Inteligentne metody optymalizacji

Laboratorium 2.

Lokalne przeszukiwane

Weronika Koga 151574
Dominika Nowak 151583
Informatyka stopień II semestr 1

7 kwietnia 2025

Spis treści

1 Opis zadania

Rozważany jest zmodyfikowany problem komiwojażera. Modyfikacja polega na uzyskaniu dwóch cykli zawierających po 50% wierzchołków instancji. Instancje - kroA200 oraz kroB200 - pobierane są z biblioteki TSPLib. Wymienione instancje zawierają współrzędne na płaszczyźnie według których wyliczana jest macierz odległości przekazywana jako wejście do algorytmów aproksymujących powyższy problem.

W ramach zadania zaimplementowano dwie wersje algorytmu lokalnego przeszukiwania - stromą oraz zachłanną. Dla obu powyższych wersji algorytmu zaimplementowano dwa rodzaje sąsiedztwa - ruchy zmieniające zbiory wierzchołków tworzące dwa cykle oraz te zmieniające kolejność wierzchołków na trasie. Algorytmy zawierają wersję rozpoczynającą z rozwiązania losowego oraz z rozwiązania uzyskanego algorytmem ważonego 2-żalu.

Zaimplementowano również algorytm losowego błądzenia, którego warunkiem stop jest osiągnięcie czasu wykonywania się funkcji równego średniemu czasowi wykonania stromego lokalnego przeszukiwania startującego z rozwiązania losowego.

2 Zaimplementowane algorytmy

2.1 Algorytm zachłanny lokalnego przeszukania

Algorithm 1 Algorytm Zachłanny Lokalnego Przeszukania

Wczytaj liczbę węzłów n oraz macierz odległości.
Utwórz rozwiązanie początkowe za pomocą algorytmu losowego lub ważonego 2-żalu.
Utwórz listę sąsiedztw zawierającą możliwe pary ruchów w postaci indeksów cykli:
    $N \leftarrow$ Unikalne pary indeksów obu cykli //ruchy zewnątrztrasowe
if wykorzystujemy ruchy wewnątrztrasowe wymiany wierzchołków **then**
    $N \leftarrow$ Unikalne pary indeksów wewnątrz cyklu
else
    $N \leftarrow$ Pary niesąsiadujących indeksów wewnątrz cyklu (wymiana krawędzi)
end if
while znaleziono poprawę **do**
    Przetasuj listę sąsiedztw.
    for Następny ruch z listy **do**
        Oblicz deltę funkcji celu δ dla ruchu.
        if $wynik + \delta < wynik$ **then**
            Aplikuj ruch na rozwiązanie
            Uaktualnij jakość rozwiązania $wynik = \delta + wynik$
            Powróć do początku WHILE (poprawa znaleziona)
        end if
    end for
end while

2.2 Algorytm stromy lokalnego przeszukania

Algorithm 2 Algorytm Stromy Lokalnego Przeszukania

Wczytaj liczbę węzłów n oraz macierz odległości.
Utwórz rozwiązanie początkowe za pomocą algorytmu losowego lub ważonego 2-żalu.
Utwórz listę sąsiedztw zawierającą możliwe pary ruchów w postaci indeksów cykli:
    $N \leftarrow$ Unikalne pary indeksów obu cykli //ruchy zewnątrztrasowe
if wykorzystujemy ruchy wewnątrztrasowe wymiany wierzchołków **then**
    $N \leftarrow$ Unikalne pary indeksów wewnątrz cyklu
else
    $N \leftarrow$ Pary niesąsiadujących indeksów wewnątrz cyklu (wymiana krawędzi)
end if
while znaleziono poprawę **do**
    $W_b \leftarrow W_{start}$ // najlepszy znaleziony wynik
    $R_b \leftarrow None$ // najlepszy znaleziony ruch
    for Następny ruch z listy **do**
        Oblicz deltę funkcji celu δ dla ruchu.
        if $W_b + \delta < W_b$ **then**
            Uaktualnij najlepszy znaleziony ruch R_b
            Uaktualnij najlepszy znaleziony wynik $W_b = \delta + W_b$
        end if
    end for
    if $W_b! = W_{start}$ **then**
        Aplikuj najlepszy znaleziony ruch R_b na rozwiązanie
        Uaktualnij jakość rozwiązania $wynik = W_b$
        Powróć do początku WHILE (poprawa znaleziona)
    end if
end while

2.3 Algorytm losowego błądzenia

Algorithm 3 Algorytm Losowego Błądzenia

Wczytaj liczbę węzłów n oraz macierz odległości.
Utwórz rozwiązanie początkowe za pomocą algorytmu losowego lub ważonego 2-żalu.
Utwórz listę sąsiedztw zawierającą możliwe pary ruchów w postaci indeksów cykli:
    $N \leftarrow$ Unikalne pary indeksów obu cykli //ruchy zewnątrztrasowe
if wykorzystujemy ruchy wewnątrztrasowe wymiany wierzchołków **then**
    $N \leftarrow$ Unikalne pary indeksów wewnątrz cyklu
else
    $N \leftarrow$ Pary niesąsiadujących indeksów wewnątrz cyklu (wymiana krawędzi)
end if
$W_b \leftarrow W_{start}$ // najlepszy znaleziony wynik
$W_c \leftarrow W_{start}$ // aktualny wynik
$C_b \leftarrow C$ // najlepsze znalezione rozwiązanie
$C_c \leftarrow C$ // aktualne rozwiązanie
$czas \leftarrow 0$
while $czas < koniec\_czas$ **do**
    Wylosuj ruch z listy sąsiedztw.
    Oblicz deltę funkcji celu δ dla ruchu.
    Uaktualnij wynik $W_c = W_c + \delta$.
    Aplikuj ruch na aktualne rozwiązanie C_c.
    if $W_c < W_b$ **then**
        Uaktualnij najlepszy znaleziony wynik $W_b = \delta + W_c$.
        Uaktualnij najlepsze znalezione rozwiązanie $C_b = C_c$.
    end if
end while

3 Wyniki

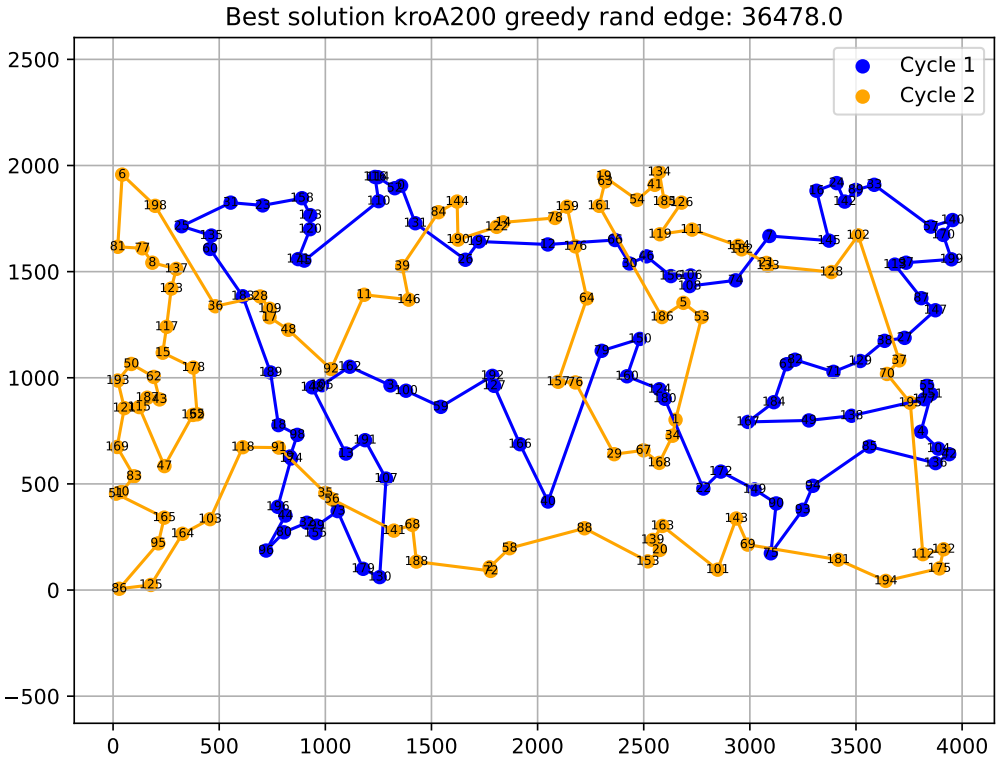
start	**swap type**	**algorithm**	**kroA200**			**kroB200**		
			min	**avg**	**max**	**min**	**avg**	**max**
random	vertex	greedy	63609	72870.94	88339	60413	71280.08	83481
		steepest	67036	78289.6	98059	62720	75756.74	88594
		randomWalk	280533	287068.29	293840	271770	282566.38	288238
	edge	greedy	36478	39993.08	43637	37505	39907.63	43047
		steepest	36571	40278.95	43376	37589	40319.2	43139
		randomWalk	278098	287788.35	294433	267997	282124.61	288939
weighted 2-regret	vertex	greedy	31865	35358.88	38491	33262	35769.03	37391
		steepest	32196	35688.87	38525	33241	35839.57	37922
		randomWalk	35211	43084.02	51114	36544	43117.43	51929
	edge	greedy	31748	34688	37418	32655	35030.27	37286
		steepest	31071	34402.5	37084	32538	35158.35	37331
		randomWalk	34047	41079.71	51227	34364	40367.46	50868

