Sprawozdanie z laboratorium nr 4

Inteligentne Metody Optymalizacji

Autorzy: Jakub Gołąb, Mariusz Hybiak

Wprowadzenie

Celem zadania było rozszerzenie implementacji lokalnego przeszukiwania stosując trzy metody:

- (MSLS) Multiple start local search
- (ILS1) Iterated local search - Iteracyjne przeszukiwanie lokalne z niewielką perturbacją
- (ILS2) Iterated local search - Iteracyjne przeszukiwanie lokalne z Large-scale neighborhood search, tj. większą perturbacją typu Destroy-Repair

Algorytmy

MSLS

```
function MSLS(distance_matrix, data, n_iter):
    best_cycle1 = NULL
    best_cycle2 = NULL
    best_total_length = INFINITY

    for n_iter:
      cycle1, cycle2 = random_cycle(data)
      cycle1, cycle2 = candidate_moves_algorithm(cycle1, cycle2,
distance_matrix, data)
      length = calculate_cycles_length(cycle1, cycle2, distance_matrix)

      if length < best_length:
        best_cycle_1 = cycle1
        best_cycle_2 = cycle2
        best_length = length

    return best_cycle1, best_cycle2
```

ILS1

```
function ILS1(distance_matrix, data, time_MSLS):
    best_cycle_1, best_cycle_2 = random_cycle(data)
    best_cycle_1, best_cycle_2 = candidate_moves_algorithm(best_cycle_1,
best_cycle_2, distance_matrix, data)
    best_length = calculate_cycles_length(best_cycle_1, best_cycle_2,
distance_matrix)
```

```
    while time_elapsed <= time_MSLS:
        cycle1, cycle2 = little_perturbation(best_cycle_1, best_cycle_2)
        cycle1, cycle2 = candidate_moves_algorithm(cycle1, cycle2,
distance_matrix, data)
        length = calculate_cycles_length(cycle1, cycle2, distance_matrix)
        if length < best_length:
            best_length = length
            best_cycle_1 = cycle1
            best_cycle_2 = cycle2

    return best_cycle_1, best_cycle_2

function little_perturbation(cycle1, cycle2):
    foreach cycle in set_of[cycle1, cycle2]:
        cycle = swap_random_nodes(cycle1, cycle2, 0.2)

    return cycle1, cycle2
```

ILS2

```
function ILS2(distance_matrix, data, time_MSLS):
    best_cycle_1, best_cycle_2 = random_cycle(data)
    best_cycle_1, best_cycle_2 = candidate_moves_algorithm(best_cycle_1,
best_cycle_2, distance_matrix, data)
    best_length = calculate_cycles_length(best_cycle_1, best_cycle_2,
distance_matrix)

    while time_elapsed <= time_MSLS:
        cycle1, cycle2 = severe_perturbation(best_cycle_1, best_cycle_2)
        cycle1, cycle2 = candidate_moves_algorithm(cycle1, cycle2,
distance_matrix, data)
        length = calculate_cycles_length(cycle1, cycle2, distance_matrix)
        if length < best_length:
            best_length = length
            best_cycle_1 = cycle1
            best_cycle_2 = cycle2

    return best_cycle_1, best_cycle_2

function severe_perturbation(cycle1, cycle2, distance_matrix):
    original_size_cycle1 = length_of(cycle1)
    original_size_cycle2 = length_of(cycle2)

    random1 = get_random_nodes_of(cycle1)
    random2 = get_random_nodes_of(cycle2)

    cycle1 = swap_random_nodes_of(cycle1, random1)
    cycle2 = swap_random_nodes_of(cycle2, random2)

    # Repair
    cities = random1 + random2
```

```
    while length_of(cycle1) < original_size_cycle1:
        current_city = cycle1.last_element()
        nearest_city = find_closest_city(current_city, distance_matrix)
        cycle1.append(nearest_city)
        cities.remove(nearest_city)

    while length_of(cycle2) < original_size_cycle2:
        current_city = cycle2.last_element()
        nearest_city = find_closest_city(current_city, distance_matrix)
        cycle2.append(nearest_city)
        cities.remove(nearest_city)

    return cycle1, cycle2
```

Wyniki eksperymentu obliczeniowego

W tabeli przedstawiono sumy długości cykli dla każdej z metod dla obu instancji problemu.

