Wstęp do inteligencji komputerowej – zajęcia nr 2

Jarosław Stańczak
WSISiZ

Zagadnienia związane ze złożonością obliczeniową problemów

- pojęcie deterministycznej i niedeterministycznej maszyny Turinga
- zadania P i NP.

Heurystyki w rozwiązywaniu problemów NP

- proste metody zachłanne
- symulowane wyżarzanie
- algorytmy ewolucyjne
- metoda tabu-search
- algorytmy mrówkowe.

Złożoność obliczeniowa problemów a sztuczna inteligencja

Złożoność obliczeniowa problemów jest bardzo istotnym zagadnieniem z wielu powodów. Przede wszystkim istnieje wiele problemów obliczeniowych, których nie potrafimy skutecznie rozwiązywać, a opisują one spotykane codziennie sytuacje (np. problem TSP, problem plecakowy), które chcielibyśmy umieć łatwo, szybko i dokładnie rozwiązywać.
Z drugiej strony są też problemy (np. szyfrowanie), które nie powinny być łatwo rozwiązywane i nie chcielibyśmy, żeby były. Oba te rodzaje problemów należą do tzw. klasy problemów NP.
Kłopoty z efektywnym rozwiązywaniem tego typu problemów metodami klasycznymi skłoniły badaczy do zainteresowania się metodami heurystycznymi, bazującymi na zagadnieniach związanych ze sztuczną inteligencją.

Złożoność obliczeniowa problemów

Przyjmuje się, że algorytm jest efektywny (czyli w uproszczeniu liczy się szybko), jeśli ma złożoność obliczeniową wielomianową, czyli liczba instrukcji maszynowych potrzebnych do jego wykonania rośnie wielomianowo ze wzrostem wielkości obliczanego problemu.
Analogicznie algorytm nie jest efektywny, jeśli jego złożoność jest ponadwielomianowa, czyli w praktyce najczęściej wykładnicza lub wyższa. Algorytmy o takiej złożoności nie pozwalają na dokładne obliczenie rozwiązania problemu już dla niedużych zadań.
Dokładnie rzecz biorąc, definiuje się złożoności obliczeniowe średnią, optymistyczną i pesymistyczną, które zależą od konkretnych danych – algorytm może mieć różne złożoności dla różnych danych.
Podobnie złożoność może dotyczyć wykorzystywanej pamięci – wtedy jest mowa o złożoności pamięciowej.

Złożoność obliczeniowa problemów

Do analizy algorytmów wykorzystuje się pojęcia deterministycznej i niedeterministycznej maszyny Turinga.

Deterministyczna maszyna Turinga to zdefiniowany przez A. Turinga abstrakcyjny model komputera. Model ten służy do wykonywania dowolnych algorytmów. Ten abstrakcyjny komputer składa się z nieskończenie długiej taśmy (jednostronnie lub dwustronnie) podzielonej na pola, w których zapisuje się dane. Każde pole może znajdować się w jednym z N stanów. Maszyna zawsze jest ustawiona nad jednym z pól i znajduje się w jednym z M stanów – wykonuje rozkaz. Efektem wykonania rozkazu, zależnie od kombinacji stanu maszyny i pola na taśmie, maszyna zapisuje nową wartość w polu, zmienia stan, a następnie może przesunąć się o jedno pole w prawo lub w lewo. Maszyna Turinga jest sterowana listą zawierającą dowolną liczbę rozkazów. Liczby N i M powinny być skończone. Czasem definiuje się wyróżniony stan M+1, który oznacza zakończenie pracy maszyny. Lista rozkazów na taśmie może być traktowana jako jej program. Komputery, które używamy, są pewnymi ograniczonymi przedstawicielami deterministycznych maszyn Turinga, gdyż ich programy są zawsze ograniczone, czyli ich taśma nie może być nieskończenie długa.

Złożoność obliczeniowa problemów

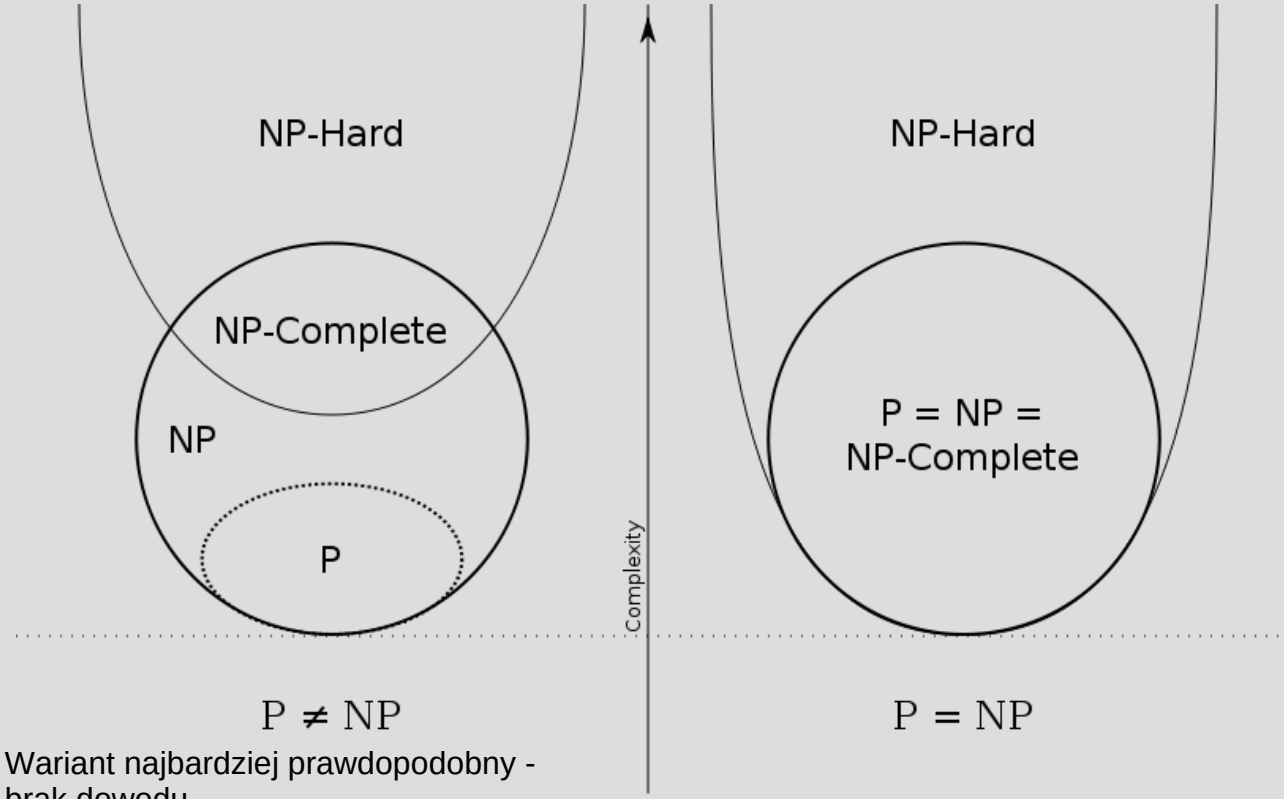
Niedeterministyczna maszyna Turinga niestety nie jest realizowalna przy obecnym stanie wiedzy. Jej działanie jest analogiczne do maszyny deterministycznej, opisanej poprzednio (maszyna deterministyczna jest szczególnym przypadkiem maszyny niedeterministycznej), różnica polega na tym, że maszyna może dowolnie tworzyć swoje kopie tak, aby jednocześnie przejść do wszystkich możliwych stanów następnych. W praktyce byłoby to urządzenie o dowolnie zwielokrotniającej się liczbie procesorów wraz z pamięcią i odpowiednimi peryferiami. Maszyna o tego typu właściwościach rzeczywiście mogłaby szybko rozwiązywać najtrudniejsze nawet problemy.
Systemy wieloprocesorowe są w jakimś stopniu przybliżeniem takiej maszyny, lecz oczywiście liczba dostępnych procesorów i innych zasobów jest zawsze ograniczona.

Złożoność obliczeniowa problemów -
klasy złożoności zadań

- P – (polynomial) wielomianowa – istnieją dla nich efektywne algorytmy obliczeniowe (choć przy wysokich potęgach też nie jest dobrze);
- NP – (non-deterministic polynomial) – wielomianowe na niedeterministycznej maszynie Turinga, najprawdopodobniej $P \neq NP$ (nie ma na to dowodu!) i wobec tego raczej nie istnieją tu efektywne algorytmy rozwiązujące;
- NP-zupełne (NPC) – dowolny problem z klasy NP może być do niego zredukowany w czasie wielomianowym;
- NP-trudne (NPH) - to taki problem obliczeniowy, którego rozwiązanie jest co najmniej tak trudne jak rozwiązanie każdego problemu z klasy NP.

Złożoność obliczeniowa cd.

możliwe relacje klas złożoności zadań



Wariant najbardziej prawdopodobny - brak dowodu.

[Rys. na podst. Wikipedii]

Złożoność obliczeniowa cd.

porównanie czasów obliczeń

Funkcja złożoności obliczeniowej	n=10	n=60
n	0,00000001 s	0,00000006 s
n^2	0,0000001 s	0,0000036 s
n^3	0,000001 s	0,000216 s
n^5	0,0001 s	0,78 s
2^n	0,000001 s	336,6 lat
3^n	0,000059 s	$1{,}3 \times 10^{12}$ lat
$n!$	0,00363 s	$2{,}63 \times 10^{65}$ lat
n^n	10 s	$1{,}55 \times 10^{90}$ lat

Czas trwania pojedynczej operacji wynosi 1 ns.

[Na podst. J. Błażewicz, „Złożoność obliczeniowa problemów kombinatorycznych”.]

Co to są i do czego mogą służyć algorytmy heurystyczne?

Heurystyka - ogólny algorytm (także pomysł, metoda, metaheurystyka) rozwiązywania problemów, najczęściej obliczeniowych i optymalizacyjnych oparty na pewnym spostrzeżeniu, pomyśle, które można wykorzystać do rozwiązania takiego problemu bez dokładnej wiedzy jak ten problem rozwiązać, niejako „przy okazji”. Algorytmu heurystycznego można używać do rozwiązywania dowolnego problemu, który można opisać za pomocą pewnych pojęć (symboli) zdefiniowanych na potrzeby danego rozwiązywanego problemu i budowanego dla niego algorytmu.

Algorytmy heurystyczne są ogólnymi metodami, umożliwiającymi rozwiązywanie problemów niemożliwych do rozwiązania metodami klasycznymi z uwagi na ich złożoność obliczeniową lub brak dedykowanych algorytmów służących do ich rozwiązania. Dzięki wykorzystaniu różnych heurystyk powstaje klasa algorytmów o bardzo szerokim spektrum zastosowań w: optymalizacji, rozpoznawaniu wzorców (obrazów, dźwięków,...), tzw. sztucznej inteligencji i in.

Cechy optymalizacji klasycznej i heurystycznej

Optymalizacja klasyczna (algorytmy dokładne i aproksymacyjne):

- wykorzystywane są metody numeryczne o udowodnionej zbieżności i znanych właściwościach (otrzymane rozwiązanie jest optymalne lub znamy oszacowanie dokładności rozwiązań przybliżonych);
- stosowalność tych metod jest często bardzo ograniczona do konkretnych klas zadań lub określonych ich właściwości (np. zadania liniowe, „ciągłe”, dyskretne, bez ograniczeń, różniczkowalne, itp.);
- czas lub rzadziej ilość pamięci potrzebna do uzyskania przy ich użyciu rozwiązania są często nieakceptowalnie duże.

Optymalizacja heurystyczna:

- najczęściej brak dowodu i oszacowania charakteru zbieżności;
- metody mają dość szerokie spektrum zastosowań (często trzeba je specjalizować);
- wyniki otrzymywane są dość szybko, nawet dla dużych zadań i są sukcesywnie poprawiane, lecz nie są dokładne (nie można liczyć na optimum);
- często nie mają naturalnego kryterium stopu, przez co nie można nic powiedzieć o jakości otrzymanych rozwiązań.

