Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Umelá inteligencia

Zadanie 3 - Problém obchodného cestujúceho
B)Tabu search

Jakub Hrnčár
AIS ID: 110 800, ak. rok 2021/2022

1. Úvod do zadania

Problém obchodného cestujúceho (ďalej TSP) je populárny NP problém. Jeho podstata je hľadanie najlacnejšej Hamiltonovskej kružnice, pričom jeho obtiažnosť spočíva najmä v tom, že nikdy nevieme s istotou povedať, že sme našli najlacnejšiu - vždy môže byť lacnejšia. Pre tento dôvod používame na vypracovanie iba optimalizačné algoritmy, ktoré teoreticky môžu pracovať do nekonečna.

Použitý algoritmus v tomto zadaní sa nazýva Tabu search; je to modifikovaná verzia Hill-climbing algoritmu, takže iba neustále vytvárame nové stavy a porovnávame, či sme vytvorili niečo lepšie.

Zadanie bolo vypracované v jazyku Python 3.9, v prostredí PyCharm Ultimate s hardvérom:
CPU: Intel i3-10100F 4 jadrá 8 vlákien
RAM: 16 GB DDR4 2666 MHz CL 16

2. Stručný opis algoritmu

Celý program sa rozdeľuje do 3 častí: generovanie základného stavu, Tabu search, fitness funkcia.

Generovanie základného stavu

Základný stav je vygenerovaný náhodne; jednotlivé body sa môžu nachádzať na náhodnom mieste v simulovanom 200x200 2D poli, pri čom sa nemôžu sa nachádzať na rovnakom mieste. Stav je reprezentovaný 2D polom, v ktorom sa nachádzajú súradnice každého mesta, pri čom cesta sa tvorí čítaním súradníc zľava doprava:

```
starter = []
while len(starter) != 40:
   new = [random.randint(1, 200), random.randint(1, 200)]
   if new not in starter:
       starter.append(new)
```

Fitness funkcia

Keďže sa nachádzam v 2D poli a potrebujem zistiť priamu vzdialenosť medzi dvoma bodmi a následne tieto vzdialenosti sčítať dokopy. Táto reprezentácia dovoľuje veľmi jednoduché počítanie fitness hodnoty - pracujem s údajmi ako body v karteziánskej sústave a vypočítam veľkosť vektora medzi dvoma susednými bodmi.

```
for i in range(len(current) - 1):
   vector = [current[i + 1][0] - current[i][0], current[i + 1][1]
- current[i][1]]
   counter += math.sqrt((vector[0]**2) + (vector[1]**2))
```

Tabu search

Pri implementácií som sa riadil týmto pseudokódom:

```
1  sBest ← s0
2  bestCandidate ← s0
3  tabuList ← []
4  tabuList.push(s0)
5  while (not stoppingCondition())
6      sNeighborhood ← getNeighbors(bestCandidate)
7      bestCandidate ← sNeighborhood[0]
8      for (sCandidate in sNeighborhood)
9          if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10             bestCandidate ← sCandidate
11         end
12     end
13     if (fitness(bestCandidate) > fitness(sBest))
14         sBest ← bestCandidate
15     end
16     tabuList.push(bestCandidate)
17     if (tabuList.size > maxTabuSize)
18         tabuList.removeFirst()
19     end
20 end
21 return sBest
```

Je to teda iteratívny algoritmus, ktorý veľmi pripomína algoritmus na hľadanie minima/maxima z údajov. Pri generovaní detí (susedov) jedného stavu používam nasledovný spôsob:

Generujem toľko detí, koľko je aj miest (bodov).

Deti sa generujú výmenou prvého prvku s každým ďalším prvkom.

