SLOVENSKÁ TECHNICKÁ
UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY
A INFORMAČNÝCH TECHNOLÓGIÍ

Strojové učenie sa – evolučný algoritmus a evolučné programovanie

Matúš Krajčovič

Kód predmetu: **UI_B**
Názov predmetu: **Umelá inteligencia**
Študijný program: Informatika
Študijný odbor: Informatika
Garant predmetu: : **Ing. Lukáš Kohútka, PhD.**
Vyučujúci: Ing. Lukáš Kohútka, PhD.
Cvičiaci: Ing. Ivan Kapustík
Úloha: **Úloha 3 - Problém obchodného cestujúceho**
AIS ID: 103003
Dátum: 22. 11. 2020

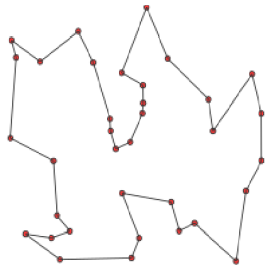
Zadanie úlohy

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 – vypočíta sa pomocou Pytagorovej vety. Celková dĺžka trasy je daná nejakou permutáciou (poradím) miest. Cieľom je nájsť takú permutáciu (poradie), ktorá bude mať celkovú vzdialenosť čo najmenšiu.

Výstupom je poradie miest a dĺžka zodpovedajúcej cesty.

Možné riešenia:

- Genetický algoritmus
- Zakázané prehľadávanie (tabu search)
- Simulované žíhanie (simulated annealing)



Obr. Príklad trasy medzi náhodne zvolenými mestami

Príklad rozloženia miest (20 miest):

(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)

Ja som si vybral simulované žíhanie a genetický algoritmus.

1 Simulované žíhanie

1.1 Reprezentácia údajov

Jednotlivé stavy reprezentujem ako vektory celých čísel. Každé číslo je poradím mesta vo vstupnom súbore. Pri výpočte fitnes sa využije graf, ktorý máme uložený v hlavnom programe - výpočet vzdialeností. Ten je reprezentovaný ako dvojrozmerné pole premenných typu double.

1.2 Algoritmus

Na začiatku algoritmu sa vygeneruje náhodná postupnosť (permutácia) všetkých miest. Jednoducho naplníme vektor mestami od 1 po N a následne náhodne poprehadzujeme.

Pri spúšťaní danej funkcie môže parametrami používateľ zadať nasledujúce:

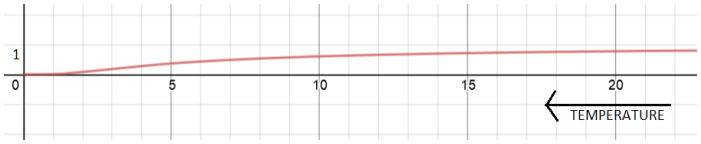
- teplota
- koeficient znižovania teploty
- maximálny počet iterácií
- počet opakovaní hľadania

Nasleduje cyklus, ktorý pokračuje, pokým hodnota teploty neklesne pod 0.001 alebo sa vykoná maximálny počet krokov. V každom cykle zistíme fitnes aktuálneho stavu, vygenerujeme nasledujúci stav poprehadzovaním dvoch náhodných hodnôt. Ak je hodnota fitnes nasledujúceho stavu vyššia, nahradíme ním aktuálny. V opačnom prípade je šanca, že sa nahradí aj tak. Táto šanca je reprezentovaná funkciou:

$$e^{\frac{-|currfit-nextfit|}{temp}}$$

Danú funkciu som našiel na internete[1].

Šanca, že sa vyberie aj horší nasledovník, sa teda postupne znižuje, keďže teplotu na konci každej iterácie znižujeme o koeficient, ktorý tiež zadáva používateľ. Rovnako dekrementujeme počet maximálnych iterácií.

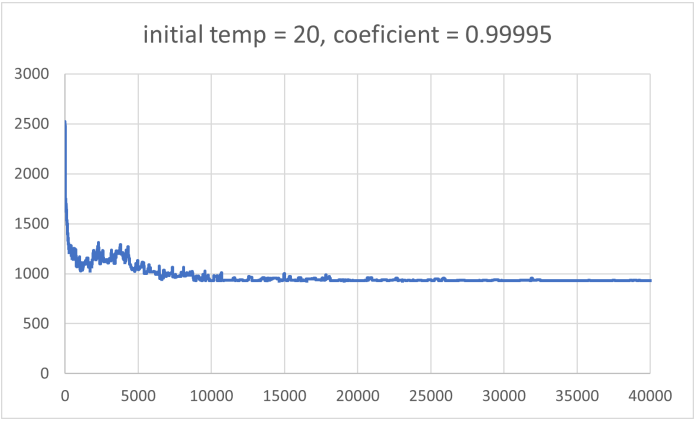


Môj algoritmus som ešte rozšíril - dané prehľadávanie realizujem viackrát, pričom si udržujem stav s najlepšou fitnes. Pri každom opakovaní vrátim hodnotu teploty na začiatočnú a hľadám znova.

