Techniki optymalizacji

Łukasz Wojnarowski (80164)
Tomasz Kujawa (75909)

11 stycznia 2011

Spis treści

1 Opis problemu **3**

2 Generowanie rozwiązania początkowego (*RP*) **3**
2.1 Opis metody . . . 3
2.1.1 Słowny . . . 3
2.1.2 Pseudokod . . . 5

3 Losowe rozwiązanie początkowe - LRP **5**
3.1 Wyniki . . . 5

4 Local search (LS) **6**
4.1 Opis metody . . . 6
4.1.1 Opis słowny metody . . . 6
4.1.2 Pseudokod . . . 7
4.2 Wyniki . . . 7
4.3 Rysunki najlepszych rozwiązań . . . 8

5 Heurystyczny Algorytm Ewolucyjny (HEA) **8**
5.1 Opis metody . . . 8
5.1.1 Opis słowny metody . . . 9
5.1.2 Pseudokod . . . 11
5.2 Eksperymenty i wyniki . . . 11

6 Porównanie LS i HEA **13**

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 (*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)
 1  Generuj podział na grupy()
 2
 3  i ← 1
 4  for ∀ i in {1,2,3,4}
 5        do
 6           v ← Pobierz losowy z grupy(i)
 7           Przydziel wierzchołek do ścieżki w grupie(v)
 8
 9  while ∃ grupa z nieprzydzielonymi wierzchołkami
10        do
11           next ← Najbliższy nieprzydzielony wierzchołek dla grupy(i)
12           Przydziel wierzchołek do ściezki w grupie(next)
13           i = (i + 1)%5
14
```

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

```
Generuj podział na grupy(V)
 1  v1 ← Pobierz losowo(V)
 2  v2 ← Pobierz najdalszy(V \ {v1})
 3  v3 ← Pobierz najdalszy(V \ {v1, v2})
 4  v4 ← Pobierz najdalszy(V \ {v1, v2, v3})
 5  Umieść wierzchołek w grupie(v1,1)
 6  Umieść wierzchołek w grupie(v2,2)
 7  Umieść wierzchołek w grupie(v3,3)
 8  Umieść wierzchołek w grupie(v4,4)
 9  i ← 1
10  while ∃ U ← wierzchołki nieumieszczone w żadnej grupie
11        do
12           closest_v ← Pobierz najbliższy do i-tej grupy (U)
13           Umieść w grupie(closest_v,i)
14           i = (i + 1)%5
15
```

3 Losowe rozwiązanie początkowe - LRP

Opisane w poprzednim rozdziale RP porównywane było z generowanym losowo podziałem na grupy. Pozwoliło to już na etapie kreacji grup zauważyć, jak ważne jest poszukanie odpowiedniej metody przygotowania rozwiązania początkowego.

Ponieważ rozwiązanie to jest proste i intuicyjne pominięto opis - zarówno w pseudokodzie, jak i słowny. Należy tylko odnotować, że w programie jest porównywanie rozwiązania *RP* z losowym rozwiązaniem początkowym.

3.1 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	33 721	33 174	3 551	26 742
	LRP	171 479	170 055	9 635	153 474
kroB100.txt	NS G	35 210	25 979	3 393	30 173
	LRP	166 257	166 053	6 220	153 757

Tabela 1: Uśrednione wyniki pomiarów.

Rysunek 1: Rozwiązanie początkowe dla *kroA100.txt*

4 Local search (LS)

4.1 Opis metody

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

4.1.1 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.

4.1.2 Pseudokod

Algorytm generowania rozwiązania można zapisać przy pomocy poniższego pseudokodu.

