Róbert Junas
102970

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

# Zadanie č. 3a – Zenová záhrada

UMELÁ INTELIGENCIA

Róbert Junas

FIIT STU

Cvičenie: Utorok 16:00

21.11.2021

Róbert Junas
102970

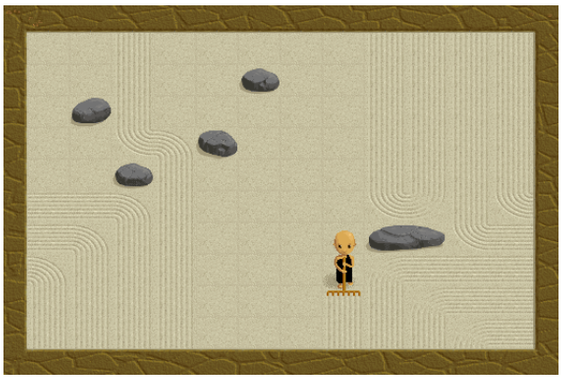
# Obsah

1. Zadanie ... 3
1.1. Zadanie ... 4
2. Implementácia ... 4
2.1. Spustenie ... 4
2.2. Štruktúry ... 5
2.3. Pohyb ... 6
2.4. Generovanie okolia ... 7
2.5. Výpočet fitness a výber najlepšieho suseda ... 8
2.6. Hľadanie ... 9
2.7. Používateľské rozhranie ... 10
3. Testovanie ... 12
4. Záver ... 24

Róbert Junas
102970

# 1. Zadanie

Zenová záhradka je plocha vysypaná hrubším pieskom (drobnými kamienkami). Obsahuje však aj nepohyblivé väčšie objekty, ako napríklad kamene, sochy, konštrukcie, samorasty. Mních má upraviť piesok v záhradke pomocou hrablí tak, že vzniknú pásy ako na nasledujúcom obrázku.

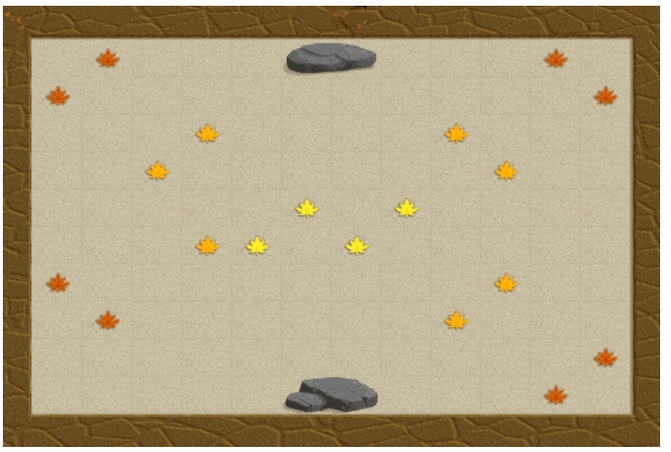


Pásy môžu ísť len vodorovne alebo zvislo, nikdy nie šikmo. Začína vždy na okraji záhradky a ťahá rovný pás až po druhý okraj alebo po prekážku. Na okraji – mimo záhradky môže chodiť ako chce. Ak však príde k prekážke – kameňu alebo už pohrabanému piesku – musí sa otočiť, ak má kam. Ak má voľné smery vľavo aj vpravo, je jeho vec, kam sa otočí. Ak má voľný len jeden smer, otočí sa tam. Ak sa nemá kam otočiť, je koniec hry. Úspešná hra je taká, v ktorej mních dokáže za daných pravidiel pohrabať celú záhradu, prípade maximálny možný počet políčok. Výstupom je pokrytie danej záhrady prechodmi mnícha. Pokrytie zodpovedajúce presne prvému obrázku (priebežný stav) je napríklad takéto:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 8 | 9 |
| 0 | 0 | 1 | 0 | 0 | K | 0 | 0 | 10 | 10 | 8 | 9 |
| 0 | K | 1 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 8 | 9 |
| 0 | 0 | 1 | 1 | K | 0 | 0 | 0 | 10 | 10 | 8 | 9 |
| 0 | 0 | K | 1 | 0 | 0 | 0 | 0 | 10 | 10 | 8 | 9 |
| 2 | 2 | 2 | 1 | 0 | 0 | 0 | 0 | 10 | 10 | 8 | 9 |
| 3 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | K | K | 8 | 8 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 5 | 5 | 5 | 5 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 11 | 5 | 6 | 6 | 6 |
| 4 | 3 | 2 | 1 | 0 | 0 | 0 | 11 | 5 | 6 | 7 | 7 |

Úlohu je možné rozšíriť tak, že mních navyše zbiera popadané lístie. Listy musí zbierať v poradí: najprv žlté, potom pomarančové a nakoniec červené. Príklad vidno na obrázku nižšie. Listy, ktoré zatiaľ nemôže zbierať, predstavujú pevnú prekážku. Je potrebné primerane upraviť fitness funkciu. Za takto rozšírenú úlohu je možné získať navyše jeden bonusový bod.

Róbert Junas
102970



### 1.1.Zadanie

Uvedenú úlohu riešte pomocou evolučného algoritmu. (Je možné použiť aj ďalšie algoritmy, ako sú uvedené v probléme obchodného cestujúceho.) Maximálny počet génov nesmie presiahnuť polovicu obvodu záhrady plus počet kameňov, v našom príklade podľa prvého obrázku 12+10+6=28. Fitnes je určená počtom pohrabaných políčok. Výstupom je matica, znázorňujúca cesty mnícha. Je potrebné, aby program zvládal aspoň záhradku podľa prvého obrázku, ale vstupom môže byť v princípe ľubovoľná mapa.

