Oleksandr Oksanich, AIS ID: 122480

Umelá inteligencia

Zadanie č. 2 — Strojové učenie sa (problém obchodného cestujúceho)

Obsah

Riešený problem

Obchodný cestujúci má navštíviť viacero miest. V jeho záujme je minimalizovať cestovné náklady a cena prepravy je úmerná dĺžke cesty, preto snaží sa nájsť najkratšiu možnú cestu tak, aby každé mesto navštívil práve raz. Keďže sa nakoniec musí vrátiť do mesta z ktorého vychádza, jeho cesta je uzavretá krivka.

Je daných aspoň 20 miest (20–40) a každé má určené súradnice ako celé čísla X a Y. Tieto súradnice sú náhodne vygenerované (rozmer mapy môže byť napríklad 200 * 200 km). Cena cesty medzi dvoma mestami zodpovedá Euklidovej vzdialenosti. Celková dĺžka trasy je daná nejakou permutáciou miest. Cieľom je nájsť takú permutáciu (poradie), ktorá bude mať celkovú vzdialenosť čo najmenšiu.

Na riešenie problému treba použiť algoritmus zakázaného prehľadávania (tabu search) a algoritmus simulovaného žíhania (simulated annealing).

Opis riešenia

Pre túto implementáciu som použil programovací jazyk Java (verzie 17) a vývojové prostredie IntelliJ IDEA (2023.3 EAP).

Reprezentácia údajov

Problém je reprezentovaný zoznamom inštancií triedy City, ktorá obsahuje súradnice každého mesta ((0, 0) až (200, 200)), v nejakom poradí. Počet miest je 20 až 40 a je náhodne generovaný.

Výstup je reprezentovaný postupnosťou tých istých miest v poradí, ktoré by malo spôsobiť čo najkratšiu cestu, a ich indexmi v pôvodnom zozname.

Použitý algoritmus

Nasledovníci sú generované ako vektory, v ktorých je vymenené poradie náhodnej dvojice susedných uzlov:

```
private static List<City> swapNeighbors(List<City> currentSolution) {
    var neighbor = new ArrayList<>(currentSolution);

    var first = new Random().nextInt(neighbor.size());
    var second = (first + 1) % neighbor.size();

    Collections.swap(neighbor, first, second);

    return neighbor;
}
```

Na generovanie počiatočného riešenia som použil algoritmus hľadania najbližšieho suseda, pri ktorom obchodný cestujúci začína v náhodnom meste a opakovane navštevuje najbližšie mestá, kým nenavštívi všetky:

```
private static List<City> initialSolution(List<City> cities) {
    var unvisited = new ArrayList<>(cities);
    var current = unvisited.remove(new Random().nextInt(unvisited.size()));

    var solution = new ArrayList<City>();

    solution.add(current);

    while (!unvisited.isEmpty()) {
        City nearest = null;

        var minDistance = Double.MAX_VALUE;

        for (var city : unvisited) {
            var distance = city.euclidDistance(current);

            if (distance < minDistance) {
                minDistance = distance;
                nearest = city;
            }
        }

        if (nearest != null) {
            unvisited.remove(nearest);
            solution.add(nearest);

            current = nearest;
        }
    }

    return solution;
}
```

Zakázané prehľadávanie

Prvým algoritmom, ktorý som mal použiť, je zakázané prehľadávanie, ktorého podstata v mojej implementácii je nasledovná: hľadanie začína počiatočným riešením, ktoré je získané podľa algoritmu hľadania najbližšieho suseda, ako už som spomenul. Celková vzdialenosť trasy medzi všetkými mestami sa používa na porovnanie, o

koľko je jedno riešenie lepšie ako iné. Aby sa zabránilo zacykleniu a uviaznutiu v lokálnom optime, riešenie sa pridá do zoznamu zakázaných riešení (tabu list, ktorého dĺžka je taktiež obmedzená pri volaní funkcie), ak je pri hľadaní medzi susedmi akceptované. Ako ukončovaciu podmienku som použil limit iterácií bez zlepšenia a limit všetkých iterácií.

```
public static Result tabuSearch(
        List<City> cities,
        int maxIterations,
        int maxWithoutImprovement,
        int maxTabuStates
) {
    var time = System.currentTimeMillis();

    var iterations = 0;
    var withoutImprovement = 0;

    var tabu = new ArrayList<>();

    var current = initialSolution(cities);
    var best = new ArrayList<>(current);

    // The terminating conditions
    while (iterations < maxIterations && withoutImprovement <
maxWithoutImprovement) {
        var neighborhood = swapNeighbors(current);

        // Aspiration criteria (i.e., even a tabu state is allowed in case it
improves the current solution)
        if (!tabu.contains(neighborhood) || totalDistance(neighborhood) <
totalDistance(best)) {
            current = new ArrayList<>(neighborhood);

            if (totalDistance(current) < totalDistance(best)) {
                best = new ArrayList<>(current);
            }

            tabu.add(neighborhood);

            if (tabu.size() > maxTabuStates) {
                tabu.remove(0);
            }

            withoutImprovement = 0;
        } else {
            withoutImprovement++;
        }

        iterations++;
    }

    time = System.currentTimeMillis() - time;
```

```
    return new Result(cities, best, iterations, time);
}
```

