Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Umelá inteligencia – zadanie č. 2

Strojové učenie sa – Problém obchodného cestujúceho pomocou simulovaného žíhania a zakázaného prehľadávania

Michaela Gubovská

Cvičiaci: Ing. Ivan Kapustík

Akad.rok 2020/21

Michaela Gubovská
ID: 102925
Streda 13:00

Zadanie

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.

V prvej časti dokumentácie sa venujem riešeniu zadaného problému pomocou **simulovaného žíhania**.

V druhej časti sa venujem riešeniu pomocou algoritmu **zakázaného prehľadávania**.

Michaela Gubovská
ID: 102925
Streda 13:00

SIMULOVANÉ ŽÍHANIE

Simulované žíhanie predstavuje optimalizačný algoritmus, ktorý sa snaží nájsť globálne optimum pre zadaný problém za pomoci vyhýbania sa lokálnym extrémom. Pre jeho najefektívnejšiu funkčnosť je dôležité nastaviť parametre teploty a chladenia teploty tak, aby program dokázal s vysokou pravdepodobnosťou nájsť globálne optimum, alebo sa k nemu blízko priblížil.

1. Použitý algoritmus

Algoritmus lokálne prehľadáva permutácie postupnosti navštívenia na základe náhodnosti a postupného znižovania teploty. Na začiatku je ako riešenie (najlepšia permutácia) problému zvolená náhodná permutácia. Pre ňu je potom pokým sa teplota nezníži na nulu robené lokálne prehľadávanie v cykle so 100*počet_miest opakovaniami.

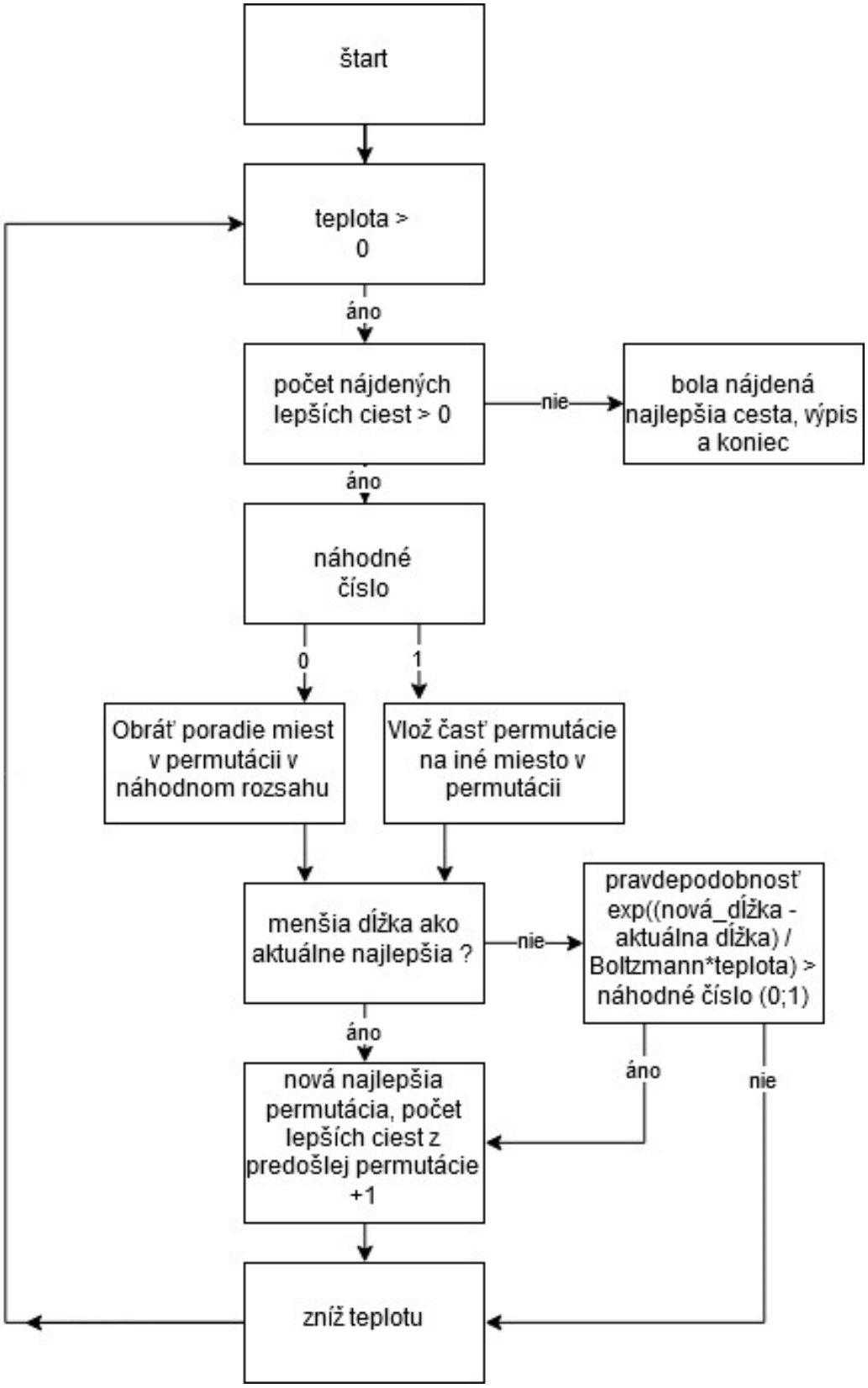
Na základe náhodného vyhodnotenia sa v každom cykle vytvorí nová permutácia z tej aktuálnej. Buď sa zvolí reverzné poradie nejakej časti permutácie, alebo vloženie časti permutácie na iné miesto v permutácii. Takto modifikovaná permutácia sa stáva novým kandidátom na najlepšie riešenie. Vypočíta sa preto dĺžka cesty tejto permutácie. Pokiaľ je dĺžka kratšia ako tá aktuálna, vždy túto novú permutáciu akceptujeme za novú najlepšiu a ďalej hľadáme v rovnakom poradí ďalšie permutácie odvodené od nej. Pokiaľ permutácia nebula lepšia ako aktuálna, nastáva ďalší náhodný krok. Porovnáme náhodne vygenerované číslo od 0 po 1 a pravdepodobnosťou výberu tejto horšej permutácie. Týmto predchádzame uviaznutiu v lokálnom extréme pretože dávame programu možnosť akceptovania horšieho riešenia. Pravdepodobnosť, že akceptujeme toto horšie riešenie je priamo úmerná teplote. Čím vyššia je teplota, tým je pravdepodobnosť výberu tohto riešenia vyššia. Preto na začiatku s vysokou teplotou akceptujeme viac horších riešení, aby sme sa vyhli lokálnom extrémom a s postupne znižujúcou sa teplotou akceptujeme stále menej horších riešení, až kým nedosiahneme optimálne riešenie, alebo sa mu nepriblížime.