Na implementáciu sme mali použiť Tabu search.

# 2. Implementácia

## 2.1. Spustenie

Na implementovanie sme použili python verzie 3.10.0. Použité knižnice:

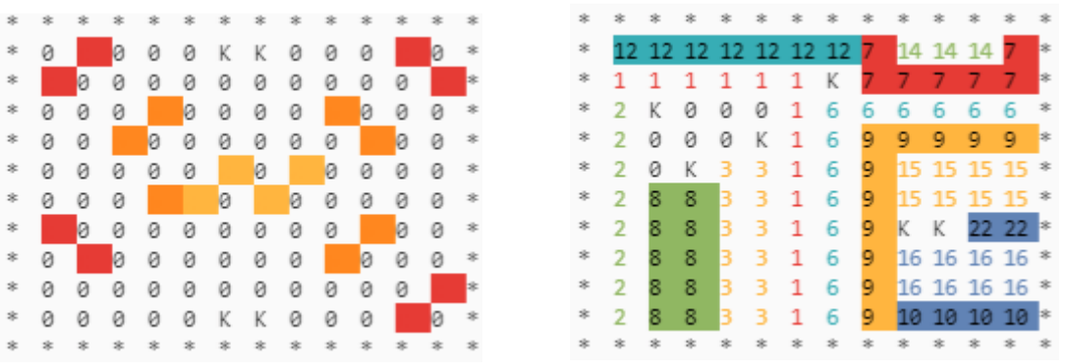
- Random
- Configparser
- Matplotlib
- Colorama
- Copy
- Time

Z toho treba nainštalovať knižnice **matlotlib** a **colorama**. Samotné spustenie vykonáme v pracovnom priečinku pomocou príkazu: py main.py. Samotná implementácia sa skladá zo súborov:

- Main.py – obsahuje funkcie testov a hlavnú funkciu
- Src/find.py – funkcia hľadania
- Src/mapa.py – trieda záhrady a funkcie nad ňou
- Src/pohyby.py – trieda zahradnika a funkcie pre jeho pohyb
- Src/gens.py – trieda frekvenčnej pamäti a funkcie na generovanie okolia a nájdenie najlepšieho nasledovníka.
- Src/plotter.py – trieda Zbierania dát a vykresľovanie grafu z nich
- Src/FileParser.py – nastavuje počiatočne parametre programu
- Src/UserInterface.py – funkcie používateľského prostredia

## 2.2. Štruktúry

V našej implementácii používame niekoľko tried, ktoré nám pomáhajú s vykonávaním algoritmu. Hlavnou triedou je **Mapa**, v ktorej si pamätáme veľkosť plochy, pozície kameňov a listov, počet listov každej farby, či je povolené preskakovanie neplatných pozícií a samotnú vygenerovanú mapu, ktorá neobsahuje žiadne pohyby. Samotná mapa sa skladá zo znakov *, ktoré tvoria orámovanie plochy, písmena **K** reprezentujúce kameň, z písmen **z** – žltý list; **p** – oranžový list; **c** – červený list. Políčka, cez ktoré záhradník neprešiel sú označené číslom 0 a cesty, ktoré vykonal sú označené číslom indexu v pohybovom vektore + 1.



*Obrázok 1 príklad výpisu plochy*

Medzi funkcie mapy patrí vytvorenie prázdnej mapy „createEmptyMap()“, z jej parametrov, vyplnenie mapy pohybmi „fillMap(moves)“, výpočet fitness „fitness(moves)“, výpočet fitness z listov „leafFitness(leafCount)“ a ako poslednú funkciu má výpis mapy printMap(moves).

Potom máme štruktúru hrabača **Gardener,** v ktorom sa uchováva jeho pozícia na ploche, zbieraná farba listov a počet pozbieraných listov, fitness a počet vykonaných ciest.

```
class Gardener:
    def __init__(self,col,row) -> None:
        self.row = row
        self.col = col
        self.cColor = leaves[0]
        self.pocetPozbieranych = [0,0,0]
        self.fitness = 0
        self.pocetCiest = 0
```

*Obrázok 2 trieda hrabača*

Ďalšou triedou je **FrequencyTable**, ktorá slúži na dlhodobú pamäť tabu searchu. V tejto triede si vytvárame tabuľku, kde si pamätáme koľkokrát sa štartovná pozícia ocitla na určitom indexe v najlepšom nasledovníkovi. Funkcie sú pridanie hodnoty do pamäte addOccurance (num, movePos), kde movePos je index v pohybovom vektore a num je index štartu. Ďalej má funkciu na výpočet penalizácie calcPenalty(number,movePos).

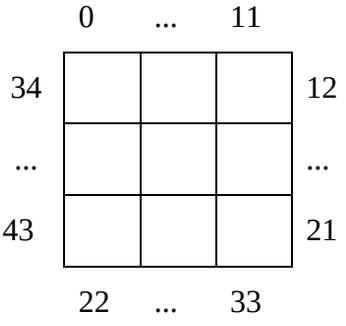
Poslednou triedou je zberač dát **DataCollector**, ktorý slúži na vytvorenie grafu z dát, ktoré zbieral. Funkciami sú createCollection(self,numOfAxis,Name), kde Name je názov kolekcie, numOfAxis je počet osí grafu a teda aj z koľkých údajov sa skladá zozbieraný údaj. Keď sa vytvorí kolekcia, tak funkcia vráti jej identifikačný kód, podľa ktorého pridávame prvky do kolekcie (funkcia addToCollection(id,data)) a aj vkresľujeme graf (create2DGraphFromIds).