Porównanie czasów obliczeń rzeczywistego zadania optymalizacyjnego metodą dokładną i heurystyczną

		CPLEX			EA		
SIZE		WAITING TIME [min]	# APRON*	SOLUTION TIME [s]	WAITING TIME [min]	# APRON*	SOLUTION TIME [s]
n = 4	λ = (1.00, 0.00)	0	2	0.03	0	2	1.38
	λ = (0.00, 1.00)	45	0	0.05	45	0	
n = 5	λ = (1.00, 0.00)	0	3	0.04	0	3	2.69
	λ = (0.00, 1.00)	45	1	0.07	45	1	
n = 10	λ = (1.00, 0.00)	0	6	0.10	0	6	3.23
	λ = (0.00, 1.00)	30	5	0.60	30	5	
n = 15	λ = (1.00, 0.00)	0	9	0.08	0	9	3.80
	λ = (0.00, 1.00)	35	8	3.10	35	8	
n = 30	λ = (1.00, 0.00)	0	17	24.63	0	17	4.74
	λ = (0.00, 1.00)	15	15	66.70	15	15	
n = 100	λ = (1.00, 0.00)	0	45	1348,24	0	46	11.98
	λ = (0.00, 1.00)	235	39	3462.41	10	45	

Problem optymalizacyjny

definicja

- zbiór D reprezentuje zbiór rozwiązań (dziedzinę);
- funkcja oceny/celu lub kryterium jakości

$$f: D \rightarrow R$$

- znalezienie optimum polega na znalezieniu $x^* \in D$ takiego, że dla każdego $x \in D \setminus \{x^*\}$ zachodzi

$f(x) > f(x^*)$ - minimalizacja lub

$f(x) < f(x^*)$ - maksymalizacja

- zazwyczaj w optymalizacji uwzględnia się też zbiór ograniczeń O, a znalezione rozwiązanie musi spełniać ograniczenia.

Optymalizacja lokalna

Optymalizacja lokalna to poszukiwanie jakiegokolwiek ekstremum (czyli najczęściej ekstremum lokalnego) w badanej dziedzinie problemu. W ten sposób działa spora część metod optymalizacyjnych – zbiegają do ekstremum w którego obszarze „przyciągania” rozpoczęły obliczenia.

W większości przypadków optymalizowane problemy posiadają zazwyczaj wiele optimów lokalnych i niewiele (wtedy oczywiście o takiej samej wartości funkcji celu) lub wręcz pojedyncze optimum globalne.

Typowe metody optymalizacji lokalnej to metoda największego spadku, metoda Newtona, metoda gradientów sprzężonych, algorytm wzrostu, ...

Optymalizacja globalna

Optymalizacja globalna to poszukiwanie optimum globalnego w całej dziedzinie problemu. Jest to takie działanie, jakiego oczekiwalibyśmy od metod optymalizacji. Generalnie należą tu wszystkie metody, które mają przynajmniej teoretyczną możliwość odwiedzenia w każdym kolejnym kroku lub po kilku krokach dowolnego punktu w przestrzeni rozwiązań i mają możliwość opuszczenia ekstremum lokalnego. Oczywiście należą tu również metody dokładne i pełnego przeglądu. mogą to być metody losowe i deterministyczne.

Często metody optymalizacji lokalnej stają się elementami bardziej rozbudowanych metod mających już cechy optymalizacji globalnej (np. przez dołożenie wielostartu lub reguł umożliwiających opuszczenie optimum lokalnego, czynnika losowego, itp.).

Typowe heurystyczne metody optymalizacji globalnej to algorytmy ewolucyjne, tabu search, algorytmy rojowe, algorytmy mrówkowe, itp., istnieją też metody tradycyjne: pełny przegląd, metody siatkowe metoda podziałów i ograniczeń,

Optymalizacja globalna vs. lokalna

Znaczna część używanych metod optymalizacji (poza metodami dokładnymi) to algorytmy przeszukiwania lokalnego z pewnymi rozszerzeniami, które w pewnych szczególnych sytuacjach pozwalają im znaleźć optimum globalne, a najczęściej jedynie się do niego zbliżać w czasie i/lub przestrzeni.

Niektóre metody dostarczają pewnych oszacowań o swojej dokładności, ale najczęściej nie są to metody heurystyczne. Metody heurystyczne zazwyczaj nie posiadają takich oszacowań.

Mimo tej oczywistej wady, metody heurystyczne są bardzo często używane w praktyce, gdyż szybko dostarczają satysfakcjonujących użytkowników rozwiązań.

Algorytmy heurystyczne (i nie tylko) „lepsze” i „gorsze”

Czy wobec mnogości istniejących metod optymalizacji, można je podzielić na „lepsze” i „gorsze”. Otóż okazuje się, że nie bardzo. Mogą być jedynie metody lepsze i gorsze w jakiejś klasie problemów, ale biorąc pod uwagę wszystkie możliwe problemy, takie sklasyfikowanie metod jest niemożliwe. Mówi o tym twierdzenie NFL.

Twierdzenie NFL

Twierdzenie NFL (ang. „no free lunch theorem”) D. Wolperta i W. Macready'ego stwierdza, że każde dwa algorytmy optymalizacyjne są statystycznie równie dobre, jeśli pod uwagę weźmie się wszystkie możliwe zadania optymalizacyjne.

Wnioski

1. Konieczne jest tworzenie algorytmów specjalizowanych, przystosowanych do konkretnych zadań!
2. Nieco wątpliwą wartość ma porównywanie jakości algorytmów optymalizacyjnych (często słabo dostrojonych do rozwiązywanego zadania) na postawie kilku przykładowych rozwiązanych zadań.
3. Pomimo częstego pokazywania w literaturze takich porównań, trzeba mieć świadomość ograniczonej wartości poznawczej tego typu zestawień.

Przykłady problemów optymalizacyjnych

często rozwiązywanych przy pomocy heurystyk

Dyskretne

- komiwojażera
- plecakowy
- poszukiwanie maksymalnej kliki w grafie i inne.

Parametryczne

- optymalizacja funkcji wielomodalnych np.: funkcja Rastrigina, funkcja Ackleya, funkcja Griewanka, ...

Problem komiwojażera (TSP)

- Należy odwiedzić wszystkie n miast, każde tylko raz i wrócić do punktu wyjścia (cykl Hamiltona).
- Miasta wraz z połączeniami tworzą graf pełny.
- Wagi grafu są odległościami pomiędzy miastami.
- Należy znaleźć najkrótszą trasę, która równa jest sumie odległości między odwiedzanymi miastami.
- Możliwe rozwiązania stanowią permutacje kolejności miast, stąd rozmiar przestrzeni rozwiązań wynosi $(n-1)!/2$.
- Nie znany jest dokładny algorytm o złożoności wielomianowej rozwiązujący to zadanie.
- Problem TSP jest klasy NPH.
- Możliwe są różne warianty problemu np. niesymetryczny, z niepełnym grafem połączeń, itp.

Problem plecakowy

Ze skończonego zbioru elementów $A=\{a_1, a_2, \ldots, a_n\}$, każdy o rozmiarze $s(a_i)$ i wartości $w(a_i)$ należy wybrać przedmioty (podzbiór $A' \subset A$) tak, aby:

$$\sum_{a_i \in A'} s(a_i) \leqslant b$$

$$\max_{A'} \sum_{a_i \in A'} w(a_i)$$

Problem plecakowy jest klasy NPH.

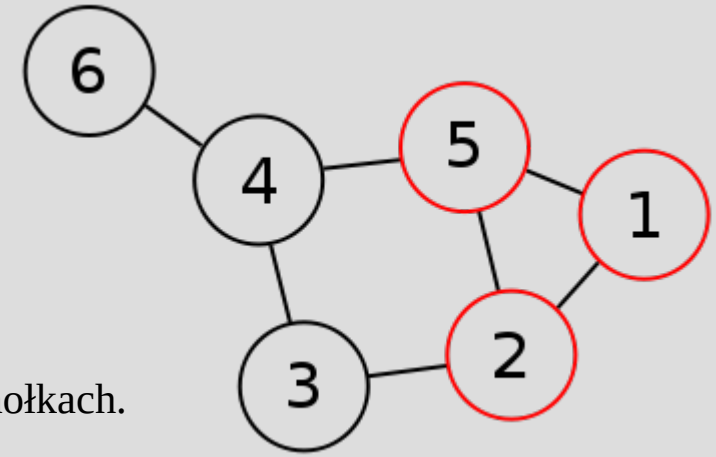
Problem maksymalnej kliki

Znalezienie podgrafu G'(V', E') grafu G(V, E) takiego że:

$$\forall v_i', v_j' \in V' \wedge j \neq i, \exists \{v_i, v_j\} \in E'$$

$$max|V'|$$

Upraszczając, jest to znalezienie największego podgrafu pełnego.
Zadanie to jest klasy NPH.



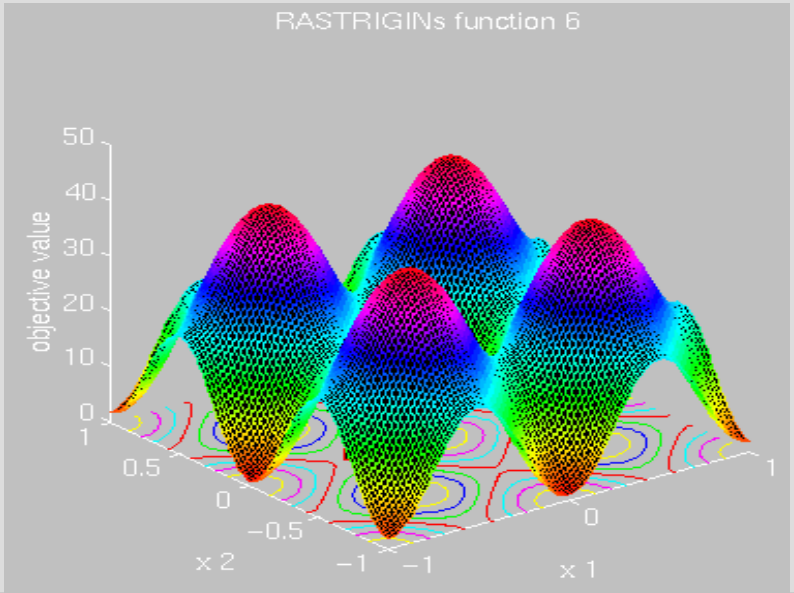
Przykład max. kliki o 3 wierzchołkach.
[Na podst. Wikipedii]

Funkcja Rastrigina

- Funkcja wielomodalna (o wielu optimach lokalnych) opisana wzorem:

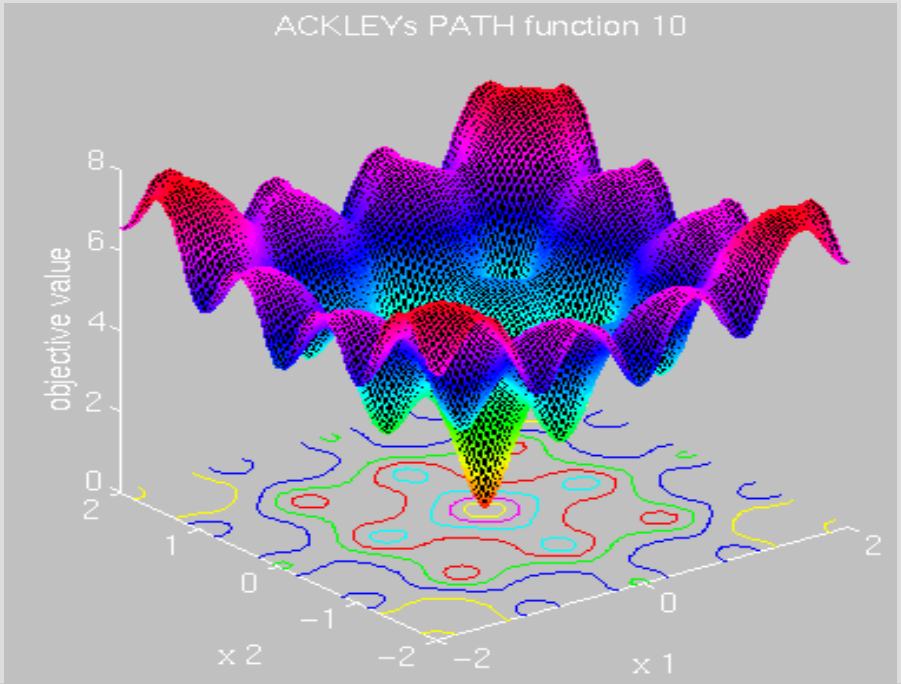
$$f(x_i) = A*n + \sum_{i=1}^{n} x_i^2 - A*\cos(\omega * x_i)$$

$$A = 10 \quad \omega = 2*\pi \quad x_i \in [-5{,}12\,;5{,}12]$$



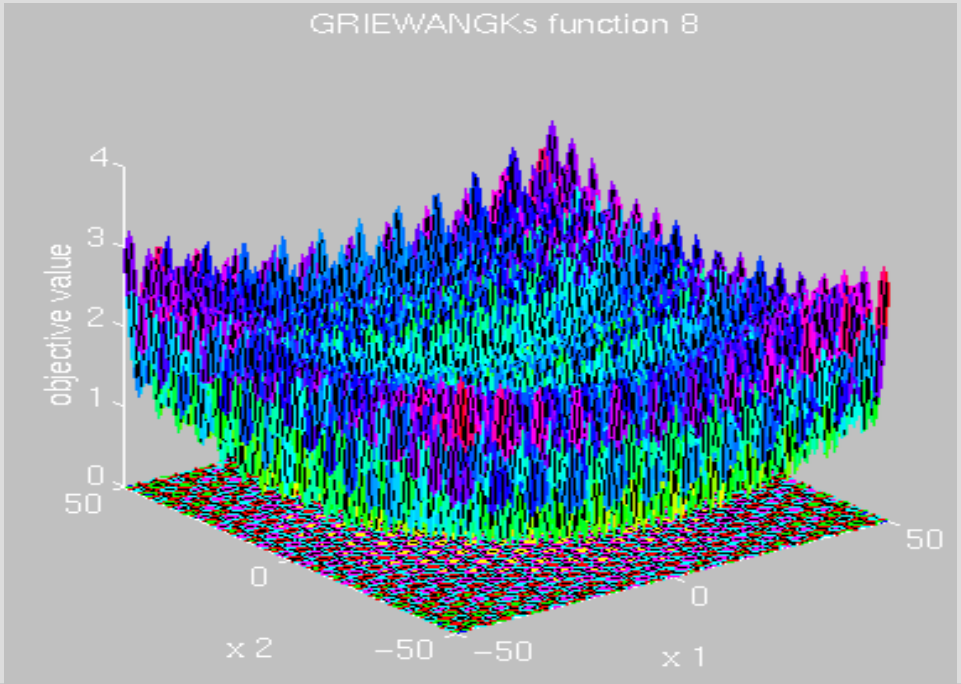
Funkcja Ackleya

$$f(x_i) = e + 20 - 20\,e^{-0.2*\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2*\pi*x_i)}$$



Funkcja Griewanka

$$f(x_i)=\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1 \qquad x_i\in[-512;512]$$



Przegląd metod lokalnych - algorytmy zachłanne

Algorytm zachłanny (ang. greedy algorithm) – w każdym kroku swego działania wybiera opcję dającą **lokalnie** największy zysk (stąd „zachłanność”).
Algorytmy takie zazwyczaj nie są optymalne i nie gwarantują znalezienia optimum, np. dla problemu TSP (zazwyczaj daje rozwiązanie o ok. 25% gorsze od optimum, w pewnych przypadkach o 100% gorsze).
Przykłady optymalnych algorytmów zachłannych:

- algorytm Dijkstry - znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi (najkrótszej ścieżki w grafie);
- algorytm Kruskala - algorytm grafowy wyznaczający minimalne drzewo rozpinające dla spójnego grafu nieskierowanego ważonego;
- algorytm Prima – algorytm grafowy wyznaczający minimalne drzewo rozpinające dla grafu spójnego, nieważonego.

Algorytm zachłanny (nieoptymalny) może służyć do generacji rozwiązań startowych dla innych metod lub lokalnego poprawiania posiadanych rozwiązań.

Algorytm przeszukiwania lokalnego algorytm (największego) wzrostu – jedna z najprostszych heurystyk obliczeniowych

begin
    t:=0
    wybierz rozwiązanie początkowe x_s
    oceń x_s
    x_0:=x_s
    repeat
        wygeneruj n nowych rozwiązań x_1... x_n przez modyfikację/perturbację/mutację x_0
        oceń rozwiązania x_1... x_n
        wybierz najlepsze rozwiązanie x^* z x_1...x_n
        jeśli x^* lepsze od x_0 to x_0:=x^*
        t:=t+1
    until t=MAX
end

Algorytm (największego) wzrostu
możliwości rozwoju

- procedura generacji rozwiązania początkowego nie musi być losowa, może zawierać informacje o poprzednio otrzymanych rozwiązaniach, rozwiązywanym problemie, wykorzystywać metodę zachłanną, itp.;
- modyfikacja rozwiązania (perturbacja, mutacja) nie musi być tylko losowa, może być adaptacyjna, o zmiennym zasięgu, rozkładzie, wykorzystywać informacje o sąsiedztwie, problemie, itp.;
- wybór najlepszego rozwiązania może również wykorzystywać informacje o historii rozwiązań, sąsiedztwie, zwierać reguły akceptacji zapobiegające zapętleniu lub osiadaniu w ekstremum lokalnym.

Niemal nieograniczone możliwości ulepszania algorytmu z wykorzystaniem kolejnych heurystyk to jedna z głównych cech metod heurystycznych!
Przykładami takich „rozwiniętych” wersji algorytmu wzrostu są np. algortmy 2-*opt*, 3-*opt* (znane też jako *k*-*opt* lub λ-*opt*), algorytm Lin-Kernighan (LKH) stosowany głównie do rozwiązywania problemu TSP, tabu search, tzw. „algorytm małych światów” i wiele innych.

Zmodyfikowany algorytm wzrostu ze zmiennym sąsiedztwem

begin
    k:=1
    t:=0
    wybierz rozwiązanie początkowe x_0
    oceń x_0
    repeat
        utwórz nowe rozwiązania $x_1 \ldots x_n$ przez modyfikację x_0 w sąsiedztwie o rozmiarze k
        oceń rozwiązania $x_1 \ldots x_n$
        wybierz najlepsze rozwiązanie x^* z $x_1 \ldots x_n$
        jeśli x^* lepsze od x_0 to x_0:=x^*; k:=1;
        jeśli nie to k:=k+1;
        t:=t+1
    until t=MAX
end

Algorytm 2-*opt*

```
repeat until no improvement is made {
      start_again:
      best_distance = calculateTotalDistance(existing_route)
      for (i = 0; i < number of nodes eligible to be swapped - 1; i++) {
          for (k = i + 1; k < number of nodes eligible to be swapped; k++) {
              new_route = 2optSwap(existing_route, i, k)
              new_distance = calculateTotalDistance(new_route)
              if (new_distance < best_distance) {
                  existing_route = new_route
                  goto start_again
              }
          }
      }
  }

2optSwap(route, i, k) {
      1. take route[0] to route[i-1] and add them in order to new_route
      2. take route[i] to route[k] and add them in reverse order to new_route
      3. take route[k+1] to end and add them in order to new_route
      return new_route;
  }
```

Algorytmy 2-*opt*, 3-*opt* i *k*-*opt*

Poza algorytm 2-*opt* wykorzystywane są czasem 3-opt i *k*-opt o ewentualnie większych wartościach *k*, ich złożoność obliczeniowa jest wielomianowa i wzrasta o kolejne potęgi dla rosnącego *k*, natomiast otrzymywane wyniki nie poprawiają się już tak spektakularnie. Algorytmy te są oczywiście rozszerzeniem 2-*opt*, tylko konieczne jest badanie znacznie większej liczby warunków w dodatkowych pętlach – dla algorytmu 3-*opt* można przełączyć krawędzie na 8 sposobów i tyle też możliwości trzeba sprawdzić.

Liczba miast	Średnia długość trasy początkowej	2-opt		3-opt	
		wynik	czas	wynik	czas
10	270,4	142,7	0,002	141,9	0,021
20	528,5	186,1	0,019	180,8	0,591
30	793,2	226,5	0,073	217,7	3,749
40	1040,5	254,2	0,182	243,2	13,074

Porównanie wyników i czasów działania algorytmów 2-*opt* i 3-*opt*.

Wady metod optymalizacji lokalnej

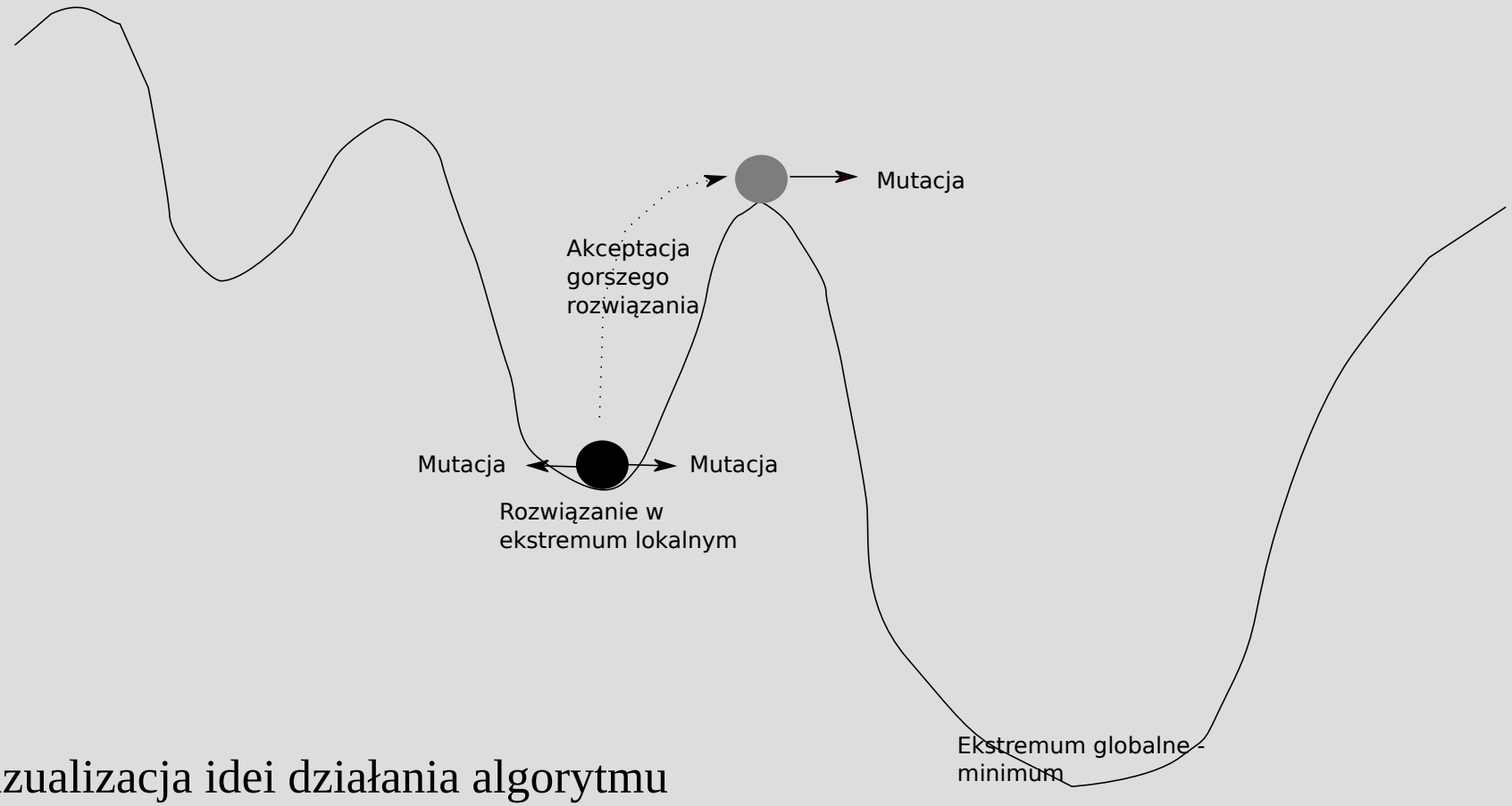
Metody optymalizacji lokalnej źle radzą sobie w przestrzeni z dużą liczbą optimów lokalnych.
W większości przypadków nie potrafią opuścić optimum lokalnego lub jest to dla nich dużym problemem.
Nie potrafią eksplorować większej liczby optimów globalnych, co bywa przydatne np. w przypadku optymalizacji funkcji wielomodalnych, optymalizacji dyskretnej lub wielokryterialnej – poszukiwania frontu Pareto.