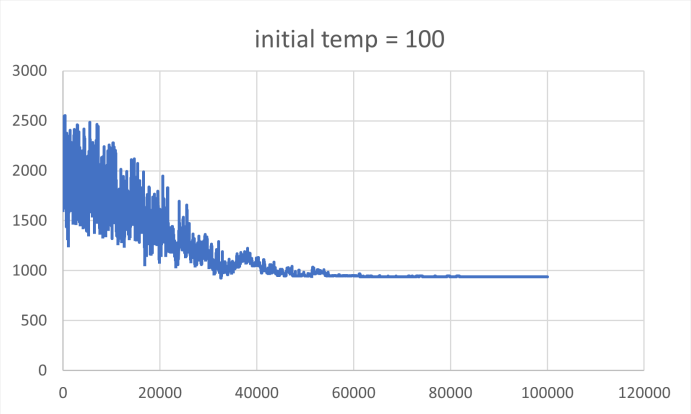
1.3 Testovanie

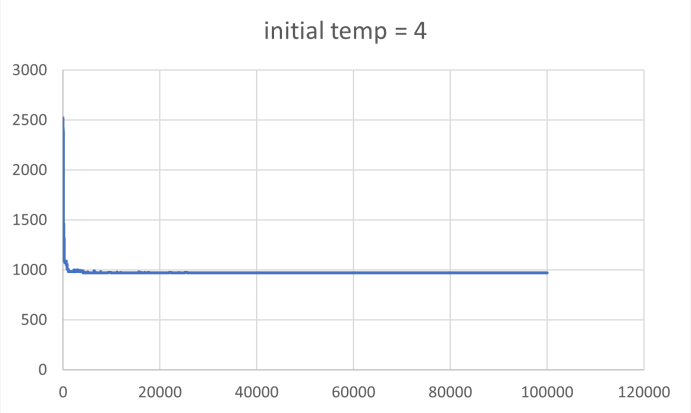
Priebeh hľadania riešenia som zaznamenával do grafov, pri rôznych parametroch. Všetky výslekdy sú zhodnotené v poslednej sekcii 6.

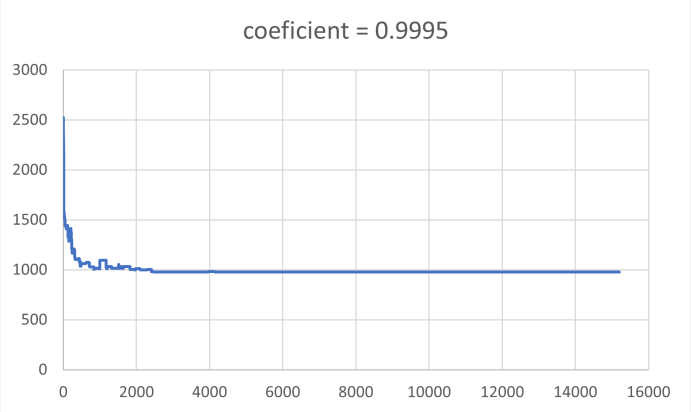
Pri žíhaní sa dá sledovať iba priebeh jednotlivých stavov v čase, resp. ich fitnes:

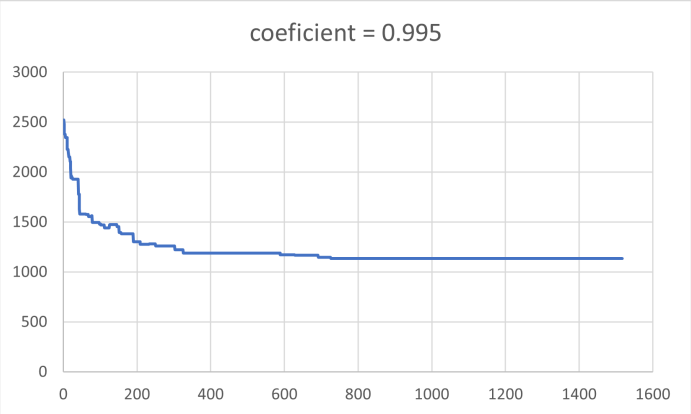


Menil som hodnoty iniciálnej teploty aj koeficientu jej znižovania:

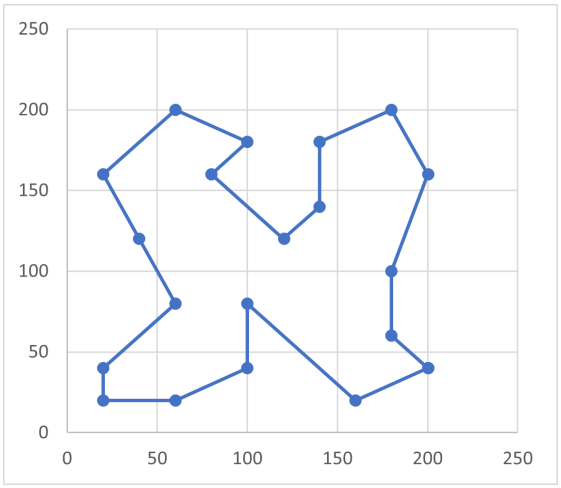




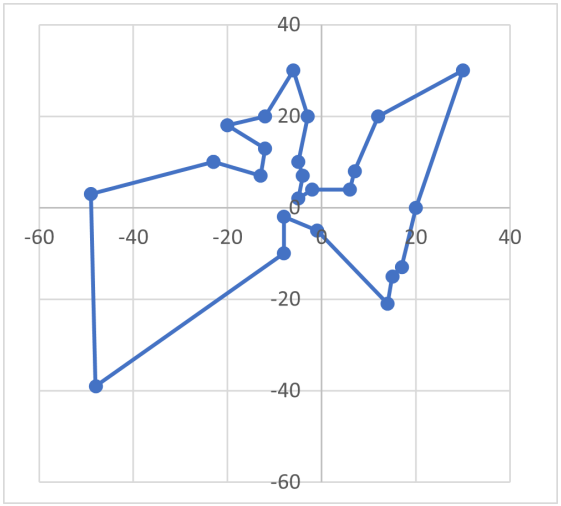


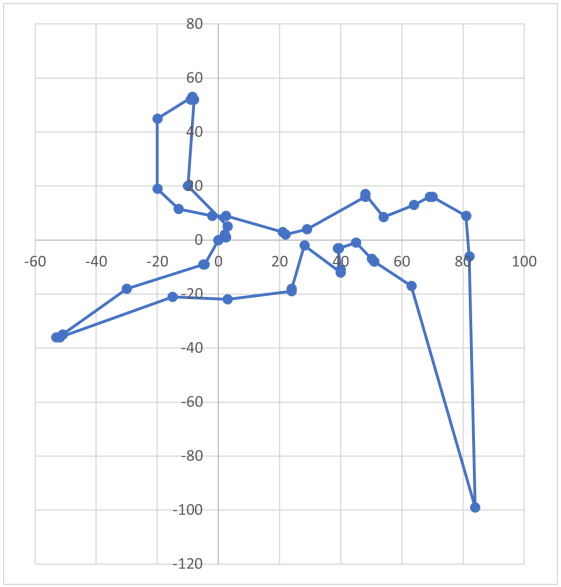


Pri jednom spustení sa často nenájde optimálne/dobré riešenie, no pri hľadaní vždy konvergujeme ku nejakej relatívne malej hodnote. Pri viacerých opakovaniach sú výsledky oveľa lepšie:



Otestoval som aj iné mapy, dve z nich som priložil aj do textového súboru. Pri použití simulovaného žíhania aj s opakovaním (v našom prípade sme nastavili na 10 krát) boli výsledné cesty takéto:





Tiež som testoval iné možnosti prvotného stavu, pričom som vyskúšal greedy search namiesto náhodnej inicializácie, no výsledky boli veľmi podobné. Zhodnotenie parametrov, veľkosti vstupov je v poslednej kapitole 6.

2 Genetický algoritmus

2.1 Reprezentácia údajov

Generácia je reprezentovaná ako vektor chromozómov, pričom každý chromozóm je vektor celých čísel = rovnako ako pri simulovanom žíhaní. Jedna generácia je teda vektor vektorov čísel (poradí miest).

2.2 Algoritmus

Na začiatku si vytvoríme náhodnú populáciu zadanej veľkosti, pričom postup je rovnaký ako pri simulovanom žíhaní, iba sa vykoná zadaný počet krát (podľa počtu generácií).

Používateľ môže parametrami zmeniť:

- veľkosť populácie
- veľkosť zápasu (využíté iba pri tournament selection)
- maximálny počet iterácií
- podiel elitizmu
- pravdepodobnosť mutácie

Následne iterujeme, pokým sa nevykoná maximálny počet krokov zadaný používateľom. Pri oboch implementáciách vyberáme konkrétnym spôsobom opísaným nižšie páry rodičov, z ktorých vytvárame zadaný počet potomkov (pop_size * (1 - elitism_ratio)). Zvyšný počet voľných miest v populácií zaplníme najlepšími jedincami z danej populácie - podľa percenta elitizmu.

2.2.1 Kríženie

Kríženie prebieha pomocou dvoch rodičov, pričom sa vygenerujú dve pozície. Tento interval sa prekopíruje z prvého rodiča, zvyšné gény sa naplnia z druhého rodiča tak, aby boli v rovnakom poradí a aby sa neopakovali - pričom začíname vkladať gény od konca intervalu po koniec chromozómu, a až potom od začiatku chromozómu po začiatok intervalu. Toto sa vykoná pre oboch rodičov, výsledkom sú teda dvaja potomkovia.

2.2.2 Mutácia

Mutácia prebieha pri každom jedincovi novej generácie, pričom sa aplikuje používateľom zadaná pravdepodobnosť. Pri mutácii sa vymenia dva náhodné gény. Moja implementácia má mutácie až tri, pričom druhá je menej pravdepodobná ako prvá (probability /= 2), a tretia je ešte menej pravedpodobná, ako druhá (probability /= 4).

2.2.3 Roulette selection

Pri tejto implementácii si pri každej iterácii spočítam globálnu fitnes. Potom postupne prechádzam všetkými chromozómami aktuálnej generácie a kumulujem ich fitnes. Ak presiahnem vopred určenú náhodnú hranicu (v intervale od 0 po globálnu fitnes), našiel som prvého rodiča. Proces ešte raz zopakujem s inou náhodnou hodnotou a z oboch rodičov vytvorím dvoch potomkov. Postup opakujem, pokým nesplním požiadavku počtu potomkov zadanú používateľom.

2.2.4 Tournament selection

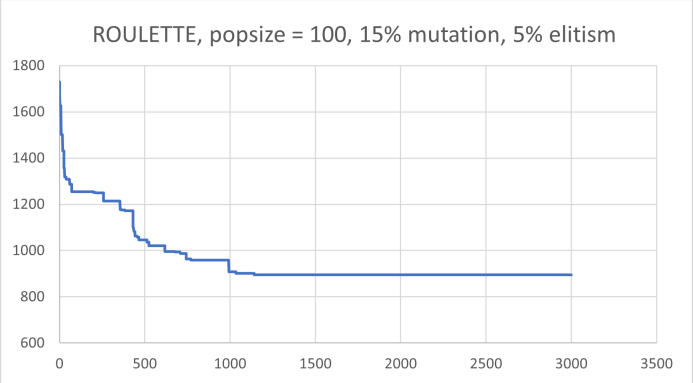
Pri tejto implementácii si vyberiem používateľom zadaný počet náhodných chromozómov, nájdem najlepšieho z nich. To zopakujem dva krát a vytvorím z oboch rodičov dvoch potomkov. Opakujem a vytváram "zápasy" a hľadám rodičov a krížim ich, pokým nenaplním daný počet potomkov.