## 2.3. Pohyb

Na vypracovanie zadania sme použili vektor pohybov, ktorého maximálna veľkosť je rovná polovici obvodu + počtu kameňov. V tomto pohybovom vektore sú indexy od 0 po polovicu obvodu používané ako štartovacie pozície záhradkára na obvode a zvyšné sú nastavené na 0 alebo 1 a určujú, ktorým smerom sa záhradkár pohne, ak bude mať na výber ľubovoľný smer otočenia. Pohybový vektor pre mapu 12x10 s 6 kameňmi môže vyzerať takto:

[36, 2, 24, 1, 29, 14, 16, 3, 18, 26, 19, 4, 31, 8, 11, 34, 10, 6, 40, 35, 22, 30, 1, 1, 0, 0, 0, 0]

Indexy vo vektore reprezentujú miesto na obvode mapy podľa obrázka 3. Ak je číslo vo vektore na hornej strane, tak začíname pohybom dole a naopak. Rovnako to platí aj pre pravú a ľavú stranu.




*Obrázok 3 Indexovanie obvodu mapy*

Pohyb zo štartu záhradník vykonáva v cykle, dokým nevyjde z hracej plochy (posunieme sa na ďalšiu pozíciu vo vektore) alebo v komplikovanejšom prípade natrafíme na kameň, nezobratelný list alebo už na pohrabané miesto. Ak takéto niečo nastane, tak pomocou ukazovateľa na miesta, kde sa rozhoduje kam sa otočí, sa rozhodne či pôjde doľava alebo doprava. Ak má na výber iba jeden smer, tak sa otočí naň. V prípade, že sa nemá kam otočiť alebo prešiel všetky štarty, tak generovanie ciest končí a vráti vyplnenú mapu a záhradníka (počet pozbieraných listov).

Pri mnohých štartovných hodnotách môže nastať, že štartovná pozícia vo vektore bude taká, že záhradník nebude môcť vstúpiť na plochu. V prípade, že takýto scenár nastane, tak v nastaveniach máme dve možnosti – ukončíme generovanie mapy alebo preskočíme túto pozíciu. Pre tento prípad je možné nastavenie skipUnStartable v conf.ini (ako aj v programe)

na 1 – preskakuj hodnoty 0 – nepreskakuj. V prípade, že máme preskakovanie zapnuté, tak číslovanie ciest je stále pôvodné, resp. index v pohybovom vektore + 1.

## 2.4. Generovanie okolia

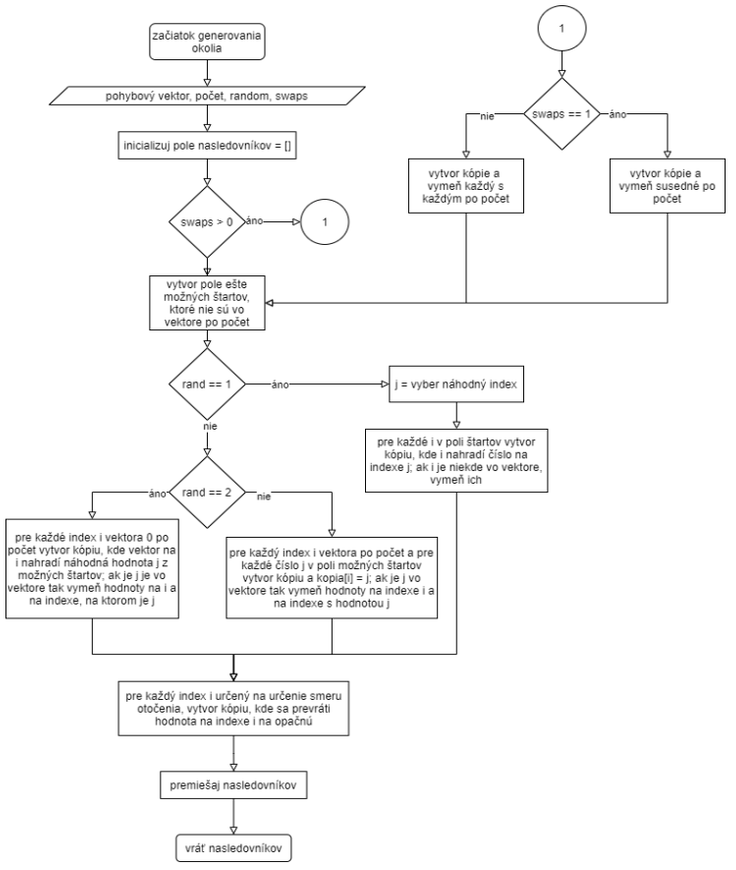
Pri generovaní okolia pohybového vektora sme sa rozhodli dať viac priestoru pre testovanie rozličných metód a naše generovanie prebieha v troch etapách. Vo všetkých etapách meníme hodnoty na indexoch, z ktorých hrabáč vyrazil (resp. indexy, ktoré sú menšie ako je maximálne číslo cesty na mape). V prvej etape (ktorá sa dá preskočiť) generujeme okolie výmenou hodnôt vo vektore. Tu je možné si vybrať či chceme vytvoriť všetky možné kombinácie prehodení hodnoty na jednom indexe s iným indexom alebo iba vymeniť susedné pozície. Pri výmene susedných pozícií vzniká počet nasledovníkov rovný počtu vykonaných ciest - 1. Pri generovaní všetkých vzniká razantne viac prehodení a to $\sum_{i=1}^{n}(n-i)$, kde $n$ je počet ciest, ktoré hrabáč prešiel v súčasnom vektore. Vývoj veľkosti je zobrazený v tabuľke 1.

| Počet ciest | Počet nasl. so všetkými možnosťami |
|---|---|
| 5 | 10 |
| 10 | 45 |
| 40 | 190 |
| 80 | 780 |
| 120 | 1770 |
| 160 | 12720 |

