Techniki optymalizacji

Łukasz Wojnarowski (80164)
Tomasz Kujawa (75909)

9 listopada 2010

Spis treści

1 Opis problemu

Rozwiązywany problem jest rozwinięciem *problemu komiwojażera* (TSP - ang. traveling salesman problem), który polega na znalezieniu 4 cykli hamiltona w pełnym grafie ważonym o minimalnej sumie wag.

Dane wejściowe składają się z grafu pełnego $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, gdzie $\mathcal{V}$ to zbiór wierzchołków (można go interpretować jako zbiór punktów na płaszczyźnie), a $\mathcal{E}$ to zbiór krawędzi. Dla każdej z krawędzi $\{v_i, v_j\}: v_i, v_j \in \mathcal{V}$ znana jest waga, będąca odległością pomiędzy wierzchołkami v_i, v_j. Rozwiązaniem problemu są cztery cykle proste

$$w_1, w_2, w_3, \ldots, w_{n-1}, w_n \; ; \; x_1, x_2, x_3, \ldots, x_{n-1}, x_n \; ; \; y_1, y_2, y_3, \ldots, y_{n-1}, y_n \text{ oraz } z_1, z_2, z_3, \ldots, z_{n-1}, z_n, \tag{1}$$

które spełniają następujące ograniczenia:

- $w_i \in \mathcal{V}'$,
- $x_j \in \mathcal{V}''$,
- $y_k \in \mathcal{V}'''$,
- $z_l \in \mathcal{V}''''$,
- $\mathcal{V}' \cup \mathcal{V}'' \cup \mathcal{V}''' \cup \mathcal{V}'''' = \mathcal{V}$,
- $\mathcal{V}' \cap \mathcal{V}'' \cap \mathcal{V}''' \cap \mathcal{V}'''' = \varnothing$,
- $|\mathcal{V}'| = |\mathcal{V}''| = |\mathcal{V}'''| = |\mathcal{V}''''| = n$, przy założeniu, że $\mathcal{V} = 4n$.

Niech $|v_i, v_j|$ oznacza wagę (koszt przebycia drogi) krawędzi pomiędzy wierzchołkami v_i, v_j. Dla tak zdefiniowanego modelu funkcja celu została określona w następujący sposób:

$$minC = \sum_{i<n}^{i=1} |w_i, w_{i+1}| + |w_n, w_1| + \sum_{i<n}^{i=1} |x_i, x_{i+1}| + |x_m, x_1| + \sum_{i<n}^{i=1} |y_i, y_{i+1}| + |y_m, y_1| + \sum_{i<n}^{i=1} |z_i, z_{i+1}| + |z_m, z_1| \tag{2}$$

gdzie $|\mathcal{V}| = 4n$.

2 Generowanie rozwiązania początkowego ($\boldsymbol{RP}$)

2.1 Opis metody

Analizowana metoda generowania rozwiązania początkowego to *grupowanie* i następnie *poszukiwanie najbliższego sąsiada*.

2.1.1 Słowny

Metoda rozpoczyna się od losowego wybrania wierzchołka początkowego, na którego podstawie stworzone zostaną grupy. Grupy mają najpierw przydzielane z puli dostępnych wierzchołków elementy początkowe - takie, że środek ciężkości od punktów już wcześniej przydzielonych jest największy. Następnie na podstawie wybranych "liderów" budowane są grupy - tak, że każdy kolejny element dodawany do grupy będzie miał najmniejszą odległość od środka ciężkości grupy. Należy zaznaczyć, że przydział po grupach odbywa się iteracyjnie - tzn. najpierw przydzielamy jeden element do grupy pierwszej, potem jeden element do grupy drugiej i iteracyjnie aż do wyczerpania się elementów nieprzydzielonych do żadnej grupy. Przydział ten jest powtarzany, aż stworzone zostaną 4 grupy o równych licznościach.

Następnie w każdej grupie następuje budowanie ścieżki (cyklu) tak, że przy każdym kroku wybierany jest taki wierzchołek, że jego odległość od środka ciężkości dotychczas wybranych wierzchołków jest najmniejsza. Algorytm zatrzymuje się, jeśli w grupie nie będzie już nieodwiedzonych wierzchołków. Należy pamiętać, by rozwiązanie uzupełnić o krawędź pomiędzy ostatnim a pierwszym wierzchołkiem - tzn. by waga zwracana uwzględniała połączenie pomiędzy ostatnim, a pierwszym elementem cyklu.

