pewnej skończonej reprezentacji stałopozycyjnej. Jeśli natomiast zastosowano reprezentację zmiennopozycyjną, to można wykorzystać “naturalną” odległość euklidesową. Istota pomysłu polega na zróżnicowaniu rozwiązań ze względu na odległość lub podobieństwo do rozwiązań bieżących. W schematach deterministycznych nowe rozwiązania wybiera się z pod-

zbioru rozwiązań najbliższych (najbardziej podobnych). W schematach zrandomizowanych nowe rozwiązania losuje się zgodnie z pewnym rozkładem prawdopodobieństwa, który zdecydowanie faworyzuje rozwiązania najbliższe (najbardziej podobne).

W praktyce do wytworzenia rozwiązań-kandydatów używa się ustalonego zestawu *transformacji* rozwiązań bieżących. W heurystykach samotnego poszukiwacza są to jednoargumentowe *transformacje lokalne*, zwane najczęściej *mutacjami*. W heurystykach populacyjnych oprócz transformacji lokalnych można używać dwu- lub wieloargumentowych *transformacji mieszających*, znanych też pod nazwą *rekombinacji* lub *krzyżowań*.

Heurystyki lokalnych ulepszeń W heurystykach samotnego poszukiwacza dla każdego rozwiązania określa się *zbiór rozwiązań sąsiednich*, krócej: *sąsiedztwo*. Mając dany zestaw transformacji lokalnych $\Phi = \{\phi_i : S \rightarrow S, i \in I\}$ definiujemy sąsiedztwo elementu $x \in S$ jako

$$N(x) = \{\phi_i(x), i \in I\}$$

Algorytm 1 *Deterministyczna heurystyka lokalnych ulepszeń*

utwórz rozwiązanie początkowe $x \in S$

while not BRAK_POPRAWY **do**

$x := RS(x, N(x))$

end while

return(x)

Jednym z najprostszych wariantów heurystyki samotnego poszukiwacza jest deterministyczna heurystyka *lokalnych ulepszeń*. Jej schemat podany jest w Algorytmie 1, przy czym $RS(x, N(x))$ to *reguła selekcji*, którą zależnie od wariantu należy interpretować następująco:

(SAHC) $x := \text{opt}(x, \text{opt}(N(x)))$

(NAHC) Niech $N(x) = (s_1, s_2, \dots, s_k)$. Szukamy pierwszego indeksu i ($1 \leq i \leq k$) takiego, że rozwiązanie s_i jest lepsze od x i zastępujemy x przez s_i . Jeśli taki indeks nie istnieje, pozostawiamy x bez zmiany.

Jak łatwo zauważyć, Algorytm 1 kończy się po wytworzeniu rozwiązania, w którego sąsiedztwie nie ma już lepszych rozwiązań. Takie rozwiązanie nazywamy *optimum lokalnym*. Optimum globalne jest oczywiście także optimum lokalnym, ale nie każde optimum lokalne jest optimum globalnym. Wynika stąd, że Algorytm 1 znajduje rozwiązanie będące *pewnym* optimum lokalnym. Używając metafory wspinaczki, jest to szczyt, na który dostajemy się idąc wyłącznie pod górę; z tego względu heurystyki lokalnych ulepszeń nazywane są też *algorytmami wspinaczkowymi* (ang. *hill-climbing*). Wynik działania takiego algorytmu zależy od miejsca startu i strategii wspinaczki. Optima lokalne mają charakter “pasożytniczy”, są bowiem konsekwencją zawężenia zakresu przeszukiwania do sąsiedztwa bieżącego elementu. Ogólnie, im “węższe” sąsiedztwo, tym więcej rozwiązań może okazać się optimami lokalnymi.

Nietrudno zauważyć, że dla danego algorytmu deterministycznego przestrzeń wszystkich rozwiązań rozpada się na rozłączne klasy złożone z rozwiązań początkowych związanych z tym samym optimum lokalnym stanowiącym rozwiązanie końcowe. Klasy te zwane są *obszarami przyciągania* danego optimum lokalnego.

Wariant zrandomizowany heurystyki lokalnych ulepszeń można przedstawić za pomocą schematu

Algorytm 2 *Zrandomizowana heurystyka lokalnych ulepszeń*

wylosuj rozwiązanie początkowe $x \in S$

while not STOP **do**

$x := \text{opt}(x, \text{losuj}(N(x)))$

end while

return(x)

przy czym warunek STOP określa zewnętrzne kryterium zatrzymania (np. przekroczenie limitu iteracji lub limitu iteracji bez poprawy rozwiązania), natomiast funkcja *losuj* generuje rozwiązanie z zadanego zbioru zgodnie z pewnym rozkładem prawdopodobieństwa (tutaj niewyspecyfikowanym); w najprostszym przypadku jest to rozkład jednostajny.

W przypadku algorytmu zrandomizowanego nie można już mówić o obszarach przyciągania, gdyż rozwiązanie końcowe jest zmienną losową (nawet przy ustalonym rozwiązaniu początkowym). Jeśli liczba iteracji jest dostatecznie duża, to otrzymane rozwiązanie końcowe jest z dużym prawdopodobieństwem pewnym optimum lokalnym. Również zrandomizowane heurystyki lokalnych ulepszeń nie gwarantują znalezienia optimum globalnego, gdyż każde napotkane optimum lokalne stanowi dla nich “pułapkę bez wyjścia”.

Na koniec zauważmy, że ciąg rozwiązań bieżących tworzonych przez heurystykę lokalnych ulepszeń (zarówno w wariantcie deterministycznym, jak zrandomizowanym) jest ciągiem *ściśle monotonicznym* ze względu na funkcję oceny. Dzięki temu nie ma potrzeby zapamiętywania historii przeszukiwania, w obawie przed wielokrotnym powtarzaniem się tych samych rozwiązań. Ujemną konsekwencją tego podejścia jest natomiast omawiane już wyżej niebezpieczeństwo wpadnięcia w pułapkę optimum lokalnego. Heurystyki generujące ściśle monotoniczny ciąg rozwiązań bieżących będziemy dalej nazywać *heurystykami monotonicznymi*.