*Tabuľka 1 nárast nasledovníkov*

V druhej etape sa nahrádzajú čísla na indexoch vektora (po počet vykonaných ciest) iným štartovacím číslom, ktoré ešte nie je v pohybovom vektore (prípadne číslami, ktoré sa nedostali na rad v súčasnom vektore), teda prinášame úplne nový štart pre záhradníka. Tu si tiež môžeme vybrať medzi náhodným generovaním (random = 2), generovaním všetkých možnosti na jednom náhodnom indexe (random = 1) alebo generovaním všetkých možných kombinácií (random = všetky ostatne hodnoty). Pri náhodnom vytváraní sa do vektoroch nasledovníka prepíše práve jedno číslo na nové. Každý takto vytvorený nasledovník má vymenené číslo na inom políčku vo vektore. Druhá možnosť, vytvorenie všetkých náhrad za náhodný index, vytvorí nasledovníkov tak, že vyberie náhodný index po počet vykonaných ciest a vymení ho s číslami, ktoré sa nedostali na radu alebo nie sú vo vektore. Poslednou možnosťou je vytvorenie všetkých kombinácií, vytvorí nasledovníkov tak, že pre každé políčko a každé číslo, ktoré nie je v pôvodnom vektore, vytvorí nasledovníka, ktorý má toto číslo na i-tom poličku.

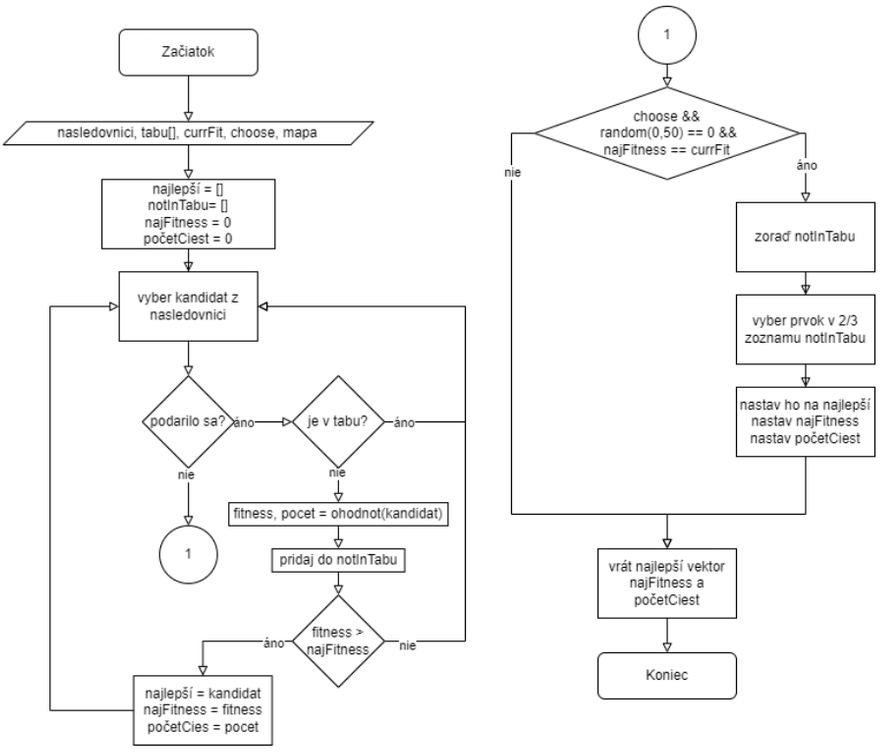
V poslednej etape sa menia čísla, podľa ktorých sa rozhoduje kam zabočí, resp. pre každé políčko určujúce smer vytvor nasledovníka, kde bude práve jedno políčko obrátenú hodnotu. Nakoniec celé pole nasledovníkov zamiešame, aby sa nestalo, že algoritmus bude preferovať iba výmeny susedov.




*Obrázok 4 diagram generovania okolia*

## 2.5.Výpočet fitness a výber najlepšieho suseda

Fitness sa počíta tak, že počas vypĺňania mapy si záhradník počíta políčka, ktoré prešiel. V prípade, že sú na mape aj listy, tak hodnota listov je rozličná podľa farby listov. Jeden žltý list ma hodnotu dvojnásobku maximálnej fitness, ktorú môžeme dostať z políčok. Oranžový má dvojnásobok hodnoty žltého listu a červený je dvojnásobok oranžového listu. Podľa týchto hodnôt vyberáme najlepšieho nasledovníka.

