PROBLEMY I ALGORYTMY

Na wstępie należy zdefiniować podstawowe pojęcia jakimi będziemy się posługiwać, a mianowicie pojęcia **problemu** i **algorytmu**.

Definicja 1 *Problem jest to zbiór danych wejściowych i ich definicje oraz pytanie lub polecenie do wykonania.*

Problemy będziemy najczęściej oznaczać symbolem π (np. π_1, π_2, itd.) lub literą P (np. P_1, P_2, itd.)

Przykłady problemów

Ogólny problem optymalizacji:

Dane: Funkcja $f : X \to \mathbb{R}$, zbiór X
Polecenie: Znaleźć $x^* \in X$ takie, że $f(x^*) = \min_{x \in X} f(x)$.

Problem programowania liniowego:

Dane: Funkcja $f : X \to \mathbb{R}$ taka, że $f(x) = c^T x$,
zbiór $X = \{x \in \mathbb{R}^n : Ax \leq b\}$, gdzie
$c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ i $A \in \mathcal{M}_{m \times n}$ oraz $n, m \in \mathbb{Z}$.
Polecenie: Znaleźć $x^* \in X$ takie, że $f(x^*) = \min_{x \in X} f(x)$.

Przykłady problemów optymalizacji kombinatorycznej

Problem Komiwojażera (PK)

Dane:

n – liczba miast, $n \in \mathbf{Z}^+$,

$c_{ji},\ i,j \in \{1,\ldots,n\},\ i \neq j$ – odległość między miastem i a miastem j,

$$c_{ji} = c_{ij},\ c_{ji} \in \mathbf{R}^+.$$

Zadanie:

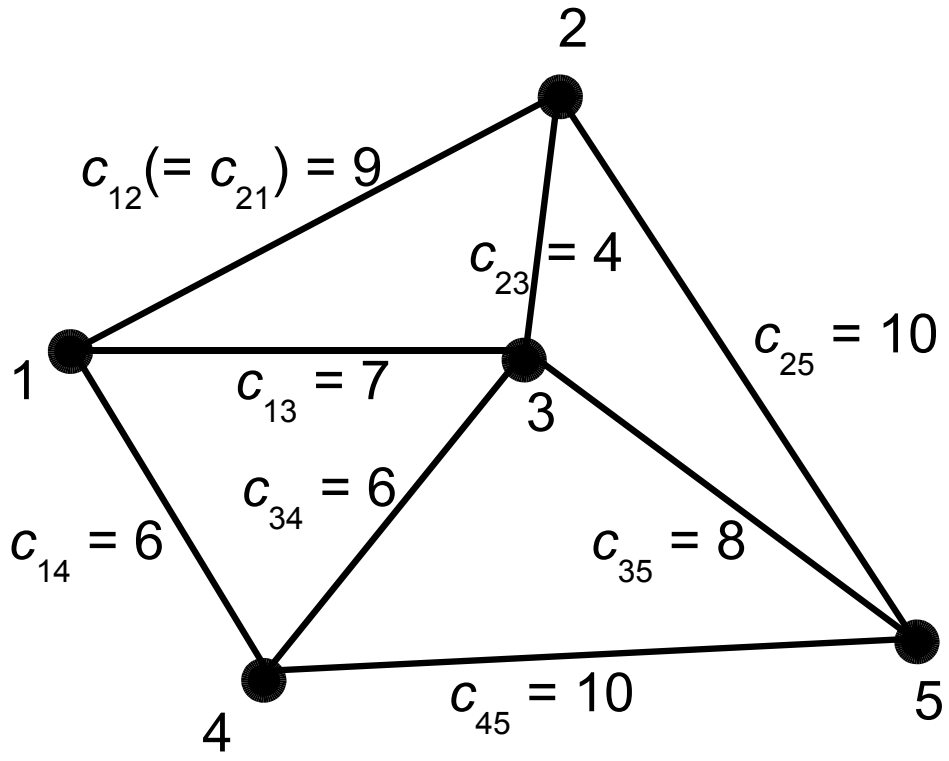
Znaleźć permutację miast π^*, dla której

$$TC(\pi^*) = \sum_{j=1}^{n-1} c_{\pi^*(j)\pi^*(j+1)} + c_{\pi^*(n)\pi^*(1)} \longrightarrow \min.$$

n, c_{ij} $(i,j \in \{1,\ldots,n\})$ – parametry PK;
permutacja π – rozwiązanie PK (π^* – rozwiązanie optymalne);
TC – funkcja celu.

