Algorytmy decyzyjne i teoria złożoności

Czym będziemy się zajmować?

- matematycznymi metodami wspierającymi podejmowanie decyzji
- modelowaniem sytuacji decyzyjnych
- poszukiwaniem decyzji optymalnych z punktu widzenia przyjętych celów

W każdym przypadku podejmowania decyzji:
- istnieją warunki określające jakie decyzje mogą być podjęte (czyli jakie decyzje są dopuszczalne)
- dąży się do podjęcia decyzji najlepszej (optymalnej z punktu widzenia pewnego kryterium)

Problem to pytanie ogólne, zwykle zależne od kilku parametrów, na które należy dać odpowiedź.
Przypadek szczególny problemu (egzemplarz problemu, instancję) uzyskuje się, jeżeli ustalone zostaną wartości wszystkich parametrów problemu.

Algorytm to określona procedura obliczeniowa umożliwiająca rozwiązanie konkretnego problemu.
Algorytm powinien być **poprawny**, tzn. dla każdego egzemplarza problemu powinien zatrzymać się i dać dobry wynik.
Algorytmy dla tego samego problemu mogą różnić się **złożonością obliczeniową**.
Rozróżniamy złożoność:
- czasową
- pamięciową

Złożoność czasowa algorytmów

Analiza czasu działania algorytmu:
- przypadek pesymistyczny
- przypadek optymistyczny
- oczekiwany czas działania algorytmu

Przykład: sortowanie przez wstawianie

```
1 for j ← 2 to lenght[A]
2   do key ← A[j]
3   ** wstaw A[j] w posortowany ciąg A[1..j-1]
4       i ← j - 1
5       while  i > 0   i   A[j] > key
6           do  A[i+1] ← A[i]
7               i ← i - 1
8       A[i+1] ← key
```

Rząd wielkości

Czas działania sortowania przez wstawianie:
- optymistyczny: $an + b$
- pesymistyczny: $an^2 + bn + c$

Rozważany jest najbardziej znaczący składnik we wzorze (istotny dla dużych n).
Mówi się wtedy o **rzędzie wielkości funkcji**.
Pesymistyczny czas działania algorytmu sortowania przez wstawianie jest zatem **rzędu n^2**

Notacja Θ - asymtptycznie dokładne oszacowanie

Dla danej funkcji $g(n)$ przez $\Theta(g(n))$ oznaczamy pewien zbiór funkcji:
$\Theta(g(n)) = \{ f(n)$: istnieją dodatnie stałe c_1, c_2 i n_0, takie że $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ dla wszystkich $n \ge n_0 \}$

Notacja O - asymptotyczne ograniczenie górne

Dla danej funkcji $g(n)$ przez $O(g(n))$ oznaczamy pewien zbiór funkcji:
$O(g(n)) = \{ f(n)$: istnieją dodatnie stałe c i n_0, takie że $0 \le f(n) \le c\, g(n)$ dla wszystkich $n \ge n_0 \}$

Notacja Ω - asymptotyczne ograniczenie dolne

Dla danej funkcji $g(n)$ przez $\Omega(g(n))$ oznaczamy pewien zbiór funkcji:
$\Omega(g(n)) = \{ f(n)$: istnieją dodatnie stałe c i n_0, takie że $0 \le c\, g(n) \le f(n)$ dla wszystkich $n \ge n_0 \}$

Notacja o - ograniczenie górne, które nie jest asymptotycznie dokładne

Dla danej funkcji $g(n)$ przez $o(g(n))$ oznaczamy pewien zbiór funkcji:
$o(g(n)) = \{ f(n)$: dla każdej dodatniej stałej $c > 0$ istnieje stała $n_0 > 0$, taka że $0 \le f(n) < c\, g(n)$ dla wszystkich $n \ge n_0 \}$

Notacja ω - ograniczenie dolne, które nie jest asymptotycznie dokładne

Dla danej funkcji $g(n)$ przez $\omega(g(n))$ oznaczamy pewien zbiór funkcji:
$\omega(g(n)) = \{ f(n)$: dla każdej dodatniej stałej $c > 0$ istnieje stała $n_0 > 0$, taka że $0 \le c\, g(n) < f(n)$ dla wszystkich $n \ge n_0 \}$

Złożoność problemów

złożoność czasowa	Rozmiar maksymalnego problemu rozwiązywalnego w ciągu 1 godziny na komputerze			
	aktualnym	10-krotnie szybszym	1000-krotnie szybszym	przykładowy rozmiar
n	**n_1**	**$10\, n_1$**	**$1000\, n_1$**	**3 600 000 000**
n^2	**n_2**	**$3{,}16\, n_2$**	**$31{,}62\, n_2$**	**60 000**
n^3	**n_3**	**$2{,}15\, n_3$**	**$10\, n_3$**	**1 532**
2^n	**n_4**	**$n_4 + 3{,}32$**	**$n_4 + 9{,}97$**	**30**
3^n	**n_5**	**$n_5 + 2{,}1$**	**$n_5 + 6{,}29$**	**20**
10^n	**n_6**	**$n_6 + 1$**	**$n_6 + 3$**	**9**

złożoność czasowa	Rozmiar problemu				
	10	20	30	40	50
n	**$10*10^{-6}$ s**	**$20*10^{-6}$ s**	**$30*10^{-6}$ s**	**$40*10^{-6}$ s**	**$50*10^{-6}$ s**
$n \log_2 n$	**$33{,}2*10^{-6}$ s**	**$86{,}4*10^{-6}$ s**	**$147{,}2*10^{-6}$ s**	**$212{,}9*10^{-6}$ s**	**$282{,}5*10^{-6}$ s**
n^2	**$0{,}1*10^{-3}$ s**	**$0{,}4*10^{-3}$ s**	**$0{,}9*10^{-3}$ s**	**$1{,}6*10^{-3}$ s**	**$2{,}5*10^{-3}$ s**
n^3	**$1*10^{-3}$ s**	**$8*10^{-3}$ s**	**$27*10^{-3}$ s**	**$64*10^{-3}$ s**	**$125*10^{-3}$ s**
2^n	**0,001 s**	**1 s**	**17,9 min**	**12,7 dnia**	**35,7 lat**
3^n	**0,059 s**	**58,1 min**	**6,53 roku**	**3 855 wieków**	**$2{,}3*10^{8}$ wieków**
10^n	**2,8 h**	**31710 wieków**	**$3{,}17*10^{14}$ wieków**	**$3{,}17*10^{24}$ wieków**	**$3{,}17*10^{34}$ wieków**

Efektywność algorytmów

Algorytm optymalny dla danego problemu to algorytm
o najmniejszej złożoności obliczeniowej.
Algorytm efektywny to algorytm o złożoności wielomianowej.
Mówimy, że algorytm ma złożoność wykładniczą, jeżeli nie da się ograniczyć jego czasu działania poprzez wielomian.
Problem łatwy - można go rozwiązać algorytmem o złożoności wielomianowej.
Problem trudny - nie istnieje dla niego algorytm efektywny.

Algorytmy przybliżone

Jeżeli czas trwania algorytmu dokładnego jest zbyt długi, stosuje się algorytm przybliżony.
Algorytm przybliżony nie daje gwarancji znalezienia rozwiązania optymalnego.

Skrócenie czasu działania może nastąpić poprzez:
- wcześniejsze przerwanie obliczeń
- uproszczenie problemu
- wykorzystanie heurystyki

Schemat postępowania (metodologia)

1. Sformułowanie problemu decyzyjnego.
2. Budowa modelu matematycznego (lub jego analogu w wersji symulacyjnej)
3. Procedura obliczeniowa (lub postępowanie symulacyjne) za pomocą wybranego algorytmu (rozwiązanie zadania).
4. Analiza jakości rozwiązań i weryfikacja modelu.
5. Wdrożenie rozwiązania.

Model matematyczny

Aby zbudować model matematyczny należy ustalić:
- jakie wielkości mają być wyznaczone (podanie zmiennych decyzyjnych)
- jakie wielkości są dane (określenie parametrów)
- jakie warunki ograniczające musi spełniać decyzja dopuszczalna (zapisanie warunków ograniczających)
- cel, jaki chcemy osiągnąć (określenie funkcji celu)

Problem decyzyjny P

Rozwiązanie: $V(P) = extr\{Q(x) : x \in D\}$

x – zmienna decyzyjna

$Q(x)$ – funkcja celu (f. kryterialna)

D – zbiór dopuszczalnych decyzji

x^* – rozwiązanie optymalne

Decyzja zgodna z warunkami ograniczającymi to **decyzja dopuszczalna**.
Decyzja najlepsza z punktu widzenia przyjętych celów to **decyzja optymalna**.

Programowanie liniowe

W programowaniu liniowym:
- funkcja celu jest funkcją liniową zmiennej decyzyjnej
- warunki ograniczające są funkcjami liniowymi zmiennej decyzyjnej
- zmienna decyzyjna przyjmuje tylko wartości nieujemne

Zadanie programowania liniowego

Zmaksymalizować funkcję celu: $f(x)=\sum_{j=1}^{n} c_j x_j$

przy ograniczeniach (warunkach): $\sum_{j=1}^{n} a_{ij} x_{ij} \le b_i \quad$ dla $i=1,\ldots,m$

$x_j \ge 0 \quad$ dla $j=1,\ldots,n$

W zapisie macierzowym: zmaksymalizować $c^T x$ przy ograniczeniach: $\begin{matrix} Ax \le b \\ x \ge 0 \end{matrix}$

Dla problemów PL z dwuwymiarową zmienną decyzyjną można zastosować metodę graficzną.
- Rysujemy układ współrzędnych
- Wyznaczamy ograniczenia
- Wyznaczamy obszar dopuszczalnych rozwiązań
- Określamy warstwice funkcji celu
- Wyznaczamy punkt, w którym funkcja celu przyjmuje wartość największą

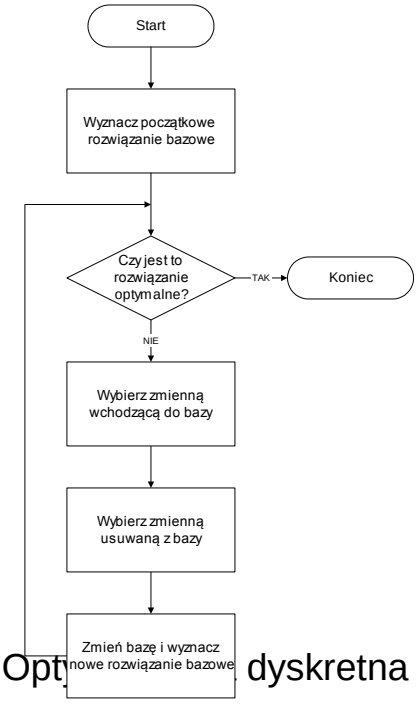
Metoda Simpleks

Postać bazowa problemu: zmaksymalizować: $c^T x$ przy ograniczeniach: $\begin{matrix} Ax = b \\ x \ge 0 \end{matrix}$

m - liczba ograniczeń

n - liczba wszystkich zmiennych

Schemat algorytmu metody SIMPLEKS



Tablica (tabela) SIMPLEKS

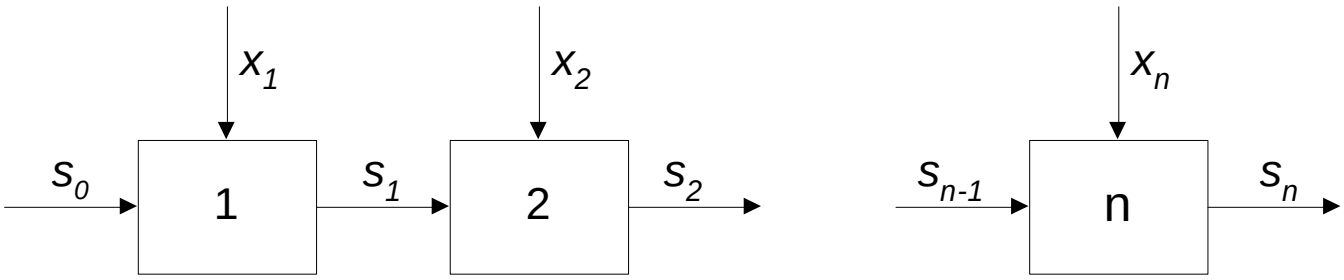
	b	x_{N1}	x_{N2}	…	$x_{N(n-m)}$
x_{B1}					
x_{B2}					
…					
x_{Bm}					
$f(x)$					

Optymalizacja dyskretna

Zadanie optymalizacji dyskretnej polega na wyznaczeniu ekstremum funkcji zwanej funkcją kryterialną określonej na dyskretnym zbiorze rozwiązań dopuszczalnych. Polega ono na przeszukiwaniu pewnej przestrzeni rozwiązań, skąd wybierane jest rozwiązanie najlepsze.

Programowanie dynamiczne

Przedmiotem programowania dynamicznego jest problem wyboru optymalnego ciągu decyzji powiązanych ze sobą w ten sposób, że decyzje późniejsze zależą od wcześniejszych. Problem n-etapowy:



$x_1, x_2, ..., x_{n-1}, x_n$ - **strategia (podjęte decyzje)**

$x_i \in X_i$ X_i - **zbiór decyzji dopuszczalnych dla etapu *i*-tego**

$s_0, s_1, s_2, ..., s_{n-1}, s_n$ - **trajektoria procesu**

$s_i \in S_i$ S_i - **zbiór stanów dopuszczalnych dla etapu *i*-tego**

$s_i = T_i(s_{i-1}, x_i)$ T_i - **funkcja przejścia**

$$Q = \sum_{i=1}^{N} q_i(s_{i-1}, x_i) \rightarrow \min$$

Q - **funkcja celu (dla całego procesu)** q_i - **funkcja oceniająca etap *i*-ty**

Programowanie dynamiczne stosuje się do procesów posiadających **własność Markowa**, tzn. takich, w których: wartość funkcji celu w zadaniu *n*-etapowym jest sumą wartości uzyskanych w poszczególnych etapach, a wartość uzyskana w *i*-tym etapie zależy od stanu w etapie poprzednim oraz od decyzji podjętej w *i*-tym etapie. Nie zależy natomiast od tego, jaką drogą system doszedł do stanu *i*-1.

Zasada optymalności Bellmana:
Dla wieloetapowych procesów decyzyjnych z własnością Markowa strategia optymalna ma tę właściwość, że jakikolwiek byłby stan początkowy i decyzja początkowa, pozostałe decyzje muszą tworzyć strategię optymalną ze względu na stan uzyskany w wyniku tej decyzji.