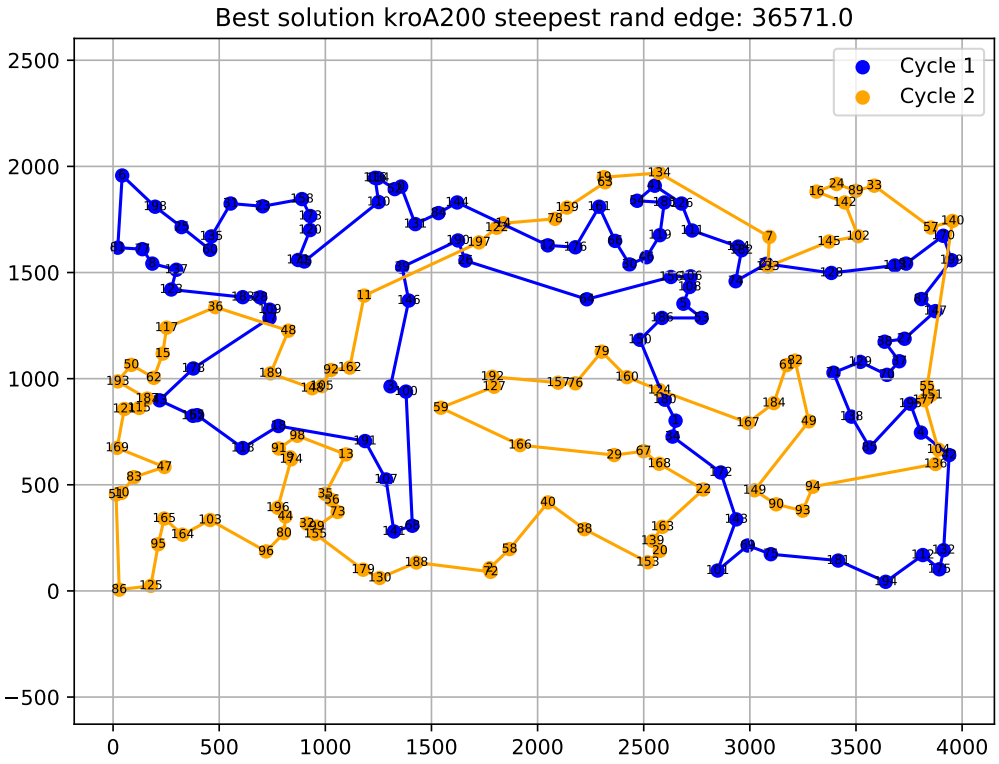
Tabela 1: Wyniki eksperymentu obliczeniowego

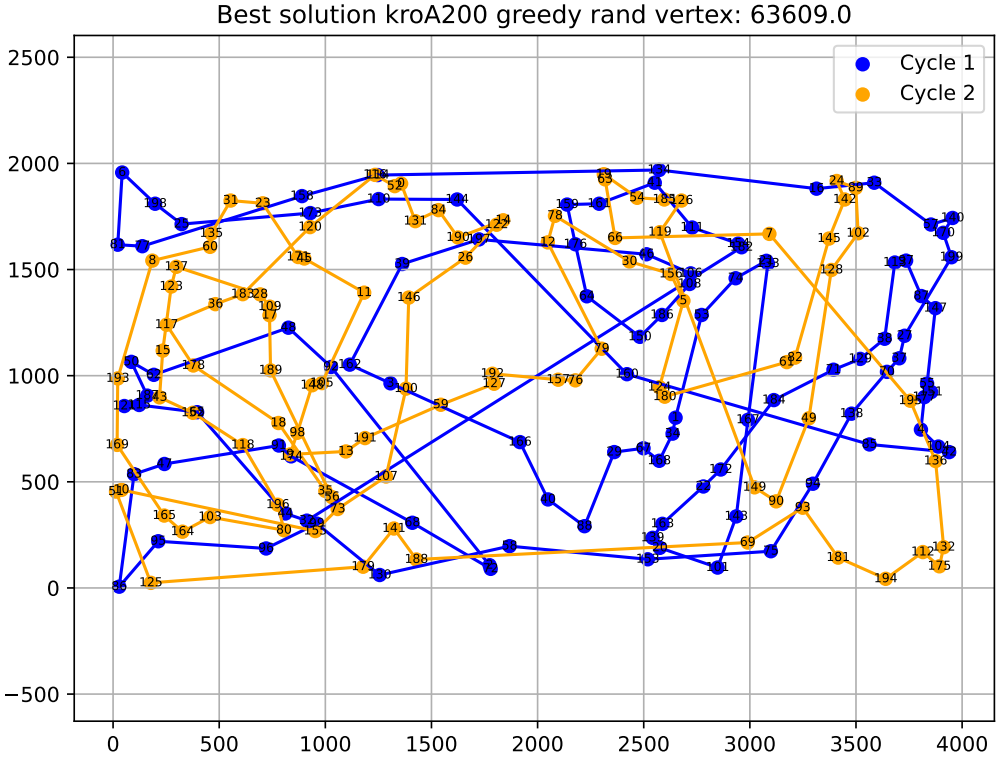
start	swap type	algorithm	kroA200			kroB200		
			min [s]	avg [s]	max [s]	min [s]	avg [s]	max [s]
random	vertex	greedy	0.9102	1.2942	1.9601	0.7931	1.1400	2.1103
		steepest	12.9211	15.8398	21.0183	11.7431	14.3152	18.1438
	edge	greedy	0.7759	1.0205	1.5061	0.7122	0.9752	1.4906
		steepest	11.9497	13.7989	16.6503	11.2829	12.9105	15.1207
weighted 2-regret	vertex	greedy	1.2664	1.4548	3.5011	1.2939	1.4314	1.7999
		steepest	1.5522	2.0298	2.7391	1.5898	1.5898	2.8504
	edge	greedy	1.3093	1.5260	1.8990	1.3143	1.4147	1.7253
		steepest	2.0735	2.7225	3.7147	1.8092	2.2261	2.8942

