UMELÁ INTELIGENCIA

ZADANIE 2

TRAVELLING SALESMAN

Maximilián Strečanský
ID: 116298

ZNENIE 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.

OBCHODNÝ CESTUJÚCI

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.

TABU SEARCH

Prvým algoritmom na riešenie problému obchodného cestujúceho je tabu search (Zakázané prehľadávanie). Tento algoritmus funguje na princípe Tabu listu.

Tento zoznam obsahuje pozície mesta, ktoré sa nachádzajú v lokálnom extréme do ktorého sa dostaneme, ak neexistuje z jednej pozície sused s lepším ohodnotením. Takto si uložíme predchádzajúcu (lepšiu) pozíciu do zoznamu a už sa do neho nedostaneme. Tabu list v mojom riešení môže obsahovať najviac 10 takýchto pozícií, v prípade naplnenia, vyhodíme najstaršiu pozíciu.

SIMULATED ANNEALING

Druhým algoritmom je Simulované žíhanie. Tento algoritmus funguje na princípe lokálneho vylepšovania. To znamená, že z aktuálnej pozície si vytvorí nasledovníkov a chce sa dostať do najlepšie ohodnoteného.

Ak taký nasledovník neexistuje, môže prejsť do horšieho ale s pravdepodobnosťou menšou ako 100%. V mojom riešení je to šanca (Cena aktuálnej najlepšej pozície / cena aktuálnej horšej pozície * 2). Ak pozíciu odmietne, tak skúša ďalšieho nasledovníka. Ak sa mu nepodarí nájsť nasledovníka, algoritmus končí a aktuálny uzol je riešením.

PRINCÍPY ALGORITMOV

Oba algoritmy začnú s rovnakou náhodne vygenerovanou pozíciou a pokračujú 100 generáciami. Následníkov vytvárame z aktuálnej pozície pomocou permutácií.

Každá pozícia je uložená ako poradie očíslovaných miest, napríklad mestá so súradnicami (10, 5), (3, 7), (4, 2), (1, 6), by mali prvotné očíslovanie 0, 1, 2, 3, čo znamená, že z mesta (10, 5) ideme do (3, 7) a následne do (4, 2) a napokon do (1, 6). Ak zmeníme pozíciu jedného čísla, zmení sa nám aj poradie ciest. Takéto permutácie využívam na ukladanie jednotlivých ciest.

Susedov generujeme pomocou všetkých kombinácií jedného zamenenia dvoch čísel. Pre náš príklad by boli potomkovia o veľkosti 6. Vygenerovanie susedov voláme vo funkcií *generateNeighborhood()*.

Následne pre všetkých vytvorených susedov zavoláme funkciu *calculateState()*, ktorá nám vráti ohodnotenie stavu. Susedov si ukladáme do poľa, posledným údajom je stav jednotlivých susedov. Pole si zoradíme od najlepšieho po najhorší a vrátime ho algoritmu.

Pracovaním s permutáciami sa vyhneme prácou s dvojrozmernými poľami a dokážeme takto jednoduchšie určiť poradie miest.

ROZPOZNANIE ÚDAJOV

COORDINATES = []
- Sem ukladáme súradnice jednotlivých miest.

INIT_PERMUTATION = []
- Slúži na uloženie prvotnej náhodne vygenerovanej permutácie.

INIT_PERMUTATION_VALUE = []
- Slúži na uloženie ohodnotenia prvotnej permutácie.

PERMUTATIONS = []
- Sem uložíme čísla miest s ktorými pracujeme, pri spustení programu sa podľa počtu miest naplní od 0 po X.

FITNESS = []
- Pole ohodnotení generácií algoritmov. Využívame na vizualizáciu práce programu.

FITNESS_LENGTH = []
- Ukladá počet prejdených generácií algoritmov pre vizualizáciu.

SIZE
- Určujeme veľkosť chceného mesta

ZAKÁZANÉ PREHĽADÁVANIE – ZHODNOTENIE

Výhodami tohoto algoritmu je minimálne opakovanie stavov. Pri správnej voľbe veľkosti tabu listu docielime, že sa nedostaneme do rovnakého uzla a oproti simulovanému žíhaniu preskúmame viacero stavov a potomkov.

V určitých prípadoch sa pomocou tabu search nedokážeme dostať do uzlov, ktoré by následne smerovali k najlepšiemu možnému riešeniu. Toto je zapríčinené spôsobom fungovania tohoto algoritmu, keďže vyberáme potomka s najkratšou cestou a tabu list nás limituje veľkosťou.

