Umelá inteligencia:  
Problém obchodného cestujúceho

**Marián Kurčina**

**ZS 2024**

[xkurcinam@stuba.sk](mailto:xkurcinam@stuba.sk)

[marian.kurcina2003@gmail.com](mailto:marian.kurcina2003@gmail.com)

Table of Contents

[Zakázané prehľadávanie (tabu search) 3](#_Toc189418761)

[Task 3](#_Toc189418762)

[Opis riešenia problému 3](#_Toc189418763)

[Opis kódu 3](#_Toc189418764)

[Conclusion 5](#_Toc189418765)

[Simulované žíhanie (simulated annealing) 6](#_Toc189418766)

[Task 6](#_Toc189418767)

[Opis riešenia problému 6](#_Toc189418768)

[Opis kódu 6](#_Toc189418769)

[Conclusion 8](#_Toc189418770)

# Zakázané prehľadávanie (tabu search)

## Task

Obchodný cestujúci má navštíviť viacero miest, pričom chce prejsť cez všetky mestá, cez každé práve raz a na konci chce skončiť v meste v ktorom začínal. Chce nájsť čo najkratšiu trasu.

Je daných 30 a 40 miest, každé majú náhodne vygenerované súradnice. Rozmer mapy je 500x500. Cena cesty medzi dvoma mestami zodpovedá Euklidovej vzdialenosti – vypočíta sa pomocou Pytagorovej vety. Cieľom je nájsť take poradie, pre ktoré bude mať celková trasa čo najmenšiu dĺžku. Výstupom je poradie miest a dĺžka zodpovedajúcej cesty.Zadanie je potrebné riešiť pomocou zakázaného prehľadávania.

## Opis riešenia problému

Cesta bude uložená vo forme zoznamu reprezentujúceho poradie miest, ktoré navštívim, pričom každé z miest bude v zozname iba raz (posledný krok do začínajúceho mesta nebude v zozname opísaný). Pri hľadaní budem vytvárať nasledovníkov aktuálnej cesty a vyberiem z nich tú najkratšiu, ktorá nie je v tabu zozname. Táto cesta sa stane mojou novou cestou. Ak je táto cesta dlhšia ako aktuálna cesta, vložím aktuálnu cestu do tabu zoznamu aby som sa dokázal dostať z lokálneho extrému. Po niekolkých iteráciách s rovnakou najlepšou cestou vypíšem a vykreslím najlepšiu cestu. Pre testovanie programu budem používať za každým rovnaké rozpoloženie miest, rozpoloženie použité na testovanie je zakomentované v kóde.

## Opis kódu

Na začiatku programu mám uložené dáta o mestách, to, koľko ich má byť a ich súradnice. Súradnice sú uložené v dvoch poliach. Na začiatku sa program spýta či chce užívateľ testovat program alebo iba hľadať najlepšiu cestu. Ak chce užívateľ testovať, program sa ho spýta na dĺžku tabu listu, po koľkých rovnakých opakovaniach má skončiť (opakovania, kde sa cesta nezlepšuje) a koľko-krát sa má výpočet zopakovať. Ak nechce testovať, spustí sa iba jeden výpočet s predom danými parametrami.

V programe sa nachádzajú funkcie draw, fit, generate1, generate2, generate3, generate4, generate5 a search.   
Funkcia **draw** zoberie pole cesty a vykreslí ju.   
Funkcia **fit** počíta fitness funkciu pre danú cestu, pričom hodnota cesty je vyjadrená pomocou celkovej vzdialenosti cesty. Čím bude táto hodnota nižšia tým lepšie. Vzdialenosť medzi jednotlivými mestami je vypočítaná pomocou pytagorovej vety.   
Funkcie generate1, generate2, generate3, generate4 a generate5 sú rôzne spôsoby implementácie generovania všetkých potomkov cesty.