Przykłady struktur sąsiedztwa

Wektory zerojedynkowe W przypadku rozwiązań postaci

$$x = (x_1, x_2, \dots, x_m)$$

gdzie $x_j \in \{0, 1\}$ dla $j = 1, 2, \dots, m$ elementarne transformacje lokalne polegają na zmianie wartości jednej ze współrzędnych wektora, tzn.

$$\phi_j(x_1, \dots, x_j, \dots, x_m) = (x_1, \dots, 1 - x_j, \dots, x_m)$$

dla $j = 1, 2, \dots, m$.

Będziemy je nazywać *mutacjami punktowymi*. Sąsiedztwo wyznaczone przez mutacje punktowe składa się ze wszystkich wektorów różniących się od danego wektora x na dokładnie jednej współrzędnej, tzn. odległych o 1 w sensie metryki Hamminga:

$$\rho_H(x, y) = \sum_{j=1}^m |x_j - y_j|$$

Permutacje Dla rozwiązań będących permutacjami zbioru $\{1, 2, \dots, m\}$ najprostsze transformacje lokalne to transpozycje dwóch elementów: $\phi_{i,j} = [i, j]$ dla $i, j = 1, 2, \dots, m$, $i \neq j$. Transpozycja $[i, j]$ zamienia *pozycje elementów* i, j w permutacji $x = (x_1, x_2, \dots, x_m)$. W tym przypadku sąsiedztwo składa się z $\binom{m}{2}$ permutacji. Równoważny zestaw transformacji to transformacje zamieniające *elementy na pozycjach* k, l . (Nie należy mylić obu sposobów zamiany). Bardziej zaawansowanym rodzajem mutacji są transformacje *zamiany z odwróceniem*. Polegają one na odwróceniu całego segmentu permutacji między pozycjami k, l (włącznie z końcami).

Metoda wielostartu Prostym sposobem zwiększającym szanse znalezienia rozwiązania bliskiego optimum globalnemu przy zastosowaniu heurystyk lokalnych ulepszeń jest wielokrotne wykonanie podstawowego algorytmu dla różnych rozwiązań początkowych, przy czym jako wynik ostateczny przyjmuje się najlepsze rozwiązanie końcowe ze wszystkich wykonanych. Nazywamy to metodą *wielostartu*. Rozwiązania początkowe można generować metodą losową. Przy podejściu deterministycznym trzeba posłużyć się dodatkową heurystyką, generującą rozwiązania początkowe.

Uogólnione struktury sąsiedztwa Innym sposobem mogącym złagodzić trudności związane z występowaniem optimów lokalnych jest uelastycznienie koncepcji sąsiedztwa. Jeżeli w zbiorze rozwiązań mamy określoną metrykę ρ , to możemy określić hierarchię sąsiedztw: k -sąsiedztwem rozwiązania x nazwiemy zbiór wszystkich rozwiązań y takich, że $\rho(x, y) \leq k$. Dobierając odpowiednio wartość k możemy dowolnie rozszerzać “zasięg” sąsiedztwa; w skrajnym przypadku może ono objąć całą przestrzeń rozwiązań. Aby ograniczyć koszt obliczeniowy przeszukiwania tak rozszerzonego sąsiedztwa, trzeba stworzyć mechanizm zapewniający w pewien sposób uprzywilejowanie bliskiego sąsiedztwa. Najłatwiej zrealizować to w przypadku metod zrandomizowanych; wystarczy wtedy użyć prawdopodobieństwa malejącego wraz ze wzrostem odległości rozwiązań. Pokażemy to na dwóch przykładach.

Mutacja równoległa Mutację równoległą stosuje się do reprezentacji binarnych. Mechanizm ten realizuje losowe, niezależne próby mutowania *wszystkich* pozycji wektora $x = (x_1, x_2, \dots, x_m)$, ze stałym prawdopodobieństwem p ($0 < p < 1/2$) wystąpienia mutacji punktowej. Prawdopodobieństwo to nazywamy *punktowym prawdopodobieństwem mutacji*. Jeśli zdefiniujemy *maskę mutacyjną* jako wektor binarny, w którym 1 oznacza zajście mutacji punktowej na danej pozycji, a 0 — brak mutacji, to prawdopodobieństwo wystąpienia konkretnej maski złożonej z k jedynek i $m - k$ zer jest równe $p^k(1 - p)^{m-k}$. Wielkość ta maleje w tempie wykładniczym wraz

ze wzrostem k . Wartość oczekiwana liczby jedynek w masce (czyli liczby mutacji punktowych w zmutowanym rozwiązaniu) jest równa mp . Chcąc uzyskać średnio jedną mutację punktową na rozwiązanie należy zatem przyjąć $p = 1/m$.

Mechanizm mutacji równoległej może być stosowany w dowolnych heurystykach przeszukiwania działających na reprezentacjach binarnych. Jest on m. in. standardowym składnikiem *algorytmów genetycznych*.

Mutacja addytywna W przypadku reprezentacji zmiennopozycyjnych zastosowanie uogólnionych struktur sąsiedztwa wydaje się jedynym sensownym rozwiązaniem. W zasadzie jedynym używanym sposobem mutacji rozwiązań będących liczbami rzeczywistymi jest dodanie losowego zaburzenia. Transformacje lokalne tworzą zatem zbiór $\Phi = \{\phi_u, u \in U\}$, gdzie $\phi_u(x) = x + u$. Na zbiorze U zaburzeń określony jest pewien rozkład prawdopodobieństwa (typowym przykładem jest rozkład normalny o średniej 0 i odpowiednio dobranym odchyleniu standardowym). Jeśli rozwiązania są wektorami liczb rzeczywistych, to mutację addytywną stosuje się niezależnie do każdej składowej wektora (być może z innymi parametrami rozkładu).