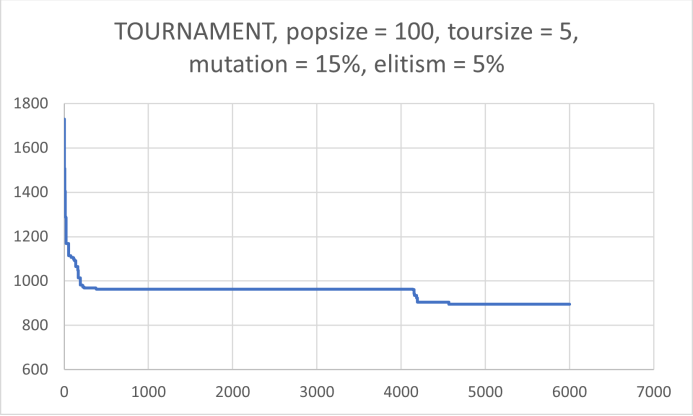
V oboch typch selekcie naplním zvýšok populácie náhodnými jedincami.

2.3 Testovanie

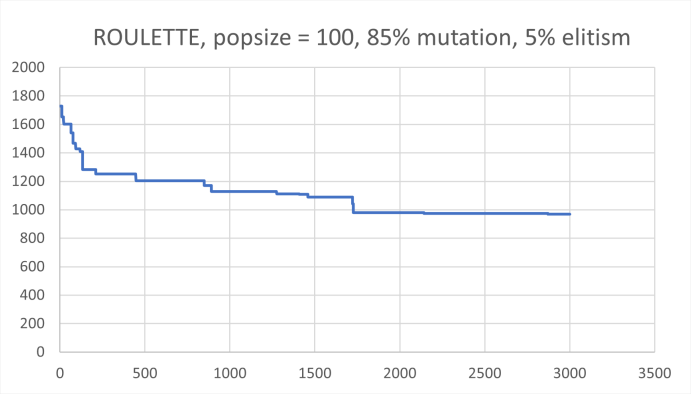
Testovanie algoritmu prebiehalo v skúšaní rôznych máp a rôznych parametrov pre dané mapy. Zhodnotenie testovania robím v poslednej sekcii 6.

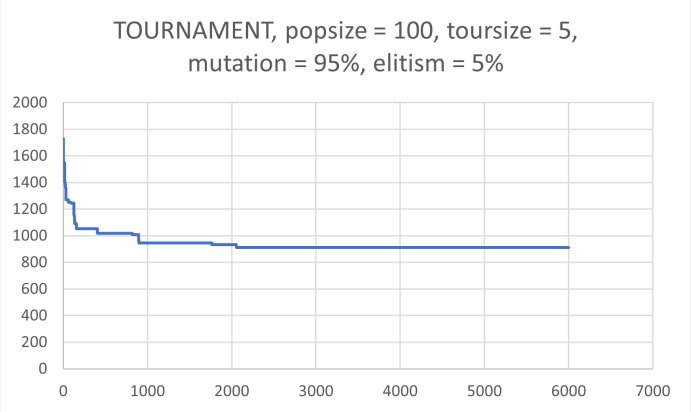
Jednu z hodnôt, ktoré som sledoval, je fitnes najlepšieho jedinca naprieč populáciami. Krivka ukazuje jeho vylepšovanie v čase. Pre dané parametre a dané spôsoby hľadania rodičov sú výsledky nasledovné:

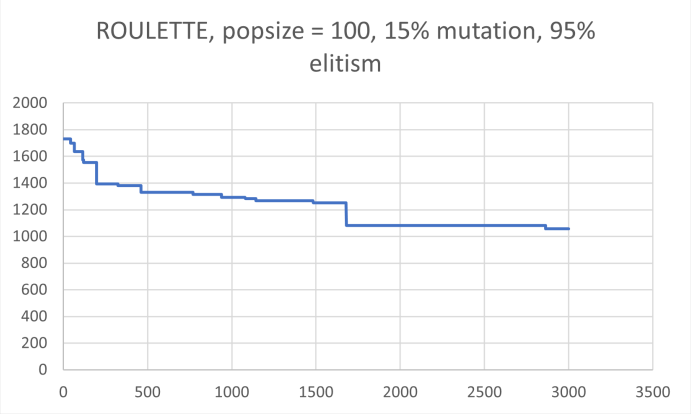


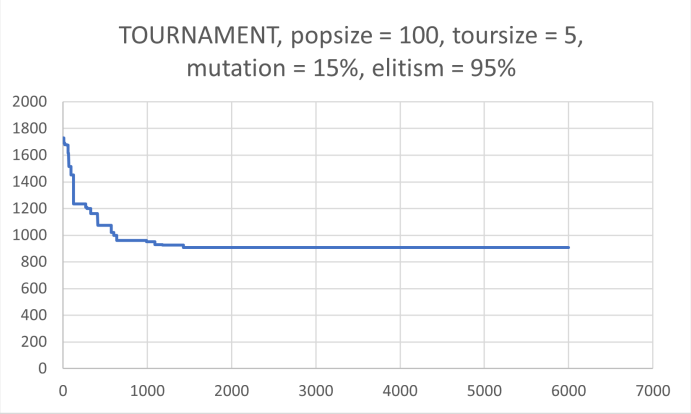


Pri nasledujúcich grafoch som menil pravdepodobnosť mutácie a podiely elitizmu:



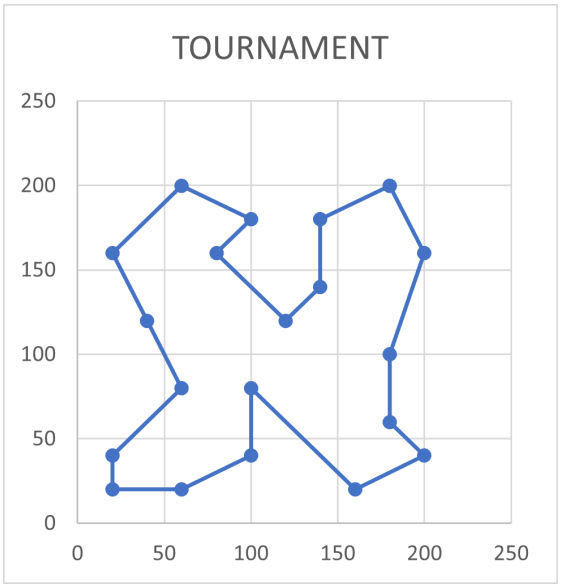


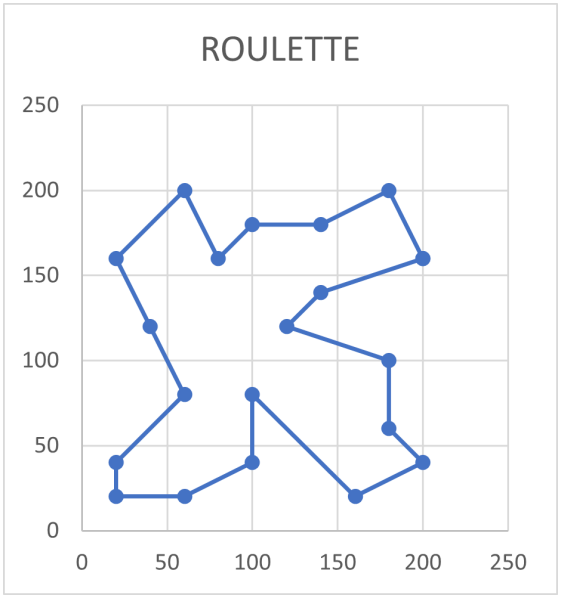




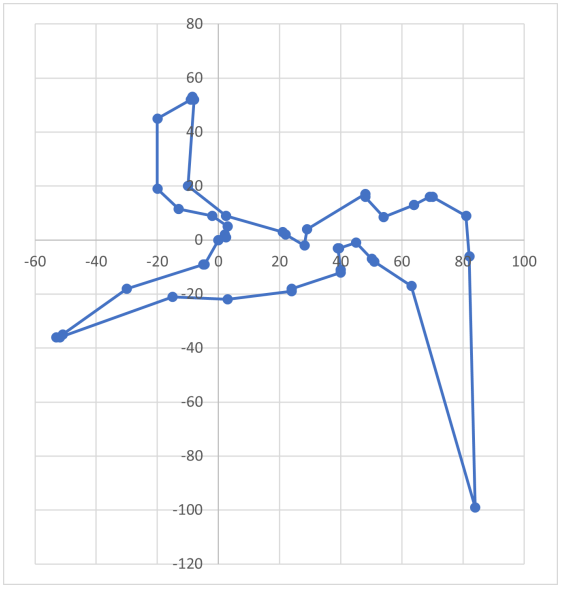
Algoritmy som spúšťal viac krát, a pri rôznych mapách som našiel rôzne parametre (veľkosť populácie, počet opakovaní), pre ktoré sa našlo minimálne riešenie najviac krát. Často sa však stávalo, že sa pri každom opakovaní hodnoty mierne líšili.

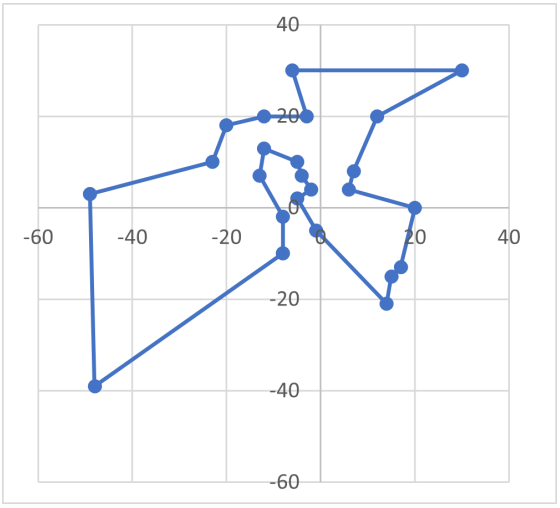
Obe z nasledujúcich výsledkov vznikli pri veľkosti populácie 200 a počte opakovaní 1000. Samozrejme, pri viacerých spusteniach algoritmu sa dané cesty líšia, no väčšinou sú veľmi blízko optimálnemu riešeniu, ak nie optimálne.



