SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Zadanie 3

PROBLÉM OBCHODNÉHO CESTUJÚCEHO

Anna Yuová

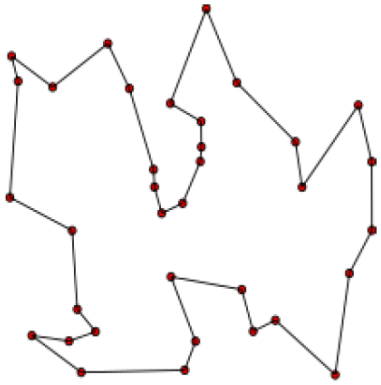
Predmet: Umelá inteligencia

Akademický rok: 2020/2021

Semester: zimný

ZADANIE ÚLOHY

Obchodný cestujúci má navštíviť viacero miest. V jeho záujme je minimalizovať cestovné náklady a cena prepravy je úmerná dĺžke cesty, 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 – 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.



Obr. Príklad trasy medzi náhodne zvolenými mestami

Riešenia:

Genetický algoritmus

Genetická informácia je reprezentovaná vektorom, ktorý obsahuje index každého mesta v nejakom poradí (nejaká permutácia miest). Keďže hľadáme najkratšiu cestu, je najlepšie vyjadriť fitnes jedinca ako prevrátenú hodnotu dĺžky celej cesty.
Jedincov v prvej generácii inicializujeme náhodne – vyberáme im náhodnú permutáciu miest. Jedincov v generácii by malo byť tiež aspoň 20. Je potrebné implementovať aspoň dve metódy výberu rodičov z populácie.

Zakázané prehľadávanie (tabu search)

Zakázané prehľadávanie patrí do skupiny algoritmov, ktoré využívajú na hľadanie riešenia v priestore možných stavov lokálne vylepšovanie (optimalizáciu). To znamená, že z aktuálneho stavu si vytvorí nasledovníkov a presunie do takého, ktorý má lepšie ohodnotenie (najlepšieho takého). Ak neexistuje nasledovník s lepším ohodnotením (a nenašli sme dostatočne dobré riešenie), tak sme v lokálnom extréme a je potrebné sa z neho dostať. Tento algoritmus si teda vyberie horšieho nasledovníka a zároveň si uloží aktuálny stav do tzv. zoznamu zakázaných stavov (tabu list).

Simulované žíhanie (simulated annealing)

Simulované žíhanie patrí do skupiny algoritmov, ktoré využívajú na hľadanie riešenia v priestore možných stavov lokálne vylepšovanie. Zároveň sa algoritmus snaží zabrániť uviaznutiu v lokálnom extréme. Z aktuálneho uzla si algoritmus klasicky vytvorí nasledovníkov. Potom si jedného vyberie. Ak má zvolený nasledovník lepšie ohodnotenie, tak doň na 100% prejde. Ak má nasledovník horšie ohodnotenie, môže doň prejsť, ale len s pravdepodobnosťou menšou ako 100%. Ak ho odmietne, tak skúša ďalšieho nasledovníka. Ak sa mu nepodarí prejsť do žiadneho z nich, algoritmus končí a aktuálny uzol je riešením.

OPIS RIEŠENIA

GENETICKÝ ALGORITMUS

Na začiatku programu si vytvorím súradnice všetkých miest. Vygenerujem si náhodné číslo od 0-200 a to pošlem do triedy Mesto, kde cez konštruktor priradím tieto hodnoty do x-ovej a y-ovej súradnice. Z triedy Mesto si tieto súradnice miest vrátim ako zoznam miest v tvare (x,y). Ďalej mám vytvorenú funkciu genetickyAlgoritmus(), v ktorej prebieha samotný algoritmus na nájdenie cesty.