Odpowiedzią na te problemy są algorytmy optymalizacji lokalnej przystosowane do optymalizacji globalnej, między innymi algorytm Metropolisa, zwany także symulowanym wyżarzaniem (ang. simulated annealing - SA).

Algorytm Metropolisa

wygeneruj rozwiązanie startowe x i oceń jego jakość.
$x^*=x$ //początkowe rozwiązanie jest także najlepszym rozwiązaniem
ustal $T(0)$ i schemat chłodzenia $g(T(n))$ //$T(0)$ – temperatura początkowa; $g(T(n))$ – funkcja zmian temperatury w trakcie obliczeń
$n=0$
do {
  $licznik=0$
  do {
    wygeneruj nowe rozwiązanie x' operatorem perturbacji/mutacji
    if ($F(x') < F(x)$) { //minimalizacja, przy maksymalizacji porównanie odwrotne
      $x=x'$ //jeśli nowe rozwiązanie jest lepsze, to jest zawsze akceptowane
      $x^*=x$
    else {
      z przedziału (0,1) wylosuj liczbę y
      if ($y<\exp(-(F(x')- F(x))/kT)$) { //tu wykorzystywany jest rozkład Boltzmanna do akceptacji rozwiązań
        $x=x'$
        if ($F(x)<F(x^*)$) $x^*=x$
      }
    $licznik=licznik+1$
  }
  while ($licznik<$MAX_P) //MAX_P – liczba powtórzeń przy stałej temperaturze
  $T(n+1)=g(T(n))$ //zmiana temperatury
  $n=n+1$
}
while ($n<$MAX_I) //MAX_I – maksymalna liczba iteracji

Losowość w algorytmie Metropolisa

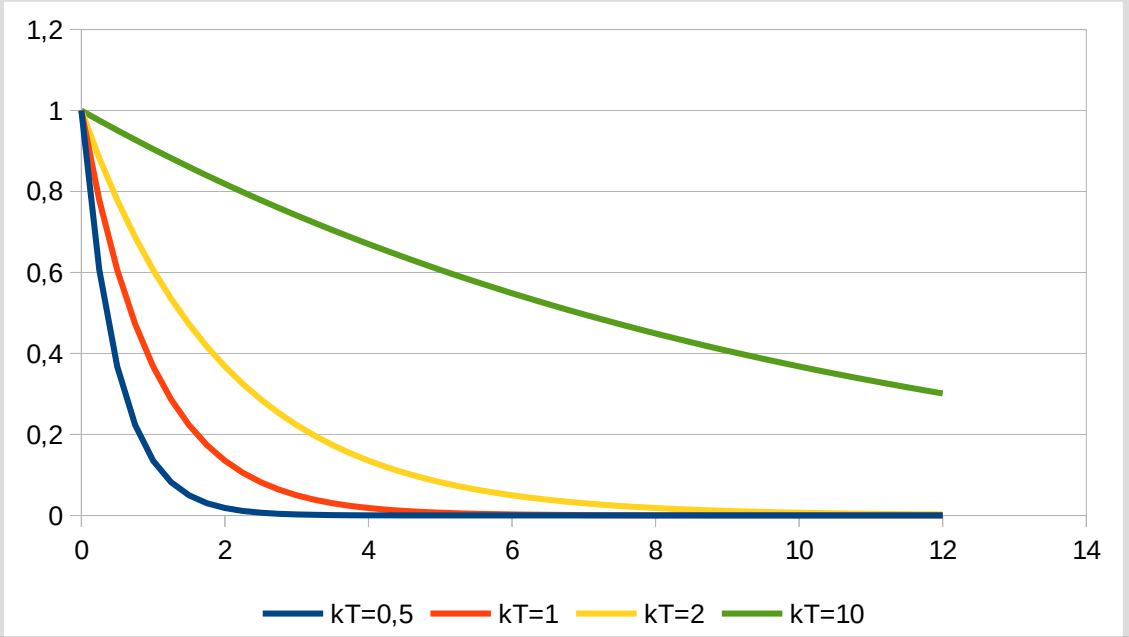


Wizualizacja idei działania algorytmu Metropolisa

Rozkład Boltzmanna

Istotnym elementem, różniącym tę metodę od innych metod przeszukiwania lokalnego, jest reguła akceptacji rozwiązań wykorzystująca rozkład Boltzmanna.
Rozkład Boltzmanna opisuje między innymi rozkład energii (stanów energetycznych) cząstek w pewnej temperaturze, wyraża się wzorem:

$$P(\delta E) = \alpha * \exp\left(\frac{-\delta E}{kT}\right)$$



Symulowane wyżarzanie

- Metoda jest kontynuacją metod lokalnego przeszukiwania.
- Nowym elementem w niej występującym jest możliwość akceptacji rozwiązania gorszego z pewnym prawdopodobieństwem, zależnym od jego jakości i opisywanym rozkładem Boltzmanna.
- Daje to możliwość opuszczenia ekstremum lokalnego i eksploracji nowych obszarów dziedziny rozwiązań.
- Operator mutacji/perturbacji wykorzystuje najczęściej rozkład normalny, tworząc nowe rozwiązanie w pewnej, losowo wybranej odległości od rozwiązania.
- Sama nazwa metody wzięta jest z dziedziny technik obróbki materiałów, w których odpowiednio prowadzony proces schładzania umożliwia wytworzenie materiałów o małej liczbie defektów w sieci krystalicznej, czyli w efekcie o lepszych parametrach mechanicznych niż chłodzone w sposób niekontrolowany.
- Odpowiedni sposób chłodzenia umożliwia cząstkom zajęcie minimalnych stanów energetycznych – najsilniejszych wiązań, czyli ekstremów funkcji energii, która w przypadku algorytmu jest zastępowana przez funkcję celu.

Symulowane wyżarzanie cd.

Głównym parametrem, modyfikującym działanie metody jest temperatura występująca w rozkładzie Boltzmanna. Temperatura początkowa i zmiany temperatury w trakcie obliczeń wpływają na kształt rozkładu, a więc i na prawdopodobieństwo akceptacji rozwiązania gorszego. Zastosowany schemat zmian temperatury (schemat chłodzenia) ma duży wpływ na efektywność metody.

Na prezentowanych wcześniej wykresach można zauważyć, że im temperatura wyższa, tym łatwiej akceptowane są rozwiązania gorsze, wraz ze spadkiem temperatury, rozkład prawdopodobieństwa zaczyna przypominać rozkład skokowy z możliwością akceptacji tylko rozwiązania równego najlepszemu. Dlatego też obliczenia rozpoczyna się przy „wysokiej” temperaturze (eksploracja przestrzeni rozwiązań), a kończy przy bliskiej zeru (eksploatacja znalezionego ekstremum).

Ważnym parametrem jest też liczba iteracji wykonywana w stałej temperaturze.

Symulowane wyżarzanie -
schematy chłodzenia

- geometryczny: $T(n+1)=\alpha*T(n)$, gdzie $0<\alpha<1$, w praktyce nieco mniejsze od 1;
- liniowy: $T(n+1)=T(n)-\alpha*n$, gdzie $\alpha>0$;
- wykładniczy: $T(n)=T_0(T_N/T_0)^{n/N}$, gdzie $T(0)=T_0$, $T(N)=T_N$;
- logarytmiczny: $T(n)=T_0/ln(n+1)$, gdzie $T(0)=T_0$;
- harmoniczny: $A/(i+1) + T_0 - A$, gdzie $A=(T_0 - T_N)*(N+1)/N$;
- i wiele innych.

- Generalną zasadą w tworzeniu schematu chłodzenia jest to, że $T_N<T_0$.
- T_N powinna być niska, bliska 0.
- Istnieją też niemonotoniczne schematy chłodzenia, jednakże zawsze T_N jest bliska 0.

Symulowane wyżarzanie -
rozszerzenia metody

- Fast Simulated Annealing wykorzystuje operator mutacji/perturbacji z rozkładem Cauchy'ego.
- Simulated Quenching, czyli bardzo szybki SA z wykładniczym schematem chłodzenia, przydatny w niektórych klasach zadań.
- Adaptive Simulated Annealing – wykorzystanie metody w zadaniach wielowymiarowych, znacznie różniących się właściwościami w różnych wymiarach, umożliwia niezależne ustalenie technik chłodzenia dla różnych wymiarów.
- Niektórzy badacze uważają, że algorytmy ewolucyjne są rozwiniętą wersją symulowanego wyżarzania.

Algorytmy ewolucyjne
początki

- Algorytmy ewolucyjne zostały wymyślone równolegle w kilku liczących się ośrodkach naukowych w różnych krajach (Niemcy, USA) ok. 40 lat temu.
- Wszystkie w różnym stopniu zainspirowane były teorią ewolucji i w różny sposób starały się uchwycić jej zasady.
- Mimo, że różniły się zarówno rozwiązywanymi problemami, jak i sposobem zapisu i samym przebiegiem działania, to jednak po pewnym czasie zauważono ich wspólne korzenie i wszystkie zostały zaklasyfikowane jako metody/algorytmy ewolucyjne.
- Te wspólne cechy algorytmów to:
  - występowanie populacji rozwiązań;
  - użycie selekcji, promującej lepsze rozwiązania do dalszych obliczeń, a odrzucających gorsze (na wzór selekcji naturalnej);
  - wykorzystywanie różnych metod zakodowania problemu (rozwiązań), zamiast rozwiązywania go bezpośrednio;
  - użycie operatorów genetycznych, czyli metod modyfikujących rozwiązania, najczęściej losowo (mutacja) lub przez losowe krzyżowanie cech rozwiązań rodzicielskich (nie wszystkie metody ewolucyjne używały krzyżowania);
  - otrzymanie dobrych rozwiązań wymaga wielu iteracji (pokoleń, generacji, epok) podczas których wymienione wyżej metody oddziałują na populację.

Algorytmy ewolucyjne
podział metod

Nazwy i podział algorytmów ewolucyjnych nie jest jeszcze w pełni ustalony, jednak najczęściej można się spotkać z następującymi kategoriami algorytmów:

- algorytmy genetyczne, charakteryzują się zastosowaniem kodowania binarnego, prostej mutacji i krzyżowania oraz selekcji ruletkowej (bardziej rozwinięte wersje wykorzystują też inne sposoby kodowania, operatory i metody selekcji);
- strategie ewolucyjne, charakteryzują się wykorzystywaniem kodowania rzeczywistoliczbowego, początkowo wykorzystywały tylko mutacją, później także krzyżowanie, selekcję rankingową, później zastosowano m. in. samoadaptację;
- programowanie genetyczne – ewoluujące programy, a bardziej wyrażenia, zakodowane jako drzewa;
- programowanie ewolucyjne – ewoluujące automaty, zakodowane jako grafy przejść między stanami;
- metody hybrydowe, specjalizowane, algorytmy memetyczne, ewolucja różnicowa, ... – wszelkiego rodzaju metody ewolucyjne, zawierające elementy powyższych, specjalizowane – zawierające wiedzę o problemie z dodatkowymi metodami optymalizacji lokalnej;
- przeszukiwanie rozproszone – deterministyczna wersja AE z precyzyjnym doborem populacji startowej (tzw. ziarna) optymalizacją lokalną i specyficznym krzyżowaniem;

Algorytmy ewolucyjne
nomenklatura

Biologiczne korzenie AE wpłynęły na powstanie w tej dziedzinie specyficznego słownictwa, zapożyczonego z genetyki i innych działów biologii. Co prawda można je zastąpić mającymi to samo lub podobne znaczenie (w tym przypadku i ujęciu) pojęciami matematyczno-informatycznymi, to jednak pojęcia biologiczne są często (nad)używane, czasem nie mają też jednoznacznych odpowiedników w AE.
Do najczęściej używanych pojęć zapożyczonych z biologii należą:

- chromosom – oryginalnie odpowiednio upakowana nić DNA w jądrze komórki; w AE ciąg kodowy, zakodowany fragment rozwiązania (np. dla 1 wymiaru);
- gen – w genetyce fragment chromosomu, nici DNA, kodujący konkretny aminokwas (podstawową cegiełkę budującą białka); w AE fragment fragment kodu, kodujący jakąś specyficzną właściwość, element rozwiązania, jeden z wymiarów w przestrzeni rozwiązań, często trudny do praktycznego wyodrębnienia;
- allel – w genetyce wariant cechy kodowanej przez gen; w AE konkretna wartość przypisana genowi;
- locus – w genetyce pozycja genu w nici DNA; w AE położenie odpowiedniego genu w kodzie rozwiązania (pojęcie może być trudne do interpretacji przy różnych metodach kodowania);

Algorytmy ewolucyjne
nomenklatura c.d.