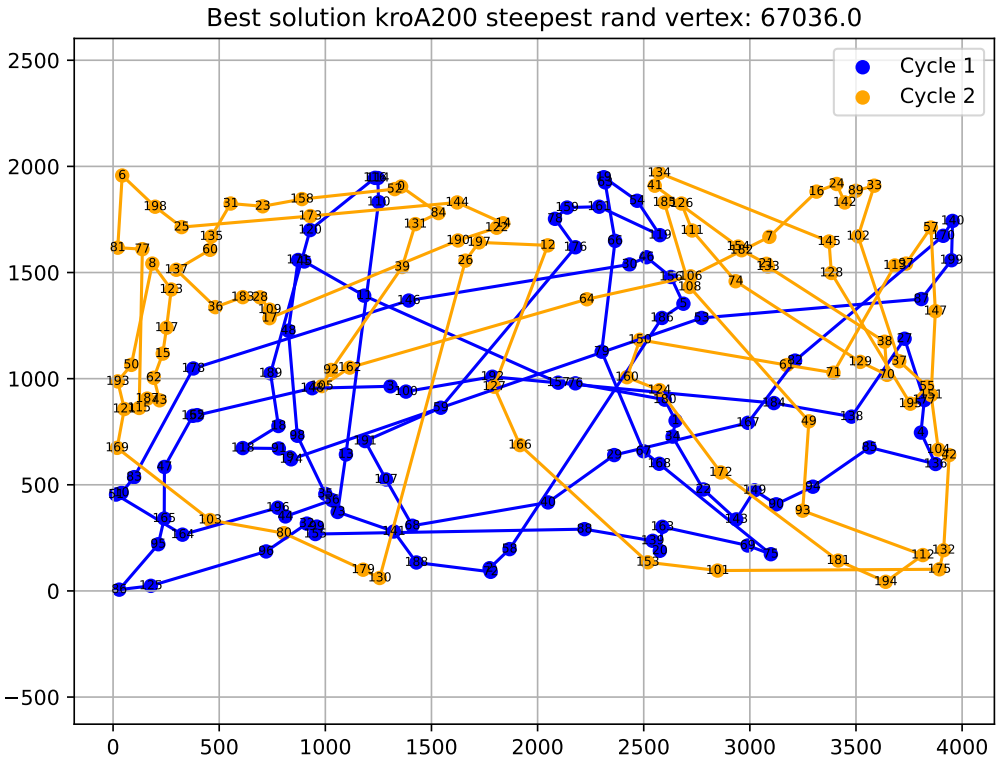
Tabela 2: Wyniki eksperymentu czasowego

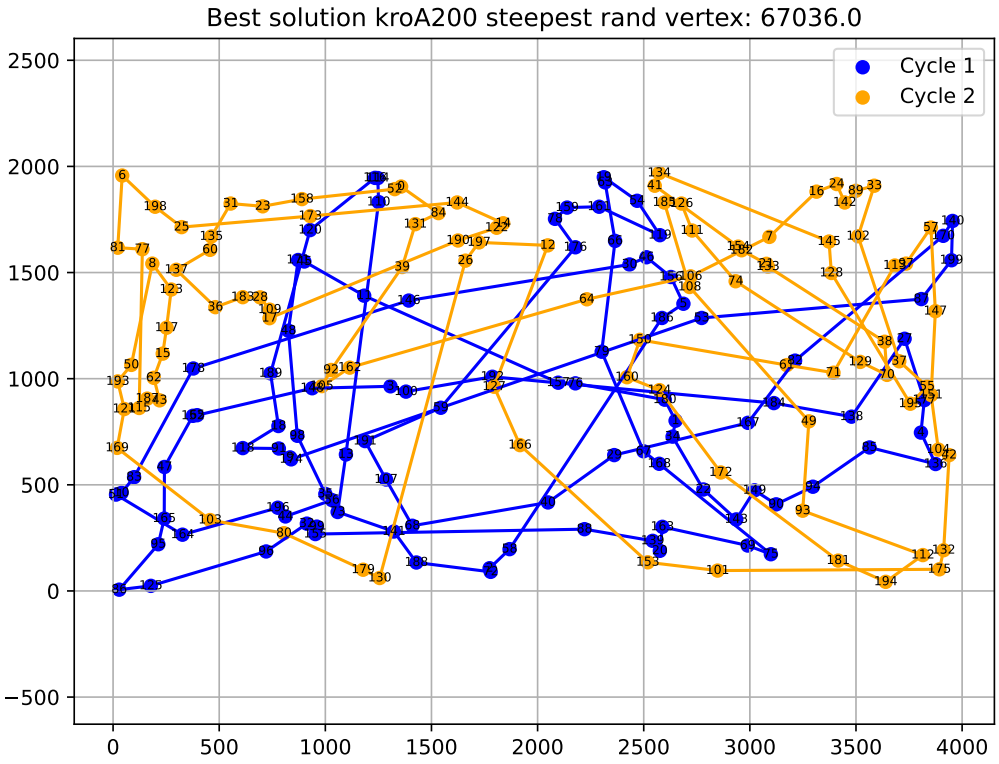
4 Wizualizacje najlepszych rozwiązań

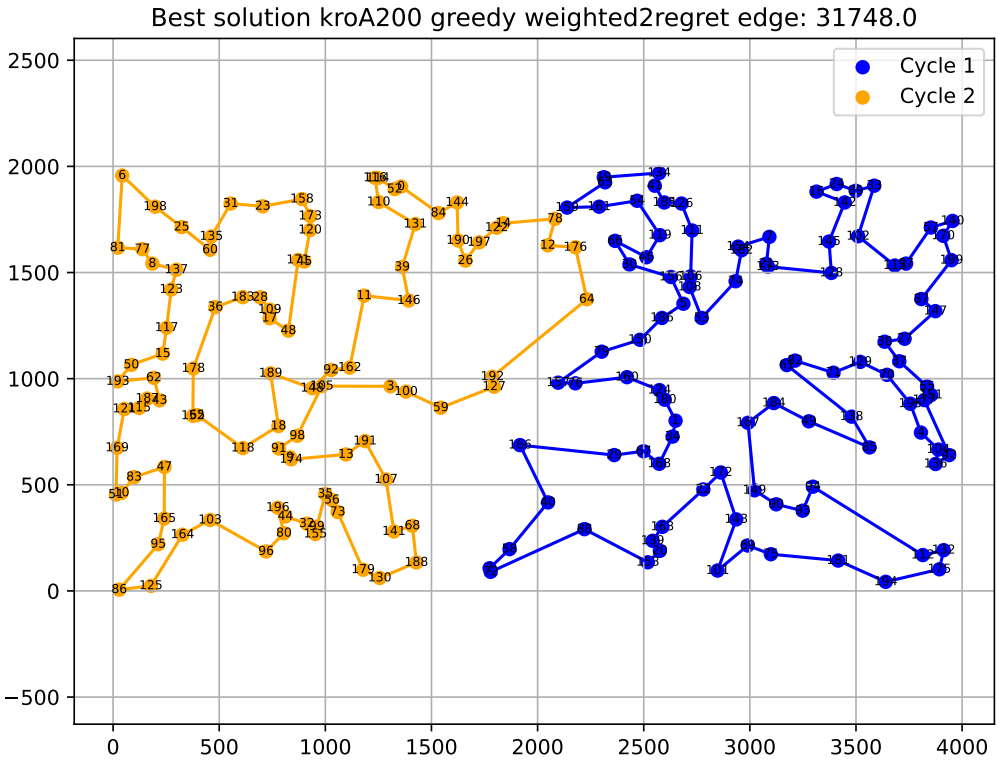


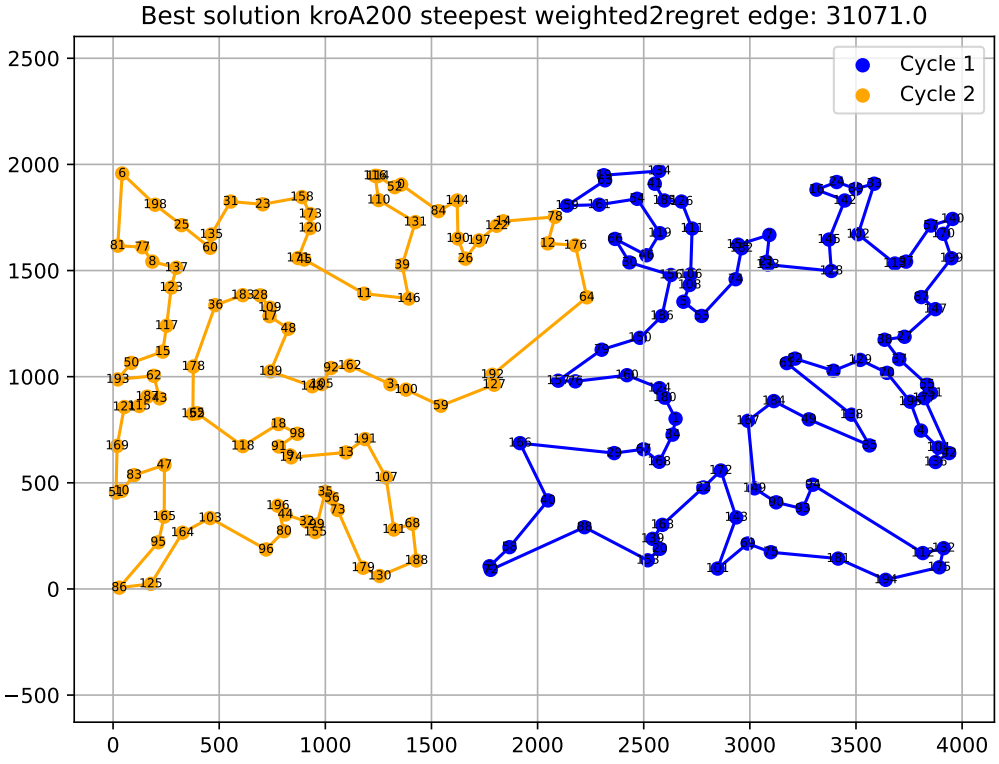

Best solution kroA200 steepest rand edge: 36571.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

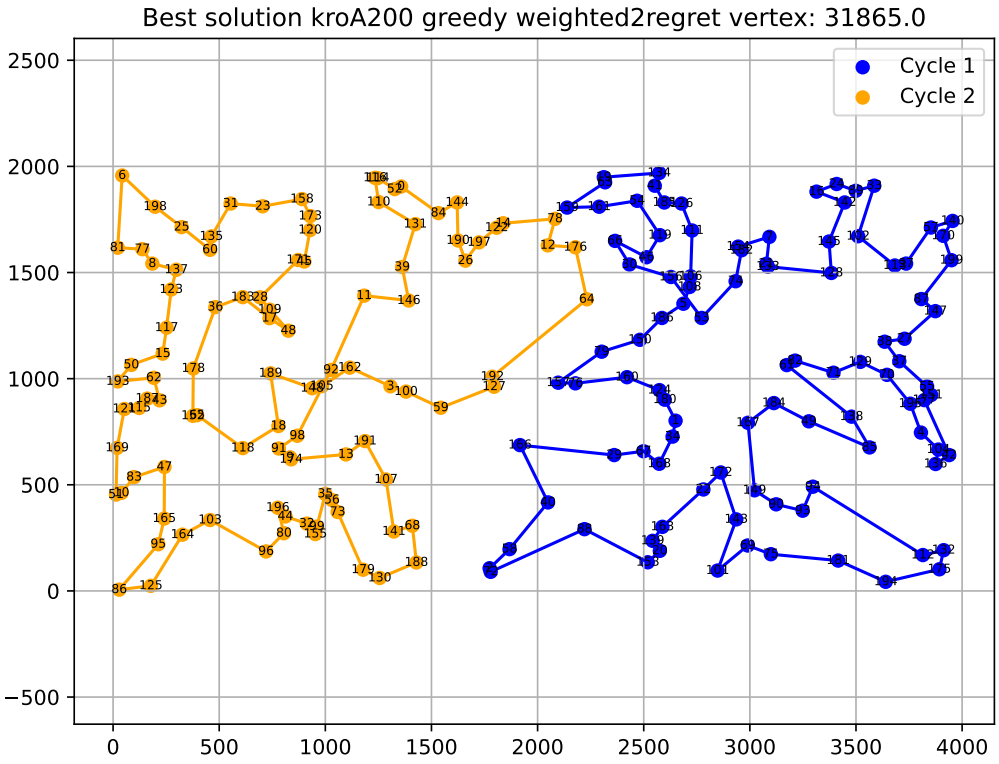

Best solution kroA200 greedy rand vertex: 63609.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