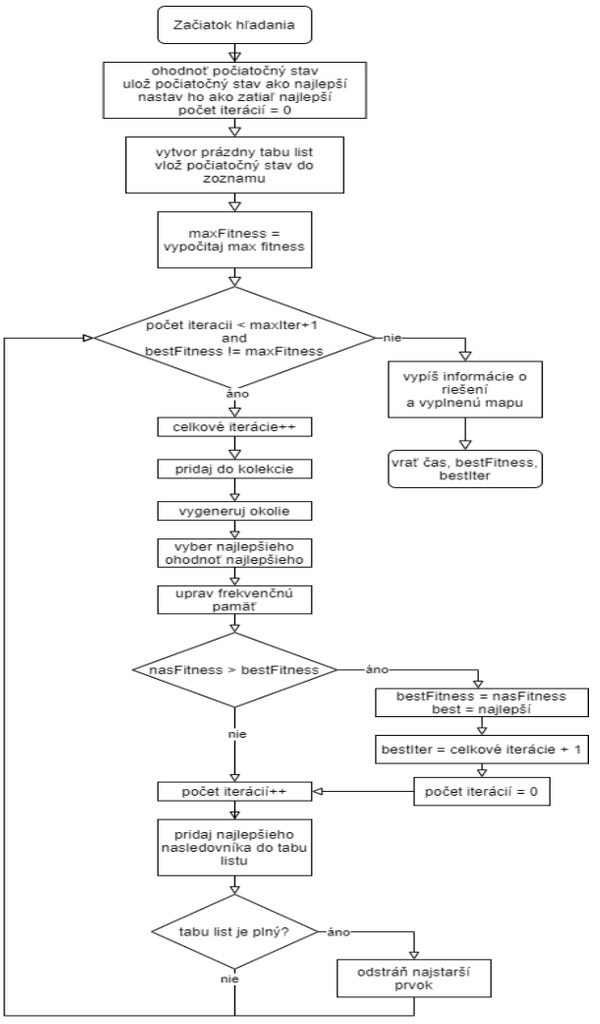



*Obrázok 5 algoritmus na výber najlepšieho*

Najlepšieho nasledovníka si vyberáme tak, že pre každého nasledovníka skontrolujeme či jeho vektor je v tabu liste a či má lepšiu fitness ako doposiaľ najlepší nasledovník. Ak sú všetky podmienky splnené, tak sa ako novým najlepším nasledovníkom stane najlepšou možnosťou. Ak najlepší nasledovník nie je lepší ako súčasný (resp. najlepší nasledovník vykonal zmenu vo vektore, ktorá nemala dopad na výslednú mapu), tak si vieme nastaviť aby s pravdepodobnosťou 1/50, aby bol vybratý nasledovník, ktorý je medzi 1/3 najlepších nasledovníkov, ktorý nie sú v tabu liste.

## 2.6.Hľadanie

Hľadanie začína tým že vytvoríme počiatočný pohybový vektor vypočítame jeho fitness a nastavíme ho na zatiaľ na najlepšie nájdené riešenie. Následne vstupujeme do cyklu ktorý, končí iba vtedy, ak počet iterácií bez vylepšenia presiahne maximálnu hodnotu, ktorú si nastavíme alebo vtedy ak nájdeme riešenie, ktorého ohodnotenie bude maximálne. V cykle vytvoríme nasledovníkov, vyberieme z neho najlepšieho. Ak si vyberieme možnosť s frekvenčnou pamäťou, tak sa nájde to číslo a pozíciu, na ktorú sme ho pridali a pripočítame výskyt do tabuľky na indexe pozícia a číslo jednotku. Následne sa zistí či je nasledovník lepší ako zatiaľ najlepší vektor. Ak áno, tak sa nastaví na najlepšieho resetuje sa počítadlo iterácií bez vylepšenia. Potom sa pridá vektor nasledovníka do tabu list, nastaví sa ako práve spracovávaný vektor a vracia sa na začiatok cyklu. Samotný tabu list pozostáva z pohybových vektorov




*Obrázok 6 diagram funkcie hľadania*

Ak máme zapnutú frekvenčnú pamäť, tak ešte predtým ako sa začne vykonávať hlavný cyklus, inicializuje sa frekvenčná pamäť. Tá je reprezentovaná ako dvojrozmerné pole, kde prvý index predstavuje index štartu v pohybovom vektore a druhý index je samotný štart na mape. Na začiatku sú všetky inicializované na 0, ale počas behu programu sa bunky v tabuľke inkrementujú podľa toho koľkokrát sa číslo na pozícii objavilo v najlepšom nasledovníkovi. Frekvenčná pamäť slúži na opakované hľadanie riešenia nad rovnakou mapou. Podľa frekvenčnej mapy vytvoríme počiatočný vektor z najmenej používaných hodnôt.

## 2.7. Používateľské rozhranie

Používateľské prostredie sa skladá zo súboru **conf.ini** (obrázok 7.), v ktorom sú uložené nastavenia tabu search ako aj parametre na vytváranie náhodných máp. Tieto nastavenie sa dajú meniť aj počas behu programu (obrázok 8. a 9.).

```
[Search]
tabuMaxSize = 500   // velkost tabu listu
maxIteracie = 600   //maximalny pocet iteracii bez zlepsenia
numOfRepeats = 5    // pocet opakovani frekvencnej
skipUnStartable= 0  //preskoc nezacinatelne
chooseWorse=0       // vyber horsieho v 2/3 nasledovnikov

minTabuSize= 10   // minimalna velkost tabu listu
maxTabuSize= 100  //maximalna velkost tabu listu
tabuStep=10       //krok zvacsenia tabu listu

[swaps]
enable=0     // 0 - preskocenie etapy 1;
             // 1 - prehadzovanie susedov
             // 2> prehadzovanie vsetkych

[neighbors]
random = 0  // 0 - generovanie vsetkych
            // 1 - generovanie nahodnych
            // 2 - generovanie vsetkych pre jeden index


// nastavenia generovania mapy
[Stones]
count = 7      // pocet kamenov

[Leaf]
yellow = 4     // pocet zltych listov
orange = 7     // pocet oranzovych listov
red = 8        // pocet cervenych
```

*Obrázok 7 parametre v conf.ini*

Hodnota 0 pri parametroch skipUnStartable. chooseWorse znamená False, ak je tam iné číslo, tak je to True. Tieto hodnoty rovnako fungujú aj pri menení parametrov za behu programu. Teda tam, kde je na obrázkoch False/True tak pri ich menení treba zadať číslo 0 alebo iné. Ďalej ak pri menení nezadáme hodnotu, tak hodnota zostane pôvodná.

```
Velkost tabu listu:  100
Nahodny nasledovnici:  1
max. pocet iteracii bez vylepsenia:  300
maximalny pocet opakovani hladania riesenia
preskakovat zle pozicie:  False
minimalna velkost tabuListu:  80
maximalna velkost tabuListu:  100
krok zvacsenia:  10
vymena:  1
vyber horsieho:  False
```

