IMO – LAB 2 Lokalne przeszukiwanie

Jan Bróździak 141142

Opis zadania

Celem zadania było zaimplementowanie lokalnego przeszukiwania w wersjach stromej (steepest) i zachłannej (greedy) z dwoma różnymi rodzajami sąsiedztwa (wymiana wierzchołków w cyklu, oraz wymiana krawędzi w cyklu), startując z rozwiązań losowych oraz rozwiązań uzyskanych za pomocą jednej z heurystyk opracowanych w ramach poprzedniego zadania. Dodając do tego ruchy międzytrasowe (wymiana wierzchołków pomiędzy cyklami) w sumie zostało zaimplementowanych 8 kombinacji lokalnego przeszukiwania. Jako punkt odniesienia zaimplementowano również algorytm losowego błądzenia.

Opis zaimplementowanych algorytmów w pseudokodzie

Pseudokod dla funkcji generujących ruchy

Generowanie wszystkich możliwych wymian wierzchołków pomiędzy cyklami:

```
function generate_all_vertex_swaps_between_cycles(cycle1, cycle2,
distance_matrix):
    possible_swaps = []

    for i in range(len(cycle1) - 1):
        for j in range(len(cycle2) - 1):
            new_cycle1, new_cycle2 = swap_between_cycles(cycle1,
cycle2, i, j)
            score = calculate_score(new_cycle1, new_cycle2,
distance_matrix)
            possible_swaps.append((new_cycle1, new_cycle2, score))

    return possible_swaps
```

Generowanie wszystkich wymian wierzchołków wewnątrz cyklu

```
function generate_all_vertex_swaps_within_cycles(cycle1, cycle2,
distance_matrix):
    possible_swaps = []

    // Wymiana wierzchołków w pierwszym cyklu
    for i in range(len(cycle1) - 1):
        for j in range(i + 1, len(cycle1) - 1):
            new_cycle1 = swap_within_cycle(cycle1, i, j)
            score = calculate_score(new_cycle1, cycle2,
distance_matrix)
            possible_swaps.append((new_cycle1, cycle2, score))

    // Wymiana wierzchołków w drugim cyklu
    for i in range(len(cycle2) - 1):
        for j in range(i + 1, len(cycle2) - 1):
            new_cycle2 = swap_within_cycle(cycle2, i, j)
            score = calculate_score(cycle1, new_cycle2,
distance_matrix)
            possible_swaps.append((cycle1, new_cycle2, score))

    return possible_swaps
```

Generowanie wszystkich wymian krawędzi wewnątrz cyklu

```
function generate_all_edge_swaps_within_cycles(cycle1, cycle2,
distance_matrix):
    possible_swaps = []

    // Wymiana krawędzi w pierwszym cyklu
    for i in range(len(cycle1) - 1):
        for j in range(i + 1, len(cycle1) - 1):
            new_cycle1 = swap_edges_in_cycle(cycle1, i, j)
            score = calculate_score(new_cycle1, cycle2,
distance_matrix)
            possible_swaps.append((new_cycle1, cycle2, score))

    // Wymiana krawędzi w drugim cyklu
    for i in range(len(cycle2) - 1):
        for j in range(i + 1, len(cycle2) - 1):
            new_cycle2 = swap_edges_in_cycle(cycle2, i, j)
            score = calculate_score(cycle1, new_cycle2,
distance_matrix)
            possible_swaps.append((cycle1, new_cycle2, score))

    return possible_swaps
```

Generowanie listy ruchów dla wariantu z wymianą wierzchołków pomiędzy cyklami i wymianą wierzchołków wewnątrz cyklu

```
function generate_combined_moves_list_1(cycle1, cycle2,
distance_matrix):
    possible_moves = []
    possible_moves.extend(generate_all_vertex_swaps_between_cycles(cy
cle1, cycle2, distance_matrix))
    possible_moves.extend(generate_all_vertex_swaps_within_cycles(cyc
le1, cycle2, distance_matrix))
    return possible_moves
```

Generowanie ruchów dla listy ruchów z wymianą wierzchołków pomiędzy cyklami i wymianą krawędzi wewnątrz cyklu

```
function generate_combined_moves_list_2(cycle1, cycle2,
distance_matrix):

    possible_moves = []
    possible_moves.extend(generate_all_vertex_swaps_between_cycles(cy
cle1, cycle2, distance_matrix))
    possible_moves.extend(generate_all_edge_swaps_within_cycles(cycle
1, cycle2, distance_matrix))
    return possible_moves
```

Pseudokod dla funkcji lokalnego przeszukiwania

Lokalne przeszukiwanie w wersji zachłannej (greedy)

```
function greedy_local_search(cycle1, cycle2, distance_matrix,
move_list_generator):
    start_time = current_time()
    best_cycle1, best_cycle2 = cycle1, cycle2
    best_score = calculate_score(best_cycle1, best_cycle2,
distance_matrix)

    while True:
        possible_moves = move_list_generator(best_cycle1,
best_cycle2, distance_matrix)
        if random_value() < 0.5:
            reverse(possible_moves)

        start_index = random_int(0, len(possible_moves) - 1)
        improved = False

        for i in range(len(possible_moves)):
            index = (start_index + i) % len(possible_moves)
            new_cycle1, new_cycle2, score = possible_moves[index]
            if score < best_score:
                best_cycle1, best_cycle2 = new_cycle1, new_cycle2
                best_score = score
                improved = True
                break

        if not improved:
            break

    time_taken = current_time() - start_time
    return best_cycle1, best_cycle2, best_score, time_taken
```

Uwaga do kodu: wybrany został sposób randomizacji polegający na wybraniu losowego indeksu początkowego podczas przeglądania listy ruchów, oraz losowy kierunek przeglądania listy.

Lokalne przeszukiwanie w wersji stromej (steepest)

```
function steepest_ascent_local_search(cycle1, cycle2,
distance_matrix, move_list_generator):
    start_time = current_time()
    best_cycle1, best_cycle2 = cycle1, cycle2
    best_score = calculate_score(best_cycle1, best_cycle2,
distance_matrix)

    while True:
        possible_moves = move_list_generator(best_cycle1,
best_cycle2, distance_matrix)
        improved = False

        for new_cycle1, new_cycle2, score in possible_moves:
            if score < best_score:
                best_cycle1, best_cycle2 = new_cycle1, new_cycle2
                best_score = score
                improved = True

        if not improved:
            break

    time_taken = current_time() - start_time
    return best_cycle1, best_cycle2, best_score, time_taken
```

Algorytm losowego błądzenia

```
function randomized_optimization(cycle1, cycle2, distance_matrix,
time_limit):

    start_time = current_time()
    end_time = start_time + time_limit

    best_cycle1, best_cycle2 = cycle1, cycle2
    best_score = calculate_score(best_cycle1, best_cycle2,
distance_matrix)

    while current_time() < end_time:
        possible_moves = generate_all_moves(best_cycle1, best_cycle2,
distance_matrix)
        random_move = random_choice(possible_moves)
        new_cycle1, new_cycle2, score = random_move

        if score < best_score:
            best_cycle1, best_cycle2 = new_cycle1, new_cycle2
            best_score = score

    time_taken = current_time() - start_time
    return best_cycle1, best_cycle2, best_score, time_taken
```

Przebieg eksperymentu

Dla każdej instancji została wygenerowana para rozwiązań startowych – jedno losowe, oraz jedno najlepsze po 100 przebiegach heurystyki 2 regret. Dla każdej kombinacji instancja-rozwiązanie startowe-sąsiedztwo-metoda + losowe błądzenie eksperyment został uruchomiony 100 razy

Wyniki eksperymentu obliczeniowego

Instancja	Start	Sąsiedztwo	Metoda	Min	Max	Średnia
kroA100	Random	Vertices	Greedy	33060	54400	42931,26
kroA100	Random	Vertices	Steep	48429	48429	48429
kroA100	Random	Edges	Greedy	26053	31782	28399,49
kroA100	Random	Edges	Steep	28470	28470	28470
kroA100	Random	-	Random	108541	133853	118674
kroA100	2-regret	Vertices	Greedy	22733	22733	22733
kroA100	2-regret	Vertices	Steep	22733	22733	22733
kroA100	2-regret	Edges	Greedy	22733	22733	22733
kroB100	2-regret	Edges	Steep	22733	22733	22733
kroB100	Random	Vertices	Greedy	36286	52117	43517,77
kroB100	Random	Vertices	Steep	48272	48272	48272
kroB100	Random	Edges	Greedy	26521	32157	29083,84
kroB100	Random	Edges	Steep	30656	30656	30656
kroB100	Random	-	Random	106735	135017	120776,3
kroB100	2-regret	Vertices	Greedy	23770	23770	23770
kroB100	2-regret	Vertices	Steep	23770	23770	23770
kroB100	2-regret	Edges	Greedy	23571	23609	23603,3
kroB100	2-regret	Edges	Steep	23609	23609	23609
kroB100	2-regret	-	Random	23921	24172	24154,93