Mechanizm mutacji addytywnej jest standardowym składnikiem *strategii ewolucyjnych*.

Heurystyki niemonotoniczne. Reguła Metropolis Jeszcze inne podejście do problemu optyimów lokalnych polega na rezygnacji z monotoniczności algorytmu. Można to uzyskać przez odpowiednią modyfikację reguły selekcji, dopuszczając w pewnym stopniu rozwiązania *gorsze* od rozwiązania bieżącego jako kolejne rozwiązanie bieżące. Odstępstwo to nie może być zbyt radykalne, gdyż w przeciwnym przypadku utracilibyśmy główny mechanizm ukierunkowujący prowadzenie poszukiwań ku rozwiązaniu optymalnemu.

Reguła selekcji Metropolis określona jest następująco: Niech x — rozwiązanie bieżące, y — rozwiązanie-kandydat. Jeśli y jest lepsze od x , to akceptuj y bezwarunkowo. W przeciwnym przypadku akceptuj y z prawdopodobieństwem $p = \exp(-|f(x) - f(y)|/T)$, gdzie $T > 0$ jest zadany parametrem rzeczywistym, zwanym ze względu na interpretację fizyczną *temperaturą*. Dobierając wartość temperatury możemy regulować “siłę” selekcji: dla $T \rightarrow 0$ otrzymujemy regułę selekcji z heurystyk monotonicznych ($p \rightarrow 0$), natomiast dla $T \rightarrow +\infty$ — regułę błędzenia losowego ($p \rightarrow 1$). Dla wartości pośrednich, przy ustalonym T prawdopodobieństwo akceptacji gorszego rozwiązania maleje w tempie wykładniczym wraz z różnicą jakości, tak więc reguła Metropolis spełnia postulaty sformułowane powyżej.

Heurystykę opartą na regule Metropolis można stosować samodzielnie lub w powiązaniu ze schematem wielostartu. Stanowi ona także “jądro” *symulowanego wyżarzania*.

Część II.

Symulowane wyżarzanie Metaheurystyka *symulowanego wyżarzania* (ang. *simulated annealing*) stanowi połączenie dwóch heurystyk: zrandomizowanej heurystyki lokalnego przeszukiwania z regułą selekcji Metropolis oraz tzw. *schematu chłodzenia*, regulującego dynamicznie temperaturę. Zgodnie z przyjętą klasyfikacją jest to heurystyka samotnego poszukiwacza. Jej schemat można przedstawić następująco:

gdzie $\tau(k)$ oznacza temperaturę w kroku k (wyznaczoną zgodnie ze schematem chłodzenia), a

Algorytm 3 *Symulowane wyżarzanie*

```

wylosuj rozwiązanie początkowe  $x \in S$ 
 $k := 0; T := \tau(0)$ 
while not STOP do
     $x := RM(T, x, losuj(N(x)))$ 
     $k := k + 1; T := \tau(k)$ 
end while
return( $x$ )

```

$RM(T, x, y)$ — regułę selekcji Metropolis’a dla temperatury T i rozwiązań x, y . Warunek zatrzymania STOP jest tu powiązany ze spadkiem temperatury poniżej pewnego progu krytycznego (bliskiego 0).

Algorytm symulowanego wyżarzania, wywodzący się z technologii stosowanych w metalurgii i wytopie szkła (stąd nazwa), był przedmiotem wielu badań teoretycznych i empirycznych. W szczególności próbowano określić cechy, zapewniające skuteczność schematu chłodzenia. W literaturze opisano trzy podstawowe typy schematów chłodzenia.

1. schemat logarytmiczny (Boltzmann’a): $\tau(k) \cong 1/(1 + \log(k))$
2. schemat liniowy (Cauchy’ego) $\tau(k) \cong 1/k$
3. schemat geometryczny $\tau(k) \cong a^k$, gdzie $0 < a < 1$

Symbol \cong oznacza tu asymptotyczną proporcjonalność.

Dowodzono, że przy pewnych założeniach schemat logarytmiczny (w przeciwieństwie do pozostałych) *gwarantuje* w dostatecznie długim czasie znalezienie optimum globalnego z prawdopodobieństwem 1. Wariant ten nie ma jednak zastosowania praktycznego, gdyż jest zbyt powolny. Badania empiryczne sugerują, że największą przydatność praktyczną ma schemat geometryczny (najszybszy). W każdym ze schematów trzeba odpowiednio dobrać temperaturę początkową.

Tabu search Metaheurystyka *tabu search* (“przeszukiwanie z tabu”) jest algorytmem samotnego poszukiwacza, stanowiącym niemonotoniczne rozwinięcie deterministycznej heurystyki lokalnych ulepszeń. “Blokadę” wywoływaną przez optima lokalne przełamuje się tu dzięki osłabieniu reguły selekcji. Rozwiązanie bieżące jest *zawsze* zastępowane przez *najlepsze rozwiązanie w sąsiedztwie*, nawet jeśli powoduje to pogorszenie jakości. Aby przeciwdziałać możliwości powtarzania się tych samych rozwiązań bieżących (a więc powstawaniu cykli), zastosowano tu metodę *dynamicznego sąsiedztwa*. Konceptyjnie, polega to na “okrojeniu” zdefiniowanego w zwykły sposób sąsiedztwa poprzez usunięcie z niego rozwiązań, które już wcześniej były zaakceptowane jako rozwiązania bieżące. Te ostatnie tworzą *zbiór tabu*. Można to symbolicznie zapisać następująco: $N_d(x) = N(x) \setminus S_T$, gdzie $N_d(\cdot)$ — sąsiedztwo dynamiczne, S_T — zbiór tabu. W rzeczywistości mamy do czynienia z dwójakiemu rodzaju odstępstwami od opisanej zasady:

- zamiast samych rozwiązań, zakazem obejmuje się transformacje (zwane tu *ruchami*) zastosowane do wytworzenia tych rozwiązań (ze względów efektywnościowych)
- zakaz powtórzenia wyboru ruchu ma charakter czasowy