Simulované žíhanie

Druhý algoritmus, ktorý som využil v tejto implementácii, je tzv. simulované žíhanie (resp. simulated annealing), ktorého princíp je taký, že uvedená teplota žíhania reprezentuje pravdepodobnosť, že za nasledovníka bude zvolený taký stav, ktorý nie je najlepší z okolia. Táto teplota postupne klesá kvôli hodnote premennej "coolingRate", až kým nedosiahne hodnotu premennej "minTemperature", a vtedy sa vyhľadávanie skončí.

```
public static Result simulatedAnnealing(
        List<City> cities,
        double initialTemperature,
        double minTemperature,
        double coolingRate
) {
    var time = System.currentTimeMillis();
    var iterations = 0;

    var current = initialSolution(cities);
    var best = new ArrayList<>(current);

    while (initialTemperature > minTemperature) {
        var neighborhood = swapNeighbors(current);

        // If negative, then `neighborhood` is a better solution than the current
one
        var delta = totalDistance(neighborhood) - totalDistance(current);

        // If the new solution isn't better, then the probability of accepting the
worse solution is checked
        if (delta < 0 || Math.random() < Math.exp(-delta / initialTemperature)) {
            current = neighborhood;

            if (totalDistance(neighborhood) < totalDistance(best)) {
                best = new ArrayList<>(current);
            }
        }

        // The temperature is decreased by ((1 - coolingRate) * 100) percent
        initialTemperature *= 1 - coolingRate;

        iterations++;
    }

    time = System.currentTimeMillis() - time;

    return new Result(cities, best, iterations, time);
}
```

Oleksandr Oksanich, AIS ID: 122480

Zhodnotenie riešenia

Výsledky testovania

Podľa toho, čo vyžaduje zadanie, pre každý algoritmus uvádzam dva príklady riešenia pre dva rôzne počty miest: 20 a 30.

Na ručnú vizualizáciu výstupu bol použitý Desmos.

Zakázané prehľadávanie

Limit iterácií bez zlepšenia: 50

Limit všetkých iterácií: 50000

Dĺžka zoznamu zakázaných stavov: 15

1. príklad

Postupnosť miest: (41, 199), (95, 176), (147, 102), (3, 62), (47, 20), (151, 126), (106, 148), (158, 97), (80, 120), (165, 70), (54, 144), (113, 91), (180, 189), (91, 38), (5, 142), (91, 99), (2, 116), (21, 54), (120, 41), (109, 116)

Poradie miest v riešení: (95, 176), (106, 148), (109, 116), (91, 99), (113, 91), (147, 102), (158, 97), (165, 70), (120, 41), (91, 38), (47, 20), (21, 54), (3, 62), (2, 116), (5, 142), (54, 144), (80, 120), (151, 126), (180, 189), (41, 199)

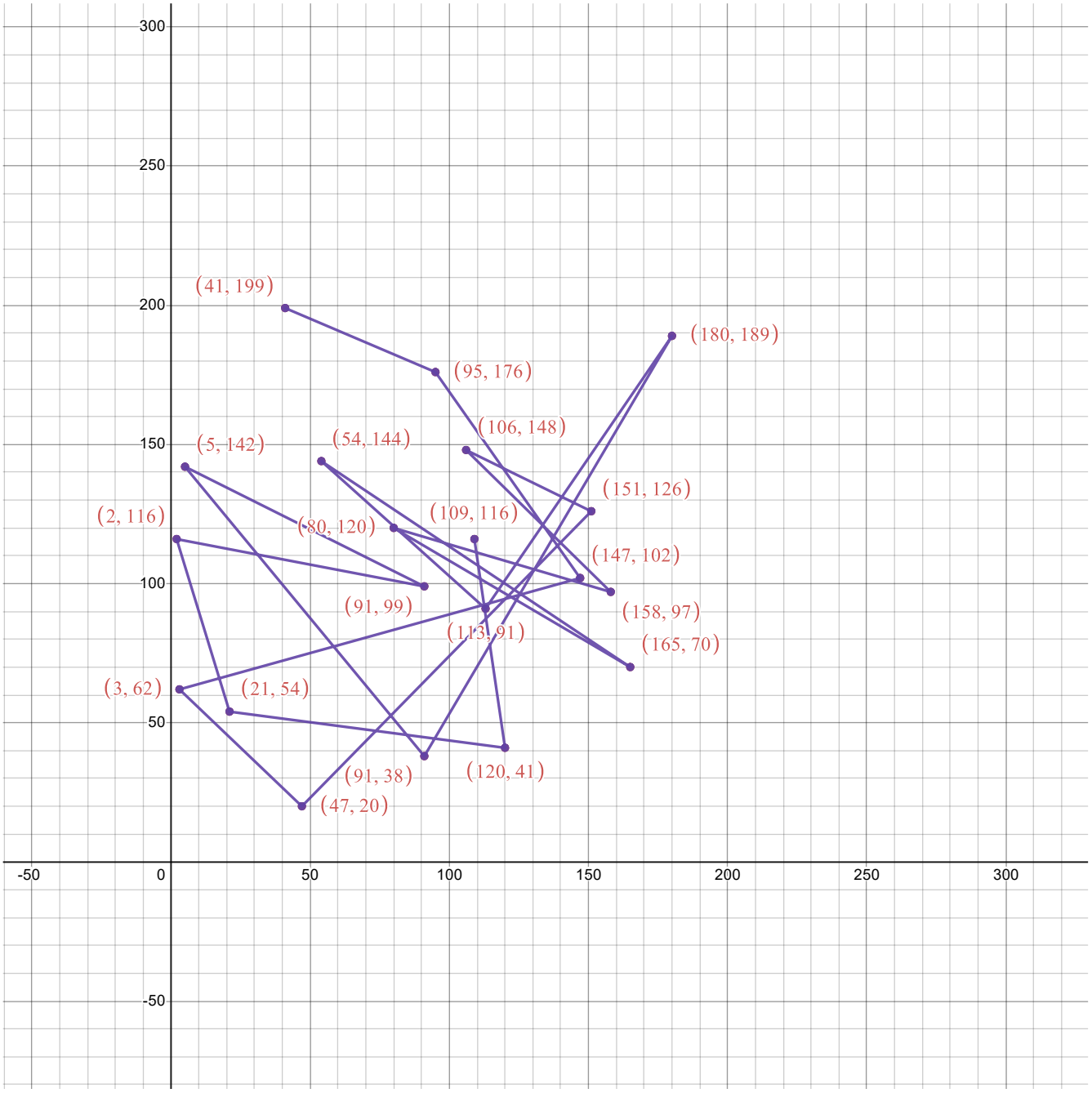
Indexy miest v riešení: [1, 6, 19, 15, 11, 2, 7, 9, 18, 13, 4, 17, 3, 16, 14, 10, 8, 5, 12, 0]