Rozważmy **rozwiązania** $f_n(s_0)$ takiego problemu (n – ilość etapów, s_0 – stan początkowy)

$$f_n(s_0) \overset{def}{=} \min_{x_1}\min_{x_2}\ldots\min_{x_n}\left(q_1(s_0,x_1)+q_2(s_1,x_2)+\ldots+q_n(s_{n-1},x_n)\right)=$$

$$=\min_{x_1}\left(q_1(s_0,x_1)+\min_{x_2}\ldots\min_{x_n}\left(q_2(s_1,x_2)+\ldots+q_n(s_{n-1},x_n)\right)\right)=$$

$$=\min_{x_1}\left(q_1(s_0,x_1)+f_{n-1}(s_1)\right)=$$

$$=\min_{x_1}\left(q_1(s_0,x_1)+f_{n-1}(T_1(s_0,x_1))\right)$$

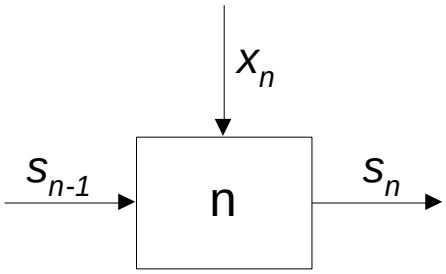
Ostatecznie otrzymujemy **wzór Bellmana**:

$$f_n(s_0)=\min_{x_1}\left(q_1(s_0,x_1)+f_{n-1}(T_1(s_0,x_1))\right)$$

Typy procesów n-etapowych:

a) s_n – dowolne, n – zadane (zadanie z dowolnym końcem)
b) s_n – zadane, n – zadane (musimy trafić w pewien stan końcowy)
c) s_n – zadane, n – minimalne (trafienie w cel w jak najmniejszej liczbie etapów)

Proces jednoetapowy:



$$f_1(s_{n-1})=?$$

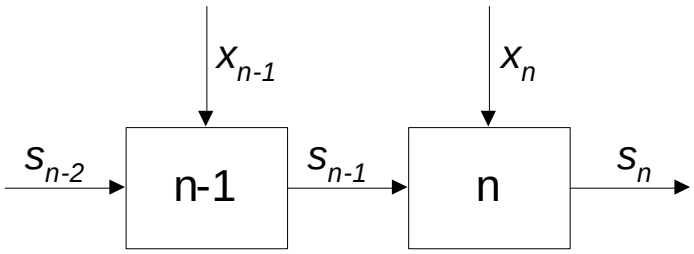
Dla każdego $s_{n-1} \in S_{n-1}$ należy znaleźć optymalną decyzję x_n^*

W zależności o typu zadania:

a) $\forall s_{n-1}\in S_{n-1} \qquad f_1(s_{n-1})=\min_{x_n} q_n(s_{n-1},x_n)$

b) $\forall s_{n-1}\in S_{n-1} \qquad f_1(s_{n-1})=\begin{cases} q_n(s_{n-1},x_n) & \text{jeżeli istnieje } x_n^* \\ \infty & \text{w p.p.} \end{cases}$

Proces dwuetapowy:



$f_2(s_{n-2}) = ?$

Należy dla każdego $s_{n-2} \in S_{n-2}$ obliczyć:

$$f_2(s_{n-2}) = \min_{x_{n-1}}(q_{n-1}(s_{n-2}, x_{n-1}) + f_1(T_{n-1}(s_{n-2}, x_{n-1})))$$

Proces n-etapowy:

$$f_n(s_0) = \min_{x_1}(q_1(s_0, x_1) + f_{n-1}(T_1(s_0, x_1)))$$

Metoda podziału i ograniczeń

Metoda podziału i ograniczeń (ang. B&B = Branch and Bound) polega na dekompozycji i sterowanym przeszukiwaniu zbioru rozwiązań dopuszczalnych danego problemu.

Problem komiwojażera

(ang. TSP = Travelling Salesman Problem)
Dany jest zbiór *n* miast.
Komiwojażer musi:

- odwiedzić wszystkie miasta
- w każdym z nich będąc tylko raz
- i powrócić do miasta z którego wyruszył.

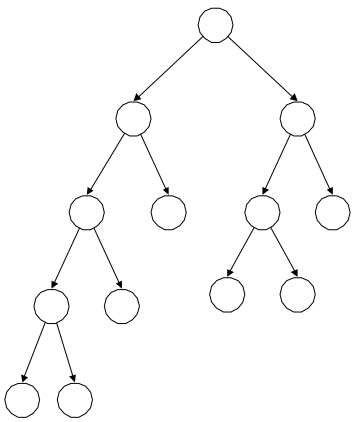
Znając odległości pomiędzy każdą parą miast należy tak ułożyć trasę komiwojażera, aby jej długość była minimalna.

TSP to problem trudny (NP-trudny), dla którego nie jest znany algorytm efektywny, tzn. rozwiązujący ten problem w czasie wielomianowym.
Aby znaleźć optymalne rozwiązanie takiego problemu należy (w najgorszym przypadku) przeglądnąć wszystkie możliwe rozwiązania i wybrać najlepsze.

W problemie komiwojażera z 20 miastami liczba możliwości to liczba permutacji n-1 elementowego zbioru czyli (n-1)! . Wynosi ona 19! > 10^{17}.

Algorytm podziału i ograniczeń o binarnym drzewie rozwiązań

1. Redukcja macierzy kosztów i obliczenie dolnego ograniczenia. (ang. LB = Lower Bound).
2. Wybór łuku *(i,j)* według którego nastąpi podział zbioru rozwiązań.
3. Podział zbioru rozwiązań na dwa podzbiory rozwiązań: z wybranym łukiem *(i,j)* (lewe poddrzewo) oraz nie zawierających tego łuku (prawe poddrzewo).
4. W lewym poddrzewie: zmniejszenie rozmiaru macierzy, zapobieżenie pojawieniu się cykli, redukcja macierzy i obliczenie dolnego ograniczenia.
5. W prawym poddrzewie: zabronienie łuku *(i,j)* (wstawienie ∞ w pozycji *(i,j)* macierzy), redukcja macierzy i obliczenie dolnego ograniczenia.
6. Jeżeli rozmiar macierzy w lewym poddrzewie wynosi 2×2 to dobierane są odpowiednio dwa pozostałe łuki, w przeciwnym przypadku dla tej macierzy wykonywany jest krok 2.
7. Rozpatrzenie wszystkich podzbiorów rozwiązań dla których dolne ograniczenie jest mniejsze od kosztu najlepszego znalezionego rozwiązania.



Przykład

Dana jest następująca macierz kosztów A:

	1	2	3	4	5	6
1	∞	3	93	13	33	9
2	4	∞	77	42	21	16
3	45	17	∞	36	16	28
4	39	90	80	∞	56	7
5	28	46	88	33	∞	25
6	3	88	18	46	92	∞

Każdy element a_{ij} oznacza koszt przejścia pomiędzy miastem i-tym a miastem j-tym, czyli reprezentuje łuk (i,j) .

Krok 1 – redukcja macierzy kosztów

	1	2	3	4	5	6	
1	∞	3	93	13	33	9	3
2	4	∞	77	42	21	16	4
3	45	17	∞	36	16	28	16
4	39	90	80	∞	56	7	7
5	28	46	88	33	∞	25	25
6	3	88	18	46	92	∞	3

	1	2	3	4	5	6
1	∞	0	90	10	30	6
2	0	∞	73	38	17	12
3	29	1	∞	20	0	12
4	32	83	73	∞	49	0
5	3	21	63	8	∞	0
6	0	85	15	43	89	∞
	0	0	15	8	0	0

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

macierz zredukowana
dolne ograniczenie LB = 81

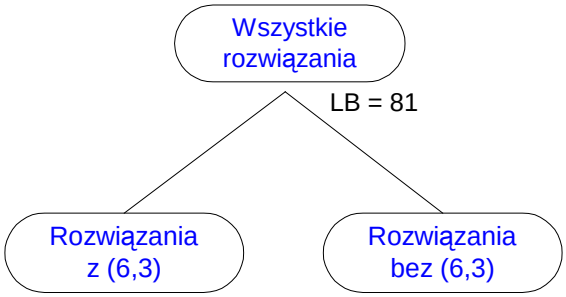
Krok 2 – wybór łuku wg którego nastąpi podział

Wybierany jest łuk, który powoduje największy wzrost dolnego ograniczenia dla rozwiązań niezawierających tego łuku.

(1,2): 2 + 1 = 3
(2,1): 12 + 0 = 12
(3,5): 1 + 17 = 18
(4,6): 32 + 0 = 32
(5,4): 0 + 2 = 2
(5,6): 0 + 0 = 0
(6,1): 0 + 0 = 0
(6,3): 0 + 48 = 48

Krok 3 – podział zbioru rozwiązań

Wybrany łuk to *(6,3)*.
Zbiór rozwiązań dzielony jest na dwa podzbiory:
- lewe poddrzewo – rozwiązania z łukiem *(6,3)*
- prawe poddrzewo – rozwiązania bez łuku *(6,3)*



Krok 4 – macierz lewego poddrzewa

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	12
4	32	83	∞	49	0
5	3	21	0	∞	0

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

macierz dla rozwiązań z łukiem *(6,3)*
dolne ograniczenie LB = 81

Krok 5 – macierz prawego poddrzewa

	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	0	35	89	∞

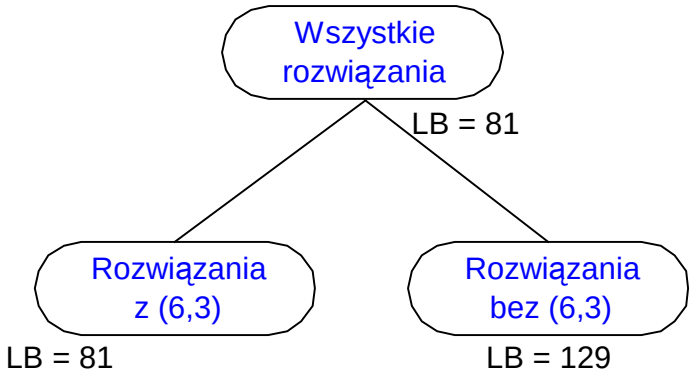
	1	2	3	4	5	6
1	∞	0	75	2	30	6
2	0	∞	58	30	17	12
3	29	1	∞	12	0	12
4	32	83	58	∞	49	0
5	3	21	48	0	∞	0
6	0	85	∞	35	89	∞

48

	1	2	3	4	5	6
1	∞	0	27	2	30	6
2	0	∞	10	30	17	12
3	29	1	∞	12	0	12
4	32	83	10	∞	49	0
5	3	21	0	0	∞	0
6	0	85	∞	35	89	∞

macierz dla rozwiązań bez łuku *(6,3)*
dolne ograniczenie LB = 81 + 48 =129

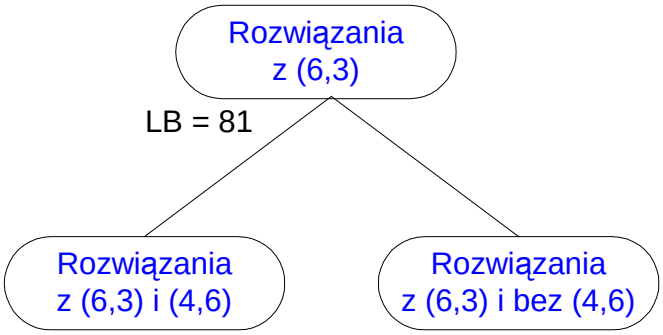
Drzewo rozwiązań po pierwszym etapie



Krok 2 i 3 – wybór łuku i podział zbioru rozwiązań

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

(4,6): 32 + 0 = 32



Krok 4 – macierz lewego poddrzewa

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	12	0
5	3	21	0	∞

(6,3) i (4,6): 4→6→3

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

macierz dla rozw. z łukiem *(6,3)* i *(4,6)*
dolne ograniczenie LB = 81

Krok 5 – macierz prawego poddrzewa

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	0
5	3	21	0	∞	0

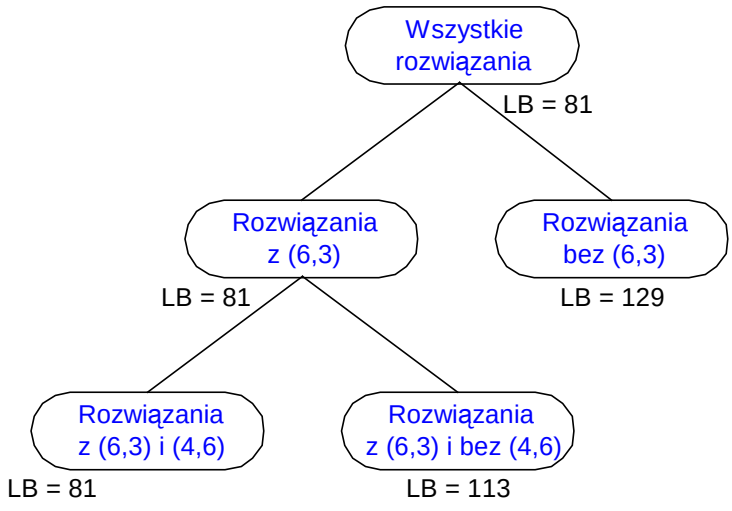
	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	32	83	∞	49	∞
5	3	21	0	∞	0

32

	1	2	4	5	6
1	∞	0	2	30	6
2	0	∞	30	17	12
3	29	1	12	0	∞
4	0	51	∞	17	∞
5	3	21	0	∞	0

macierz dla rozw. z *(6,3)* i bez *(4,6)*
dolne ograniczenie LB = 81 + 32 =113

Drzewo rozwiązań po drugim etapie



Krok 2 i 3 – wybór łuku i podział zbioru rozwiązań

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

(2,1): 17 + 3 = 20

Krok 4 – macierz lewego poddrzewa

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

	2	4	5
1	0	2	30
3	1	∞	0
5	21	0	∞

	2	4	5	
1	∞	2	30	2
3	1	∞	0	
5	21	0	∞	

1

	2	4	5
1	∞	0	28
3	0	∞	0
5	20	0	∞

macierz dla rozw. z *(6,3)*, *(4,6)* i *(2,1)*
dolne ograniczenie LB = 81+2+1 = 84

Krok 5 – macierz prawego poddrzewa

	1	2	4	5
1	∞	0	2	30
2	0	∞	30	17
3	29	1	∞	0
5	3	21	0	∞

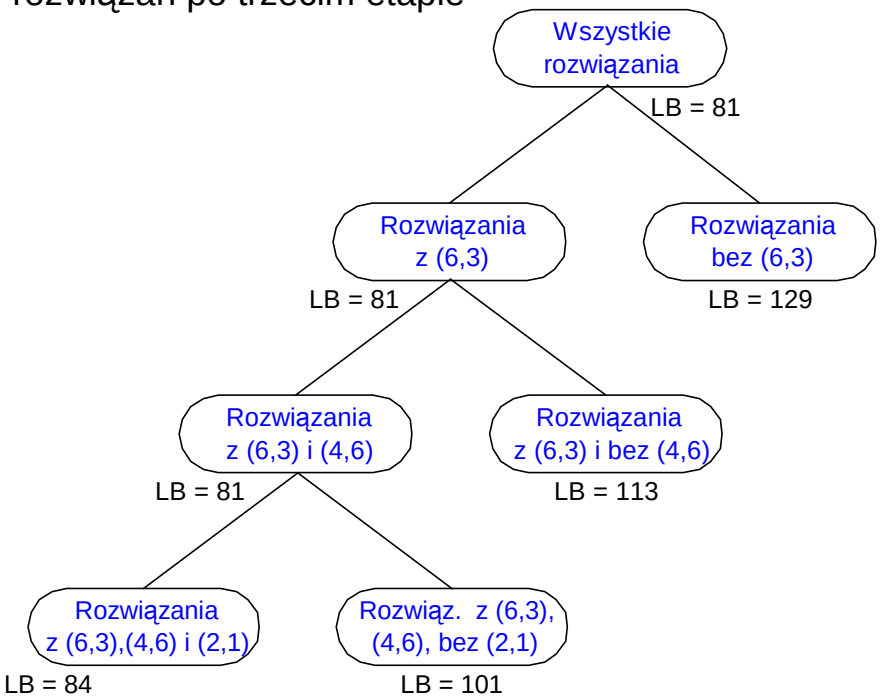
	1	2	4	5	
1	∞	0	2	30	
2	∞	∞	30	17	17
3	29	1	∞	0	
5	3	21	0	∞	