```
LOKALNE PRZESZUKIWANIE(V, k)
 1  rozwiązanie ← GENERUJ ROZWIĄZANIE POCZĄTKOWE(V)
 2  while (TRUE)
 3        do
 4           zysk ← 0
 5           wybrani ← WYBIERZ ŁUKI(k, V)
 6           możliwe_przydziały ← GENERUJ MOŻLIWE PRZYDZIAŁY(wybrani)
 7           wartość ← OBLICZ WARTOŚĆ ROZWIĄZANIA(rozwiązanie)
 8           for ∀ ruch in możliwe_przydziały
 9                 do
10                    aktualne_rozwiązanie ← WYKONAJ RUCH(rozwiązanie, ruch)
11                    aktualna_wartość ← OBLICZ WARTOŚĆ ROZWIĄZANIA(aktualne_rozwiązanie)
12                    aktualny_zysk ← wartość − aktualna_wartość)
13                    if aktualny_zysk ⩾ zysk
14                      then
15                            zysk ← aktualny_zysk
16                            ZAPAMIĘTAJ RUCH(ruch)
17                      else
18                            return
19
20           if zysk > 0
21             then
22                    WYKONAJ ZAPAMIĘTANY RUCH(rozwiązanie)
23             else
24                    return
25
```

4.2 Wyniki

W tabeli przedstawione zostały zbiorcze wyniki pomiarów:

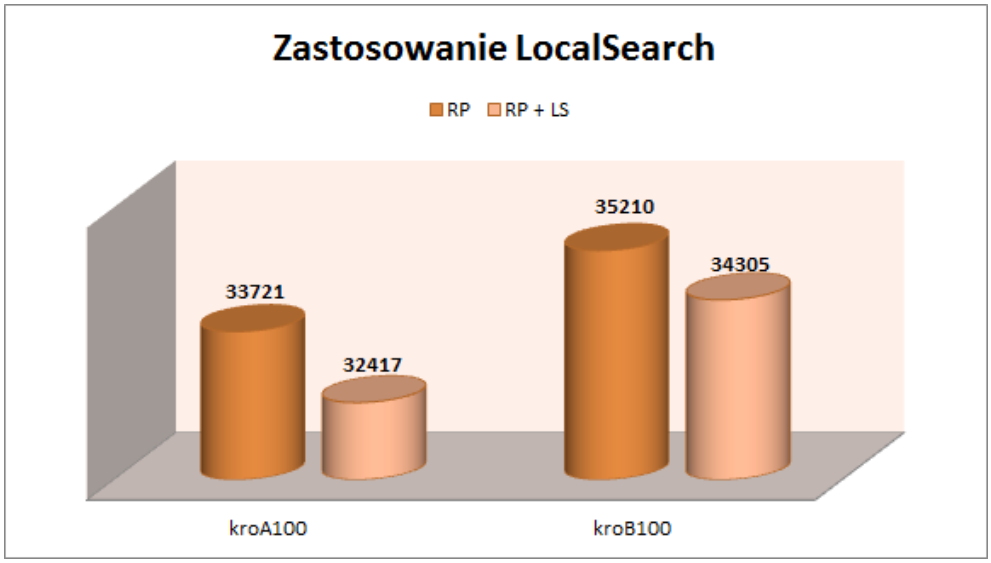
- *RP* - metoda z pierwszego ćwiczenia - generowanie rozwiązania początkowego,
- *RP + LS* - metoda lokalnego przeszukiwania rozpoczynająca się od wygenerowania rozwiązania początkowego zgodnie z zasadami z ćwiczenia numer 1,
- *LRP + LS* - metoda lokalnego przeszukiwania rozpoczynająca się od wygenerowania losowego rozwiązania początkowego zgodnie z zasadami z ćwiczenia numer 2.

Rysunki 2 oraz 3 prezentują kolejno wyniki zastosowania LocalSearch do optymalizacji rozwiązania generowanego przez RP (rys.2, a także LRP (rys.3). Analizując dane łatwo zauważyć, że przy generowaniu rozwiązania metodą RP zastosowanie LS przynosi jedynie niewielką poprawę rozwiązania - na poziomie $4 - 5\%$. Wynika z tego, że przyjęty sposób generowania rozwiązania początkowego (zobacz rozdział 2.1) generuje bardzo dobre rozwiązanie początkowe - w sensie lokalnej jego optymalizacji.