*Obrázok 8 nastavenia search-u v aplikácií*

```
pocet kamenov:  7
------------
pocet zltych:  4
pocet oranzovych:  7
pocet cervenych:  8
chcete zmenit parametre? (y/n):
```

*Obrázok 9 nastavenia generovania máp*

V hlavnom menu (obrázok 10.) používateľ zadá číslo alebo písmeno operácie, ktorú chce vykonať. Prepínač **p** vypíše parametre tabu searchu, **n** vypíše menu s natavením mapy (obrázok 11.). Číslo 0 zavrie aplikáciu, 1 spustí test, kde používame frekvenčnú pamäť, 2 je test, kde porovnávame riešenia preskakovania neplatných pozícií a skončenie na neplatných riešeniach. Posledný prepínač je 3, ktorý spustí test, kde zisťujeme efektivitu rôznych veľkostí tabu listu.

```
vyberte si z možnosti:
0 - exit
p - parametre tabu search algoritmu
n - nacitaj mapu
1 - test s frequency memory
2 - porovnanie preskakovania neplatnych krokov vs nepreskakovanie
3 - postupne zvacsovanie tabu listu a poctu maximalneho poctu do najdenia lepsieho riesenia
$
```

*Obrázok 10 hlavné menu*

V druhom menu (obrázok 11.) má používateľ na výber niekoľko možnosti ako sú 1 – otvorí menu, kde si používateľ vyberie jednu z máp zadaných v zadaní (obrázok 12.). 2 si vyberie náhodne vygenerovanú mapu podľa parametrov, ale musí zadať jej dĺžku a výšku. 3 – vypíše parametre generovania mapy (obrázok 9.), kde ich môže aj zmeniť. 4 – zapne načítavanie mapy manuálnym zadaním veľkosti, počtu a pozícií kameňov a listov (obrázok 13.). Pri zadávaní pozícií treba zadať stĺpec,riadok presne v tomto poradí. Pozície, ktoré sú mimo mapy sa preskočia a bude menej kameňov/listov na mape. To platí aj keď už kameň alebo list zabral pozíciu na mape, tak nebude prepísaný inou prekážkou.

```
------------------
moznosti:
0 - vrat sa spat
1 - vyber default mapu
2 - vygeneruj
3 - parametre generovania
4 - manualne zadat mapu
5 - vypis nacitanu mapu
Zadajte moznost:
```

*Obrázok 11 menu pre načítavanie máp*

```
moznosti:
1 - zadanie - bez listov
2 - zadanie - s listami
Zadajte moznost:
```

*Obrázok 12 prednastavene mapy*

```
Zadajte moznost: 5
Zadajte dlzku: 10
Zadajte vysku: 10
Zadajte pocet kamenov: 3
zadaj poziciu v tvare x,x: 5,5
zadaj poziciu v tvare x,x: 2,3
zadaj poziciu v tvare x,x: 4,8
Zadajte zltych listov: 2
zadaj poziciu v tvare x,x: 0,1
zadaj poziciu v tvare x,x: 2,0
Zadajte oranzovych listov: 1
zadaj poziciu v tvare x,x: 9,4
Zadajte pocet cervenych: 0
```

```
* * * * * * * * * * * *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 K 0 0 0 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 0 0 0 K 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* 0 0 0 0 K 0 0 0 0 0 *
* 0 0 0 0 0 0 0 0 0 0 *
* * * * * * * * * * * *
```

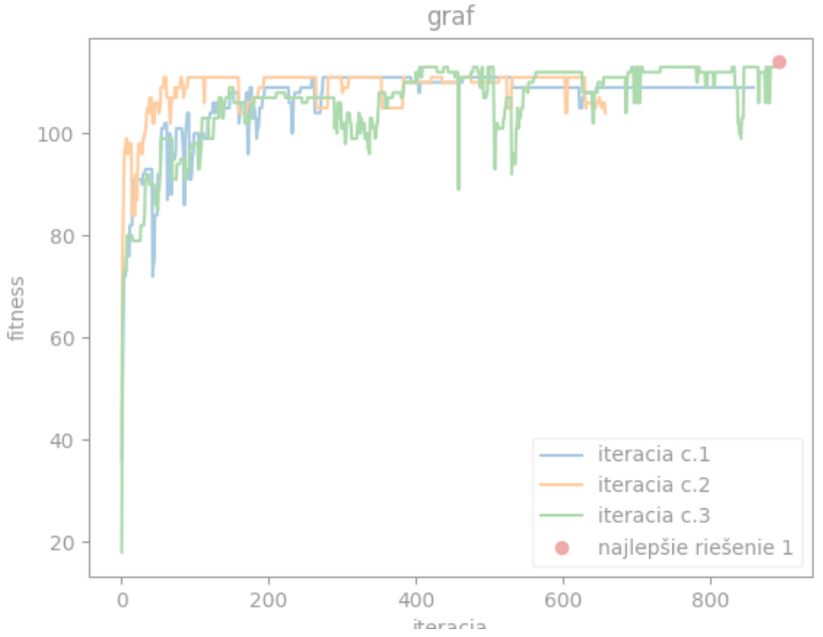
*Obrázok 13 manuálne generovanie mapy a vygenerovaná mapa*

# 3. Testovanie

V prvom testovacom scenári je vykonanie 20 iterácií hľadania s pomocou dlhodobej pamäti nad základnou mapou bez listov. V tomto scenári sme chceli otestovať korektnosť hľadania riešenia naprieč viacerými iteráciami zakázaného prehľadávania s použitím frekvenčnej pamäti. Ako parametre vyhľadávania sme pre tento test mali nastavené maximálny počet iterácií bez vylepšenia na 600 a veľkosť tabu listu sme nastavili na 100. Na generovanie nasledovníkov sme použili iba generovanie všetkých možností pre jeden index vektora, bez výmen. Na obrázkoch 14. a 15. je možné vidieť že pomocou tohto riešenia sa nám podarilo nájsť riešenie v celku skorej iterácií hľadania, ale hľadanie nemusí vždy takto rýchlo nájsť riešenie.