3

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

macierz dla rozwiązań z *(6,3)*, *(4,6)* i bez *(2,1)*
dolne ograniczenie LB = 81+17+3 =101

Drzewo rozwiązań po trzecim etapie

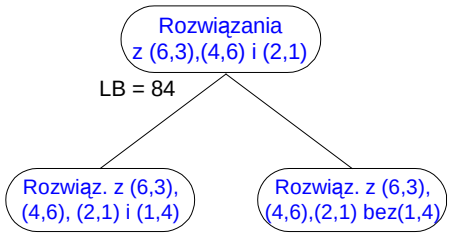


Krok 2 i 3 – wybór łuku i podział zbioru rozwiązań

	2	4	5
1	∞	0	28
3	0	∞	0
5	20	0	∞

(1,4): 28 + 0 = 28

(3,5): 0 + 28 = 28



Krok 4 – macierz lewego poddrzewa

	2	4	5
1	∞	0	28
3	0	∞	0
5	20	0	∞

	2	5
3	0	0
5	20	∞

20

	2	5
3	0	0
5	0	∞

macierz dla rozw. z *(6,3)*, *(4,6)*, *(2,1)* i *(1,4)*
dolne ograniczenie LB = 84+20 = 104

Krok 5 – macierz prawego poddrzewa

	2	4	5
1	∞	0	28
3	0	∞	0
5	20	0	∞

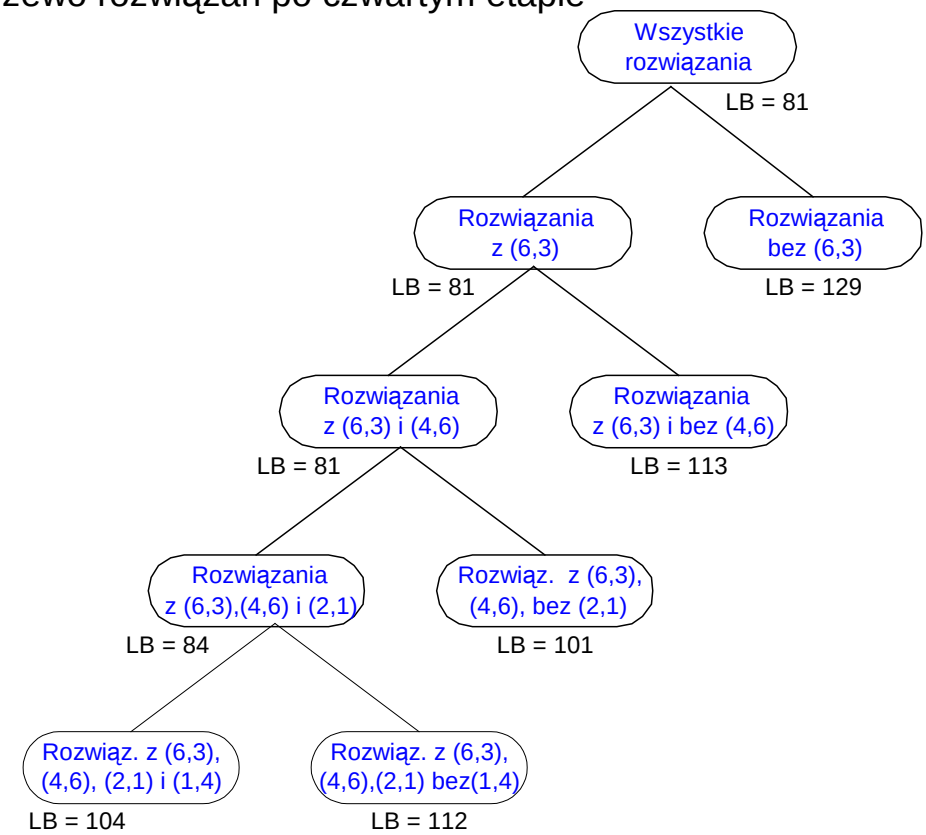
	2	4	5
1	∞	∞	28
3	0	∞	0
5	20	0	∞

28

	2	4	5
1	∞	∞	0
3	0	∞	0
5	20	0	∞

macierz dla rozwiązań z *(6,3)*, *(4,6)*, *(2,1)* bez *(1,4)*
dolne ograniczenie LB = 84+28 =112

Drzewo rozwiązań po czwartym etapie

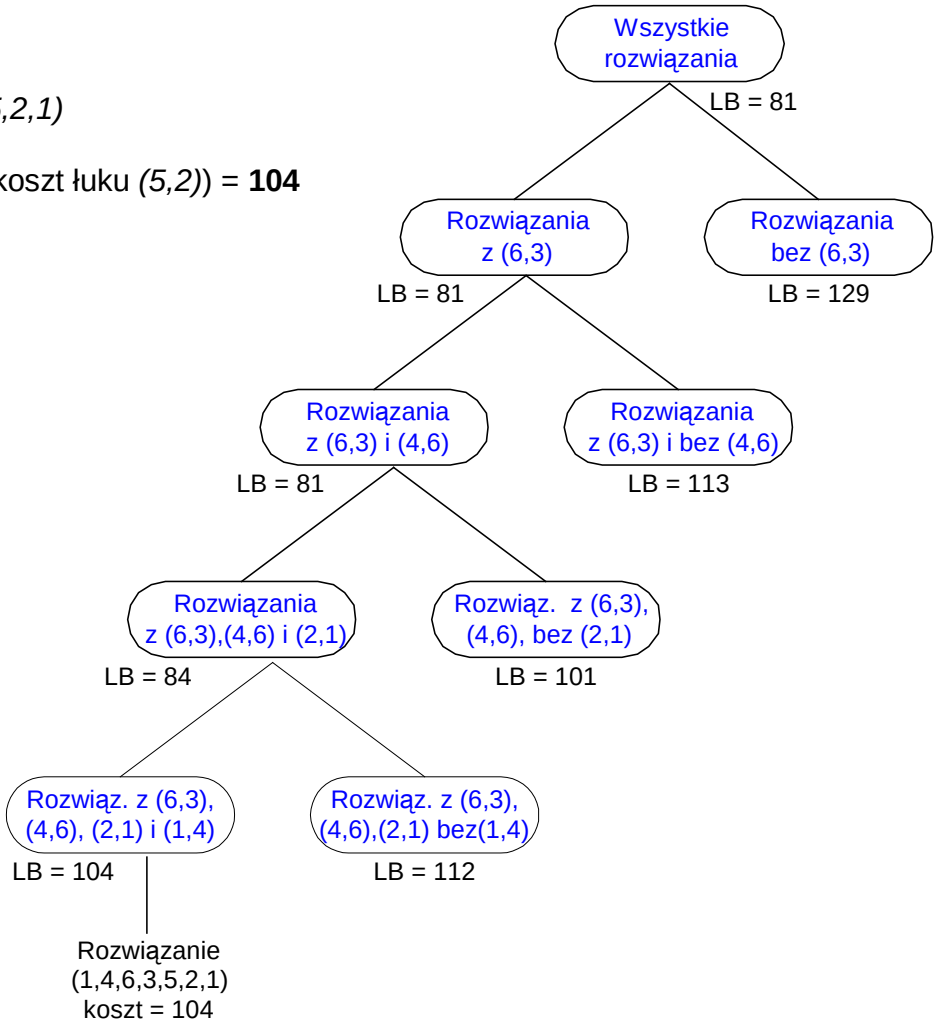


Krok 6 – uzupełnianie trasy komiwojażera

W macierzy 2x2 dobieramy tak pozostałe dwa łuki, aby utworzyć kompletną trasę komiwojażera (cykl Hamiltona).

	2	*5*
3	0	0
5	0	∞

Wybieramy łuki *(3,5)* i *(5,2)*
Trasa komiwojażera: *(1,4,6,3,5,2,1)*
Koszt trasy:
104 + 0 (koszt łuku *(3,5)*) + 0 (koszt łuku *(5,2)*) = **104**
Górne ograniczenie = **104**

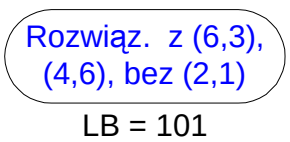


Krok 7 - pozostałe podzbiory rozwiązań

Przeglądamy pozostałe, nieprzebadane odnogi drzewa rozwiązań.

Wyszukujemy wszystkie te węzły, których dolne ograniczenie jest mniejsze od górnego ograniczenia (kosztu aktualnie najlepszego znalezionego rozwiązania).

Jest tylko jeden podzbiór rozwiązań, dla którego LB < 104.



Powracamy do tego węzła i realizujemy dla niego algorytm od kroku 2

Krok 2 i 3 – wybór łuku i podział zbioru rozwiązań

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

(5,1): 0 + 26 = 26

Krok 4 – macierz lewego poddrzewa

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

	2	4	5
1	0	2	30
2	∞	13	0
3	1	∞	0

	2	4	5
1	0	2	∞
2	∞	13	0
3	1	∞	0

2

	2	4	5
1	0	0	∞
2	∞	11	0
3	1	∞	0

macierz dla rozw. z *(6,3)*, *(4,6)*, *(5,1)* i bez *(2,1)*
dolne ograniczenie LB = 101+2 = 103

Krok 5 – macierz prawego poddrzewa

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	0	21	0	∞

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	26	1	∞	0
5	∞	21	0	∞

26

	1	2	4	5
1	∞	0	2	30
2	∞	∞	13	0
3	0	1	∞	0
5	∞	21	0	∞

macierz dla rozwiązań z *(6,3)*, *(4,6),* bez *(2,1)* i *(5,1)*
dolne ograniczenie LB = 101+26 =127

Krok 2 i 3 – wybór łuku i podział zbioru rozwiązań

	2	4	5
1	0	0	∞
2	∞	11	0
3	1	∞	0

(1,4): 0 + 11 = 11

(2,5): 11 +0 = 11

Krok 4 – macierz lewego poddrzewa

	2	4	5
1	0	0	∞
2	∞	11	0
3	1	∞	0

	2	5
2	∞	0
3	1	0

1

	2	5
2	∞	0
3	0	0

macierz dla rozw. z *(6,3)*, *(4,6)*, *(5,1)*, *(1,4)* i bez *(2,1)*
dolne ograniczenie LB = 103+1 = 104

Krok 5 – macierz prawego poddrzewa

	2	4	5
1	0	0	∞
2	∞	11	0
3	1	∞	0

	2	4	5
1	0	∞	∞
2	∞	11	0
3	1	∞	0

11

	2	4	5
1	0	∞	∞
2	∞	0	0
3	1	∞	0

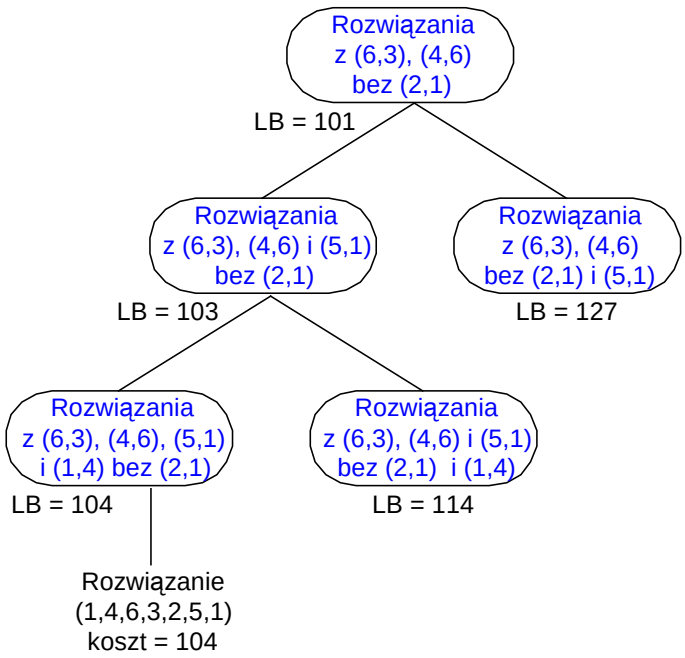
macierz dla rozw. z *(6,3)*, *(4,6)*, *(5,1)*, bez *(2,1)* i *(1,4)*
dolne ograniczenie LB = 103+11 =114

Krok 6 – uzupełnianie trasy komiwojażera

	2	5
2	∞	0
3	0	0

Wybieramy łuki *(2,5)* i *(3,2)*
Trasa komiwojażera: *(1,4,6,3,2,5,1)*
Koszt trasy: **104**

Nowy fragment drzewa rozwiązań



Ogólny schemat podziału i ograniczeń

W metodzie podziału i ograniczeń zbiór wszystkich dopuszczalnych rozwiązań danego problemu rozbijany jest na podzbiory.

Każdy taki podzbiór odpowiada problemowi o rozmiarze mniejszym niż problem wyjściowy (podproblemowi). Rozwiązywany jest on na jeden z następujących sposobów:

- poprzez relaksację
- bezpośrednio
- pośrednio
- poprzez podział

Relaksacja problemu (podproblemu) polega na takim jego „uproszczeniu", żeby był o wiele łatwiejszy do rozwiązania. Termin ten oznacza usunięcie, osłabienie i może odnosić się do ograniczeń lub funkcji celu.
Problem zrelaksowany jest rozwiązywany i sprawdzane jest czy znalezione rozwiązanie może być również rozwiązaniem problemu niezrelaksowanego.
Jeżeli tak, to jest to rozwiązanie optymalne danego problemu (podproblemu).
Jeżeli nie, to wartość funkcji celu rozwiązania problemu zrelaksowanego używane jest jako dolne ograniczenie dla tego problemu (podproblemu).

Bezpośrednio rozwiązuje się problem (podproblem) przeglądając wszystkie rozwiązania w danym podzbiorze, obliczając dla nich wartość funkcji celu i wybierając takie, dla którego wartość ta jest najmniejsza. W praktyce problem rozwiązuje się bezpośrednio wtedy, gdy istnieje już tylko jedno rozwiązanie w podzbiorze rozwiązań.

Pośrednio rozwiązany jest problem (podproblem), dla którego stwierdzono, że dolne ograniczenie jest większe od górnego ograniczenia całego problemu. Górne ograniczenie (ang. upper bound) to zazwyczaj wartość funkcji celu najlepszego dotychczas znalezionego rozwiązania.

Podział stosuje się do problemów, których nie da rozwiązać:
- bezpośrednio (zbiór rozwiązań jest zbyt liczny)
- pośrednio (dolne ograniczenie jest mniejsze niż górne ograniczenie)
- przy pomocy relaksacji (rozwiązanie problemu zrelaksowanego nie jest rozwiązaniem problemu)

Algorytmy podziału i ograniczeń (B&B)

Każdy algorytm oparty na metodzie podziału i ograniczeń wymaga precyzyjnego określenia następujących elementów:
- reguła wyboru określająca kolejność rozwiązywania podproblemów,
- przyjęta relaksacja dostarczająca dolne ograniczenie
- reguły eliminacji
- zasada podziału (tworzenia podproblemów)
- technika dostarczająca górne ograniczenie

Strategie wyboru kolejnych węzłów

Istnieją różne strategie wyboru kolejnych węzłów, z których kontynuowany jest proces podziału. Wyróżnia się następujące trzy podstawowe strategie:
- podział z węzła o minimalnym dolnym ograniczeniu
- podział z kolejnego otrzymanego węzła
- strategia mieszana

ALGORYTMY EWOLUCYJNE

literatura: Jarosław Arabas: „Wykłady z algorytmów ewolucyjnych”

Obliczenia ewolucyjne polegają na przeszukiwaniu przestrzeni rozwiązań w sposób naśladujący zasady ewolucji. Zakłada się, że niejawnym celem ewolucji jest optymalizacja dopasowania osobników do środowiska.