- genotyp – w genetyce zapis genetyczny danego organizmu; w AE struktura rozwiązania, rozwiązanie problemu w przestrzeni kodowej;
- fenotyp – w biologii osobnik powstały z genotypu; w AE rozwiązanie zadania w jego naturalnej przestrzeni;
- osobnik – w biologii konkretny organizm; w AE konkretne rozwiązanie zadania;
- populacja – w biologii zbiór osobników jednego gatunku; w AE zbiór rozwiązań;
- mutacja – w genetyce przypadkowa, losowa zmiana w nici DNA; w AE operator genetyczny modyfikujący rozwiązania (losowy lecz uruchamiany specjalnie w celu modyfikacji rozwiązania);
- krzyżowanie – w biologii rozmnażanie płciowe osobników z rekombinacją genów rodziców; w AE drugi operator genetyczny tworzący nowe rozwiązanie z 2 lub więcej rozwiązań składowych przez rekombinację (odpowiednie zmieszanie) cech (kodu) rozwiązań „rodzicielskich”;
- selekcja – w biologii zespół czynników środowiskowych, powodujących „odsiewanie” gorzej przystosowanych osobników (selekcja naturalna), w AE operator genetyczny polegający na (zazwyczaj miękkiej) eliminacji rozwiązań słabiej ocenianych przy użyciu funkcji celu/dopasowania.

Algorytmy ewolucyjne
ogólny schemat działania

t=0
Inicjacja populacji startowej P(0)
Ocena rozwiązań w P(0)
do {
  T(t)=Reprodukcja P(t) //powielenie i krzyżowanie osobników
  O(t)=Modyfikacja T(t) //mutacja osobników
  Ocena rozwiązań w O(t)
  P(t+1)=Selekcja/Sukcesja(O(t) lub O(t)∪P(t))
   t=t+1
}
while(warunek stopu)

Algorytmy ewolucyjne
ogólny schemat działania cd.

- Metoda polega na iteracyjnym powtarzaniu tych samych czynności: tworzenia nowej populacji (potomnej) przez modyfikację rozwiązań z populacji poprzedniej (rodzicielskiej) oraz selekcję w jakimś sensie lepszych rozwiązań z tej zmodyfikowanej (lub obu).
- Operatory modyfikujące populację (krzyżowanie i mutacja) posiadają prawdopodobieństwa wykonania raczej mniejsze od 1, nie każdy osobnik musi być przez nie zmieniony. Mutacja najczęściej ma bardzo małe (~0,01), a krzyżowanie duże (~0,9) prawdopodobieństwo wykonania.
- Jak widać do działania tej metody optymalizacyjnej potrzebna jest jedynie informacja o jakości rozwiązań (łatwa do obliczenia funkcja celu, często nazywana funkcją przystosowania, ang. fitness function).
- Metoda selekcji jak i metody reprodukcji i modyfikacji rozwiązań zależą nieco od wybranej konkretnej wersji AE i są w dużej mierze dobierane do wymagań rozwiązywanego problemu spośród wielu możliwych.
- Nowoczesne metody ewolucyjne bazują raczej na kryterium użyteczności danego zestawu metod (reprodukcji, mutacji, selekcji) do danego problemu niż na ortodoksyjnej poprawności i zgodności z metodami zaproponowanymi przez ich twórców.

Algorytmy ewolucyjne
wady metody

- Obliczenia są dość powolne i zasobochłonne.
- Dobre efekty można uzyskać tylko po dobrym przystosowaniu metody do wymogów zadania (wybór odpowiedniej metody kodowania, operatorów, selekcji, dostrojeniu parametrów). Standardowa wersja metody, stosowana do wszystkich problemów raczej nie spełni oczekiwań,
- Metoda często cierpi na chorobę „przedwczesnej zbieżności”, czyli zablokowania się rozwiązań w optimach lokalnych i stagnacji obliczeń.
- Problemowi „przedwczesnej zbieżności” trudno jest zapobiegać, wynika on często ze źle dobranych parametrów metody, ale też i ograniczeń środowiska komputerowego (skończona dokładność obliczeń, słabe parametry generatorów liczb losowych).
- Z problemem przedwczesnej zbieżności wiąże się też problem małego zróżnicowania populacji. Jeśli osobniki są prawie jednakowe, to efektywność obliczeń ewolucyjnych znacząco spada (wymiana jednakowej informacji między osobnikami).
- Brak sensownego kryterium stopu, ewolucja zapewne nie ma końca (koniec świata?) i trzeba zapewnić je sztucznie.
- Jak wszystkie metody heurystyczne nie gwarantuje znalezienia optimum w skończonej liczbie iteracji (dlatego symulacja AE prawie zawsze kończy się stagnacją, bo nie możemy czekać na wynik w nieskończoność).

Algorytmy ewolucyjne
zalety metody

- Nie potrzebne są założenia o postaci funkcji celu, istnieniu pochodnych, ciągłości, itp.
- Tego samego schematu rozwiązywania można używać do szerokiej gamy zadań.
- Obliczenia prowadzone są równolegle przez dużą grupę „osobników” - rozwiązań, więc jest spora szansa na odwiedzenie wielu obiecujących obszarów.
- Obliczenia trudniej utykają w ekstremach lokalnych i łatwiej mogą je opuścić dzięki wymianie informacji pomiędzy „osobnikami”.
- Obliczenia można łatwo zrównoleglić w systemach wieloprocesorowych, rozproszonych, itp.
- Metoda jest skalowalna, łatwo ją przystosować do konkretnego zadania i jego rozmiaru.
- Mimo braku gwarancji na otrzymanie rozwiązań optymalnych, dobrze dobrana metoda działa szybko i skutecznie.

Algorytmy ewolucyjne
eksploracja i eksploatacja

- Jednym ze sposobów walki z „przedwczesną zbieżnością” jest zbalansowanie działania operatorów genetycznych mutacji i krzyżowania (lub analogicznych).
- Operator mutacji wprowadza losowe zaburzenie do rozwiązania, jest odpowiedzialny za właściwości eksploracyjne ewolucji (operator globalny), dzięki jego działaniu możliwe jest tworzenie nowych cech w osobnikach (np. przeniesienie osobnika do innego obszaru dziedziny, tworzenie nowych sekwencji w rozwiązaniu), niestety pozytywne zmiany generowane są dość rzadko i aby zminimalizować niekorzystne, mutacja wykonywana jest dość rzadko, z niskim prawdopodobieństwem. W naturze organizmy również wkładają dużo wysiłku, aby bronić się przed mutacjami. Zbyt intensywna mutacja powoduje, że AE zaczyna przypominać metodę Monte-Carlo – losowe przeszukiwanie przestrzeni rozwiązań bez dziedziczenia informacji po „przodkach”. Zbyt słaba mutacja zatrzyma eksplorację w AE.
- Operator krzyżowania odpowiedzialny jest za eksploatacyjne właściwości ewolucji. Nowy osobnik powstaje z rekombinacji cech rodziców, czyli jest do nich podobny (operator lokalny) możliwe, że odziedziczy najlepsze ich cechy, operator ten wykonywany jest bardzo często (z prawdopodobieństwem bliskim 1). W naturze krzyżowanie (rozmnażanie płciowe) zostało wynalezione prawdopodobnie „dość późno”, niecały 1 mld lat temu (historia życia na ziemi to ok. 3,7 mld lat), jednakże stało się to początkiem gwałtownego rozwoju życia.

Algorytmy ewolucyjne
eksploracja i eksploatacja c.d.

- Wyłączenie mutacji w AE skutkowałoby szybką zbieżnością do najbliższych optimów lokalnych. Algorytm ewolucyjny bez krzyżowania działałby zdecydowanie wolniej, tzn. suboptymalne lub przy odrobinie szczęścia optymalne rozwiązania pojawiłyby się po bardzo długim czasie, lecz droga do nich nie byłaby zamknięta. Zastosowanie obu operatorów z dobrze dobranymi prawdopodobieństwami wykonania zdecydowanie przyspiesza obliczenia i zwiększa efektywność metody.
- Duże znaczenie w walce z przedwczesną zbieżnością ma też zastosowana metoda selekcji, która wprowadza wprowadza presję selekcyjną. Silną mutację można też zrównoważyć silną selekcją, promującą tylko najlepsze osobniki.
- Zwiększenie właściwości eksploracyjnych uzyskuje się też przy zastosowaniu selekcji nowej populacji tylko z osobników z populacji tymczasowej (pomijając populację rodziców), jest to tzw. strategia rozwoju populacji (μ, λ) - stosowana jest głównie przy optymalizacji zadań niestacjonarnych, bardziej zachowawcza (eksploatacyjna) jest strategia rozwoju populacji $(\mu+\lambda)$, w której nową populację wybiera się z obu podpopulacji (rodzicielskiej i tymczasowej), stosowana jest w rozwiązywaniu problemów stacjonarnych.

Algorytmy ewolucyjne
kodowanie rozwiązań

W algorytmach ewolucyjnych operuje się nie na rzeczywistym rozwiązaniu, a jego symbolicznej reprezentacji, którą w każdym momencie powinno się dać odkodować do konkretnego rozwiązania.
W tradycyjnych metodach ewolucyjnych używano kodowania:

- **binarnego**, w którym należy odpowiednio zakodować problem tak, aby można było go wyrazić przy użyciu zapisu binarnego, np. dla przypadku optymalizacji funkcji ciągłej skończony podzbiór zbioru liczb rzeczywistych (osi liczbowej) dzieli się na liczbę przedziałów o skończonej długości i liczbie będącej całkowitą potęgą 2 (w przypadku wielowymiarowym, operację przeprowadza się dla każdego wymiaru), przedziałom przydziela się kolejne numery zapisane binarnie, chcąc uzyskać większą dokładność obliczeń należy użyć większej liczby bitów, przekodowanie na liczbę rzeczywistą uzyskujemy ze wzoru:

$$z = z_o + \frac{z_k - z_0}{2^n - 1} * \sum_{i=0}^{n-1} x_i * 2^i$$

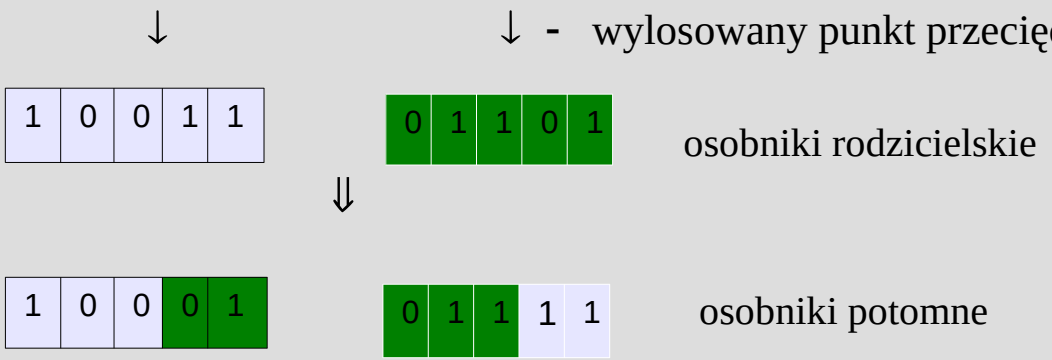
gdzie: z_0, z_k- początek i koniec przedziału, n – liczba bitów, z – odkodowana liczba, x_i – wartość bitu o numerze i (licząc od najmniej znaczącego bitu zerowego).

