# Problem dystrybucji towarów z najwcześniejszymi i najpóźniejszymi terminami dostaw

Dawid Ryznar, Krzysztof Zielonka

5 Grudnia 2012

# Opis problemu (orginalny)

Jako firma transportowa mamy dostarczyć towary do miast. Każde miasto ma ustaloną karę za spóźnienie lub przybycie zawcześnie. Przemieszczenie się między miastami trwa pewną liczbę czasu. Należy znaleść ciąg miast, dla którego ciężarówka odwiedza każde miasto i minimalizujący sumę kar jaką trzeba zapłacić za zbyt wczesne lub zbyt późne przybycie.

# Nasze założenia

- Ciężarówka ma nieskończoną pojemność. Jest w stanie zabrać towary dla wszystkich miast
- Dla każdego miasta mamy funkcje kary od czasu. Ogólniejszy wariant kar
- Czas jest w postaci liczby naturalnej np. liczba sekund
- Znamy pewne górne ogranicznie czasowe, które określa maksymalny czas przejazdu
- Rozładunek i załadunek nie wymaga czasu
- W kązdym miesic możemy czekać dowolną liczbe czasu znim wyładujemy towar
- Znamy wierzchołek startowy (magazyn)

# Opis problemu

- Pełny graf ważony z $n$ wierzchołkami
- Wyróżniony jeden wierzchołek startowy $v_{start}$
- Wprowadzmy funckje kary $p$ jaką trzeba zapłacić za dostarczenie towaru w czasie $t$ do pewnego miasta (bardziej ogólny wariant, miasta mogą nadawać kary bardziej swobodnie oraz mogą nadawać nagorody)
- Każda krawędź ma przyparzadkowany czas potrzebny na jej pokonanie
- Dodatkowo zakładamy, że w każdym mieście możemy przeczekać pewien okres czasu
- Dla uproszczenie zakładamy, że jest pewne górne ograniczenie na czas potrzebny na pokonaniem trasy. Jeżeli rozwiązanie potrzebuje więcej czasu zakładamy, że jest ono nieakceptowalne

# Opis problemu

$$DT = \langle V, w, p, t_{max} \rangle \tag{1}$$

$$t : V \to N \tag{2}$$

$$p : V \times N \to R \tag{3}$$

$w$ – przyporządkowuje krawędziom wagi (czasy podróży)

$p$ – funckja kary, dla danego wierzchołka i czasu przybcia zwraca kare w postaci liczby rzeczywistej

$t_{max}$ – górne ograniczenie na czas potrzebny na pokonaniem dystansu

## Instancja problemu

Miastom przyporządkowujmey kolejno numery $2, \cdots, n$. Dla uproszczenia będziemy zakładać, że magazyn zawsze ma numer 1. Instancją problemu jest para:

$$P = \langle T, p \rangle \tag{4}$$

## Macierz czasu przajazdów

$$T = [t_{ij}]_{n \times n} \tag{5}$$

Kwadratowa maciarz gdzie element $t_{ij}$ to czas potrzebny na przejazd najszybszą drogą z miasta $i$ do $j$.

## Funkcja kary

$$p : N \to N \to R \tag{6}$$

Funkcja kary, przyjmująca kolejno numer miasta, czas rozładunku i zwracająca karę w postaci liczby rzeczywistej.

# Graf miast

W grafie miast, wierzchołki interpretujemy jako miasta, a wagi na krawędziach jako czasy potrzebne na przedostanie sie z jednego miasta do drugiego. Czas na krawędzi $\{u, v\}$ jest najszybszym czasem potrzebnym na przedostanie sie z miasta $u$ do $v$.

- Graf miast spełnia nierówność trójkąta

$$a < b + c \ \wedge \ b < a + c \ \wedge \ c < a + b \tag{7}$$

$$\forall_{a,b,c \in V} t(\{a, c\}) < t(\{a, b\}) + t(\{b, c\}) \tag{8}$$

- Graf miast jest grafem pełnym

Jeżeli graf niespłnia jednego z powyższych założen to możemy go do takiego przekonwertować wywołujac dla każdego wierzchołka *BFS*.