... w naturze

Populacja - zespół osobników (organizmów) danego gatunku żyjących równocześnie w określonym środowisku i wzajemnie na siebie wpływających, zdolnych do wydawania płodnego potomstwa.
Każdy osobnik posiada materiał genetyczny zorganizowany w postaci chromosomów.
Na chromosom składają się geny, które kodują poszczególne cechy osobnika.
Genotyp to skład genetyczny danego osobnika.
Fenotyp to ogół „widocznych” cech morfologicznych, fizjologicznych i biochemicznych danego osobnika.

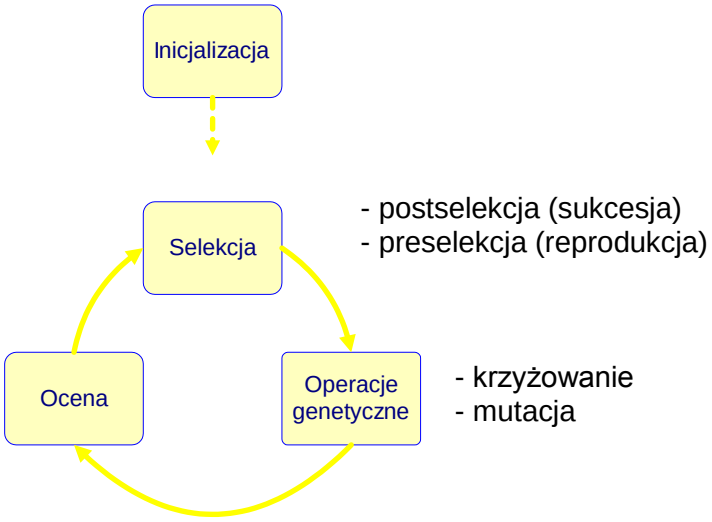
Algorytmy ewolucyjne

Algorytm ewolucyjny przetwarza populację osobników, z których każdy reprezentuje pojedyncze rozwiązanie problemu.

Każdy osobnik wyposażony jest w informację stanowiącą jego genotyp. Na podstawie genotypu określany jest fenotyp podlegający ocenie środowiska.

Dla każdego osobnika wyznaczana jest wartość przystosowania będącą miarą jakości rozwiązania.

Schemat algorytmów ewolucyjnych



Techniki obliczeń ewolucyjnych

- Algorytmy genetyczne
- Strategie ewolucyjne
- Programowanie ewolucyjne
- Programowanie genetyczne

Algorytm genetyczny (GA)

Przetwarzane populacje:

P^t - populacja bazowa

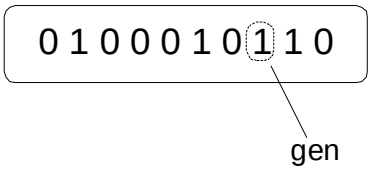
O^t - populacja potomna
T^t - populacja tymczasowa
W populacjach jest jednakowa ilość osobników. Populacja początkowa powstaje przez losowe wygenerowanie odpowiedniej liczby osobników.

Algorytm genetyczny- schemat

```
begin
    t := 0
    inicjalizacja P^0
    ocena P^0
    while (not warunek stopu) do
    begin
        T^t := reprodukcja P^t
        O^t := krzyżowanie i mutacja T^t
        ocena O^t
        P^(t+1) := O^t
        t:=t+1
    end
end
```

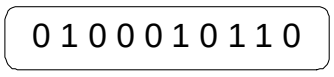
Algorytm genetyczny - kodowanie osobników
Kodowanie binarne - chromosom jest *n*-elementowym wektorem genów, z których każdy jest zerem lub jedynką.

chromosom:

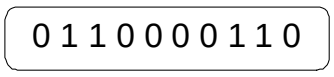


Algorytm genetyczny - operacje genetyczne
Mutacja jest operatorem wykonywanym dla każdego genu osobno (z prawdopodobieństwem p_m wartość genu zmienia się na przeciwną).

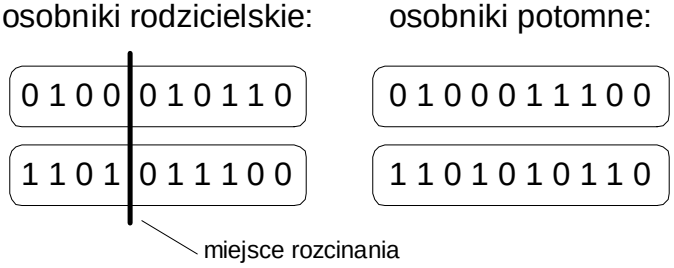
osobnik rodzicielski:



osobnik potomny:



W krzyżowaniu biorą udział dwa osobniki rodzicielskie. Jest ono jednopunktowe, a miejsce rozcięcia jest wybierane losowo z rozkładem równomiernym.



Algorytm genetyczny - sposób reprodukcji
Populacja $\boldsymbol{T^t}$ jest tworzona przy użyciu reprodukcji proporcjonalnej (ruletkowej).
Prawdopodobieństwo skopiowania (reprodukcji) osobnika *X* ze zbioru $\boldsymbol{P^t}$ do zbioru $\boldsymbol{T^t}$ wynosi:

$$p_r(X) = \frac{\Phi(X)}{\sum_{Y \in P^t} \Phi(Y)}$$

gdzie: *X* , *Y* – osobniki, Φ – funkcja przystosowania

<u>Algorytm genetyczny dla problemu załadunku</u>
Problem załadunku (plecakowy):
n przedmiotów
w_i - waga *i*-tego przedmiotu
c_i - korzyść z zabrania *i*-tego przedmiotu
W - udźwig plecaka
Zmienna decyzyjna: $x=(x_1,x_2,...x_n)$, $x_i \in \{0,1\}$

Funkcja celu: $\sum_{i=1}^{n} c_i x_i \rightarrow \max$

Ograniczenie: $\sum_{i=1}^{n} w_i x_i \leq W$

Binarny chromosom *X* reprezentuje zmienną decyzyjną.

Operatory genetyczne:
- mutacja z prawdopodobieństwem p_m
- krzyżowanie jednopunktowe, z prawdopodobieństwem zaistnienia p_c

Ograniczenie udźwigu plecaka jest uwzględniane poprzez zewnętrzną funkcję kary.

Funkcja przystosowania: $\Phi(X) = \sum_{i=1}^{n} c_i x_i - K \cdot \max\left(0, \sum_{i=1}^{n} w_i x_i - W\right)$

Współczynnik wagowy:

$$K = \frac{\max_{i=1,...,n} c_i}{\min_{i=1,...,n} w_i}$$

Reprodukcja jest proporcjonalna. Aby uniknąć „ujemnego prawdopodobieństwa", stosowana jest metoda skalowania przystosowania. Prawdopodobieństwo reprodukcji wynosi zatem:

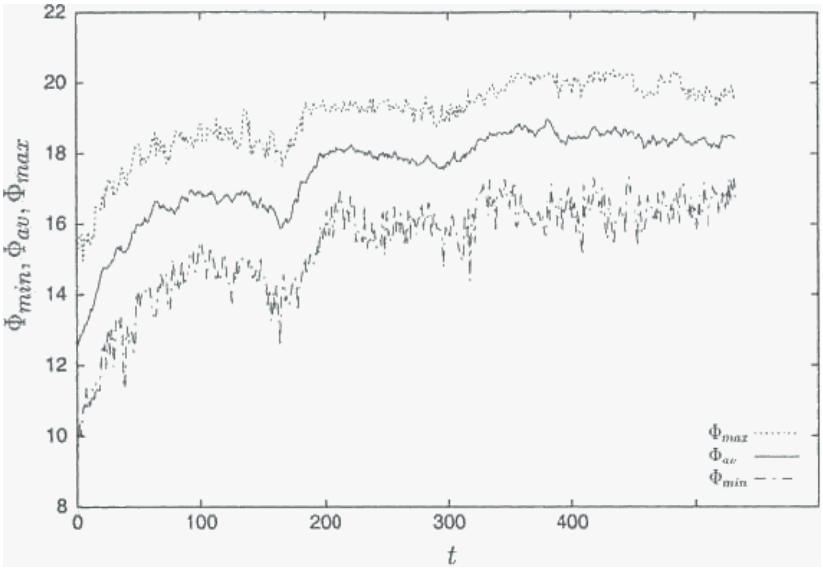
$$p_r(X) = \frac{\Phi(X) - \Phi_{\min}}{\sum_{Y \in P^t} \Phi(Y) - \Phi_{\min}}$$

gdzie: $\Phi_{\min}$ - wartość przystosowania najgorszego osobnika w populacji $\boldsymbol{P^t}$

<u>Eksperyment:</u>
<u>Dane:</u>
n = 50
wartości c_i i w_i - z przedziału [0.0001,1]
W = 13 (ok. 25% sumy wag przedmiotów)
Liczba osobników w populacji bazowej: 100

W populacji początkowej P^0 liczba jedynek w chromosomach: około 20%
Prawdopodobieństwa dla operatorów genetycznych:
$p_m = 0.02$ $p_c = 0.7$

Krzywe zbieżności



Strategie ewolucyjne
Strategia ewolucyjna polega na mutowaniu pewnego początkowego rozwiązania. Podstawowy algorytm to strategia (1+1). Występuje w nim mechanizm adaptacji zasięgu mutacji zwany regułą 1/5 sukcesów.
Chromosomy: bazowy $\boldsymbol{X}^t$ oraz tymczasowy $\boldsymbol{Y}^t$
Gen w chromosomie - liczba rzeczywista.

Strategia ewolucyjna (1+1) - schemat

```
begin
    t := 0
    inicjacja X^t
    ocena X^t
    while (not warunek stopu) do
    begin
        Y^t := mutacja X^t
        ocena Y^t
        if (Φ(Y^t) > Φ(X^t)) than X^(t+1) := Y^t
            else X^(t+1) := X^t
        t:=t+1
        end
end
```

Strategia ewolucyjna (1+1) - mutacja
Chromosom $\boldsymbol{Y}^t$ jest generowany przez dodanie losowej modyfikacji do każdego genu chromosomu $\boldsymbol{X}^t$:

$$Y_i^t = X_i^t + \sigma\xi_{N(0,1),i}$$

gdzie: σ - zasięg mutacji
ξ - wartość zmiennej losowej o rozkładzie normalnym

Reguła 1/5 sukcesów
Jeżeli przez kolejnych *k* generacji liczba mutacji zakończonych sukcesem jest większa niż 1/5, to należy zwiększyć zasięg mutacji ($\sigma := c_i \sigma$).
Gdy dokładnie 1/5 mutacji kończy się sukcesem, wartość σ nie wymaga modyfikacji.
W przeciwnym przypadku należy zawęzić zasięg mutacji ($\sigma := c_d \sigma$).
Wartości parametrów modyfikujących:
$c_d = 0.82$, $c_i = 1/0.82$.

Strategia (1+1) ma niewielką odporność na minima lokalne, w związku z tym powstały nowe schematy:
- strategia (1+λ)
- strategia (μ + λ)
- strategia (μ, λ)

Strategia (μ + λ)
Każdy osobnik składa się z dwóch chromosomów:
1) wektor *X* wartości zmiennych niezależnych
2) wektor σ zawierający wartości odchyleń standardowych wykorzystywanych podczas mutacji

Strategia (μ + λ) – operacje genetyczne
Mutacja
Podczas mutacji najpierw modyfikowane są elementy wektora σ, a następnie na podstawie tego wektora mutowany jest chromosom *X*.
Dokonuje się samoczynna adaptacja zasięgu mutacji.

Krzyżowanie (rekombinacja)
Najczęściej używa się operatora polegającego na uśrednianiu lub na wymianie wartości wektorów *X* i σ chromosomów macierzystych.

Strategia (μ + λ) – schemat

```
begin
    t := 0
    inicjacja P^t
    ocena P^t
    while (not warunek stopu) do
    begin
        T^t := reprodukcja P^t
        O^t := krzyżowanie i mutacja T^t
        ocena O^t
        P^(t+1) := μ najlepszych osobników z P^t ∪ O^t
        t:=t+1
        end
end
```

Programowanie ewolucyjne - początki

Rozważany był problem odkrywania gramatyki nieznanego języka, gdy:
- znany był zestaw jego symboli
- dane były przykłady wyrażeń syntaktycznie poprawnych

Gramatyka była modelowana za pomocą automatu skończonego (ewoluujące automaty). Przedmiotem poszukiwań były:
- zbiór stanów automatu
- funkcja przejść
- funkcja wyjść

Osobnik - automat skończony
Funkcja przystosowania osobnika - liczba prawidłowych klasyfikacji wyrażeń poznawanego języka.
Rodzaje mutacji :
- dodanie stanu
- usunięcie stanu
- zmiana stanu początkowego
- zmiana funkcji wyjść
- zmiana funkcji przejść

Programowanie ewolucyjne

Programowanie ewolucyjne uległo zmianom upodabniającym je do strategii ewolucyjnych:
- zastosowanie w optymalizacji numerycznej
- wprowadzenie reprezentacji rzeczywistoliczbowej
- dodanie do genotypu osobnika dodatkowego chromosomu (wektora zasięgu mutacji)
- wprowadzenie mechanizmu modyfikacji tego zasięgu

Programowanie genetyczne

Początki:

Podejmowane były próby znalezienia sposobu automatycznego generowania tekstów programów (w języku LISP), przy znanych kryteriach oceny prawidłowości działania. Program i dane reprezentowane były w postaci drzewa.

W programowaniu genetycznym optymalizacji podlegają zarówno wartości parametrów jak i struktura ich powiązań.
Chromosom: drzewo składające się z węzłów i krawędzi.
Operatory genetyczne: mutacja (zmiana zawartości losowo wybranego węzła) oraz krzyżowanie (wymiana poddrzew pomiędzy parą rodziców).

Definiowanie algorytmów ewolucyjnych

Tworząc algorytm ewolucyjny należy określić przede wszystkim:
- metody selekcji
- operatory genetyczne
- sposób kodowania
- środowisko działania (funkcję przystosowania)
- wielkość populacji
- kryteria zatrzymania

Metody selekcji

Reprodukcja (preselekcja):
- proporcjonalna (ruletkowa)
- rangowa
- turniejowa
- progowa

Sukcesja (postselekcja):
- z całkowitym zastępowaniem
- z częściowym zastępowaniem
- elitarna

Kodowanie

Najczęściej spotykane kodowania:
- binarne
- rzeczywiste
- drzewiaste

Kodowanie powinno być tak dobrane, aby można było przedstawić każde rozwiązanie.

Pożądaną cechą kodowania jest również zachowanie proporcji odległości w przestrzeni genotypu i fenotypu.

