PROBLEMY OPTYMALIZACJI

PROBLEM:

Dany jest zbiór (*przestrzeń rozwiązań*) $S \subset R^n$ **oraz**

***funkcja* (*celu*, kryterium optymalizacyjne)** $F : S \to R^n$.

Wyznaczyć element (optymalny, minimalny) $x^* \in S$ **taki, że**

$$F(x^*) = \min\{F(x) : x \in S\}.$$

Brak ogólnych metod rozwiązywania takich problemów (także twierdzeń o istnieniu rozwiązania).

Nakładając pewne ograniczenia (założenia) na *F* lub *S* generujemy problemy optymalizacji: *ciągłej*, *wypukłej, liniowej, dyskretnej, kombinatorycznej, z regularną funkcją*, itp. Badając je opracowano szereg specyficznych metod i algorytmów.

Klasyczne metody optymalizacji

Klasyczne metody poszukiwania optymalnych rozwiązań dzielą się na:

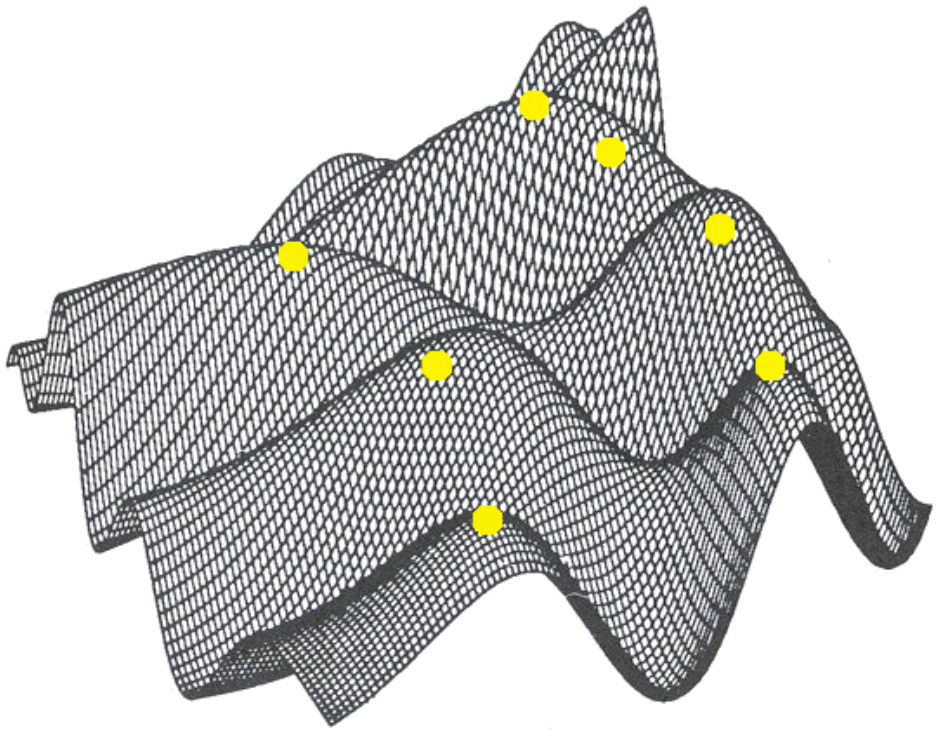
- ***analityczne***,
- ***przeglądowe*** deterministyczne (dokładne, przybliżone - poszukiwań),
- ***losowe***.

W ***metodach analitycznych*** poszukujemy lokalnych ekstremów funkcji rozwiązując układy równań (zwykle nieliniowych). Metody te maja liczne wady – są złożone, nie zawsze dadzą się zastosować, często „utykają” w lokalnych ekstremalnch funkcji.

Metody przeglądowe polegają, w uproszczeniu, na obliczaniu wartości funkcji celu na całej (lub części) przestrzeni rozwiązań po kolei. Oczywiście dla dużych przestrzeni jest to bardzo nieefektywne i często po prostu niemożliwe.