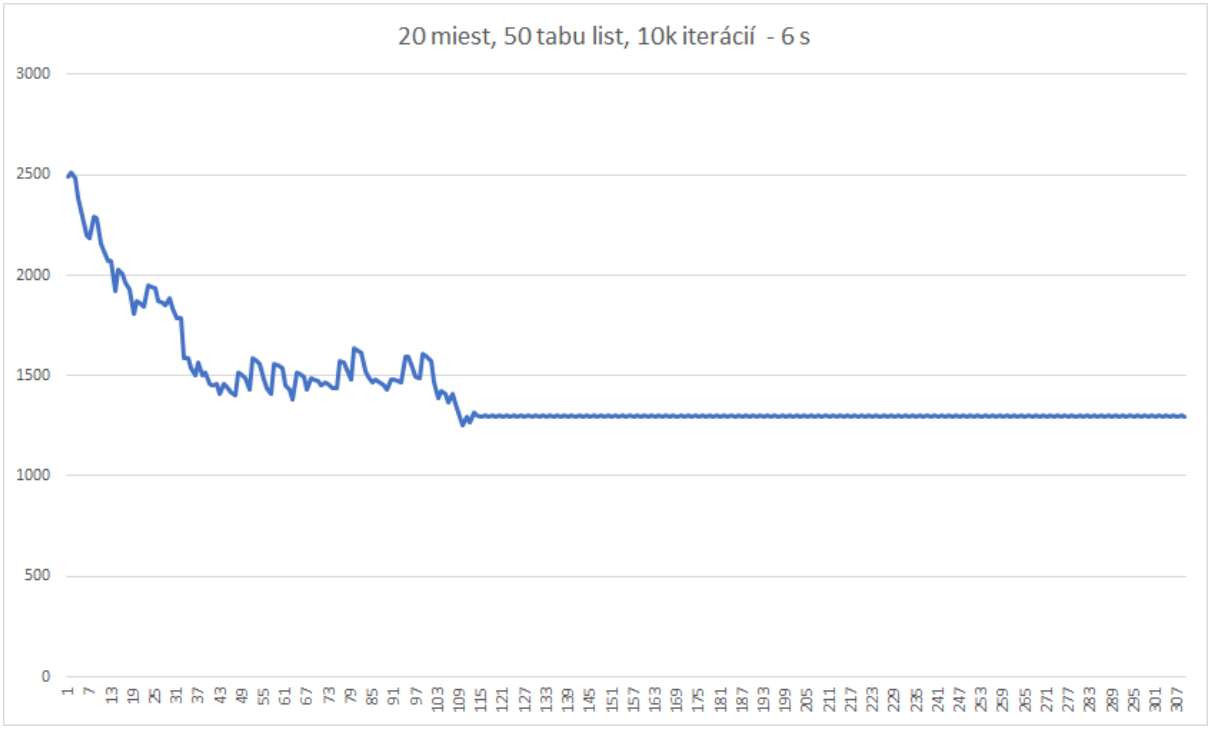
Posledné dieťa sa vygeneruje výmenou druhého prvku s posledným.

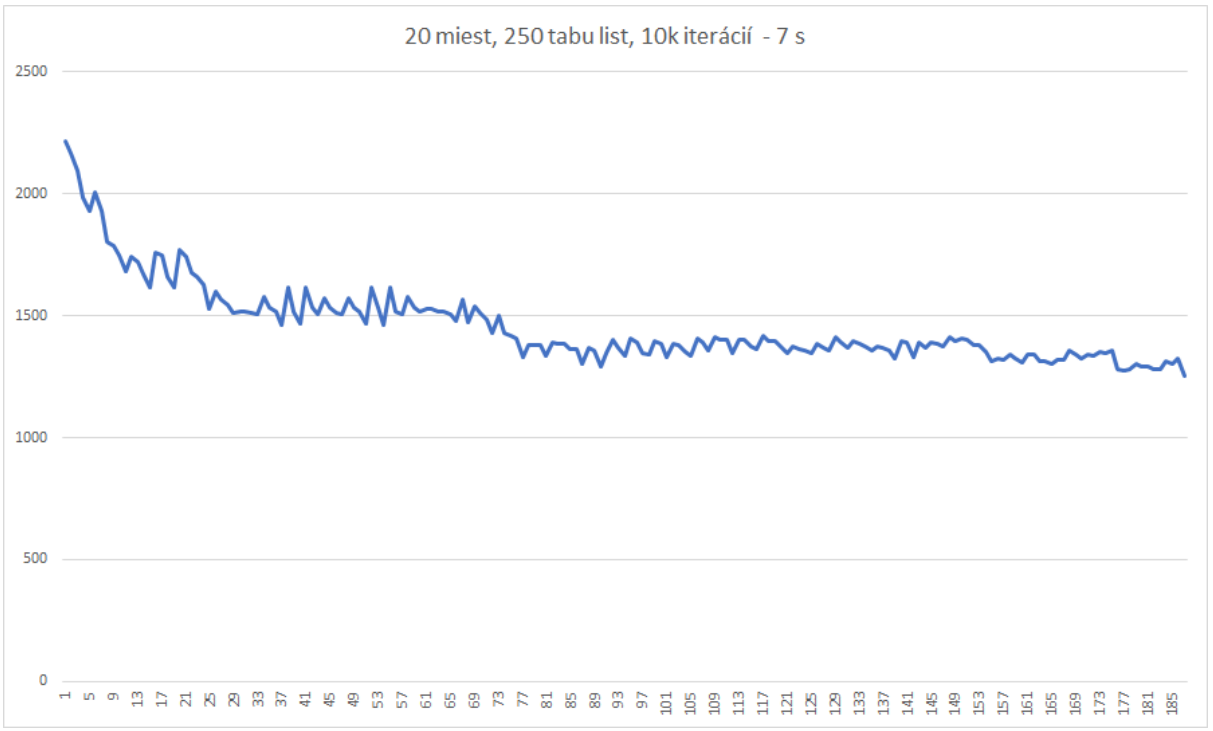
Tabu list je iba jednoduché pole, do ktorého sa vkladajú prvky, ktoré boli vyhodnotené ako Najlepšie z detí (ak Najlepšie neexistuje, tak sa vloží prvý).

