

Slovenská Technická Univerzita v Bratislave
Fakulta informatiky a informačných technológií

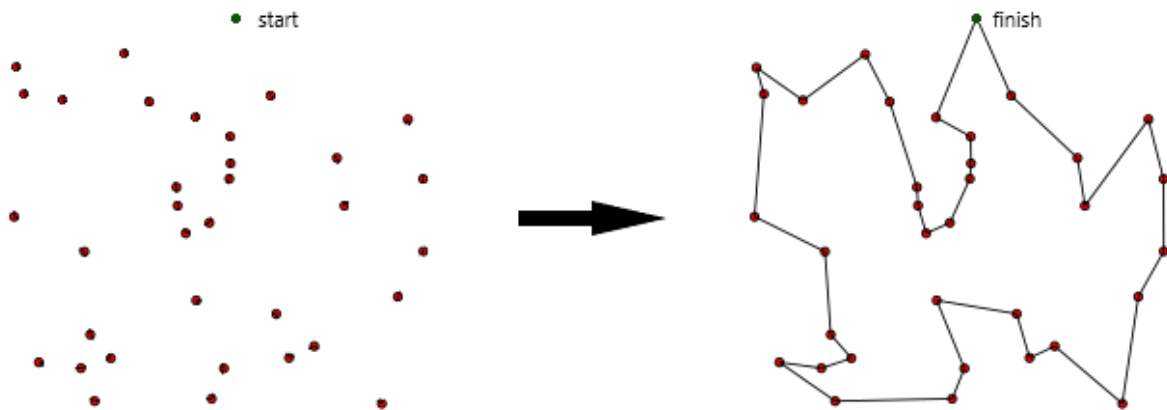
Umelá inteligencia

Zadanie 3 - Problém obchodného cestujúceho

Simulované žižhanie

Riešený problém

Zadanie sa týka riešenia problému obchodného cestujúceho. Jeho cieľom je navštíviť niekoľko miest, kedy každé mesto navštívi práve raz, pričom chce minimalizovať jeho cestovné náklady. Celková cena prepravy je úmerná dĺžke cesty, ktorú obchodný cestujúci prešiel. Preto sa snaží nájsť takú cestu, aby jeho náklady boli čo najmenšie. Keďže sa nakoniec musí vrátiť do mesta z ktorého vychádza, tak je jeho cesta uzavretá krivka.



Príklad trasy obchodného cestujúceho

Je daných aspoň 20 miest, kde má každé určené súradnice ako celé čísla X a Y. Tieto súradnice sú náhodne vygenerované. Cena cesty medzi dvoma mestami je určená pomocou Euklidovej vzdialenosti. Celková dĺžka cesty je reprezentovaná postupnosťou (permutáciou) jednotlivých miest. Cieľom je nájsť takú permutáciu, ktorá bude mať celkovú dĺžku trasy čo najmenšiu.

Napríklad:

(60, 200), (180, 200), (100, 180), (140, 180), (20, 160), (80, 160), (200, 160), (140, 140), (40, 120), (120, 120), (180, 100), (60, 80), (100, 80), (180, 60), (20, 40), (100, 40), (200, 40), (20, 20), (60, 20), (160, 20)

Opis riešenia

Úlohou je použiť algoritmus simulovaného žihania so správne zvolenými parametrami tak, aby algoritmus neuviazol v lokálnom extréme a našiel najlepšie riešenie/cestu (globálny extrém).

Na začiatku sa vygeneruje mapa s náhodnými súradnicami miest, alebo sa mapa načíta so súboru. Následne sa vytvorí náhodná počiatočná cesta (permutácia miest), od ktorej sa bude hľadať tá najlepšia. Od počiatočnej cesty sa postupne vytvárajú jej nasledovníci, ktorí môžu byť lepší alebo horší.

Na to aby algoritmus neuviazol v lokálnom extréme je potrebné pripustiť možnosť akceptovania aj horšieho nasledovníka ako riešenie. Týmto spôsobom je možné obísť lokálne extrémny a priblížiť sa k tomu globálnemu s najlepším výsledkom.