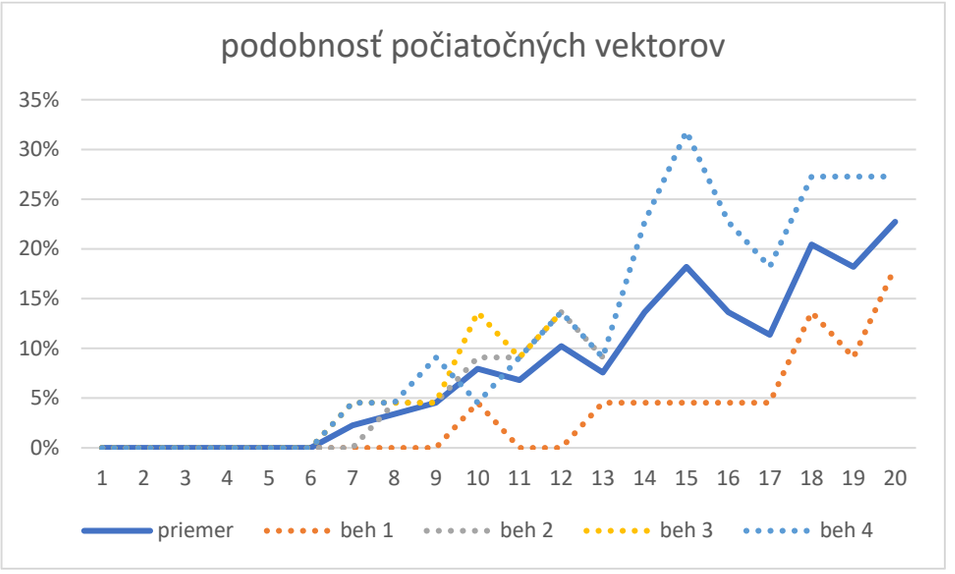
```
------------------------------------

iteracia c.1
počet všetkých iteracií:  858
najlepšie riešenie v iteacií:  259
najlepsi vektor:  [14, 18, 34, 3, 10, 41, 15, 42, 8, 39, 33, 43, 16, 13, 38, 2, 19, 28, 6, 1, 11, 35, 0, 1, 1, 0, 0, 1]
fitness: 111/114
*  *  *  *  *  *  *  *  *  *  *  *  *  *
*  3  3  1  4  0  4  0  5  9  9  5  14 *
*  3  3  1  4  4  4  K  5  5  5  5  14 *
*  0  K  1  1  1  1  1  1  1  1  1  1  *
*  10 10 10 10 K  7  7  7  7  7  7  7  *
*  15 15 K  10 10 7  13 13 13 13 13 13 *
*  10 10 10 10 10 7  13 13 13 13 13 13 *
*  7  7  7  7  7  7  13 13 K  K  2  2  *
*  6  6  6  6  6  6  6  6  6  6  2  11 *
*  8  8  8  8  8  8  8  8  8  6  2  11 *
*  12 12 12 12 12 12 12 12 8  6  2  11 *
*  *  *  *  *  *  *  *  *  *  *  *  *  *
cas :  53.44938778877258 s

iteracia c.2
počet všetkých iteracií:  657
najlepšie riešenie v iteacií:  58
najlepsi vektor:  [41, 1, 23, 21, 43, 39, 16, 14, 13, 34, 2, 12, 38, 25, 19, 36, 27, 24, 33, 31, 20, 3, 0, 1, 0, 0, 0, 1]
fitness: 111/114
*  *  *  *  *  *  *  *  *  *  *  *  *  *
*  10 2  11 11 11 11 11 9  12 12 12 12 *
*  10 2  2  2  2  2  K  9  9  9  9  9  *
*  10 K  6  6  0  2  8  8  8  8  8  8  *
*  6  6  6  6  K  2  8  8  8  8  8  8  *
*  13 13 K  6  6  2  7  7  7  7  7  7  *
*  6  6  6  6  6  2  7  7  7  7  7  7  *
*  2  2  2  2  2  2  7  7  K  K  0  0  *
*  1  1  1  1  1  1  1  1  1  1  1  1  *
*  5  3  3  3  3  3  3  3  3  3  3  3  *
*  5  3  4  4  4  4  4  4  4  4  4  4  *
*  *  *  *  *  *  *  *  *  *  *  *  *  *
cas :  45.33074069023132 s

iteracia c.3
počet všetkých iteracií:  892
najlepšie riešenie v iteacií:  893
najlepsi vektor:  [14, 13, 34, 27, 33, 22, 12, 32, 26, 31, 25, 4, 6, 43, 28, 38, 40, 9, 37, 36, 16, 5, 0, 0, 1, 0, 0, 0]
fitness: 114/114
*  *  *  *  *  *  *  *  *  *  *  *  *  *
*  3  3  1  12 12 13 13 2  7  7  7  7  *
*  3  3  1  12 12 13 K  2  2  2  2  2  *
*  6  K  1  1  1  1  1  1  1  1  1  1  *
*  6  9  9  9  K  4  4  4  4  4  4  4  *
*  6  9  K  9  9  4  5  5  5  5  5  5  *
*  6  9  11 11 9  4  5  8  8  8  8  5  *
*  6  9  11 11 9  4  5  8  K  K  8  5  *
*  6  9  11 11 9  4  5  8  10 10 8  5  *
*  6  9  11 11 9  4  5  8  10 10 8  5  *
*  6  9  11 11 9  4  5  8  10 10 8  5  *
*  *  *  *  *  *  *  *  *  *  *  *  *  *
cas :  58.32144236564636 s
```