ZAKÁZANÉ PREHĽADÁVANIE – DĹŽKA ZOZNAMU

Zvolením správnej dĺžky pri Tabu search dokážeme urýchliť hľadanie nakoľko skracujeme prehľadávanie prvkov v tomto zozname. Zvolením krátkej dĺžky zoznamu, môže algoritmus opakovane prehľadávať rovnaké stavy.

SIMULOVANÉ ŽÍHANIE - ZHODNOTENIE

Hlavnou výhodou tohto algoritmu oproti tabu je, že má väčšiu pravdepodobnosť na dosiahnutie lepšieho výsledku. Keďže horší nasledovník nemá garantované aby bol zvolený, môžeme sa tak dostať do potomka, ktorý je napríklad z aktuálnych najhorší, ale po desiatich generáciách dosiahne najlepšieho možného potomka.

Nevýhodou je, že vo veľa prípadoch sa potomkovia dookola vymieňajú, to sa stane v prípade, že z jednej pozície vyberieme najlepšieho potomka a pri vygenerovaní ďalšej generácie sa dostaneme do prvotného uzla. Takto sa vieme zacykliť a algoritmus môže mať malú šancu na odcyklenie.

SIMULOVANÉ ŽÍHANIE – ZVOLENIE ROZVRHU

Pri simulovanom žíhaní sa stretneme s problémom zvolenia správnej metódy pravdepodobnosti pri horších potomkoch. V mojom riešení používam:

*(Cena aktuálnej najlepšej pozície / cena aktuálnej horšej pozície * 2)*

S týmto riešením môžeme mať veľmi vysoké percentuálne šance zlých potomkov. Nakoľko môže nastať situácia kde je cena aktuálnej najlepšej pozície 100 a cena aktuálnej horšej pozície 101, čo nám dá percentuálnu šancu 99%. Pri prvotnom riešení bol algoritmus vo väčšine prípadov zacyklený a preto používam násobenie dvojkou pre nižšiu šancu takýchto prípadov.

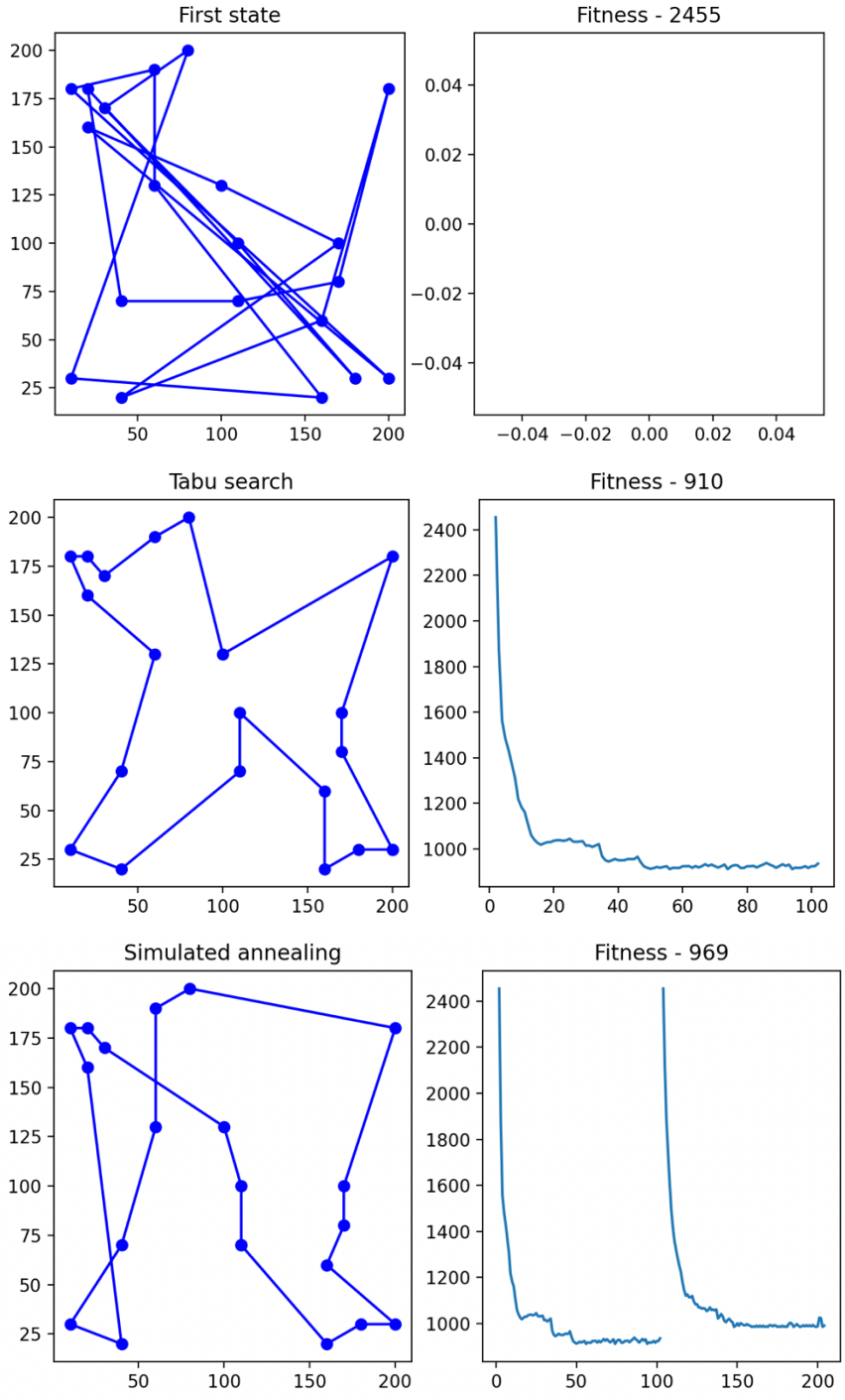
Nakoľko sú potomkovia zoradení od najmenšej cesty po najväčšiu, dosiahneme znižovanie šance pre horších potomkov.