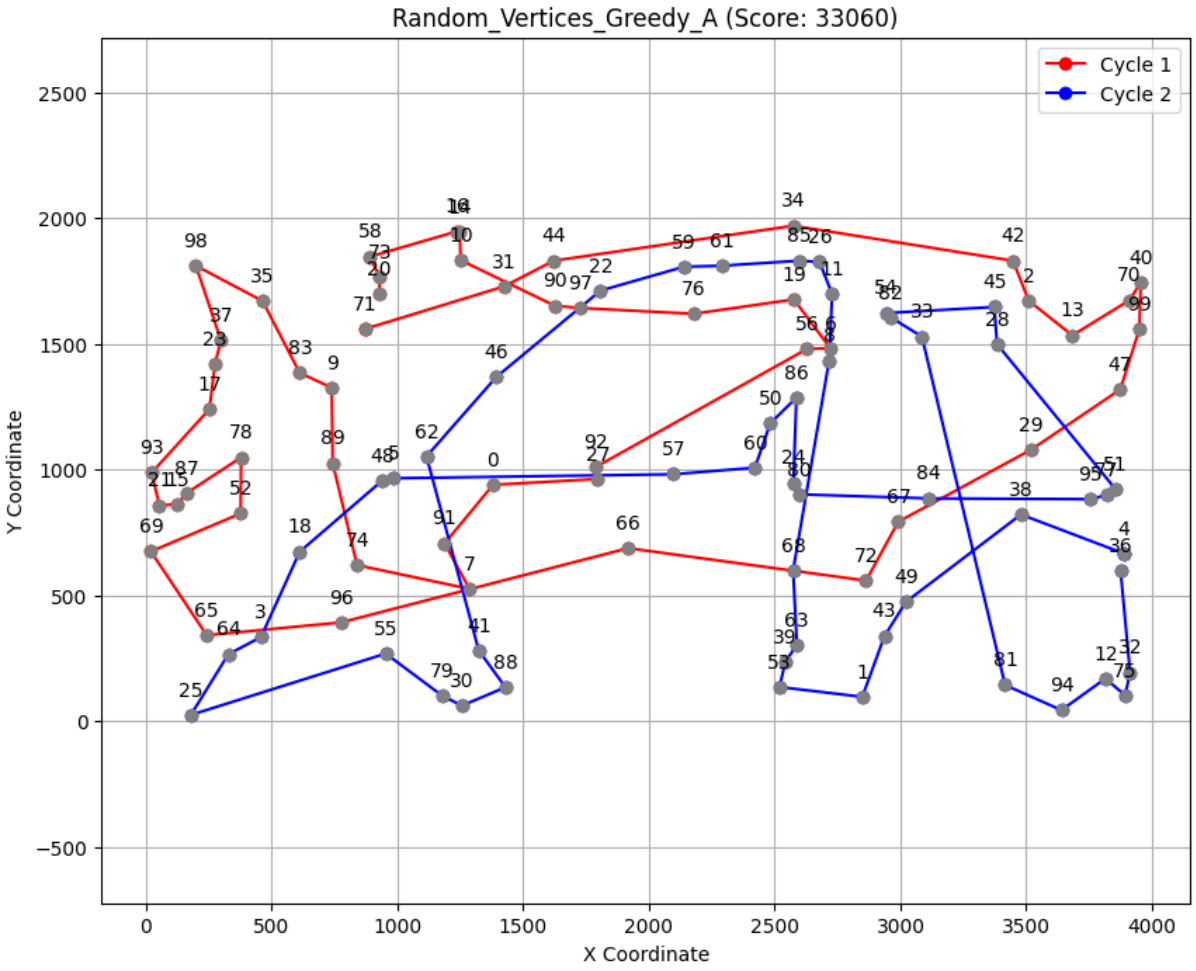
Ze względu na to, że projekt był napisany w języku Python powstawały anomalie czasowe – np. wersja greedy działała dłużej niż wersja steep. Działo się tak z powodu dodatkowego narzutu czasowego na zrandomizowanie (a i tak bardzo proste) przeszukiwania listy ruchów. W związku z tym wyniki czasowe zostały pominięte, gdyż mogłyby prowadzić do błędnych wniosków.

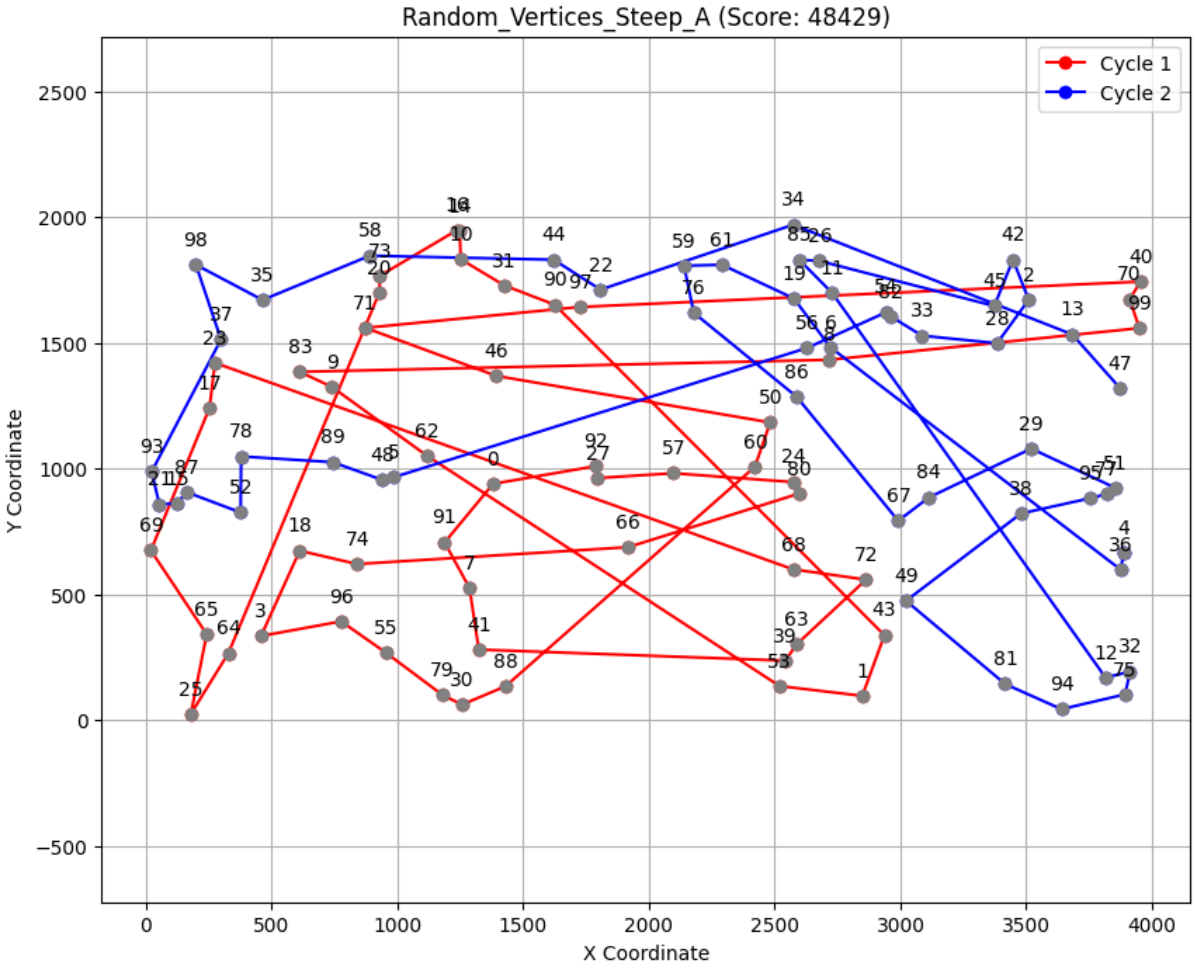
Wnioski

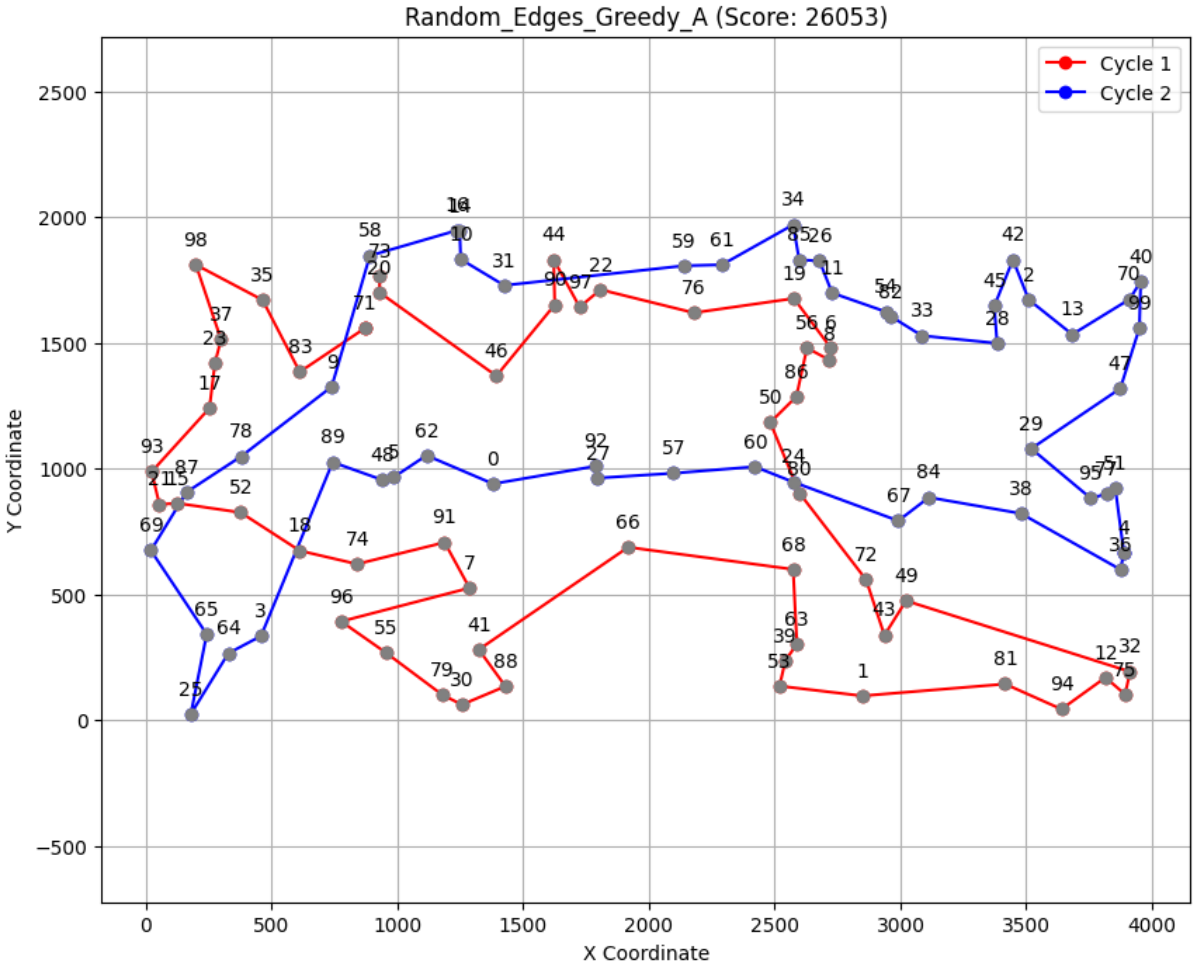
Analizując powyższe wyniki można dojść do ciekawych wniosków. Algorytm w wersji steep, za każdym razem doszedł do takiego samego rozwiązania. Jest to na swój sposób oczywiste, gdyż mając to samo rozwiązanie startowe, generuje tę samą listę ruchów, wybiera ten sam najlepszy ruch itd. Powstaje pytanie, dlaczego uzyskany w ten sposób wynik jest gorszy od rozwiązania greedy. Dzieje się tak, ponieważ dla tego konkretnego rozwiązania startowego, kilka pierwszych ruchów, pozornie najlepszych, oddala rozwiązanie od lepszego. Gdyby za każdym razem mieć inne rozwiązanie startowe, czy to losowe, czy to osiągnięte za pomocą 2-regret, wynik steep byłby lepszy.