Teplota sa znižuje na konci cyklu, po vyskúšaní 100*N permutácií. Tak isto obsahuje algoritmus optimalizáciu časovej náročnosti. Pokiaľ sme totiž v cykle narazili už aspoň na 10 lepších riešení, z cyklu vyjdeme a ďalej pracujeme s poslednou najlepšou permutáciou. Ušetrí to čas a máme zaručené, že sme sa posunuli do nového riešenia problému už so zníženou teplotou.

Pri nájdení novej permutácie sa navyše kontroluje, či sme ju ešte v predošlých behoch neskúšali. Permutácia sa zahashuje a jej hodnota sa porovná s hodnotami v sete navstivene(). Ak sa nejaká hodnota zhoduje, permutáciu sme už v riešení našli a preskočíme ju.

Najlepšie riešenie sme našli, pokiaľ sme v cykle nenašli žiadne lepšie riešenie pre aktuálnu permutáciu. Túto permutáciu vyhlásime za finálnu cestu a vypíšeme ju.

Michaela Gubovská
ID: 102925
Streda 13:00



Michaela Gubovská
ID: 102925
Streda 13:00

2. Reprezentácia údajov

Permutácie poradia navštívených miest sú uložené v obyčajnom globálnom poli **cities[]**. Napríklad poradie navštívenia miest môže pre 4 mestá vyzerať nasledovne: cities[1, 0, 2, 3]. V programe sa pracuje s mestami ako číslami z rozsahu <0; N), kde N = počet miest. Na konci pri výpise sa inkrementuje číslo mesta, aby bolo očíslovanie začaté od 1. Keďže sa jedná o lokálny algoritmus, nové permutácie sa generujú „za chodu" a ukladajú sa iba do poľa navštívených.

Vzdialenosti medzi mestami sú uložené v globálnom 2D poli **distances[[]],** kde je na indexe *i* uložená vzdialenosť mesta *i* od miest na indexoch *j*. Teda napríklad vzdialenosť medzi mestom 0 a mestom 3 získame takto: distances[0][3]. Vzdialenosti sú vypočítané pomocou Pytagorovej vety cez Euklidovu vzdialenosť vo funkcii *calculate_distance().*

Prehľadané permutácie sú uložené v globálnom sete **visited[]**, aby sa predišlo opätovnému prehľadávaniu nejakej permutácie. Použila som dátovú štruktúru *set* nakoľko mi nezáleží na poradí permutácii v nej a takisto je prehľadávanie tejto štruktúry zložité O(1), čo značne šetrí čas behu programu. Pre šetrenie pamäte a efektívnejšie prehľadávanie sa v tomto sete nachádzajú **zahashované permutácie**, pomocou jednoduchej hash funkcie, ktorá z čísla každého mesta spraví char, ktorý zaberie iba 1B pamäte a vloží ho do stringu. Každá permutácia má tak unikátny hash a vieme ľahko skontrolovať, či sme ju už v riešení skúšali, alebo nie.

Počiatočná teplota je reprezentovaná globálnou premennou **temperature**. Je nastavená na 0.5, nakoľko odporúčaná teplota pri začiatku algoritmu je medzi 40 až 60%. Postupne sa v programe znižuje podľa toho, aký koeficient chladenia nastavím. Svoj program som skúšala na chladení o 2%, 10% a 40%.

3. Zhodnotenie vlastností a ich závislosť na rozvrhu

Hľadanie riešenia pomocou simulovaného žíhania závisí od nastavených parametrov. Pre maximalizovanie efektívnosti algoritmu je najvhodnejšie nastaviť chladenie teploty v rozsahu od 10% až 1%, pretože sa tým zohľadňuje dostatok horších riešení v procese, vďaka ktorým sa program dostane skôr ku globálnemu optimu.