Operatory genetyczne

Operatory: krzyżowanie i mutacja.
Pożądane cechy:
- gwarantowanie spójności przestrzeni genotypów
- brak obciążeń operatorów

Operator krzyżowania

Pod względem sposobu krzyżowania wyróżniamy:
a) krzyżowanie wymieniające
- jednopunktowe
- dwupunktowe
- równomierne

b) krzyżowanie uśredniające

Pod względem liczby osobników rodzicielskich oraz potomnych wyróżniamy:
- krzyżowanie dwuosobnicze
- krzyżowanie wieloosobnicze
- krzyżowanie globalne

Operatory mutacji

- mutacja dla kodowania binarnego
- mutacja dla chromosomów, w których geny przyjmują wartości ze zbioru liczb rzeczywistych

Wprowadzana może być adaptacja zasięgu mutacji.

W przypadku adaptacji samoczynnej parametry odpowiadające za zasięg mutacji stanowią dodatkowy chromosom osobnika.

Funkcja przystosowania

Funkcja przystosowania powstaje w wyniku przekształcenia funkcji celu.

W celu polepszenia własności algorytmów ewolucyjnych stosuje się skalowanie przystosowania.

Metody uwzględniania ograniczeń

Ograniczenia występujące w problemie mogą zostać uwzględnione poprzez:

- wprowadzenie funkcji kary do funkcji przystosowania
- specjalizowane kodowanie i operatory genetyczne
- „naprawianie" chromosomów

Kryteria zatrzymania algorytmu

Najczęściej stosowane kryteria to:

- kryterium maksymalnego kosztu algorytmu
- kryterium zadowalającego poziomu funkcji przystosowania
- kryterium minimalnej szybkości poprawy
- kryterium zaniku różnorodności populacji
- kryterium zaniku samoczynnie adoptowanego zasięgu mutacji

Techniki zaawansowane

- sterowanie czasem życia osobników
- diploidalność i dominowanie
- inwersja i inne operacje rekonfiguracji
- zróżnicowanie płciowe
- nisze i gatunki, optymalizacja wielomodalna
- algorytmy koewolucyjne
- pamięć na poziomie populacji i na poziomie osobnika

Przykład - problem komiwojażera

Dany jest zbiór *n* miast i sieć połączeń między nimi. Każde miasto należy odwiedzić dokładnie jeden raz. Szukamy takiej trasy komiwojażera, która ma minimalną długość (minimalny koszt).

Funkcja przystosowania - łatwa do określenia.
Kluczowy dla tego problemu jest wybór reprezentacji rozwiązania (trasy komiwojażera) oraz dobór operatorów genetycznych.

Kodowanie
Kodowania binarne dla problemu komiwojażera okazuje się nieadekwatne.
Kodowania specjalizowane:

- przyległościowe
- porządkowe
- ścieżkowe

Kodowanie przyległościowe

Chromosom:

2 4 8 3 9 7 1 5 6

reprezentuje trasę 1-2-4-3-8-5-9-6-7
Miasto *j* znajduje się na pozycji *i* jeżeli trasa wiedzie z miasta *i* do *j*.
Stosowane są operatory krzyżowania:

- z wymianą krawędzi
- z wydzielaniem podtras
- heurystyczne

Kodowanie porządkowe

Chromosom:

1 1 2 1 4 1 3 1 1

reprezentuje trasę 1-2-4-3-8-5-9-6-7
Element *i* odnosi się do określonego miasta w pewnym ciągu miast, który jest punktem odniesienia (tutaj jest to: (1 2 3 4 5 6 7 8 9))
Główną zaletą tego sposobu kodowania jest to, że działają przy nim standardowe operatory krzyżowania.

Kodowanie ścieżkowe

Chromosom:

1 2 4 3 8 5 9 6 7

reprezentuje trasę 1-2-4-3-8-5-9-6-7
Jest to najbardziej naturalny sposób kodowania, wymaga jednak specjalizowanych operatorów krzyżowania, np.:

- z częściowym odwzorowaniem (PMX)
- z porządkowaniem (OX)
- cykliczne (CX)

Szeregowanie sieciowe

Co to jest projekt?

Projekt (przedsięwzięcie) to zbiór powiązanych ze sobą kroków prowadzących do osiągnięcia konkretnego celu.
Projekt realizowany jest jednorazowo.
Cel projektu wyznacza zakres działań jakie należy podjąć.

Przedsięwzięcie (projekt)

- zbiór zadań (czynności) z określonymi czasami trwania
- ograniczenia kolejnościowe (relacja częściowego porządku)

Cel: Minimalizacja czasu potrzebnego do ukończenia przedsięwzięcia.
Zadania wykonywane są przy użyciu różnych zasobów.
Ze względu na sposób reprezentacji przedsięwzięcia stosuje się określenie szeregowanie sieciowe.

Reprezentacja projektu

Dwie podstawowe metody reprezentacji przedsięwzięcia (projektu):

1) Sieć czynności (czynność-w-węźle)
   - czynności (zadania) przyporządkowane są węzłom
   - ograniczenia kolejnościowe mają postać łuków
2) Sieć zdarzeń (czynność-na-łuku)
   - węzły sieci reprezentują zdarzenia
   - czynności przypisane są łukom
   - przy budowaniu sieci zachowane są ograniczenia kolejnościowe

Przykład przedsięwzięcia

Uruchomienie nowej obrabiarki w ciągu produkcji

a) Wykonanie fundamentów dla obrabiarki
b) Zamówienie i dostawa obrabiarki
c) Podprowadzenie mediów dla obrabiarki
d) Montaż fizyczny obrabiarki
e) Przygotowanie specjalistycznego oprzyrządowania
f) Testy rozruchowe
g) Szkolenia pracowników na maszynie

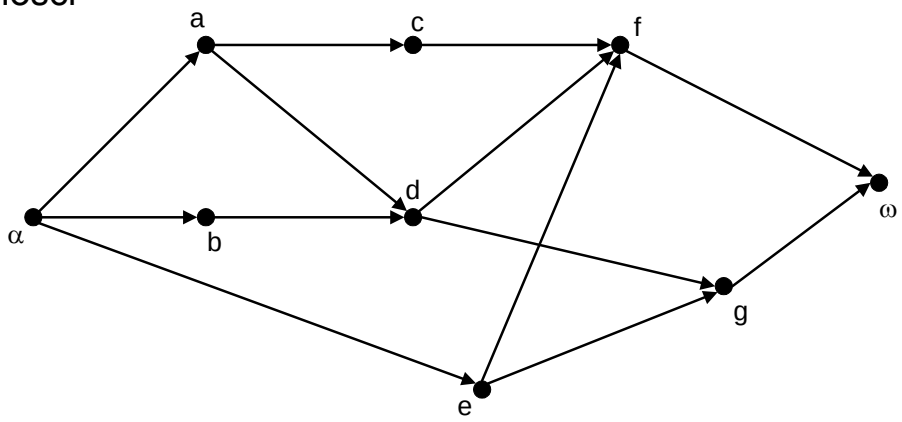
Przykład przedsięwzięcia

Czynność	Bezpośrednie poprzedniki	Czas trwania
a	-	5
b	-	3
c	a	3
d	a, b	4
e	-	5
f	c, d, e	3
g	d, e	4

Sieć czynności

Czynność	Bezpośrednie poprzedniki	Bezpośrednie następniki	Wszystkie następniki
α	-	a, b, e	a, b, c, d, e, f, g, ω
a	α	c, d	c, d, f, g, ω
b	α	d	d, f, g, ω
c	a	f	f, ω
d	a, b	f, g	f, g, ω
e	α	f, g	f, g, ω
f	c, d, e	ω	ω
g	d, e	ω	ω
ω	g,h	-	-

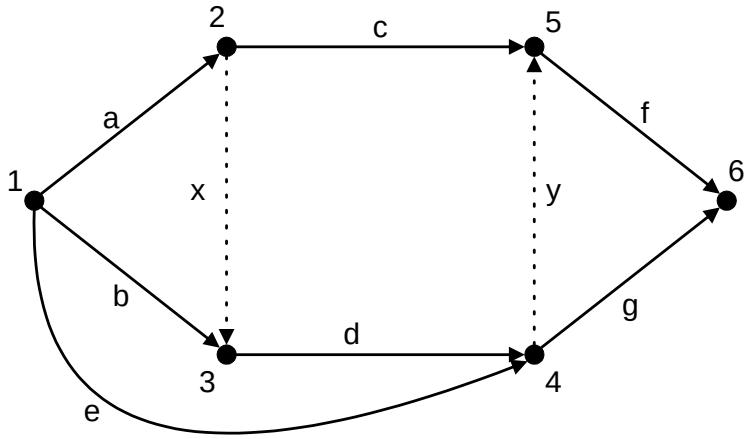
Sieć czynności



Transformacja sieci czynności do sieci zdarzeń

Węzły reprezentują zdarzenia, momenty w których rozpoczynają się i kończą się czynności.
Czynności przyporządkowujemy izolowanym łukom.
Topologia sieci odwzorowuje ograniczenia kolejnościowe. W razie potrzeby wprowadzamy czynności pozorne.
Węzły (reprezentujące zdarzenia) ponumerowane są w porządku topologicznym od 1 do *n*.
Sieć zdarzeń ma mniej węzłów i łuków niż sieć czynności.

Sieć zdarzeń



Metody przedstawienia harmonogramu przedsięwzięcia

Metoda amerykańska - wykorzystuje sieć zdarzeń)
Metoda francuska (potencjałów) - wykorzystuje sieć czynności
Wykres Gantta

Metoda ścieżki krytycznej

Zaplanowanie przedsięwzięcia polega na stworzeniu harmonogramu.

W tym celu stosuje się tzw. metodę ścieżki krytycznej, minimalizującą całkowity czas potrzebny do ukończenia przedsięwzięcia.

Najkrótszy czas trwania przedsięwzięcia jest równy długości najdłuższej drogi prowadzącej z węzła początkowego do węzła końcowego.

Zadania leżące na ścieżce krytycznej to zadania krytyczne.

Analiza czasowa sieci

W wyniku przeprowadzenia analizy określa się:
- najwcześniejsze możliwe terminy rozpoczęcia czynności/zdarzeń
- najpóźniejsze możliwe terminy rozpoczęcia
- rezerwy czasowe zdarzeń
- luzy czasowe czynności (pełny, bezpieczny, swobodny, niezależny)

Luzy czasowe dla zadań krytycznych wynoszą 0.

Analiza czasowa sieci (sieć zdarzeń)

Zdarzenia:

π_i^e najwcześniejszy możliwy moment wystąpienia zdarzenia *i*-tego (*i* = 1,2, ..., *n*)

$\pi_1^e = 0$

$\pi_i^e = \max\left\{\pi_j^e + p_{ji} : (j,i) \in V\right\}$

π_i^l najpóźniejszy możliwy moment wystąpienia zdarzenia *i*-tego (*i* = *n*, *n*-1, *n*-2, ..., 1)

$\pi_n^l = \pi_n^e$

$\pi_i^l = \min\left\{\pi_j^l - p_{ij} : (i,j) \in V\right\}$

rezerwa czasowa zdarzenia: $s_i = \pi_i^l - \pi_i^e$

gdzie p_{ij} – czas trwania czynności (*ij*), która odpowiada łukowi między *i*-tym a *j*-tym zdarzeniem

Czynności:

najwcześniejszy czas rozpoczęcia czynności *(i,j)* $ES_{ij} = \pi_i^e$

najwcześniejszy czas zakończenia czynności *(i,j)* $EF_{ij} = \pi_i^e + p_{ij}$

najpóźniejszy czas zakończenia czynności *(i,j)* $LF_{ij} = \pi_j^l$

najpóźniejszy czas rozpoczęcia czynności *(i,j)* $LS_{ij} = \pi_j^l - p_{ij}$

rezerwa czasowa czynności (luz pełny) $s_{ij}^1 = \pi_j^l - \pi_i^e - p_{ij}$

luz bezpieczny $s_{ij}^2 = \pi_j^l - \pi_i^l - p_{ij}$

luz swobodny $s_{ij}^3 = \pi_j^e - \pi_i^e - p_{ij}$

luz niezależny $s_{ij}^4 = \max\left\{0, \pi_j^e - \pi_i^l - p_{ij}\right\}$

Szeregowanie zadań

Problemy szeregowania zadań

Sformułowanie problemu:

Dany jest zbiór zadań oraz zbiór zasobów służących do wykonania tych zadań. Należy wykonać wszystkie zadania z podanego zbioru w taki sposób, aby ekstremalizowane było określone kryterium jakości.

Przykłady zadań

- procesy montażu / obróbki detali w przemyśle maszynowym
- czynności inwestycyjne w budownictwie
- obsługa zgłoszenia
- przejazd odcinkiem drogi
- obliczenia komputerowe

Przykłady zasobów

- maszyny różnego typu
- siła robocza
- energia
- paliwo
- nakłady finansowe
- surowce
- systemy produkcyjne
- kanały obsługi

Maszyny

Maszyny to szczególny rodzaj zasobów:

- stanowią typ zasobów niezbędny dla wszystkich zadań
- każde zadanie może być wykonywane w danej chwili przez co najwyżej jedną maszynę

Ograniczeń tych nie muszą spełniać pozostałe zasoby.

Zadania

Zadanie składa się z mniejszych jednostek, tzw. operacji (w szczególności zadanie może składać się z jednej operacji).

Zadanie $J_j = \{O_{1j}, O_{2j}, ..., O_{kj}\}$

Zbiór zadań $\boldsymbol{J} = \{J_1, J_2, ..., J_n\}$

Charakterystyka zadań - parametry

o_j – liczba operacji w zadaniu

p_{ij} – czas wykonywania operacji O_{ij} (i–tej operacji zadania J_j)

r_j – moment gotowości do wykonania

d_j – pożądany czas zakończenia zadania (*due date*)

$\bar{d}_j$ – termin krytyczny zakończenia zadania (*deadline*)
s_{kj} – czas przezbrojenia pomiędzy zadaniem J_k a J_j
w_j - waga (priorytet)

Charakterystyka zadań

Zadania ze względu na możliwość przerywania dzieli się na:
- zadania niepodzielne (nieprzerywalne), czyli takie, których wykonywanie nie może być przerywane
- zadana podzielne (przerywalne), jeżeli przerywanie wykonywania zadania jest dopuszczalne

Charakterystyka zbioru zadań

Cały zbiór zadań $\boldsymbol{J}$ scharakteryzowany jest poprzez liczbę tych zadań n.

W zbiorze zadań mogą być określone ograniczenia kolejnościowe – rozróżniane są pod tym kątem dwa rodzaje zbiorów zadań:
- zadania niezależne, pomiędzy którymi nie występują relacje częściowego porządku
- zadania zależne, gdy występuje przynajmniej jedna taka relacja

Charakterystyka zasobów

Zasoby klasyfikowane są pod wieloma względami.