Celková vzdialenosť: 882.098221

Čas riešenia: 75 ms

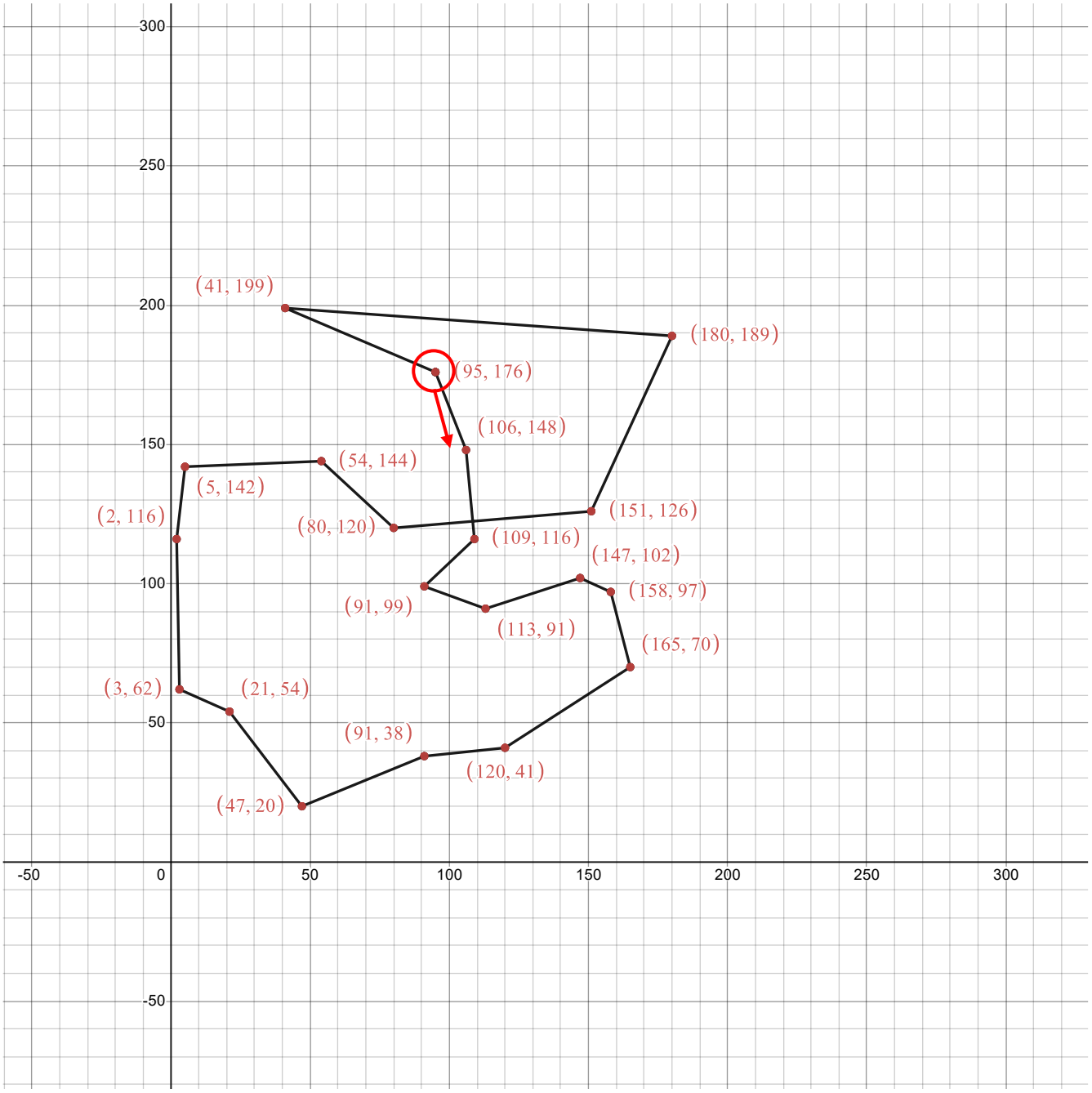
Počiatočný stav:

Oleksandr Oksanich, AIS ID: 122480



Riešenie:

Oleksandr Oksanich, AIS ID: 122480



2. príklad

Postupnosť miest: (193, 121), (7, 186), (29, 170), (91, 185), (51, 51), (77, 54), (130, 92), (64, 4), (160, 60), (126, 126), (52, 27), (79, 112), (191, 126), (167, 157), (124, 143), (11, 72), (113, 95), (52, 79), (138, 150), (55, 195), (124, 74), (47, 140), (14, 184), (97, 175), (17, 36), (84, 109), (132, 101), (137, 192), (48, 123), (67, 65)

Poradie miest v riešení: (124, 143), (138, 150), (126, 126), (132, 101), (130, 92), (113, 95), (124, 74), (160, 60), (193, 121), (191, 126), (167, 157), (137, 192), (97, 175), (91, 185), (55, 195), (29, 170), (14, 184), (7, 186), (47, 140), (48, 123), (79, 112), (84, 109), (52, 79), (67, 65), (77, 54), (51, 51), (52, 27), (64, 4), (17, 36), (11, 72)