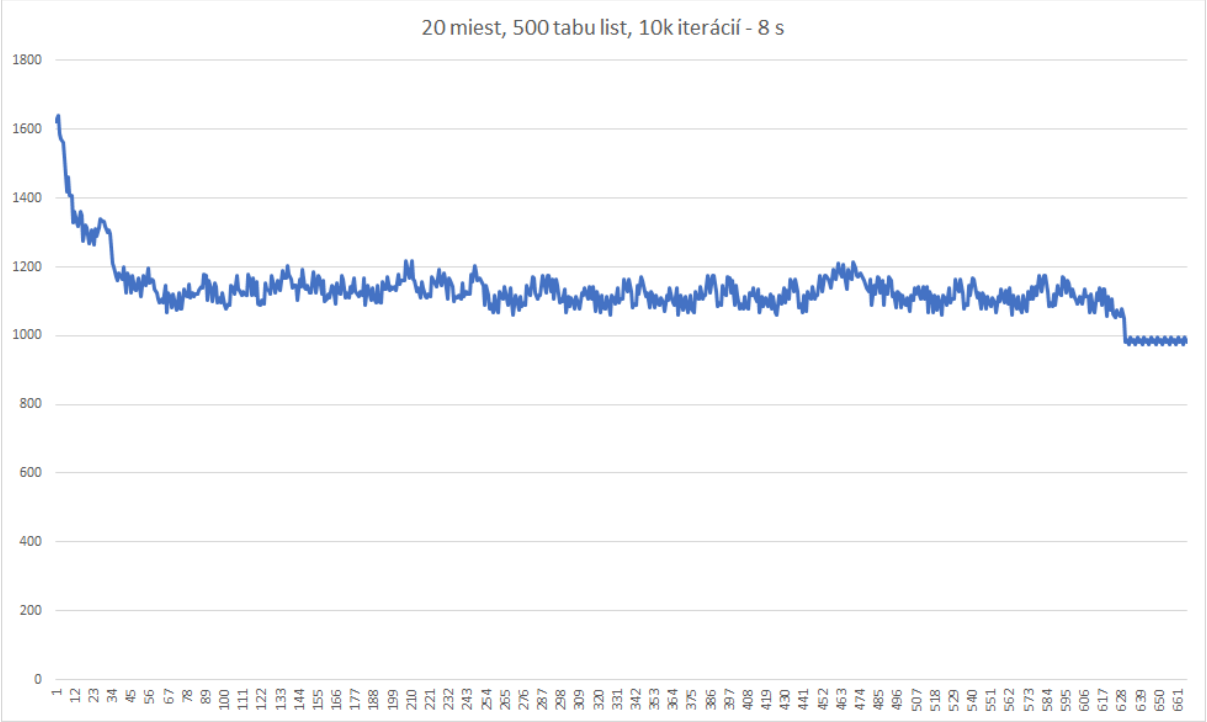
3. Testovanie programu

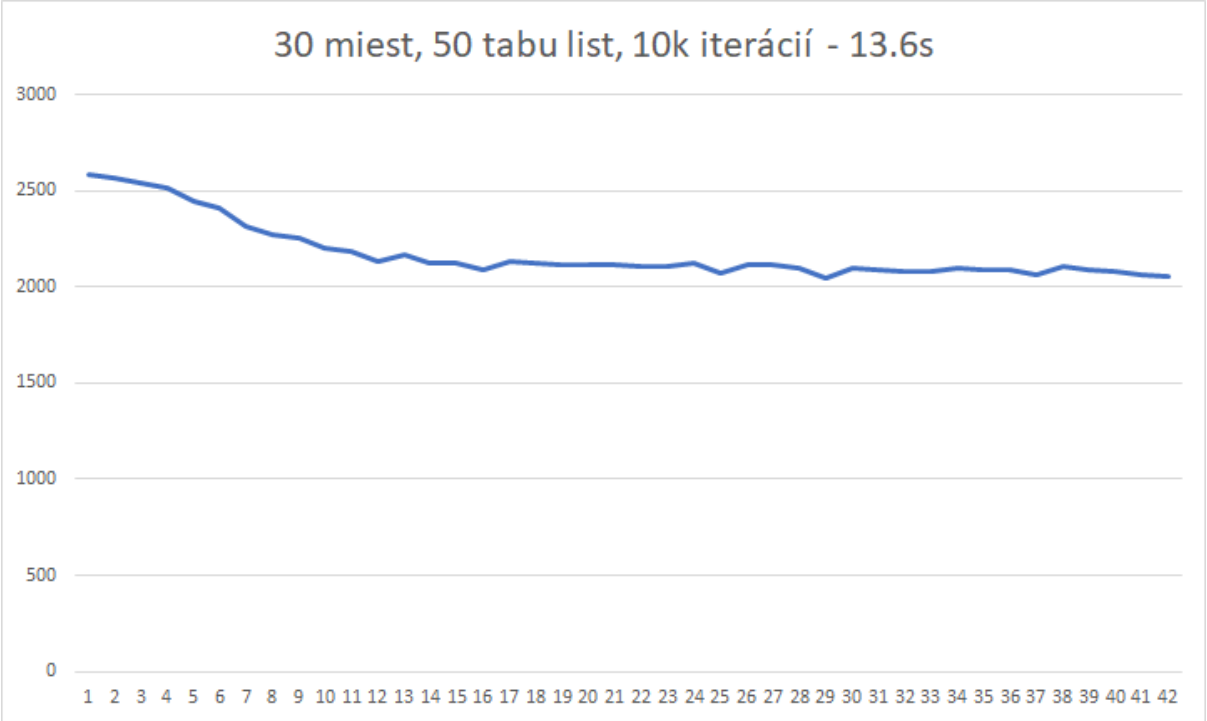
Vždy, keď sa splní podmienka nájdenia lepšieho kandidáta ako je základný kandidát (riadok 9 v pseudokóde), tak sa vloží fitness hodnota do .csv súboru. Na konci testovania tento súbor spracujem v Exceli do grafu.