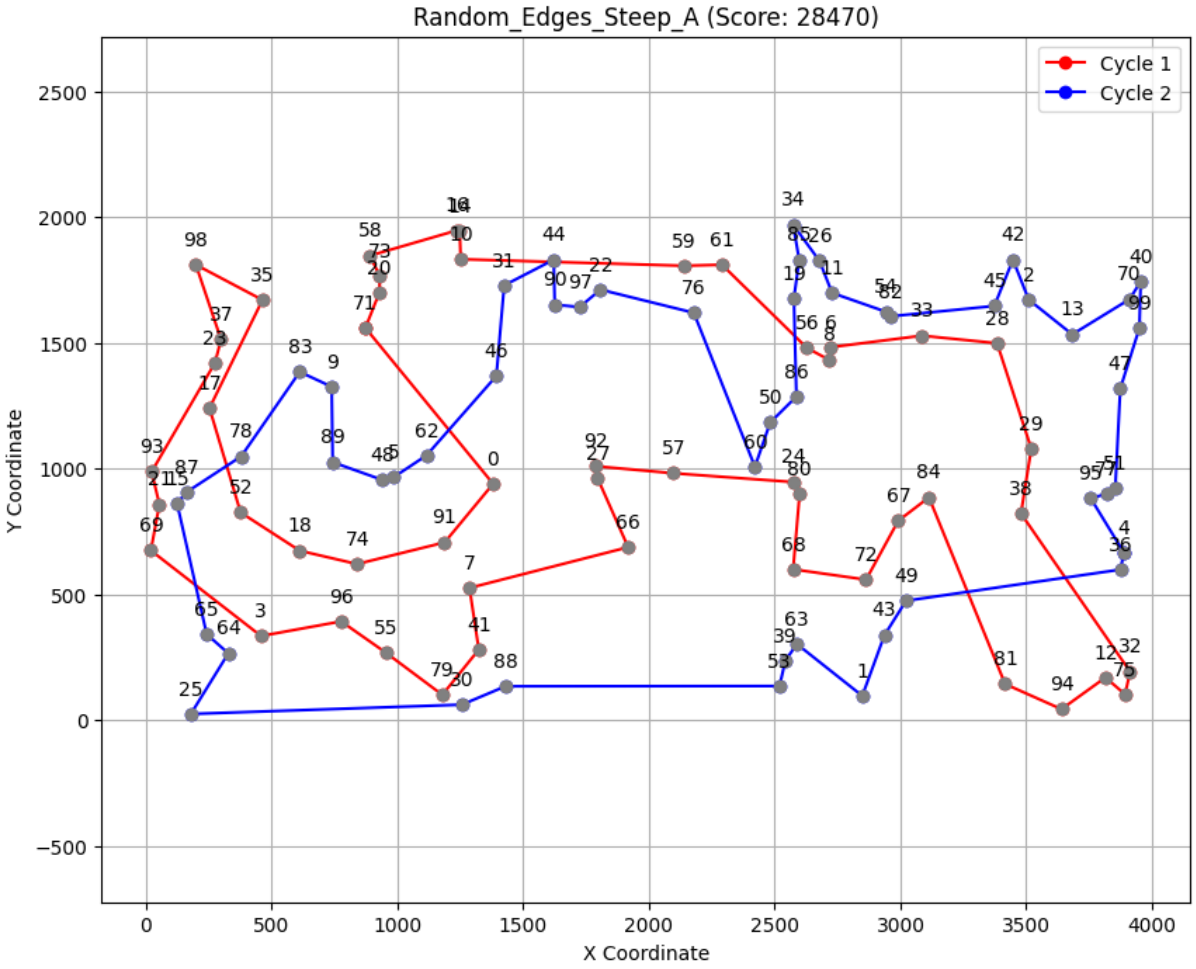
Wspólną cechą dla obu podejść (steep i greedy) jest to, ze wymiana krawędzi działała lepiej, niż wymiana wierzchołków.

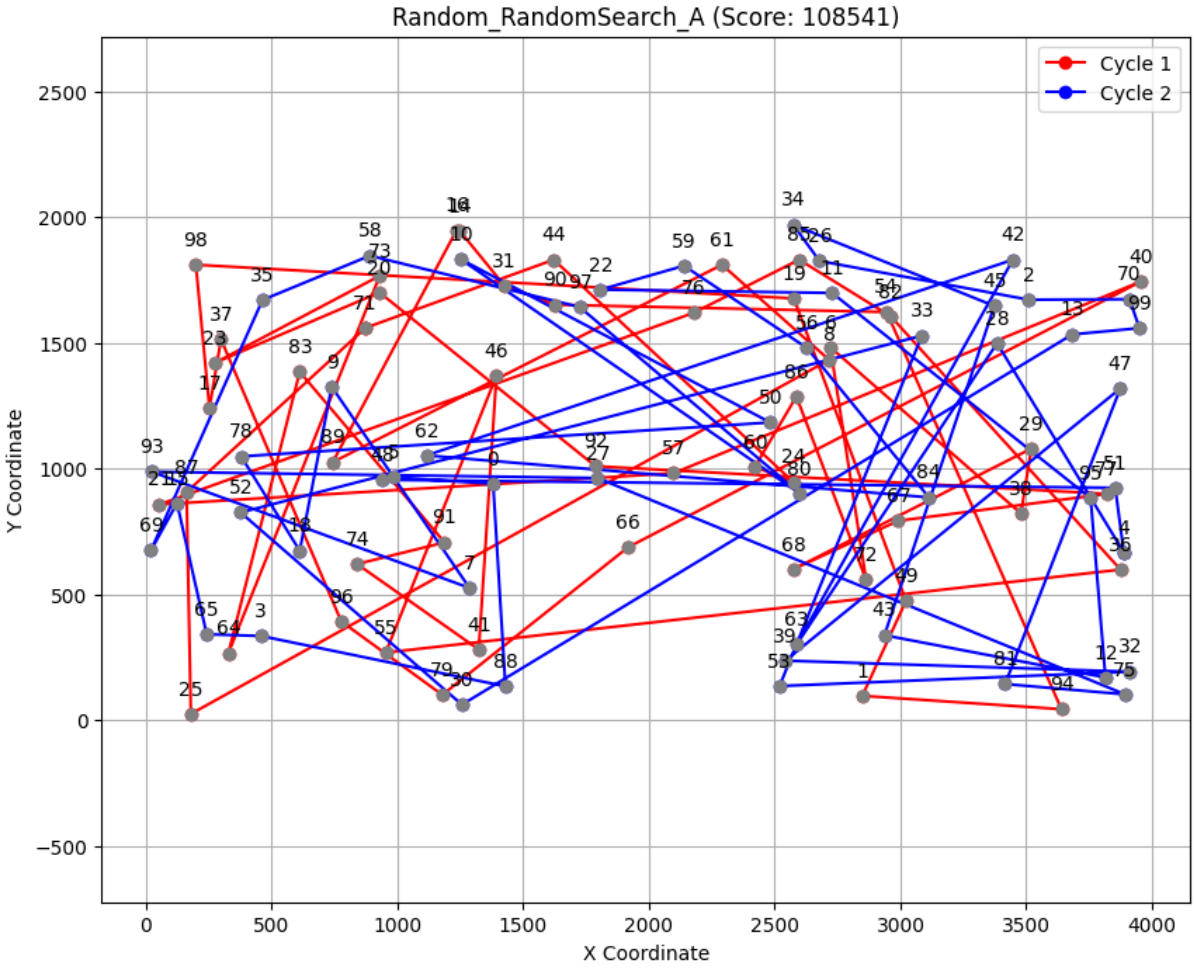
Wizualizacja najlepszych rozwiązań:

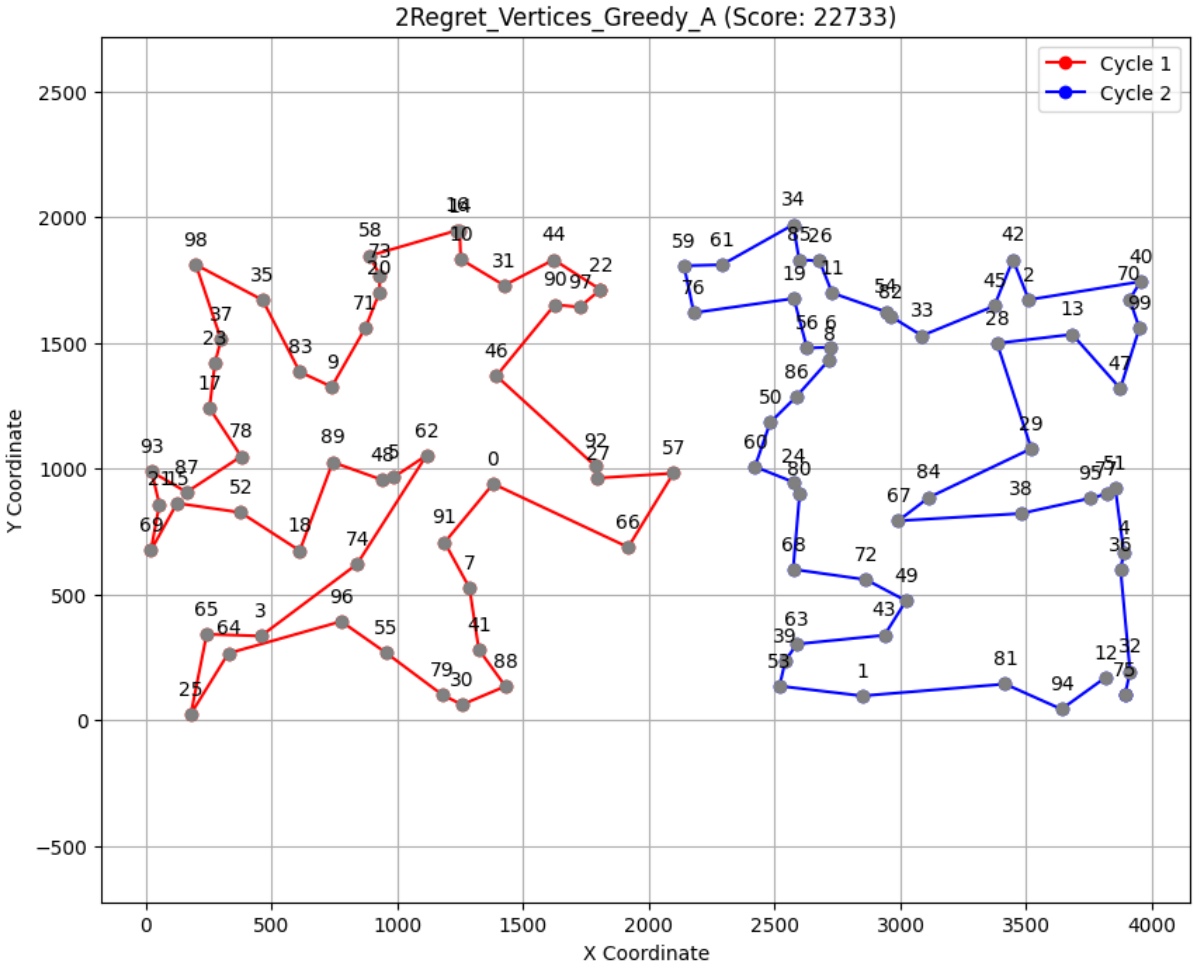


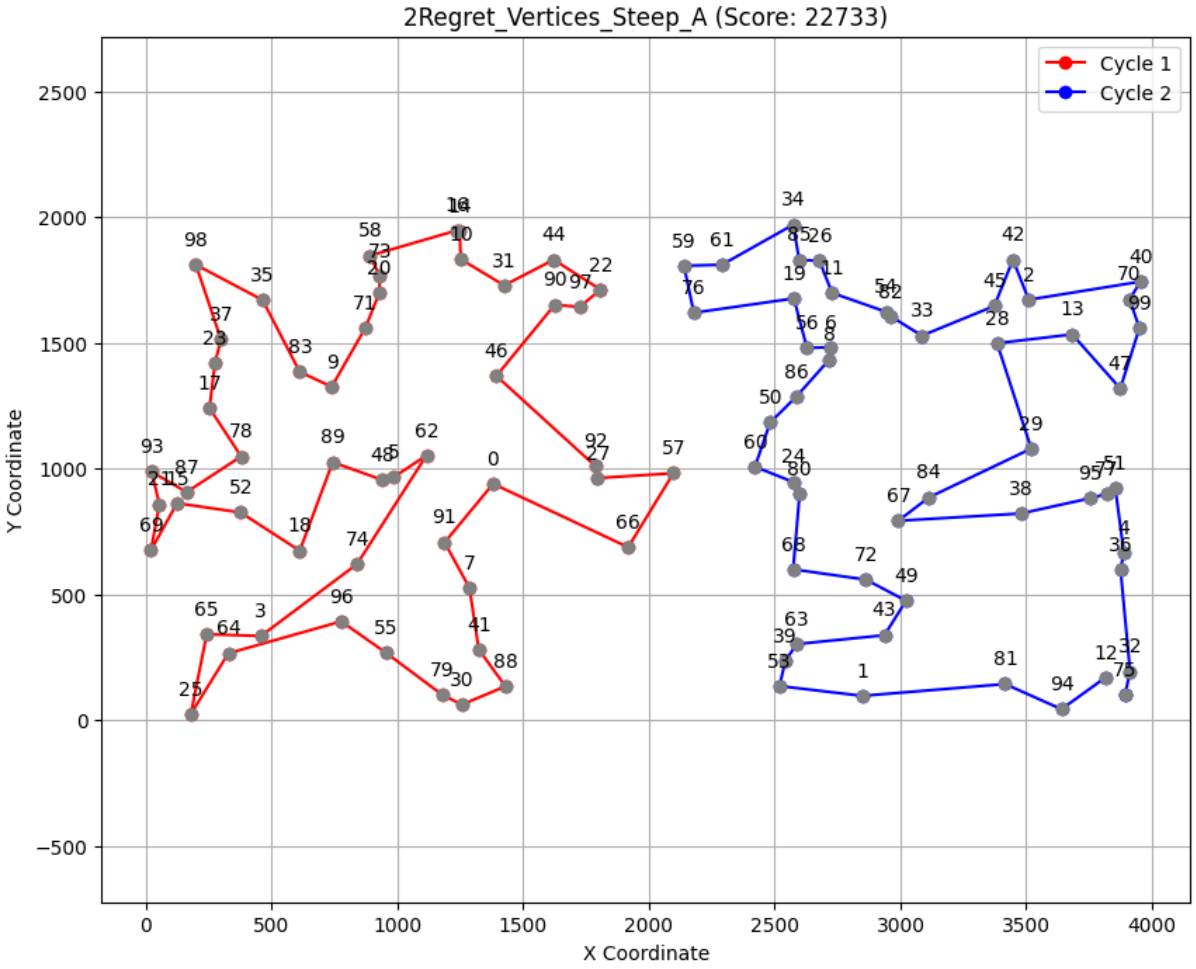

Random_Vertices_Steep_A (Score: 48429)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

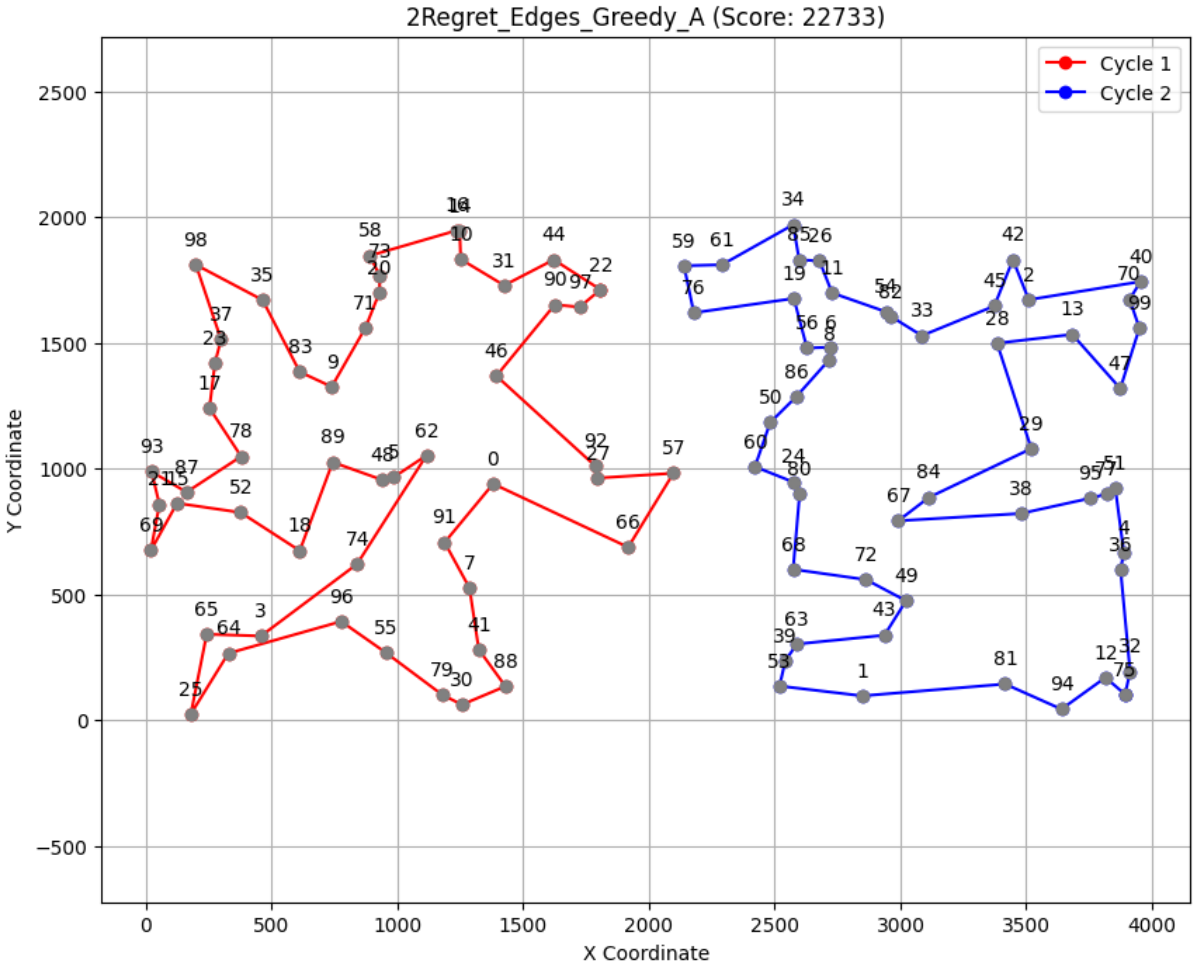

Random_Edges_Greedy_A (Score: 26053)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

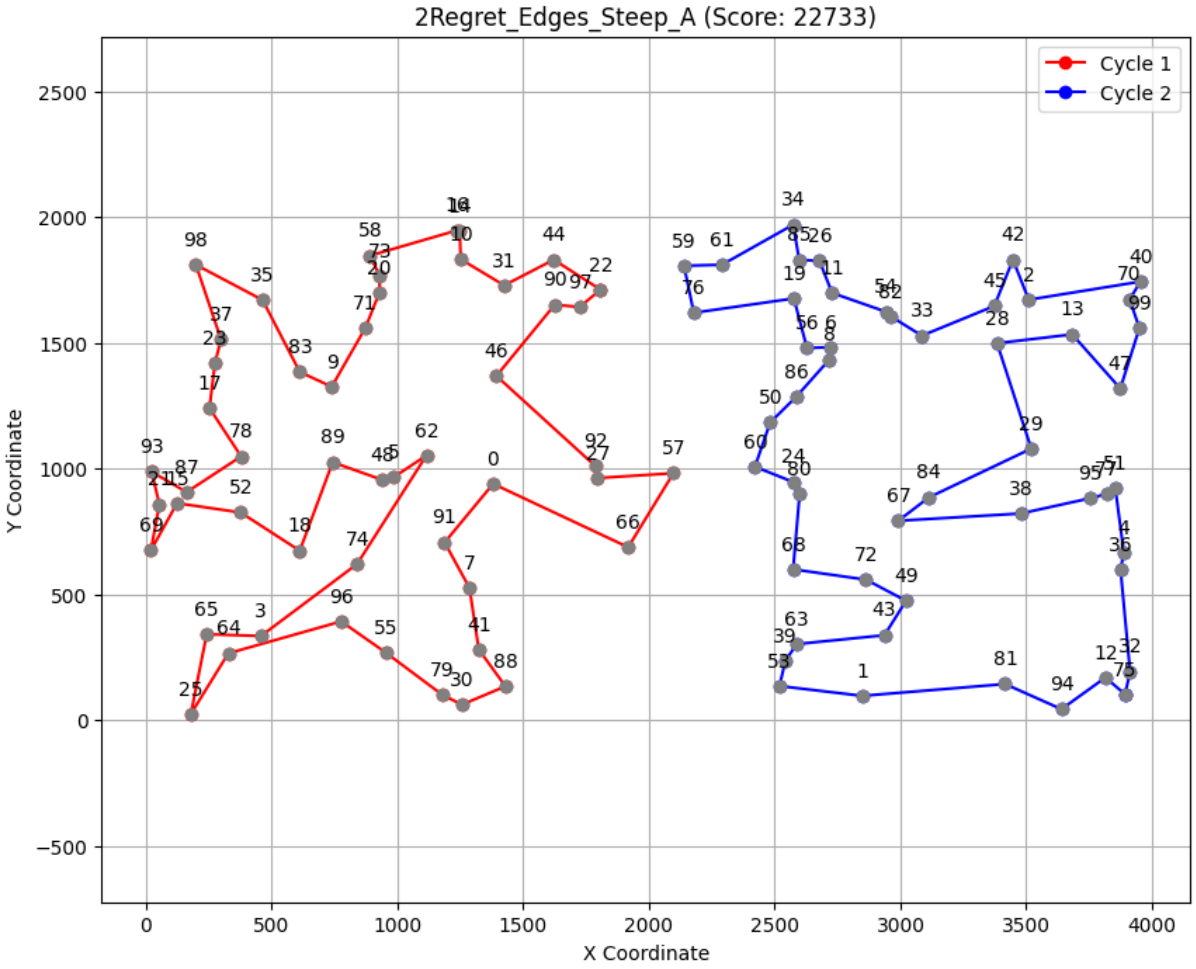

Random_Edges_Steep_A (Score: 28470)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