Hlavné parametre algoritmu:

- Počiatočná (*aktuálna*) teplota
- Minimálna teplota
- Pokles teploty
- Dĺžka etapy

Priebeh algoritmu

Algoritmus simulovaného žihania je reprezentovaný triedou ***SimulatedAnnealing***, konkrétne funkciou ***findRoute()*** s parametrami počiatočnej teploty a cesty. Po jej spustení sa ako aktuálna cesta nastaví tá počiatočná, počiatočná teplota sa nastaví ako “aktuálna” a následne sa podľa parametrov algoritmu začne prehľadávanie priestoru.

Ak je aktuálna teplota vyššia ako minimálna teplota, tak sa spustí etapa, v ktorej prebieha generovanie nasledovníkov z aktuálnej cesty pomocou funkcie ***createAdjacentRoute()***, kedy sa permutácia miest aktuálnej cesty upraví. A to tak, že sa náhodne vymenia 2 pozície miest. Takéto generovanie nasledovníkov prebieha až dovtedy, pokiaľ niektorý z nich nebude akceptovaný ako, alebo pokiaľ neskončí etapa.

O akceptovaní nasledovníka rozhoduje funkcia ***acceptRoute()***. V nej sa porovnávajú výsledky fitness funkcie pre nasledovníka a aktuálnu cestu. Fitness funkciu predstavuje funkcia ***getTotalDistance()***, ktorá vypočíta celkovú dĺžku cesty.

- Pokiaľ ma nasledovník **lepšiu** (*kratšiu*) celkovú dĺžku cesty, ako je celková dĺžka aktuálnej cesty, je akceptovaný so 100% pravdepodobnosťou.
- Pokiaľ má nasledovník **horšiu** (*dlhšiu*) celkovú dĺžku cesty, ako je celková dĺžka aktuálnej cesty, tak sa vypočíta % pravdepodobnosť akceptovania tohto nasledovníka, ktorá závisí od rozdielu porovnávaných jednotlivých dĺžok ciest. Čím je tento rozdiel väčší a aktuálna teplota nižšia, % pravdepodobnosť akceptovania je nižšia.

Pokiaľ bol vygenerovaný nasledovník akceptovaný, tak sa nastaví ako aktuálna cesta. Následne sa táto nová aktuálna cesta porovná s globálne najlepšou cestou. Pokiaľ je **lepšia** (*kratšia*), tak ju nahradí. Etapa sa následne ukončí, zníži sa aktuálna teplota a cyklus sa opakuje.

Pokiaľ nebol žiadny nasledovník z etapy akceptovaný, prehľadávanie priestoru končí a riešením je globálne najlepšia cesta. Prehľadávanie priestoru taktiež končí, pokiaľ aktuálna teplota dosiahne hodnotu minimálnej teploty.

Reprezentácia údajov

Reprezentácia cesty – trieda **Route**

```
public class Route implements Comparable<Route> {
    private final ArrayList<City> cities = new ArrayList<>();

    public Route(Route route) { this.cities.addAll(route.cities); }
    public Route(ArrayList<City> cities, boolean shuffle) {...}

    // Fitness function, return total route distance
    public double getTotalDistance() {...}

    // Return cities ArrayList
    public ArrayList<City> getCities() { return this.cities; }

    @Override
    public String toString() { return Arrays.toString(cities.toArray()); }

    @Override
    public int compareTo(Route route) {...}
}
```

Reprezentácia mesta – trieda **City**

```
public class City {
    private final int posX;
    private final int posY;
    private final int name;

    public City(int posX, int posY, int name) {...}

    // Measure distance between 2 cities
    public double measureDistance(City city) {...}

    // Getters
    public int getPosX() { return this.posX; }
    public int getPosY() { return this.posY; }
    public int getName() { return this.name; }

    @Override
    public String toString() { return Integer.toString(this.name); }
}
```