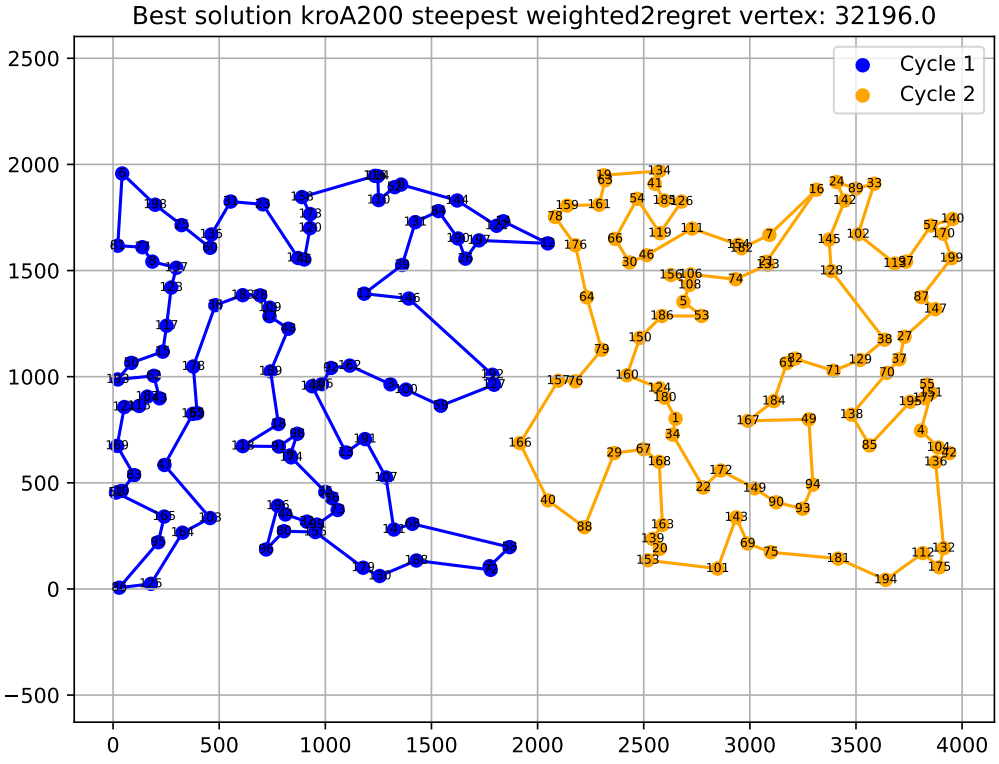

Best solution kroA200 steepest rand vertex: 67036.0
Cycle 1
Cycle 2

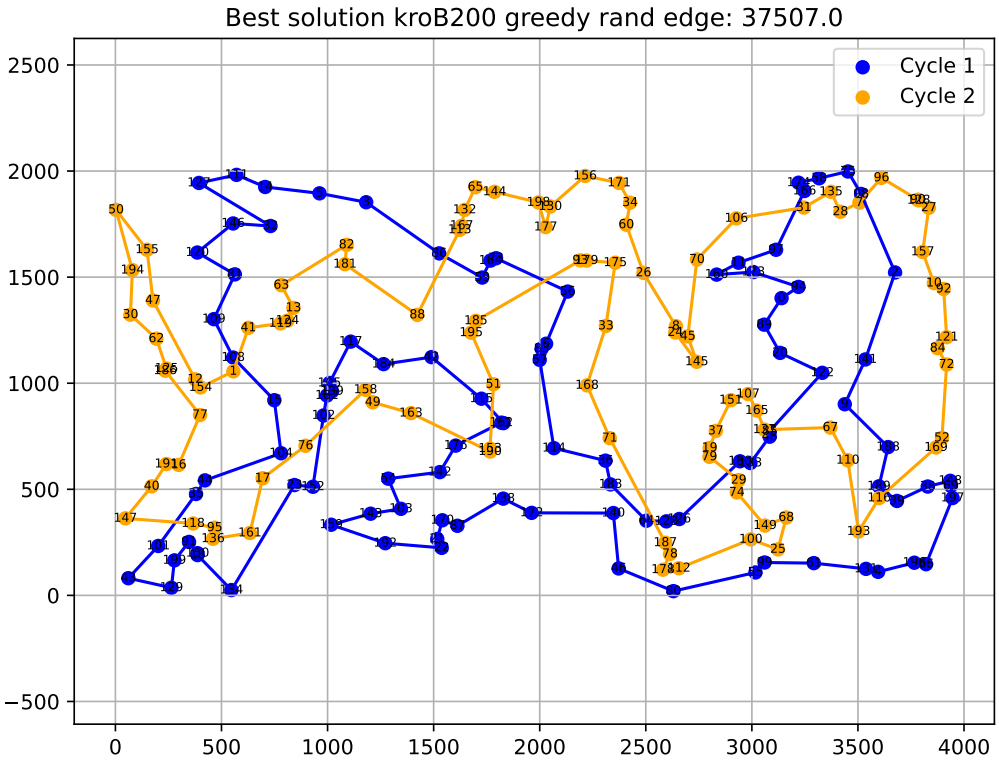

Best solution kroA200 steepest rand vertex: 67036.0
Cycle 1
Cycle 2

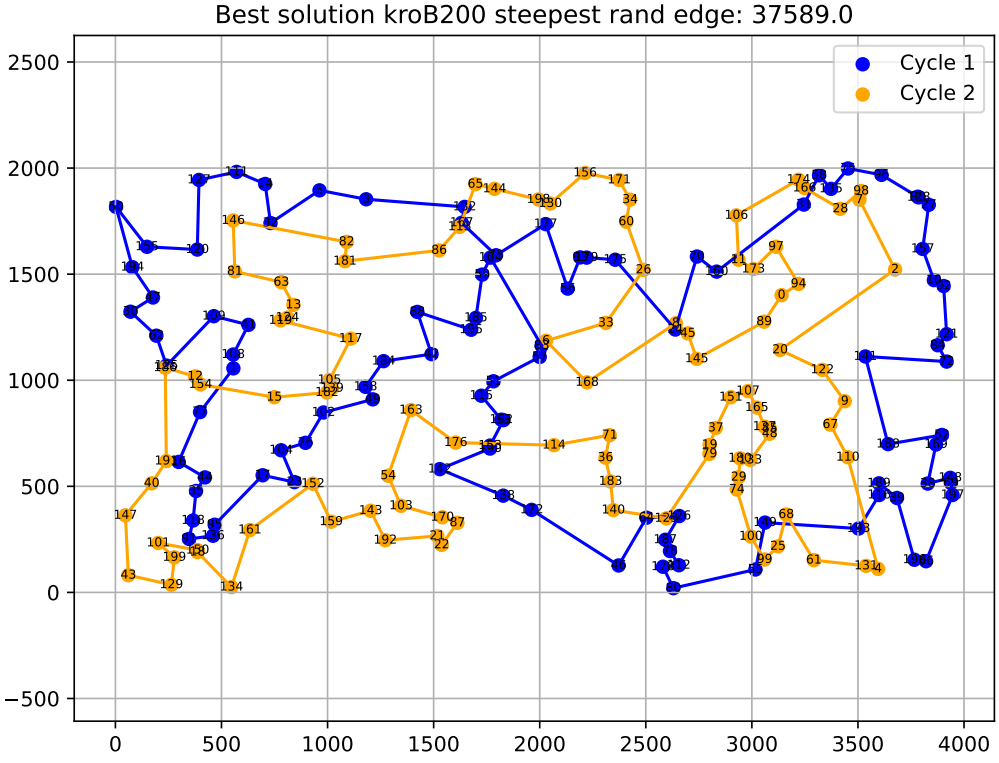

Best solution kroA200 greedy weighted2regret edge: 31748.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

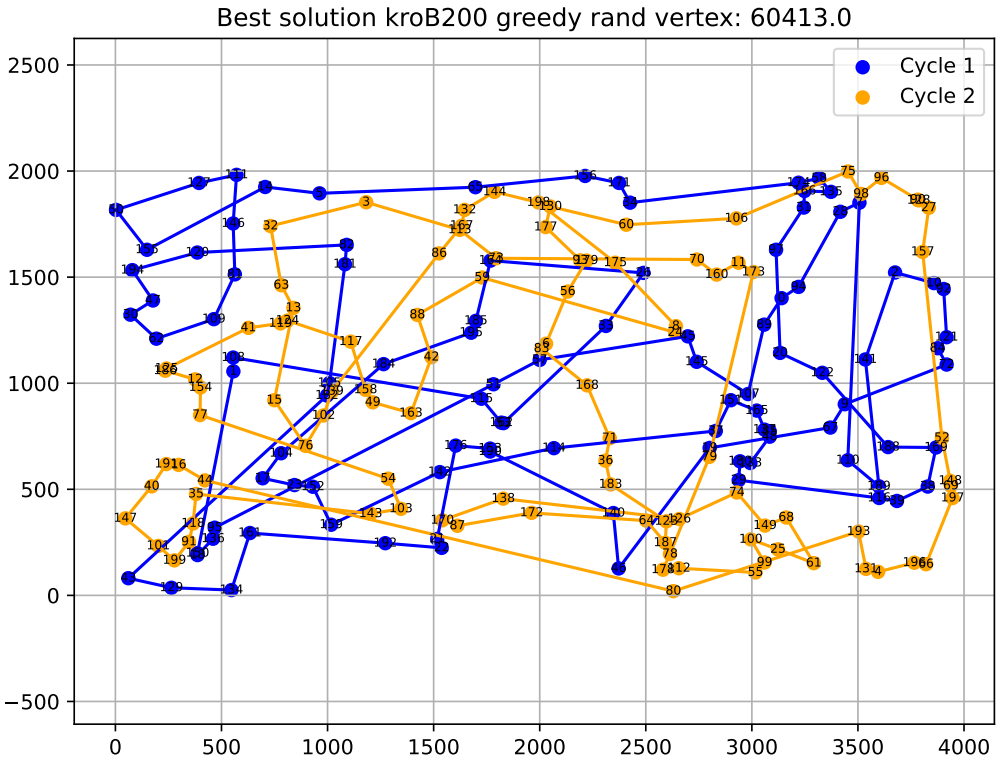

Best solution kroA200 steepest weighted2regret edge: 31071.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