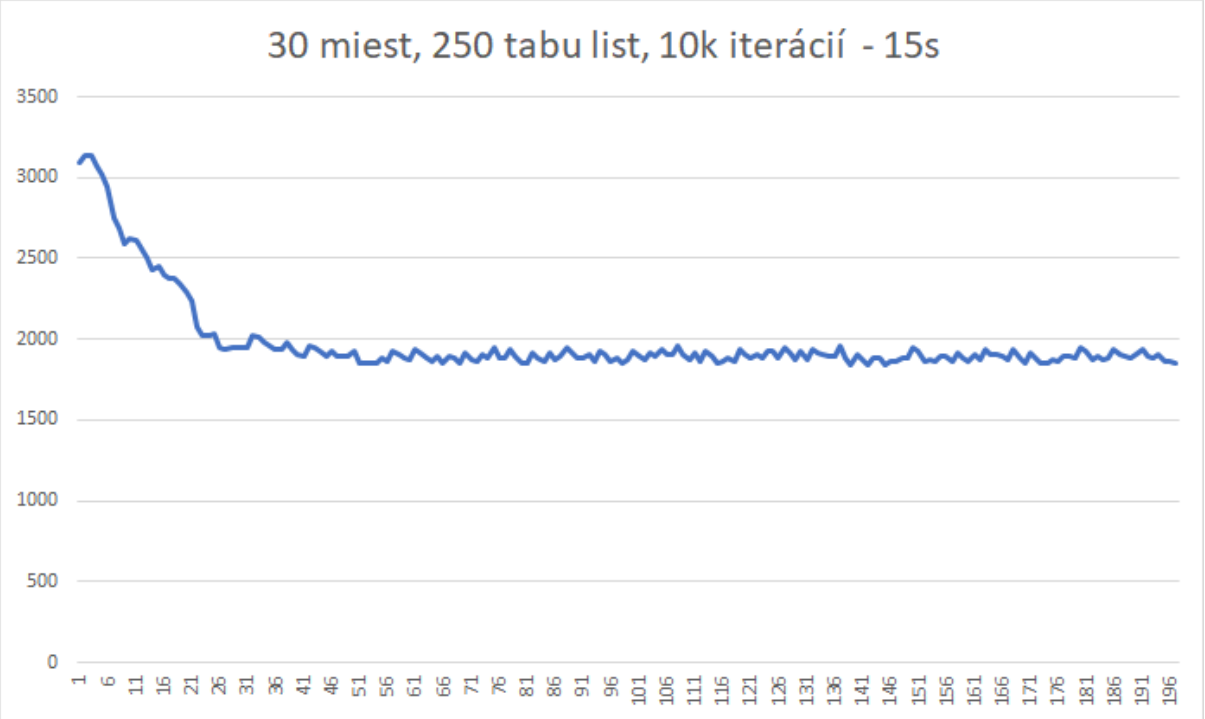
Keďže sa mestá generujú na náhodných miestach, existuje možnosť, že sa vytvorí pomerne dobrá cesta hneď na začiatku, lebo sa mestá umiestnia na výhodné miesta. Na testovanie som zvolil rôzne hodnoty počtu miest (20, 30, 40) a rôzne hodnoty veľkosti Tabu listu (50, 250, 500). Všetky testy boli spustené pre 10 000 iterácií, čo som zistil že je dostačujúce množstvo pre dané počty miest.

