Wstęp do inteligencji komputerowej – zajęcia nr 5

Jarosław Stańczak
WSISiZ

Zagadnienia związane ze złożonością obliczeniową problemów

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

Heurystyki w rozwiązywaniu problemów NP i optymalizacji funkcji wielomodalnych

- proste metody zachłanne
- symulowane wyżarzanie

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.

Złożoność obliczeniowa problemów

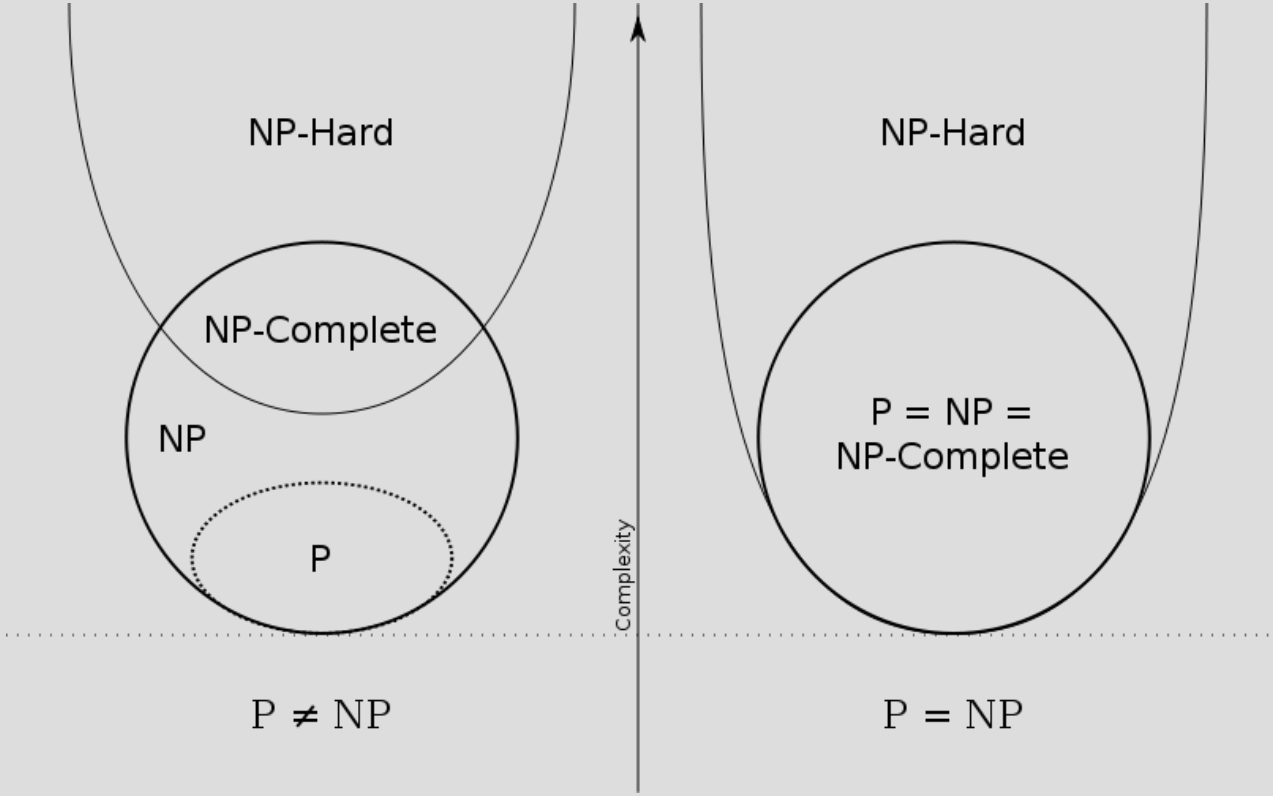
Komputery, które używamy, są pewnymi ograniczonymi przedstawicielami deterministycznych maszyn Turinga, gdyż ich programy są zawsze ograniczone, czyli ich taśma nie może być nieskończenie długa.
Niedeterministyczna maszyna Turinga niestety nie jest realizowalna przy obecnym stanie wiedzy. Jej działanie 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≠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



[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”.]