Lepszą poprawe zaobserwować można na wykresie 3, gdzie zastosowanie LS poprawiało wynik generowany przy pomocy LRP o około 13%.

instancja	metoda	śr. jakość (odch. standardowe)	śr. czas [ms]	jakość najlepszego przeszukiwania
kroA100.txt	*RP*	33 721 (3 551)	0	26 742
	RP + LS	32 417 (3 508)	423	25 604
	LRP	171 479 (9 635)	0	153 474
	LRP + LS	152 039 (9 376)	620	127 767
kroB100.txt	*RP*	35 210 (3 393)	0	35 979
	RP + LS	34 305 (3180)	363	29 428
	LRP	166 257 (6 220)	0	153 757
	LRP + LS	145 413 (8 695)	733	148 168

Tabela 2: Uśrednione wyniki pomiarów.



Rysunek 2: Wyniki eksperymentów - zastosowanie LocalSearch do poprawy jakości rozwiązania generowanego przez RP

4.3 Rysunki najlepszych rozwiązań

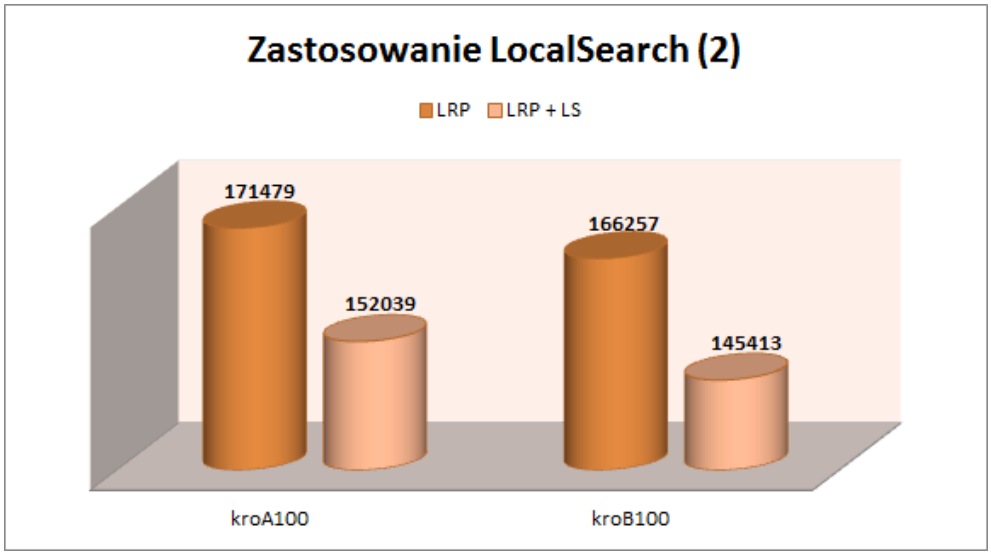
Na poniższych rysunkach przedstawione zostały rozwiązania wygenerowane przy pomocy metody *RP + LS*.

5 Heurystyczny Algorytm Ewolucyjny (HEA)

5.1 Opis metody

W ramach wykonania zadania przygotowano implementację Heurystycznego Algorytmu Ewolucyjnego dla problemu komiwojażera.

W kolejnych podrozdziałach przygotowano opis słowny, a także pseudokod rozwiązania w ramach opracowywanego zadania.



