Sprawozdanie Laboratorium 3

Wykorzystanie ocen ruchów z poprzednich iteracji i ruchów

kandydackich w lokalnym przeszukiwaniu

Patryk Wojtalak 148236 Maciej Wieczorek 148141

Opis problemu
Celem zadania jest poprawa efektywności czasowej lokalnego przeszukiwania w wersji stromej
(steepest) z sąsiedztwem, które okazało się lepsze w poprzednim zadaniu.
Stosujemy dwa mechanizmy poprawy efektywności:

1. Wykorzystanie ocen ruchów z poprzednich iteracji z uporządkowaną listą ruchów. Na liście należy

umieszczać ruchy zarówno między-, jak i wewnątrztrasowe. W przypadku ruchów wewnętrztrasowych wymiany dwóch krawędzi, należy dokładnie zapoznać się opisem z wykładów dotyczącym problemu komiwojażera.

2. Ruchy kandydackie.

Opis zaimplementowanych algorytmów w pseudokodzie:

Wykorzystanie ocen ruchów z poprzednich iteracji:

Wygeneruj rozwiązanie losowe
Inicjacja LM:
    Dla każdej ścieżki:
        Dla każdej krawędzi wewnątrz ścieżki:
            Dla każdej innej krawędzi wewnątrz ścieżki:
                Sprawdź czy zamiana pary krawędzi poprawia rozwiązanie
                Jeśli tak
                Dodaj ruch oraz ruch dla krawędzi odwrotnych do LM
                Jeśli nie kontynuuj
    Dla każdego wierzchołka ścieżki pierwszej:
        Dla każdego wierzchołka ścieżki drugiej
            Sprawdź czy zamiana wierzchołków poprawia rozwiązanie
            Jeśli tak dodaj ruch do LM
            Jeśli nie kontynuuj

Dopóki w LM znajdują się aplikowalne ruchy powtarzaj:
    Posortuj LM po zmianie długości ścieżki malejąco
    Wybierz pierwszy aplikowalny ruch, zaaplikuj go i usuń z LM, jeżeli jakiś obejrzany zawiera nieistniejącą już krawędź usuń go (jeżeli obydwie krawędzie istnieją ale jedna jest

odwrócona pomiń ruch bez usuwania go)

Sprawdź wszystkie nowe ruchy zamiany krawędzi / wierzchołków dla wszystkich nowo powstałych krawędzi oraz krawędzi odwrotnych dodaj je do LM waz z ruchami dla krawędzi odwrotnych.

Ruchy kandydackie:

Wygeneruj rozwiązanie losowe

sasiedzi = Znajdź dla każdego wierzchołka k najbliższych mu wierzchołków
Dopóki prawda:
    Najlepszy ruch międzytrasowy = candidateInter(instancja, rozwiazanie, sasiedzi)
    Najlepszy ruch wewnątrztrasowy = candidateIntra(instancja, rozwiazanie, sasiedzi)
    Jeżeli nie znaleziono żadnego ruchu poprawiającego rozwiązanie przerwij.
    Wykonaj najlepszy ruch.

```
candidateInter(instancja, rozwiazanie, sasiedzi):
    Dla każdej scieżki
        Dla każdego v1 w ścieżce:
            Dla wszystkich k najbliższych sąsiadów wierzchołka v1:
                Jeśli v2 jest na innej ścieżce niż v1:
                    Jeśli ruch zamieniający v1 i v2 jest najlepszy:
                        Najlepszy ruch międzytrasowy = (v1, v2)
```