Z testovania vyplynulo, že simulované žíhanie dokáže vo väčšine prípadov nájsť optimálne riešenie problému, alebo sa mu veľmi približuje. Občas sa však môže stať, že nájdené riešenie nie je výhodné, nakoľko algoritmus z veľkej časti funguje na náhodnosti. Na menšej mape sa ukázalo, že algoritmus dokáže nájsť až v tretine prípadov najoptimálnejšie riešenie.

Rozsah nájdených riešení závisí od dĺžky cesty pôvodnej permutácie, ktorá vojde do funkcie ako prvá.

Celkovo pre simulované žíhanie platí, že čím pomalšie sa ochladzuje teplota, tým rýchlejšie vie program nájsť riešenie, pretože povoľuje viacero horších medzikrokov. Tieto vlastnosti sú vidno na grafoch nižšie. Tak isto pri vyššej pôvodnej teplote je pravdepodobnosť nájdenia optimálneho riešenia vyššia, nakoľko je vyššia pravdepodobnosť akceptovania horšieho riešenia, čo umožní skoršie nájdenie optima v zadanom množstve cyklov hľadania. Rovnako sa pravdepodobnosť nájdenia optimálneho riešenia zvyšuje počtom cyklov, v ktorých program hľadá riešenie. Má totiž viac času (behov programu) na uniknutie lokálnym optimám.

Michaela Gubovská
ID: 102925
Streda 13:00

3.1 Testovanie

N – počet miest

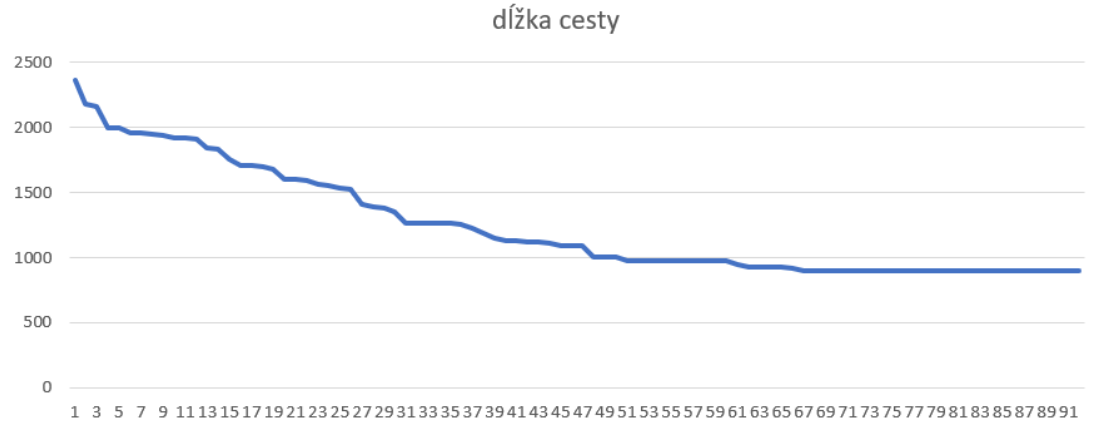
Počet cyklov	Počet miest	Teplota (pôvodná)	Ochladzovanie	Úspešnosť nájdenia glob. optima	Min. dĺžka cesty	Max. dĺžka cesty
100*N	4	0.5	10%	35%	231.5	289.1
100*N	20	0.5	40%	6%	895.7	1106.3
100*N	4	0.7	10%	45%	231.5	289.1
100*N	20	0.5	10%	6%	895.7	966.7
100*N	20	0.7	10%	19%	895.7	1989.5
10*N	20	0.7	10%	6%	895.7	2166.4

Nasledujúce grafy zobrazujú vývoj dĺžky cesty nájdenej algoritmom po cykloch.

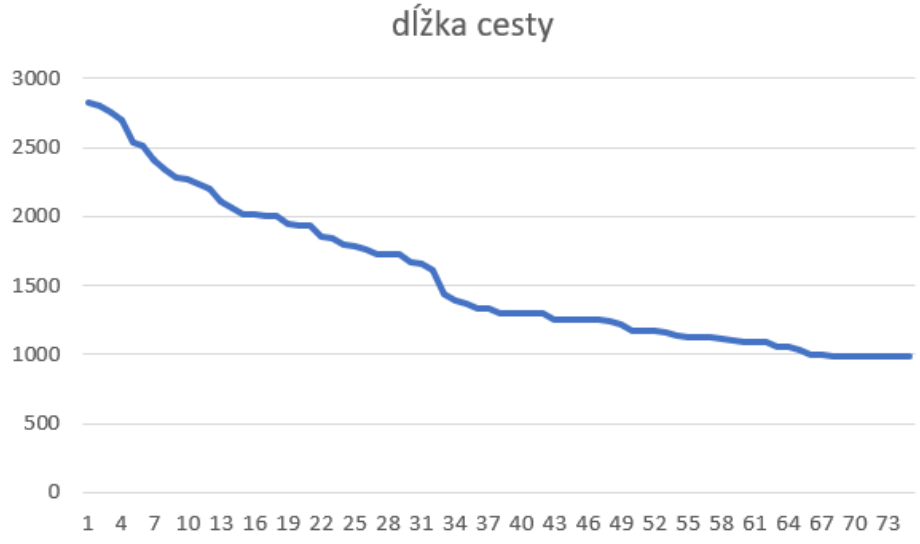
Obrázok 1: počiatočná teplota 0.5, 2% chladenie, 20 miest