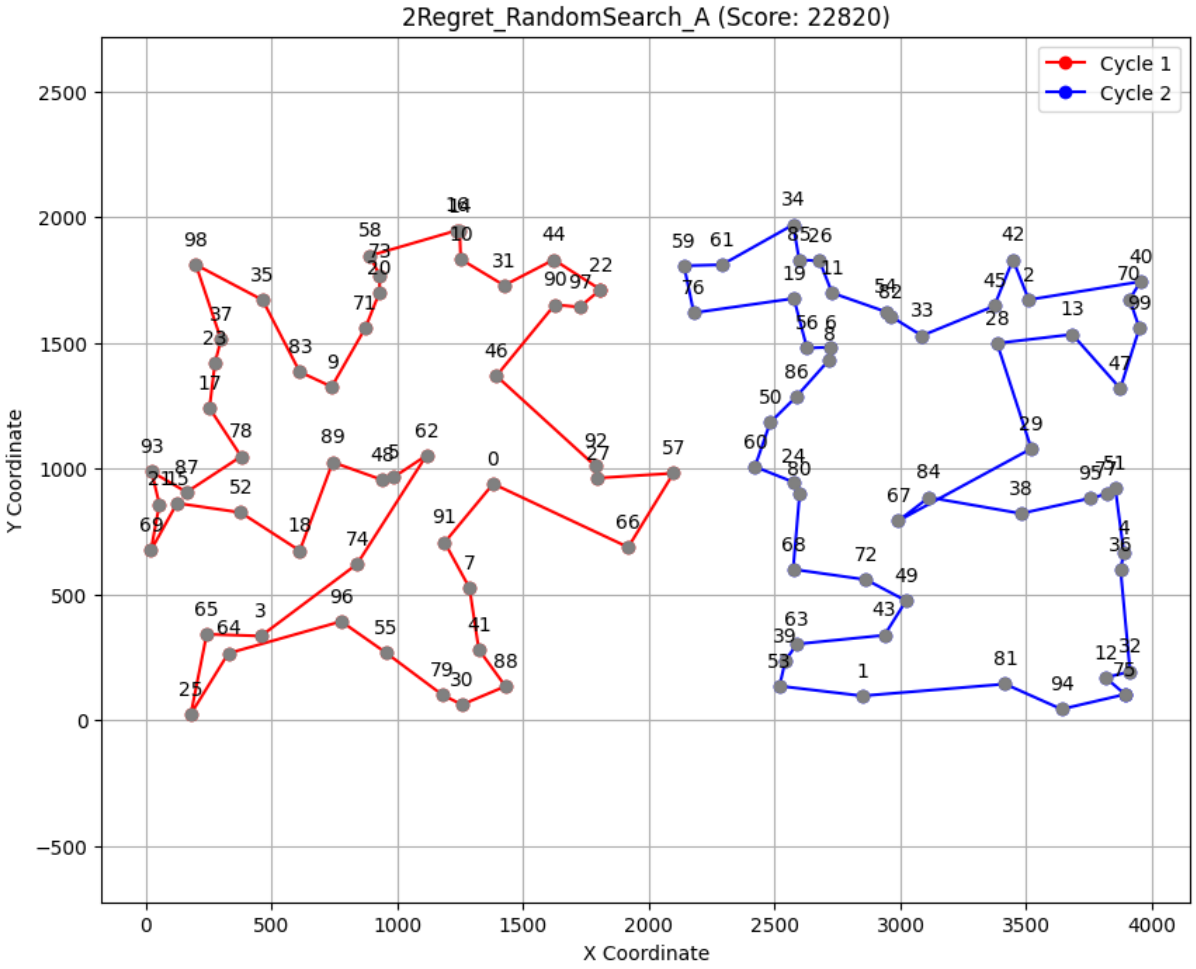

Random_RandomSearch_A (Score: 108541)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

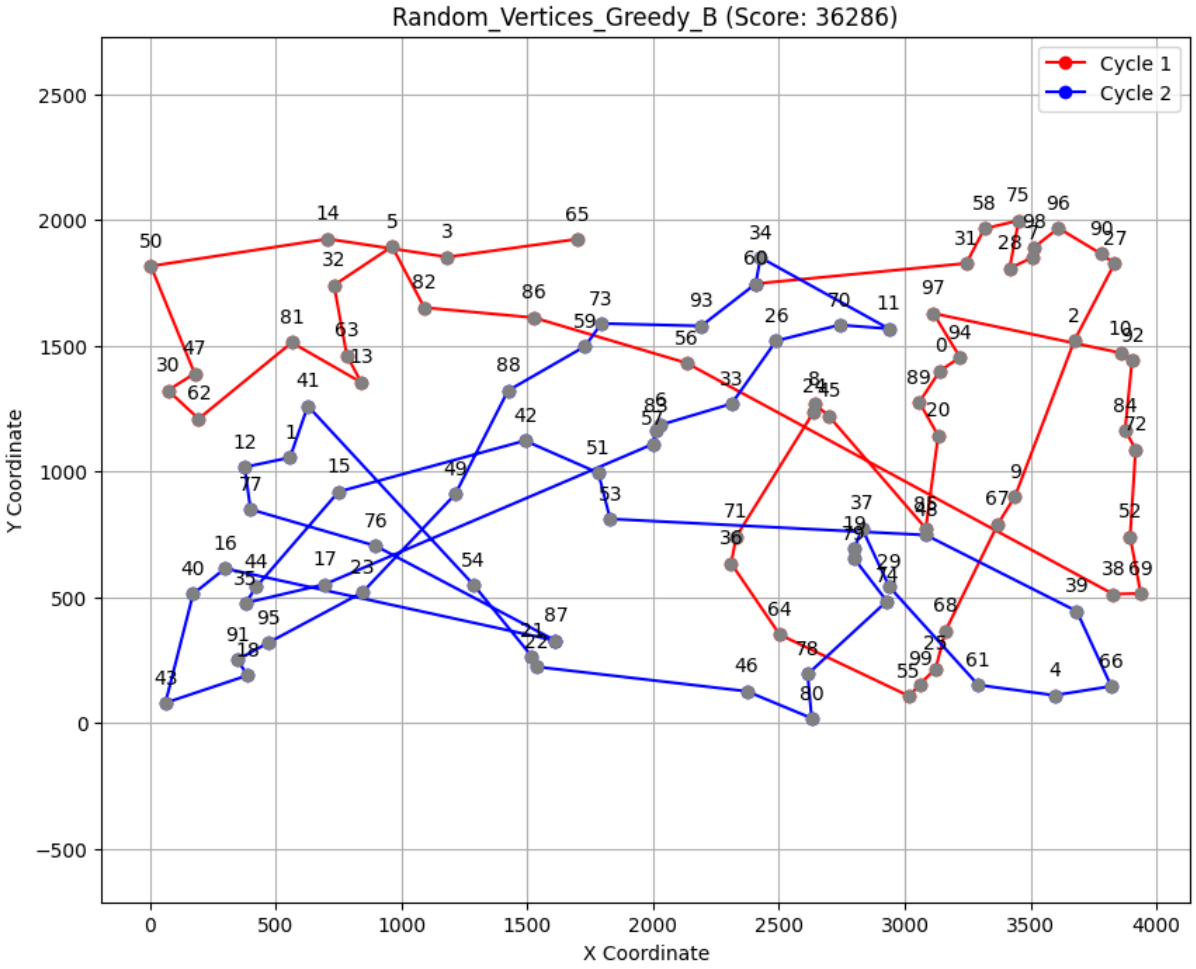

2Regret_Vertices_Greedy_A (Score: 22733)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

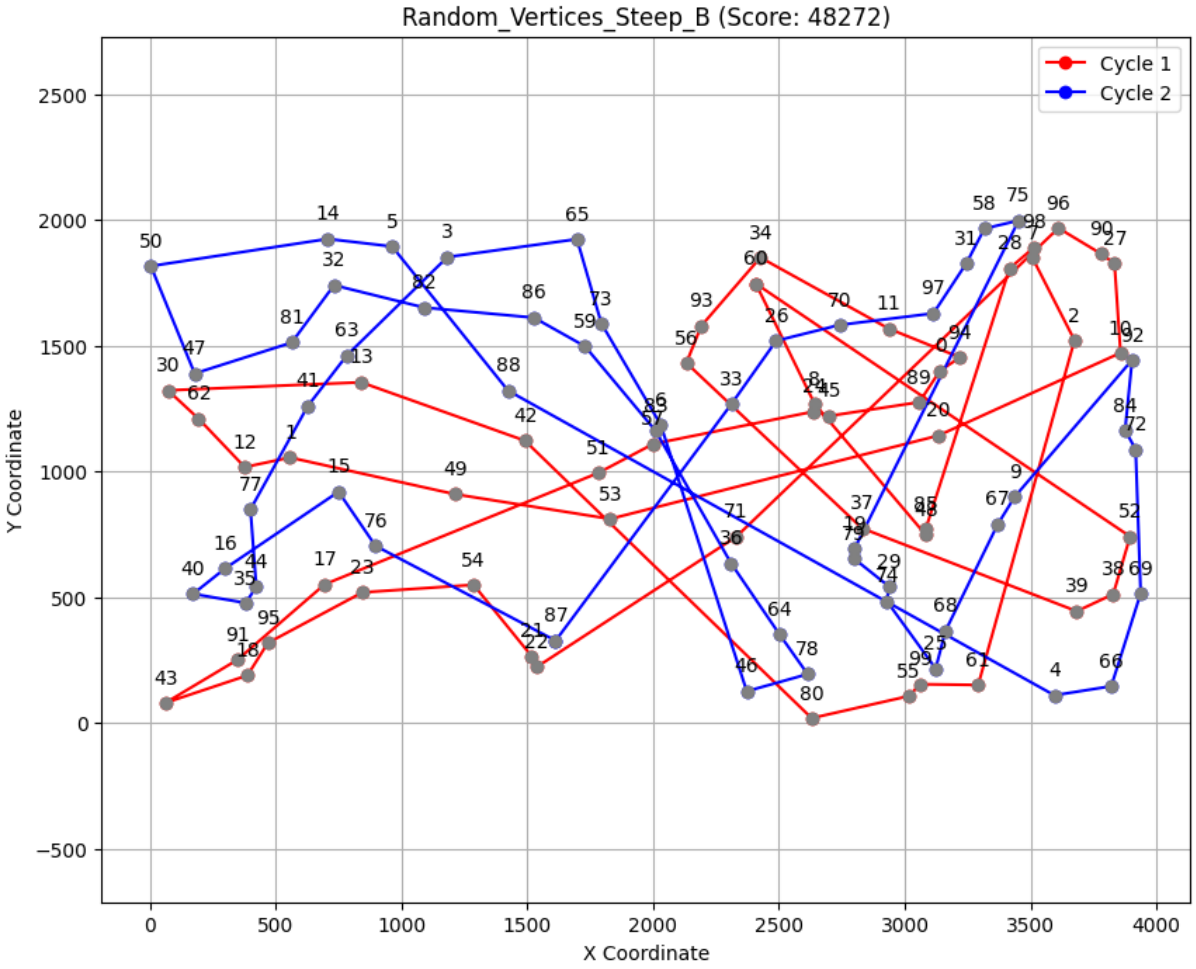

2Regret_Vertices_Steep_A (Score: 22733)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