		Długość cyklu		
		min	**mean**	**max**
Instancja	**Algorytm**			
kroA200.tsp	**MSLS**	43440.074	44181.734	44552.210
	ISL1	47933.072	48803.432	50472.015
	ISL2	31617.228	34258.148	39236.820
kroB200.tsp	**MSLS**	43922.327	43964.201	44285.143
	ISL1	45593.615	46553.961	47873.625
	ISL2	32118.157	36719.520	39958.189

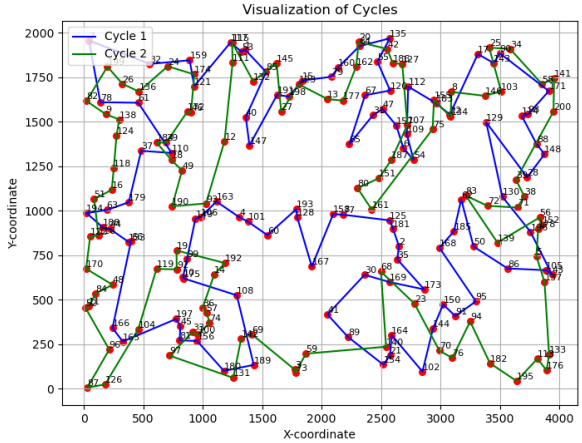
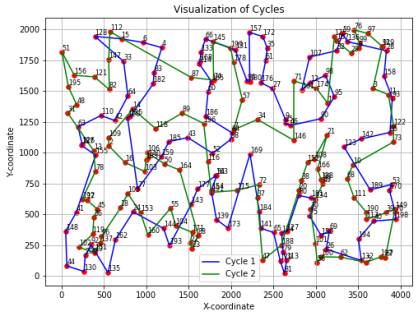
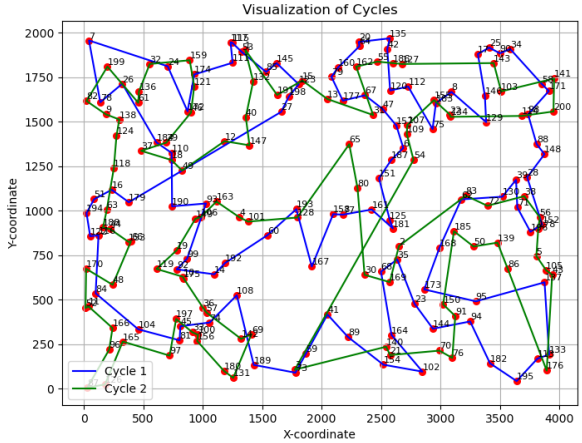
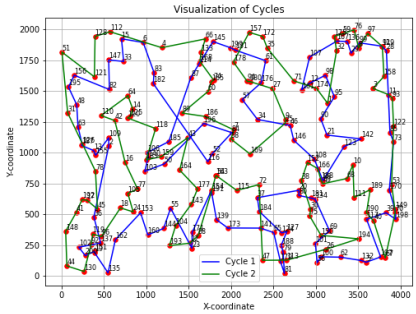
Czas działania algorytmu

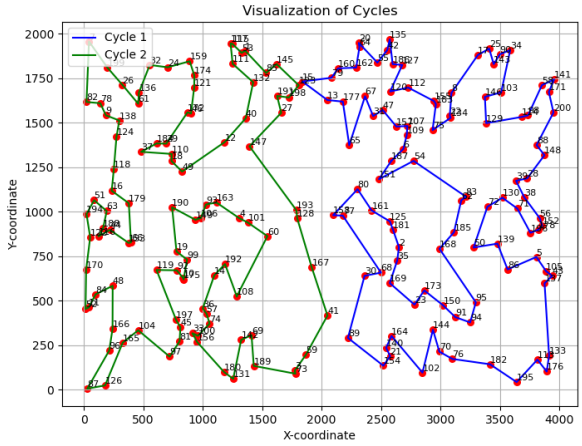
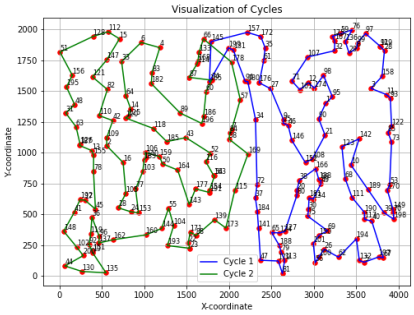
W tabeli przedstawiono średni czas działania algorytmu.

		Czas wykonania		
		min	**mean**	**max**
Instancja	**Algorytm**			
kroA200.tsp	**MSLS**	593.978	595.418	596.176
	ISL1	600.318	601.727	602.610
	ISL2	600.034	602.170	603.832
kroB200.tsp	**MSLS**	403.026	407.896	414.910

		Czas wykonania		
		min	**mean**	**max**
Instancja	**Algorytm**			
	ISL1	407.059	412.071	419.148
	ISL2	408.103	412.903	420.416

Wizualizacje najlepszych rozwiązań

Metoda	**KroA200**	**KroB200**
MSLS		
ISL1 (greedy cycle)		

Metoda	KroA200	KroB200
ISL2		

Wnioski

- Najwaniejszą obserwacją jest fakt, e algorytm ISL2, którego zadaniem było znaczące zepsucie biezącego rozwiązania (w naszym przypadku 30% całego cyklu) oraz próba jego naprawy przy uzyciu heurystyki daje najlepsze rezultaty. Widać to po sumarycznej długości najlepszych cykli, oraz to, ze na płaszczyźnie dwuwymiarowej obydwa cykle są wyraźnie rozdzielone i praktycznie w ogóle na siebie nie nachodzą.

Kod programu

Kod programu znajduje się pod tym linkiem.