Michaela Gubovská
ID: 102925
Streda 13:00

Obrázok 2: počiatočná teplota 0.5, 10% chladenie, 20 miest



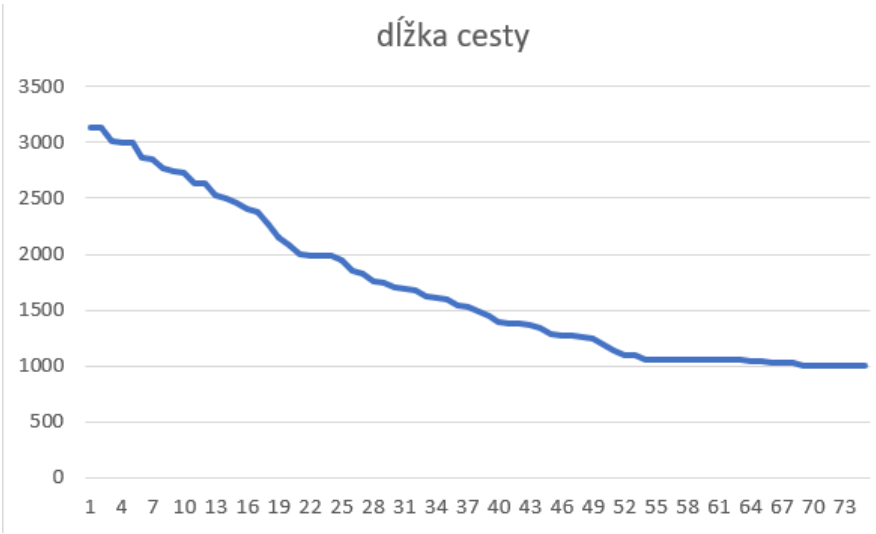
Obrázok 3: počiatočná teplota 0.5, 40% chladenie, 20 miest



Obrázok 4: počiatočná teplota 0.5, 2% chladenie, 30 miest

Michaela Gubovská
ID: 102925
Streda 13:00

Obrázok 5: počiatočná teplota 0.5, 10% chladenie, 30 miest



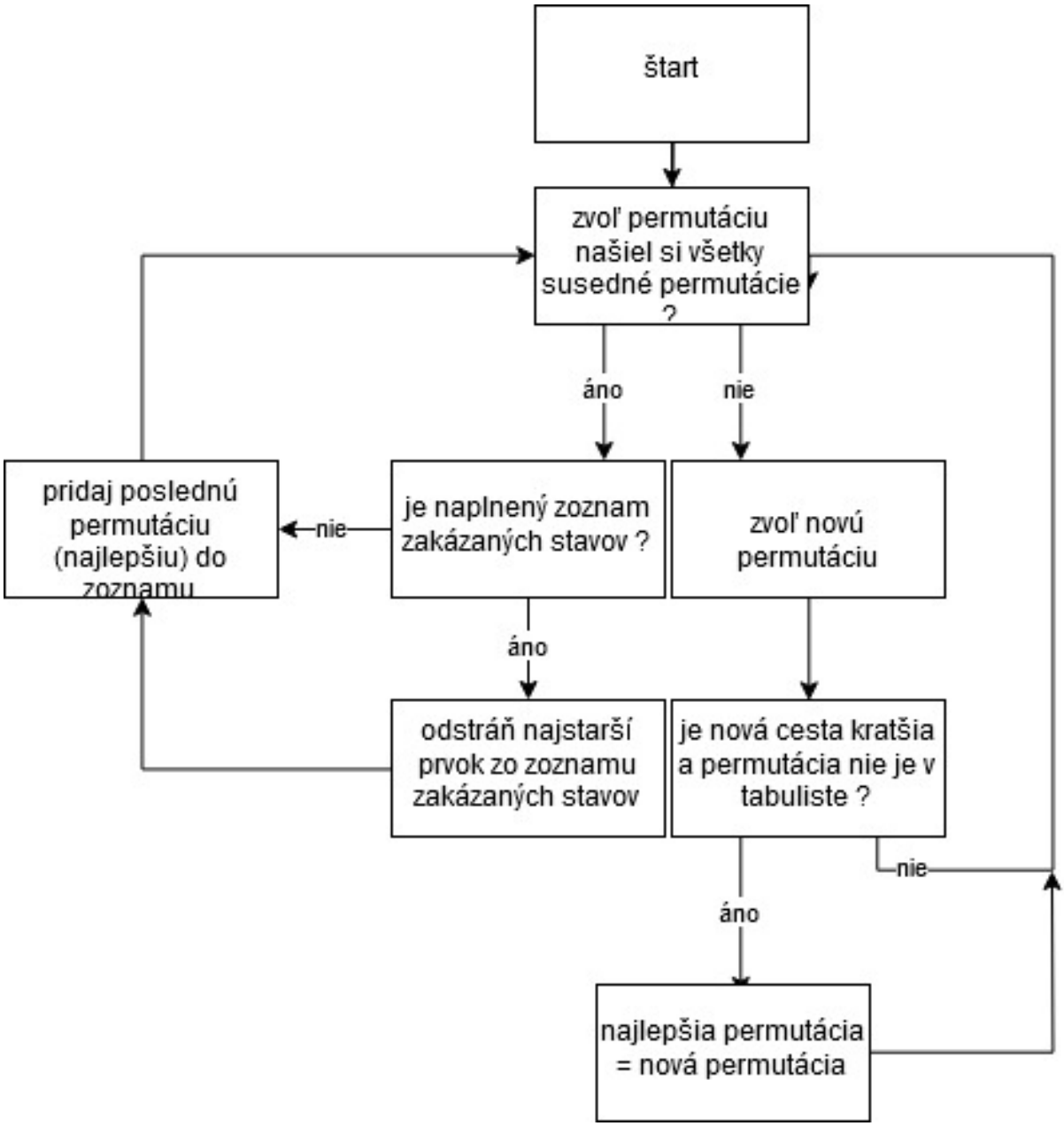
Obrázok 6: počiatočná teplota 0.5, 40% chladenie, 30 miest