- **rzeczywistoliczbowego**, używano go oczywiście tylko do optymalizacji funkcji ciągłych, w przypadku funkcji wielowymiarowych mamy do czynienia z wektorem liczb rzeczywistych.

Obecne metody ewolucyjne używają w zasadzie dowolnych struktur danych do zakodowania rozwiązań: wektorów, tablic, drzew zawierających symbole binarne, całkowitoliczbowych i rzeczywiste, automatów skończonych, a nawet ewoluujących programów ...

Algorytmy ewolucyjne
operatory genetyczne – przypadek binarny

W tradycyjnych AE starano się być jak najbliżej biologicznego pierwowzoru i stosowano najchętniej kodowanie binarne (akurat w DNA są 4 symbole kodujące!), mutację i krzyżowanie binarne. Obecnie też są one stosowane, choć nie „na siłę”.
Mutacja binarna polega na losowaniu bitu w którym z pewnym niewielkim prawdopodobieństwem następuje przekłamanie wartości (z 0 na 1 lub odwrotnie).
Krzyżowanie polega na przecinaniu w wylosowanym miejscu ciągów binarnych w 2 rozwiązaniach rodzicielskich, a następnie łączeniu ich „na krzyż” tak, aby nowe rozwiązania tworzyły fragmenty pochodzące z różnych osobników – ilustruje to poniższy schemat:



Algorytmy ewolucyjne
operatory genetyczne – inne przypadki

W przypadku kodowania rzeczywistoliczbowego mutacja to zwykła modyfikacja liczby występującej w rozwiązaniu o wartość wylosowaną z rozkładu normalnego (czasem też Cauchy'ego):

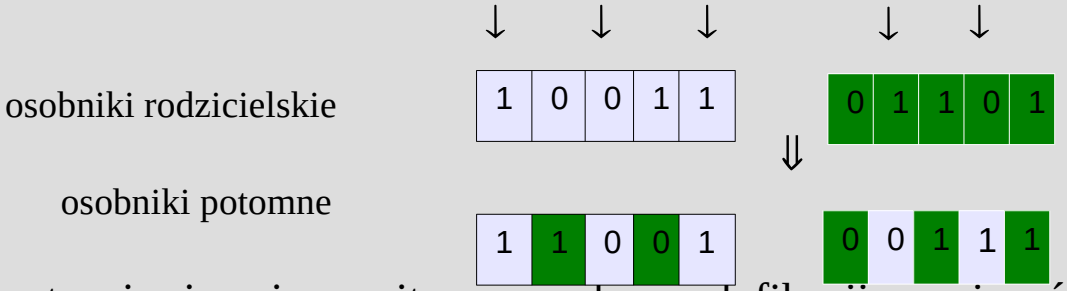
$$X^{t+1}_i = X^t_i + N(0,1)$$

a krzyżowanie to tzw krzyżowanie uśredniające:

$$X^{t+1}_i = \alpha * X^t_i + (1-\alpha) * Y^t_i \quad Y^{t+1}_i = \alpha * Y^t_i + (1-\alpha) * X^t_i$$

X^{t+1}_i, Y^{t+1}_i – i-ta współrzędna osobnika potomnego, X^t_i, Y^t_i – i-ta współrzędna osobnika rodzicielskiego, α- wartość losowana z rozkładu U(0,1).

W kodowaniu binarnym i rzeczywistoliczbowym (tylko w przypadku wielowymiarowym) można stosować też krzyżowanie z maską (równomierne). Polega ono na losowym wybieraniu do osobników potomnych wartości kolejnych współrzędnych osobników rodzicielskich. Ta która zostanie wylosowana trafi do pierwszego potomka, elementy niewylosowane utworzą 2 osobnika:



Obecnie stosuje się najrozmaitsze sposoby modyfikacji rozwiązań: losowe i deterministyczne, łącznie z metodami bazującymi na wiedzy o problemie, prostymi metodami optymalizacji lokalnej i heurystykami.

Algorytmy ewolucyjne

selekcja osobników do populacji potomnej
selekcja proporcjonalna (ruletkowa)

- Każdy osobnik ma przyporządkowaną wartość funkcji dopasowania *f(j)*.
- Prawdopodobieństwo wyboru i wartość oczekiwana liczby potomków to:

$$p(j)=\frac{f(j)}{\sum_{l=1}^{k} f(l)} \qquad E(p(j))=\frac{k*f(j)}{\sum_{l=1}^{k} f(l)}=\frac{f(j)}{\bar{f}}$$

gdzie: *k* – liczność populacji, *f(j)* – wartość funkcji dopasowania osobnika *j*, $\bar{f}$ - średnia wartość funkcji dopasowania populacji.

- Procedurę losowego wyboru przeprowadzamy *k* (dokładniej tyle razy, ile osobników ma mieć populacja potomna, nie zawsze musi to być taka sama liczba jak dla populacji rodzicielskiej) razy.

Wniosek!
Osobniki o większym prawdopodobieństwie powinny mieć więcej potomków niż osobniki o mniejszym, a tym samym populacja powinna ewoluować w kierunku lepszego przystosowania – lepszych rozwiązań.

Algorytmy ewolucyjne
selekcja osobników do populacji potomnej

- **selekcja rangowa** - polega na nadaniu rozwiązaniom wartości (rang) na podstawie wartości funkcji dopasowania przez ich posortowanie i nadanie kolejnych wartości rang według wzorów $r(X)=a+k(1-f(X)/r_{max})$ lub $r(X)=a+k(r_{max}-f(X))^b$, prawdopodobieństwo wyboru określa się przez normalizację rang, a dalszy ciąg działania tak jak w selekcji proporcjonalnej, *a, b* i *k* to parametry metody, $r(X)$, r_{max} – ranga rozwiązania X i ranga maksymalna $f(X)$ – wartość funkcji dopasowania rozwiązania X. Metoda ta ma lepsze parametry niż s. ruletkowa i jest stosowana w rozwiązaniach praktycznych.
- **wybór najlepszych osobników z populacji** – algorytm działania jest oczywisty (sortowanie), w metodzie tej należy wybierać rozwiązania z większej populacji niż docelowa, metoda powoduje dość łatwe wpadanie populacji w minima lokalne lub unifikację populacji, nie ma wobec tego najlepszych parametrów i nie jest zbyt chętnie stosowana.
- **selekcja turniejowa** – z populacji losowanych jest k osobników ($k \geq 2$), najlepszy jest wybierany, możliwe są wersje ze zwracaniem i bez; jest to zdecydowanie jedna z najlepszych i jednocześnie najprostszych metod selekcji, jest i powinna być używana w rozwiązaniach praktycznych, jej użycie w rozważaniach teoretycznych jest dość skomplikowane.

Algorytmy ewolucyjne
selekcja osobników do populacji potomnej

- **selekcja deterministyczna** – części całkowite wartości oczekiwanej liczby potomków obliczane są jak w selekcji proporcjonalnej, stają się liczbami potomków dla osobników, ewentualny nadmiar lub niedomiar jest odpowiednio poprawiany przez dołączenie kolejnych najlepszych lub usunięcie najsłabszych; metoda ta charakteryzuje się bardzo silnym naciskiem selekcyjnym, jednakże prowadzi to do szybkiej unifikacji populacji (straty różnorodności) i osiadania w optimach lokalnych, metoda nie jest więc zbyt często używana.
- **selekcje sterowane –** istnieją metody selekcji, które zmieniają swoje parametry w trakcie działania programu lub składają się z kilku metod składowych o różnej sile nacisku selekcyjnego i zmieniają, w zależności od stanu populacji, częstość ich stosowania.
- **czas życia osobnika** – istnieją rozwiązania, w których zarówno liczność populacji jak i selekcja nie są typowo ustalone, a każdy osobnik otrzymuje czas życia w iteracjach algorytmu, zależny od jego jakości;

Algorytmy ewolucyjne
strategie rozwoju populacji

Oprócz samej selekcji, ważna jest też **strategia rozwoju populacji (sukcesja)**, czyli które osobniki będą mogły być poddane selekcji. Istnieje tu kilka możliwości:

- **strategia (μ, λ)** – osobniki do nowej populacji rodzicielskiej są wybierane tylko z (pod)populacji potomnej, tu każdy osobnik może żyć tylko jedną iterację (epokę, pokolenie), tu μ - oznacza (pod)populację rodzicielską, λ - (pod)populację potomków.
- **strategia (μ + λ) –** osobniki do nowej populacji rodzicielskiej są wybierane z (pod)populacji potomnej i rodzicielskiej, czyli osobnik może żyć prawie dowolnie długo (wiele iteracji, pokoleń, epok), jeśli jest dobry.
- **elitarna** – część najlepszych osobników z obu (pod)populacji trafia do nowej rodzicielskiej bez selekcji;
- **mieszana** – można zezwolić na przejście pewnego procentu osobników (np. najlepszych – elitarność lub wyselekcjonowanych) z obu (pod)populacji do nowej rodzicielskiej.

Algorytmy ewolucyjne
uwzględnienie ograniczeń

Bardzo często mamy do czynienia z problemem, w którym występują ograniczenia. AE sam z siebie jest metodą optymalizacji bez ograniczeń i aby je uwzględnić, należy zastosować jedną z kilku możliwych metod:

- w przypadku ograniczeń „kostkowych” można sprawdzać, czy w trakcie inicjacji lub po działaniu operatorów genetycznych nie pojawiają się rozwiązania niedopuszczalne i je odpowiednio „przycinać” lub rzutować
- w przypadku bardziej skomplikowanych ograniczeń funkcyjnych można stosować transformację zmiennych – metody znane z tradycyjnej optymalizacji funkcji rzeczywistych
- w pewnych przypadkach (np. gdy ograniczenia wynikają z samej idei problemu) można zastosować odpowiednie rodzaje kodowania i specjalnie zaprojektowane w tym celu operatory genetyczne, które nie pozwolą na wygenerowanie rozwiązań niedopuszczalnych (np. w problemie TSP zakodowanie w postaci listy miast, krzyżowanie jako operator PMX i mutacja jako operator inwersji)

Algorytmy ewolucyjne
uwzględnienie ograniczeń

- metoda naprawy – polega na „naprawieniu”, czyli przekształceniu rozwiązania w taki sposób, aby było dopuszczalne, metoda taka w niektórych przypadkach może być kosztowna obliczeniowo („przycinanie” i rzutowanie na ograniczenie też można potraktować jako naprawę).
- metoda funkcji kary – gdy nie można zastosować innych metod, to należy stosować pogorszenie funkcji celu rozwiązań niedopuszczalnych tak, aby zostały wyeliminowane przez selekcję, granicznym przypadkiem takiego podejścia jest metoda „kary śmierci”, która prowadzi do natychmiastowej eliminacji rozwiązań niedopuszczalnych przez nadanie im maksymalnie nieodpowiedniej (np. $-\infty$ lub $+\infty$) wartości funkcji dopasowania. W przypadku AE funkcje kary mogą nie posiadać pochodnych lub być nieciągłe, inaczej niż w przypadku optymalizacji klasycznej.