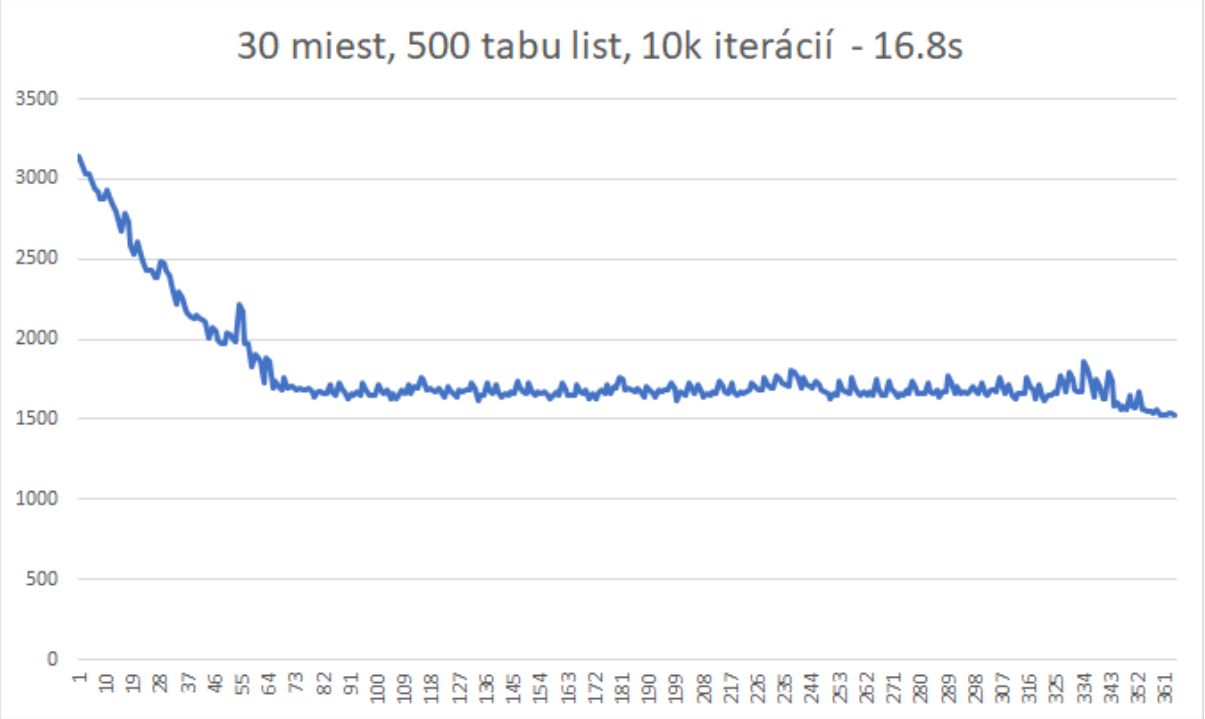

20 miest, 50 tabu list, 10k iterácií - 6 s
3000
2500
2000
1500
1000
500
0
1 7 13 19 25 31 37 43 49 55 61 67 73 79 85 91 97 103 109 115 121 127 133 139 145 151 157 163 169 175 181 187 193 199 205 211 217 223 229 235 241 247 253 259 265 271 277 283 289 295 301 307