Przede wszystkim dzieli się je na:
- dyskretne, czyli podzielne w sposób nieciągły (np. maszyny w systemie produkcyjnym, siła robocza)
- podzielne w sposób ciągły (np. paliwo, energia)

Można też wyróżnić zasoby:
- przywłaszczalne (jeżeli możliwe jest odebranie konkretnej jednostki takiego zasobu operacji aktualnie wykonywanej i przydzielenie jej gdzie indziej)
- nieprzywłaszczalne

Wyróżnia się trzy podstawowe kategorie:
- zasoby odnawialne (ograniczona jest liczba jednostek zasobu dostępnych w danej chwili), np. procesor, maszyna, robot, siła robocza
- zasoby nieodnawialne (ograniczona jest globalna ilość całkowitego zużycia zasobu), np. surowce, nakłady finansowe, energia
- zasoby podwójnie ograniczone (ograniczona jest dostępność w danej chwili i zużycie łączne), np. rozdział mocy z ograniczeniem zużycia całkowitego

Charakterystyka zasobów - parametry

Każdy zasób scharakteryzowany jest poprzez następujące parametry:
- dostępność (czasowe przedziały dostępności)
- ilość
- koszt

- dopuszczalne obciążenie jednostki zasobu (najczęściej przyjmuje się, że liczba operacji/zadań, które mogą być jednocześnie wykonywane przy użyciu tej jednostki jest równa jedności)

Charakterystyka zbioru zasobów

Cały zbiór zasobów jest określony poprzez podanie:
- rodzajów elementów (jednostek zasobu)
- ogólnej liczby jednostek zasobu każdego rodzaju

Charakterystyka maszyn

Ze względu na spełniane funkcje maszyny dzieli się na:
- równoległe (uniwersalne) – spełniające te same funkcje
- dedykowane (wyspecjalizowane) – różniące się spełnianymi funkcjami

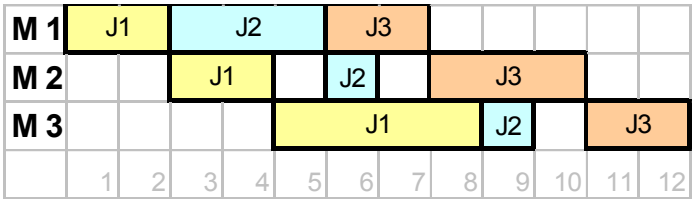
Istnieją trzy rodzaje maszyn równoległych:
- identyczne – każda z maszyn pracuje z taką samą prędkością
- jednorodne – maszyny różnią się prędkością, ale ich prędkość jest stała i nie zależy od wykonywanego zadania
- dowolne (niezależne) – czas wykonywania poszczególnych zadań na maszynach jest różny

W przypadku maszyn dedykowanych rozróżniane są następujące systemy obsługi zadań:
- przepływowy (flow-shop)
- ogólny / gniazdowy (job-shop)
- otwarty (open-shop)

System przepływowy

W systemie przepływowym każde zadanie musi przejść przez wszystkie maszyny w ściśle określonym porządku (każde zadanie składa się zatem z m operacji).

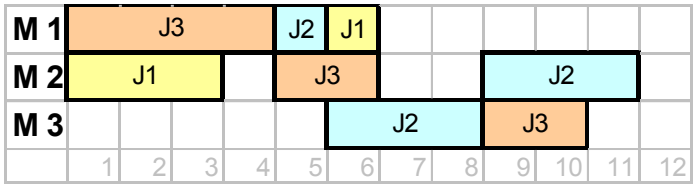
Przykład:
p_1 = (2,2,4), p_2 = (3,1,1), p_3 = (2,3,2)



System gniazdowy

W systemie gniazdowym (ogólnym) kolejność maszyn mających wykonać operacje jest różna, ale ściśle określona dla każdego zadania (zadania mogą mieć różną ilość operacji).

Przykład:
v_1 = (2,1), v_2 = (1,3,2), v_3 = (1,2,3)
p_1 = (3,1), p_2 = (1,3,3), p_3 = (4,2,2)



System otwarty

W systemie otwartym wytworzenie każdego wyrobu wymaga operacji na wszystkich maszynach, ale kolejność ich wykonywania jest dowolna i nieustalona.

Uszeregowanie

Rozwiązaniem problemu szeregowania zadań jest uszeregowanie, czyli ustalona kolejność wykonywania operacji na poszczególnych maszynach. Natomiast zbudowanie harmonogramu to wyznaczenie momentów, w których rozpoczyna się realizacja tych operacji.

W problemach, w których rozpatrywane są zasoby dodatkowe, należy również określić ich przydział.

W danym uszeregowaniu dla każdego zadania J_j można określić:
v_{ij} – sposób wykonania *i*-tej operacji zadania J_j (przydział do maszyn)
s_{ij} – termin rozpoczęcia wykonywania i-tej operacji zadania J_j
S_j – termin rozpoczęcia wykonywania zadania
C_j – termin zakończenia wykonywania zadania
L_j – nieterminowość zakończenia zadania
E_j – przyspieszenie rozpoczęcia zadania
F_j – czas przepływu zadania przez system
W_j – czas przestoju zadania przy przepływie przez system

Najczęściej spotykane kryteria (minimalizowane):
C_{max} – czas zakończenia wykonania wszyst kich zadań (długość uszeregowania)
L_{max} – maksymalna nieterminowość
T_{max} – maksymalne opóźnienie
c_{max} – maksymalny koszt wykonania zadania
$\Sigma w_j C_j$ – suma ważonych czasów zakończenia wykonania zadań
$\Sigma w_j T_j$ – suma ważonych opóźnień
Σc_j – całkowity koszt wykonania zadań

Klasyfikacja $\alpha|\beta|\gamma$

α - opisuje zbiór maszyn M i tym samym określa typ zagadnienia
β - opisuje zbiór zadań oraz dodatkowe specyficzne ograniczenia zagadnienia

γ - określa kryterium optymalizacji (czyli funkcję celu)

Symbol α

Symbol α jest złożeniem dwóch symboli $\alpha_1 \alpha_2$.

α_1 - charakteryzuje rodzaj maszyn

α_2 - określa liczbę maszyn w zbiorze $\boldsymbol{M}$

(Jeśli liczba ta nie jest określona z góry to używa się również symbolu pustego (Ø) mającego sens dowolnej liczby maszyn w systemie)

Symbol α_1

P – identyczne maszyny równoległe
Q – jednorodne maszyny równoległe
R – niezależne maszyny równoległe
F – system przepływowy (flow shop)
FP – system przepływowy permutacyjny
O – system otwarty (open shop)
J – system gniazdowy (job-shop)
Ø (symbol pusty) – zbiór $\boldsymbol{M}$ zawiera 1 maszynę

Symbol β

β może zawierać dowolny podzbiór symboli:
setup – występują przezbrojenia
batch - porcjowanie
no wait - bez czekania
pmtn – zadania można przerywać
prec - istnieje narzucony częściowy porządek technolo-giczny wykonywania zadań *(tree, outree, intree)*
r_j - zadania mają różne terminy zgłoszeń
inne...
(Znaczenie symboli jest dość często modyfikowane, wprowadzane są nowe)

Symbol γ

Symbol γ przyjmuje wartość jednej z symbolicznych postaci funkcji celu (kryterium).

Przykład: $F3|r_j|C_{max}$

Problem przepływowy - flow shop

Tw. Johnsona (1954r.)

Jeżeli w problemie $\boldsymbol{F2||C_{max}}$ (dwumaszynowy problem przepływowy z kryterium minimalizacji czasu wykonania wszystkich zadań):

$\min \{ p_{1i}, p_{2j} \} \leq \min \{ p_{2i}, p_{1j} \}$

to w uszeregowaniu optymalnym zadanie J_i jest wcześniej niż zadanie J_j

Algorytm Johnsona

Krok 1 : Zbuduj dwa zbiory: $N1$ i $N2$.

$N1 = \{J_j: p_{1j} < p_{2j}\}$ $N2 = \{J_j: p_{1j} \geq p_{2j}\}$

(zbiór ***N1*** zawiera zadania, których czas pierwszej operacji jest mniejszy niż drugiej, zbiór ***N2*** - pozostałe)

Krok 2 : Uporządkuj zbiory:

$N1$ - wg niemalejącej kolejności p_{1j}

$N2$ - wg nierosnącej kolejności p_{2j}

Krok 3 : Utwórz uszeregowanie optymalne π łącząc uporządkowane zbiory $N1$ i $N2$

(najpierw wszystkie zadania z N1, potem zadania z N2)

Problem przepływowy $F2||C_{max}$ - przykład

2 maszyny $\{M_1, M_2\}$

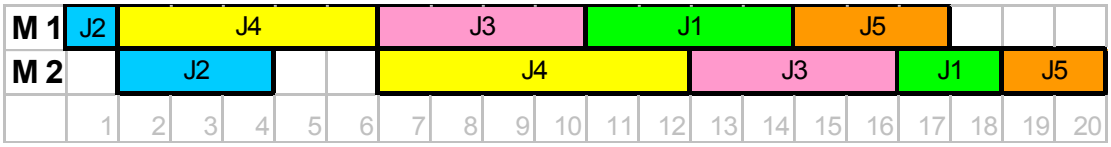
5 zadań $\{J_1, J_2, J_3, J_4, J_5\}$

$p_1 = (4,2)$ $p_4 = (5,6)$

$p_2 = (1,3)$ $p_5 = (3,2)$

$p_3 = (4,4)$

Rozwiązanie: $p = (J_2, J_4, J_3, J_1, J_5)$



Permutacyjne problemy przepływowe

Algorytm Johnsona wykorzystywany jest również do rozwiązywania szczególnych przypadków permutacyjnych problemów przepływowych.

Jeżeli w problemie $F3||C_{max}$ druga maszyna jest zdominowana przez jedną z pozostałych to można uzyskać rozwiązanie optymalne sprowadzając ten problem do problemu dwumaszynowego i następnie rozwiązać stosując algorytm Jonhsona.

Permutacyjne problemy przepływowe

Maszyna druga jest zdominowana jeżeli:

$$\min_j p_{kj} \geq \max_j p_{2j} \quad k = 1 \text{ lub } 3$$

Sprowadzenie do problemu dwumaszynowego:

- czas na pierwszej maszynie = $p_{1j} + p_{2j}$
- czas na drugiej maszynie = $p_{2j} + p_{3j}$

Problem ogólny (gniazdowy) - job shop

Problem $J2|o_j \leq 2|C_{max}$

- w problemie występują 2 maszyny

- każde zadanie składa się z jednej lub dwóch operacji
- kolejność wykonywania operacji danego zadania na poszczególnych maszynach jest określona
- minimalizowany jest czas wykonywania wszystkich zadań

Problem ogólny (gniazdowy) - job shop

Algorytm dla problemu $J2|o_j \leq 2|C_{max}$

Krok 1: Podziel zadania na podzbiory:

$\boldsymbol{I_1}$ (jedna operacja wykonywana na maszynie M_1)

$\boldsymbol{I_{1,2}}$ (pierwsza operacja na M_1, druga na M_2)

$\boldsymbol{I_2}$ (jedna operacja wykonywana na maszynie M_2)

$\boldsymbol{I_{2,1}}$ (pierwsza operacja na M_2, druga na M_1)

Krok 2: Uporządkuj zadania w zbiorach $I_{1,2}$ i $I_{2,1}$ algorytmem Johnsona (w I_1 i I_2 kolejność dowolna)

Krok 3: Przydziel zadania do maszyny M_1 w kolejności: $(I_{1,2}, I_1, I_{2,1})$, a do M_2 : $(I_{2,1}, I_2, I_{1,2})$

Problemy z maszynami równoległymi

Wykorzystywane są tu często algorytmy konstrukcyjne oparte na regułach priorytetowych (wolnej w danej chwili maszynie przydzielana jest gotowa do wykonania operacja z najwyższym priorytetem).

Przykładowe reguły priorytetowe:

- LPT *(Longest Processing Time)* - najwyższy priorytet dla operacji o najdłuższym czasie wykonywania
- SPT *(Shortest Processing Time)* - najwyższy priorytet dla operacji o najkrótszym czasie wykonywania
- EDD *(Earliest Due Date)* - najw. priorytet dla operacji o najwcześniejszym pożądanym terminie wykonania
- FCFS *(First Come First Served)* - najwyższy priorytet dla operacji, która najdłużej czeka na obsługę

Problemy grafowe

Podstawowe pojęcia (przypomnienie)

Graf nieskierowany (niezorientowany) to uporządkowana para *G = (V, E)* gdzie:
V - zbiór wierzchołków
E - zbiór krawędzi

$$E \subseteq \{\{u,v\} : u,v \in V \wedge v \neq u\}$$

Graf skierowany (digraf) to uporządkowana para *G = (V, A)* gdzie:
V - zbiór wierzchołków
A - zbiór łuków

$$A \subseteq \{(u,v) : u,v \in V\}$$

Krawędź (łuk) przyporządkowaną parze węzłów *u,v* nazywamy przyległą (incydentną) do obu tych łuków.
Dwie krawędzie (łuki) kończące się w jednym wierzchołku nazywamy przyległymi (sąsiednimi).
Dwa wierzchołki są przyległe (sąsiednie), jeśli istnieje krawędź pomiędzy nimi.

Stopień wierzchołka *v* to ilość kończących się w nim krawędzi (łuków) grafu, czyli ilość krawędzi (łuków) incydentnych z tym wierzchołkiem. Oznaczenie: *deg(v)*.
W przypadku grafów skierowanych często rozróżnia się stopień wejściowy oraz wyjściowy: *degIn(v)* , *degOut(v)* .

Grafem pełnym nazywany jest graf, w którym pomiędzy każdą parą wierzchołków istnieje krawędź.
Graf pełny o *n* wierzchołkach oznaczany jest K_n.

Drogą (ścieżką) nazywamy trasę w grafie polegająca na przemieszczaniu się od wierzchołka do wierzchołka po łączących je krawędziach. Wyznaczona jest poprzez ciąg krawędzi (wierzchołków).
Drogę (ścieżkę) nazywamy prostą jeżeli krawędzie (wierzchołki) się nie powtarzają.
Długość drogi (ścieżki) to ilość krawędzi (wierzchołków) ją tworzących.

Cykl to taka droga (ścieżka) prosta, której krawędź końcowa kończy się w początkowym wierzchołku trasy.

Spójność grafu
Nieskierowany graf *G* jest grafem spójnym jeżeli dla dowolnych dwóch jego wierzchołków istnieje droga łączaca te wierzchołki.
Skierowany graf *G* jest spójny jeżeli po zastąpieniu łuków krawędziami otrzymuje się graf spójny.
Grafem ściśle spójnym nazywa się skierowany graf *G*, dla którego dla dowolnych dwóch wierzchołków istnieje droga łącząca te wierzchołki.
Spójnością węzłową nazywamy minimalną liczbę węzłów grafu G, których usunięcie zmienia go w graf niespójny lub trywialny.
Złączem grafu nazywamy węzeł, którego usunięcie spowoduje wzrost liczby składowych spójności grafu.
Spójnością krawędziową nazywamy minimalną liczbę krawędzi grafu G, których usunięcie zmienia go w graf niespójny lub trywialny.

Mostem grafu nazywamy krawędź, której usunięcie spowoduje wzrost liczby składowych spójności grafu.

Cyklem Eulera nazywany jest taki cykl grafu *G*, który przechodzi dokładnie raz przez każdą krawędź (łuk) grafu *G*.
Graf posiadający cykl Eulera to graf eulerowski.
Cyklem Hamiltona nazywany jest taki cykl grafu *G*, który przechodzi dokładnie raz przez każdy wierzchołek grafu *G*.
Graf posiadający cykl Hamiltona to graf hamiltonowski.

Graf *H* uzyskany poprzez usunięcie części wierzchołków z *G*, wraz z kończącymi się w nich krawędziami nazywany jest **podgrafem** grafu *G*.
Graf *J* uzyskany poprzez usunięcie części krawędzi z grafu *G* (z zachowaniem wszystkich wierzchołków) nazywany jest **grafem częściowym** grafu *G*.