Michaela Gubovská
ID: 102925
Streda 13:00

Zakázané prehľadávanie

1. Použitý algoritmus

Zakázané prehľadávanie je optimalizáciou hill climbing algoritmu. Využíva sa pri ňom štruktúra, ktorá si dočasne pamätá stavy, ktoré boli prehľadané a v danom cykle najvýhodnejšie. Do týchto stavov potom program nemôže vojsť v ďalších niekoľko behoch, čím sa vyhne zaseknutiu v lokálnom optime. Štruktúra tohto zoznamu sa mení až po jeho naplnení, kedy sa z neho odstráni najstarší prvok. Môj program funguje nasledovne. Na začiatku sa zvolí náhodná permutácia, ktorá je označená za najlepšiu. Pre túto a každú ďalšiu permutáciu sa vytvorí N „susedných" permutácií a z nich sa vyberie tá najvýhodnejšia. Keď nájdeme takú, ktorá predstavuje kratšiu cestu ako dovtedy najlepšia cesta a súčasne sa nenachádza v zozname zakázaných stavov, prehlásime ju za najlepšiu. Takto pokračujeme zvolený počet ráz a po doiterovaní hlavného cyklu vypíšeme permutáciu, ktorá bola ako posledná vyhlásená za najlepšiu.



Michaela Gubovská
ID: 102925
Streda 13:00

2. Reprezentácia údajov

Permutácie poradia navštívených miest sú uložené v obyčajnom globálnom poli **cities[]**. Napríklad poradie navštívenia miest môže pre 4 mestá vyzerať nasledovne: cities[1, 0, 2, 3]. V programe sa pracuje s mestami ako číslami z rozsahu <0; N), kde N = počet miest. Na konci pri výpise sa inkrementuje číslo mesta, aby bolo očíslovanie začaté od 1. Keďže sa jedná o lokálny algoritmus, nové permutácie sa generujú „za chodu" a ukladajú sa iba do poľa navštívených.

Vzdialenosti medzi mestami sú uložené v globálnom 2D poli **distances[[]],** kde je na indexe *i* uložená vzdialenosť mesta *i* od miest na indexoch *j*. Teda napríklad vzdialenosť medzi mestom 0 a mestom 3 získame takto: distances[0][3]. Vzdialenosti sú vypočítané pomocou Pytagorovej vety cez Euklidovu vzdialenosť vo funkcii *calculate_distance().*

Zoznam zakázaných stavov predstavuje jednorozmerné pole **tabulist[]** do ktorého sa postupne ukladajú zakódované stavy v podobe stringov. Zoznam môže mať rôzne dĺžky definované užívateľom.

3. Zhodnotenie vlastností a ich závislosť na dĺžke zakázaného zoznamu

Pri zakázanom prehľadávaní je dôležité správne zvolenie metódy mutovania permutácie a veľkosti zoznamu zakázaných stavov. Z testovania vyplynulo, že pre nájdenie globálneho optima je napríklad vhodná forma mutácie obrátenie náhodného úseku v permutácii.