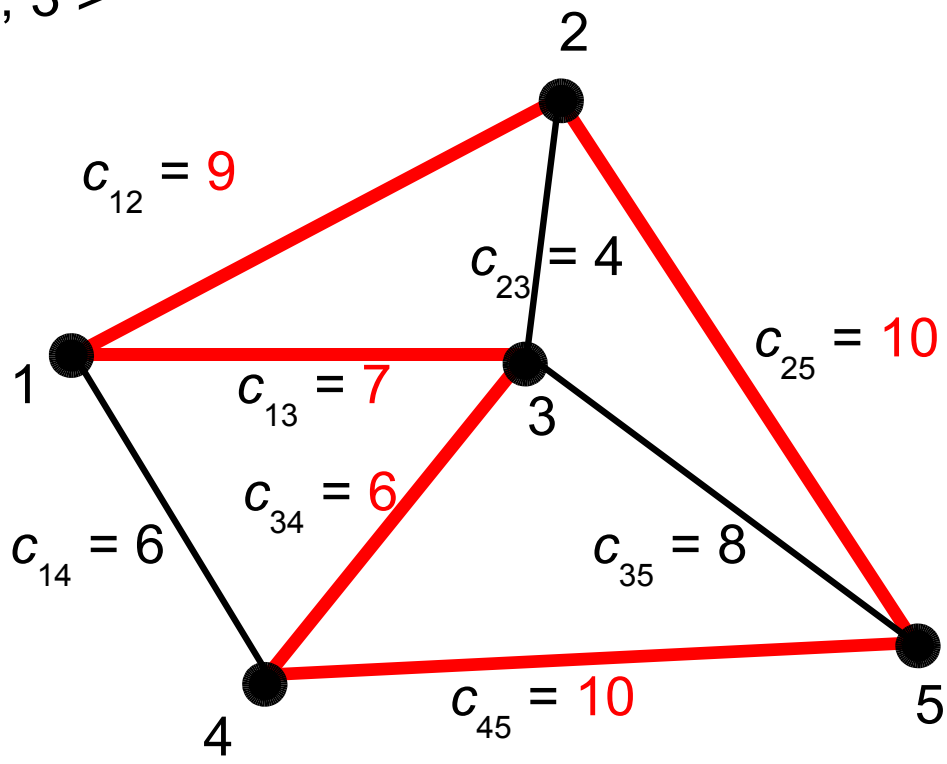
Przykładowa instancja PK

$n = 5$



Przykładowe rozwiązanie π_1 instancji PK

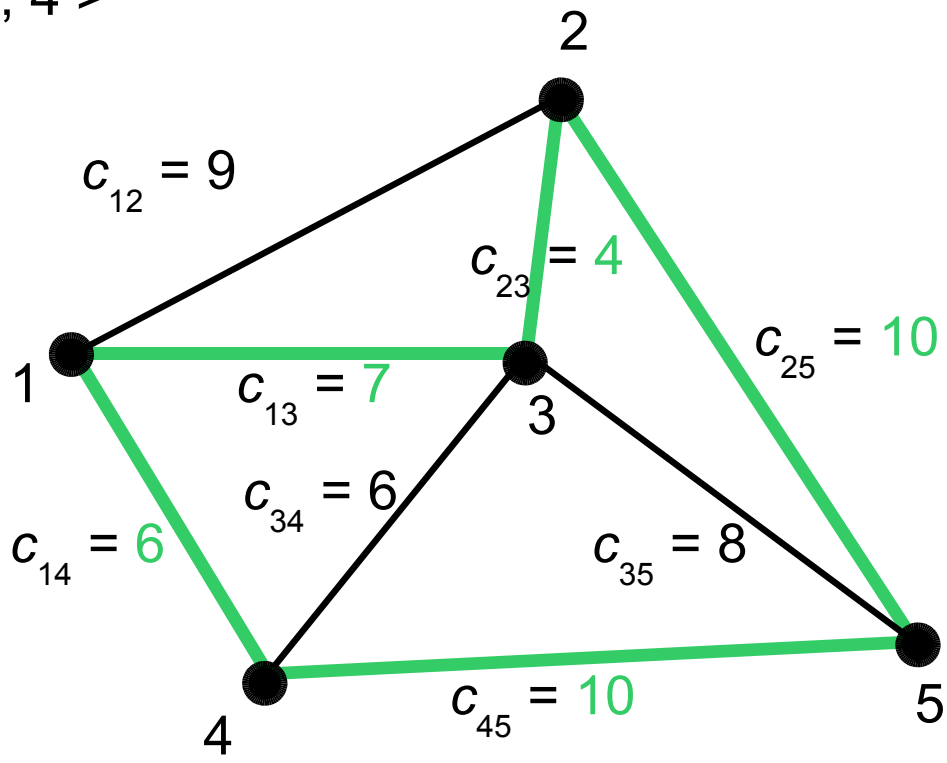
$\pi_1 = < 1, 2, 5, 4, 3 >$



$TC(\pi_1) = 9 + 10 + 10 + 6 + 7 = \mathbf{42}$

Rozwiązanie π_2

$\pi_2 = < 1, 3, 2, 5, 4 >$



$TC(\pi_2) = \mathbf{37} < TC(\pi_1) = \mathbf{42}$

Problem Plecakowy (PP)

Dane:
$A = \{a_1, a_2, \ldots, a_n\}$ – zbiór n przedmiotów, $n \in \mathbf{Z}^+$,
s_j, $j = 1, \ldots, n$ – rozmiar przedmiotu a_j, $s_j \in \mathbf{Z}^+$,
w_j, $j = 1, \ldots, n$ – wartość przedmiotu a_j, $w_j \in \mathbf{Z}^+$,
B – rozmiar plecaka.

Zadanie: Znaleźć podzbiór $A' \subseteq A$ zbioru przedmiotów taki, że

$$TS = \sum_{j \in A'} s_j \leq B \quad \text{oraz}$$

$$TW = \sum_{j \in A'} w_j \longrightarrow \max.$$

n, A, s_j, w_j $(j = 1, \ldots, n)$ – parametry PP;
podzbiór A' – rozwiązanie PP;
TW – funkcja celu (TS – dodatkowe ograniczenie).

Przykładowa instancja PP

$n = 5$, $A = \{a_1, a_2, a_3, a_4, a_5\}$,

j	1	2	3	4	5
s_j	5	3	2	4	5
w_j	3	4	2	6	1

$B = 10$.

Rozwiązanie (dopuszczalne) optymalne: $A' = \{a_2, a_3, a_4\}$,

$\sum_{j \in A'} s_j = 9 \leq B = 10$,

$\sum_{j \in A'} w_j = 12$.

Problem szeregowania zadań $1|r_j|C_{\max}$

Dane:
$J = \{J_1, J_2, \ldots, J_n\}$ – zbiór n zadań, $n \in \mathbf{Z}^+$,
pojedynczy procesor mający zrealizować wszystkie zadania $J_1, \ldots, J_n$,
p_j, $j = 1, \ldots, n$ – czas wykonywania (długość) zadania J_j, $p_j \in \mathbf{R}^+$,
r_j, $j = 1, \ldots, n$ – termin dostępności zadania J_j, $r_j \in \mathbf{R}^+ \cup \{0\}$,