Vo funkcii genetický algoritmus si na začiatku vytvorím počiatočnú populáciu tým, že náhodne zoradím (pre usporiadam) daný zoznam miest poslaný ako argument do tejto funkcie. Hneď ako vytváram túto náhodnú populáciu, tak jeden prvok populácie je jeden jedinec. Pri vytváraní populácie rovno teda každému jedincovi nastavím jeho náhodný zoznam miest, vypočítam mu fitness hodnotu a nastavím vážený priemer zatiaľ každému na 0. Toto robím v triede Jedinec volanej z funkcie pociatocnaPopulacia() pomocou konštruktora.

Fitness hodnoty rátam vo funkcii vypocetFitness() volanej z triedy Jedinec ako prevrátenú hodnotu celej dĺžky cesty. Dĺžku cesty zisťujem medzi jednotlivými mestami Pytagorovou vetou vo funkcii vzdialenostMiest(). Do tejto funkcie si pošlem ako argumenty 2 za sebou idúce mestá, v absolútnej hodnote vypočítam rozdiely x-ových a y-ových súradníc týchto miest a potom podľa vzorca na pytagorovu vetu vypočítam výsledok (vysledok = odmocnina z (x^2 + y^2)). Takto si postupne pri vytváraní počiatočnej populácie vypočítam aj fitness hodnoty pre jedincov.

Keď už mám vytvorenú počiatočnú populáciu, vypočítam si vážený priemer pre každého potomka. Vážený priemer počítam ako fitness hodnota aktuálne jedinca / suma všetkých fitness hodnôt jedincov. Vo for cykle iba postupne každému jedincovi priradím jeho vážený priemer. Potom si týchto jedincov zoradím podľa váženého priemeru – cez funkciu sort.

Zo zoradenej populácie si idem vybrať 2 najlepších jedincov – rodičov, ktorých budem krížiť. Ide o to, že ak mám 2 silných (s najkratšou cestou) rodičov, tak očakávam, že aj dieťa bude silné. Prvá možnosť výberu, ktorú robím je výber podľa váženého priemeru a druhý výber je pomocou turnaja. Pri váženom priemere si vygenerujem vo funkcii vyberJedincaNaKrizenie() náhodne číslo od 0-1. Ak je toto číslo menšie ako vážený priemer jedincov (na začiatku iba jedného a postupne tie priemery pripočítavam s jedincami), tak si ho rovno vrátim a toho jedinca budem krížiť. Ak to nie je menšie ako to moje náhodné číslo, pripočítam vážený priemer ďalšieho jedinca a idem pracovať s ďalším jedincom. Toto robím v cykle a túto funkciu volám 2krát, keďže vyberám 2 jedincov na kríženie týmto istým spôsobom. Ak by moje náhodné číslo náhodou nikdy nebolo menšie ako tie vážené priemery, vrátim si populáciu s indexom -1 (posledný jedinec). Ešte skontrolujem či 2 vybraní jedinci nie sú rovnakí a ak sú nájde druhého odlišného ako prvého.

Ak vyberám rodičov turnajom vyberiem vo funkcii turnaj() náhodne 3 jedincov z populácie. Z týchto 3 jedincov následne vyberiem toho, ktorý má najvyššiu fitness hodnotu (najkratšiu cestu).

Keď už mám vybraných 2 rodičov, idem ich skrížiť, aby som dostala ich potomka. Do funkcie krizenie() si pošlem ako argumenty oboch vybraných rodičov. Náhodne vygenerujem 2 čísla – indexy, kde bude začínať a končiť prvá prevzatá časť dieťaťa z prvého rodiča. Keď už mám prevzatú časť génu z prvého rodiča, zvyšok génov doplním z druhého rodiča – tie, ktoré sa tam ešte nenachádzajú a v takom poradí v akom sa nachádzajú v druhom rodičovi.

Napr.