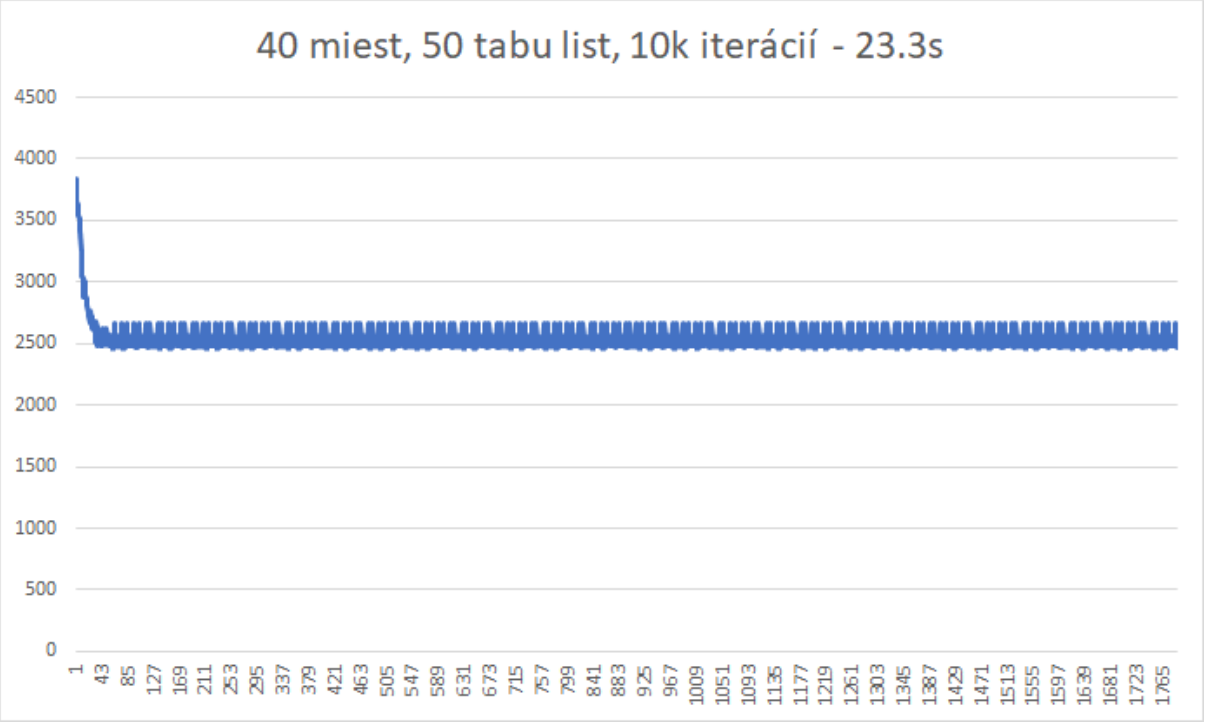

20 miest, 250 tabu list, 10k iterácií - 7 s
2500
2000
1500
1000
500
0
1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77 81 85 89 93 97 101 105 109 113 117 121 125 129 133 137 141 145 149 153 157 161 165 169 173 177 181 185