Tak więc w praktycznych implementacjach zbiór (zwany niekiedy listą) tabu składa się z ruchów wykonanych w ostatnich T krokach, gdzie T to *czas ważności tabu*. Ponieważ przeniesienie zakazu z rozwiązań na transformacje może w konsekwencji uniemożliwiać akceptację “legalnych” rozwiązań (które jeszcze się nie pojawiły), wprowadzono dodatkową regułę selekcji opartą na tzw. *kryteriach aspiracji*, spełniającą rolę “wyjątku od reguły”. Reguła ta określa, że *zakazany ruch, spełniający pewne kryterium aspiracji, jest ruchem dozwolonym*. Podstawowym kryterium aspiracji jest wytworzenie w wyniku zastosowania danego ruchu rozwiązania lepszego niż bieżące. Inne kryterium aspiracji może być związane z pustością dynamicznego sąsiedztwa.

Do realizacji zbioru tabu używa się tzw. *pamięci krótkoterminowej*. W metodzie tabu search wykorzystuje się również *pamięć długoterminową*, gromadzącą różne informacje statystyczne o przebiegu procesu przeszukiwania, jak np. częstość wykonania “zwycięskich” ruchów. Informacje te mogą być użyte w celu *intensyfikacji* lub *dywersyfikacji* procesu przeszukiwania. Intensyfikacja polega na “zagęszczeniu” próbkowania pewnego obszaru przestrzeni, natomiast dywersyfikacja — na rozproszeniu próbkowania w taki sposób, by żaden istotny obszar nie został pominięty. Jedną z technik opartych na pamięci długoterminowej jest *modyfikacja funkcji oceny*. Przykładem może być dodawanie do oryginalnej funkcji oceny członu “kary” będącego funkcją częstości użycia odpowiedniego ruchu.

Metoda tabu search obfituje w rozmaitego rodzaju rozszerzenia i modyfikacje i mogłaby stanowić przedmiot osobnego wykładu. Poniżej podany schemat ma więc bardzo “zgrubny” charakter. W szczególności reguła selekcji stosowana w tej metodzie może być sama zaawansowaną strategią heurystyczną. Przyjęto następujące oznaczenia: T — zbiór tabu, RT — reguła selekcji dla tabu search (uwzględniająca kryteria aspiracji), $N_d(\cdot, \cdot)$ — sąsiedztwo dynamiczne, zależne od zbioru tabu. W algorytmie zaniedbano aspekty związane z pamięcią długoterminową.

Algorytm 4 *Tabu search*

```

utwórz rozwiązanie początkowe  $x \in S$ 
 $T := \emptyset$ 
while not STOP do
     $x := RT(x, N_d(x, T))$ 
    uaktualnij  $T$ 
end while
return( $x$ )

```

Część III.

Heurystyki populacyjne Heurystyki populacyjne opierają się na idei przeszukiwania zespołowego. Twórcy tego rodzaju algorytmów czerpali inspirację głównie z mechanizmów rządzących procesem ewolucji biologicznej, stąd powszechnie przyjęta terminologia została zapożyczona z genetyki i biologii populacyjnej. Sama nazwa “populacja” wywodzi się z terminologii biologicznej. Zamiast o rozwiązaniach, mówimy często o *osobnikach*. Dalsze przykłady zostaną podane przy omawianiu poszczególnych metod.

Heurystyki populacyjne są algorytmicznie bardziej skomplikowane niż heurystyki samotnego po-

szukiwacza. Jest to spowodowane z jednej strony koniecznością zarządzania przemianą populacji, a z drugiej zastosowaniem dodatkowego mechanizmu generowania rozwiązań-kandydatów.

Mechanizm selekcji w heurystykach populacyjnych W heurystykach samotnego poszukiwacza reguła selekcji określa *czy* lub *które* rozwiązanie-kandydat zastępuje rozwiązanie bieżące. Wszyscy kandydaci są tworzeni za pomocą transformacji lokalnych z *jednego* rozwiązania bieżącego. Ponieważ w pojedynczym cyklu heurystyki populacyjnej tworzy się populację rozwiązań bieżących liczącą więcej niż jeden element, zarządzanie przemianą populacji wymaga odpowiedzi na następujące pytania:

- jak wybierać rozwiązania bieżące (*osobniki rodzicielskie*), z których będą tworzone rozwiązania-kandydaci (*osobniki potomne*)?
- jak decydować, które rozwiązania wejdą do następnego *pokolenia* rozwiązań bieżących?

Sposób rozwiązania powyższych kwestii będziemy nazywać odpowiednio *selekcją wstępną* i *selekcją końcową*. (Niektórzy autorzy używają terminów *reprodukcja* i *sukcesja*).

Jednym z podstawowych zagadnień związanych z selekcją jest zakres produkcji i wymiany osobników. W znakomitej większości heurystyk populacyjnych wielkość populacji pozostaje stała. Przy tworzeniu następnego pokolenia zachodzi konieczność usuwania części lub wszystkich osobników wchodzących w skład bieżącego pokolenia. Możliwe są następujące podejścia:

- Pełna wymiana pokoleń: całe pokolenie bieżące “wymiera”, a jego miejsce zajmują osobniki potomne. Wymaga to wyprodukowania co najmniej tylu osobników potomnych, ile wynosi wielkość populacji.
- Konkurencja międzypokoleniowa: skład następnego pokolenia dobiera się spośród bieżącego pokolenia i osobników potomnych. W tym przypadku liczba osobników potomnych może być mniejsza, w skrajnym przypadku wystarczy jeden osobnik potomny (metoda *steady-state*).