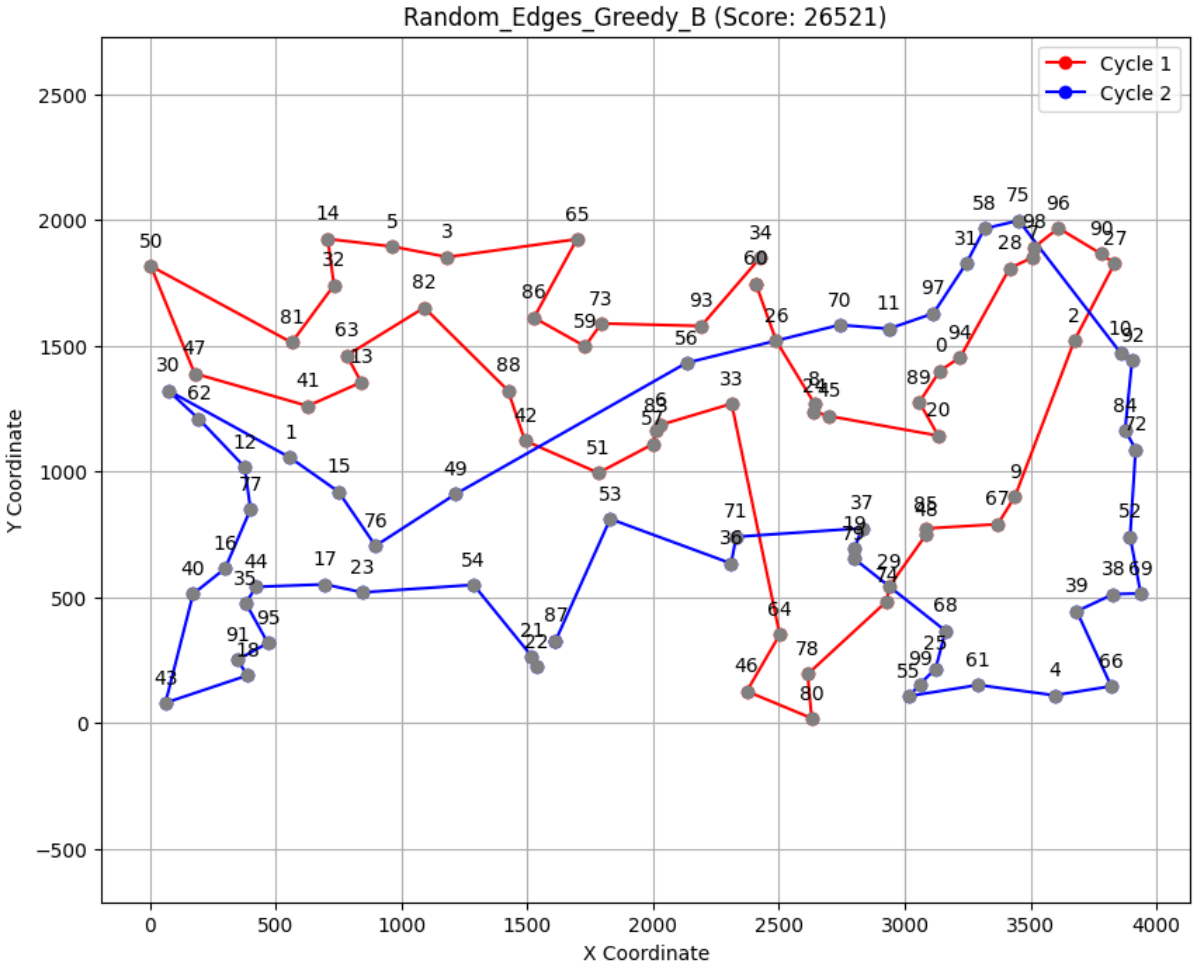

2Regret_Edges_Greedy_A (Score: 22733)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

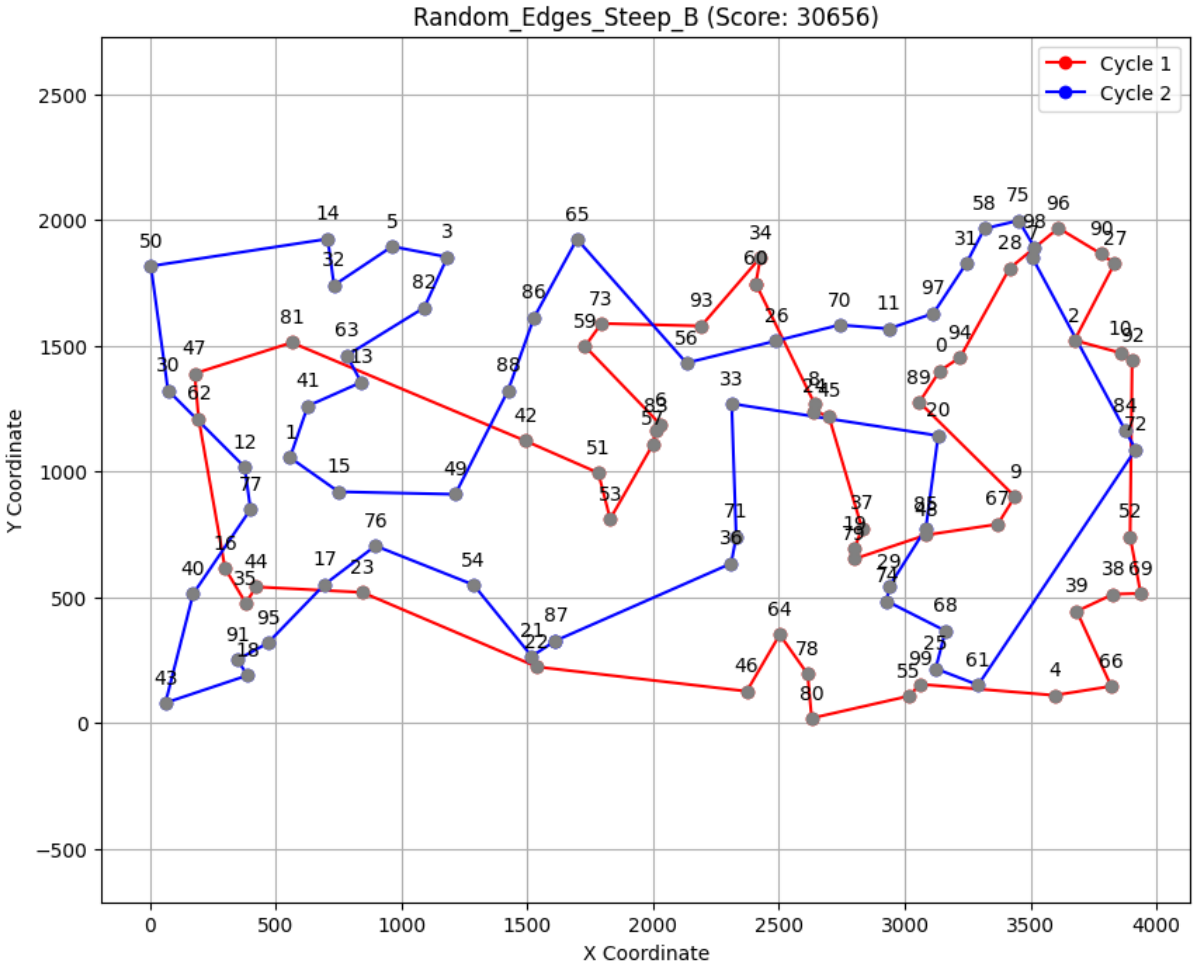

2Regret_Edges_Steep_A (Score: 22733)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