Algorytmy ewolucyjne
kryterium stopu

Ewolucja nie ma naturalnego kryterium stopu (koniec świata?). Z tego też powodu nie mają jej i AE.
Ponieważ są one metodami heurystycznymi, więc nie potrafią odróżnić rozwiązania optymalnego i jego osiągnięcie (choć mało prawdopodobne) w ogólnym przypadku nie może być kryterium stopu. Można wyróżnić następujące kryteria

- **kryteria maksymalnego kosztu –** najczęściej po określonej liczbie wywołań funkcji oceny następuje zakończenie obliczeń. Jednym z jego wariantów (zresztą bardzo często używanym) jest narzucona maksymalna liczba iteracji/generacji;
- **kryterium osiągnięcia w trakcie symulacji zadowalającej wartości rozwiązania;**
- **kryterium minimalnej szybkości poprawy –** gdy od ostatniej poprawy rozwiązania minie odpowiednio dużo iteracji, program jest przerywany;
- **kryterium zaniku różnorodności populacji**, która jest jak wiadomo niezbędna do efektywnego działania ewolucji, jej zanik to praktycznie koniec możliwości znalezienia lepszych rozwiązań;
- **kryterium zaniku adaptowanego zasięgu operatora mutacji**, możliwe do zastosowania tylko tam, gdzie go zastosowano (np. w SE), mały zasięg uniemożliwia szerszą eksplorację przestrzeni i tym samym oznacza, że algorytm osiadł w ekstremum.

Podstawowe metody ewolucyjne

- **Prosty algorytm ewolucyjny (SGA)**: problemy binarne, dyskretne i rzeczywistoliczbowe, kodowanie binarne z binarnymi operatorami krzyżowania i mutacji, selekcja ruletkowa, obecnie rzadko używane w czystej postaci w praktyce.
- **Strategie ewolucyjne (SE)**: problemy rzeczywistoliczbowe, kodowanie rzeczywistoliczbowe, operatory mutacja z rozkładem normalnym i krzyżowanie uśredniające lub równomierne, selekcja rangowa lub turniejowa z sukcesją (μ,λ) lub $(\mu+\lambda)$, adaptacja operatora mutacji, używane w dalszym ciągu w praktyce.
- **Programowanie ewolucyjne (PE)**: ewoluujące automaty skończone, kodowanie w postaci grafów przejść między stanami – np. macierzowe, operatory to 5 rodzajów mutacji, selekcja – zastępowanie osobników gorszych lepszymi, zastosowania dość specyficzne, rzadko uzywane w czystej postaci.
- **Programowanie genetyczne (PG)**: ewoluujące programy kodowane w postaci drzewiastej zawierającej elementarne fragmenty kodu, operatory to specjalizowane krzyżowanie i mutacja dostosowane do operacji na drzewach, selekcja ruletkowa lub nowocześniejsze metody, zastosowania również specyficzne i skomplikowane – generowanie prgramów/algorytmów rozwiązujących problemy, wirusy komputerowe, antywirusy, ...

Inne metody ewolucyjne: algorytmy memetyczne i przeszukiwanie rozproszone

- **Algorytmy memetyczne (AM)**: mogą zawierać elementy wszystkich wymienionych wcześniej metod ewolucyjnych a także innych metod: heurystycznych, zachłannych i klasycznych, mają możliwość uczenia się w trakcie działania i dziedziczenia nauczonych informacji (stąd nazwa, mem – jednostka informacji dziedziczonej niegenetycznie, podobnie jak gen – dziedziczony tradycyjnie), są dostosowane do rozwiązania konkretnego problemu (wykorzystują wiedzę o rozwiązywanym problemie) ze specyficznym kodowaniem i operatorami, selekcja turniejowa lub selekcje z SI, zastosowania – wszelkie skomplikowane problemy obliczeniowe, w których potrzebna jest optymalizacja.
- **Przeszukiwanie rozproszon**e (scatter search, **SS**) – to metoda w której zrezygnowano z losowości, wszelkie operację są wykonywane deterministycznie, losowość może być obecna tylko przy generacji populacji startowej lub w użytych metodach optymalizacji lokalnej, jako operator modyfikujący rozwiązania występuje tu jedynie operator mieszania rozwiązań, tworzący nowe przez zestawienie cech dwójek, trójek, czwórek, ... wybranych rozwiązań, dodatkowo trakcie działania metody można korzystać z innych metod optymalizacji, poprawiających posiadane rozwiązania, najlepsze z wytworzonych osobników przechodzą do kolejnej iteracji.

Inne metody ewolucyjne:
ewolucja różnicowa

- **Ewolucja różnicowa (DE)** - jest jedną z nowszych metod ewolucyjnych. Najlepiej nadaje się do optymalizacji problemów rzeczywistoliczbowych, choć istnieją wersje binarne i całkowitoliczbowe. Metoda działa bardzo efektywnie, a jest bardzo prosta. Ogólny algorytm jest zbliżony do typowego algorytmu ewolucyjnego, jednak szczegółowe operacje są specyficzne dla metody. W typowej wersji osobniki populacjo o rozmiarze S są n-wymiarowymi wektorami liczb rzeczywistych. Mutacja polega na wytworzeniu nowego rozwiązania (mutanta v) na bazie kilku (najczęściej 3, czasem 5 i więcej) wylosowanych (różnych!) osobników:

$$v_i = x^1_i + F*(x^2_i - x^3_i) \text{ lub } v_i = x^1_i + F*(x^2_i - x^3_i) + F*(x^4_i - x^5_i)$$

Krzyżowanie w DE polega na zastosowaniu metody krzyżowania równomiernego, zwanego także krzyżowaniem z maską (losowy wybór współrzędnych od rodzica x^0 lub mutanta v)

$$u_i = \begin{cases} v_i & \text{jeśli } U(0,1) < CR \\ x^0_i & \text{w przeciwnym przypadku} \end{cases}$$

gdzie: F – współczynnik amplifikacji (wzmocnienia), najczęściej ok. 0,5; CR – prawdopodobieństwo mieszania cech, u – osobnik „próbny”, i – indeks rozpatrywanej współrzędnej rozwiązania.
Selekcja – osobnik próbny u jest porównywany z rodzicem x^0, lepszy przechodzi do kolejnej generacji.

Algorytmy ewolucyjne

selekcja osobników do populacji potomnej
ćwiczenie do domu

W algorytmie ewolucyjnym maksymalizującym pewną funkcję celu mamy (pod)populację rodzicielską o następujących wartościach funkcji celu: {5,5; 4,5; 1,2; 3,3}. W wyniku działań operatorów genetycznych powstała (pod)populacja potomna o wartościach funkcji celu: {6,1; 4,9; 3,2; 3,4}. Jak będzie wyglądać skład nowe populacji rodzicielskiej jeśli do wyboru osobników do niej użyto (zakładamy tę samą liczność (pod)populacji):

1) metody selekcji najlepszych osobników i strategii $(\mu+\lambda)$,
2) metody selekcji najlepszych osobników i strategii (μ, λ),
3) metody selekcji deterministycznej i strategii (μ, λ)?

Algorytm przeszukiwania z tabu
tabu search (TS)

Metoda **tabu search** jest kolejnym krokiem rozwoju algorytmów optymalizacji lokalnej (podobnych do algorytmu wzrostu), polegających na przeszukiwaniu sąsiedztwa ostatnio zaakceptowanego rozwiązania. Metoda szczególnie nadaje się do rozwiązywania problemów kombinatorycznych, dyskretnych, zadania ciągłe mogą być rozwiązywane dopiero po odpowiedniej dyskretyzacji.

Jak wiadomo metody optymalizacji lokalnej dość łatwo wpadają w optima lokalne lub ulegają zapętleniu i w związku z tym ich efektywność nie jest duża. Badacze usiłują poprawić te metody, wprowadzając mechanizmy umożliwiające opuszczenie optimów lokalnych znane np. z symulowanego wyżarzania, algorytmu „małych światów", metody wielostartu lub restartu, makromutacje zapobiegające stagnacji itp. Jedną ze skuteczniejszych metod jest też wprowadzenie reguł sterujących przebiegiem algorytmu. Reguły te mają w tym przypadku postać zakazów pewnych ruchów, wyborów pewnych rozwiązań ponownie i w związku z tym zostały nazwane „tabu" od nazwy zakazów religijnych (czasowych lub nieograniczonych czasowo) dotyczących przebywania w pewnych miejscach, stykania się z pewnymi ludźmi, wykonywania pewnych czynności, występujących pod tą nazwą w religiach Polinezji.

Algorytm przeszukiwania z tabu
tabu search (TS)

W algorytmie zakazy te są tworzone przez samą metodę na podstawie istniejących w niej metareguł, czyli są przykładem „uczenia się” algorytmu (algorytm posiada struktury pamięci do przechowywania odpowiednich faktów i reguł), innymi słowy adaptacji do zaistniałych warunków.
Dodatkowo algorytm został również wyposażony w reguły przełamywania tabu (tzw. kryterium aspiracji), jeśli zaistniała sytuacja jest wyjątkowo korzystna i odrzucenie dobrego rozwiązania byłoby ewidentną stratą dla obliczeń lub gdy przez dłuższy czas nie można znaleźć nowego dopuszczalnego rozwiązania bo wszystkie lub prawie wszystkie możliwe „sąsiednie” rozwiązania, możliwe do wygenerowania przez operator przeszukiwania sąsiedztwa, są na liście tabu.

Algorytm przeszukiwania z tabu

Reguły tabu mogą występować na trzech poziomach działania metody (nie wszystkie muszą wystąpić w każdej realizacji algorytmu) lub też jako różne rodzaje pamięci działań metody:

- pamięć krótkotrwała pamięta ostatnio odwiedzone rozwiązania i wprowadza (najczęściej czasowy) zakaz powrotu do nich – jest to dość naturalny zakaz, zabezpieczający metodę przed zapętleniem;
- pamięć średnioterminowa wzmacniająca reguły umożliwiające przeszukiwanie bardziej obiecujących obszarów przestrzeni rozwiązań;
- pamięć długoterminowa zawiera reguły dywersyfikujące (różnicujące) umożliwiające przeszukiwanie nowych obszarów przestrzeni rozwiązań.

Dane w pamięci mogą być przechowywane w sposób bezpośredni np. jako gotowe rozwiązania lub pośrednio np. jako ślady wykonywanych działań (np. przestawień elementów rozwiązań) lub częstości występowania pewnych zdarzeń, czyli pewne statystyki procesu przeszukiwania.

W przypadku bardziej skomplikowanych reguł tabu, opartych na częstotliwościach wystąpień pewnych atrybutów, reguły tabu przybierają raczej postać funkcji kary, osłabiania lub wzmacniania pewnych akcji proporcjonalnie do stopnia naruszenia zakazu niż bezwzględnej eliminacji danego działania.

Algorytm przeszukiwania z tabu
schemat działania

1. Inicjalizacja danych w strukturach pamięci algorytmu
2. Wybór/wylosowanie rozwiązania startowego x_o oraz jego ocena
3. Rozwiązanie startowe staje się najlepszym aktualnie rozwiązaniem $x^*=x_o$
4. do {
    5. Wygenerowanie *n* nowych zmodyfikowanych rozwiązań w sąsiedztwie x^*
    6. Ocena nowych rozwiązań z uwzględnieniem wpływu reguł tabu (funkcje kary, odrzucenie rozwiązań zakazanych,...)
    7. Wybór x' najlepszego z wygenerowanych rozwiązań
    8. Jeśli jest ono lepsze od x^* to $x^*=x'$
    9. Aktualizacja danych w pamięci: reguł tabu, częstości przejść, …
    }
10. while (warunek stopu)

Algorytm przeszukiwania z tabu
schemat działania

Wygenerowane rozwiązania (i sposób ich wygenerowania) są oceniane przy pomocy funkcji celu oraz reguł tabu. W tym kroku niektóre rozwiązania mogą zostać odrzucone lub słabiej ocenione w efekcie zadziałania reguł tabu. W wyniku zadziałania tzw. kryterium aspiracji rozwiązania „będące na indeksie”, lecz bardzo dobre (lub najlepsze) mogą być „ułaskawione” i dopuszczone do dalszych etapów metody.

Z pozostałych po ocenie rozwiązań, wybierane jest najlepsze x'.

Jeśli jest ono lepsze od x^*, to staje się nowym x^*.