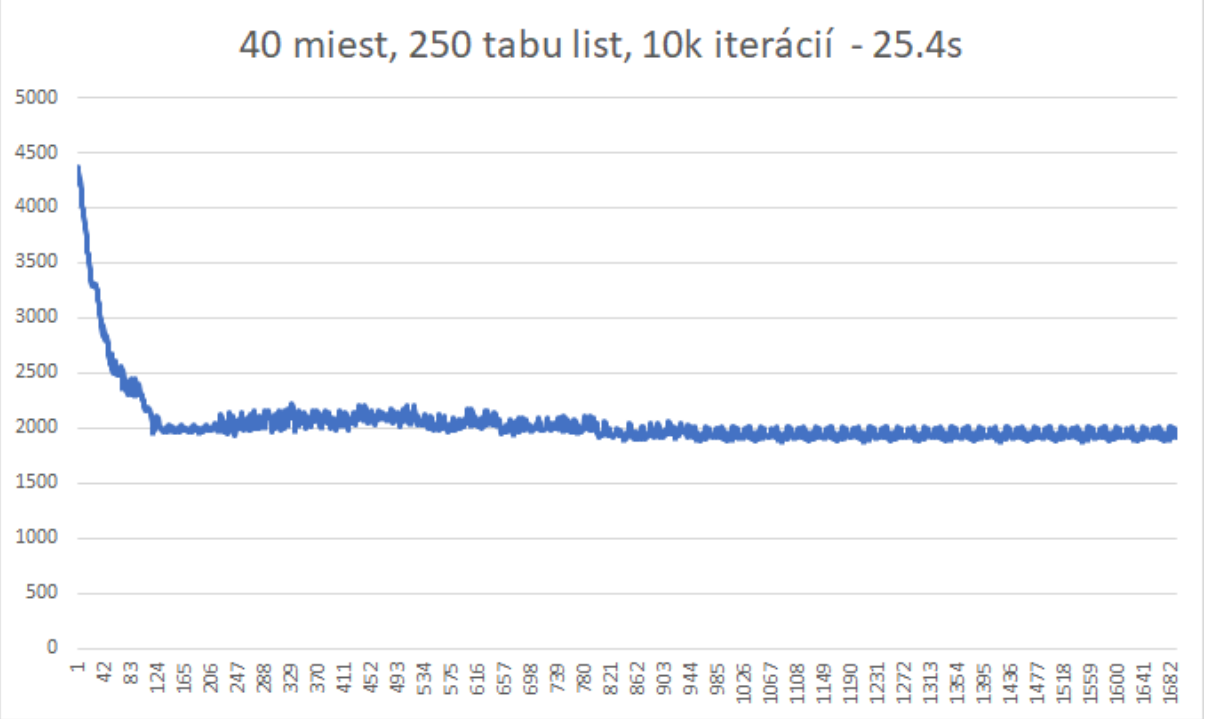

20 miest, 500 tabu list, 10k iterácií - 8 s
1800
1600
1400
1200
1000
800
600
400
200
0
1 12 23 34 45 56 67 78 89 100 111 122 133 144 155 166 177 188 199 210 221 232 243 254 265 276 287 298 309 320 331 342 353 364 375 386 397 408 419 430 441 452 463 474 485 496 507 518 529 540 551 562 573 584 595 606 617 628 639 650 661