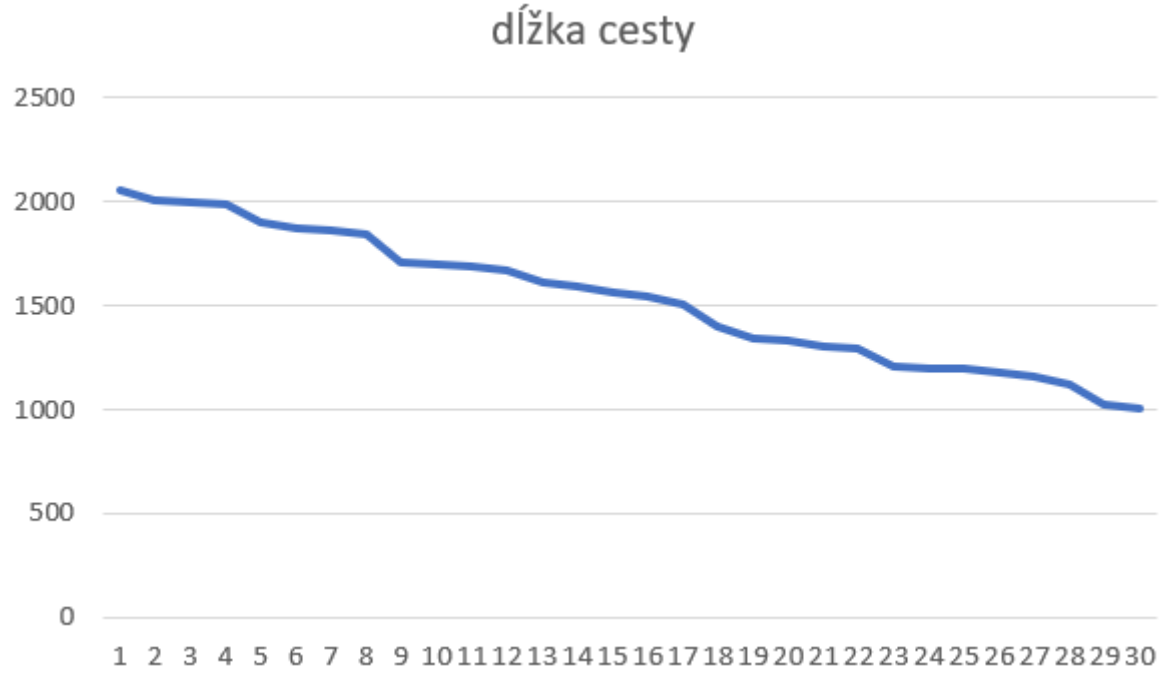
Vo všeobecnosti platí, že čím väčší je zoznam zakázaných stavov, tým plynulejšie program hľadá globálne minimum a s vyššou pravdepodobnosťou ho nájde. Samozrejme, tieto vlastnosti sú premenlivé, nakoľko je prvá permutácia, ktorej susedné permutácie sú prehľadané, náhodná.

3.1 Testovanie

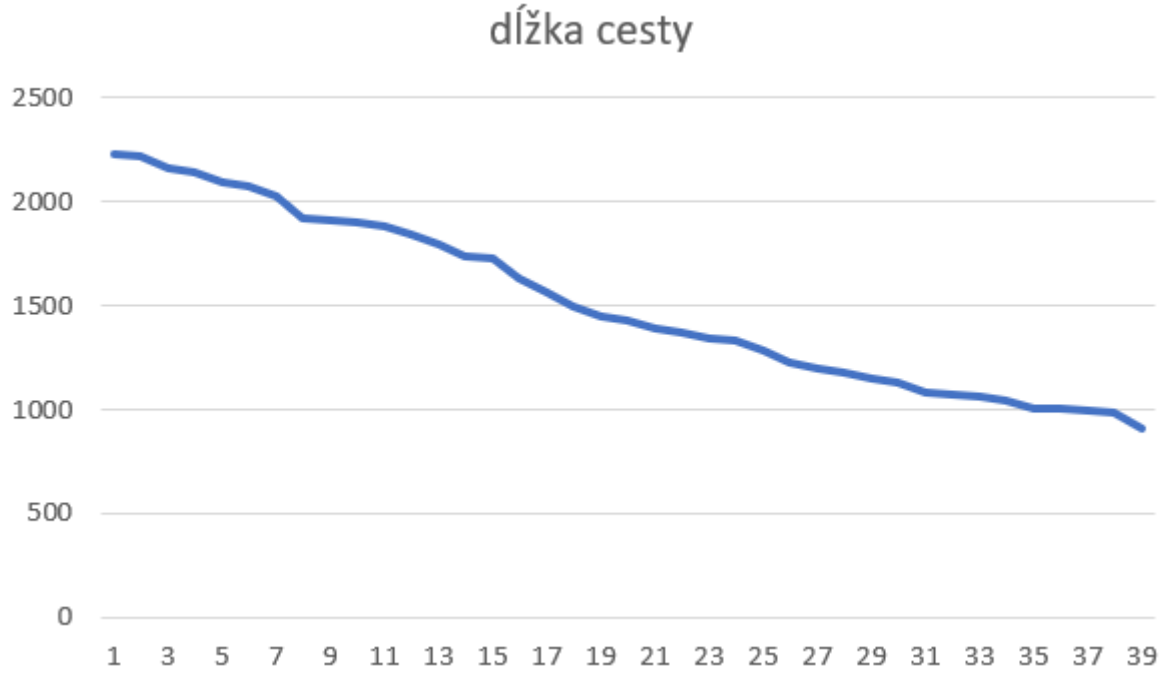
N – počet miest

Počet cyklov	Počet miest	Veľkosť zoznamu	Mutovanie	Úspešnosť nájdenia glob. optima	Min. dĺžka cesty	Max. dĺžka cesty
100	4	2N	Reverz časti permutácie	100%	231.5	231.5
100	20	2N	Reverz časti permutácie	6%	895.7	1106.3
1000	4	3N	Reverz časti permutácie	100%	231.5	231.5
1000	20	3N	Reverz časti permutácie	14%	895.7	1026.5
100	4	2N	Výmena dvoch prvkov	36%	231.5	289.1
100	20	2N	Výmena dvoch prvkov	0%	1835.5	2820.8