# Nierówność trójkąta

$$a < b + c \ \wedge \ b < a + c \ \wedge \ c < a + b \tag{9}$$

- Zakładamy, że dany czas przejazdu między dowolnymi dwoma miastami to średni czas potrzebny na pokonanie najszybszej trasy łączącej te dwa miasta
- Dzięki temu założeniu graf dla miast spełnia nierówność trójkąta czyli:

$$\forall_{a,b,c \in V} t(\{a, c\}) < t(\{a, b\}) + t(\{b, c\}) \tag{10}$$

- Eleminujemy możliwość rozwiązania gdzie miasta mogą sie powtarzać, jeżeli mamy dostarczyć towar do miasta $b$, a znajdujemy sie w mieścia $a$ to najszybsza droga między tymi miastami zajmuje $t(\{a, b\})$

## Rozwiązania dopuszczalne

Rozwiązaniem dopuszczalnym (sepłniającym warunki zadania) jest permutacja liczb $1, \cdots, n$.

## Uzasadnienie

- W rozwiązniu musza znaleść sie wszystkie miasta. (definicja problemu)
- W rozwiązaniu miasta nie mogą sie powtarzać. (nierówność trójkąta + założenie o nieskończonej ładowności ciężarówki + możliwość czekania w dowolnym mieście)
- W rozwiazniu nie uwzgledniamy magazynu. (to jest punkt startowy i końcowy + założenie o nieskończonej ładowności ciężarówki)

# Funkcja celu

$$F : N^n \rightarrow R \tag{11}$$

$$F(v_1 \cdots v_n) = C(v_1, \cdots V_n, 0) \tag{12}$$

# Funkcja pomocnicza

$$C : N^m \times N \rightarrow R \quad \text{gdzie } m > 0 \tag{13}$$

$$C(v,t) = \begin{cases} p(v,t) & \text{gdy } t < t_{max} \\ +inf & \text{wpp} \end{cases} \tag{14}$$

$$C(v_1, \cdots, v_n, t) = \min_{t \leq t_c \leq t_{max}} \{C(v_2, \cdots, v_n, t_c + t_{1,2}) + p(v_1, t_c)\} \tag{15}$$

# Złożonośc problemu

## Złożnosc problemu

Problem dystrupucji towarów jest NP trudny. Udowodnimy to konstrując wielomianową redukcje problemu komiwojażera do problemu dystrybucji towarów.

## Problem komiwojażera

Mamy dany pełny ważony graf G. Rozwiązaniem problemu jest minimalny cykl Hamiltona na tym grafie.

# Złożonośc problemu

## Dowód

**Data**: G -graf pełny wazony, w - funkcja wagowa
**Result**: trojka $\langle T, p, t_{max} \rangle$
$f$ – funkcja przyporządkowywujaca wierzchołkom grafu kolejne liczby naturalne
$g$ – funkcja przyporządkowwywująca krawędziom kolejne liczby naturalne
$p(v,t) = \lambda(v,t) \rightarrow \sum_{\{u,v\} \in E} (2^{g(\{u,v\})} \& t) \cdot w(\{u,v\})$
**forall the** $v, u \in V$ **do**
$\quad T[f(v), f(u)] = 2^{g(\{v,u\})}$
**end**
$t_{max} = \sum_{i=2}^{|V|} w(f(v_{i-1}), f(v_i))$
**return** $\langle T, p, t_{max} \rangle$

# Algorytm konstrukcyjny

## Opis algorytmu konstrukcyjnego

Algorytm konstrukcyjny oparliśmy o algorytm Local Search. Sąsiądów wybieramy wykorzystując wszystkie możliwe transpozycje.