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.
A ile bedą wynosić czasy obliczeń dla n=100?

[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, metody trajektorii cząstki, metoda simplex...

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.

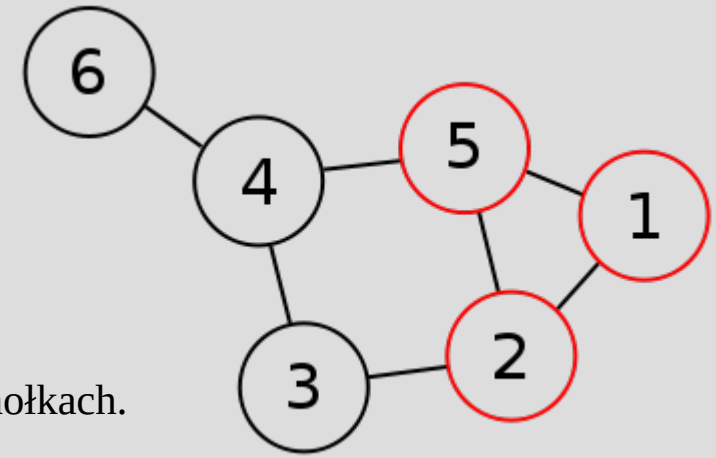
Problem maksymalnej kliki

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.



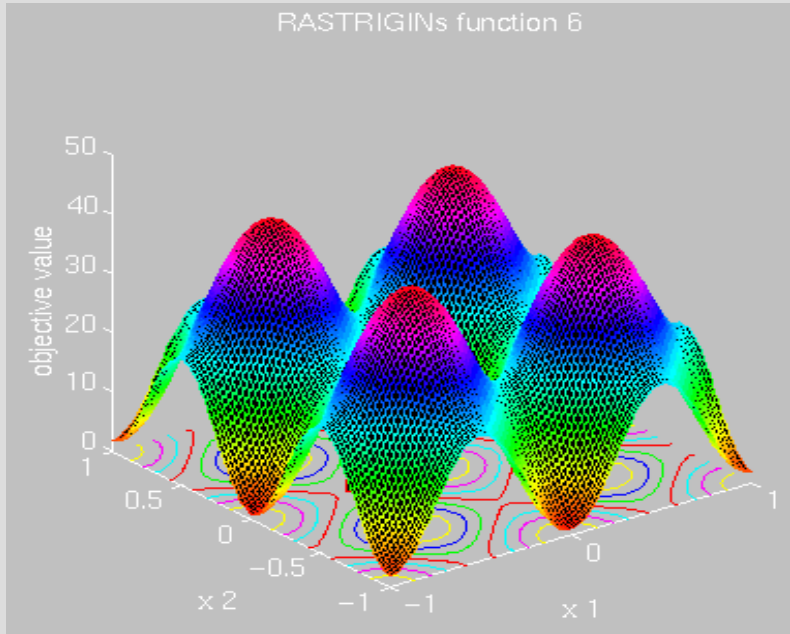
Przykład max. kliki o 3 wierzchołkach.
[Na podst. Wikipedii]