GRAFICKÉ ROZHRANIE

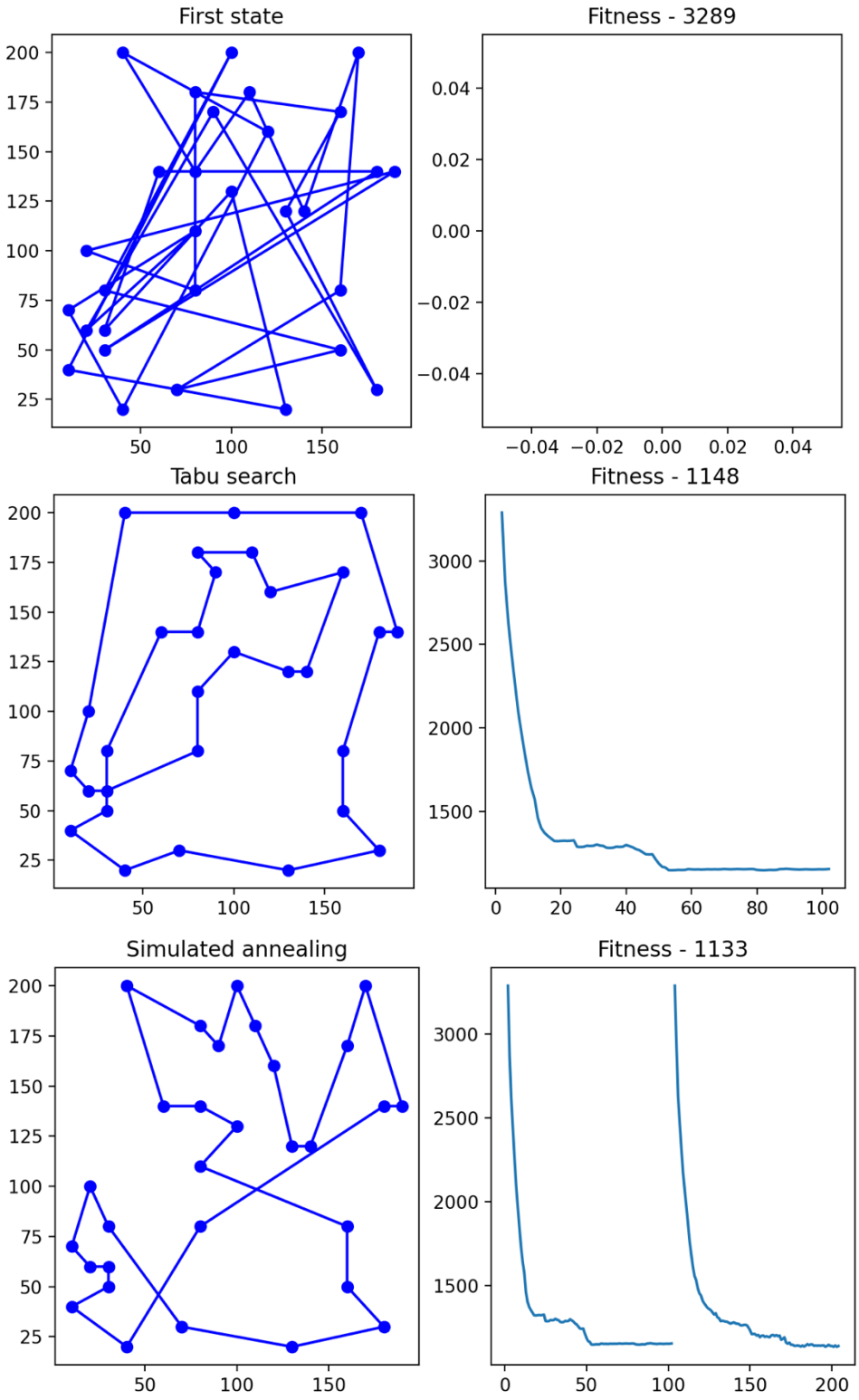
Pre vizualizáciu využívam knižnicu *matplotlib*. Po spustení programu sa najprv vykoná Tabu search. Pre každú generáciu vykreslíme aktuálny zvolený stav a jeho fitness. Takto môžeme pekne vidieť pracovanie algoritmu ako aj klesajúcu dĺžku cesty. Po sto generáciách sa zobrazí uzol s najkratšou dĺžkou.