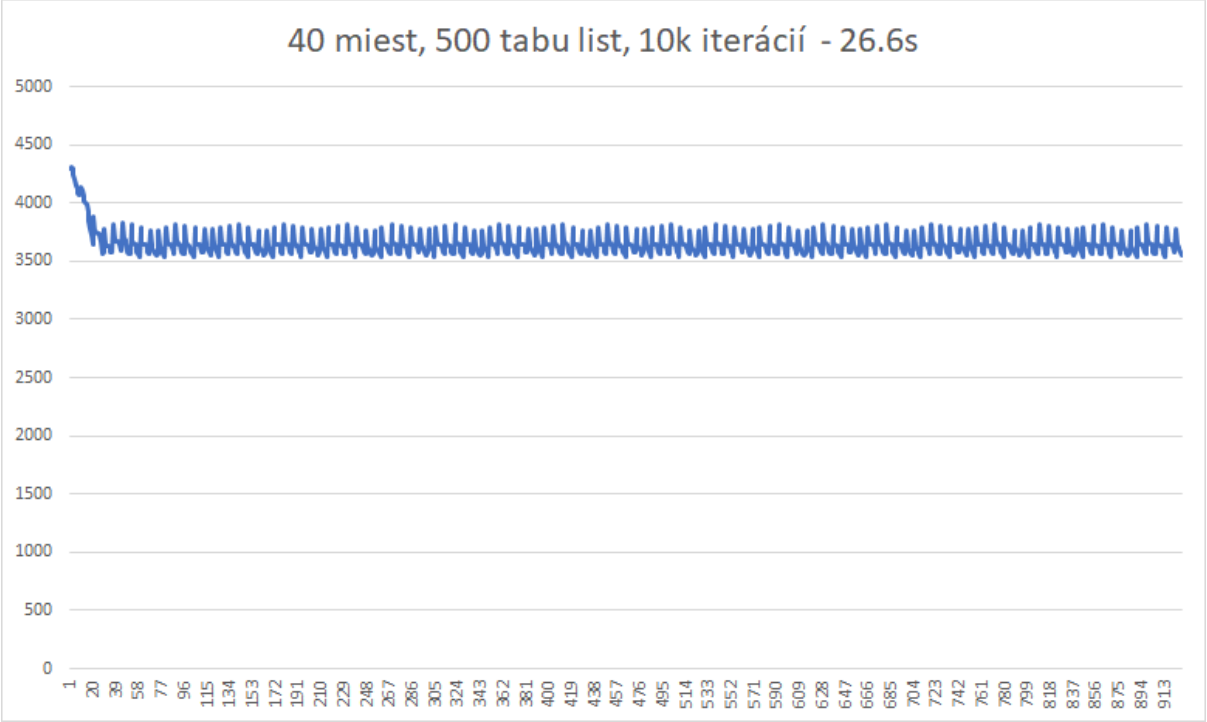

30 miest, 50 tabu list, 10k iterácií - 13.6s
3000
2500
2000
1500
1000
500
0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42


30 miest, 250 tabu list, 10k iterácií - 15s
3500
3000
2500
2000
1500
1000
500
0
1 6 11 16 21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96 101 106 111 116 121 126 131 136 141 146 151 156 161 166 171 176 181 186 191 196


30 miest, 500 tabu list, 10k iterácií - 16.8s
3500
3000
2500
2000
1500
1000
500
0
1 10 19 28 37 46 55 64 73 82 91 100 109 118 127 136 145 154 163 172 181 190 199 208 217 226 235 244 253 262 271 280 289 298 307 316 325 334 343 352 361


40 miest, 50 tabu list, 10k iterácií - 23.3s
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
43
85
127
169
211
253
295
337
379
421
463
505
547
589
631
673
715
757
799
841
883
925
967
1009
1051
1093
1135
1177
1219
1261
1303
1345
1387
1429
1471
1513
1555
1597
1639
1681
1723
1765


40 miest, 250 tabu list, 10k iterácií - 25.4s
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
1
42
83
124
165
206
247
288
329
370
411
452
493
534
575
616
657
698
739
780
821
862
903
944
985
1026
1067
1108
1149
1190
1231
1272
1313
1354
1395
1436
1477
1518
1559
1600
1641
1682



4. Záver

Na väčšine testov vidíme pokles ceny cesty o 40-50%, pričom generovanie je náhodné, takže toto číslo nie je smerodajné; napríklad v poslednom teste sa nepodarilo pri 10 000 iteráciach vylepšiť hodnotu o také významné číslo ako v predošlých prípadoch. Ďalšie závery z testovania sú nasledovné:

- vyrovnanie krivky znamená nájdenie najlacnejšej cesty
- krivka zvykne oscilovať medzi rovnakými hodnotami po nájdení najlacnejšej cesty
- čím je väčší tabu list, tým menej vyrovnaný graf je a vzniká viac oscilácií
- tempo zlepšovania sa postupom iterácií výrazne spomaľuje; na začiatku je veľmi rýchle
- 10 000 iterácií je pre 20, 30, 40 miest dostatočný počet
- čím je väčší tabu list, tým je dlhšie aj vykonávanie programu
- optimalizácia by bola použitie hash tabuľky namiesto poľa v tabu liste
- ďalšia optimalizácia by sa dala vykonať skrz zlacnenie výpočtu vzdialenosti medzi mestami; napríklad pomocou matice susednosti, pričom by sa vzdialenosť nepočítala vždy nanovo