2.1.2 Pseudokod

Poniżej zaprezentowano pseudokod algorytmu opisanego w części 2.1.1.

```
Generuj rozwiązanie początkowe(𝒱)
Generuj podział na grupy()

i ← 1
for ∀ i in {1, 2, 3, 4}
    do
        v ← Pobierz losowy z grupy(i)
        Przydziel wierzchołek do ścieżki w grupie(v)

while ∃ grupa z nieprzydzielonymi wierzchołkami
    do
        next ← Najbliższy nieprzydzielony wierzchołek dla grupy(i)
        Przydziel wierzchołek do ściezki w grupie(next)
        i = (i + 1)%5

rozwiązanie ← Policz sumę sciezek()
```

W kodzie wykorzystano metode przygotowania grup, która została zaprezentowana poniżej:

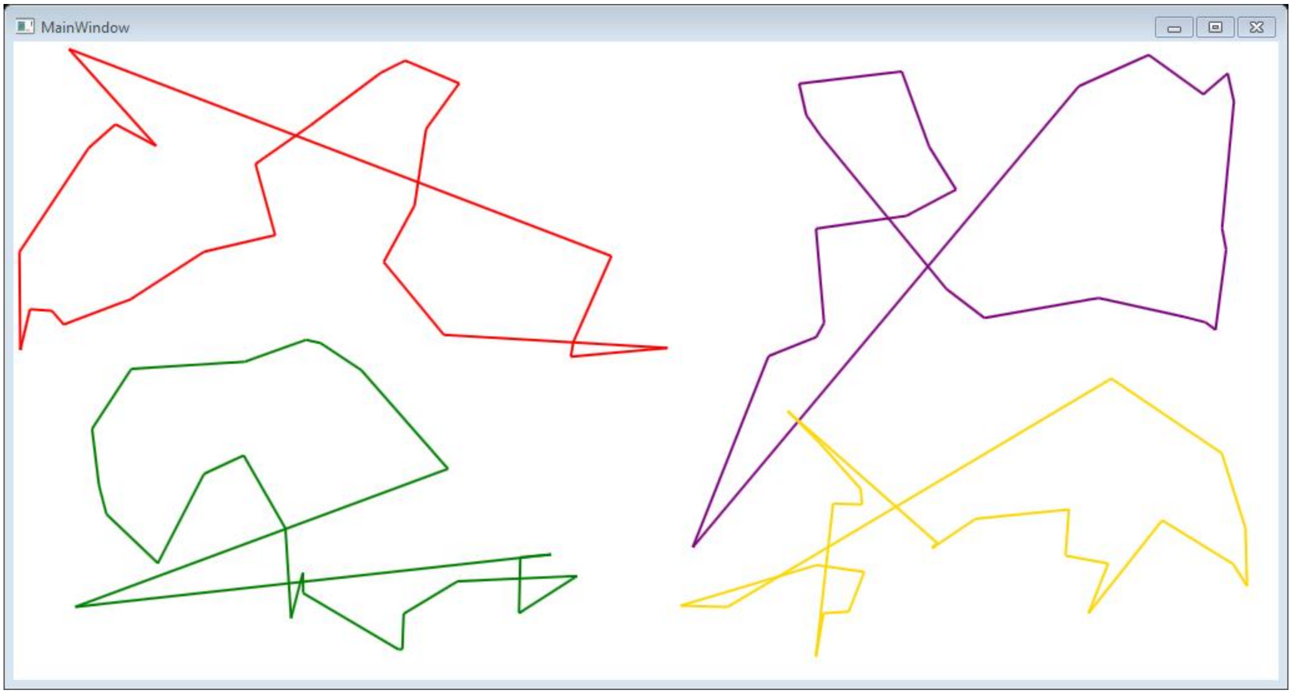
```
Generuj podział na grupy(𝒱)
v1 ← Pobierz losowo(𝒱)
v2 ← Pobierz najdalszy(𝒱 \ {v1})
v3 ← Pobierz najdalszy(𝒱 \ {v1, v2})
v4 ← Pobierz najdalszy(𝒱 \ {v1, v2, v3})
Umieść wierzchołek w grupie(v1,1)
Umieść wierzchołek w grupie(v2,2)
Umieść wierzchołek w grupie(v3,3)
Umieść wierzchołek w grupie(v4,4)
i ← 1
while ∃ 𝒰 ← wierzchołki nieumieszczone w żadnej grupie
    do
        closest_v ← Pobierz najbliższy do i-tej grupy (𝒰)
        Umieść w grupie(closest_v,i)
        i = (i + 1)%5
```