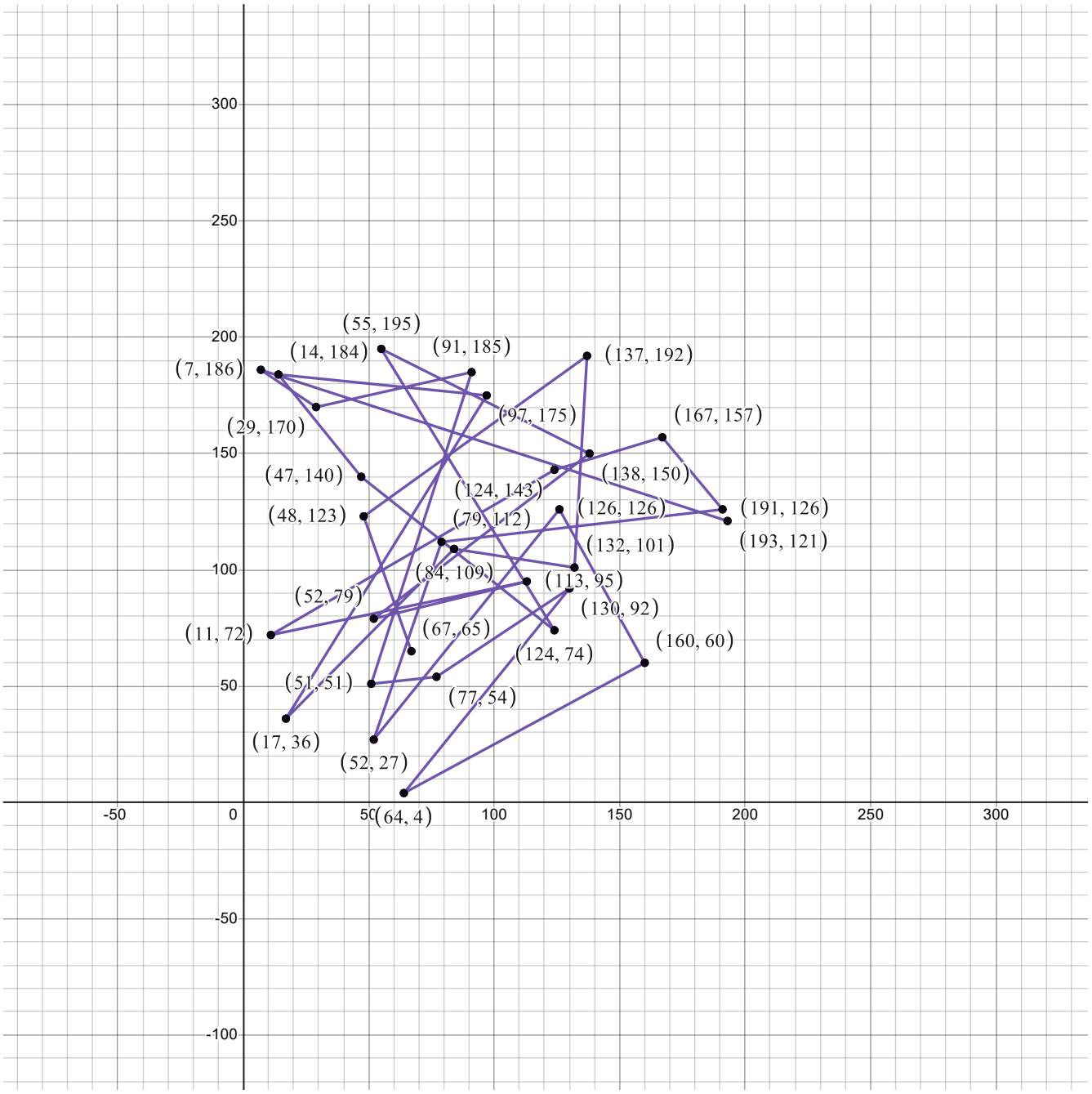
Indexy miest v riešení: [14, 18, 9, 26, 6, 16, 20, 8, 0, 12, 13, 27, 23, 3, 19, 2, 22, 1, 21, 28, 11, 25, 17, 29, 5, 4, 10, 7, 24, 15]