* **Generate1** generuje potomkov pomocou vymenenia poradia dvoch susedných miest.
* **Generate2** generuje potomkov pomocou vymenenia poradia dvoch náhodných miest.
* **Generate3** generuje potomkov pomocou premiestnenia mesta z jednej pozície na inú. Tento spôsob je efektívny na pripojenie na pripojenie mesta k dvom bližším mestám. (Obrázok 1)
* **Generate4** generuje potomkov pomocou vymenenia poradia miest medzi dvoma náhodnými mestami. Týmto spôsobom eliminujem “križovatky”, čo znamená cesty ktoré sa prekrívajú (Obrázok 2)
* **Generate5** je výpočtovo najnáročnejší, pretože je kombináciou generate3 a generate4.

A black background with a black square

Description automatically generated with medium confidence

Porovnal som si efektivitu týchto štyroch spôsobov generovania potomkov. Pre každý spôsob som si vypočítal počet vytvorených potomkov a zistil som priemerne nájdenú cestu pre 30 a 40 miest (pri zisťovaní priemernej cesty bola dĺžka tabu listu 10, mestá mali vždy rovnaké rozpoloženie a priemer je vypočítaný zo 100 pokusov). Výsledok porovnania je zobrazený v tabuľke:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | generate1 | generate2 | generate3 | generate4 | generate5 |
| Počet vytvorených potomkov (30 miest) | 29 | 870 | 870 | 870 | 1740 |
| Priemerná nájdená cesta(30 miest) | 5526.389 | 2802.459 | 2535.322 | 2298.810 | 2288.259 |
| Počet vytvorených potomkov(40 miest) | 39 | 1560 | 1560 | 1560 | 3120 |
| Priemerná nájdená cesta(40 miest) | 7341.443 | 3515.668 | 2989.976 | 2608.948 | 2577.468 |

Ako môžme vidieť, funkcie generate4 a generate5 hľadajú porovnateľne dlhé priemerné cesty avšak generate4 to robí s menej potomkami a teda aj s menej krokmi, preto budem ďalej generovať potomkov iba s funkciou generate4, avšak kódy ostatných funkcií nechám v programe.

Funkcia **search** vykonáva samotné hľadanie najkratšej cesty. Na začiatok vytvorí náhodné poradie miest a potom pomocou zakázaného prehľadávania zlepšuje túto počiatočnú cestu. Hlavným parametrom v tejto funkcii sú dĺžka tabu listu a to, po koľkých opakovaniach s rovnakou najdlhšou cestou sa funkcia skončí. Pre nájdenie optimálnej hodnoty týchto dvoch premenných som pre ne postupne skúšal rôzne hodnoty, až kým som nenašiel tú najviac optimálnu. Priemer je počítaný zo 100 pokusov. Moje zistenia sú zobrazené v tabuľke:

30 miest 40 miest

|  |  |  |
| --- | --- | --- |
| Dĺžka tabu listu | Počet opakovaní | Priemerná dĺžka |
| 0 | 10 | 2323.913 |
| 10 | 10 | 2303.115 |
| 15 | 10 | 2295.424 |
| 20 | 10 | 2309.027 |
| **20** | **20** | 2293.825 |
| **20** | **30** | 2295.003 |
| **25** | **20** | 2297.969 |
| **30** | **20** | 2292.631 |
| 25 | 40 | 2300.249 |
| 20 | 100 | 2295.571 |
| 100 | 100 | 2284.800 |

|  |  |  |
| --- | --- | --- |
| Dĺžka tabu listu | Počet opakovaní | Priemerná dĺžka |
| 0 | 10 | 2633.348 |
| 10 | 10 | 2631.283 |
| 15 | 10 | 2627.93 |
| 20 | 10 | 2618.376 |
| 20 | 20 | 2619.612 |
| **25** | **20** | 2611.878 |
| **25** | **40** | 2608.411 |
| 30 | 40 | 2611.068 |
| 25 | 60 | 2605.695 |
| 25 | 100 | 2613.241 |
| 100 | 100 | 2601.680 |