## Local Search

**Data**: $x$ – initial node
**Result**: *best_neighbour* – the best local node
$current = none$
$best\_neighbor = x$
**repeat**
  $current = best\_neighbour$
  $neighbours =$ find all neighbours of *current*
  $best\_neighbour =$ select best from $neighbours \cup \{current\}$
**until** $F(best\_neighbour) == F(current)$;
**return** *best_neighbour*

# Algorytm konstrukcyjny

## Neighbors

**Data**: x – current node
**Result**: neighbours – set of x's neighbours
$neighbours = \emptyset$
**forall the** $1 \leq i, j \leq n \ \wedge i < j$ **do**
$\quad neighbours \cup = x_1 \cdots x_{i-1}, x_j, x_{i+1} \cdots x_{j-1}, x_i, x_{j+1} \cdots x_n$
**end**
**return** neighbours

# Algorytm konstrukcyjny

## Algorytm konstrykcyjny dla problemu dystrybucji towarów

**Data**: $T, p, t_{max}, x$ – initial node
**Result**: *best_neighbor* – best local neighbour
$current = none$
$best\_neighbor = x$
**repeat**
  $current = best\_neighbor$
  $neighbours = Neighbours(x) \cup \{current\}$
  $best\_neighbour =$ select best from *neighbours*
**until** $F(best_neighbour) == F(current)$;
**return** *best_neighbor*

# Symulowane wyżarzanie [ang. *Simulated Annealing*]

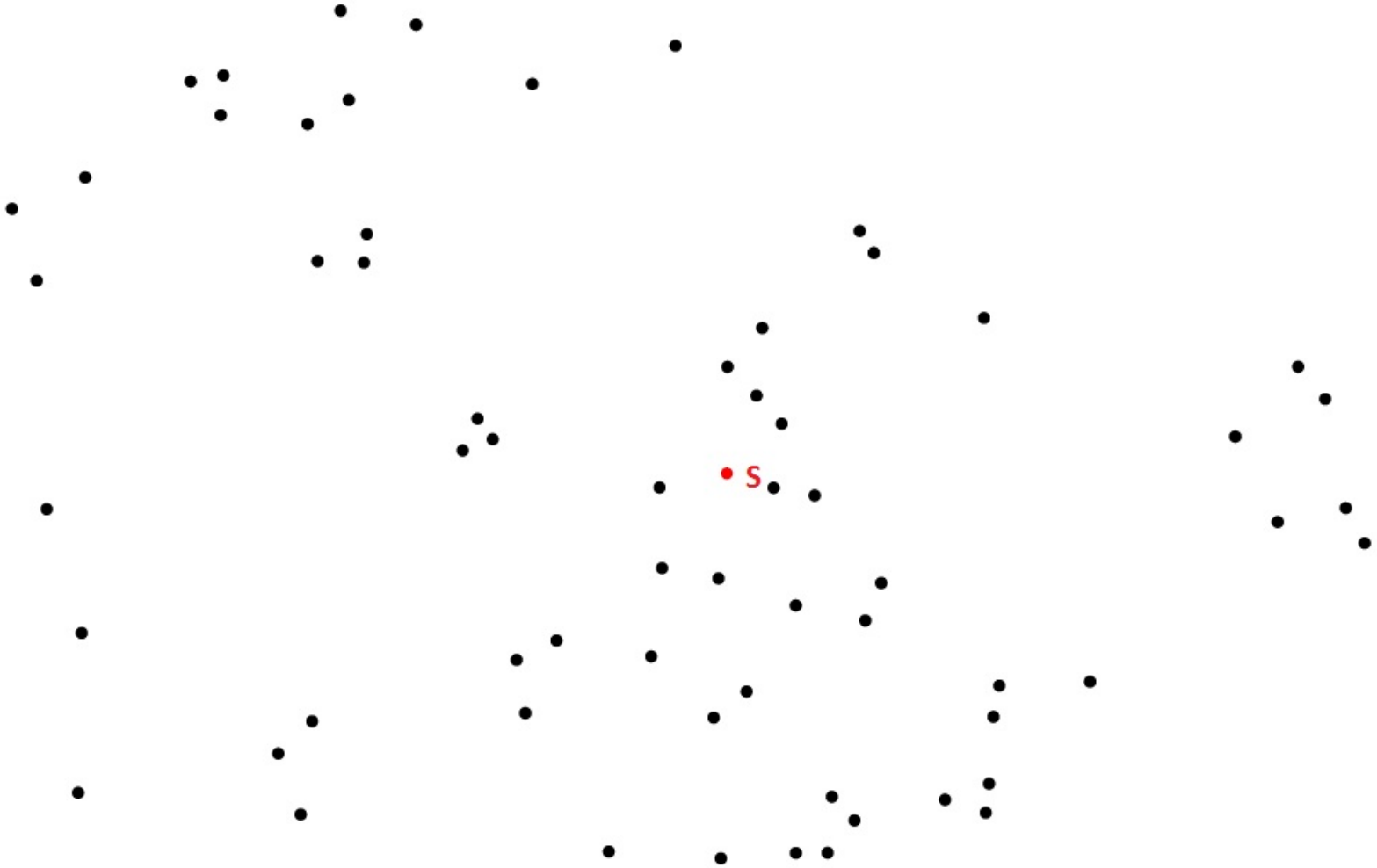
Rodzaj algorytmu heurystycznego przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukiwania rozwiązań najlepszych. Jest wariantem metody przeszukiwania lokalnego [ang. *Local Search*]. Ogólny szkic działania algorytmu symulowanego wyżarzania:

1. losowy wybór punktu startowego
2. losowy wybór sąsiada
3. odpowiednia akceptacja sąsiada
4. po każdej iteracji temperatura zostaje zaktualizowana:

$$T = n \cdot T \wedge n \in (0, 1)$$

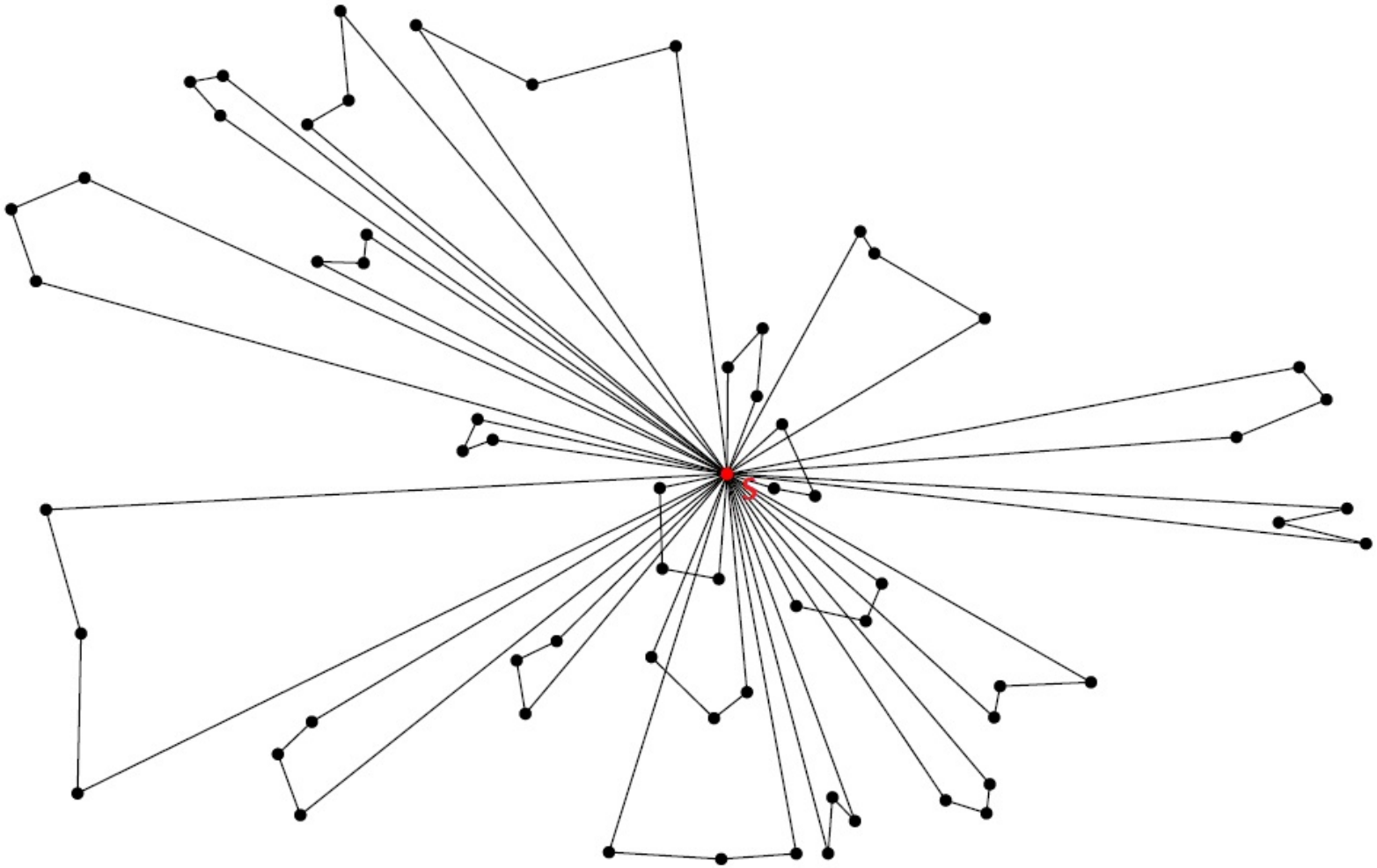
5. algorytm zatrzymuje się gdy w ciągu ustalonej liczby iteracji nie uda się osiągnąć lepszego wyniku