Po skrížení nasleduje mutovanie. Keď si vo funkcii krizenie() vytvorím nového skríženého potomka, zavolám ešte funkciu mutacia(), do ktorej si ako argument pošlem nové skrížené dieťa a pravdepodobnosť mutácie, ktorú som si určila na začiatku na pevno. Mutujem tak, že si vygenerujem náhodné číslo od 0-1. Ak je toto číslo menšie ako pravdepodobnosť mutácie, idem skrížené dieťa zmutovať, ak nie je neurobím nič – nemutujem. Normálne sa táto pravdepodobnosť mutácie dáva ako veľmi nízke číslo – 3 až 5%. To by som ale musela vygenerovať číslo menšie ako 0,03 alebo 0,05 a pravdepodobnosť toho bola veľmi malá, preto som si dala trochu vyššiu pravdepodobnosť – 20%, len na odskúšanie toho, či to dobre zmutuje. Mutujem tak, že si vygenerujem náhodné číslo od 0-1. Ak je toto číslo menšie ako pravdepodobnosť mutácie, idem skrížené dieťa zmutovať, ak nie je vyššia neurobím nič – nemutujem. Ak je to číslo teda menšie, idem mutovať dieťa tým, že vymením 2 náhodné miesta vedľa seba.
Napr.

Do funkcie geneticky Algoritmus si teda vrátim buď iba skrížené dieťa alebo aj zmutované. Skrížených alebo aj zmutovaných potomkov si teda uložím do pola novaPopulacia. Toto celé – výber rodičov, kríženie, mutovanie a ich fitness hodnoty a vážené priemery robím vo for cykle toľkokrát, koľko si na začiatku zadám, že budem mať generácii – na pevno zadané číslo. Na konci populácie vypíšem najlepšieho potomka a jeho dĺžku cesty – prejdem celé pole novaPopulacia a nájdem toho, ktorý má najkratšiu cestu podľa fitness hodnoty a toho aj vypíšem.

Porovnanie výberu rodiča pomocou turnaja a váženého priemeru:

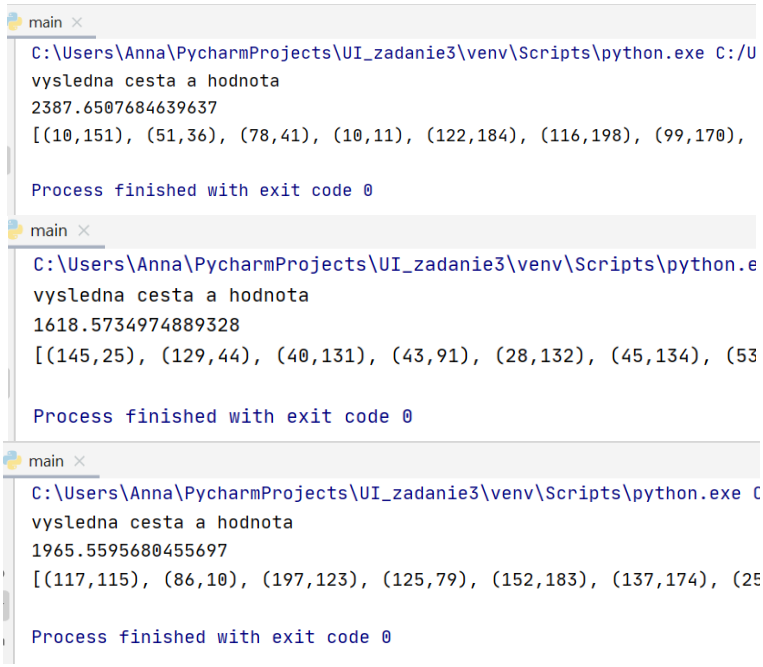
Vyberám vždy 2 najlepších rodičov na základe váženého priemeru a náhodne vygenerovaného čísla, keďže mám ale všetkých jedincov zoradených podľa váženého priemeru dáva mi to lepšie výsledky ako pri turnaji. Pri výbere podľa turnaja vyberám náhodne hocijakých jedincov do turnaja a z nich vyberiem 2 z najlepšou fitness hodnotou. Tí dvaja vybraní ale stále môžu mať oveľa horšiu fitness hodnotu ako zvyšní jedinci, ktorí ani neboli vybraní do turnaja.
Pri váženom priemere očakávam, že ak vyberiem silných rodičov, dostanem ešte silnejšie dieťa, ktoré bude mať ešte kratšiu cestu. Pri turnaji dávam možnosť aj slabším jedincom, a preto sa môže stať aj možnosť, že vyberiem ako rodičov slabších jedincov. Ak mám slabých jedincov je málo pravdepodobné, že po ich krížení sme dostali veľmi silné dieťa (s ideálnou cestou).