Zadanie: Znaleźć permutację zadań π^*, dla której

$$\mathbf{C}_{\max} = \max_{j \in J}\{C_{\pi^*(j)}\} \longrightarrow \min,$$

gdzie $C_{\pi^*(j)} = S_{\pi^*(j)} + p_{\pi^*(j)}$ jest czasem zakończenia wykonywania zadania zajmującego j-tą pozycję w π^*; $S_{\pi^*(j)} = \max\{C_{\pi^*(j-1)}, r_{\pi^*(j)}\}$ – czas rozpoczęcia wykonywania zadania $\pi^*(j)$, $C_{\pi^*(0)} = 0$, $j = 1, \ldots, n$.

Przykłady problemów cd.

Problem podziału zbioru (**PARTITION**):

Dane Zbiór $N = \{1, ..., m\}$ m elementów o wartościach $x_j > 0$, $j \in N$, taki, że $\sum_{j=1}^{m} x_j = 2B$.
Pytanie: Czy istnieje podzbiór $X \in N$ taki, że $\sum_{j \in X} q_j = B$?

Problem spełnialności wyrażeń logicznych (boolowskich) (**SATISFIABILITY**):

Dane: Funkcja boolowska $f : \{0,1\}^n \to \{0,1\}$
(innymi słowy $f(x_1, ..., x_n)$ jest funkcją boolowską zmiennych logicznych $x_1, ..., x_n$)
Pytanie: Czy istnieje przyporządkowanie wartości 0 i 1 (logicznego fałszu i prawdy) do zmiennych $x_1, ..., x_n$ takie, że $f(x_1, ..., x_n) = 1$?

Na czym polega trudność rozwiązania problemów optymalizacji kombinatorycznej?

Liczność zbioru rozwiązań $\mathbf{X}$ większości realnie istniejących problemów jest tak duża, że:

(a) procedura polegająca na sprawdzeniu wszystkich możliwych rozwiązań problemu (tzw. przegląd zupełny), i wyznaczenie wśród nich rozwiązania optymalnego, wymaga nieakceptowalnie długiego czasu (np. milionów lat);

(b) skonstruowanie algorytmów wyznaczających optymalne (bądź chociaż bliskie optymalnym) rozwiązania tych problemów w sensownym czasie jest zadaniem nietrywialnym.

ALGORYTMY

Definicja 2 *Algorytm jest to procedura (krok po kroku) działająca w skończonym czasie, rozwiązująca dany problem π.*

Algorytmy będziemy oznaczać literami A, np. A_1, A_2, itd.

Przykład:

Problem: Znajdź wszystkie wartości $x \in \mathbb{C}$ spełniające równanie

$$ax^2 + bx + c = 0,$$

gdzie $a, b, c \in \mathbb{R}$.

Algorytm rozwiązania:

Krok 1. Jeżeli $a = 0$ i $b = 0$, to równanie nie ma rozwiązania lub jest tożsamością. STOP.

Krok 2. Jeżeli $a = 0$ i $b \neq 0$, to równanie ma jedno rozwiązanie $x_1 = -c/b$. STOP.

Krok 3. Wyznacz $\Delta = b^2 - 4ac$.

Krok 4. Jeżeli $\Delta = 0$, to równanie ma jedno rozwiązanie $x_1 = -\frac{b}{2a}$. STOP.

Krok 5. Jeżeli $\Delta \neq 0$, to równanie ma dwa rozwiązania $x_1 = -\frac{b+\sqrt{\Delta}}{2a}$ i $x_2 = -\frac{b-\sqrt{\Delta}}{2a}$, $x_1, x_2 \in \mathbb{C}$.

INSTANCJE PROBLEMU

Definicja 3 *Instancją problemu jest konkretny problem z ustalonymi wartościami danych wejściowych.*

Instancje problemu będziemy oznaczać literą I np. I_1, I_2, itd.

Zbiór wszystkich instancji problemu π będziemy oznaczać przez D_π.

Należy rozróżniać problemy i instancje problemu!!!