Celková vzdialenosť: 972.314039

Oleksandr Oksanich, AIS ID: 122480

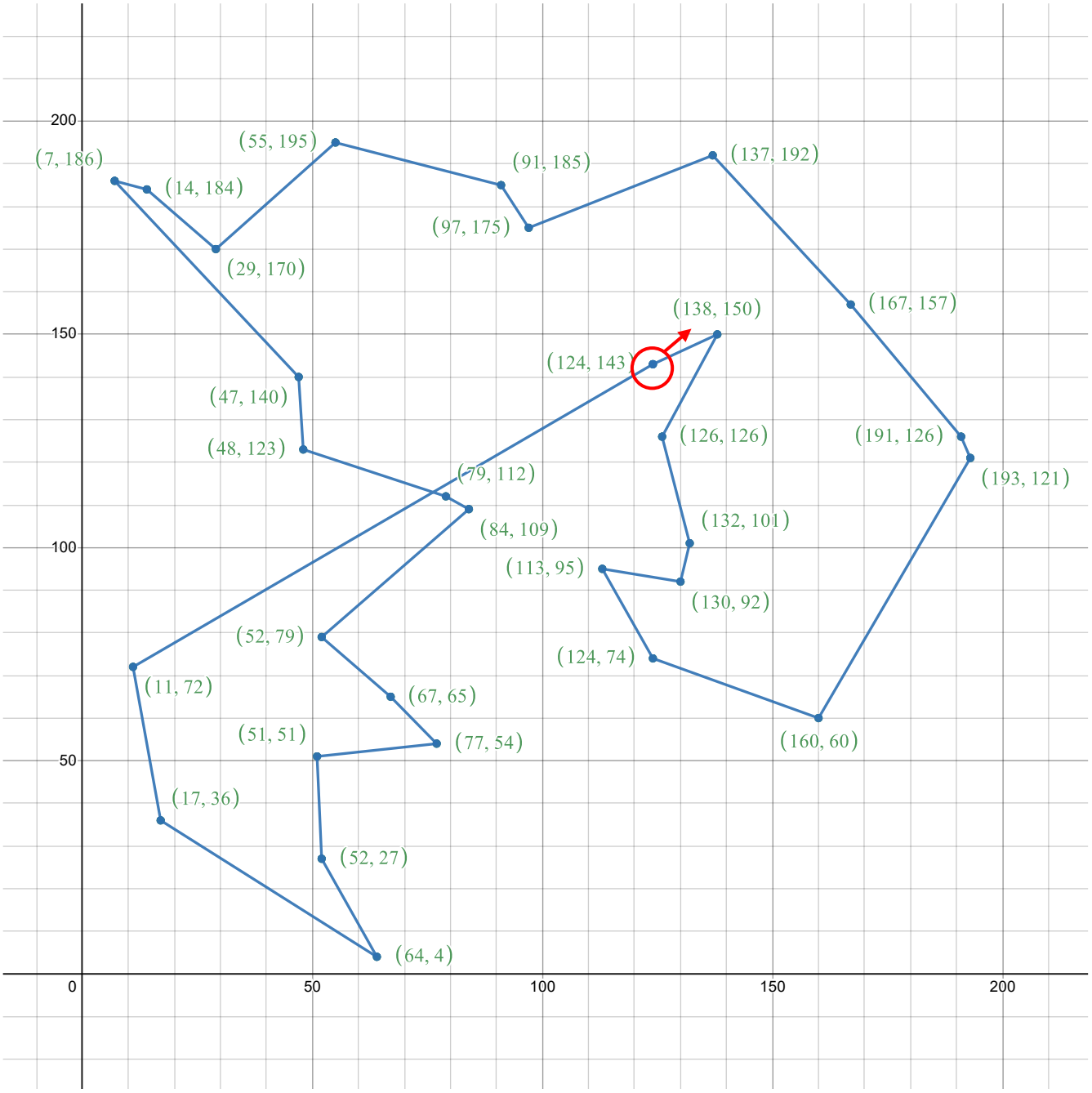
Čas riešenia: 93 ms

Počiatočný stav:



Riešenie:

Oleksandr Oksanich, AIS ID: 122480



Tabuľka pre iné počty miest

Počet miest	Čas, ms	Počet iterácií
5	25	19751
10	32	50000
15	25	50000
20	28	50000
25	28	50000
30	29	50000
35	28	50000
40	34	50000

Oleksandr Oksanich, AIS ID: 122480

Simulované žíhanie

Počiatočná teplota: 100

Minimálna teplota: 0.1

Pokles teploty: 0.05

1. príklad

Postupnosť miest: (52, 44), (70, 139), (101, 59), (43, 148), (69, 69), (26, 14), (198, 194), (61, 2), (193, 43), (7, 156), (9, 2), (73, 68), (90, 162), (97, 182), (33, 18), (141, 42), (147, 4), (133, 22), (10, 32), (137, 161)

Poradie miest v riešení: (147, 4), (133, 22), (141, 42), (101, 59), (73, 68), (69, 69), (52, 44), (61, 2), (33, 18), (26, 14), (9, 2), (10, 32), (7, 156), (43, 148), (70, 139), (90, 162), (97, 182), (137, 161), (198, 194), (193, 43)

Indexy miest v riešení: [16, 17, 15, 2, 11, 4, 0, 7, 14, 5, 10, 18, 9, 3, 1, 12, 13, 19, 6, 8]