Pre 30 miest je najlepšie použiť dĺžku tabu listu 20, nakoľko keď ho ďalej zväčšujem zlepšenie nie je také signifikantné, zatiaľčo výpočet trvá dlhšie. Pre počet opakovaní som vybral číslo 20. Po jeho zväčšení nevidíme veľký rozdiel v zlepšení.  
Pre 40 miest je najlepšie použiť dĺžku tabu listu 25, nakoľko pri zväčšovaní je zlepšenie nízke. Pre počet opakovaní som použil číslo 20.  
Keďže hľadám jedno číslo pre dĺžku tabu listu a jedno číslo pre počet opakovaní nezávisle od počtu miest, použijem vyššiu hodnotu pre obe hodnoty, takže v programe nastavím dĺžku tabu listu na 25 a počet opakovaní na 20. Pre 1000 výpočtov najkratšej cesty mi vyšla priemerná dĺžka cesty s tabu searchom pre 30 miest 2297.296 a pre 40 miest 2617.100, pričom bez použitia tabu searchu mi vyšla priemerná cesta pre 30 miest 2322.106 a pre 40 miest 2629.235.

## Conclusion

Ako môžme vidieť, implementácia tabu zoznamu dokáže zlepšiť kvalitu výsledku aj keď iba o približne 1.5%. Pre porovnanie som si pripravil porovnanie zlepšenia hľadania najkratšej cesty pre ostatné funkcie generovania potomkov, keďže sú tieto funkcie menej efektívne, tak rozdiel medzi použitím tabu zoznamu a jeho nepoužitia je zrejmejší:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 30 miest | | | 40 miest | | |
| Dĺžka tabu listu | Počet opakovaní | generate2 | generate3 | generate4 | generate2 | generate3 | generate4 |
| 0 | 10 | 2953.840 | 2533.053 | 2323.913 | 3471.023 | 3000.675 | 2633.348 |
| 10 | 10 | 2848.950 | 2496.793 | 2303.115 | 3469.831 | 2945.603 | 2631.283 |
| 20 | 10 | 2843.556 | 2521.327 | 2309.027 | 3480.812 | 2994.746 | 2618.376 |
| 20 | 20 | 2808.444 | 2517.808 | 2293.825 | 3425.156 | 2944.229 | 2619.612 |
| 25 | 20 | 2816.923 | 2503.434 | 2297.969 | 3453.272 | 2954.565 | 2611.878 |
| 25 | 40 | 2762.053 | 2501.691 | 2300.249 | 3394.160 | 2947.775 | 2608.411 |

# Simulované žíhanie (simulated annealing)

## Task

Obchodný cestujúci má navštíviť viacero miest, pričom chce prejsť cez všetky mestá, cez každé práve raz a na konci chce skončiť v meste v ktorom začínal. Chce nájsť čo najkratšiu trasu.

Je daných 30 a 40 miest, každé majú náhodne vygenerované súradnice. Rozmer mapy je 500x500. Cena cesty medzi dvoma mestami zodpovedá Euklidovej vzdialenosti – vypočíta sa pomocou Pytagorovej vety. Cieľom je nájsť take poradie, pre ktoré bude mať celková trasa čo najmenšiu dĺžku. Výstupom je poradie miest a dĺžka zodpovedajúcej cesty.Zadanie je potrebné riešiť pomocou simulovaného žíhania.