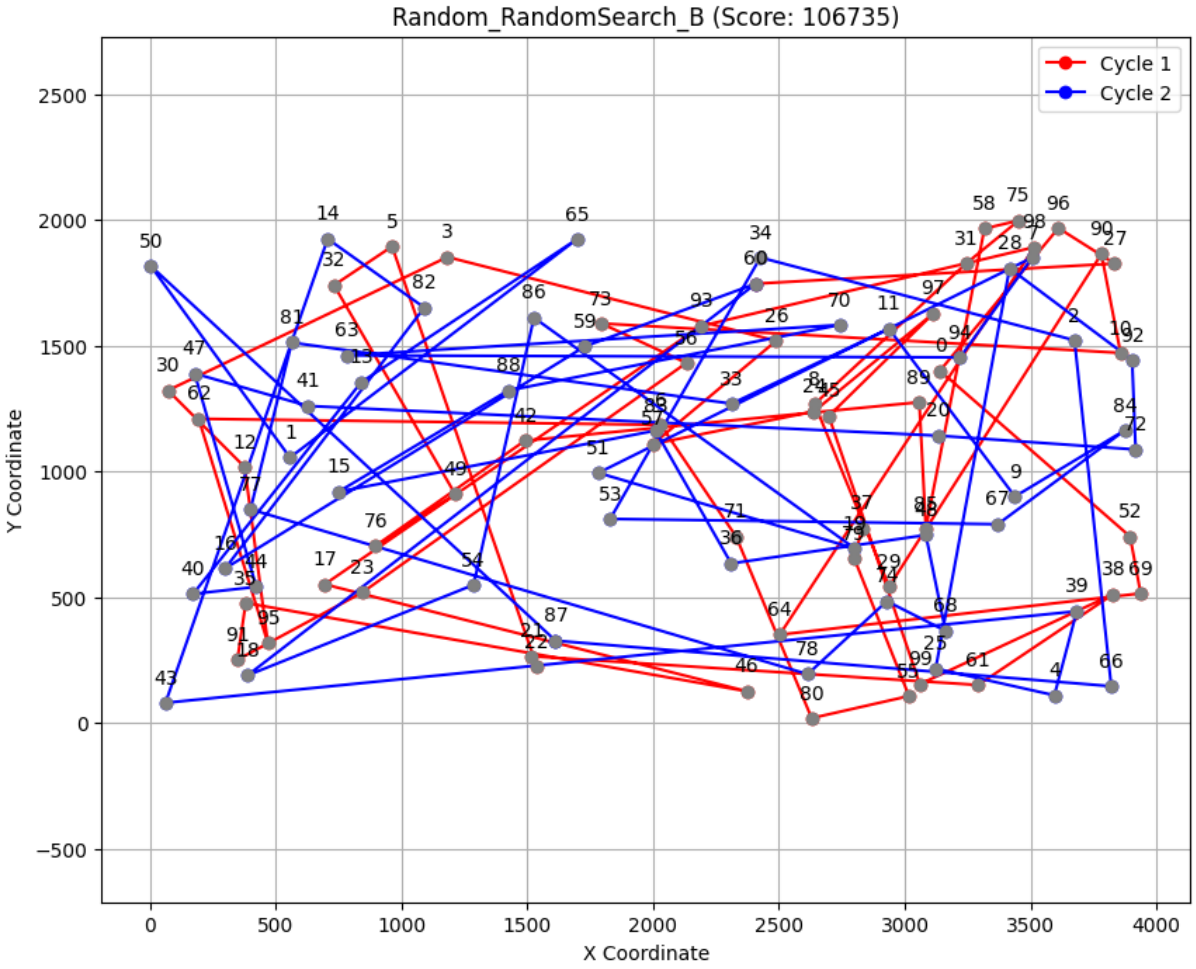

2Regret_RandomSearch_A (Score: 22820)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

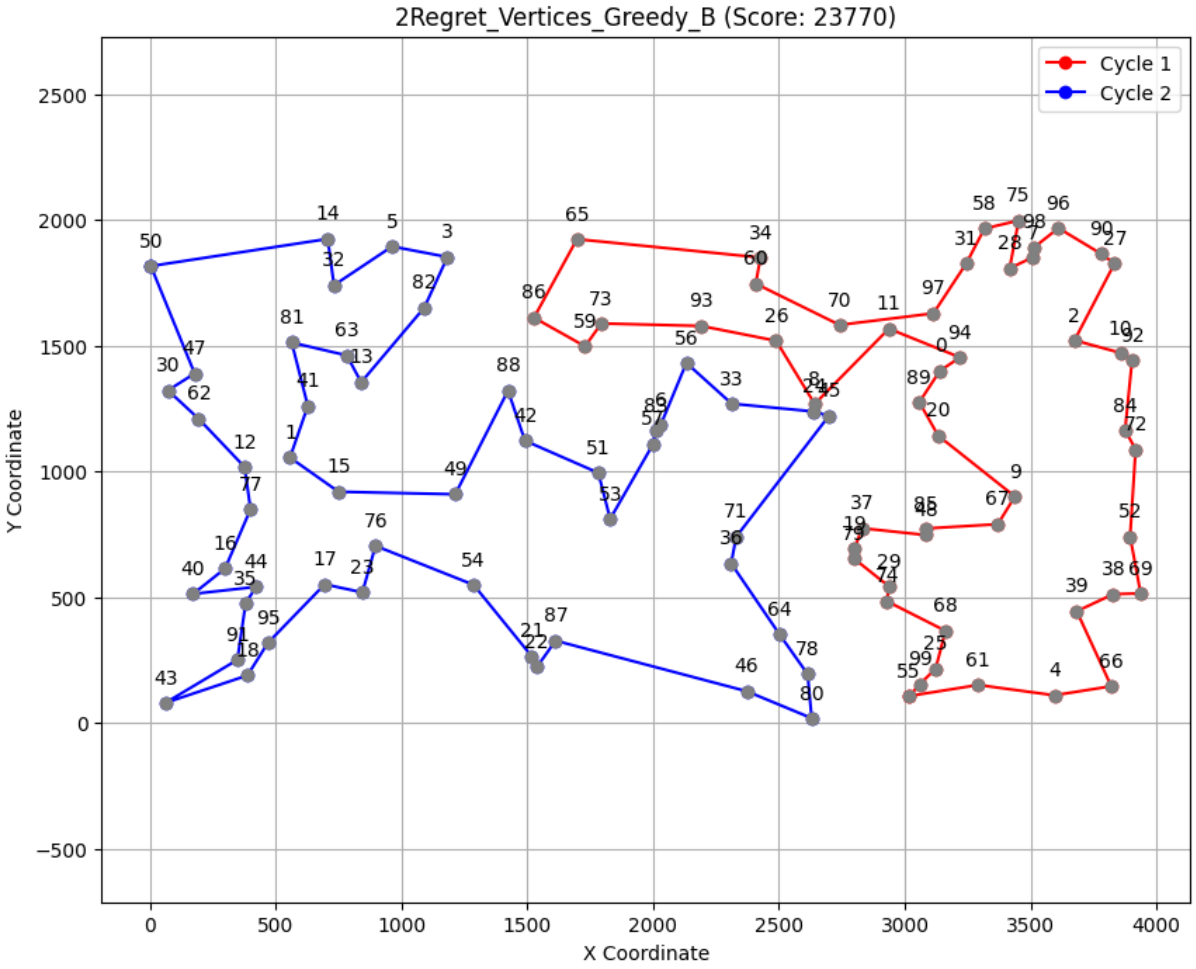

Random_Vertices_Greedy_B (Score: 36286)
Cycle 1
Cycle 2
X Coordinate
Y Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

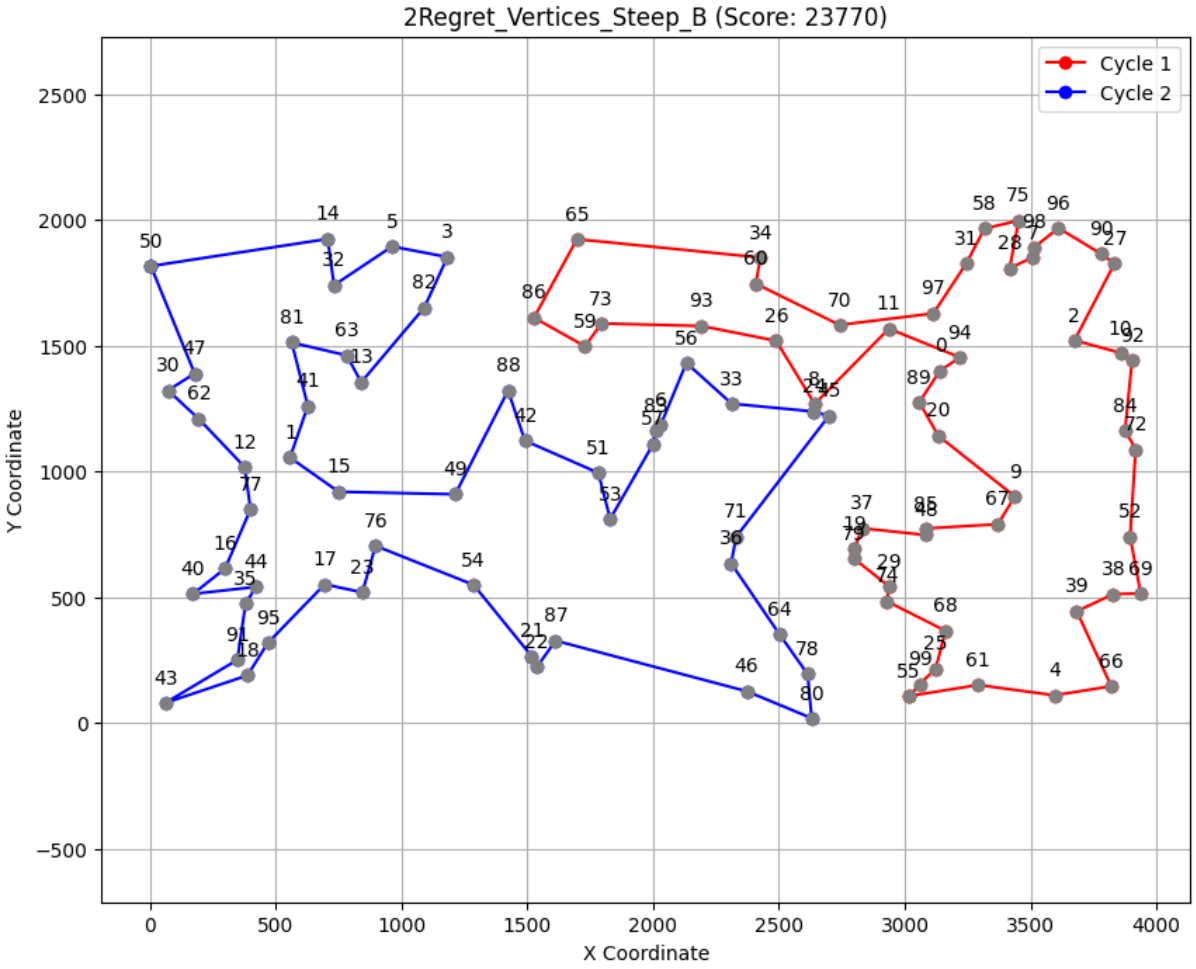

Random_Vertices_Steep_B (Score: 48272)
Cycle 1
Cycle 2
X Coordinate
Y Coordinate
0
500
1000
1500
2000
2500
3000
3500
4000
−500
0
500
1000
1500
2000
2500