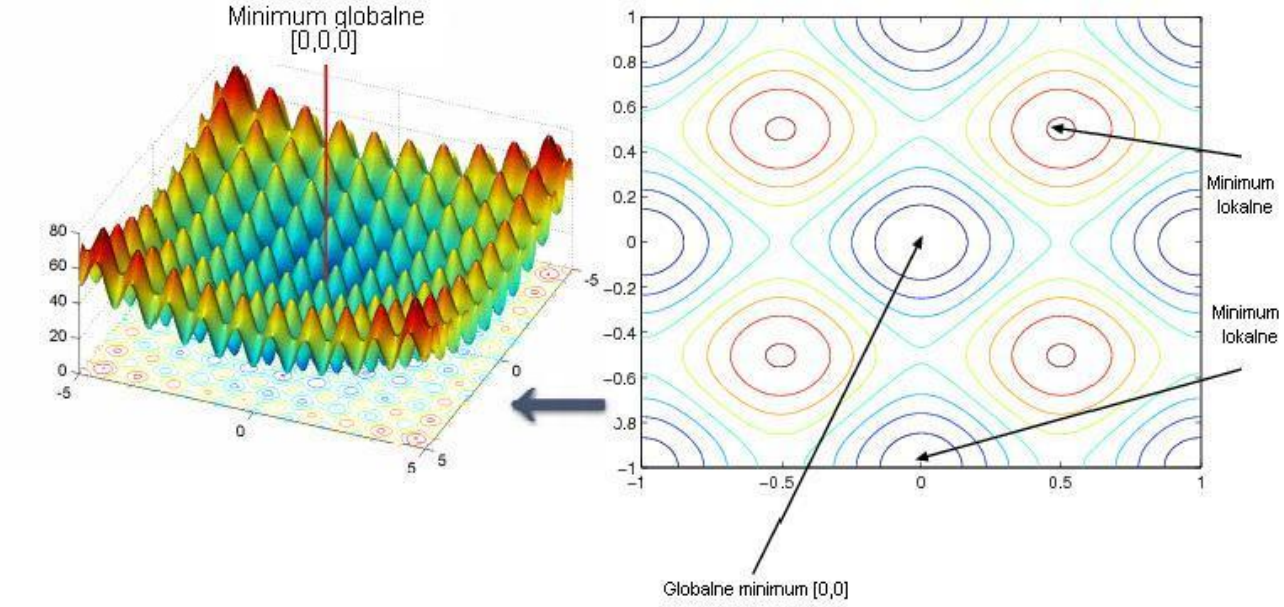
Metody losowe polegają na losowym przeszukiwaniu przestrzeni rozwiązań i zapamiętywaniu otrzymanych wyników, z których potem wybiera się najlepsze. Proste, są mało efektywne. Główna wada - brak powtarzalności obliczeń.

Wiele spośród metod optymalizacji opartych jest na idei ukierunkowanego poszukiwania optimum, a więc są jakąś odmianą metody najszybszego spadku. Źródłem problemów przy ukierunkowanej optymalizacji są głównie ekstrema lokalne (df. ekstremum).



Widać, jak trudne może być trafienie globalnego minimum.

Przedstawiony na rysunku wykres tzw. funkcji Rastrigina obrazuje trudności jakie napotkać można przy poszukiwaniu optimum. Funkcja ta posiada wartość najmniejszą w punkcie (0,0,0), jednak zanim algorytm przeszukiwania znajdzie to minimum globalne, może napotkać wiele minimów lokalnych.



Od problemu minimów lokalnych wolne są probabilistyczne metody optymalizacji (lecz mają niestety wiele innych wad).

Generalnie (ze względu na moc zbiory S),w literaturze wyróżnia się problemy:

- ciągłe (*S* – nieprzeliczalny)
- dyskretne (*S* równoliczny ze zbiorem liczb naturalnych)
- kombinatoryczne (*S* skończony).

Praktyczne problemy mają zasadniczo charakter kombinatoryczny, co niestety nie ułatwia ich rozwiązywania, bowiem cechuje je między innymi:

Globalizacja oraz rozwój nowych technologii wymagają z jednej strony rozwiązywania klasycznych problemów o ogromnych rozmiarach, a z drugiej generują zupełnie nowe złożone zagadnienia, których bardzo skomplikowane modele są niestety dalekie od rzeczywistości.