## Opis riešenia problému

Cesta bude uložená vo forme zoznamu reprezentujúceho poradie miest, ktoré navštívim, pričom každé z miest bude v zozname iba raz (posledný krok do začínajúceho mesta nebude v zozname opísaný). Na začiatku prehľadávania bude mať prostredie určitú teplotu, pri každej iterácii sa teplota zníži, keď teplota bude nižšia ako najmenšia možná teplota alebo keď už nebudem mať žiadnych nasledovníkov, hľadanie končí a aktuálna cesta je riešením. Pri hľadaní budem vytvárať nasledovníkov aktuálnej cesty a vyberiem si jedného z nich, do ktorého sa pokúsim vojsť, vojdem doň, ak je táto cesta lepšia, alebo ak náhodne vygenerované číslo je menšie ako výsledok výrazu [exp((current\_distance-pos\_distance)/temp)](https://en.wikipedia.org/wiki/Simulated_annealing). Ak sa mu to podarí, táto cesta sa stáva mojim aktuálnym riešením. Ak sa mu to nepodarí vyberám si z ostatných nasledovníkov. Pre testovanie programu budem používať za každým rovnaké rozpoloženie miest, rozpoloženie použité na testovanie je zakomentované v kóde.

## Opis kódu

Na začiatku programu mám uložené dáta o mestách, to koľko ich má byť a ich súradnice. Súradnice sú uložené v dvoch poliach. Na začiatku sa program spýta či chce užívateľ testovat program alebo iba hľadať najlepšiu cestu. Ak chce užívateľ testovať, program sa ho spýta na počiatočnú teplotu, mieru klesania teploty, minimálnu teplotu a koľko-krát sa má výpočet zopakovať. Ak nechce testovať, spustí sa iba jeden výpočet s predom danými parametrami.

V programe sa nachádzajú funkcie draw, fit, generate1, generate2, generate3, generate4, generate5 a search.   
Funkcia **draw** zoberie pole cesty a vykreslí ju.   
Funkcia **fit** počíta fitness funkciu pre danú cestu, pričom hodnota cesty je vyjadrená pomocou celkovej vzdialenosti cesty, čím bude táto hodnota nižšia tým lepšie. Vzdialenosť medzi jednotlivými mestami je vypočítaná pomocou pytagorovej vety.   
Funkcie generate1, generate2, generate3, generate4 a generate5 sú rôzne spôsoby implementácie generovania všetkých potomkov cesty.

* **Generate1** generuje potomkov pomocou vymenenia poradia dvoch susedných miest.
* **Generate2** generuje potomkov pomocou vymenenia poradia dvoch náhodných miest.
* **Generate3** generuje potomkov pomocou premiestnenia mesta z jednej pozície na inú Tento spôsob je efektívny na pripojenie na pripojenie mesta k dvom bližším mestám. (Obrázok 1)
* **Generate4** generuje potomkov pomocou vymenenia poradia miest medzi dvoma náhodnými mestami. Týmto spôsobom eliminujem “križovatky”, čo znamená cesty ktoré sa prekrívajú. (Obrázok 2)
* **Generate5** je výpočtovo najnáročnejší, pretože je kombináciou generate3 a generate4

A black background with a black square

Description automatically generated with medium confidence

Porovnal som si efektivitu týchto štyroch spôsobov generovania potomkov. Pre každý spôsob som si vypočítal počet vytvorených potomkov a zistil som priemerne nájdenú cestu pre 30 a 40 miest (pri zisťovaní priemernej cesty bola počiatočná teplota 50, miera ochladzovania 0.99 a najnižšia teplota 1, mestá mali vždy rovnaké rozpoloženie a priemer je vypočítaný zo 100 pokusov). Výsledok porovnania je zobrazený v tabuľke:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | generate1 | generate2 | generate3 | generate4 | generate5 |
| Počet vytvorených potomkov (30 miest) | 29 | 870 | 870 | 870 | 1740 |
| Priemerná nájdená cesta(30 miest) | 5463.709 | 2663.060 | 2362.604 | 2318.053 | 2283.089 |
| Počet vytvorených potomkov(40 miest) | 39 | 1560 | 1560 | 1560 | 3120 |
| Priemerná nájdená cesta(40 miest) | 7238.903 | 3291.386 | 2750.340 | 2672.184 | 2605.113 |

Ako môžme vidieť, funkcia generate5 hľadá lepšie priemerné cesty, avšak generate4 to robí s menej potomkami, a teda aj s menej krokmi, preto budem ďalej generovať potomkov iba s funkciou generate4, avšak kódy ostatných funkcií nechám v programe.