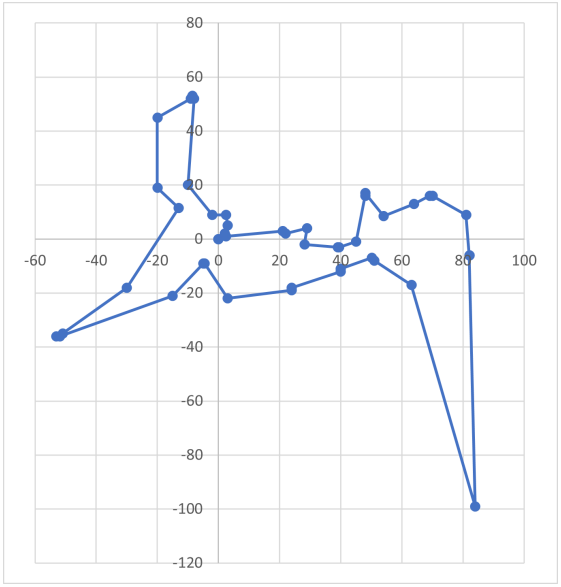


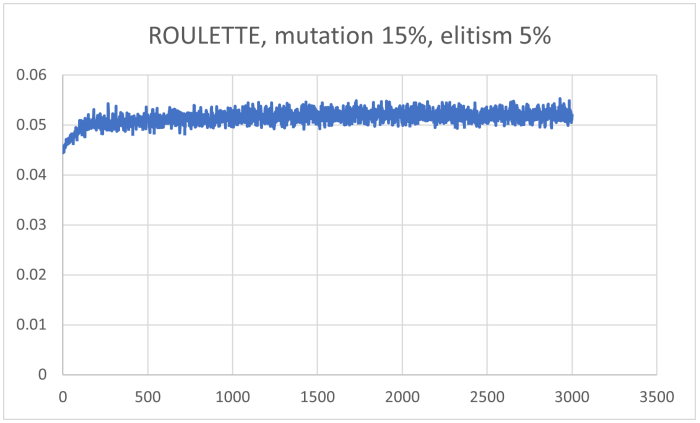
Pri mapách s väčším množstvom miest sú výsledné cesty nasledovné (prvé dva sú ruleta, druhé tournament):



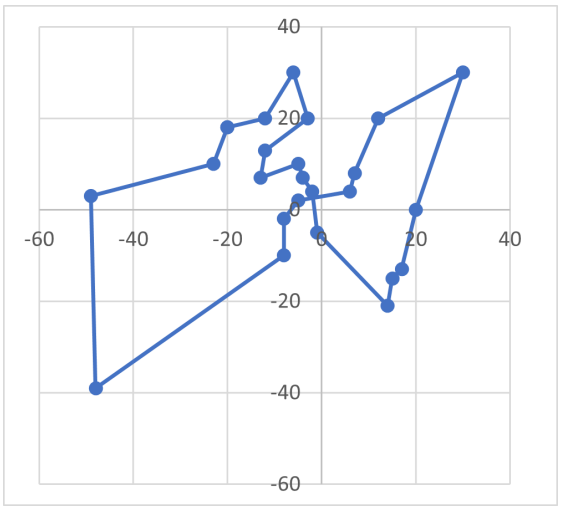


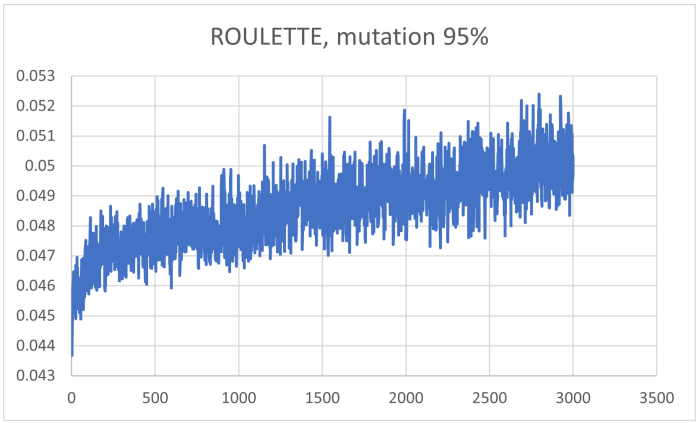
Okrem toho som testoval aj priebeh globálnej fitnes pri rôznych parametroch, ktoré sú napísane nad grafmi:

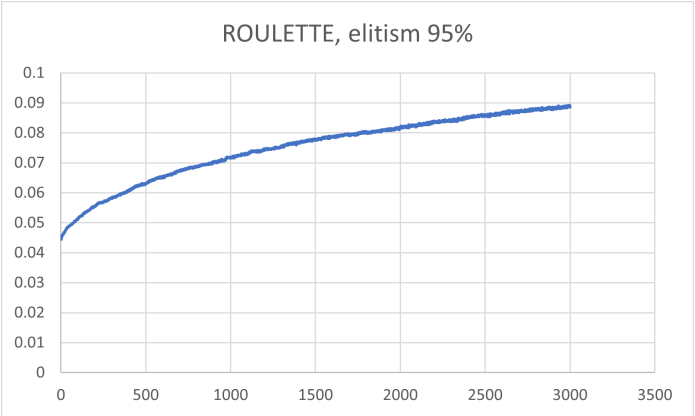


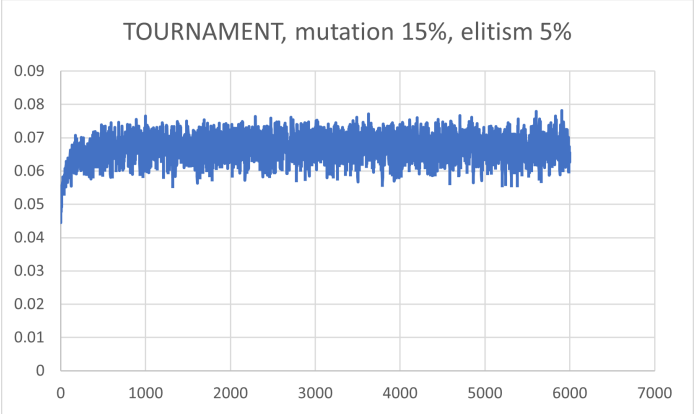


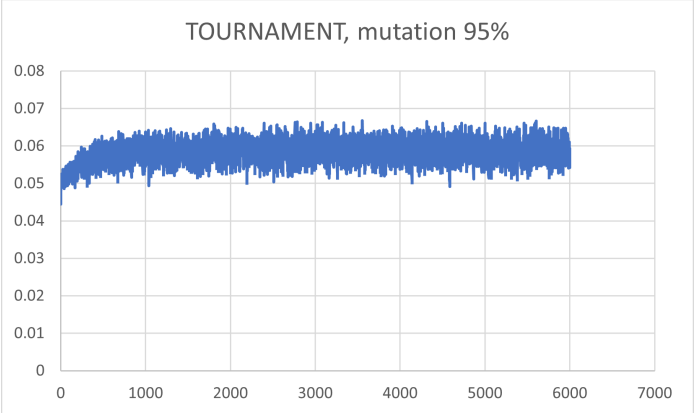
Menil som hodnoty mutácie, aj elitizmu pri oboch typoch selekcie, pričom výsledky sú veľmi podobné:

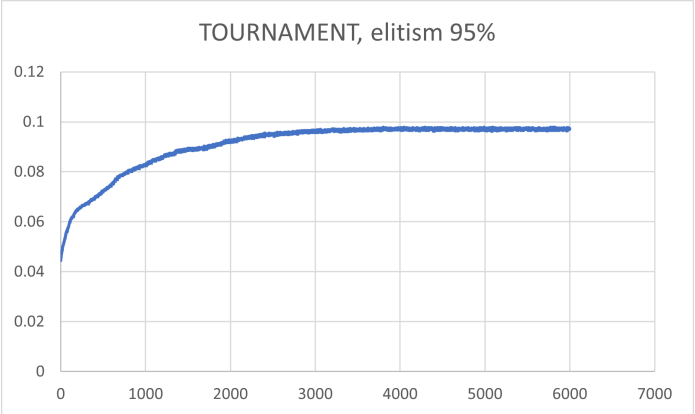












3 Štruktúra kódu

3.1 Simulované žíhanie

Funkcia **annealing()** obsahuje samotný algoritmus simulovaného žíhania, ktorý sme v predchádzajúcej časti podrobne opísali.

Funkcia **get_fitnes()** sa využíva na získanie fitnes hodnoty daného stavu (permutácie) - teda dĺžky cesty. Počíta sa z grafu uloženom v hlavnom programe. Využíva sa aj v genetických algoritmoch.

Funkcia **get_next_state()** vygeneruje dve náhodné pozície, pričom zamení hodnoty vo vstupnom vektore na týchto miestach. Výsledný vektor vráti.

Funkcia **random_init_annealing()** vráti stav (vektor) s náhodným poradím miest. Používame ju na začiatku simulovaného žíhania.

3.2 Genetický algoritmus

Funkcia **genetic()** obsahuje algoritmus genetických algoritmov. V jednom z parametrov volíme typ algortimu (ruleta alebo zápas).

Funkcia **breed()** slúži na kríženie dvoch rodičov, pričom vracia dvoch potomkov. Spolu s funkciou **mutate()** sa používa v genetických algoritmoch. Mutate vráti rovnaký stav, no s pozmeneným poradím niektorých prvkov.

Funkcia **get_global_fitnes()** vráti súčet všetkých prevrátených fitnes hodnôt danej populácie.

Funkcia **get_parents_roulette()** slúži na výber rodičov, ktorí sa v genetickom algoritme krížia a tvoria potomkov - konkrétne pre výber ruletou. Podobný účel má aj funkcia **get_parents_tournament()**.

Funkciu **random_init_genetic()** využívame na začiatku genetického algoritmu, pričom vráti populáciu náhodných stavov.

Funkcia **is_fitter()** sa využíva pri zoraďovaní stavov v populácii funkciou std::sort().

3.3 Ostatné

Na koniec sú ešte funkcie **init_graph()** pre inicializáciu grafu z externého súboru a **print_solution()**, ktoráej funkcionalita je asi samozrejmá.

Funkciu **greedy_srch()** nepoužívam, no testoval som ňou simulované žíhanie. Vyberie prvý bod, nájde ďalší, najbližší k nemu, atď..

4 Externé súbory a knižnice

Program využíva externý súbor input.txt, v ktorom je počet miest a následne všetky súradnice daných miest. Okrem toho som v externom súbore ponechal dve z máp, ktoré som testoval.

V programe používam iba štandardné knižnice ako <iostream> a <fstream> na prácu so súbormi, <vector> na chromozómy a populácie, <cmath> na pravdepodobnostnú funkciu pri simulovanom žíhaní, <algorithm> na sortovanie a <random> pre náhodné distribúcie čísel.

5 Prostredie

Program sa dá spustiť s príkazového riadku, pričom ak zadáme parameter "help", zobrazí sa návod na použitie - možnosti parametrov.

Program môžeme spustiť s tromi možnosťami (annealing, roulette, tournament), pričom po spustení help sa zobrazí, aké parametre môžeme vpísať.