Program

1. **Problemy optymalizacyjne ciągłe i dyskretne (ograniczenia i metody rozwiązywania).**
2. **Problemy istotne z teoretycznego i praktycznego punktu widzenia: TSP, QAP, szeregowanie (flow shop, jednomaszynowe sumokosztowe), spełnialność, rozbicia, skojarzenia.**
3. **Własności problemów dyskretnych: „eksplozja" rozmiaru przestrzeni rozwiązań, wykładnicza liczba minimów lokalnych, nieregularność funkcji celu (przykłady: ft10, TSP z 1.2 milionem miast)**
4. **Algorytmy dokładne i przybliżone (błąd przybliżenia), algorytmy aproksymacyjne, twierdzenie „no free lunch".**
5. **Przegląd współczesnych metod konstrukcji algorytmów przybliżonych.**
6. **Metody lokalnych popraw (*k*-opt dla TSP, NEH, itp). Algorytm zstępujący.**
7. **Algorytmy metaheurystyczne:**
   (a) **przeszukiwanie z tabu,**
   (b) **symulowane wyżarzanie,**
   (c) **metody ewolucyjne.**
8. **Algorytmy hybrydowe:**
   (a) **scatter search, metoda ścieżek łączących,**
   (b) **metoda analizy minimów lokalnych.**
9. **Kierunki rozwoju (sztuczna inteligencja, wieloprocesorowość).**

Literatura:

1. **P. Brucker, Scheduling Algorithm, Springer, 2007.**
2. **T.H. Cormen, C.E.Leiserson, R.L. Rivest, Wprowadzenie do algorytmów, WNT, 1997.**
3. **C. Smutnicki, Algorytmy szeregowania, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2002.**
4. **V.V. Vazirani, Algorytmy aproksymacyjne, WNT, 2005.**
5. **Z. Michalewicz, D.B.Fogel, Jak rozwiązywać, czyli nowoczesna heurystyka, WNT, 2007.**

EGZAMIN

Klasyczne Problemy

Problem Komiwojażera (TSP)

Dane:

S={1,2,...,n} – zbiór miast,

$c_{i,j}$ - macierz odległości ($i, j \in S$).

Wyznaczyć najkrótszą drogę zawierającą każde miasto zbioru S.

Przykład 1:

Czas potrzebny do rozwiązania problemu komiwojażera w zależności od liczba miast (przy założeniu, że komputer przetwarza milion instrukcji na sekundę).

Ilość miast	10	50	100	300
Czas [mikrosekundy]	~3,6 * 10^6	~10^{16}	~10^{31}	~10^{623}

Dla porównania – liczba mikrosekund jaka upłynęła od wielkiego wybuchu, w którym narodził się nasz Wszechświat jest rzędu 10^{24} .

Problem przepływowy

$J=\{1,2,...,n\}$, *zbiór zadań*,

które mają być wykonane przy użyciu maszyn ze zbioru:

$M=\{1,2,...,m\}$, *zbiór maszyn*.

Każde zadanie $j \in J$ **jest ciągu operacji:**

$$j=(O_{j1},O_{j2},...,O_{jm}), \quad j=1,2,...,n.$$

Operacja O_{jk} **wykonywana jest na** k**–tej maszynie przez czas** $p_{jk} \geq 0$.

Ograniczenia (permutation flow shop):

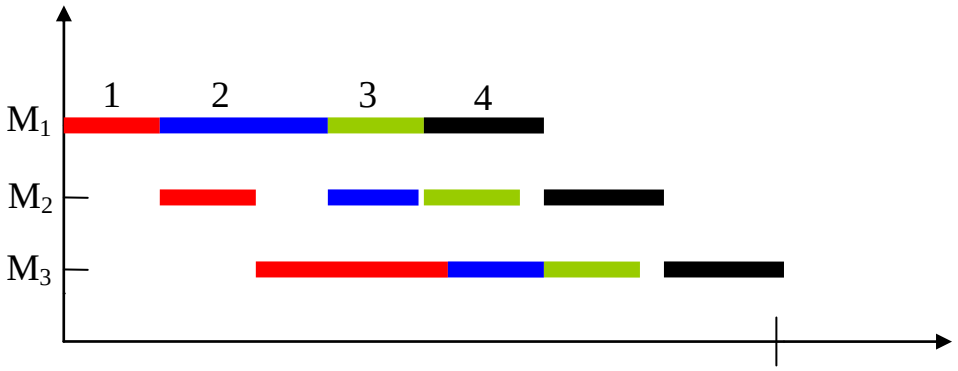
- **operację** O_{jk} **należy wykonać po zakończeniu** O_{jk-1}**, a przed rozpoczęciem** O_{jk+1} **(porządek tech.),**
- **kolejność wykonywania zadań na maszynach jest dowolna,**
- **maszyna może wykonywać w danej chwili tylko jedno zadanie,**
- **zadanie może być wykonywane w danej chwili tylko na jednej maszynie,**
- **wykonywanie zadania nie może być przerywane.**