Przykład

Problem π jest zdefiniowany następująco:

Znajdź wszystkie wartości $x \in \mathbb{C}$ spełniające równanie

$$ax^2 + bx + c = 0,$$

gdzie $a, b, c \in \mathbb{R}$.

Instancją tego problemu jest przykładowo:

I_1: Znajdź wszystkie wartości $x \in \mathbb{C}$ spełniające równanie

$$x^2 - 5x + 6 = 0.$$

lub I_2 : Znajdź wszystkie wartości $x \in \mathbb{C}$ spełniające równanie

$$3x^2 + 7x + 12 = 0.$$

FAKT: Istnieje algorytm rozwiązywanie równania kwadratowego.

PYTANIA:

1.Czy istnieje algorytm znajdowania pierwiastków równania trzeciego stopnia?

2.Czy istnieje algorytm znajdowania pierwiastków równania czwartego stopnia?

3.Czy istnieje algorytm znajdowania pierwiastków równania stopnia piątego?

4.Czy istnieje algorytm znajdowania pierwiastków równania stopnia $n > 5$?

ODPOWIEDZI NA PYTANIA:

1.Czy istnieje algorytm znajdowania pierwiastków równania trzeciego stopnia? **TAK.** (Cardano 1545r.)

2.Czy istnieje algorytm znajdowania pierwiastków równania czwartego stopnia? **TAK.** (S. del Ferro 1545r.)

3.Czy istnieje algorytm znajdowania pierwiastków równania stopnia piątego? **NIE.** (P. Ruffini 1813r.)

4.Czy istnieje algorytm znajdowania pierwiastków równania stopnia $n > 5$? **NIE.** (N. H. Abel 1827r.)

PODSTAWOWY PODZIAŁ PROBLEMÓW

Problemy dzielą się na dwie grupy:

1. **Problemy rozstrzygalne, rozwiązywalne**: Problemy dla których istnieją algorytmy ich rozwiązania.

2. **Problemy nierozstrzygalne, nierozwiązywalne**: Problemy dla których nie istnieją algorytmy ich rozwiązania. Przy czym problem nierozstrzygalny jest to problem dla którego wykazano, że algorytm jego rozwiązania nie istnieje.

PRZYKŁAD PROBLEMU NIEROZSTRZYGALNEGO

Dziesiąty problem Hilberta: Dane jest równanie diofantyczne dowolnej liczby zmiennych. Czy równanie to jest rozwiązywalne w liczbach całkowitych?

Równanie diofantyczne jest to równanie postaci:

$$p(x_1, ..., x_n) = 0,$$

gdzie $p : \mathbb{Z}^n \to \mathbb{Z}$ jest wielomianem o całkowitych współczynnikach.

Przykład 1: $a^2 + b^2 = c^2$
Odpowiedź: TAK, np. a=3, b=4, c=5

Przykład 2: $2x + 4y = 1$
Odpowiedź: NIE

Przykład 3: $a^n + b^n = c^n$, $a, b, c > 0$, $n \geq 3$
Odpowiedź: ?

J. Matjasiewicz (1970r.) udowodnił, że 10 problem Hilberta jest nierozstrzygalny.

Funkcja złożoności obliczeniowej algorytmu A

$f_A(N(I)) = \max\{t :$ $t -$ ilość operacji (jednostek czasu) potrzebnych do rozwiązania dowolnej instancji I problemu o rozmiarze $N(I)$ przez algorytm $A\}$

$(N(I) = n)$

W praktyce ważny jest tylko kształt funkcji $f_A(N(I))$ (tzn. jej zachownie dla rosnących wartości rozmiaru problemu $N(I)$), a nie konkretny czas (konkretne wartości funkcji f_A).

Notacja $O(\cdot)$ – *funkcja $f(n)$ jest rzędu $O(g(n))$ jeśli*

$$\bigvee_{c,N} \bigwedge_{n \geq N} 0 \leq f(n) \leq c \cdot g(n), \text{ czyli}$$

$$\frac{f(n)}{g(n)} \longrightarrow c \text{ dla } n \to \infty,$$

(tzn. dla $n \to \infty$ funkcje $f(n)$ i $g(n)$ zachowują się podobnie).