Príklad výberu na základe váženého priemeru:

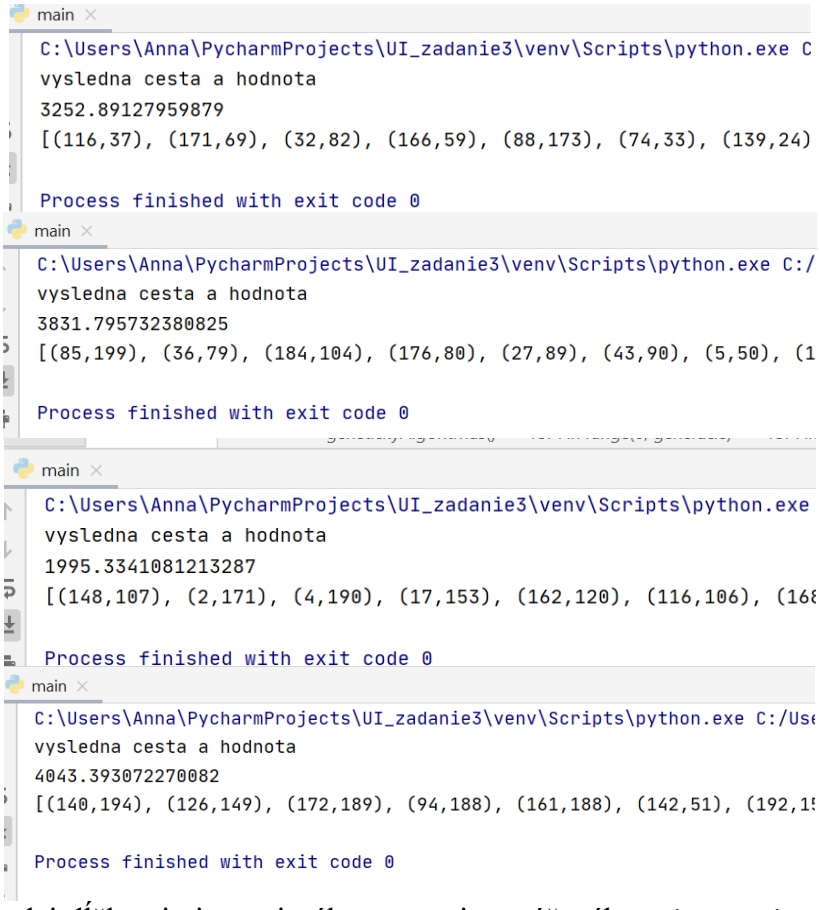
Program som spustila párkrát po sebe a dĺžky výsledných ciest sa pohybovali v priemere okolo 2136.

```
C:\Users\Anna\PycharmProjects\UI_zadanie3\venv\Scripts\python.exe
vysledna cesta a hodnota
2575.2593208404182
[(76,30), (190,2), (85,168), (37,25), (12,181), (83,121), (108,187

Process finished with exit code 0
```



Príklad výberu na základe turnaja:
Program som spustila párkrát po sebe a dĺžky výsledných ciest sa pohybovali v priemere okolo 3280.



Rozdiel teda medzi dĺžkami ciest pri výbere turnaja a váženého priemeru je skoro až o 1000 odlišný, preto hodnotím vážený priemer na výber rodičov ako efektívnejší a vhodnejší.