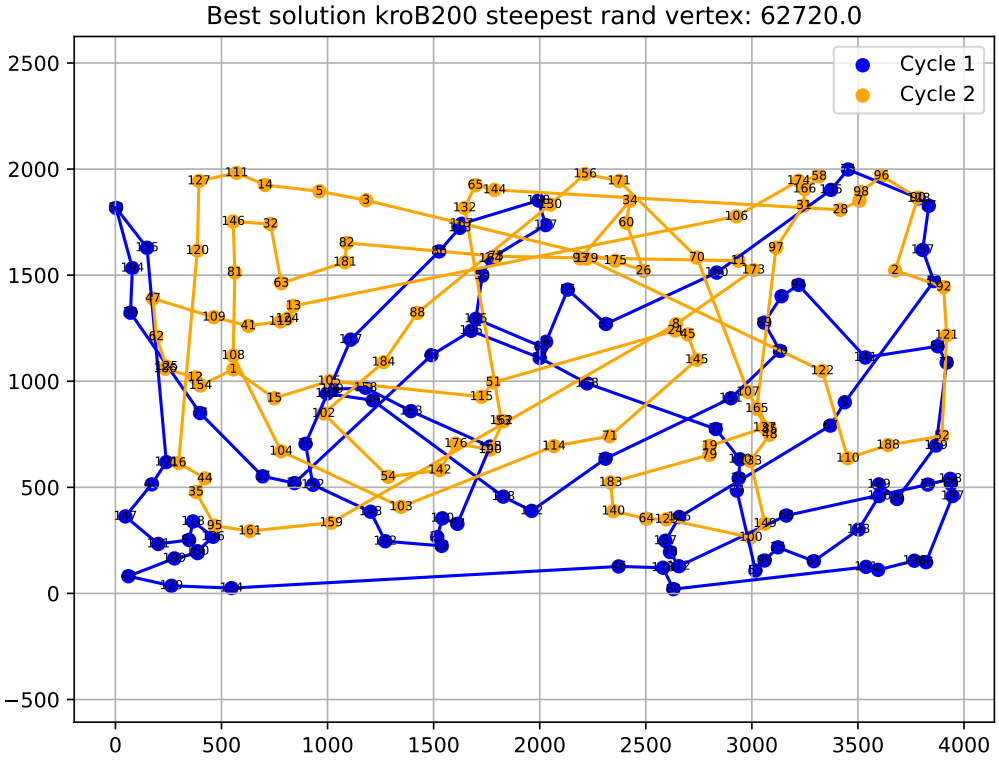

Best solution kroA200 greedy weighted2regret vertex: 31865.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

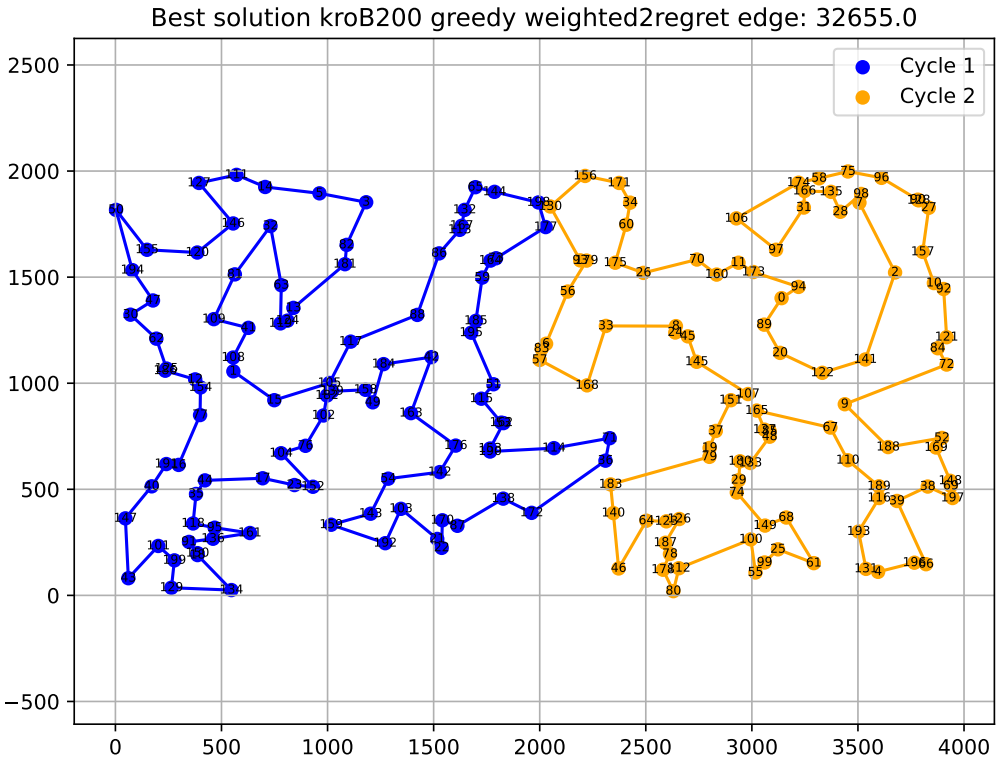

Best solution kroA200 steepest weighted2regret vertex: 32196.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

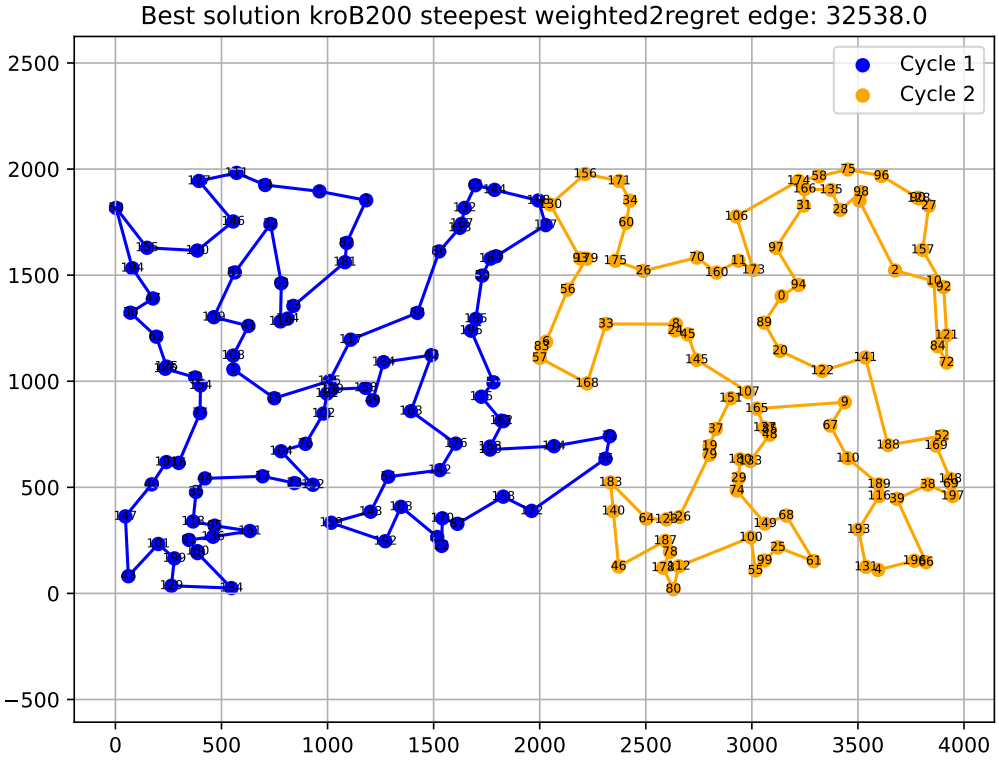

Best solution kroB200 greedy rand edge: 37507.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