Notacja O

Definicja 4 *Funkcja* $f(n) = O(g(n))$ *jeżeli:*

$$\exists M > 0 : n > M \; \exists C \geq 0 : f(n) \leq Cg(n).$$

Inaczej:

$f(n) = O(g(n))$ jeżeli

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \leq C.$$

Zapis $f(n) = O(g(n))$ czytamy: "funkcja $f(n)$ jest $O(g(n))$łub "funkcja $f(n)$ jest rzędu funkcji $g(n)$".

Notacja O pozwala nam opisać w jaki sposób zmienia się dana funkcja dla dużych wartości n.

Przykładowo: $3n^2 - n + 1 = O(n^2)$,

bo

$$\lim_{n\to\infty} \frac{3n^2 - n + 1}{n^2} \leq 3.$$

Czyli funkcja $3n^2 - n + 1$ zmienia się tak jak funkcja n^2 dla dużych wartości n.

Własności notacji O

1. Jeżeli $f(n) = O(a(n))$ i $g(n) = O(b(n))$, to $f(n) + g(n) = O(a(n) + b(n)) = O(\max\{a(n), b(n)\})$.

2. Jeżeli $f(n) = O(a(n))$ i $g(n) = O(b(n))$, to $f(n)g(n) = O(a(n)b(n))$.

3. Jeżeli $p(n)$ jest wielomianem stopnia k to $p(n) = O(n^k)$

Hierarchia przykładowych ciągów (każdy z nich jest O od wszystkich na prawo od niego):

$$1, \log n, ..., \sqrt{n}, n, n \log n, n\sqrt{n}, n^2, n^3, ..., 2^n, n!, n^n$$

Przykłady:

$\mathbf{f(n)}$	$\mathbf{O(g(n))}$
$n^2 - 2n + 5$	$O(n^2)$
$\frac{1}{2}n \cdot \pi \log \frac{n}{2}$	$O(n \log n)$
$\frac{1}{2}n^{\pi \log \frac{n}{2}}$	$O(n^{\log n})$
2^{3n-8}	$O(2^n)$
$n! - 100n^{25} + 0,3n^{0,3}$	$O(n!)$

Czasy działania algorytmów o określonych **funkcjach złożoności obliczeniowej**, przy założeniu, że jedna operacja matematyczna zajmuje $1\mu s$.

$f_A(n)$	n					
	10	20	30	40	50	60
$O(n)$	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
$O(n^2)$	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s	0,0036 s
$O(n^5)$	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
$O(n^{10})$	2,7 h	118,5 dni	18,7 lat	3,3 wieków	30,9 wieków	192 wieki
$O(2^n)$	0,001 s	1,0 s	17,9 min	12,7 dni	35,7 lat	366 wieków
$O(3^n)$	0,59 s	58 min	6,5 roku	3855 wieków	**2 * 10^8 wieków**	1,3 * 10^{13} wieków
$O(n!)$	3,6 s	770 wieków	8,4 *10^{16} wieków	2,5 * 10^{32} wieków	9,6 * 10^{48} wieków	2,6 * 19^{66} wieków

Wpływ wzrostu szybkości komputerów na czasy działania algorytmów o określonych funkcjach złożoności obliczeniowej.