Reprezentácia cestovateľa – trieda **Salesman**

```
public class Salesman {
    private static final int MAX_MAP_SIZE = 200;
    private final ArrayList<City> initialCities;

    public Salesman(String filePath) { this.initialCities = loadCities(filePath); }
    public Salesman(int citiesCount) { this.initialCities = generateCities(citiesCount); }

    // Load cities from file
    public ArrayList<City> loadCities(String filePath) {...}

    // Generate cities with random positions
    public ArrayList<City> generateCities(int citiesCount) {...}

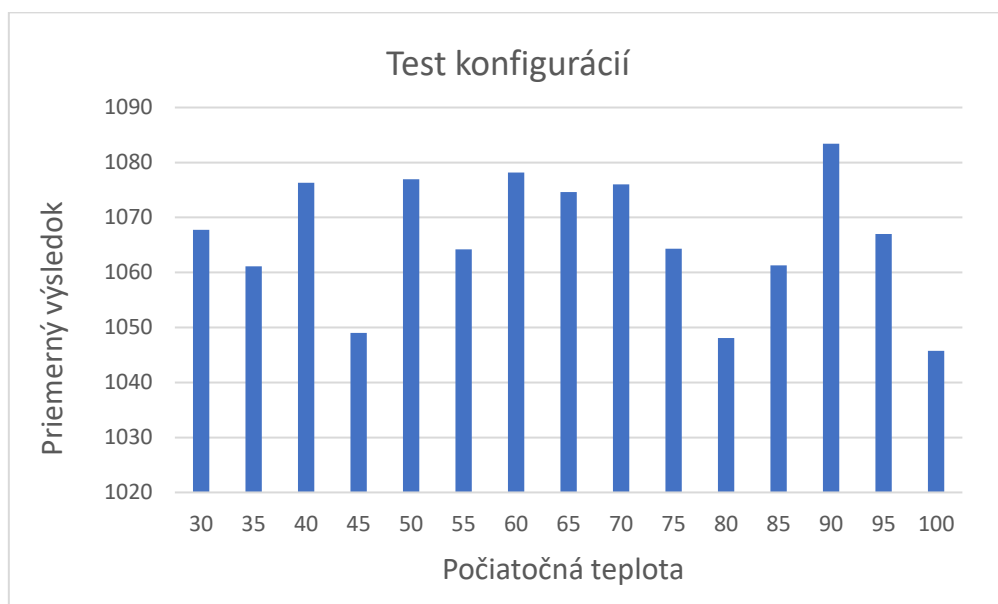
    // Return initial cities
    public ArrayList<City> getInitialCities() { return this.initialCities; }
}
```

Reprezentácia algoritmu – trieda **SimulatedAnnealing**

```
public class SimulatedAnnealing {  
    public static final double RATE_OF_COOLING = 0.0001;  
    public static final double INITIAL_TEMPERATURE = 40;  
    public static final double MIN_TEMPERATURE = 0.99;  
    public static final int MAX_ADJACENT_ROUTES = 1000;  
  
    // Function to find the best route  
    public Route findRoute(double temperature, Route currentRoute) {...}  
  
    // Decide if adjacent route will be accepted  
    public boolean acceptRoute(double currentDistance, double adjacentDistance, double temperature) {...}  
  
    // Generate new adjacent route  
    public Route createAdjacentRoute(Route route) {...}  
  
    // Function to find the best route with own parameters  
    public Route findRouteTester(double maxTemp, double minTemp, double rateOfCooling, int maxAdjRoutes, Route currentRoute) {...}  
}
```