Selekcja wstępna: mechanizm ruletki Metody stosowane w selekcji wstępnej wygodnie opisać w terminach *mechanizmu ruletki*. Każdemu osobnikowi x z populacji bieżącej przypisuje się *prawdopodobieństwo reprodukcji* p_x (w sposób zależny od konkretnej metody selekcji). Koło ruletki o obwodzie 1 dzieli się na sektory o długości łuku p_x . W celu wylosowania pojedynczego osobnika uruchamia się koło i czeka na jego zatrzymanie. Umocowany na zewnątrz koła wskaźnik ruletki pokazuje wówczas sektor, odpowiadający zwycięzcy losowania. Postępowanie to powtarza się tyle razy, ile osobników należy wylosować. Wylosowane w ten sposób osobniki tworzą *pulę rodzicielską*, i są następnie poddawane transformacjom służącym do wytworzenia osobników potomnych, czyli rozwiązań-kandydatów. Z opisu procedury losowania wynika, że prawdopodobieństwo wylosowania osobnika x (a właściwie konkretnej kopii osobnika) w pojedynczej próbie jest równe p_x . Przypadek, gdy wszystkie prawdopodobieństwa reprodukcji są równe odpowiada tzw. *dryfowi losowemu*.

Uwagi: Kolejność, w jakiej rozmieszczone są sektory jest statystycznie obojętna. Wszystkie kopie danego osobnika x możemy umieścić obok siebie i przyporządkować im sektor o łuku $n_x p_x$, gdzie n_x — liczba kopii osobnika x w bieżącej populacji. Wielkość $n_x p_x$ jest prawdopodobieństwem

reprodukcji *dowolnej* kopii tego osobnika.

Pewną odmianę opisanego mechanizmu stanowi ruletka *wielowskaźnikowa*, która zamiast pojedynczego wskaźnika ma N wskaźników rozmieszczonych w jednakowych odstępach wokół koła ruletki. Taką ruletkę uruchamia się tylko raz, wyłaniając od razu N zwycięzców losowania. Mechanizm ten (znany pod nazwą *stochastic universal sampling*) ma inne właściwości statystyczne i służy redukcji fluktuacji losowych przy wyborze osobników rodzicielskich. W tym wariacie kolejność rozmieszczenia sektorów *nie jest* statystycznie obojętna, więc powinna być ustalana również losowo.

Selekcja wstępna preferująca osobniki lepsze jakościowo jest stosowana w *algorytmach genetycznych*. Selekcja wstępna neutralna ze względu na jakość (dryf losowy) jest stosowana w *strategiach ewolucyjnych*. W *przeszukiwaniu rozproszonym* stosuje się specyficzną, deterministyczną metodę selekcji wstępnej, która nie wyraża się poprzez mechanizm ruletki.

Przykłady metod selekcji wstępnej

1. **Selekcja proporcjonalna.** Ten sposób selekcji jest wyznaczony zależnością

$$p_x = \frac{f(x)}{\sum_{y \in P} f(y)}$$

gdzie P jest populacją bieżącą, a sumowanie przebiega po elementach populacji, czyli kopiach osobników. O funkcji celu f zakłada się, że jest dodatnia. Ten sposób selekcji preferuje osobniki o ponadprzeciętnej wartości funkcji oceny (którą w tym kontekście przyjęto nazywać *funkcją dostosowania*). Selekcja proporcjonalna może być więc stosowana bezpośrednio w zadaniach *maksymalizacji*. W przypadku zadań *minimalizacji* trzeba uprzednio przekształcić w odpowiedni sposób funkcję oceny, aby otrzymać właściwą funkcję dostosowania.

Średnia liczba potomków σ_x danej kopii osobnika x przy N -krotnym losowaniu jest równa Np_x . Jeśli N jest wielkością populacji, to tę zależność można zapisać w postaci $\sigma_x = f(x)/\bar{f}(P)$, gdzie $\bar{f}(P) = \sum_{y \in P} f(y)/N$ jest *średnim dostosowaniem populacji P* . Wielkość σ_x nazywamy też *współczynnikiem reprodukcji* osobnika x . Określa on średnią “wydajność reprodukcyjną” osobnika w obrębie danej populacji.

Selekcja proporcjonalna była początkowo silnie lansowana przez twórców algorytmów genetycznych, gdyż ma bezpośrednie odniesienia do koncepcji doboru naturalnego w teorii ewolucji. Jednak w praktyce nie okazała się zbyt wydajna. Jedną z podstawowych jej wad jest jej wrażliwość na *wględne* różnice dostosowania. Jeśli w populacji dojdzie do spłaszczenia tych różnic, selekcja proporcjonalna degeneruje się do dryfu losowego. Można temu przeciwdziałać poprzez tzw. *skalowanie* funkcji dostosowania.

2. **Selekcja liniowa wg rang** W prostszym wariacie tej metody selekcji prawdopodobieństwa reprodukcji określa się następująco: sortujemy elementy populacji wg jakości od najgorszego do najlepszego i nadajemy im zgodne z tym porządkiem numery, zwane *rangami* (największa ranga przypada najlepszemu elementowi). Prawdopodobieństwo reprodukcji osobnika x jest równe

$$p_x = \frac{2r_x}{N(N+1)}$$

gdzie r_x jest rangą osobnika, a N - wielkością populacji.

W bardziej zaawansowanej wersji tej metody określamy dodatkowo stałą c , $0 < c < 1$ i przyjmujemy

$$p_x = \frac{1}{N}(c + 2(1 - c)\frac{r_x - 1}{N - 1})$$

W obu wariantach różnica prawdopodobieństw reprodukcji dla osobników o sąsiednich rangach jest stała. Metoda selekcji liniowej wg rang jest więc dość radykalną próbą przezwyciężenia słabości selekcji proporcjonalnej. Parametr c służy do wyskalowania przyrostów prawdopodobieństwa. Ta metoda selekcji nadaje się w równym stopniu do zadań maksymalizacji, jak i minimalizacji.