Niech $\pi=(\pi(1),\pi(2),...,\pi(n))$ **będzie permutacją określającą kolejność wykonywania zadań.**

Zagadnienie optymalizacji polega na znalezieniu permutacji π^***, takiej, że**

$$C_{max}(\pi^*)= \min\{\pi: \ C_{max}(\pi)\},$$

gdzie $C_{max}(\pi)=\max\{j: \ C_{\pi(j)}\}$**, przy czym** $C_{\pi(j)}$ **jest terminem zakończenia wykonywania zadania** $\pi(j)$.



Przykład: $n=4$, $m=3$

Rozmiar przestrzeni rozwiązań:

$n!$ **- problem permutacyjny,**

$(n!)^m$ **– problem ogólny (nie permutacyjny).**

W roku 1973 Fisher podał przykład problemu przepływowego (10 zadań na 10 maszynach) nie permutacyjnego znanego w literaturze ft10, który doczekał się rozwiązania dokładnego dopiero po 25 latach (na PC dobry algorytm metaheurystyczny wyznacza to rozwiązanie po kilku sekundach - bez gwarancji optymalności).

Charakterystyka przestrzeni rozwiązań:

- **przy reprezentacji rozwiązań przez permutacje, wszystkich rozwiązań jest $(10!)^{10}$, tj. około $4*10^{65}$.**
- **gdyby każde rozwiązanie było punktem o rozmiarze 0.01 na 0.01 mm. to cała powierzchnia była by równa 3 powierzchniom Jowisza.**
- **najlepsze algorytmy przybliżone sprawdzają wartość funkcji celu dla co najwyżej od 10^6 do 10^8 punktów. odpowiada to na tej powierzchni kwadratowi o boku 10 cm x 10 cm.**

Analizie przestrzeni rozwiązań przykładu ft10 poświęcono kilka prac.

Praktyczne problemy optymalizacyjne cechuje:

- dyskretność
- silna NP-trudność, a co za tym idzie wykładniczo rosnący czas obliczeń algorytmów dokładnych,
- nieregularne, nieróżniczkowalne funkcje celu oraz nieliniowe ograniczenia,
- ogromny wymiar przestrzeni (liczba zmiennych decyzyjnych w praktyce wynosi od 100 do 10000),
- astronomiczna moc zbioru rozwiązań (dla niewielkiego przykładu problemu gniazdowego z 10 zadaniami i 10 maszynami wynosi $(10!)^{10}$,
- podobnie astronomiczna liczba ekstremów lokalnych, od 10^{50} do 10^{500},
- chaotyczny rozkład ekstremów w przestrzeni (tzw. szorstkość krajobrazu),
- silna, a przy tym przeciwstawna, zależność pomiędzy czasem obliczeń i jakością uzyskanego rozwiązania (odległością od optymalnego),
- brak efektywnych własności przyśpieszających obliczenia.

Stąd, do ich rozwiązywania stosuje się niemal wyłącznie algorytmy przybliżone.

1. ***konstrukcyjne*** **(głównie priorytetowe, ze stałym lub zmiennym priorytetem),**
2. ***popraw*****- sprowadzają się do generowania i przeglądu zupełnego pewnego podzbioru zbioru rozwiązań dopuszczalnych, tj. wyboru, ze skończonego zbioru, elementu o minimalnej wartości „*funkcji celu*”.**
3. ***dokładne*** **(dedykowane),**

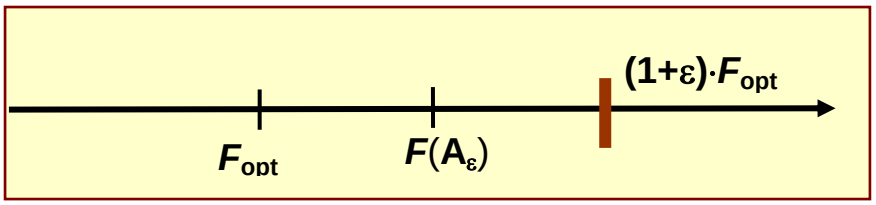
Konstrukcja algorytmu (popraw) wymaga określenia:

1. **metody obliczania wartości „funkcji celu” (działań na zmiennych losowych lub liczbach rozmytych),**
2. **relacji – porównania wartości „funkcji celu” dla różnych rozwiązań.**

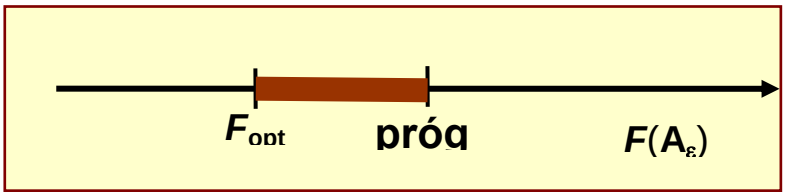
Problemy optymalizacji kombinatorycznej są mocno zróżnicowane. Są wśród nich takie, dla których

1) istnieje algorytm ε-aproksymacyjny, tj.,

$$\exists \varepsilon, \exists A_\varepsilon \ (alg.\ wiel.) \Rightarrow F(A_\varepsilon) \leq (1+\varepsilon) \cdot F_{opt}$$



2) mają próg aproksymacji (jeżeli $P \neq NP$),



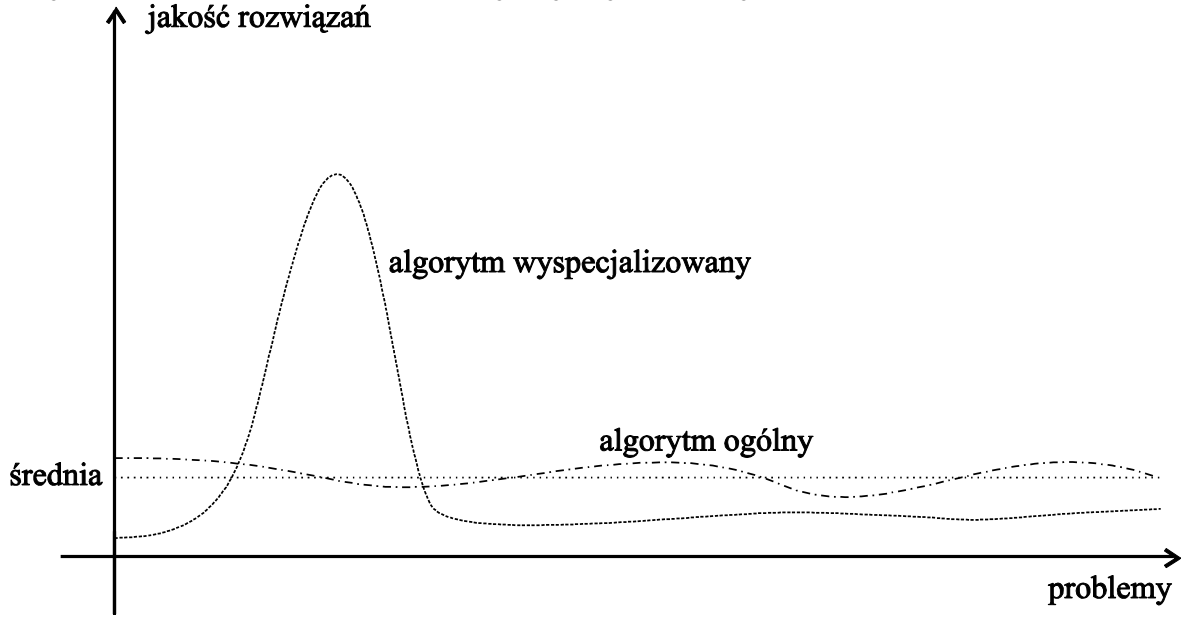
3. są nieaproksymowalne (jeżeli $P \neq NP$),

$$\forall \varepsilon, \textbf{nie istnieje } A_\varepsilon \ (alg.\ wiel.) \Rightarrow F(A_\varepsilon) \leq (1+\varepsilon) \cdot F_{opt}$$

Z tego powodu trudno spodziewać się uniwersalnych metod ich związywania. Ponadto

Twierdzenie „*no free lunch*”

Nie istnieje algorytm optymalizacyjny rozwiązywania wszystkich problemów, który byłby lepszy od innych.