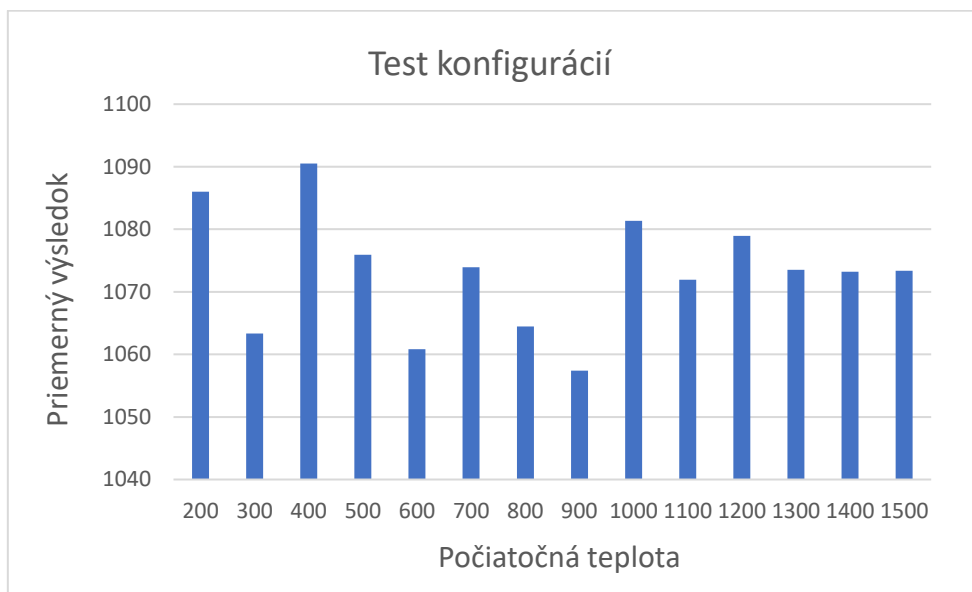
Riešený problém testovania a výsledky experimentov

Počas automatizovaného testu pre to isté rozloženie 40 miest s 20 iteráciami pre každú konfiguráciu počiatočnej teploty. Najlepšie výsledky boli dosiahnuté pri počiatočných teplotách 45, 80, a 100. Najlepšia voľba bola teplota s hodnotou 100.



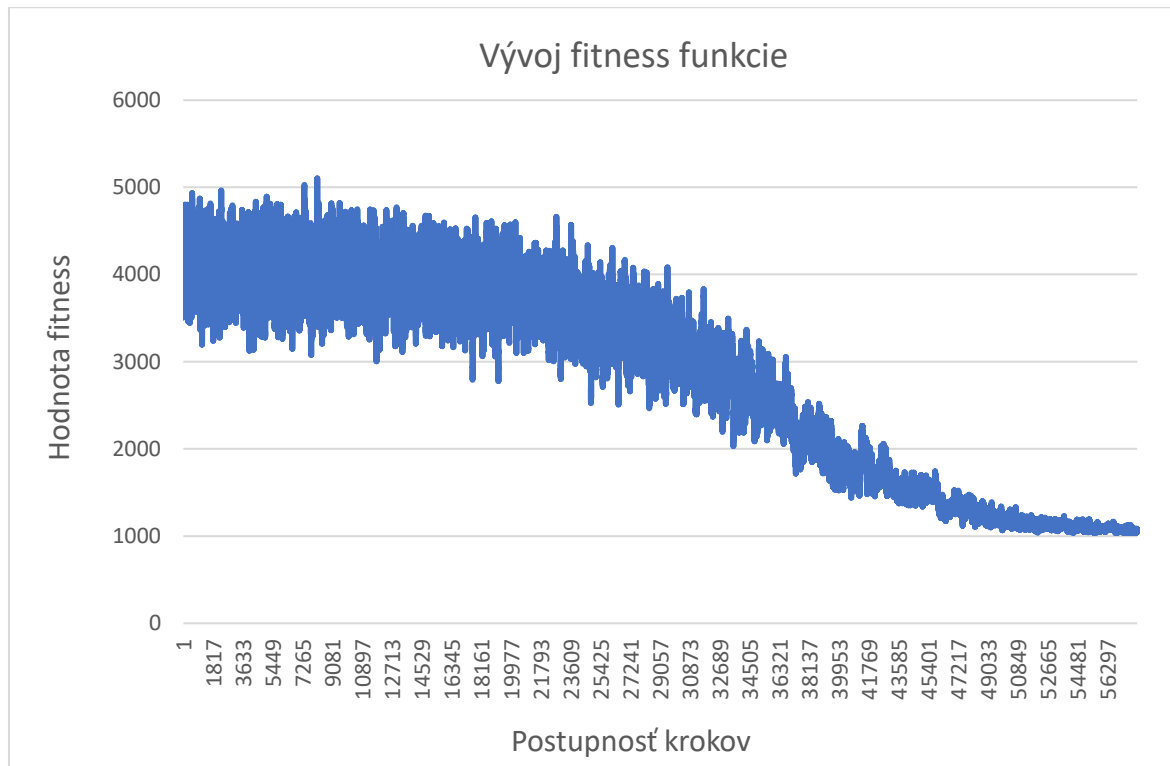
Následne som si všimol, že tento rozsah teplôt nie je dostatočne vysoký na to, aby som dosiahol globálny extrém a neuviazol v lokálnom extréme. Tým že bol rozsah teplôt príliš malý, tak pravdepodobnosť akceptovania horšieho nasledovníka bola obmedzená.