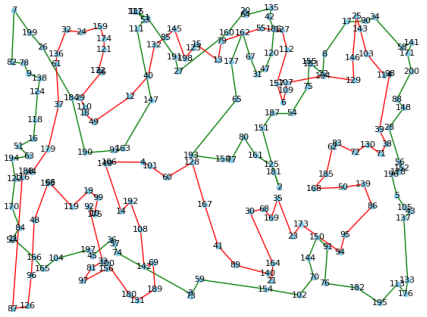
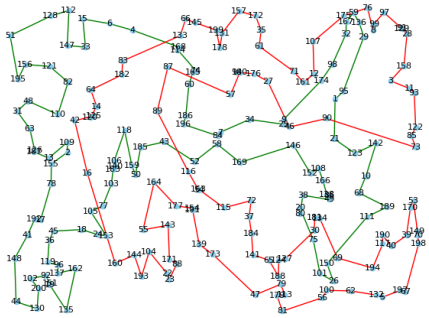
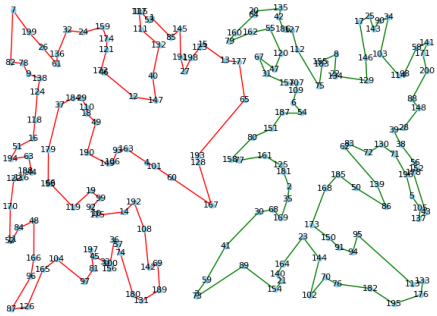
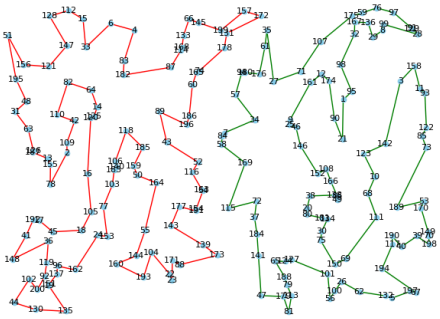
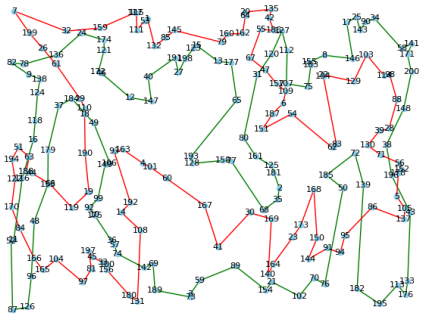
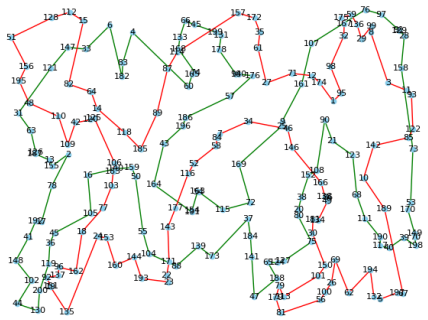
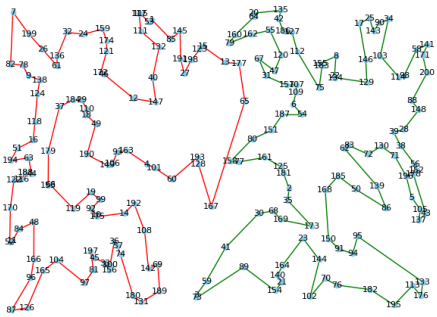
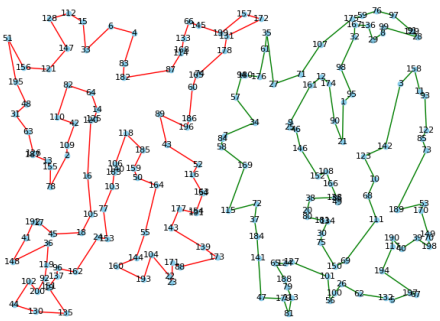
```
candidateIntra(instancja, rozwiazanie, sasiedzi):
    Dla każdej scieżki
      Dla każdego v1 w ścieżce:
        Dla wszystkich k najbliższych sąsiadów wierzchołka v1:
          Jeśli v2 jest na tej samej ścieżce co v1:
            Jeśli ruch zamieniający krawędzie (v1, v1After) i (v2, v2After) jest najlepszy:
              Najlepszy ruch wewnątrztrasowy = (v1, v2)
```