3. **Selekcja turniejowa** Selekcja turniejowa nie odwołuje się do mechanizmu ruletki, chociaż daje się pośrednio wyrazić za jego pomocą. Procedura selekcji wygląda tu następująco: Z populacji bieżącej losujemy k elementów, gdzie k to *rozmiar turnieju*. Zależnie od wariantu, może to być losowanie z *powtórzeniami* lub *bez powtórzeń*. W obydwu przypadkach mamy do czynienia ze schematem urnowym (wszystkie “kule” w urnie mają jednakowe prawdopodobieństwo wyciągnięcia). Zwycięzcą *turnieju* zostaje najlepszy wylosowany osobnik (w razie niejednoznaczności stosuje się rozstrzyganie losowe). Postępowanie to kontynuuje się aż do wylosowania potrzebnej liczby osobników.

Prawdopodobieństwo reprodukcji osobnika x w jednej rundzie selekcji turniejowej można wyznaczyć z klasycznego schematu prawdopodobieństwa (znajdujemy liczbę wszystkich możliwych składów turnieju dla danej populacji i liczbę tych, w których x jest zwycięzcą; prawdopodobieństwo jest równe stosunkowi drugiej liczby do pierwszej).

Selekcja turniejowa jest obecnie jedną z najczęściej stosowanych metod selekcji wstępnej. Może być używana z równym powodzeniem w obu typach zadań optymalizacyjnych. Typowym rozmiarem turnieju jest $k = 2$.

Metody selekcji końcowej Potrzeba selekcji końcowej występuje w heurystykach z konkurencją międzypokoleniową lub z nadmiarowym potomstwem. Ograniczymy się tu do przedstawienia najprostszej, deterministycznej procedury selekcji końcowej. Polega ona na posortowaniu wszystkich konkurujących osobników wg jakości i akceptacji pierwszych N najlepszych, gdzie N jest wielkością populacji. Ten rodzaj selekcji jest stosowany w *strategiach ewolucyjnych*, a także, z pewną modyfikacją — w *przeszukiwaniu rozproszonym*.

Transformacje mieszające Transformacje mieszające stanowią dodatkowy aparat do generowania rozwiązań-kandydatów, inspirowany w pewnej mierze przykładem powszechnego wśród organizmów wyższych rozmnażania płciowego. W rozmnażaniu płciowym w wyniku procesów *rekombinacji genetycznej* i zapłodnienia potomek dziedziczy geny pochodzące od obojga rodziców. Umożliwia to wytwarzanie nowych kombinacji genów, których łączny efekt ze względu na dostosowanie osobnika (fenotypu) może mieć charakter nieliniowy.

Typowym przykładem transformacji mieszających są dwuargumentowe operacje *krzyżowania* (ang. *crossover*), stosowane standardowo w algorytmach genetycznych. Mechanizm krzyżowania

naśladuje pewien subtelny proces wymiany genów zachodzący podczas mejozy, tzw. *crossing-over*. Na tym poziomie abstrakcji osobnika utożsamiamy z pojedynczym *chromosomem* (niekiedy z zestawem chromosomów, zwanym też *genotypem*). Elementarne (niepodzielne) składowe chromosomu to *geny*, które mogą występować w różnych odmianach, zwanych *allelami*. Chromosom ma zatem strukturę wektorową, a poszczególne pozycje tego wektora reprezentują geny. Operacje krzyżowania można w sposób naturalny zdefiniować dla obiektów o strukturze “genetycznej”, choć niektórzy autorzy używają tego terminu w stosunku do wszelkich transformacji mieszających, mających zastosowanie np. dla permutacji.

Krzyżowanie binarne Operacje krzyżowania binarnego określone są dla wektorów zerojedynkowych. W tym przypadku każdy gen ma dwa allele: 0 i 1. Każda operacja krzyżowania binarnego jest odwzorowaniem $\chi : S^2 \rightarrow S^2$, a więc składa się z dwóch składowych $\chi_1, \chi_2 : S^2 \rightarrow S$, gdzie S — przestrzeń m -wymiarowych wektorów zerojedynkowych. W wyniku zastosowania operacji krzyżowania binarnego do pary chromosomów x, y otrzymujemy parę chromosomów potomnych $x' = \chi_1(x, y)$, $y' = \chi_2(x, y)$. Najczęściej spotykane warianty krzyżowania binarnego to

1. **Krzyżowanie proste (jednopunktowe)** Polega na “przecięciu” obu argumentów między pozycjami k i $k + 1$ ($1 \leq k \leq m - 1$) i zamianie “ogonów”. Parametr k nazywa się *punktem krzyżowania*, jego wartość losuje się ze wskazanego zakresu (z jednakowym prawdopodobieństwem $1/(m - 1)$ dla każdej wartości). Formalnie: mamy rodzinę transformacji $X = \{\chi_k : S^2 \rightarrow S^2, k \in \{1, \dots, m - 1\}\}$, gdzie

$$\chi_{k,1}(x, y) = (x_1, \dots, x_k, y_{k+1}, \dots, y_m), \quad \chi_{k,2}(x, y) = (y_1, \dots, y_k, x_{k+1}, \dots, x_m)$$

przy czym na zbiorze indeksów zadany jest jednostajny rozkład prawdopodobieństwa.

2. **Krzyżowanie jednostajne** Dla każdej pozycji j , $j = 1, \dots, m$ decydujemy losowo, w sposób niezależny i z jednakowym prawdopodobieństwem, czy zamienić miejscami odpowiednie allele, czy pozostawić je bez zmian. Formalny opis jest analogiczny, jak w przypadku krzyżowania prostego, z tą różnicą, że transformacje są indeksowane wektorami zerojedynkowymi. Prawdopodobieństwo wylosowania każdej wartości indeksu jest więc równe $1/2^m$.

Oba powyższe schematy, a również każdy inny podobny schemat krzyżowania binarnego można wyrazić w ramach ogólnego modelu *krzyżowania wg maski*. *Maską krzyżowania* nazywamy m -wymiarowy wektor zerojedynkowy, w którym jedynka na pozycji j oznacza zamianę odpowiednich alleli w osobnikach rodzicielskich, a zero — pozostawienie ich bez zmian. Rozkład prawdopodobieństwa na zbiorze masek określa konkretny wariant krzyżowania binarnego.