$f_A(n)$	Rozmiar instancji rozwiązywany w określonym czasie przez „wolny komputer”	Rozmiar instancji rozwiązywany w tym samym czasie przez komputer **1000 razy szybszy**
$O(n)$	n_1	$1000 \cdot n_1$
$O(n^2)$	n_2	$31,62 \cdot n_2$
$O(n^5)$	n_3	$3,98 \cdot n_3$
$O(n^{10})$	n_4	$1,99 \cdot n_4$
$O(2^n)$	n_5	n_5+10
$O(3^n)$	n_6	n_6+6
$O(n!)$	n_7	n_7+3 dla $n_7 \leq 10$ n_7+2 dla $10 < n_7 \leq 30$ n_7+1 dla $30 < n7 \leq 1000$

Rodzaje algorytmów ze względu na złożoność obliczeniową:

Algorytmy wielomianowe – algorytmy, których funkcja złożoności obliczeniowej $f(n)$ jest rzędu $O(p(n))$, gdzie $p(n)$ jest pewnym wielomianem zależnym od rozmiaru problemu n, np. $O(n)$, $O(n^2)$, $O(n \log n)$ (algorytmy **efektywne** obliczeniowo).

Algorytmy wykładnicze (ponadwielomianowe) – algorytmy, których funkcji złożoności obliczeniowej $f(n)$ nie da się ograniczyć żadnym wielomianem $p(n)$, np. $O(2^n)$, $O(n^{\log n})$, $O(n!)$ (algorytmy **nieefektywne** obliczeniowo).

Klasy złożoności algorytmów:

1. **Klasa P** – zawiera wszystkie problemy, dla których skonstruowano wielomianowe algorytmy optymalne.
2. **Klasa NP** – zawiera wszystkie problemy, dla których skonstruowano wykładnicze algorytmy optymalne.
   $P \subset NP$, ponieważ jeśli dla pewnego problemu mamy alg. wielomianowy, zawsze możemy skonstruować alg. mniej efektywny (wykładniczy), np. przegląd zupełny.
3. **Klasa problemów NP-trudnych** – podklasa NP problemów wielomianowo ekwiwalentnych, dla których (najprawdopodobniej) nie można skonstruować algorytmów wielomianowych.
   $NP-trudne \subset NP$, ale $P \cap NP-trudne = \emptyset$.
4. **Klasa problemów silnie NP-trudnych** – podklasa problemów NP-trudnych, których nie można rozwiązać optymalnie w czasie pseudo-wielomianowym.

Dokładne określenie przynależności danego problemu do klasy złożoności pozwala skonstruować **najodpowiedniejsze algorytmy** jego rozwiązania.

W tym celu:

(i) albo szukamy dla danego problemu optymalnego algorytmu wielomianowego (klasa P),

(ii) albo udowadniamy jego (silną) NP-trudność,

przy czym nie ma reguły, od którego z punktów należy zacząć analizę.

Problemy, dla których nie skonstruowano algorytmów wielomianowych (i) ani nie udowodniono (silnej) NP-trudności (ii), tworzą **klasę tymczasową** (tzw. problemów otwartych).