Michaela Gubovská
ID: 102925
Streda 13:00

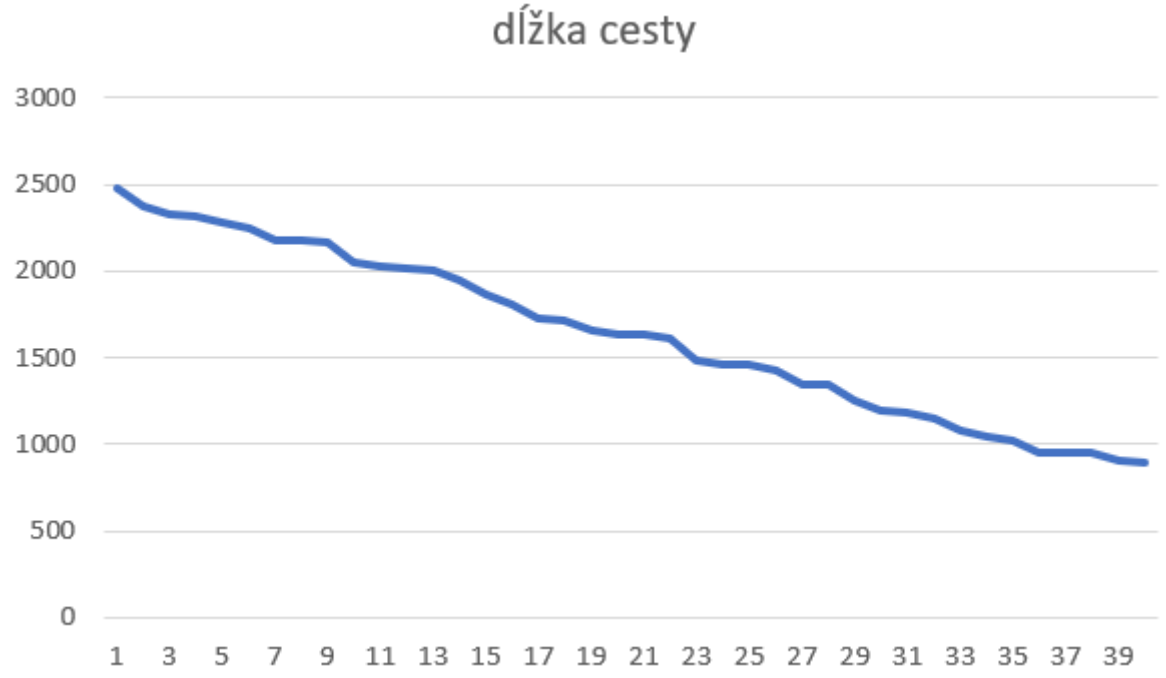


Obrázok 7: veľkosť tabuľky N, N = 20 miest

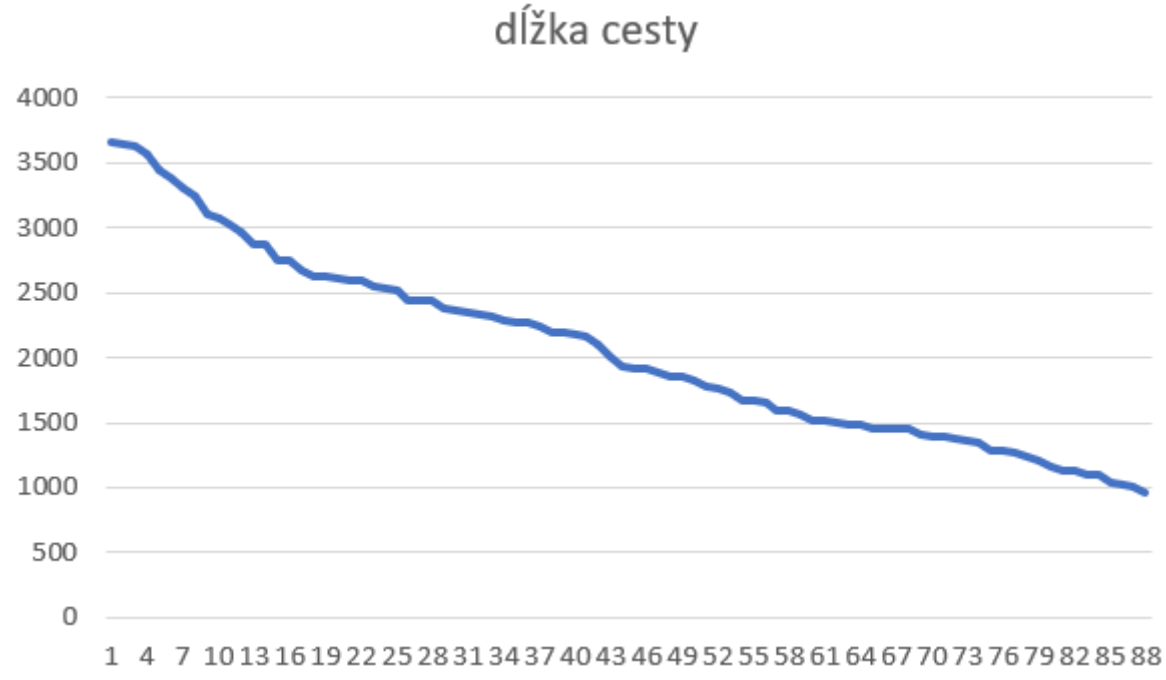


*Obrázok 8: veľkosť tabuľky 2*N, N = 20 miest*

Michaela Gubovská
ID: 102925
Streda 13:00



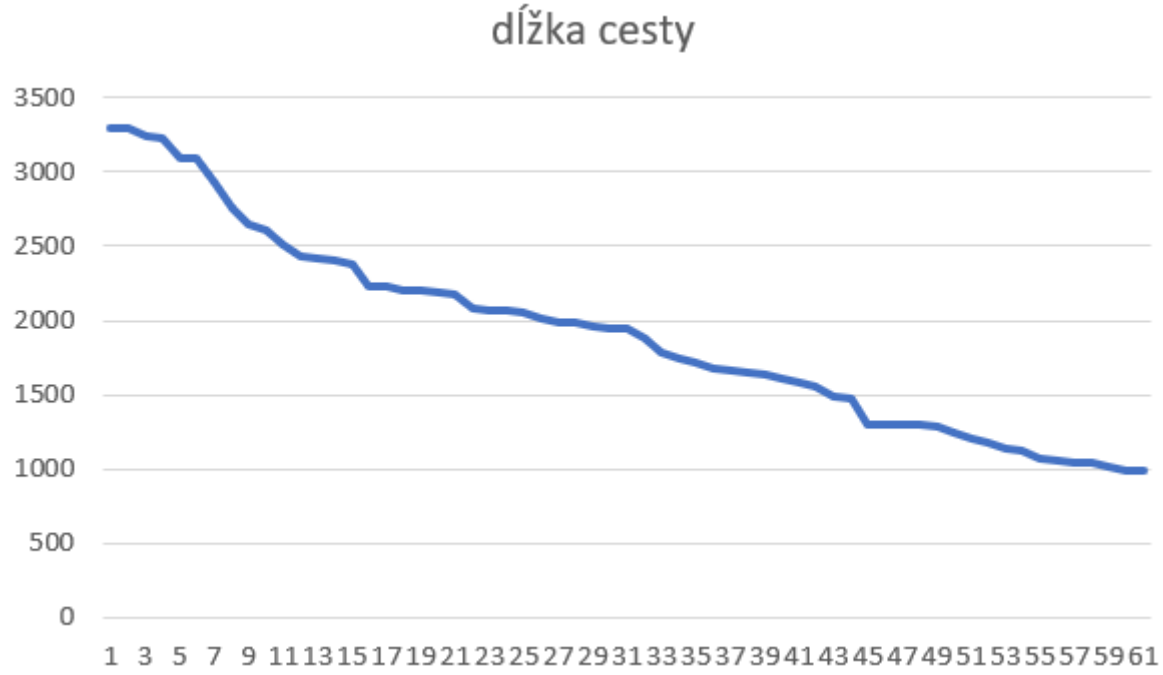
*Obrázok 9: veľkosť tabuľky 3*N, N = 20 miest*



Obrázok 10: veľkosť tabuľky N, N = 30 miest

Michaela Gubovská
ID: 102925
Streda 13:00

*Obrázok 11: veľkosť tabuľky 2*N, N = 30 miest*



*Obrázok 12: veľkosť tabuľky 3*N, N = 30 miest*

Michaela Gubovská
ID: 102925
Streda 13:00

Zhodnotenie

Správne som implementovala oba algoritmy a počas testovania som došla k záveru, že optimálnejším algoritmom lokálneho prehľadávania stavov je simulované žíhanie. Dosahovalo lepšie výsledky, čo sa týka počtu nájdení globálneho optima pre zadaný problém na rovnakých mapách.

V implementácii zakázaného prehľadávania je priestor pre optimalizáciu. Napríklad je možné implementovať pamäť na frekvenciu permutácií, aby pri zložitejších problémoch program vedel, ktoré permutácie už viackrát skúšal a nikdy nevybral za tú lepšiu možnosť. Tieto by mohol v prípade cyklenia vyberať z pamäte a skúšať nájsť ich susedné permutácie, aby vyviazol z cyklu a naďalej hľadal globálne optimum.