Klika to taki podzbiór wierzchołków danego grafu, z których każdy jest wierzchołkiem przyległym do wszystkich pozostałych z tego podzbioru (tworzą podgraf pełny).

Graf, który można przedstawić na płaszczyźnie tak, aby żadne krawędzie nie przecinały się, to **graf planarny**.
Graf płaski to takie przedstawienie graficzne grafu, w którym żadne dwie krawędzie nie przecinają się.

Drzewo to graf spójny acykliczny. Zbiór drzew określa się mianem lasu.
Drzewem rozpinającym w spójnym grafie nieskierowanym *G* nazywamy spójny acykliczny graf częściowy grafu *G*.

Wyznaczanie najkrótszej drogi w grafie:
algorytm Dijkstry
algorytm Minty'ego,
algorytm Forda
algorytm Bellmana-Kalaby
Tworzenie minimalnego drzewa rozpinającego:
algorytm Prima
algorytm Kruskala

Izomorfizm grafów

Problem izomorfizmu grafów polega na sprawdzeniu czy dane dwa grafy są „tożsame".
Rozwiązanie problemu izomorfizmu grafów sprowadza się do znalezienia przekształcenia wzajemnie jednoznacznego zbioru wierzchołków grafu pierwszego w zbiór wierzchołków grafu drugiego zachowującego relację przylegania wierzchołków.

Zastosowania:
- Rozpoznawanie i porównywanie obrazów
- Identyfikacja optyczna związków chemicznych
- Integracja danych z różnych czujników w obrazowaniu medycznym

Algorytmy:
- Dowodzenie nieizomorficzności grafów (poprzez sprawdzenie niezmienniczości grafów)
- NAUTY (No Automorphism?, Yes! :))
- Algorytm wykorzystujący heurystyczne oznaczanie wierzchołków
- VF-2 (M.Vento, P.Foggia)
- Algorytm Ullmana

Kolorowanie grafu

Klasyczne (wierzchołkowe) kolorowanie grafu to przyporządkowywanie wierzchołkom grafu liczb naturalnych w taki sposób, aby końce żadnej krawędzi nie miały przypisanej tej samej liczby.

Ze względów historycznych oraz dla lepszego zobrazowania problemu mówi się o kolorowaniu, przy czym różnym kolorom odpowiadają różne liczby.

Pokolorowaniem wierzchołków grafu nazywamy jedno konkretne przyporządkowanie liczb (kolorów) wierzchołkom. Pokolorowanie jest legalne (dozwolone), gdy końcom żadnej krawędzi nie przyporządkowano tej samej liczby (koloru).

Optymalnym pokolorowaniem grafu nazywamy legalne pokolorowanie zawierające najmniejszą możliwą liczbę kolorów.

Liczbą chromatyczną grafu G nazywamy liczbę $\chi(G)$ równą minimalnej liczbie kolorów potrzebnych do legalnego pokolorowania wierzchołków grafu G.

Pokolorowanie k kolorami nazywamy ***k*-pokolorowaniem**.

Graf jest ***k*-barwny**, jeśli istnieje l-pokolorowanie grafu G, gdzie $l \leq k$.
Każdy graf o n wierzchołkach jest zatem n-barwny.

Najmniejsza wartość k, dla której G jest k-barwny, jest **liczbą chromatyczną** $\chi(G)$ tego grafu.

Graf G jest ***k*-chromatyczny**, jeżeli $\chi(G) = k$.

Zbiór wierzchołków tego samego koloru nazywamy **klasą barwności**.

Zastosowania:
- Kolorowanie mapy
- Przydział częstotliwości
- Alokacja rejestrów
- Planowanie sesji egzaminacyjnej
- Planowanie rozgrywek sportowych

Algorytmy:
Problemy kolorowania grafów rozwiązywane są za pomocą dwóch podstawowych metod:
- metoda zbiorów niezależnych
- metody sekwencyjne

Graf dwudzielny – graf, którego zbiór wierzchołków można podzielić na dwa rozłączne zbiory tak, że w danym podzbiorze nie ma wierzchołków sąsiednich (połączonych krawędziami).
Jeśli pomiędzy wszystkimi parami wierzchołków należących do różnych zbiorów istnieje krawędź, graf taki nazywamy pełnym grafem dwudzielnym lub kliką dwudzielną i oznaczamy $K_{n,m}$ gdzie n i m oznaczają liczności zbiorów wierzchołków.

Zbiór dominujący grafu, to taki zbiór wierzchołków, że każdy wierzchołek tego grafu:
- należy do zbioru dominującego
lub
- jeden z jego bezpośrednich sąsiadów należy do zbioru dominującego.

Minimalny zbiór dominujący, to zbiór dominujący o najmniejszej liczności.

Zastosowania:
– ustalanie lokalizacji przystanków, stacji przesiadkowych
– planowanie połączeń lotniczych
– lokalizacja ośrodków ratownictwa i pierwszej pomocy
– trasy dostaw towarów w sieci hurtowni i sklepów
– kierowanie ruchem w sieciach komputerowych
– rozstawienie nadajników radiowych, sieci telefonii komórkowej itp.
– lokalizacja sił i baz wojskowych, zaopatrzeniowych, baz dowodzenia na danym terenie
– rozplanowanie układów komunikacyjnych w budynkach
– wyznaczenie dróg ewakuacji w budynkach

Pokrycie wierzchołkowe grafu G to taki podzbiór jego wierzchołków, że każda krawędź grafu G jest incydentna do jakiegoś wierzchołka z tego podzbioru.
Problem pokrycia wierzchołkowego – zagadnienie znajdowania w danym grafie pokrycia wierzchołkowego o najmniejszym rozmiarze (zawierającego możliwie najmniejszą liczbę wierzchołków).

Skojarzeniem grafu $G = (V,E)$ nazywamy podzbiór M krawędzi grafu E taki, że żadne dwie krawędzie w M nie są sąsiednie.
Skojarzenie doskonałe to taki podzbiór M krawędzi grafu G, w którym każdy wierzchołek grafu G jest końcem jednej z krawędzi podzbioru M. Skojarzenie doskonałe jest zawsze skojarzeniem największym**.**

Zbiór niezależny w grafie $G=(V,E)$, to zbiór takich wierzchołków $W \subseteq V$, które nie są wzajemnie sąsiednie.
Problem największego zbioru niezależnego polega na znalezieniu w danym grafie zbioru niezależnego o maksymalnej liczbie wierzchołków.

Problem komiwojażera (TSP)

Dany jest zbiór *n* miast.
Komiwojażer musi odwiedzić wszystkie miasta w każdym z nich będąc tylko raz i powrócić do miasta z którego wyruszył.
Znając odległości pomiędzy każdą parą miast należy tak ułożyć trasę komiwojażera, aby jej długość była minimalna.
Jest to równoznaczne ze znalezieniem cyklu Hamiltona o minimalnej długości w *n*-wierzchołkowej sieci pełnej.

Problem komiwojażera – algorytmy przybliżone
Algorytmy konstrukcyjne:
- algorytm najbliższego sąsiada
- algorytm zachłanny
- algorytm wymiatania
- algorytmy wstawiania wierzchołka
- algorytm wykorzystujący technikę klastrowania

Algorytmy poprawy:
- algorytmy generujące rozwiązania λ-optymalne
- algorytm Lina-Kernighana

Problemy pochodne do TSP
- m-TSP czyli problem z *m* komiwojażerami (ang. multiple traveling salesman problem)
- Problem podróżującego polityka (ang. travelling politician problem)
- Problem komiwojażera z uwzględnieniem wąskiego gardła (Bottleneck TSP)

Problem komiwojażera z uwzględnieniem wąskiego gardła (Bottleneck TSP)

Dany jest zbiór *n* miast.
Komiwojażer musi odwiedzić wszystkie miasta w każdym z nich będąc tylko raz i powrócić do miasta z którego wyruszył.
Znając odległości pomiędzy każdą parą miast należy tak ułożyć trasę komiwojażera, aby wchodzące w jej skład najdłuższe połączenie pomiędzy miastami było minimalne.
Dane: graf *G=(V, E)* oraz macierz wag *C*
Znajdujemy cykl Hamiltona $\pi = (v_1, v_2, \ \ldots \ , v_n, v_1)$
taki, że: $\max\left(C(v_n, v_1) \cup \{C(v_i, v_{i+1}) : i = 1,2,...,n-1\}\right)$ jest minimalne

Problem chińskiego listonosza (CPP = Chinese Postman Problem)

W swojej pracy listonosz wyrusza z poczty, dostarcza przesyłki adresatom, by na końcu powrócić na pocztę.

Aby wykonać swoją pracę musi przejść po każdej ulicy w swoim rejonie co najmniej raz. Oczywiście chciałby, aby droga, którą przebędzie, była możliwie najkrótsza.
Problem chińskiego listonosza

Jeśli dany graf posiada cykl Eulera, to istnieje taka droga, która zaczyna i kończy się w tym samym punkcie i wymaga przejścia po każdej ulicy dokładnie raz.

Jeśli graf nie posiada cyklu Eulera, to listonosz będzie zmuszony przejść niektórymi ulicami wielokrotnie. Rozwiązanie jest więc cyklem, w którym suma długości krawędzi wybranych więcej niż raz jest możliwie najmniejsza.

Rozwiązanie problemu chińskiego listonosza istnieje zawsze o ile graf jest spójny.

Zastosowanie problemu w praktyce
- praca listonosza, firm kurierskich
- trasy autobusów, śmieciarek czy pługów śnieżnych
- diagnostyka sieci komputerowych, linii elektrycznych
- zwiedzania miasta, zwiedzanie muzeum

Algorytmy

Jeśli dany graf posiada cykl Eulera rozwiązaniem będzie znalezienie tego cyklu (algorytm Fleury'ego). Jeśli nie, to trzeba przejść niektórymi krawędziami wielokrotnie (algorytm CPP).

Algorytm Fleury'ego

Wybieramy dowolny wierzchołek jako punkt startowy
Poruszamy się kolejno wierzchołkami po dowolnej trasie pod warunkiem, że wybieramy krawędź, która nie jest mostem (spowoduje to rozspójnienie grafu), chyba, że nie można inaczej. Kontynuujemy krok 2, aż osiągniemy punkt startowy

Algorytm CPP

Szukamy wierzchołków o stopniu nieparzystym (musi być ich parzysta ilość), jeśli wszystkie wierzchołki są stopnia parzystego to przechodzimy do kroku 5
Znajdujemy wszystkie możliwe ich połączenia (tzw. parowanie) tak, aby w konsekwencji utworzyć graf tylko z parzystymi wierzchołkami
Dla każdego połączenia znajdujemy najkrótszą ścieżkę, jaką należy pokonać, aby dostać się z pierwszego wierzchołka pary do drugiego.

Wybieramy to parowanie, które ma najmniejszy całkowity koszt i „dodajemy” odpowiednie krawędzie (otrzymamy w rezultacie graf z wierzchołkami o stopniach parzystych)
Stosujemy algorytm Fleury'ego do znalezienia rozwiązania

Całkowity koszt optymalnego przejścia danego grafu to suma kosztów wszystkich krawędzi grafu plus suma wszystkich dodanych przez nas krawędzi.

Zagadnienie przydziału (assignment problem)

Mamy *n* pracowników i *n* stanowisk pracy (maszyn). Znamy koszt c_{ij} przydzielenia j-tego pracownika do i-tego stanowiska (podana jest macierz kosztów).
Zadanie polega na takim przydziale pracowników do stanowisk, który będzie minimalizować całkowity koszt przydziału.

Zmienna decyzyjna:

$$x_{ij} = \begin{cases} 1, & \text{jeśli j-ty pracownik został przydzielony do i-tego stanowiska} \\ 0, & \text{w przeciwnym przypadku} \end{cases}$$

Problem przydziału można sformułować w postaci następującego liniowego zadania decyzyjnego:
Znajdź takie wartości zmiennych x_{ij}, aby:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \rightarrow \min$$

przy warunkach (ograniczeniach):

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,2,...,n$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,2,...,n$$

$$x_{ij} = \begin{cases} 1 \\ 0 \end{cases} \qquad i = 1,2,...,n \qquad j = 1,2,...,n$$

Algorytm węgierski dla zagadnienia przydziału:

Krok 1. Znaleźć minimalny element w każdym wierszu macierzy *n x n*. Zbudować nową macierz przez odjęcie tych elementów od każdego elementu danego wiersza. To samo wykonać dla kolumn tej macierzy. (redukcja macierzy)

Krok 2. Narysować minimalną liczbę linii (pionowych i/lub poziomych), tak żeby pokryć wszystkie zera w tej macierzy. Gdy liczba linii wynosi *n* to mamy rozwiązanie, gdy liczba linii jest mniejsza niż *n* należy przejść do kroku 3.

Krok 3. Znaleźć najmniejszy niezerowy element w części niezakreślonej. Odjąć go od niezakreślonych elementów macierzy i dodać do podwójnie zakreślonych. Przejść do kroku 2.

Następnie należy podać rozwiązanie, czyli utworzyć macierz przydziału oraz podać koszt tego optymalnego rozwiązania (na podstawie macierzy początkowej).

Kwadratowe zagadnienie przydziału (**QAP** = Quadratic assignment problem)

Mamy dane *n* obiektów (fabryk, magazynów), które należy umieścić w *n* lokalizacjach. Znane są odległości pomiędzy poszczególnymi lokalizacjami oraz zależności (przepływ towarów) pomiędzy obiektami. Należy tak rozmieścić obiekty, żeby zminimalizować koszty (związane z przepływem na określonych dystansach).

Metody rozwiązywania problemów

1. Metody dokładne
   - programowanie liniowe
   - programowanie dynamiczne
   - metoda podziału i ograniczeń
   - algorytmy wyspecjalizowane
2. Metody przybliżone
   - metody przeszukiwania sąsiedztwa
   - metody oparte na schemacie podziału i ograniczeń
   - metody czerpiące inspiracje z natury

   - konstruowanie rozwiązania poprzez generowanie i przeszukiwanie grafu stanów

Algorytmy przybliżone

Ze względu na sposób prowadzenia obliczeń można wyróżnić dwa podstawowe typy algorytmów (opartych na różnych metodach):
- algorytmy konstrukcyjne (generują krokowo pojedyncze rozwiązanie)
- algorytmy poprawy (w kolejnych iteracjach poprawiają rozwiązanie początkowe)

Metody przeszukiwania sąsiedztwa

Metody te polegają na analizie rozwiązań leżących w pewnym lokalnym otoczeniu (sąsiedztwie) wybranego rozwiązania bieżącego. W wyniku analizy wybierane jest rozwiązanie, które zastępuje rozwiązania bieżące i staje się kolejnym źródłem sąsiedztwa. Proces ten jest następnie powtarzany.
- poszukiwanie zstępujące (descending search)
- symulowane wyżarzanie (simulated annealing)
- poszukiwanie z zakazami (tabu search)

Metody czerpiące inspiracje z natury
- algorytmy ewolucyjne (evolutionary search/algorithms)
- podejście immunologiczne (artificial immune system)
- poszukiwanie mrówkowe (ant search)
- poszukiwanie biochemiczne (DNA method)

Algorytmy zachłanne (greedy)

W algorytmach zachłannych rozwiązanie tworzone jest stopniowo przy wykorzystaniu optymalizacji lokalnej. W danej chwili podejmowana jest taka decyzja, która wydaje się najlepsza, mimo iż później ten wybór może się okazać błędny. Funkcje optymalizacji lokalnej (inaczej funkcje preferencji) dobierane są w sposób intuicyjny.