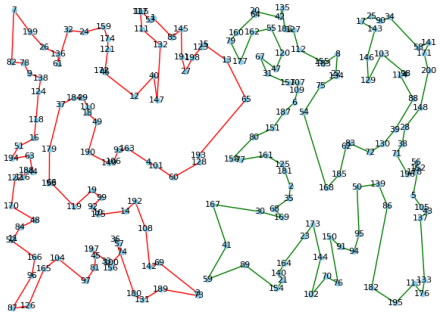
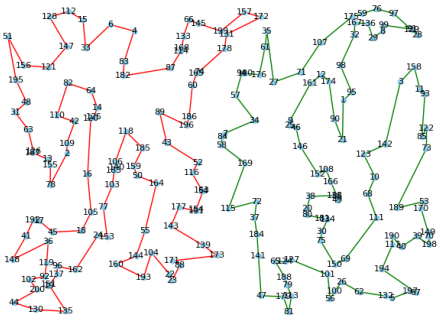
Wartość funkcji celu - średnia (min-max):

Instancja Algorym	kroA200 random	kroB200 random	kroA200 greedy start	kroB200 greedy start
Steep	39307 (36496-43960)	39437 (36960-42691)	34357 (31064-37012)	34522 (32235-36861)
Steep z wykorzystaniem ocen z poprzedniej iteracji	40220 (37325-42704)	39931 (36960-42303)	34518 (31064-37480)	34564 (32235-36870)
Steep ruchy kandydackie (k=10)	40885 (38205-44015)	40502 (37542-44699)	34566 (31454-37216)	34671 (32235-36863)
Greedy Regret	36065 (32104-41568)	35815 (32473-38414)	-	-

Czas wykonania - średni (min-max) [ms]:

Instancja Algorym	kroA200 random	kroB200 random	kroA200 greedy	kroB200 greedy
Steep	16.9406 (13.4193-45.1554)	17.3043 (13.8527-40.1975)	1.72015 (0.5439-6.0086)	1.17776 (0.3846-3.0208)
Steep z wykorzystaniem ocen z poprzedniej iteracji	3.2615 (2.018-4.5674)	3.1523 (2.105-4.3122)	0.8331 (0.3124-3.0152)	0.52815 (0.1648-1.6781)
Steep ruchy kandydackie (k=10)	2.7512 (1.8921-3.8910)	2.6941 (1.8921-3.8910)	0.91711 (0.8176-1.7890)	0.76711 (0.4512-1.2156)

	kroA200	kroB200
SteepestWithLM-RandomSolver		
SteepestWithLM-GreedyRegret		
CandidateLocalSearch(10)-RandomSolver		
CandidateLocalSearch(10)-GreedyRegret		

GreedyRegret		

Wnioski: Przetestowaliśmy 3 warianty algorytmu przeszukiwania w wersji steepest, wersję z wykorzystaniem ocen z poprzednich iteracji oraz z wykorzystaniem ruchów kandydackich a także algorytm konstrukcyjne greedy regret. Wykorzystanie ocen z poprzednich iteracji oraz ruchów kandydackich pozwoliło znacznie przyspieszyć przeszukiwanie zmniejszając złożoność z $O(n^3)$ na $O(n^2)$ przy jednoczesnym nieznacznym pogorszeniu średniej wartości funkcji celu. W przapdku rozpoczęcia przeszukiwania z rozwiązania losowego pogorszenie wartości funkcji celu jest nieco bardziej zauważalne niż w przypadku rozpoczęcia z rozwiązania wygenerowanego przez greedy regret. Warto zauważyć, że najlepsze uzyskane wartości nie różnią się prawie wcale. Wszystkie algorytmy lokalnego przeszukiwania startujące z rozwiązania losowego uzyskują gorszą wartość funkcji celu niż algorytm konstrukcyjny greedy regret. W przypadku rozpoczęcia przeszukiwania z rozwiązania wygenerowanego przez algorytm greedy regret wszystkie algorytmy przeszukiwania w bardzo krtókim czasie nieznacznie poprawiają wygenerowaną funkcję celu uzyskując niemalże identyczne wyniki.

Kod programu:
https://github.com/maciej-wieczorek/imo-lab