Po ukončení zopakujeme rovnaký proces pre Simulated annealing. Na pravom grafe môžeme vidieť postup oboch algoritmov, ako aj porovnanie ich hladania.

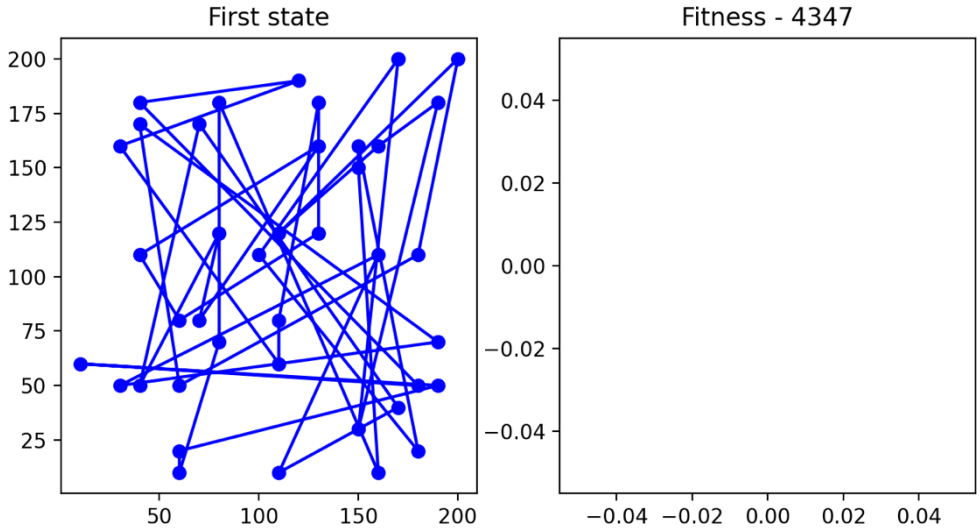
PRÍKLADY NA POROVNANIE ALGORITMOV

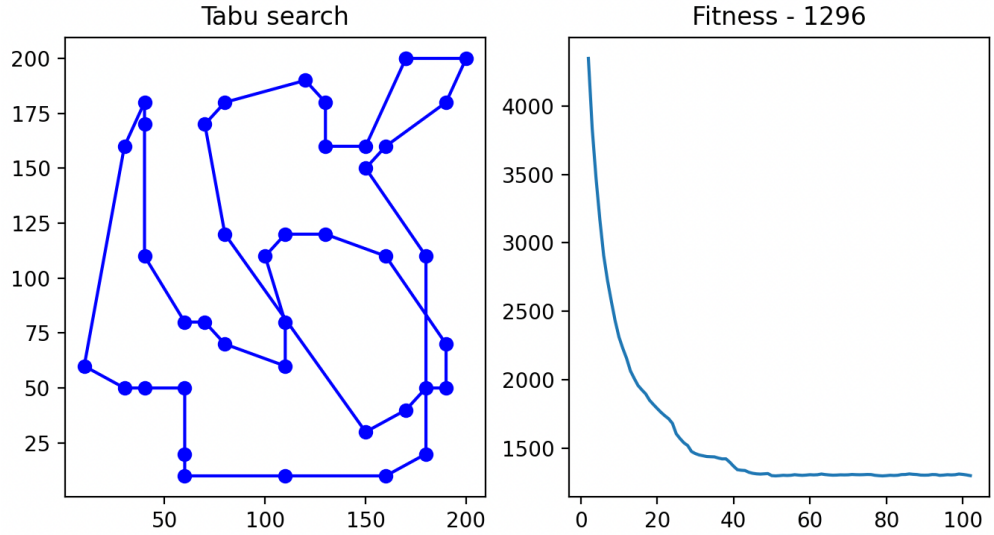


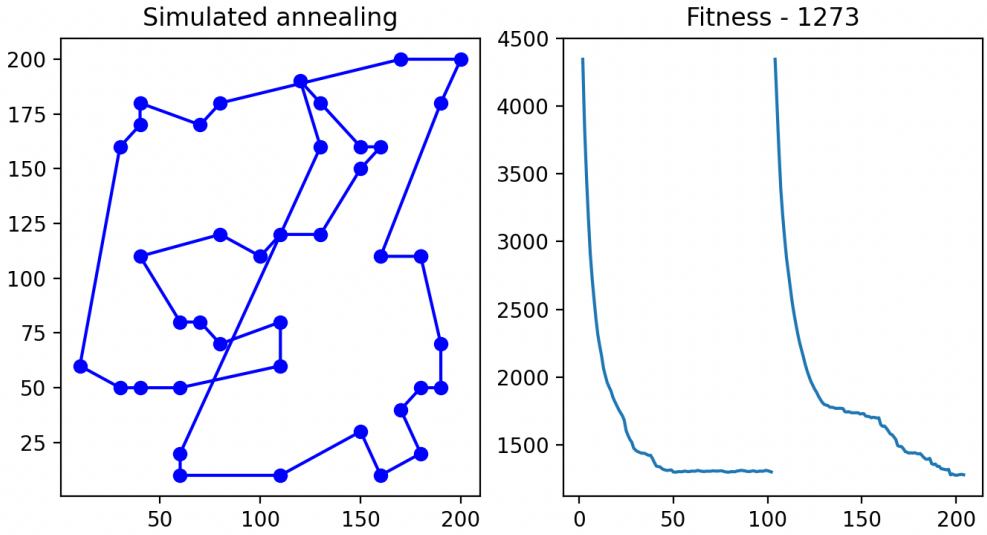
V tomto prípade náhodne vygenerovaných 20 miest môžeme vidieť ako tabu search dokázal nájsť z rovnakej pozície efektívnejšiu cestu. V cca 50. generácií sa dostal na najlepšiu pozíciu aká je na obrázku a následne sa pohyboval +-15 od tejto pozície. Pri druhom algoritme môžeme vidieť rovnaké opakovanie hodnôt, čo nám indikuje opakovanie stavov.



Pri druhom prípade vygenerovaných 30 náhodných miest, môžeme vidieť, že zvíťazilo simulované žíhanie. Krivka fitness pri zakázanom hľadaní dosiahla iba v jednom bode radikálne zlepšenie a následne ostala pri rovnakých hodnotách. Simulované žíhanie dosiahlo zlepšenie stavu vo väčšine stavov nakoľko sa krivka viac podobá exponenciálnemu klesaniu.







Vo finálnom prípade vygenerovaných 40 náhodných miest, môžeme znova vidieť, ako zvíťazilo simulované žíhanie. Podľa grafu usudzujem, že toto bol jeden z prípadov kedy sa najlepšia možnosť nachádzala v neskorších susedoch. Nakoľko je počiatočná pozícia oboch grafov rovnaká.

ZÁVER

Cieľom tohoto zadania bolo aplikovať metódy strojového učenia na problém obchodného cestujúceho, kde sa snažíme nájsť najkratšiu cestu medzi množinou miest.

Mojou úlohou bolo implementovať algoritmus simulovaného žíhania a zakázaného prehľadávania. Zistil som, že každý z algoritmov má svoje výhody a nevýhody. Program som naprogramoval v *pythone*, kvôli knižnici *matplotlib*, ktorá mi umožňuje jednoducho vizualizovať jednotlivé algoritmy.