Funkcja Rastrigina

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

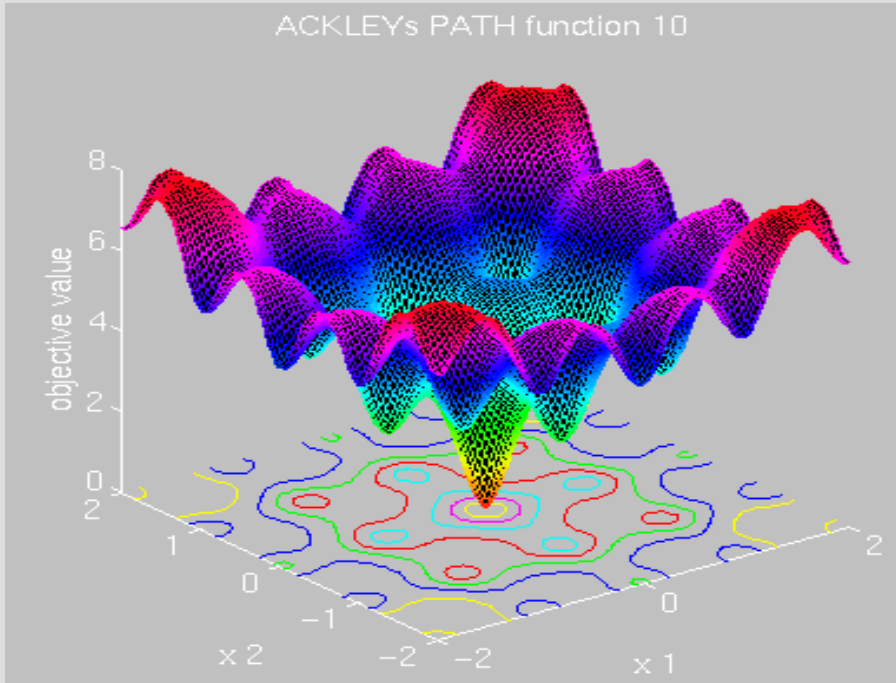
$$f(x_i) = A*n + \sum_{i=1}^{n} x_i^2 - A*\cos(\omega * x_i)$$

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



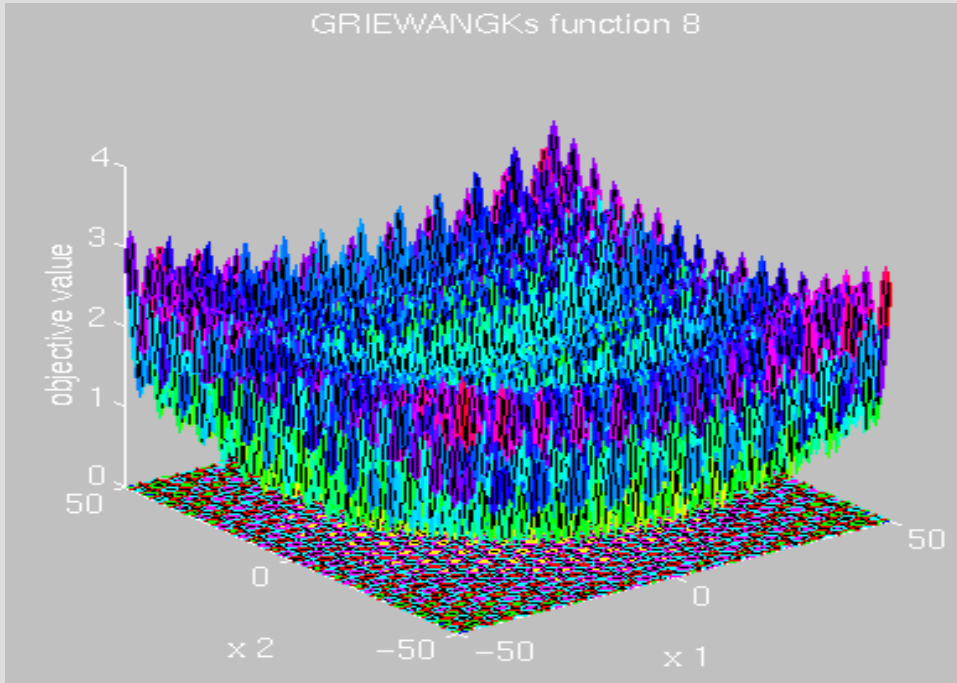
Funkcja Ackleya

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



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, istnieją jednak przykłady optymalnych algorytmów zachłannych, są to np.:

- 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.