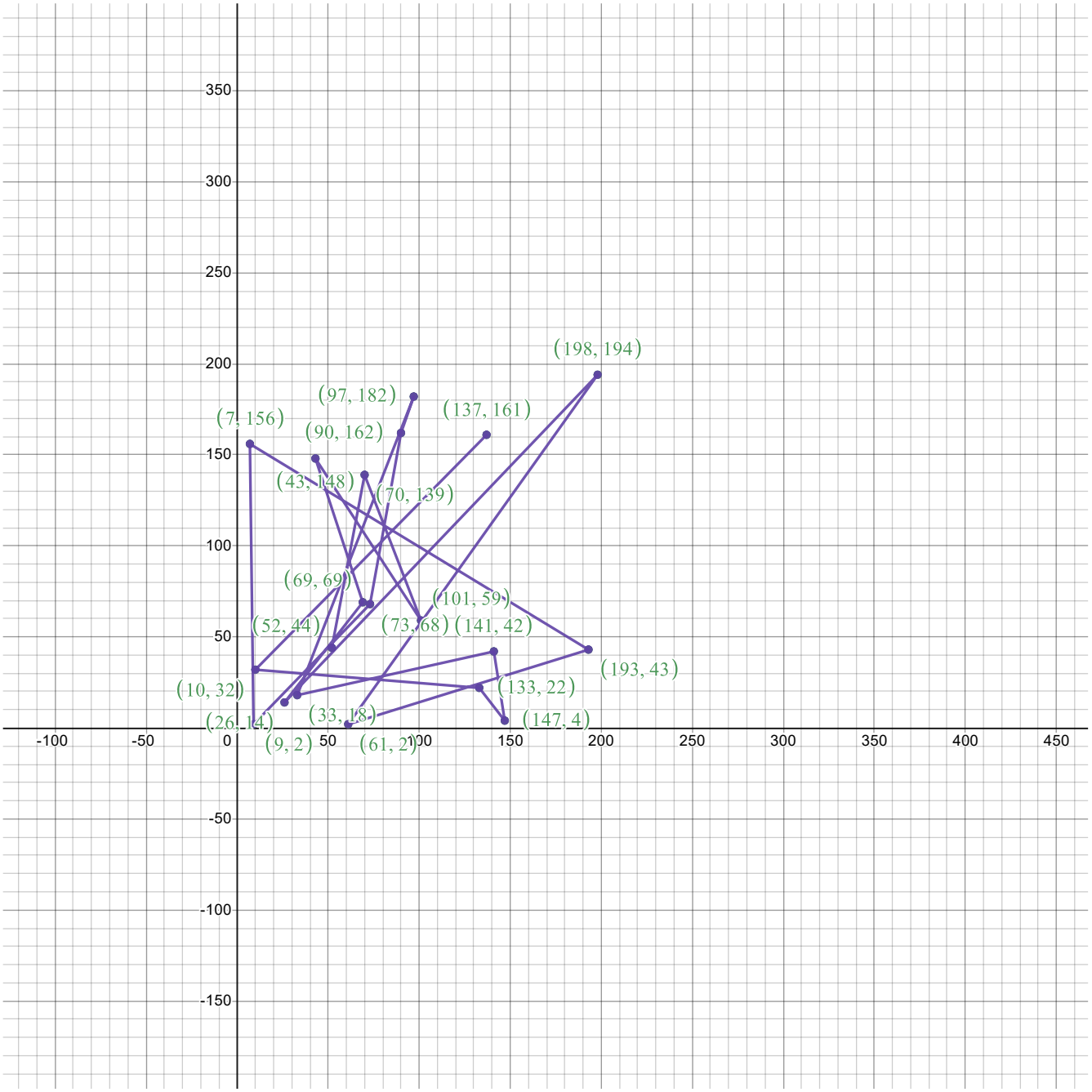
Celková vzdialenosť: 852.629261

Čas riešenia: 26 ms

Počet iterácií: 135

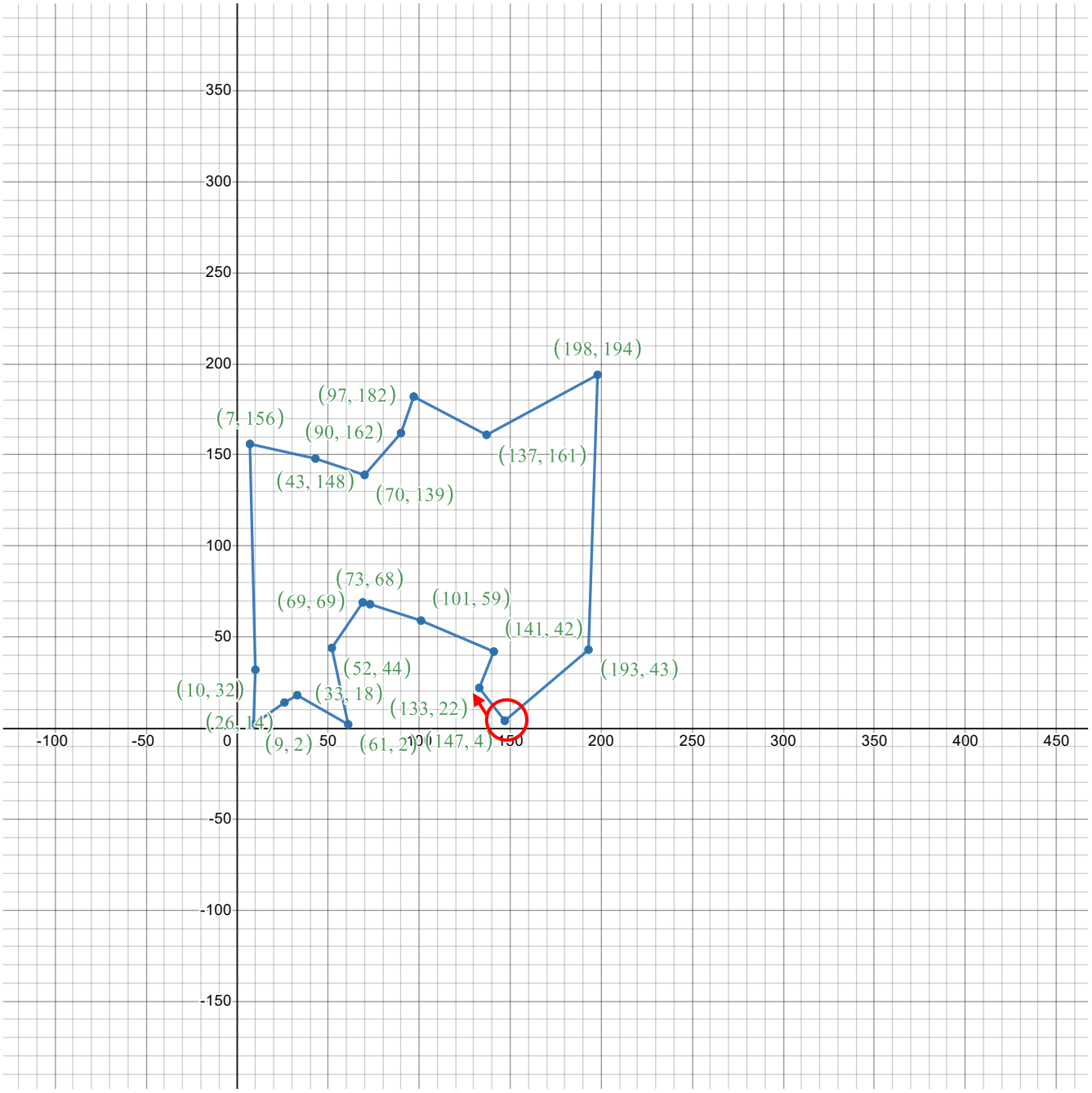
Počiatočný stav:

Oleksandr Oksanich, AIS ID: 122480



Riešenie:

Oleksandr Oksanich, AIS ID: 122480



2. príklad

Postupnosť miest: (8, 163), (153, 179), (61, 120), (15, 93), (15, 194), (169, 77), (139, 184), (20, 141), (159, 61), (54, 10), (83, 85), (37, 53), (166, 67), (134, 178), (162, 162), (77, 28), (129, 93), (113, 174), (26, 175), (163, 60), (138, 14), (105, 40), (102, 48), (142, 115), (173, 18), (193, 171), (38, 160), (183, 12), (66, 36), (49, 81)

Poradie miest v riešení: (113, 174), (134, 178), (139, 184), (153, 179), (162, 162), (193, 171), (142, 115), (129, 93), (169, 77), (166, 67), (163, 60), (159, 61), (173, 18), (183, 12), (138, 14), (105, 40), (102, 48), (77, 28), (66, 36), (54, 10), (37, 53), (49, 81), (83, 85), (61, 120), (20, 141), (8, 163), (26, 175), (38, 160), (15, 194), (15, 93)

Indexy miest v riešení: [17, 13, 6, 1, 14, 25, 23, 16, 5, 12, 19, 8, 24, 27, 20, 21, 22, 15, 28, 9, 11, 29, 10, 2, 7, 0, 18, 26, 4, 3]