Metody przeszukiwania sąsiedztwa

Metody te polegają na analizie rozwiązań leżących w pewnym lokalnym otoczeniu (sąsiedztwie) wybranego rozwiązania bieżącego. W wyniku analizy sąsiedztwa wybierane jest rozwiązanie, które zastępuje rozwiązania bieżące i staje się kolejnym źródłem sąsiedztwa. Proces ten jest następnie powtarzany. Początkowe rozwiązanie bazowe generowane jest losowo lub za pomocą algorytmów konstrukcyjnych.

Sąsiedztwo (lokalne otoczenie) - zbiór rozwiązań otrzymanych z pewnego rozwiązania bazowego (przy zastosowaniu określonego mechanizmu generowania)
Cechy jakimi powinno się charakteryzować sąsiedztwo:

- korelacja
- tworzenie rozwiązań dopuszczalnych
- poprawa rozwiązań
- odpowiednia wielkość
- łączność

Przykładowe rozwiązanie bazowe: $(J_1, J_2, J_3, J_4, ..., J_{n-1}, J_n)$
Najprostsze sąsiedztwo:
$(J_2, J_1, J_3, J_4, ..., J_{n-1}, J_n)$
$(J_1, J_3, J_2, J_4, ..., J_{n-1}, J_n)$
$(J_1, J_2, J_4, J_3, ..., J_{n-1}, J_n)$
.....
$(J_1, J_2, J_3, J_4, ..., J_n, J_{n-1})$

Poszukiwanie zstępujące (descending search)

Przeszukiwanie zstępujące jest historycznie najstarszą i jedną z prostszych metod przeszukiwania sąsiedztwa.

Algorytm przeszukiwania zstępującego polega na tym, że w każdym kroku przeglądany jest cały podzbiór rozwiązań należący do sąsiedztwa. Następnie wybierane jest z niego rozwiązanie z najmniejszą wartością funkcji celu. Jeżeli to rozwiązanie jest lepsze od bieżącego, to staje się ono nowym rozwiązaniem. Proces ten jest kontynuowany dopóki wartość celu maleje. Trajektoria poszukiwań zbiega się zatem monotonicznie w kierunku ekstremum lokalnego, gdzie algorytm kończy swoje działanie.

Algorytm:

Krok 1. Wygeneruj początkowe rozwiązanie bazowe x^0 i podstaw $k := 0$

Krok 2. Wygeneruj sąsiedztwo $N(x^k)$ bieżącego rozwiązania bazowego x^k

Krok 3. Przeszukaj całe sąsiedztwo $N(x^k)$ i wybierz rozwiązanie $\tilde{x}$ z najmniejszą wartością funkcji celu $f(x)$

Krok 4. Jeżeli $f(\tilde{x}) < f(x^k)$ czyli znalezione rozwiązanie jest lepsze od bazowego, to przyjmij je za nowe rozwiązanie bazowe $x^{k+1} := \tilde{x}$, następnie podstaw $k := k+1$ i przejdź do kroku 2. W przeciwnym przypadku zakończ obliczenia (x^k stanowi rozwiązanie zadania).

Symulowane wyżarzanie

Termin wyżarzanie pochodzi z metalurgii i dotyczy termodynamicznego procesu studzenia. Jeżeli płynna stal jest schładzana wystarczająco wolno, to ma tendencję do krzepnięcia w strukturze o minimalnej energii. Metoda symulowanego wyżarzania opiera się na analogii do tego procesu. Jej celem jest wyprowadzenie trajektorii poszukiwań z ekstremum lokalnego lub uniknięcie takiego ekstremum.

W metodzie tej, z sąsiedztwa aktualnego rozwiązania nie jest wyszukiwane najlepsze rozwiązanie, tylko wybiera się rozwiązanie w sposób losowy. Jeżeli to wylosowane rozwiązanie jest lepsze od bieżącego, to staje się ono nowym rozwiązaniem. W przeciwnym przypadku znalezione rozwiązanie zastępuje poprzednie z pewnym prawdopodobieństwem zależnym m.in. od współczynnika nazywanego temperaturą. Temperatura jest zmieniana w czasie iteracji według tzw. schematu chłodzenia.

Algorytm:

Krok 1. Wygeneruj początkowe rozwiązanie bazowe x^0, ustal temperaturę początkową T_0 i podstaw $k := 0$

Krok 2. Wybierz (wylosuj) rozwiązanie $\tilde{x} \in N(x^k)$

Krok 3. Jeżeli $f(\tilde{x}) < f(x^k)$ czyli znalezione rozwiązanie jest lepsze od bazowego, to przyjmij je za nowe rozwiązanie bazowe $x^{k+1} := \tilde{x}$ i przejdź do kroku 6. W przeciwnym przypadku przejdź do kroku 4.

Krok 4. Wylosuj liczbę *r* z przedziału [0,1] i oblicz prawdopodobieństwo

$$P(x^k, \tilde{x}) = \min\left\{1, \exp\left(\frac{f(x^k) - f(\tilde{x})}{T_k}\right)\right\}$$

Jeżeli $r \le P(x^k, \tilde{x})$ przyjmij $\tilde{x}$ jako nowe rozwiązanie bazowe $x^{k+1} := \tilde{x}$. W przeciwnym przypadku podstaw $x^{k+1} := x^k$.

Krok 5. Oblicz nową wartość temperatury T_{k+1}. Podstaw $k := k+1$.

Krok 6. Sprawdź warunek stopu. Jeżeli jest spełniony, to zakończ obliczenia (rozwiązanie stanowi najlepsze z dotychczasowych rozwiązań bazowych). Jeżeli warunek stopu nie jest spełniony to przejdź do kroku 2.

Schemat studzenia:

- geometryczny $T_{k+1} = \lambda_k T_k$
- logarytmiczny $T_{k+1} = \frac{T_k}{1 + \lambda_k T_k}$

Warunki stopu:

- osiągnięcie maksymalnej ilości iteracji
- przekroczenie założonego czas przebiegu
- zbliżenie temperatury do zera
- osiągnięcie określonej wartości funkcji celu rozwiązania bazowego

Poszukiwanie z zakazami (tabu search)

Poszukiwanie z zakazami jest metodą przeszukiwania sąsiedztwa, w której wykorzystywane są pewne techniki, aby uniknąć powrotu w już przebadane obszary przestrzeni rozwiązań. W tym celu wprowadza się pamięć historii poszukiwań. Najczęściej używana jest pamięć krótkoterminowa, zwana listą zabronień.
Sąsiedztwo w tej metodzie jest definiowane poprzez ruchy które można wykonać z danego rozwiązania (ruch transformuje jedno rozwiązanie w inne rozwiązanie).

Zabronione są ruchy związane z ostatnio znalezionymi rozwiązaniami. Lista zabronień przechowuje przez pewien okres czasu najświeższe rozwiązania (wybrane atrybuty, ruchy itp.) traktując je wszystkie jako formę zabronienia dla ruchów wykonywanych w przyszłości.

Istnieje też możliwość, że pewne zabronione ruchy mogą zostać dopuszczone (jeżeli tzw. funkcja aspiracji uzna taki ruch za wystarczająco korzystny).

KLASY ZŁOŻONOŚCI PROBLEMÓW

Problemy decyzyjne a optymalizacyjne

Problem **decyzyjny** jest sformułowany w postaci pytania, na które odpowiedź brzmi "tak" lub "nie". *(Pytanie: „Czy istnieje droga komiwojażera o koszcie mniejszym od 130?")*

Problem **optymalizacyjny**, to taki problem, w którym należy ekstremalizować pewną funkcję celu.*(„Znajdź drogę komiwojażera o najmniejszym koszcie.")*

Z danym problemem optymalizacyjnym można związać odpowiadający mu problem decyzyjny. Taki problem decyzyjny jest obliczeniowo nie trudniejszy, niż odpowiadający mu pierwotny problem optymalizacyjny.

Jeśli problem decyzyjny jest obliczeniowo „trudny", to „trudny" jest również odpowiadający mu problem optymalizacyjny.

Problem decyzyjny

Problem decyzyjny Π to pewien zbiór parametrów oraz pytanie, na które odpowiedź brzmi „tak" lub „nie". Ustalając wartości tych parametrów otrzymujemy instancję *I* (konkretny przypadek problemu).

Problem można również zdefiniować jako zbiór instancji D_Π oraz jego podzbiór Y_Π, zawierający instancje, na które odpowiedź brzmi „tak".

Instancja problemu

Dane instancji $I \in D_\Pi$ zapisuje się za pomocą skończonego łańcucha *x(I)* symboli należących do z góry określonego alfabetu Σ zgodnie z ustaloną regułą kodowania.

Reguły kodowania powinny być jednoznaczne i zwięzłe.

Przez rozmiar instancji *N(I)* rozumiana jest długość łańcucha *x(I)*, czyli |*x(I)*|.

Alfabet Σ to dowolny skończony i niepusty zbiór symboli.

Słowo *w* nad danym alfabetem Σ to dowolny skończony ciąg (łańcuch) elementów Σ. Długość słowa |*w*| to liczba symboli alfabetu w słowie.

Deterministyczny automat skończony (**DFA**) opisany jest poprzez piątkę $A=(Q,\Sigma,\delta,q_{0t},F)$, gdzie:

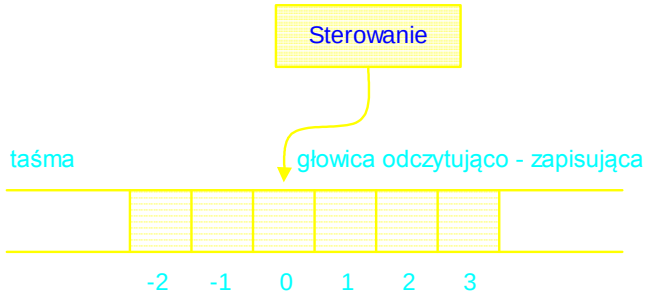
- Q jest skończonym zbiorem stanów,
- Σ jest alfabetem,
- $\delta: Q\times\Sigma\to Q$ jest funkcją przejścia (parze (q,a) przypisuje nowy stan, w którym znajdzie się automat po przeczytaniu symbolu *a* w stanie *q*)
- $q_0 \in Q$ jest stanem początkowym automatu
- $F \subset Q$ jest zbiorem stanów akceptujących (końcowych)

Modele obliczeń

- deterministyczna (jednotaśmowa) maszyna Turinga (DTM)
- k-taśmowa deterministyczna maszyna Turinga (kDTM)
- model RAM (random access machine)
- niedeterministyczna maszyna Turinga (NDTM)
- niedeterministyczna kDTM
- niedeterministyczna maszyna RAM
- maszyna Turinga z wyrocznią (orzecznikiem) (OTM)

Deterministyczna maszyna Turinga

Jednotaśmowa maszyna Turinga (DTM) to sterowanie, głowica odczytująco-zapisująca i taśma zawierająca nieskończoną ilość komórek ponumerowanych: ...-2, -1, 0, 1, 2, ...



Program dla DTM jest określony przez podanie:

- skończonego zbioru symboli taśmy Γ, podzbioru symboli wejściowych $\Sigma \in \Gamma$ i wyróżnionego symbolu pustego $b \in (\Gamma - \Sigma)$
- skończonego zbioru stanów Q, zawierającego wyróżniony stan początkowy q i dwa wyróżnione stany końcowe q_y (odpowiedź „tak") i q_n (odpowiedź „nie")
- funkcji przejść δ: $(Q - \{q_y, q_n\}) \times \Gamma \rightarrow Q \times \Gamma \times \{-1,+1\}$

Przykład DTM: Sprawdzenie czy liczba jest parzysta (liczba zapisana w postaci binarnej)

Program dla DTM:

- zbiór symboli taśmy $\Gamma = \{0,1,b\}$,
- podzbiór symboli wejściowych $\Sigma = \{0,1\}$,
- zbiór stanów $Q = \{q_0, q_1, q_n, q_y\}$,
- funkcja przejść:

q \ s	0	1	b
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_1, b, -1)$
q_1	$(q_y, 0, -1)$	$(q_n, b, -1)$	$(q_n, b, -1)$

Niedeterministyczna maszyna Turinga

Niedeterministyczna maszyna Turinga (NDTM), to DTM wyposażona dodatkowo w moduł generujący. Wykonanie programu dla danego łańcucha (słowa) $x(I)$ przebiega w dwóch etapach:

1) moduł generujący generuje i zapisuje na taśmie w komórkach -1,-2,.... łańcuch S symboli ze zbioru Γ

2) NDTM sprawdza (w taki sam sposób jak DTM) czy wygenerowany łańcuch S spełnia warunki określone w pytaniu instancji I

Rozwiązywanie problemów przez DTM i NDTM

Twierdzenie:

Jeśli niedeterministyczna jednotaśmowa maszyna Turinga rozwiązuje decyzyjny problem Π w czasie wielomianowym, wówczas istnieje wielomian p taki, że jednotaśmowa

deterministyczna maszyna Turinga rozwiązuje ten problem w czasie $O(2^{p(N(I))})$, gdzie $I \in D_{\Pi}$.

Klasa P i NP

Klasę P tworzą wszystkie problemy decyzyjne, które w co najwyżej wielomianowym czasie rozwiązuje deterministyczna maszyna Turinga.

Klasa NP problemów decyzyjnych zawiera wszystkie problemy decyzyjne, które w co najwyżej wielomianowym czasie rozwiązuje niedeterministyczna maszyna Turinga.

$P \subset NP$ $P \neq NP$?

Transformacja wielomianowa

Transformacja wielomianowa problemu Π_2 do problemu Π_1 (co zapisujemy $\Pi_2 \propto \Pi_1$) to taka funkcja f: $D_{\Pi 2} \rightarrow D_{\Pi 1}$, która spełnia warunki:

- dla każdej instancji $I_2 \in D_{\Pi 2}$ odpowiedź brzmi "tak" wtedy i tylko wtedy, gdy dla instancji f(I_2) odpowiedź brzmi również "tak"
- czas obliczania funkcji f przez DTM dla każdej instancji $I_2 \in D_{\Pi 2}$ jest ograniczony od góry przez wielomian od $N(I_2)$

NP-zupełność problemu decyzyjnego

Mówimy, że problem decyzyjny Π_1 jest **NP-zupełny**, jeśli $\Pi_1 \in NP$ i dla każdego innego problemu decyzyjnego $\Pi_2 \in NP$ zachodzi: $\Pi_2 \propto \Pi_1$.

Wynika stąd, że dla wykazania NP-zupełności badanego problemu decyzyjnego wystarczy przetransformować do niego wielomianowo dowolny znany problem NP-zupełny.

NP-trudność problemu optymalizacyjnego

Dla wykazania trudności rozważanego problemu optymalizacyjnego wystarcza wykazanie NP-zupełności odpowiadającego mu problemu decyzyjnego.

Mówimy wtedy, że dany problem optymalizacyjny jest **NP-trudny**.

Techniki dowodzenia NP-zupełności

Istnieją trzy zasadnicze techniki dowodzenia NP-zupełności problemów decyzyjnych:

- ograniczenie
- lokalna zamiana
- projektowanie części składowych

Klasy problemów decyzyjnych