Przegląd metod lokalnych - algorytmy zachłanne

Przykładem algorytmu zachłannego niedającego gwarancji uzyskania optimum może być prosta metoda rozwiązywania problemu TSP (TSP jest problemem klasy NP), zazwyczaj daje ona rozwiązanie o ok. 25% gorsze od optimum, w pewnych przypadkach jednak nawet o 100% gorsze.

Prosty algorytm zachłanny dla problemu TSP:
1. Suma dróg S(0)=0, t=0;
2. Wylosuj jedno z miast na liście miast do odwiedzenia;
3. Usuń wybrane miasto z listy, jeśli lista jest pusta, to zwróć S(t) i koniec;
4. Znajdź wśród pozostałych na liście miast najbliższe do ostatnio wybranego;
5. S(t+1)=S(t)+d(c(t-1),c(t)) (d – odległość między ostatnio wybranymi miastami);
6. Idź do pkt. 3.

Algorytm ten ma kwadratową złożoność obliczeniową względem liczby miast.

Przegląd metod lokalnych - algorytmy zachłanne

Podobnych metod zachłannych jest bardzo wiele, dla problemu poszukiwania maksymalnej kliki w grafie można wymyślić podobną metodę

Prosty algorytm zachłanny dla problemu poszukiwania maksymalnej kliki w grafie:
1. Rozmiar kliki $K(0)=0$, $t=0$;
2. Wylosuj jeden z wierzchołków grafu jako startowy;
3. $t=t+1$, $K(t)=K(t-1)+1$;
4. Usuń wybrany wierzchołek z listy wierzchołków, jeśli lista jest pusta, to zwróć $K(t)$ i koniec;
5. Znajdź wśród pozostałych wierzchołków taki, który jest połączony ze wszystkimi wybranymi;
6. Jeśli takiego wierzchołka nie ma, zwróć $K(t)$ i koniec, w przeciwnym wypadku idź do pkt. 3.

Algorytm ten ma kwadratową złożoność obliczeniową.
Algorytm zachłanny (nieoptymalny) może służyć do rozwiązywania problemu, jednak z uwagi na zazwyczaj słabe parametry raczej wykorzystywany jest do generacji rozwiązań startowych dla innych metod lub lokalnego poprawiania posiadanych rozwiązań uzyskanych innymi metodami.

Algorytm przeszukiwania lokalnego
algorytm (największego) wzrostu
przykład najprostszej heurystyki

Algorytm wzrostu to przykład najprostszej heurystyki obliczeniowej nienależącej do grupy metod zachłannych. Można go uznać za „praprzodka” właściwie wszystkich metod heurystycznych, w których rozwinięto różne aspekty tej metody, mające tu najprostsze możliwe warianty:

- modyfikacja operatorów (np. wprowadzenie wiedzy o problemie)
- wielostart
- wprowadzenie populacji rozwiązań
- zrównoleglenie obliczeń
- wprowadzenie wyrafinowanych metod akceptacji/selekcji nowych rozwiązań
- pozyskiwanie wiedzy o charakterze rozwiązywanego problemu
- adaptacja parametrów
- połączenie z innymi metodami optymalizacji heurystycznymi i klasycznymi

Są to przykłady sposobów rozwinięcia tej metody.

Algorytm przeszukiwania lokalnego
algorytm (największego) wzrostu
przykład najprostszej heurystyki

begin
    wylosuj rozwiązanie początkowe x_s
    oceń x_s
    $x_0 := x_s$
    repeat
        utwórz n nowych rozwiązań $x_1 ... x_n$ przez losową modyfikację x_0
        oceń rozwiązania $x_1 ... x_n$
        wybierz najlepsze rozwiązanie x^* z $x_1 ... x_n$
        jeśli x^* lepsze od x_0 to $x_0 := x^*$
    until spełniony_warunek_stopu
    wypisz najlepsze x_0
end