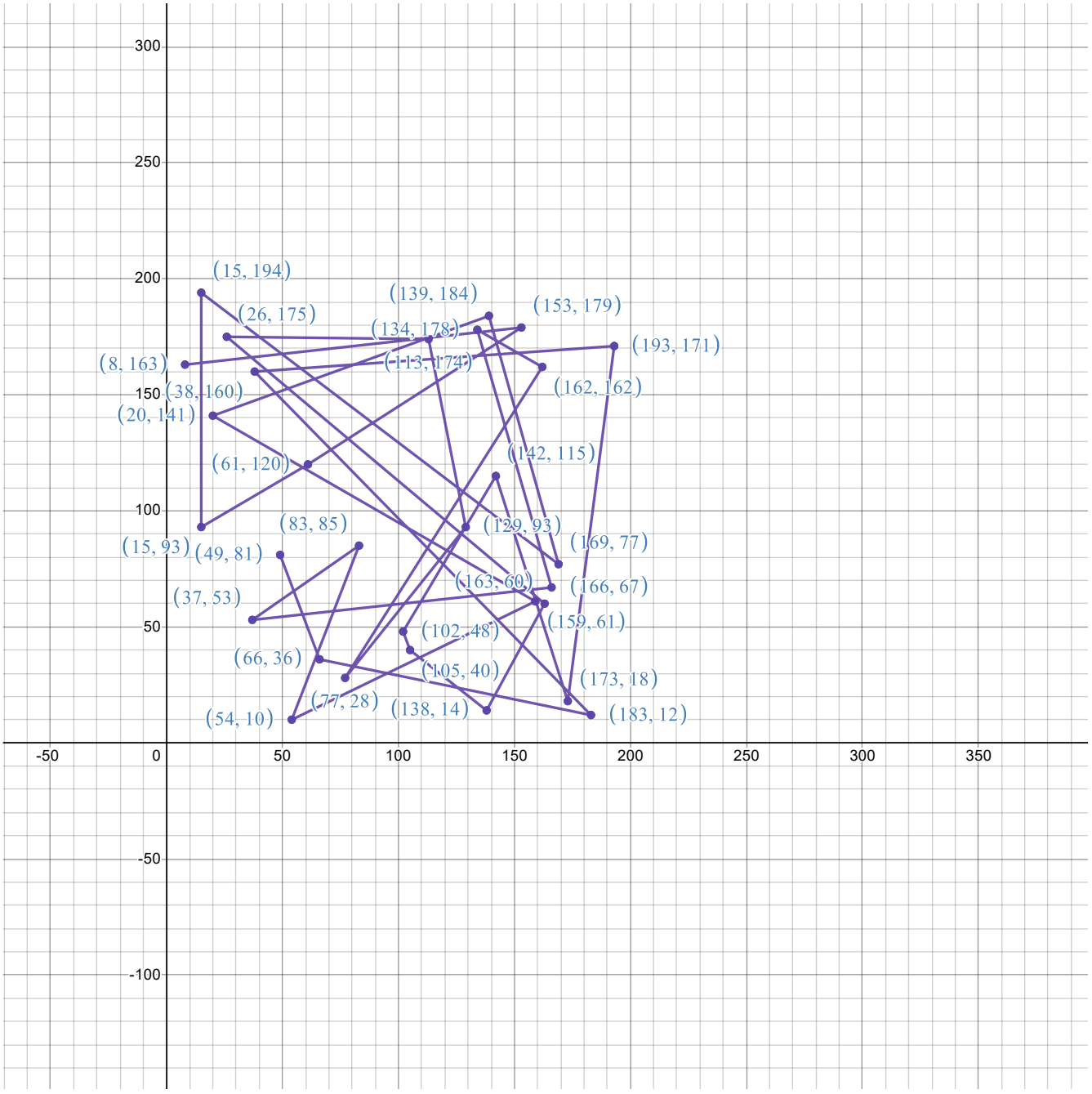
Celková vzdialenosť: 1022.297611

Oleksandr Oksanich, AIS ID: 122480

Čas riešenia: 28 ms

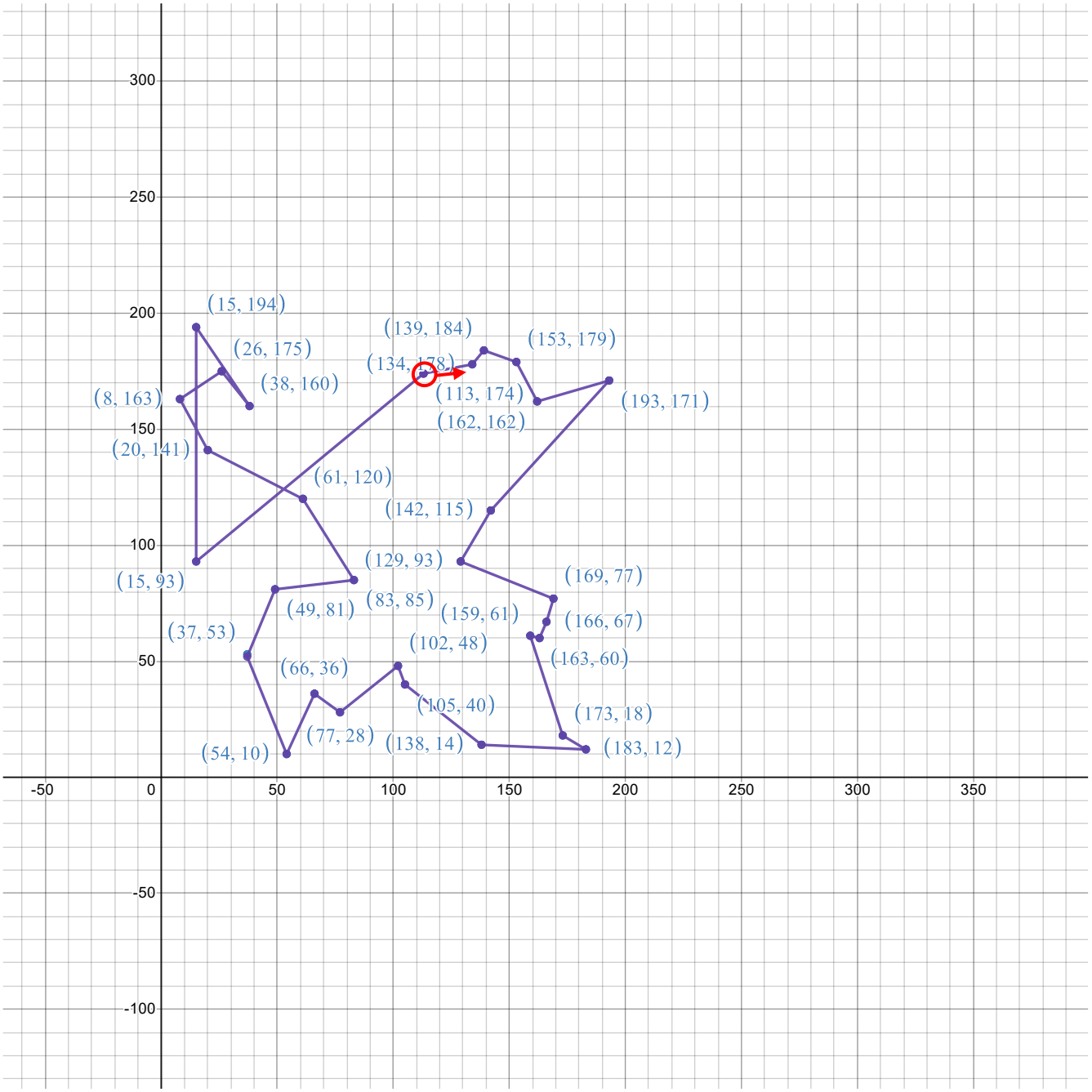
Počet iterácií: 135

Počiatočný stav:



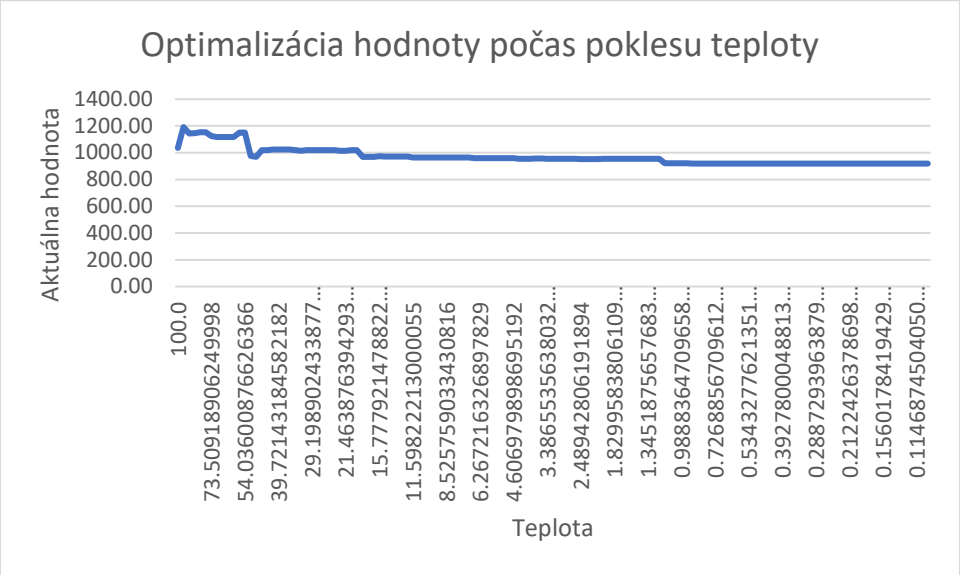
Riešenie:

Oleksandr Oksanich, AIS ID: 122480



Oleksandr Oksanich, AIS ID: 122480

Korelácia teploty a dĺžky riešenia (20 miest)



Záver

Takže som implementoval riesenie pre problém obchodného cestujúceho pomocou dvoch algoritmov: zakázané prehľadávanie a simulované žíhanie. V závislosti od zadaných parametrov algoritmus simulated annealing vyžaduje menší počet iterácií, avšak nezaručuje lepšie riešenie, ako tabu search. Samotné parametre pre každý z algoritmov (buď dĺžka zakázaných stavov zoznamu alebo teplota žíhania) taktiež ovplyvňujú efektivitu a rýchlosť možného riešenia, a preto je dôležité nájsť a zvoliť optimálne hodnoty pre tieto parametre.