Rysunek 3: Wyniki eksperymentów - zastosowanie LocalSearch do poprawy jakości rozwiązania generowanego przez LRP

Rysunek 4: Rozwiązanie RP+LS dla *kroA100.txt*

5.1.1 Opis słowny metody

Algorytm rozpoczyna się od wygenerowania pewnej populacji początkowej - przeprowadzane są kroki opisane we wcześniejszych roździałach. Oprócz wykorzystania LS przy budowie wstępnej populacji, wykorzystuje się go również do optymalizacji rozwiązań otrzymywanych w populacji rozwiązań po przeprowadzaniu mutacji.

Rysunek 5: Rozwiązanie RP+LS dla *kroB100.txt*

Kroki algorytmu powtarzane są do momentu, w którym przekroczony zostanie dostępny na obliczenia czas - który to jest parametrem wejściowym do algorytmu. W testach (zgodnie z poleceniem prowadzącego) przyjęto wartość 30 sekund (bądź 60, dla bardziej rozbudowanego zestawu danych testowych - *kroA150*).

Pierwszy krok, to wybór rozwiązań z populacji rozwiązań, które poddawane będą ewolucji. Dokonuje się to przy pomocy losowego wyborów rozwiązań dostępnych w populacji. Należy podkreślić, że liczba wybranych rozwiązań jest parametrem definiowalnym dla algorytmu. Następnie odbywa się drugi krok algorytmu - właściwy dla heurystycznego algorytmu ewolucyjnego.

Drugi krok to dokonanie rekombinacji na parach (losowo dobieranych) z rozwiązań wybranych w kroku poprzednim. Analizując parę rozwiązań wybierane są wszystkie ścieżki wspólne w dwóch rozwiązaniach oraz punkty, które nie leżą na żadnej wspólnej ścieżce.

Następny krok to mutacja, którą logicznie można podzielić na dwa etapy:

- *premutacja* - czyli opracowanie wstępnych rozwiązań na podstawie dostarczonych list wspólnych ścieżek (znalezionych w poprzednim punkcie - przez porównanie pary losowych rozwiązań) dla par rozwiązań,
- *właściwa mutacja* - czyli losowe wypełnienie brakujących połączeń punktami, które nie znajdują się na żadnej ze wspólnych scieżek.

Każdy z tych etapów może być konfigurowalny parametrami, które definiują liczbę generowanych rozwiązań na poszczególnych etapach. Pozwala to w elastyczny sposób zarządzać przestrzenią rozwiązań, tak by nie rozrastała się ona w zbyt szybkim tempie.

Następnie każde takie rozwiązanie jest poddawane lokalnej optymalizacji przy pomocy *Local Search*, które przedstawione zostało w rozdziale 4.

Ostatni etap, to ograniczenie przestrzeni rozwiązań, tak by nie rozrastała się ona w zbyt intensywnym tempie - chodzi o ograniczenia pamięciowe poszczególnych maszyn testowych. Także i w tym wypadku definiuje się parametr wejściowy, który określa maksymalny rozmiar populacji rozwiązań algorytmu ewolucyjnego.

5.1.2 Pseudokod

Wcześniej zdefiniowane wartości:

```
1  max_rozmiar_populacji
2  liczba_rozwiązań_do_ewolucji
3  liczba_premutacji
4  liczba_mutacji
```

Ogólna idea heurystycznego algorytmu ewolucyjnego zaimplementowanego w ramach wykonywania zadania:

```
Heurystyczny Algorytm Ewolucyjny(V, max_czas)
 1  populacja_rozwiązań ← Generuj populację początkową(V)
 2  while (true)
 3        do
 4           do_ewolucji ← Wybór rozwiązań do ewolucji(populacja_rozwiązań)
 5           wspólne_ściezki ← Rekombinacja losowych par(do_ewolucji)
 6           dodatkowa_populacja ← Mutacja(wspólne_ścieżki, pozostałe_punkty)
 7           dodatkowa_populacja ← Lokalne przeszukiwanie na populacji(dodatkowa_populacja)
 8           populacja_rozwiązań ← populacja_rozwiązań ∪ dodatkowa_populacja
 9           Ograniczenie przestrzeni rozwiązań(populacja_rozwiązań)
10           if czas_trwania_algorytmu > czas
11              then
12                   Przerwij algorytm()
13
14  rezultat ← Wybór najlepszego rozwiązania(populacja_rozwiązań)
```

Funkcje użyte w pseudokodzie są na tyle intuicyjne, że ich opis ograniczony zostanie do akceptowalnego minimum.