Aktualizacja struktur pamięci polega na zmniejszeniu indeksu tabu rozwiązań i ruchów, które się nie pojawiły, zwiększenia indeksu tym, które się pojawiły, zaakceptowane, odpowiednio powiększenia/zmniejszenia statystyk zadziałania operatora (-ów) modyfikujących rozwiązania.

Kryterium stopu metody jest typowe, jak dla innych metod heurystycznych.

Algorytm przeszukiwania z tabu
schemat działania

W trakcie inicjalizacji struktur pamięci wszelkie reguły tabu są zerowane, ich ustawienie nastąpi w trakcie działania metody.

Do wygenerowania rozwiązania początkowego można użyć losowania lub też innych bardziej wyspecjalizowanych metod (algorytmu zachłannego, metody losowej+algorytm optymalizacji lokalnej, wybór najlepszego z kilku wylosowanych, itp.). Rozwiązanie to staję się początkowym bieżącym najlepszym rozwiązaniem.

Wygenerowanie rozwiązań zmodyfikowanych w sąsiedztwie najlepszego to oczywiście konieczność użycia pewnych specjalizowanych operatorów (inteligentnych?). W metodzie standardowo jest jeden, ale heurystyki są po to, żeby je rozwijać...

Algorytm przeszukiwania z tabu
szczegóły techniczne

Do przechowywania całych rozwiązań potrzebna jest oczywiście macierz, wektor lub lista o odpowiedniej wielkości, np. o długości tabu (kolejka). W zasadzie można również przechowywać bardziej ogólne informacje np. o „klasach” lub regionach bardzo zbliżonych rozwiązań (niewiele się różniących).

Do przechowywania informacji o zmianach w rozwiązaniu powodowanych przez operator sąsiedztwa potrzebna będzie macierz kwadratowa (lub kilka takich macierzy) o rozmiarze liczby elementów rozwiązania (np. liczby miast), w której po zastosowaniu danej zmiany wstawia się do odpowiednich komórek liczby mówiące jak długo takie przejście jest zabronione. Macierze takie mogą zawierać znacznie więcej elementów (kilka macierzy lub macierz bardziej złożonych struktur danych), zależnie od wykorzystywanych atrybutów. Należy też zauważyć, że ich odświeżanie po każdej iteracji może być kosztowne obliczeniowo.

Algorytm przeszukiwania z tabu
rozszerzenia metody

- czasy obowiązywania tabu mogą być zmienne w trakcie działania algorytmu
- przy nieskomplikowanych operatorach sąsiedztwa, postulowano przejrzenie wszystkich sąsiadów, jednak w bardziej skomplikowanych przypadkach jest to niemożliwe i stosuje się przejrzenie pewnej ich części (np. wylosowanej)
- stosuje się bardzo różnorodne rodzaje reguł tabu, oparte na różnych danych możliwych do uzyskania od procesu optymalizacji
- stosuje się różne rodzaje kryteriów aspiracji np. zmniejsza się okres trwania tabu
- stosuje się metody hybrydowe np. połączenia z algorytmami ewolucyjnymi – jako wykorzystywane tam algorytmy optymalizacji lokalnej
- pomysły wykorzystania reguł tabu wykorzystuje się też do sterowania innymi procesami, nie tylko optymalizacyjnymi

Algorytmy mrówkowe

Algorytmy mrówkowe, podobnie jak wiele innych heurystyk, mają źródło w biologii i są jednymi z pierwszych algorytmów „odzwierzęcych” (obecnie jest ich kilkanaście). W tym przypadku obserwacje działania pojedynczych mrówek, jak i całego mrowiska oraz sposobów komunikacji między owadami zainspirowały badaczy do wykorzystania ich jako pewnej heurystyki, na podstawie której można stworzyć algorytm optymalizacyjny.

W omawianym przypadku największe znaczenia ma przekazywanie informacji między owadami (np. dotyczących drogi do pożywienia) drogą sygnałów chemicznych (feromonowych) – mrówki starają się podążać po śladzie feromonowym pozostawionym przez inne mrówki. Zauważono również, że mrówki po pewnym czasie wybierają coraz krótszą drogę do pożywienia, co oznacza, że występuje u nich mechanizm optymalizacji drogi, dzięki większej koncentracji feromonu na krótszej drodze niż na dłuższej – jest to efekt dodatniego sprzężenia zwrotnego: wybieranie krótszej drogi i wzrost stężenia feromonu oraz częstości przechodzenia mrówek zachęca mrówki do jeszcze częstszego odwiedzania, itd.

Algorytmy mrówkowe

Sposób działania algorytmu mrówkowego powoduje, że dość intuicyjnie można go zastosować do problemów związanych z grafami, optymalizacją drogi (np. TSP), nie znaczy to jednak, że metoda może służyć tylko do tego celu. Stworzony algorytm jest metodą uniwersalną i może być wykorzystywany do rozwiązywania dowolnych zadań dyskretnych - po odpowiednich modyfikacjach np. przedstawieniu problemu w postaci grafu przejść.

Algorytmy mrówkowe
algorytm metody

l. Ustalenie początkowego poziomu feromonu w miejscach (krawędziach grafu) odpowiadających przejściom między kolejnymi elementami rozwiązywanego problemu
do {
2. Odparowanie feromonu (czyli zmniejszenie wartości wag feromonowych krawędzi).
3. Wylosowanie punktu startu agenta-mrówki.
4. Wylosowanie ścieżek na podstawie poziomu feromonu (wag) na przejściach między kolejnymi elementami rozwiązania.
5. Naniesienie feromonu (zwiększenie wag) krawędzi wykorzystanych w rozwiązaniu.
6. Operacje dodatkowe – np. zbalansowanie feromonu ścieżek (próba wyjścia z optimum lokalnego), dodatkowe wzmocnienie pewnych krawędzi, itp.
}
while (warunek stopu)

Algorytm mrówkowy
schemat działania

Schemat działania AS wymaga doprecyzowania kilku parametrów:

- liczba mrówek – wpływa na dokładność i czas działania metody
- poziom feromonu początkowego (wagi feromonowe początkowe krawędzi) – nie może być zbyt duży, gdyż działanie metody będzie przypominać losową generacje rozwiązań – nie zaistnieje dodatnie sprzężenie zwrotne, umożliwiające poprawę rozwiązań
- odparowanie feromonu (zmniejszenie wag feromonowych) – również musi być na odpowiednim poziomie: zbyt duże przetnie sprzężenie zwrotne, zbyt małe może doprowadzić do dominacji lokalnie lepszych rozwiązań
- punkt startowy losuje się dowolnie, natomiast wyborem kolejnych ścieżek zarządza prawdopodobieństwo wyboru drogi, które opisane jest wzorem:

$$p_{i,j}^{k}(t)=\frac{[\tau_{i,j}(t)]^{\alpha}*[\eta_{i,j}]^{\beta}}{\sum_{l=1}^{n}\left([\tau_{i,l}(t)]^{\alpha}*[\eta_{i,l}]^{\beta}\right)}$$

$p^{k}_{i,j}(t)$ – prawdopodobieństwo wyboru drogi; $\tau_{i,j}(t)$ – natężenie feromonu, $\eta_{i,j}$ – waga krawędzi wynikająca z optymalizowanego zadania, np. odwrotność odległości między miastami w przypadku TSP; α, β - parametry określające wpływ feromonu i wagi krawędzi wynikającej z zadania na p. wyboru drogi; i, j - indeksy elementów rozwiązania (np. miast); k – numer mrówki; n – liczba punktów dopuszczalnych, które można włączyć do rozwiązania w danym momencie.

Algorytm mrówkowy
schemat działania cd.

- następny krok do przejścia jest losowany na podstawie prawdopodobieństwa z poprzedniego slajdu metodą ruletkową (proporcjonalną), parametry α, β umożliwiają sterowanie znaczeniem feromonu (informacji uzyskanej w trakcie działania algorytmu) i wagi pochodzącej od zadania, czasem konieczne może być uwzględnienie tu ograniczeń
- aktualizacja feromonu na wygenerowanej ścieżce wykonywana jest według następującego wzoru:

$$\tau_{i,j}(t+1)=(1-\rho)*\tau_{i,j}(t)+\sum_{l=1}^{n}\Delta\tau_{i,j,l}(t)$$

$\tau_{i,j}(t)$ – natężenie feromonu; ρ - współczynnik parowania feromonu, zazwyczaj 0,5...0,9; $\Delta\tau_{i,j,l}(t)$ – feromon zostawiony przez mrówki wykorzystujące dane połączenie.

- aktualizacja feromonu jest jednym z najczęściej modyfikowanych i rozwijanych kroków algorytmu, odkładanie feromonu przez mrówki jest jedyną formą komunikacji mrówek i metoda wykonywania tego kroku ma decydujący wpływ na pojawienie się informacyjnego sprzężenia zwrotnego między mrówkami oraz efektywność algorytmu; możliwych jest kilka metod odkładania feromonu:
  - z poziomem stałym: mrówka odkłada jednakową ilość feromonu na wykorzystanych ścieżkach grafu przejść;
  - z poziomem średnim: ilość odkładanego feromonu zależy od wagi (np. długości) krawędzi;

Algorytm mrówkowy
schemat działania cd.

- z poziomem cyklicznym: ilość odkładanego feromonu zależy od wagi (np. długości) całego rozwiązania (w tym przypadku aktualizacja musi być koniecznie osobnym punktem algorytmu, w pozostałych dwóch aktualizacje można wykonywać na bieżąco wraz z wyborem kolejnych ścieżek);
- często wprowadza się pojęcie **elity** – zbioru najlepszych ścieżek, rozwiązania, a właściwe ich krawędzie składowe są dodatkowo punktowane za posiadanie ich fragmentów przez dodanie dodatkowych porcji feromonów (ta część algorytmu może też odpowiadać pkt. 6 metody);
- w tzw. systemach kolonii mrówkowych stosuje się połączenie 2 rodzajów aktualizacji globalnego i lokalnego, czyli w zasadzie aktualizacji z poziomem cyklicznym i aktualizacji z poziomem stałym lub średnim z tą różnicą, że aktualizacji globalnej dokonuje tylko najlepsza mrówka w danej iteracji, aktualizacji lokalnej dokonują wszystkie według wzoru:

$$\Delta\tau'_{i,j}(t)=(1-\xi)*\Delta\tau_{i,j}(t)+\xi*\tau_0$$

$\Delta\tau_{i,j}(t)$ – feromon jak w metodzie z poziomem cyklicznym równy $1/L_k$ (odwrotność długości/wagi drogi);
ξ - parametr z zakresu (0,1); τ_0 – parametr o znaczeniu zbliżonym do startowego poziomu feromonu.

Algorytmy mrówkowe
rozszerzenia metody

- **elitarny system mrówkowy (EAS)** – najlepsze globalnie rozwiązanie pozostawia dodatkową porcję feromonu po każdej iteracji.
- **system mrówkowy Max-Min (MMAS) -** feromon jest pozostawiany tylko przez najlepszą globalnie mrówkę lub najlepszą w danej iteracji. Wszystkie krawędzie są ustawiane na wartości startowe (reset) w przypadku stagnacji algorytmu.
- **system kolonii mrówkowych (ACO)** – prezentowane wcześniej – najlepsza globalnie mrówka jest odpowiedzialna za aktualizację globalną, pozostałe za aktualizację lokalną.
- **rankingowy system mrówkowy (ASRank)** – wszystkie rozwiązania/mrówki otrzymują rangę powiązaną z jakością wyniku, ilość pozostawianego feromonu jest ważona współczynnikiem proporcjonalnym do wagi rozwiązania, lepsze zostawiają więcej, gorsze niewiele feromonu.
- **ortogonalny ciągły system mrówkowy (COAC)** metoda rozwiązywania zadań ciągłych ze specjalnym sposobem przeglądania wybranych fragmentów zdyskretyzowanej przestrzeni poszukiwań (metoda ortogonalnego przeglądu).
- **rekurencyjny system mrówkowy (RACO)** – rekurencja polega na sprawdzaniu sąsiednich miast do tych odwiedzonych przez najlepsze mrówki (coś w rodzaju dodatkowej optymalizacji lokalnej).

Dziękuję za uwagę.