Preto som spravil automatizovaný test pre to isté rozloženie miest som vyskúšal aj s vyššími hodnotami teplôt, avšak výsledky boli podobné.



Pri manuálnom testovaní som skúšal rôzne rozvrhy všetkých parametrov algoritmu. Najlepšie priemerné výsledky dosahovalo nastavenie parametrov s týmito hodnotami, kedy som pre ten istý príklad rozloženia miest získal z 20 iterácií 18-krát globálne maximum:

- Počiatočná teplota - **INITIAL_TEMPERATURE = 1500.0**
- Minimálna teplota - **MIN_TEMPERATURE = 1.0**
- Pokles teploty - **RATE_OF_COOLING = 0.0001**
- Dĺžka etapy - **MAX_ADJACENT_ROUTES = 1000**



Všeobecné informácie

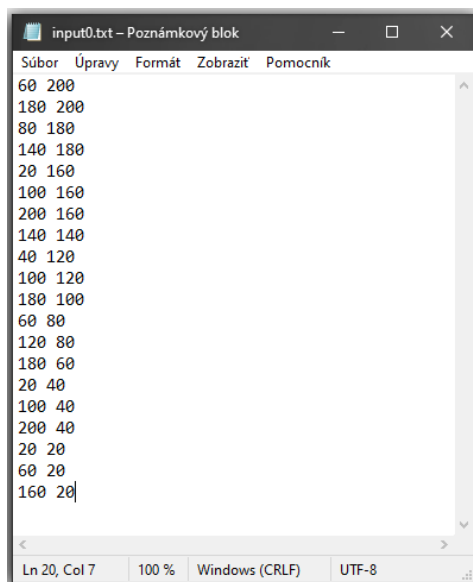
Program:

- Program bol vyhotovený v jazyku Java (SDK 16)
- Použité bolo IDE IntelliJ IDEA Ultimate 2021.2

Vstupy:

- Jednotlivé mestá sa generujú náhodne pri vytvorení objektu inštancie triedy ***Salesman***, kedy sa použije konštruktor, ktorý pomocou funkcie ***generateCities()*** vygeneruje náhodné rozloženie miest.
- Taktiež je možné vstup načítať z .txt súboru, ktorý sa musí nachádzať v priečinku inputs. Textový súbor musí mať štruktúru <X-ováPozíciaMesta> <Y-ováPozíciaMesta>.

Príklad súboru:



```
input0.txt - Poznámkový blok
Súbor  Úpravy  Formát  Zobrazit  Pomocník
60 200
180 200
80 180
140 180
20 160
100 160
200 160
140 140
40 120
100 120
180 100
60 80
120 80
180 60
20 40
100 40
200 40
20 20
60 20
160 20
```