Wybór rozwiązań do ewolucji Polega na wylosowaniu określonej parametrem liczby rozwiązań z globalnej populacji rozwiązań. Odbywa się to w sposób niedeterministyczny, wykorzystując losowe wybieranie pary rozwiązań (tak, by w parzenie nie znalazły się dwa takie same rozwiązania).

Rekombinacja losowych par Dla pary rozwiązań następuje analiza obydwu, by wydobyć wspólne ścieżki, na podstawie których budowane będzie nowe rozwiązanie. Analizowane są wszystkie punkty, w taki sposób by odnaleźć części wspólne.

Mutacja Metoda ta podzielona jest na dwa logiczne punkty:

- *premutacja* - na podstawie listy wspólnych ścieżek przygotowywane jest rozwiązanie wstępne - tzn. wspólne ścieżki przydzielane są w "puste miejsca" w rozwiązaniu docelowym,
- *właściwa mutacja* - następnie w puste miejsca przydzielane są punkty, które nie znalazły się na ścieżkach stanowiących części wspólne rozwiązań.

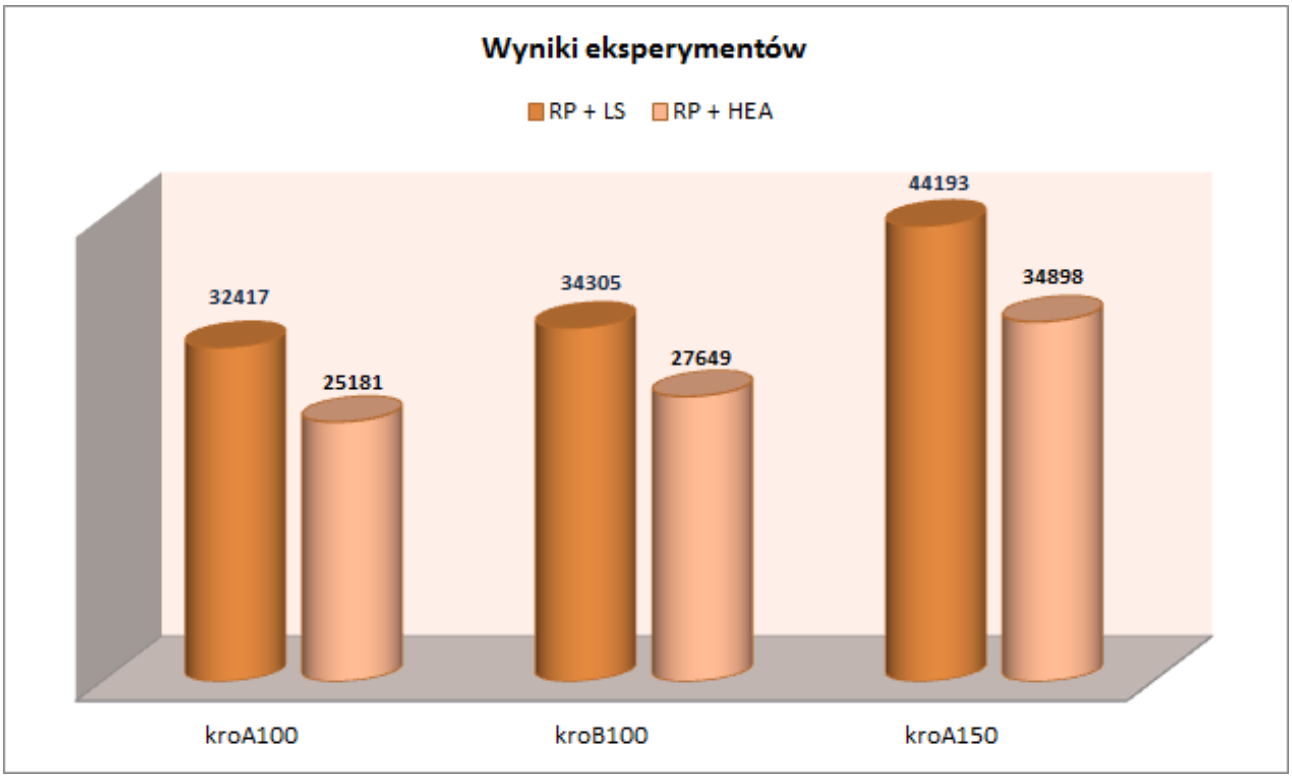
5.2 Eksperymenty i wyniki

W tabeli przedstawione zostały zbiorcze wyniki pomiarów:

- *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 + LS*	32 417 (3 508)	423	25 604
	RP + HEA	25 181 (95)	30 000	25 061
kroB100.txt	*RP + LS*	34 305 (3180)	363	29 428
	RP + HEA	27 649 (530)	30 000	26 446
kroA150.txt	*RP + LS*	44 193 (3 765)	503	36 629
	RP + HEA	34 898 (266)	60 000	34 420

Tabela 3: Uśrednione wyniki pomiarów - porównanie efektywności algorytmu heurystycznego w stosunku do zastosowania lokalnego przeszukiwania.



Rysunek 6: Wyniki eksperymentów

- *RP + HEA* - metoda heurystycznego algorytmu ewolucyjnego z wykorzystaniem algorytmu lokalnego przeszukiwania.

Należy zauważyć, że zbiór *kroA150.txt* zawierał 150 punktów - ponieważ w założeniach do zadania przyjęto, że zbiór powinien móc podzielić się na 4 równe podzbiory (podgrupy) - przyjęto, że do zadania skierowane zostanie pierwsze 148 punktów ze zbioru.