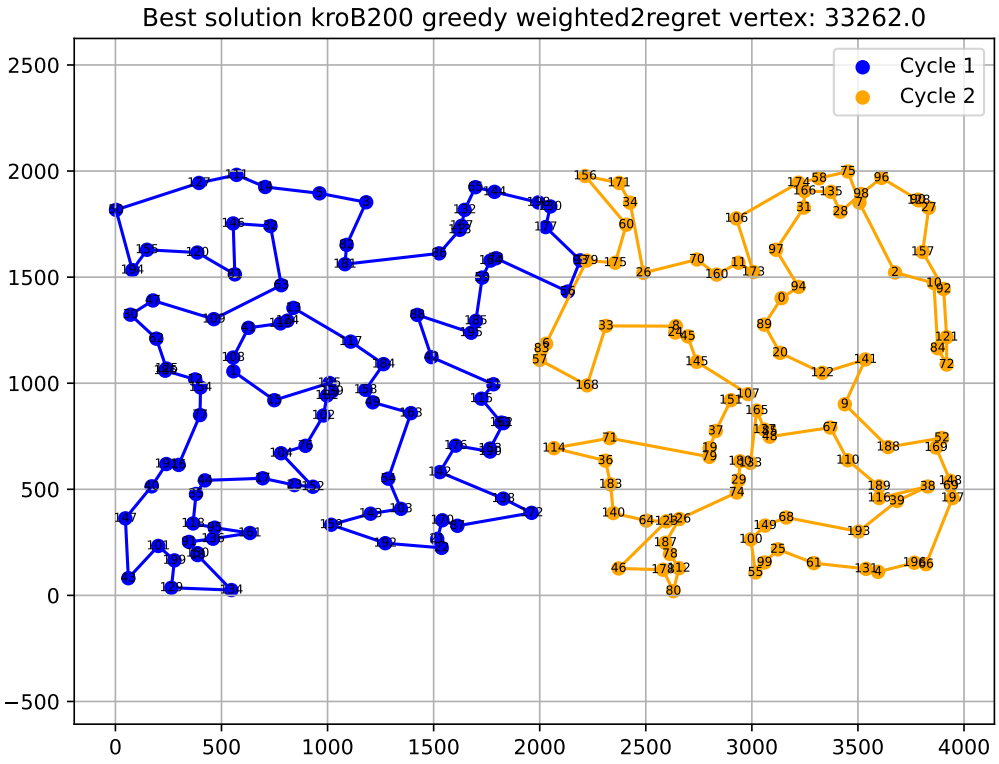

Best solution kroB200 steepest rand edge: 37589.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