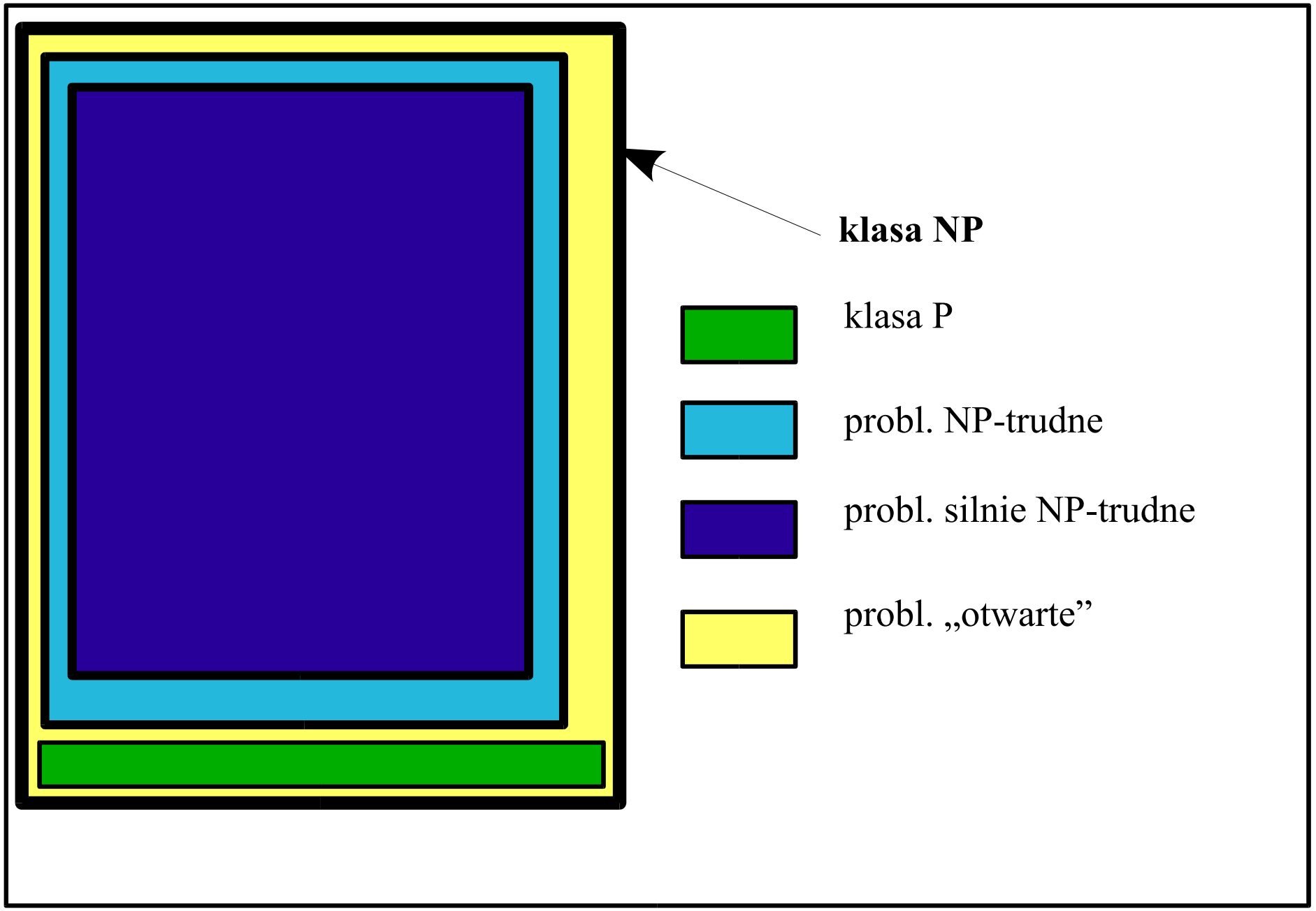
W praktyce:

9% istniejących problemów należy do klasy P,

84% należy do klasy problemów NP-trudnych,

z czego **79%** należy do klasy problemów silnie NP-trudnych,

7% to problemy otwarte.


klasa NP
klasa P
probl. NP-trudne
probl. silnie NP-trudne
probl. „otwarte”

Rodzaje algorytmów (metod) **optymalnych**:

- Wielomianowe algorytmy dokładne (dedykowane) – tylko dla problemów z klasy P.

- **Programowanie dynamiczne** – głównie dla problemów NP-trudnych w zwykłym sensie (tzn. nie silnie NP-trudnych).

- Programowanie całkowitoliczbowe.

- **Metoda podziału i ograniczeń** – głównie dla problemów (silnie) NP-trudnych.

- Przegląd zupełny.

Rodzaje algorytmów (metod) **przybliżonych**:

- Algorytmy konstrukcyjne i zachłanne – głównie dla problemów NP-trudnych.

- Algorytmy typu popraw – głównie dla problemów (silnie) NP-trudnych:
  - lokalnego poszukiwania (np. poszukiwanie zstępujące, poszukiwanie losowe),
  - **metaheurystyczne** (np. poszukiwanie z zabronieniami (*tabu search*), symulowane wyżarzanie, poszukiwanie genetyczne (ewolucyjne), poszukiwanie mrówkowe).

- Wielomianowe i w pełni wielomianowe **schematy aproksymacyjne** – głównie dla problemów NP-trudnych.