Na rysunku 6 przedstawiony został wykres z pomiarów. Łatwo zauważyć, że algorytm HEA wykazuje około 20% "zysk" w stosunku do zastosowania jedynie algorytmu lokalnego przeszukiwania - dla każdego z zestawów są to wartości bliskie takiemu współczynnikowi.

6 Porównanie LS i HEA

Na polecenie Prowadzącego przygotowany został test porównania opracowanych algorytmów. Głównym założeniem działania eksperymentu było uruchomienie obu algorytmów dla tych samych warunków stopu - tj. ograniczyć czas działania algorytmów do 30 sekund.

Opis oznaczeń:

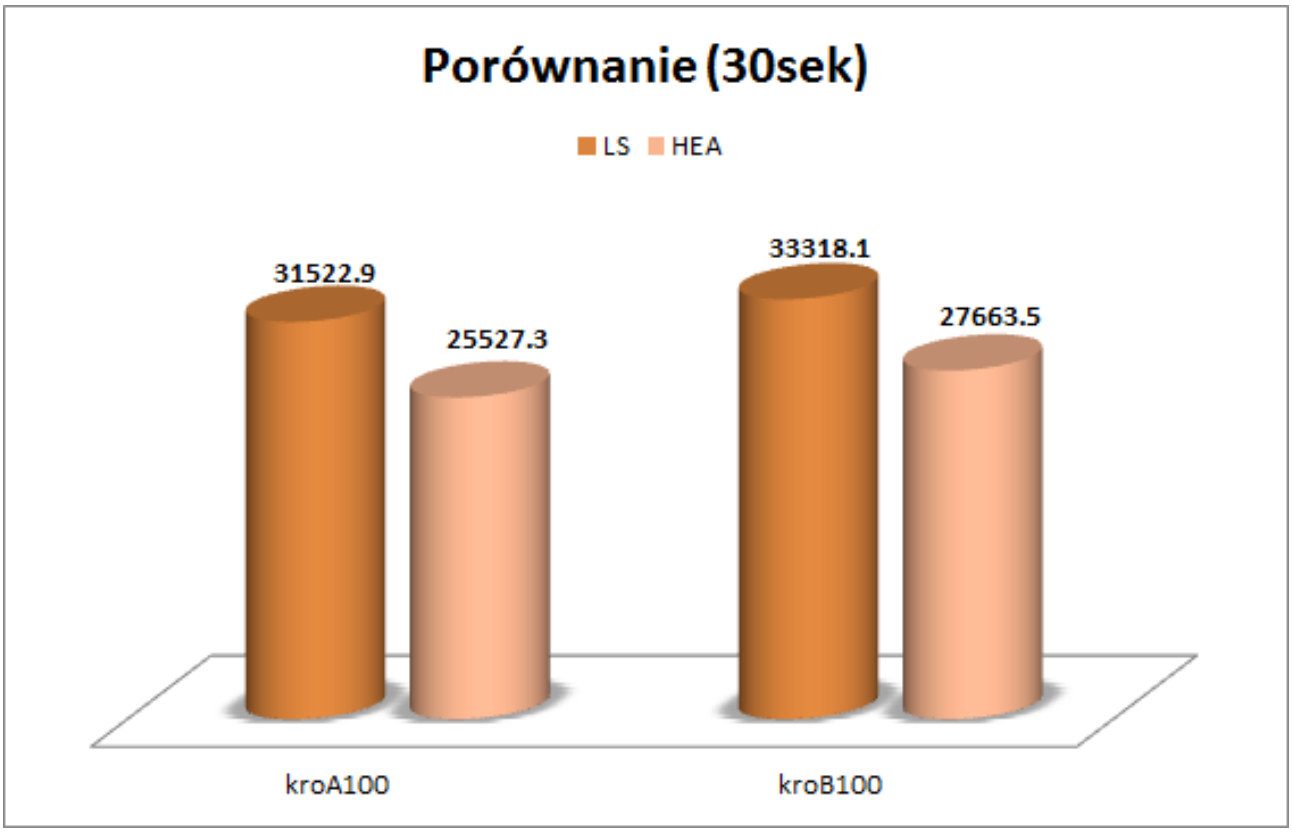
- *RP + LS* - metoda lokalnego przeszukiwania rozpoczynająca się od wygenerowania rozwiązania początkowego zgodnie z zasadami z ćwiczenia numer 1,
- *RP + HEA* - metoda heurystycznego algorytmu ewolucyjnego z wykorzystaniem algorytmu lokalnego przeszukiwania.