# Symulowane wyżarzanie [ang. *Simulated Annealing*]




Przykładowy DP z magazynem w centralnym punkcie.

# Symulowane wyżarzanie [ang. *Simulated Annealing*]




Przykładowe optymalne rozwiązanie DP z magazynem w centralnym punkcie.

# Symulowane wyżarzanie [ang. *Simulated Annealing*]

```
1  create an initial old_solution as the set of random routes of size 3;
2  best_solution := old_solution;
3  equilibrium_counter := 0;                  # set the equilibrium counter
4  T := cost (best_solution) / 1000;          # initial temperature of annealing
5  repeat
6      for iteration_counter := 1 to n^2 do
7          annealing_step (old_solution, best_solution);
8      end for;
9      T := T · α;                            # temperature reduction
10     equilibrium_counter := equilibrium_counter + 1;
11 until equilibrium_counter > 20;
```

Sekwencyjny algorytm symulowanego wyżarzania dla DP.

# Symulowane wyżarzanie [ang. *Simulated Annealing*]

```
1  procedure annealing_step (old_solution, best_solution);
2      select randomly a customer;
3      select randomly a route (distinct from the customer's route selected above)
           from the set containing the routes of the old_solution and an empty
           route;
4      if the route size is less than 3 then
5          create the new_solution by moving the customer into the chosen route;
6      else
7          select randomly a customer in the route;
8          create the new_solution by exchanging the selected customers between
               their routes;
9      end if;
10     δ := cost (new_solution) − cost (old_solution);
11     generate random x uniformly in the range (0, 1);
12     if (δ < 0) or (x < T/(T + δ)) then
13         old_solution := new_solution;
14         if cost (new_solution) < cost (best_solution) then
15             best_solution := new_solution;
16             equilibrium_counter := 0;
17         end if;
18     end if;
19 end annealing_step;
```

Procedura implementująca jeden krok wyżarzania.

# Wykonanie

Projekt wykonano przy użyciu:

1. Ruby (język programowania)
2. Git (CVS)
3. RubyMine (IDE)

Projekt ma swoje repozytorium w serwisie github.com
`https://github.com/kzielonka/Praktyka-Optymalizacji-Projekt`