Ukážka:

```
C:\Users\Matus\Desktop\UI_zad_3\Release>UI_zad_3.exe help
annealing   [initial temperature] [coefficient] [iterations] [repeatings] [*print]
roulette    [population size] [iterations] [elitism percentage] [mutation probability]
 [*print]
tournament  [population size] [tournament size] [iterations] [elitism percentage] [mut
ation probability] [*print]

C:\Users\Matus\Desktop\UI_zad_3\Release>UI_zad_3.exe annealing 30 0.99995 100000 10
SIMULATED ANNEALING:

14: 180 60
17: 200 40
20: 160 20
13: 100 80
16: 100 40
19: 60 20
18: 20 20
15: 20 40
12: 60 80
9: 40 120
5: 20 160
1: 60 200
3: 100 180
6: 80 160
10: 120 120
8: 140 140
4: 140 180
2: 180 200
7: 200 160
11: 180 100

Length: 895.706
```

6 Zhodnotenie testovania

6.1 Simulované žíhanie vs. genetické algoritmy

Pri mapách s menším počtom miest sa mi viac osvedčili genetické algoritmy, pri ktorých som vždy, aj keď niekedy nie na prvý krát, našiel optimálne riešenie. Negatívum bolo ale to, že to trvalo o niečo dlhšie ako simulované žíhanie

Pri mapách s väčším počtom miest treba nastavovať aj väčší počet generácii, a v niektorých prípadoch mi viac-krát spustené simulované žíhanie našlo lepšie riešenie rýchlejšie. Pri jednorázovom spustení je to ale úplný opak, keďže sa mi pri tomto spôsobe jednotlivé výsledky veľmi líšili.

V konečnom dôsledku sú výsledky genetického algoritmu spusteného raz v priemere porovnateľné so simulovaným žíhaním spusteným viackrát.

6.2 Ruleta vs. turnaj

Pri testovaní som nepostrehol významný rozdiel medzi týmito metódami okrem jedného: rýchlosť. Genetický algoritmus s výberom pomocou turnajov mi aj pri väčších veľkostiach populácie bežal rýchlejšie, ako ruletový výber. Ide tam pravdepodobne o neustále spočítavanie kumulatívnej fitnes, pričom pri turnajoch iba volím náhodných jedincov.

Na grafoch ale možno vidieť, že pri tournament selection generácie konvergujú rýchlejšie, ako pri výbere ruletou.

6.3 Simulované žíhanie - parametre

Pri simulovanom žíhaní parametrov nie je veľa, osvedčilo sa mi veľmi pomaly znižovať hodnotu teploty (napr. koeficient 0.9995, prípadne až 0.99995). Pri rýchlom znižovaní môžeme z grafov vidieť, že je veľká šanca zaseknúť sa v lokálnom optime, keďže menej skáčeme "hore-dolu". Počet iterácii je tam iba na to, aby sme nevyhľadávali pri malých koeficientoch veľmi dlho, no hľadanie sa tak či tak zastaví keď teplota klesne pod 0.01. Iniciálna teplota musí byť dostatočne vysoká, lebo inak skonvergujeme ku riešeniu veľmi rýchlo. Na grafe je vidno, ako rýchlo skonvergovala, algoritmus vtedy nestihol dostatočne preskúmať stavový priestor.

Počet opakovaní hľadania som väčšinou ponechával na 10, alebo viac, pričom často nájdem dosť optimálne riešenie aj pri 10.

6.4 Genetické algoritmy - parametre

Pri menších mapách sa mi pri genetických algoritmoch oplatili menšie generácie (napr. 50), pričom počet generácii nemusí byť veľmi veľký. S väčším počtom miest sa musí zväčšiť aj celkový počet generácii (iterácii), pričom som dosahoval lepšie výsledky, keď som o niečo zvýšil aj veľkosť generácie.

Veľkosť zápasu som ponechal vždy medzi 3 až 6, aby sa veľmi nezvýhodňovali najlepší jedinci. Nemám to zdokumentované v grafoch, no pri veľkých zápasoch algoritmus veľmi rýchlo konverguje a generácie nie sú také rôznorodé. Je to vlastne výber väčšinu populácie elitizmom, čo nie je dobré.

Percento elitizmu som ponechal na 5% pri menších mapách, a znižoval som ho pri väčších, aby som neposielal veľa najlepších jedincov do novej generácie. Pri veľkých podieloch sa stáva, že generácie nie sú veľmi rôznorodé, čo vidno aj z grafov globálnych fitnes hodnôt.

Pri zmene pravdepodobnosti mutácie som vo výsledkoch nevidel výrazné rozdiely (teda ak som nedal pravdepodobnosť veľmi veľkú, vtedy sú jedince v populáciách až veľmi rôznorodé, algoritmus sa vtedy podobá na náhodný výber s uchovávaním tých najlepších jedincov elitizmom - bez elitizmu by som asi optimálne riešenie nenašiel). To tiež môžeme vidieť na grafoch globálnej fitnes.

7 Zhodnotenie

Môj program obsahuje implementácie dvoch algoritmov v jazyku C++ na hľadanie cesty pri probléme obchodného cestujúceho. Simulované žíhanie, ktoré som rozšíril o jeho opakovanie, a genetický algoritmus, pri ktorom mám naimplementované dva typy selekcie rodičov - roulette a tournament.

Pri použitých mapách sa mi darilo nachádzať veľmi dobré riešenia, aj keď nie vždy sa toto dobré riešenie našlo.

7.1 Rozšírenia

Podarilo sa mi rozšíriť simulované žíhanie o jeho viacnásobné opakovanie, čo veľmi zvýšilo jeho efektívnosť.

Pri genetickom algoritme by mohlo jedno z rošírení byť veľmi podobné. Pri konvergovaní k nejakému riešeniu (veľké množstvo populácií v rade s rovnakým najlepším jedincom) by sa mohlo hľadanie ukončiť a začať odznova.

Literatúra

[1] Simulated annealing. `https://en.wikipedia.org/wiki/Simulated_annealing`.