Uwagi: Chociaż zdefiniowane powyżej operacje krzyżowania wytwarzają jednocześnie dwóch potomków (wariant 2×2), można stosować również wariant 2×1 , wybierając losowo jednego z nich.

Rekombinacja wektorów rzeczywistych Naturalnym odpowiednikiem krzyżowania binarnego jest tu tzw. *rekombinacja dyskretna*, polegająca na wymianie składowych, w analogiczny sposób, jak w przypadku wektorów zerojedynkowych. Praktyczny pożytek z tego typu mieszania

jest jednak raczej ograniczony. Dlatego wprowadzono innego rodzaju transformacje mieszające, zwane również (niezbyt poprawnie) rekombinacją lub krzyżowaniem, choć naruszają one zasadę niepodzielności genów. Podamy tu dwa przykłady takich transformacji. (Nazewnictwo nie jest w tym przypadku dobrze ustalone, więc poszczególni autorzy mogą używać odmiennych terminów).

1. **Rekombinacja uśredniająca** W operacji bierze udział $r \geq 2$ osobników rodzicielskich x_1, \dots, x_r , wynikiem jest pojedynczy potomek y o współrzędnych

$$y_j = \frac{1}{r} \sum_{i=1}^r x_{i,j}, \text{ dla } j = 1, \dots, m$$

2. **Krzyżowanie arytmetyczne** W operacji bierze udział dwóch osobników rodzicielskich x_1, x_2 , wynikiem jest pojedynczy potomek y o współrzędnych

$$y_j = u_j x_{1,j} + (1 - u_j) x_{2,j} \text{ dla } j = 1, \dots, m$$

przy czym współczynniki u_j są losowane niezależnie z przedziału $[0, 1]$ zgodnie z rozkładem jednostajnym.

W strategiach ewolucyjnych używa się podobnych, choć jeszcze bardziej wyrafinowanych schematów mieszania.

Transformacje mieszające dla obiektów kombinatorycznych Transformacje mieszające można określić także dla różnego rodzaju obiektów kombinatorycznych, choć nie widać tu żadnego naturalnego sposobu definiowania, gdyż nie wiadomo, jak określić pojęcie genu. W literaturze można spotkać wiele takich propozycji, na ogół związanych z konkretnymi zadaniami optymalizacji kombinatorycznej. Na przykład, w przypadku permutacji zaproponowano operacje zwane w skrócie PMX (krzyżowanie z częściowym dopasowaniem), OX (krzyżowanie porządkowe) i CX (krzyżowanie cykliczne). Szczegółowe opisy tych transformacji można znaleźć w literaturze uzupełniającej.

Pula genetyczna populacji Aby lepiej zrozumieć działanie operacji krzyżowania, wprowadzimy pojęcie *puli genetycznej* populacji. Mówiąc nieformalnie, pula genetyczna składa się ze wszystkich alleli, obecnych w populacji. Ponieważ operacje krzyżowania polegają na “przetasowaniu” alleli między osobnikami wchodzącymi w skład populacji (odnosi się to oczywiście do “prawdziwych” operacji krzyżowania, a nie innych transformacji mieszających, por. wcześniejsze uwagi), jest rzeczą oczywistą, że nie mogą one *wytworzyć* żadnego nowego allelu. Allele mogą być natomiast “zgubione” w wyniku działania selekcji lub innych efektów losowych (np. w przypadku stosowania wariantu 2×1). Tę właściwość krzyżowania można podsumować w postaci następującego twierdzenia: *Pula genetyczna populacji jest zamknięta ze względu za operacje krzyżowania*. Wypływa stąd wniosek, że aby zapobiec utracie alleli w procesie ewolucji populacji, należy stosować mutację, która jest mechanizmem umożliwiającym *wytwarzanie* alleli.

Część IV.

Algorytmy ewolucyjne Do *algorytmów ewolucyjnych* zaliczamy metaheurystyki populacyjne, oparte na paradygmacie ewolucji. Począwszy od lat sześćdziesiątych ubiegłego wieku w różnych ośrodkach naukowych na świecie powstawały niezależnie różne szkoły lansujące wypracowane przez siebie metodologie i algorytmy przeszukiwania heurystycznego, inspirowane mechanizmami ewolucji naturalnej. Na początku lat dziewięćdziesiątych przedstawiciele większości tych szkół spotkali się na wspólnej konferencji i uznali, że ich wspólnym przedmiotem zainteresowania są *obliczenia ewolucyjne* (ang. *evolutionary computation*). Od tej pory przyjęto określać wspólnym mianem “algorytmy ewolucyjne” oryginalnie opracowane metody i ich późniejsze modyfikacje, będące często wynikiem hybrydyzacji. Pierwotne podziały nie zanikły jednak całkowicie i do dziś utrzymują się spory i kontrowersje między przedstawicielami różnych kierunków. Najbardziej ogólny schemat algorytmu ewolucyjnego jest przedstawiony niżej jako Algorytm 5.

Algorytm 5 Szkielet algorytmu ewolucyjnego

```

utwórz populację początkową
while not STOP do
    utwórz pulę rodzicielską z populacji bieżącej
    wygeneruj osobniki potomne wykonując transformacje mieszające i lokalne w puli rodzicielskiej
    uaktualnij populację bieżącą
end while
return (najlepsze rozwiązanie w populacji bieżącej)
```

W praktycznych implementacjach etap tworzenia puli rodzicielskiej jest na ogół zintegrowany z generowaniem osobników potomnych, co czyni go w istocie etapem “wirtualnym”.

Do algorytmów ewolucyjnych zaliczamy w szczególności *algorytmy genetyczne* i *strategie ewolucyjne*. Niektórzy autorzy włączają tu także *przeszukiwanie rozproszone*, choć ta ostatnia metaheurystyka koncepcyjnie różni się od poprzednich w dość istotny sposób.