SIMULOVANÉ ŽÍHANIE

Na začiatku programu si vytvorím súradnice všetkých miest. Vygenerujem si náhodné číslo od 0-200 a to pošlem do triedy Mesto, kde cez konštruktor priradím tieto hodnoty do x-ovej a y-ovej súradnice. Z triedy Mesto si tieto súradnice miest vrátim ako zoznam miest v tvare (x,y). Robím to rovnako ako v genetickom algoritme.

Tento zoznam miest si pošlem do funkcie simulovaneZihanie() odkiaľ najprv zavolám funkciu nahodnaPodstupnostMiestNaZaciatku(), v ktorej si tiež náhodne pre usporiadam tento zoznam miest.

Ďalej vyberám nasledovníkov z tohto pomixovaného zoznamu miest. Nasledovníkov si náhodne vytvorím tak, že im vymením dve susedné miesta vedľa seba a vytvorím ich toľko, koľko je počet miest.

Potom si náhodne si vygenerujem číslo od 0-počet miest a dané číslo bude index nasledovníka, ktorého skúsim vybrať. Pozriem sa, či je dĺžka cesty nasledovníka horšia ako bola dĺžka cesty predchádzajúcej cesty. Ak je horšia vypočítam pravdepodobnosť tak, že spočítam dĺžku cesty aktuálne a dĺžku cesty nasledovníka a vydelím teplotou. Tento výsledok dám do exponenciálnej funkcie.

```
pravdepodobnost = math.exp((dlzkaCestyZaciatok - dlzkaCestyNasledovnik) / teplota)
```

Teplotu som si na začiatku stanovila ako 1000. Normálne sa zvykne postupne znižovať o veľmi malé číslo napr. 0,01 ale aby som videla nejaké výraznejšie výsledky znížila som ju vždy o 10. Taktiež musím kontrolovať či neklesne pod 0, lebo nemôže byť záporná. Potom si vygenerujem nejaké náhodné číslo a ak je moja pravdepodobnosť menšia ako toto číslo, tak prejde do horšieho riešenia na toľko % koľko je pravdepodobnosť. Ak dĺžka cesty nasledovníka nie je horšie ale je lepšia ako dĺžka cesty aktuálneho, tak do nasledovníka prejde na 100%. Ak by náhodou nikdy neprešiel do horšieho nasledovníka a už prejde všetkých nasledovníkov, tak výsledná bude cesta počiatočná cesta, ktorú som dostala náhodným premiešaním zadaných miest. Toto celé robím vo for cykle toľkokrát, koľko je počet miest. Počas toho ešte hľadám úplne najkratšiu cestu z celého programu. Na konci viem porovnať túto úplne najkratšiu cestu s poslednou cestou, ktorá mi ostala ako výsledok z celého procesu simulovaného žíhania. V simulovanom žíhaní často prejde aj do horších riešení, a tak výsledné riešenie zo simulovaného žíhania nemusí byť najlepšie, najkratšie. Počas programu ale musím vchádzať a vyberať aj z horších riešení, pretože ak by som vyberala stále najlepšie riešenia uviazla by som v nejakom lokálnom extréme.

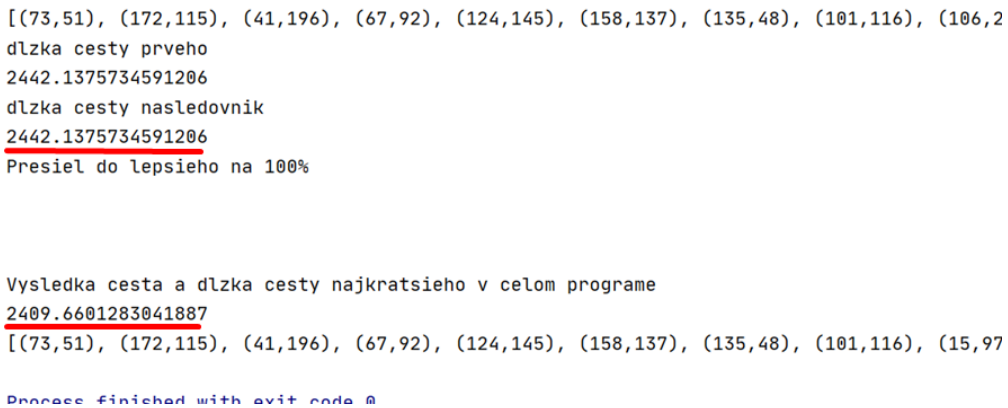
Zhodnotenie simulovaného žíhania:

Porovnávala som posledného aktuálneho nasledovníka, do ktorého program v cykle prešiel (výsledok môjho simulovaného žíhania) s úplne najkratšou možnosťou, ktorá v programe počas cyklu bola, ale neostal v nej, pretože prešiel do horšieho nasledovníka. Porovnala som teda ako veľmi sa tieto hodnoty od seba odlišujú.

```
posledný v cykle, ktorý mi ostal ako výsledok cyklu simulovaného žíhania:
[(5,193), (50,110), (3,118), (141,20), (49,141), (101,27), (181,83), (122,114), (4,64), (38
dlzka cesty prveho
4131.879706666631
dlzka cesty nasledovnik
4073.4331876934125
Presiel do lepsieho na 100%


najkratšia cesta, ktorú som zvažovala počas simulovaného žihania ale prešla som do horších
Vysledka cesta a dlzka cesty najkratsieho v celom programe
4010.6963430966075
[(5,193), (50,110), (3,118), (141,20), (49,141), (101,27), (181,83), (122,114), (4,64), (38

Process finished with exit code 0
```



Ako vidieť z predchádzajúcich 2 obrázkov rozdiel medzi výsledkom simulovaného žíhania a úplne najkratšej cesty v celom procese nie je viditeľne rozdielny. V prvom prípade je rozdiel ciest 63 a v druhom prípade je rozdiel 33. Zvolená cesta teda nebude úplne najhoršie možné riešenie ako sa tie mestá dajú prejsť a v celom programe sa blížia k najlepšiemu riešeniu alebo sa pohybujú v jeho okolí. Aby som neuviazla buď okolo lokálneho extrému alebo optimálneho riešenia, zvolila som možnosť vyberať aj najhorších aj najlepších nasledovníkov. Keďže vyberám aj najhorších nasledovníkov a vchádzam aj do horších nasledovníkov, nemôžem očakávať úplne najvhodnejšie riešenie ale tie cesty nie sú až tak odlišné od najvhodnejšej cesty, preto hodnotím moje simulované žíhanie ako úspešné. Zakázané prehľadávanie som nerobila.

ZHODNOTENIE A IMPLEMENTÁCIA:

Program som implementovala v jazyku Python a je spustiteľný. Prostredie, ktoré som použila je Pycharm (verzia 2020.2.3).

Celý program s genetickým algoritmom pracuje podľa mňa v poriadku, je to celkom efektívny algoritmus a v porovnaní so simulovaným žíhaním, ked som porovnávala lepší spôsob z genetického algoritmu (výber s váženým priemerom) obe riešenia boli efektívne. Ak porovnám, v ktorom prípade mi dávalo kratšie cesty bolo to pri genetickom algoritme. Často dávalo podobné hodnoty pre tie isté cesty aj pri použití simulovaného žíhania. Genetický algoritmus som sa navyše snažila vyladiť tým, že som jedincov zoradila podľa vážených priemerov. Možno by sa to dalo urobiť ešte efektívnejšie tým, že by som ich usporiadala podľa fitness hodnôt a vyberala stále tých, ktorí majú najlepšie fitness hodnoty zo všetkých. Ďalším možným vylepšením by mohlo byť pridanie ďalšej možnosti výberu jedincov na kríženie. Toto zadanie mi pomohlo s objasnením evolučných algoritmov a s hľadaním viacerých možností riešení a porovnávaním ich efektívnosti.