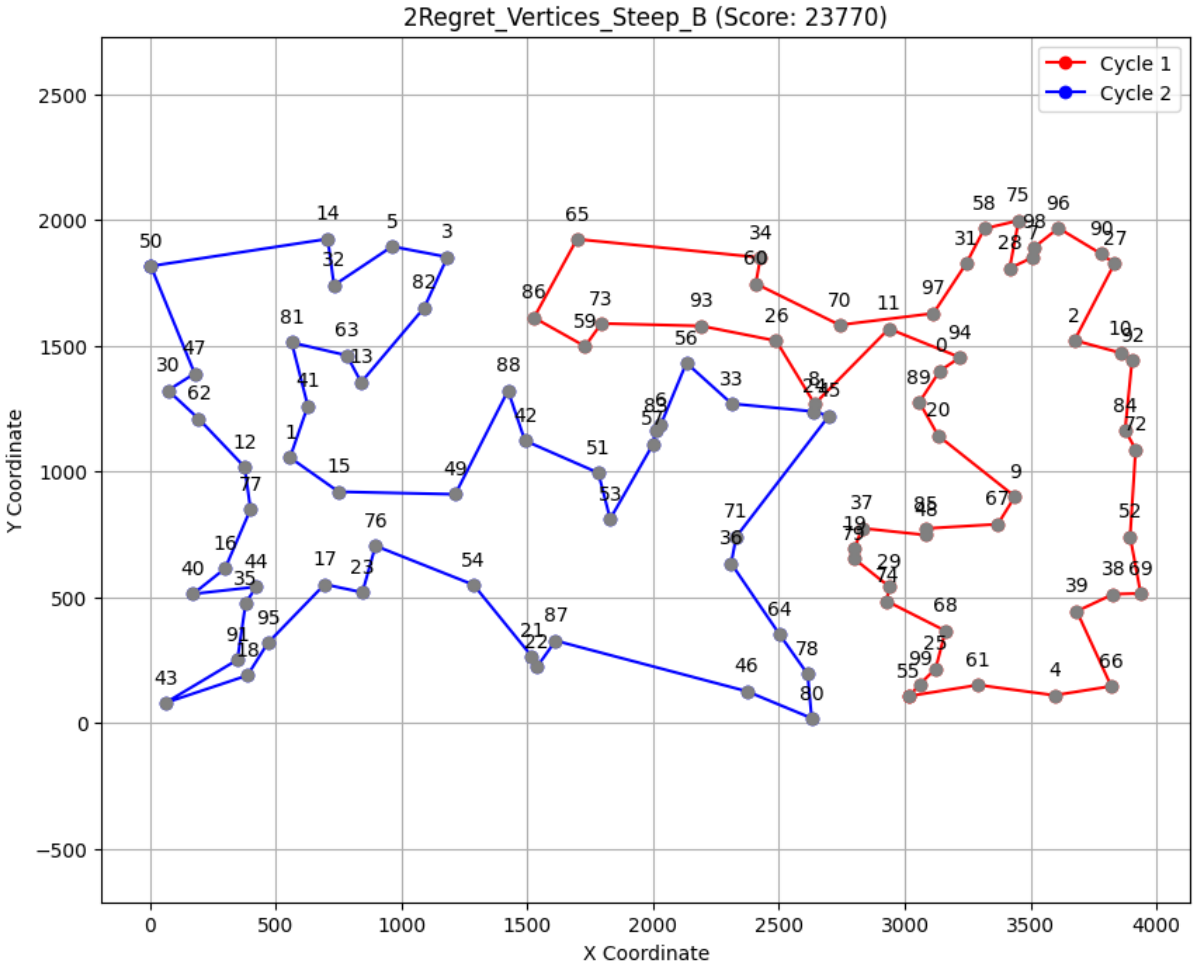

Random_Edges_Greedy_B (Score: 26521)
Cycle 1
Cycle 2
X Coordinate
Y Coordinate
0
500
1000
1500
2000
2500
3000
3500
4000
−500
0
500
1000
1500
2000
2500

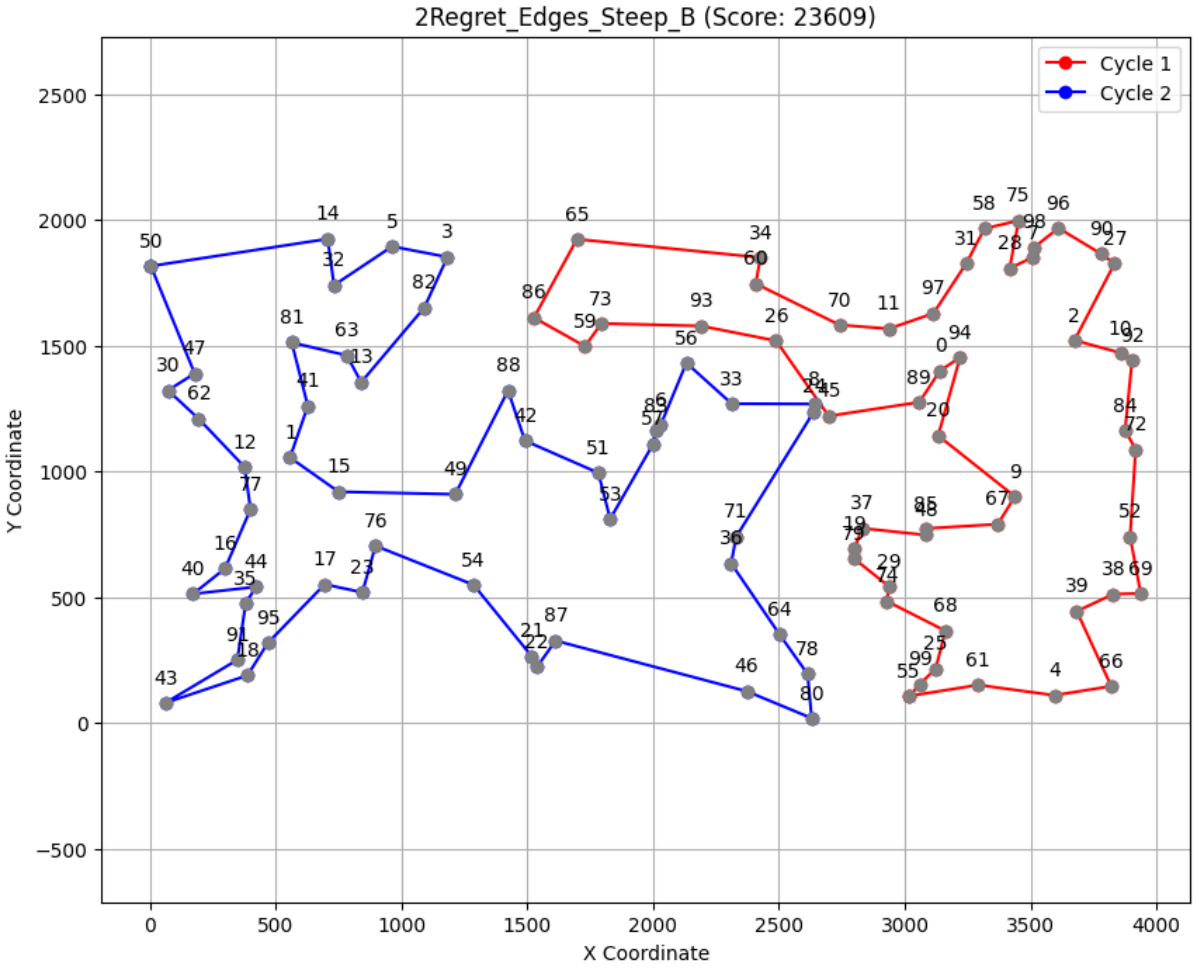

Random_Edges_Steep_B (Score: 30656)
Cycle 1
Cycle 2
X Coordinate
Y Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

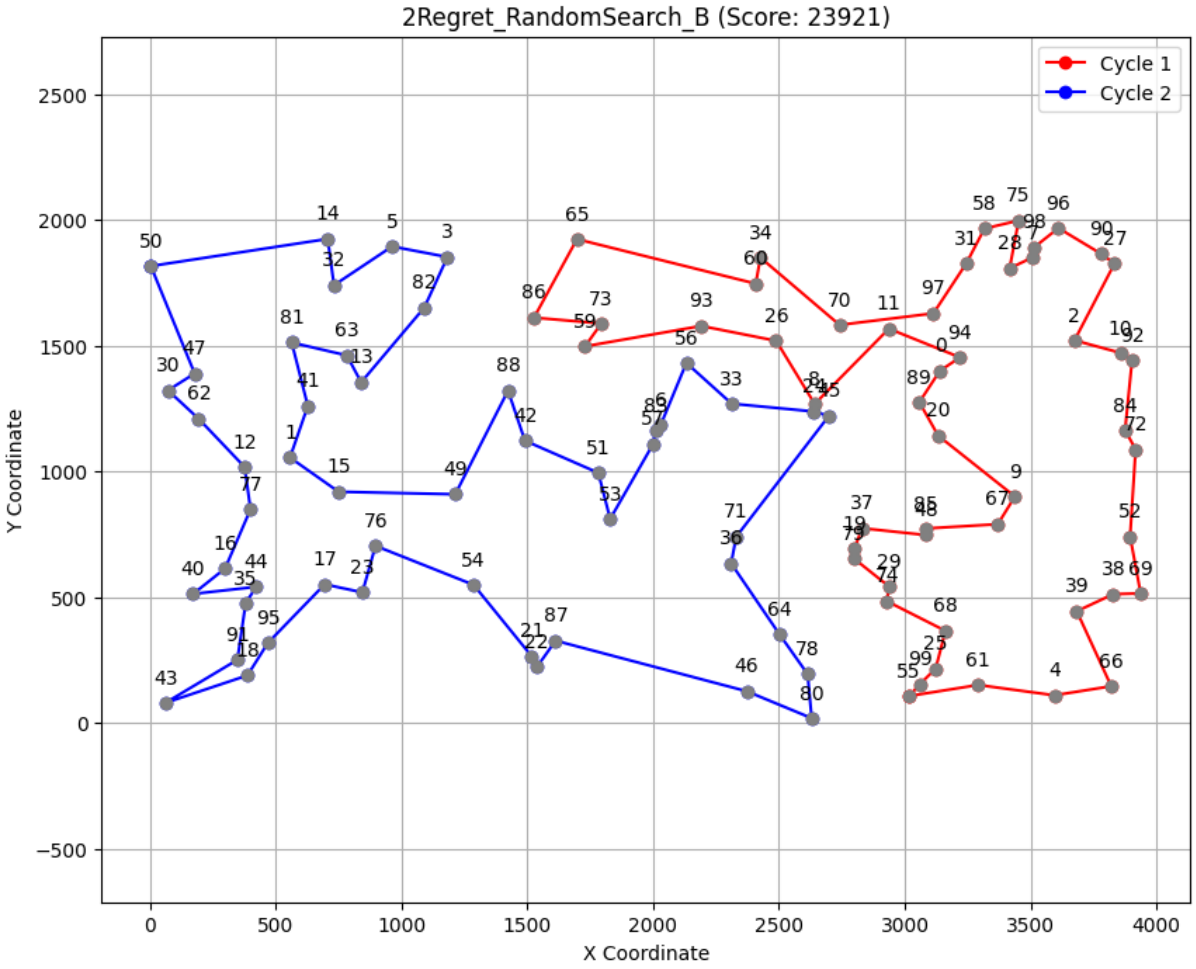

Random_RandomSearch_B (Score: 106735)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000


2Regret_Vertices_Greedy_B (Score: 23770)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000


2Regret_Vertices_Steep_B (Score: 23770)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000


2Regret_Vertices_Steep_B (Score: 23770)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000


2Regret_Edges_Steep_B (Score: 23609)
Cycle 1
Cycle 2
Y Coordinate
X Coordinate
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000



Kod programu

Kod programu jest dostępny pod linkiem: https://github.com/Evarios/IMO