Funkcia **search** vykonáva samotné hľadanie najkratšej cesty. Na začiatok vytvorí náhodné poradie miest a potom pomocou simulovaného žíhania zlepšuje túto počiatočnú cestu. Hlavným parametrom v tejto funkcii sú počiatočná teplota, miera ochladzovania a najnižšia možná teplota. Pre nájdenie optimálnej hodnoty týchto troch premenných som pre ne postupne skúšal rôzne hodnoty, až kým som nenašiel tú najviac optimálnu. Dal som si za ciel, nech je počet iterácií pre každú variant menší ako 500, pričom som si zadal nejakú minimálnu teplotu a mieru ochladzovania. Počiatočnú teplotu som si zadal tak, aby bol počet iterácií čo najbližšie k 500 pomocou rovnice x\*Power[0.99,500]<1. Priemer je počítaný zo 100 pokusov. Moje zistenia sú zobrazené v tabuľke:

30 miest 40 miest

|  |  |  |  |
| --- | --- | --- | --- |
| Počiatočná teplota | Miera ochladzovania | Najnižšia možná teplota | Priemerná dĺžka |
| 152 | 0.99 | 1 | 2311.201 |
| 12.25 | 0.995 | 1 | 2309.705 |
| 4.49 | 0.997 | 1 | 2326.961 |
| 7.41 | 0.996 | 1 | 2337.068 |
| 33.52 | 0.993 | 1 | 2309.620 |
| **55.48** | **0.992** | **1** | 2300.361 |
| 91.86 | 0.991 | 1 | 2314.221 |
| 15.21 | 0.99 | 0.1 | 2334.671 |
| 2437.59 | 0.98 | 0.1 | 2335.716 |
| 1521.95 | 0.99 | 10 | 2497.713 |
| 76.09 | 0.99 | 0.5 | 2305.456 |
| 6.12 | 0.995 | 0.5 | 2326.503 |
| 45.93 | 0.991 | 0.5 | 2306.432 |
| 47.69 | 0.9923 | 1 | 2305.517 |

|  |  |  |  |
| --- | --- | --- | --- |
| Počiatočná teplota | Miera ochladzovania | Najnižšia možná teplota | Priemerná dĺžka |
| 152 | 0.99 | 1 | 2659.870 |
| 12.25 | 0.995 | 1 | 2650.666 |
| 4.49 | 0.997 | 1 | 2677.642 |
| 7.41 | 0.996 | 1 | 2677.509 |
| 33.52 | 0.993 | 1 | 2640.848 |
| 55.48 | 0.992 | 1 | 2646.923 |
| 43.12 | 0.9925 | 1 | 2641.359 |
| 38.99 | 0.9927 | 1 | 2643.400 |
| **47.69** | **0.9923** | **1** | 2638.450 |
| 15.21 | 0.99 | 0.1 | 2688.258 |
| 191.37 | 0.985 | 0.1 | 2669.466 |
| 41.83 | 0.988 | 0.1 | 2664.186 |
| 53.88 | 0.9875 | 0.1 | 2656.075 |
| 56.68 | 0.9874 | 0.1 | 2663.730 |

## Conclusion

Zistil som, že najlepšia počiatočná teplota je okolo 50, miera ochladzovania okolo 0.992 a najnižšia možná teplota 1, v tejto kombinácii sú priemerné dĺžky najkratšie. Pre svoj program som si teda vybral možnosť počiatočná teplota = 55.48, miera ochladzovania = 0.992 a najnižšiu možnú teplotu = 1. Pre 1000 výpočtou najkratšej cesty mi s touto konfiguráciou vyšla priemerná najkratšia cesta pre 30 miest 2305.661 a pre 40 miest 2645.046.