instancja	metoda	śr. jakość (odch. standardowe)	śr. czas [ms]	jakość najlepszego przeszukiwania
kroA100.txt	*RP + LS*	31 522 (2 829)	30 000	26 968
	RP + HEA	25 527 (441)	30 000	25 313
kroB100.txt	*RP + LS*	33 318 (2 945)	30 000	27 228
	RP + HEA	27 663 (433)	30 000	27 645

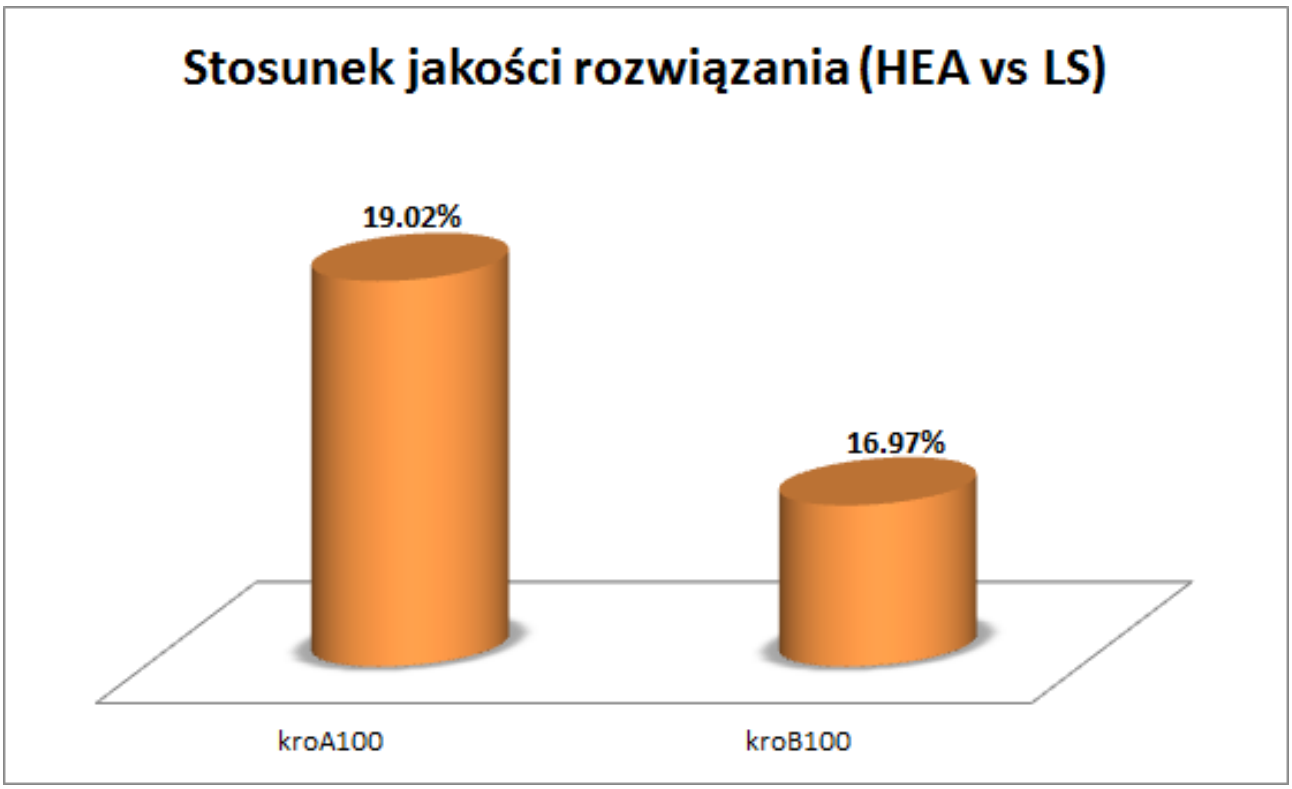
Tabela 4: Uśrednione wyniki pomiarów - porównanie efektywności algorytmu heurystycznego w stosunku do zastosowania lokalnego przeszukiwania.

Na rysunkach 7 oraz 8 przedstawione zostały wykresy wyników porównania algorytmów zaimplementowanych w ramach ćwiczenia. Jednoznacznie widać poprawę rozwiązania przy zastosowaniu heurystycznego algorytmu ewolucyjnego w stosunku do Local Search. Przy uruchomieniu obu algorytmów na taki sam okres czasu przewaga algorytmu HEA nad LS mieści się w przedziale od 10% do 20%.

Przewiduje się, że możliwość poprawy jakości rozwiązania tworzonego przez HEA możnaby uzyskać stosując nieco bardziej wyrafinowany algorytm budowy rozwiązania wstępnego ze ścieżek wspólnych dla pary rozwiązań - np. poprzez wprowadzenie miary odległości - tak by przydział do grup następował w lokalnych obszarach, a nie dowolnych, jak to miało miejsce do tej pory.



Rysunek 7: Wyniki eksperymentów porównujących HEA z LS



Rysunek 8: Jakość HEA w stosunku do LS