Algorytm przeszukiwania lokalnego
algorytm (największego) wzrostu z wielostartem
przykład ulepszonej heurystyki

```
begin
    t:=0
    repeat
        local:=FALSE
        wybierz rozwiązanie początkowe x_s
        oceń x_s
        x_0:=x_s
        repeat
            wybierz n nowych rozwiązań x_1...x_n przez modyfikację 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* jeśli nie to local:=TRUE
        until local
        zapamiętaj najlepsze x_0
        t:=t+1
    until t=MAX
    wypisz najlepsze x_0
end
```

Algorytm przeszukiwania lokalnego ze zmiennym sąsiedztwem

jeszcze inna możliwość ulepszenia prostej heurystyki

begin
    k:=1
    wybierz rozwiązanie początkowe x_0
    oceń x_0
    repeat
        utwórz nowe rozwiązanie x_1 przez modyfikację x_0 w sąsiedztwie o rozmiarze k
        oceń rozwiązanie x_1
        jeśli x_1 lepsze od x_0 to x_0:=x; k:=1;
        jeśli nie to k:=k+1;
    until k=MAX
    wypisz najlepsze x_0
end

Algorytm (największego) wzrostu
inne 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.

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).

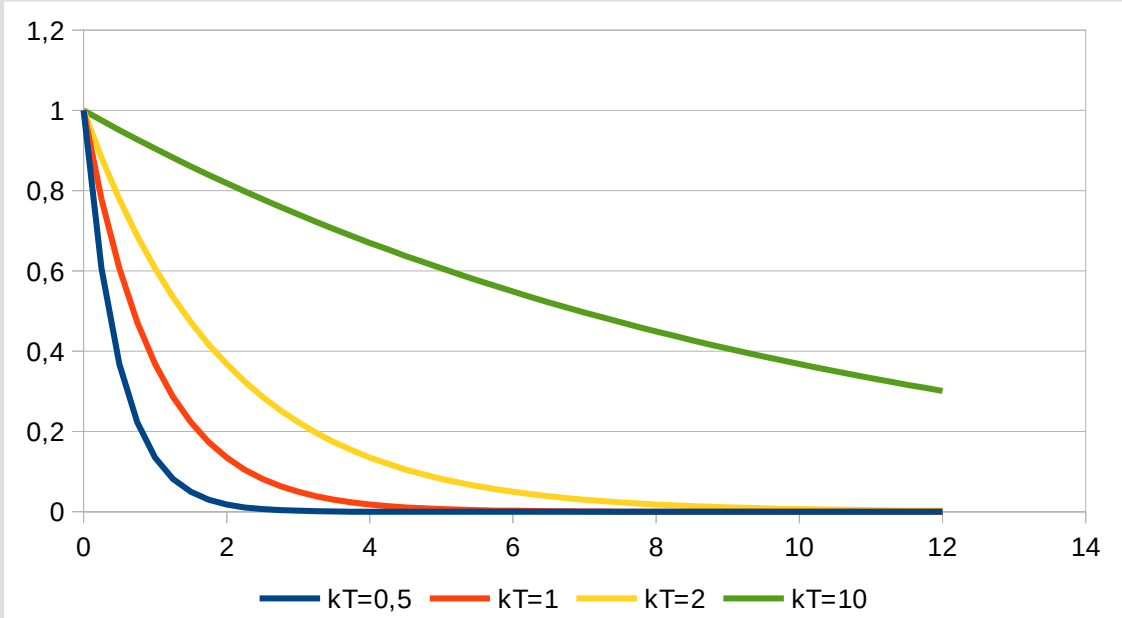
Algorytm Metropolisa

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) //MAX – liczba powtórzeń przy stałej temperaturze
  $T(n+1)=g(T(n))$ //zmiana temperatury
  $n=n+1$
}
while (warunek końca)

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.

Koniec

Dziękuję za uwagę.