„jakość” - stosunek wartości rozwiązania do czasu obliczeń.

Jakość algorytmu jest ściśle związana z ilością i jakością wiedzy, którą posiadamy na temat konkretnego problemu i efektywnie wykorzystamy w konstrukcji algorytmu.

Programowanie dyskretne

Dotyczy problemów decyzyjnych, w których pewne zmienne decyzyjne mogą przyjmować tylko dyskretne wartości np.:

- **całkowitoliczbowe**
- **binarne (0 lub 1)**

W zależności od rodzaju występujących zmiennych wyróżniamy zadania programowania

- **całkowitoliczbowego**
- **binarnego**
- **mieszanego – występują zmienne ciągłe i dyskretne**

Przykład

$$\max z = 7x_1 + 8x_2$$
$$3x_1 + 4x_2 \le 4$$
$$x_1, x_2 \ge 0$$

***Rozwiązanie*: x_1=4/3, x_2=0, z=28/3.**

Przy dodatkowym warunku

x_1 oraz x_2 całkowitoliczbowe, *rozwiązanie:* x_1=0, x_2=1, z=8.

(brak widocznej zależności pomiędzy rozwiązaniami)

Zagadnienia, w których stosuje się zmienne dyskretne

- **natura poszukiwanych rozwiązań jest dyskretna**
  - **wyznaczanie liczby niepodzielnych obiektów np. procesorów, zadań, pracowników itp.**
  - **określanie permutacji lub kombinacji pewnego zbioru obiektów – problemy kombinatoryczne**
- **wybór decyzji spośród wielu wariantów**
- **modelowanie funkcji kawałkami liniowych, np. uwzględnianie kosztów stałych**

Metody rozwiązywania problemów dyskretnych

- **algorytmy specjalizowane**
- **programowanie sieciowe**
- **metody przeglądu:**
  - **podziału i oszacowań (*branch & bound*)**
  - ***branch & cut, branch & price* i inne odmiany**
  - **programowanie w logice ograniczeń**
- **metody odcięć**
- **programowanie dynamiczne**
- **metody heurystyczne specjalizowane**
- **metaheurystyki:**
  - **algorytmy genetyczne (ewolucyjne)**
  - **przeszukiwanie „tabu search”**
  - **symulowane wyżarzanie (wychładzanie)**
  - **algorytmy randomizowane, np. GRASP**

Metody sztucznej inteligencji

Od wielu lat istnieje duże zainteresowanie metodami sztucznej inteligencji: sieciami neuronowymi, algorytmami ewolucyjnymi. Symulują one występujące w przyrodzie „naturalne" procesy prowadzące (pomimo licznych zakłóceń) do bardzo korzystnych strategii. Jednak elementy niejednoznaczności (niedeterminizmu) występują w tych algorytmach jedynie w procesie wyznaczania najlepszych rozwiązań (przeglądaniu przestrzeni rozwiązań).

Przyczyny braku dokładnej wartości (niepewność) parametrów:

1. **losowość (np. czas potrzebny na sortowanie listów),**
2. **unikalność zadania (określając parametr dodajemy: „około", „ponad", „nie więcej niż").**

W modelu można przyjąć, że niepewne dane są zmiennymi losowymi lub liczbami rozmytymi.

Istniejące „narzędzia”

- **solvery Zadań Programowania Mieszanego**
  - **wymagają zapisania problemu w postaci zadania programowania liniowego z ewentualnymi warunkami całkowitoliczbowości**
- **pakiety Programowania w Logice Ograniczeń**
  - **wymagają zapisania problemu w stosownym języku, zwykle dość „elastycznym”**
  - **dla uzyskania efektywności potrzebują sformułowania dobrych reguł sterowania procesem przeglądu**
- **programy specjalizowane dla danej klasy problemów**
  - **wymagają zapisania danych wejściowych w stosownym formacie**
- **samodzielna implementacja algorytmów (zaczerpniętych z literatury, opracowanych bądź dostosowanych przez siebie)**