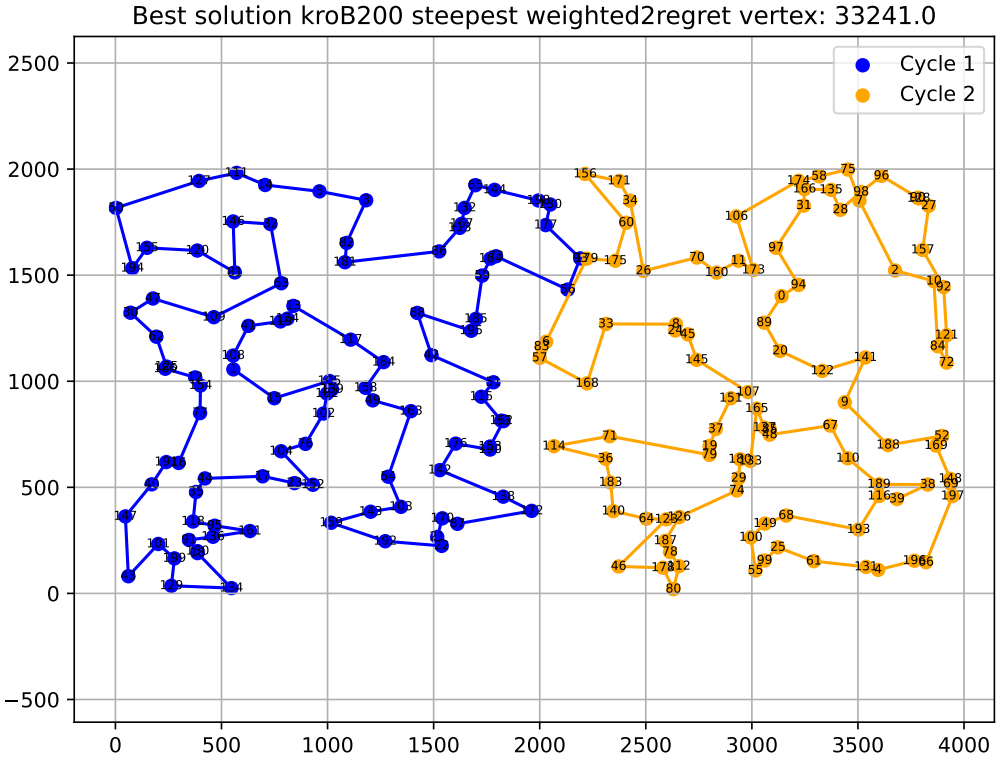

Best solution kroB200 greedy rand vertex: 60413.0
Cycle 1
Cycle 2

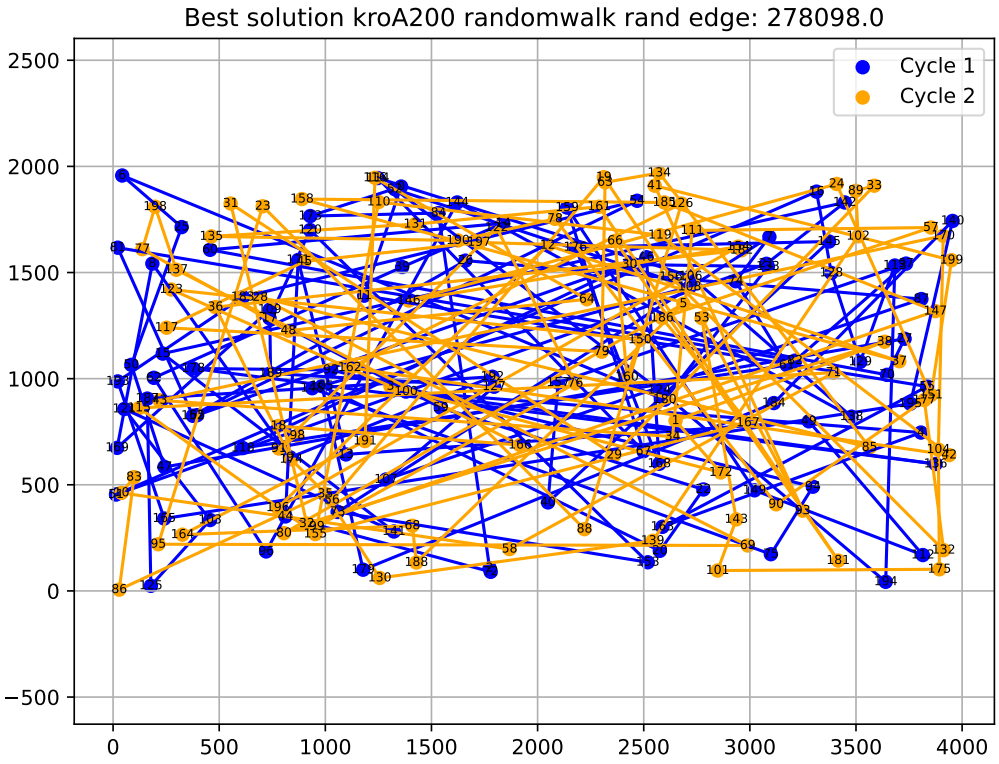

Best solution kroB200 steepest rand vertex: 62720.0
Cycle 1
Cycle 2

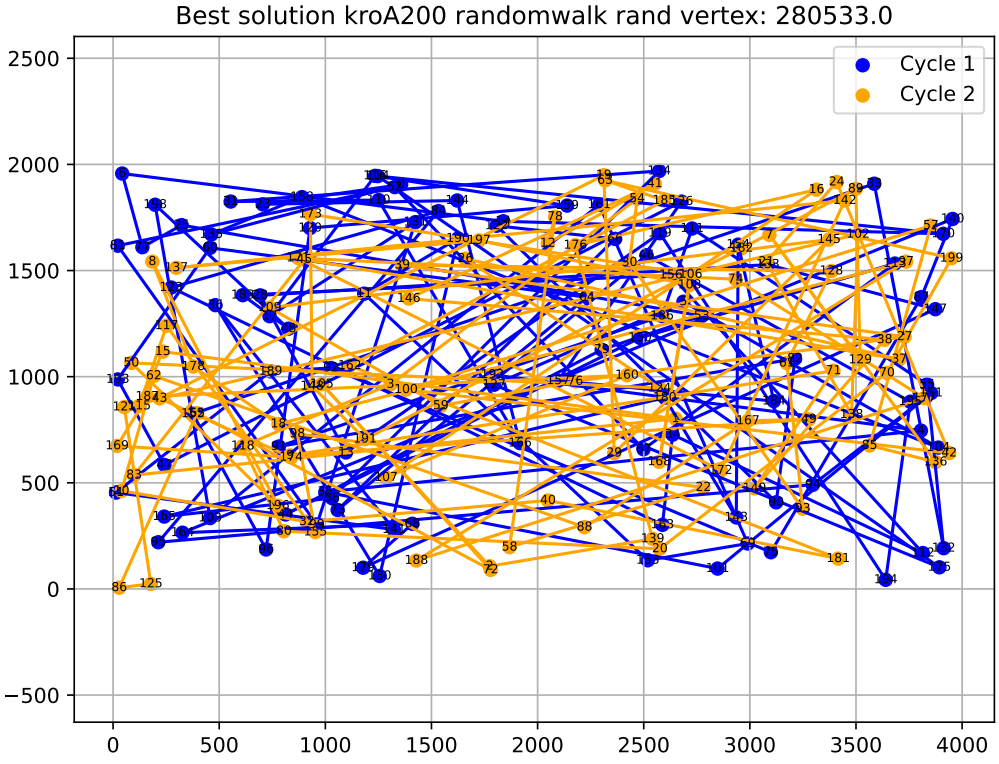

Best solution kroB200 greedy weighted2regret edge: 32655.0
Cycle 1
Cycle 2

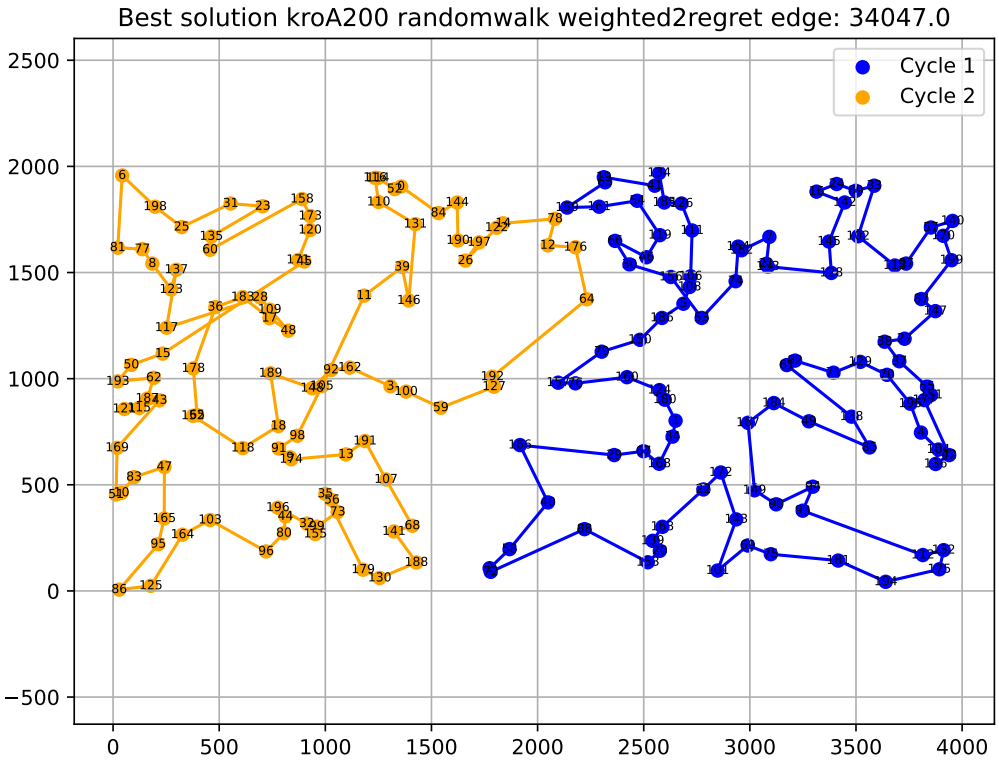

Best solution kroB200 steepest weighted2regret edge: 32538.0
Cycle 1
Cycle 2

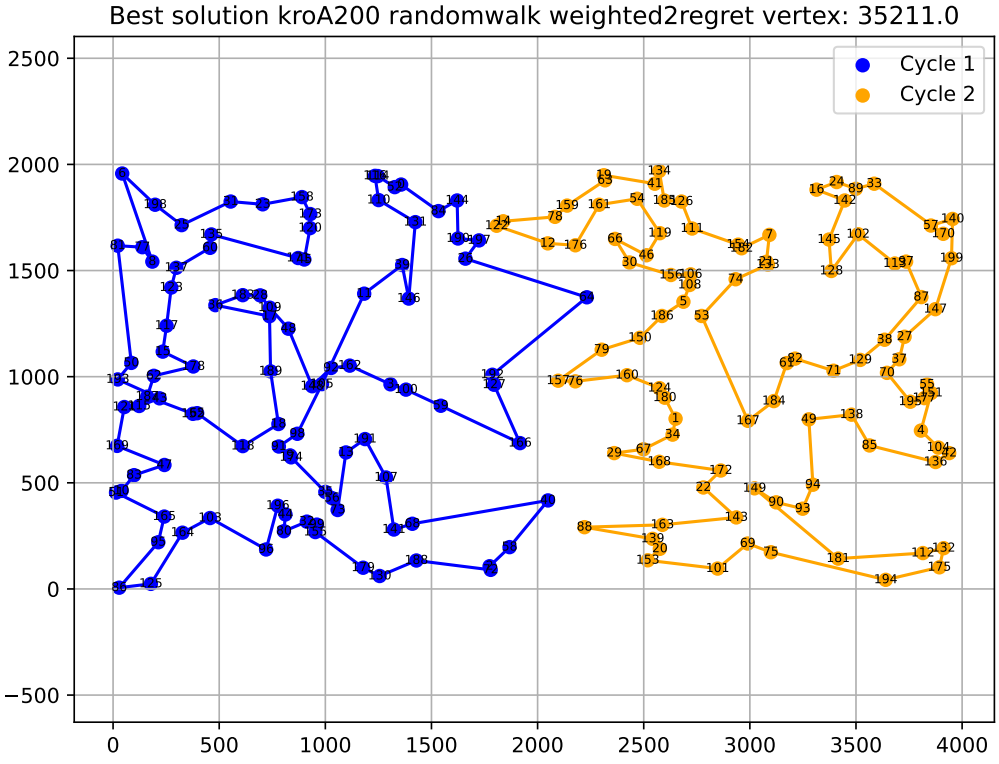

Best solution kroB200 greedy weighted2regret vertex: 33262.0
Cycle 1
Cycle 2

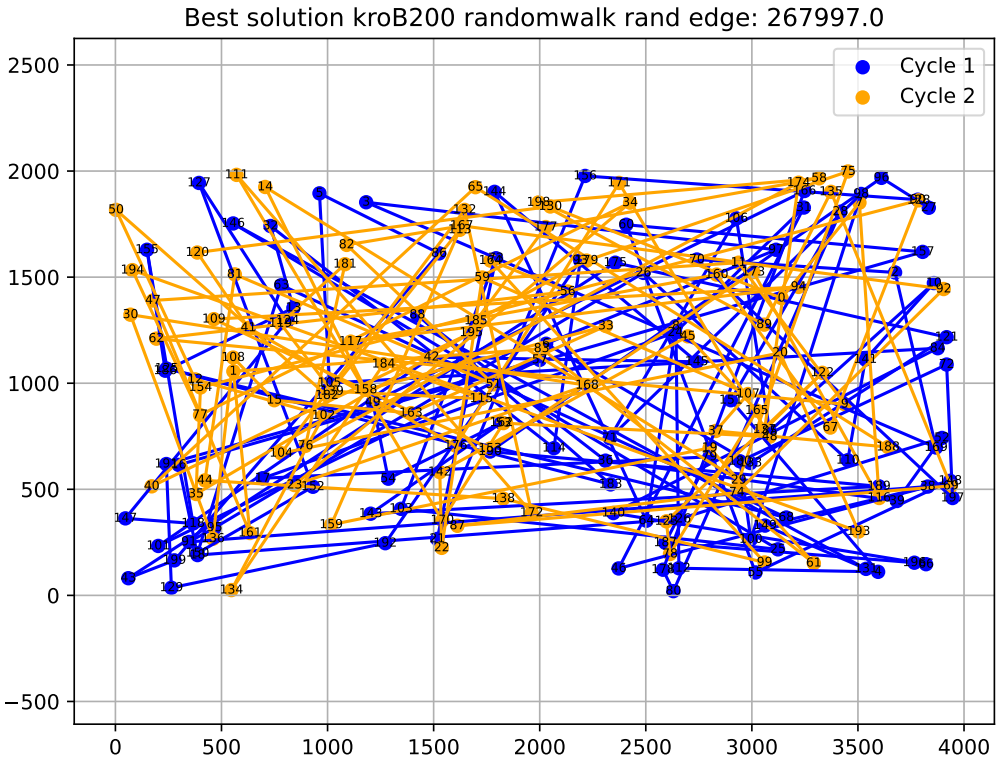

Best solution kroB200 steepest weighted2regret vertex: 33241.0
Cycle 1
Cycle 2

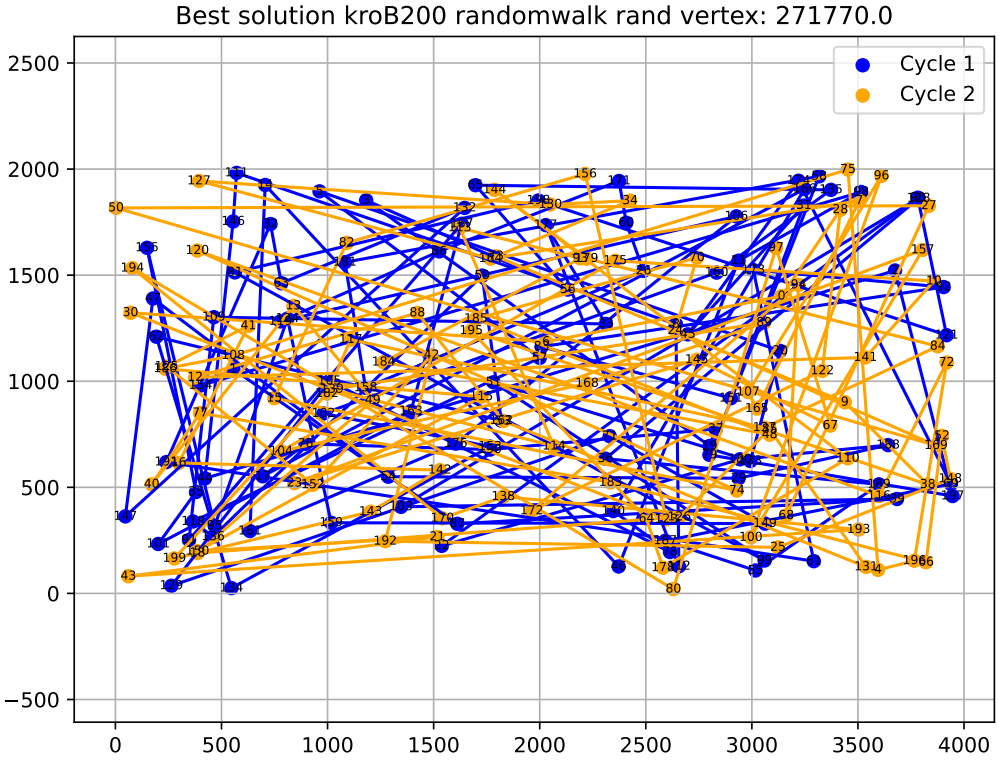

Best solution kroA200 randomwalk rand edge: 278098.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

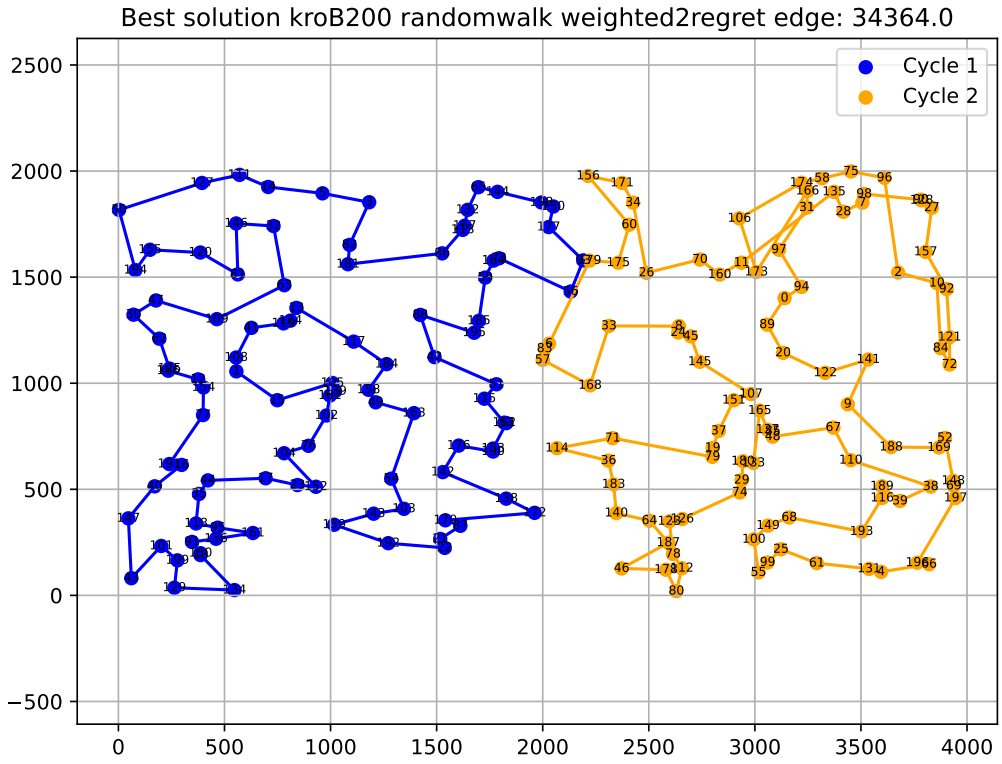

Best solution kroA200 randomwalk rand vertex: 280533.0
Cycle 1
Cycle 2
2500
2000
1500
1000
500
0
−500
0
500
1000
1500
2000
2500
3000
3500
4000

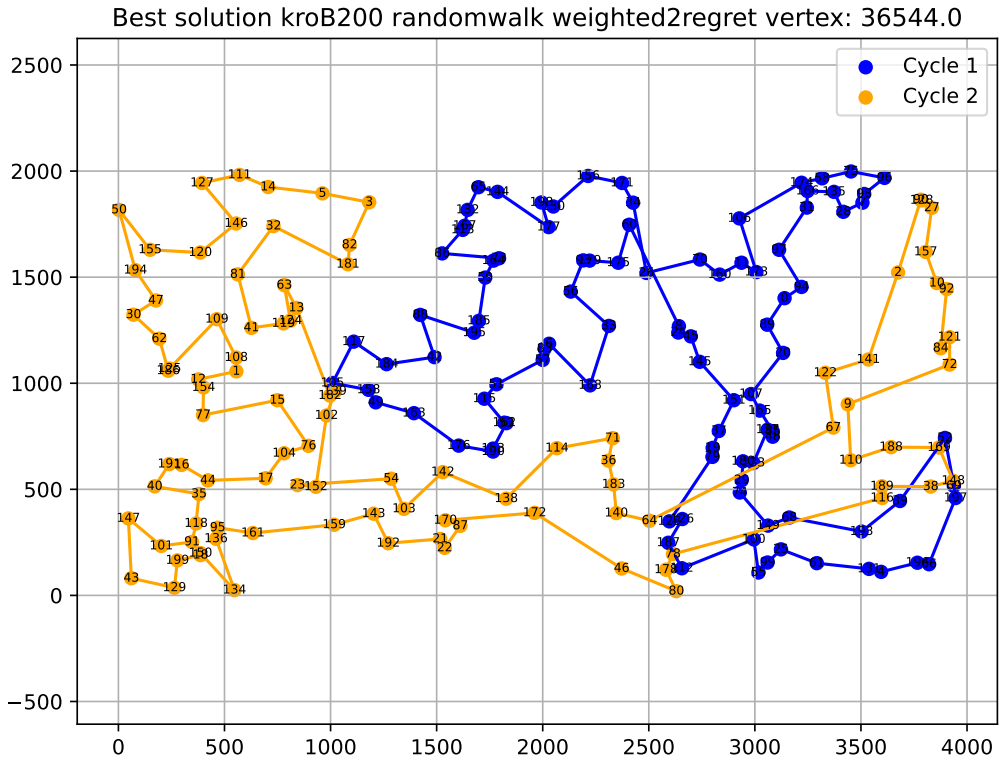

Best solution kroA200 randomwalk weighted2regret edge: 34047.0
Cycle 1
Cycle 2


Best solution kroA200 randomwalk weighted2regret vertex: 35211.0
Cycle 1
Cycle 2


Best solution kroB200 randomwalk rand edge: 267997.0
Cycle 1
Cycle 2


Best solution kroB200 randomwalk rand vertex: 271770.0
Cycle 1
Cycle 2





5 Wnioski