Algorytmy genetyczne Pierwotną wersję algorytmu genetycznego przyjęto obecnie nazywać *kanonicznym* lub *prostym algorytmem genetycznym*. Charakterystyczną cechą tej metaheurystyki jest binarne kodowanie rozwiązania jako pojedynczego chromosomu. Selekcja wstępna jest realizowana za pomocą mechanizmu selekcji proporcjonalnej (algorytm wymaga więc niekiedy odpowiedniego przekształcenia funkcji oceny). Wytwarzanie osobników potomnych jest realizowane przy użyciu krzyżowania prostego oraz następującej po nim mutacji równoległej, przy czym krzyżowanie jest uważane za mechanizm podstawowy. W algorytmie stosuje się pełną wymianę pokoleń, tak więc w każdym cyklu wytwarza się tylu osobników potomnych, ile wynosi wielkość populacji. Parametrami algorytmu są:

- wielkość populacji N
- prawdopodobieństwo krzyżowania p_c
- punktowe prawdopodobieństwo mutacji p_m

- limit liczby iteracji

Krzyżowanie jest wykonywane warunkowo. Parametr p_c jest prawdopodobieństwem, że dana para osobników rodzicielskich zostanie poddana krzyżowaniu. W efekcie krzyżowania (lub jego pominięcia) otrzymuje się — zależnie od wariantu — dwóch lub jednego osobnika potomnego. Każdy taki “pośredni” osobnik podlega następnie bezwarunkowej mutacji.

W literaturze opisano wiele ulepszonych wariantów tego podstawowego algorytmu.

Strategie ewolucyjne Strategie ewolucyjne zostały pomyślane jako narzędzie *optymalizacji parametrycznej*. Rozwiązania mają tu postać m -wymiarowych wektorów o współrzędnych rzeczywistych. Charakterystyczną cechą tej metaheurystyki jest *samoadaptacja parametrów sterujących*, które podlegają ewolucji wraz z właściwymi rozwiązaniami. Osiąga się to przez rozszerzenie reprezentacji osobnika, dołączając do wektora rozwiązania wektor parametrów sterujących. W prostszej wersji algorytmu parametrami sterującymi są odchylenia standardowe rozkładu normalnego, używane przy wykonywaniu mutacji addytywnej poszczególnych współrzędnych rozwiązania. W tym przypadku osobnik ma postać $(x_1, \dots, x_m, \sigma_1, \dots, \sigma_m)$. Tylko pierwszych m składowych, reprezentujących rozwiązanie określa jakość osobnika; pozostałe składowe, które pośrednio przyczyniły się do wytworzenia tego rozwiązania, poddane są wraz z nim działaniu selekcji.

Wielkość populacji μ jest parametrem algorytmu. Selekcja wstępna w zasadzie sprowadza się do dryfu losowego, tzn. osobniki rodzicielskie losowane są z jednakowym prawdopodobieństwem z populacji bieżącej. (Selekcja wstępna może przebiegać dwupoziomowo przy niektórych wariantach rekombinacji). Liczba λ wytworzonych osobników potomnych jest parametrem algorytmu. Wytwarzanie osobników potomnych jest realizowane za pomocą mechanizmów rekombinacji (opcjonalnie) i mutacji, która stanowi mechanizm podstawowy. Transformacje osobników przeprowadzane są dwuetapowo. W pierwszej fazie dokonuje się transformacji parametrów sterujących. W drugiej fazie zmodyfikowane parametry używane są do transformacji rozwiązań. W obu fazach mogą być stosowane odrębne mechanizmy. Do wytwarzania nowych rozwiązań używa się mutacji addytywnej z rozkładem normalnym odchyłek; każda współrzędna jest mutowana niezależnie, przy wykorzystaniu indywidualnego odchylenia standardowego, wytworzonego w pierwszej fazie. Szczegółowy opis tej procedury można znaleźć w literaturze uzupełniającej.

W strategiach ewolucyjnych używa się dwóch trybów selekcji końcowej, przy czym oba są deterministyczne i polegają na wyborze μ najlepszych osobników. Tryb $(\mu + \lambda)$ odpowiada selekcji z konkurencją międzypokoleniową, natomiast tryb (μ, λ) — pełnej wymianie pokoleń.

Przeszukiwanie rozproszone (scatter search) Jest to rozbudowana metaheurystyka deterministyczna, ze swoistą terminologią. Pojęciu populacji odpowiada tu *zbiór bazowy* (ang. *reference set*), złożony z dwóch podzbiorów; pierwszy zawiera rozwiązania *elitarne*, drugi — rozwiązania *różnicujące*. Algorytm wykorzystuje cały zestaw swoistych heurystyk. Są to:

- *generator różnorodności (diversification generator)*, służący do wytwarzania początkowego zbioru rozwiązań próbnych z zadanego *ziarna*
- *metoda ulepszania (improvement method)*, służąca do naprawy rozwiązań *niedopuszczalnych* i ulepszania rozwiązań metodą lokalnej optymalizacji

- *metoda aktualizacji zbioru bazowego (reference set update method)*, służąca jako metoda selekcji końcowej
- *metoda generowania podzbiorów (subset generation method)*, deterministyczna wersja selekcji wstępnej (poszczególne podzbiory to zestawy rozwiązań bieżących, będących argumentami transformacji mieszających)
- *metoda kombinacji rozwiązań (solution combination method)*, rodzina transformacji mieszających; jedyna składowa algorytmu, ściśle zależna od problemu źródłowego

W algorytmie wyróżnia się dwie fazy: faza początkowa służy do utworzenia pierwszego zbioru bazowego, faza iteracyjna obejmuje czynności zmierzające do aktualizacji zbioru bazowego i wykonywana jest wielokrotnie, aż do momentu, gdy nie udało się wprowadzić nowego elementu do zbioru bazowego. Wszystkie elementy zbioru bazowego są różne (duplikaty są eliminowane). Funkcję transformacji lokalnych pełni metoda ulepszania rozwiązań.

Szczegółowy opis algorytmu można znaleźć w materiałach pomocniczych.