2.2 Wyniki

W tabeli 1 zostały przedstawione uśrednione wyniki dla opracowywanej metody.

instancja	metoda	śr. wart. roz. z 10 pomiarów	mediana	odch. std.	najlepsza wartość
kroA100.txt	NS G	36305,1	35181	5171,589	30000
kroB100.txt	NS G	36434,7	37189	3499,752	30973

Tabela 1: Uśrednione wyniki pomiarów.



Rysunek 1: Rozwiązanie początkowe dla *kroA100.txt*

3 Local search (LS)

3.1 Opis metody

Analizowana metoda generowania rozwiązania to *rozrywanie (1 ruch) w wersji stromej*.

3.2 Opis słowny metody

Proces poszukiwania lokalnego optimum rozpoczyna się od wykonania kroków z opisanego w rozdziale *Generowanie rozwiązania początkowego*. Następnie na takim rozwiązaniu dokonywane jest lokalne przeszukiwanie.

Kroki metody:

1. Wybierz wierzchołek i $k - 1$ mu najbliższych wierzchołków.
2. Rozerwij łuki wokół tych wierzchołków.
3. Rozważ wszystkie możliwe sposoby naprawy do rozwiazania tego problemu.
4. Wykonaj ruch, który przynosi najwięcej zysku.

Parametr $k \in 2, 3, 4$ jest definiowany na wejściu programu.

3.3 Pseudokod

Algorytm generowania rozwiązania można zapisać przy pomocy poniższego pseudokodu.

```
Lokalne przeszukiwanie(V, k)
rozwiązanie ← Generuj rozwiązanie początkowe(V)
while (true)
      do
         zysk ← 0
         wybrani ← Wybierz łuki(k, V)
         możliwe_przydziały ← Generuj możliwe przydziały(wybrani)
         wartość ← Oblicz wartość rozwiązania(rozwiązanie)
         for ∀ ruch in możliwe_przydziały
             do
                aktualne_rozwiązanie ← Wykonaj ruch(rozwiązanie, ruch)
                aktualna_wartość ← Oblicz wartość rozwiązania(aktualne_rozwiązanie)
                aktualny_zysk ← wartość − aktualna_wartość)
                if aktualny_zysk ⩾ zysk
                  then
                       zysk ← aktualny_zysk
                       Zapamiętaj ruch(ruch)
                  else
                       return

         if zysk > 0
           then
                Wykonaj zapamiętany ruch(rozwiązanie)
           else
                return
```

3.4 Wyniki

W tabeli przedstawione zostały zbiorcze wyniki pomiarów:

- *RP* - metoda z pierwszego ćwiczenia - generowanie rozwiązania początkowego,
- *RP + LS* - metoda lokalnego przeszukiwania rozpoczynająca się od wygenerowania rozwiązania początkowego zgodnie z zasadami z ćwiczenia numer 1.

instancja	metoda	śr. jakość (odch. standardowe)	śr. czas [ms]	jakość najlepszego przeszukiwania
kroA100.txt	*RP*	29878 (1644)	0	25893
kroA100.txt	*RP + LS*	23096 (1121)	123	21359
kroB100.txt	*RP*	30661 (1648)	0	25421
kroB100.txt	*RP + LS*	24604 (665)	125	22869

Tabela 2: Uœrednione wyniki pomiarów.

3.5 Rysunki najlepszych rozwiązań

Na poniższych rysunkach przedstawione zostały rozwiązania wygenerowane przy pomocy metody *RP + LS*.