Algorytmy lokalnego przeszukiwania osiągają lepsze wyniki niż metody zaimplementowane w ramach poprzedniego zadania, dla rozwiązań wygenerowanych ze startem w rozwiązaniu z algorytmu ważonego 2-żalu, poprzez odnalezienie lokalnego optimum w zaimplementowanym sąsiedztwie. Gdy jednak zaczynamy od losowo wygenerowanego rozwiązania, zarówno zachłanna, jak i stroma zwykle zatrzymują się w lokalnym optimum o znacznie gorszej jakości, przy czym sąsiedztwo oparte na wymianie krawędzi pomiędzy trasami daje znacznie lepsze rezultaty niż wymiana wierzchołków. Ten typ

sąsiedztwa sprawdza się także lepiej przy optymalizacji już dobrych rozwiązań uzyskanych metodą ważonego 2-żalu.

Wersja stroma lokalnego przeszukiwania osiąga najniższe (najlepsze) wartości funkcji celu dla obu instancji, jednak średnio nie wydaje się przewyższać wersji zachłannej, zwłaszcza gdy startujemy z losowych rozwiązań – tam zauważalnie odstaje. Random walk daje słabe rezultaty, nawet nie będąc w stanie poprawić wyniku ważonego 2-żalu w porównaniu do wcześniejszych eksperymentów.

Jeśli chodzi o czas działania, wersja storma działa znacznie dłużej niż jej odpowiednik zachłanny przy starcie z losowego rozwiązania. Wynika to z konieczności przeszukania całego sąsiedztwa, a znalezienie dowolnego lepszego rozwiązania dla algorytmu zachłannego jest znacznie szybsze. Dla rozwiązań startowych opartych na ważonym 2-żalu różnica ta jest mniejsza, przy czym wersja zachłanna wciąż jest nieco szybszy niż wersja stroma. Dzieje się tak, ponieważ zaczynamy już z dobrym rozwiązaniem i musimy przeszukać większą część sąsiedztwa, by znaleźć ruch ulepszający.

Metoda wymiany krawędzi generuje rozwiązania, w których nie ma przecięć w obrębie cyklu.

6 Kod programu

[link do repozytorium na GitHubie].