Navigácia v programe

Po spustení sa v konzole vypíše možnosť zadania .txt súboru, ktorý obsahuje súradnice miest. Na výber sú 3 mapy, 1.txt – 20 miest, 2.txt – 4 mestá, 3.txt – 30 miest. Po správnom zadaní čísla súboru sa vypíše výzva na zadanie algoritmu, ktorým chceme nájsť riešenie. Na výber je „zihanie“ alebo „tabu“. Pri vyžiadaní zakázaného prehľadávania sa objaví možnosť zadania veľkosti zoznamu zakázaných stavov. Po zadaní správneho vstupu sa spustí program a po ukončení vypíše výslednú postupnosť miest a dĺžku danej cesty.

```
Zadaj cislo suboru - 1-13
Pre ukoncenie programu napis 'koniec'
1
Zadaj algoritmus - 'zihanie' = simulovane zihanie
'tabu' = tabu search
zihanie
Najdena postupnost navstivenia miest:
14 17 20 13 16 19 18 15 12 9 5 1 3 6 10 8 4 2 7 11

Dlzka vyslednej cesty:  895.7063250228808
```

Implementačné prostredie

Program je implementovaný v jazyku Python, verzia 3.7 na platforme Windows 10 Pro. Pracovala som v prostredí Pycharm Professional 2020.1.3. Využila som knižnice **random** (pre náhodné premenné), **math** (výpočet vzdialenosti medzi mestami pomocou sqrt()), **scipy** (Boltzmannova konštanta vo vzorci akceptovania horšieho riešenia).