*Obrázok 14 úspešné prehľadanie mapy zo zadania*




*Obrázok 15 priebeh fitness pre úspešne prehľadanie z obrázka 10*

Keďže sa snažíme týmto diverzifikovať hľadanie riešenia, tak sme si porovnali generovanie počiatočných vektorov ako moc sa podobajú na niektorý z predchádzajúcich iterácií hľadania. Ako je vidieť na obrázku 16. tak prvé iterácie sú vždy úplne rozličné od predchádzajúcich, ale ako sa opakuje hľadanie, tak aj väčšina čísiel už zabrala niektorú pozíciu vo vektore a teda podobnosť počiatočných vektorov stúpa. Stúpa pomaly do 20 iterácie, podobnosť bola do 25%, čo je približne 6 štartov z 22, ktoré sú rovnaké ako niektorý predchádzajúci.




*Obrázok 16 rast podobnosti počiatočných vektorov*

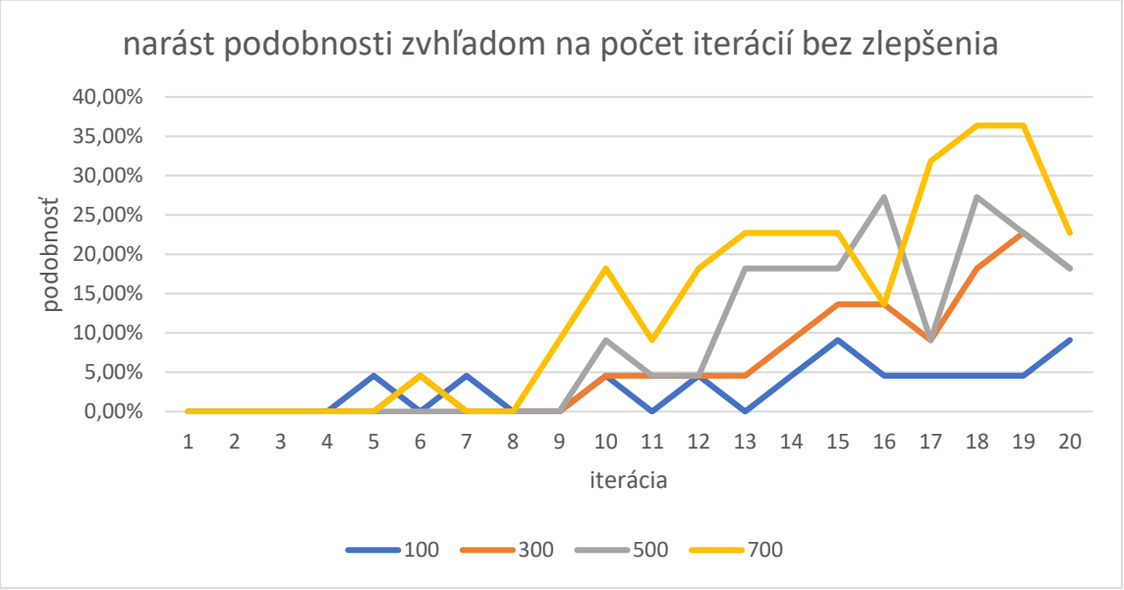
Nižšie je uvedená tabuľka 2. podobností naprieč iteráciami. Iterácie 1 až 6 sú združené do jedného riadka, nakoľko všetky behy hľadania mali počiatočný vektor unikátny. Ako je vidieť tak pomocou frekvenčnej pamäti sa nám podarilo nájsť riešenia v dvoch behoch v 12. a 13. iterácií hľadania. Celkovo bude podobnosť stúpať, čo ale neznamená, že ak sú vektory podobné, že to nebude lepšie hľadanie ako predchádzajúce podobné.

*Tabuľka 2 nárast podobnosti pre 4 behy hľadania s dlhodobou pamäťou*

| iterácia | beh 1 | beh 2 | beh 3 | beh 4 | priemer |
|---|---|---|---|---|---|
| 1 - 6 | 0% | 0% | 0% | 0% | 0 |
| 7 | 0% | 0% | 4,55% | 4,55% | 0,022725 |
| 8 | 0% | 4,55% | 4,55% | 4,55% | 0,034088 |
| 9 | 0% | 4,55% | 4,55% | 9,09% | 0,045453 |
| 10 | 4,55% | 9,09% | 13,64% | 4,55% | 0,079543 |
| 11 | 0% | 9,09% | 9,09% | 9,09% | 0,068183 |
| 12 | 0% | 13,64% | 13,64% | 13,64% | 0,10227 |
| 13 | 4,55% | 9,09% | | 9,09% | 0,075757 |
| 14 | 4,55% | | | 22,73% | 0,13636 |
| 15 | 4,55% | | | 31,82% | 0,181815 |
| 16 | 4,55% | | | 22,73% | 0,13636 |
| 17 | 4,55% | | | 18,18% | 0,113635 |
| 18 | 13,64% | | | 27,27% | 0,204545 |

| | | | | | |
|---|---|---|---|---|---|
| 19 | 9,09% | | | 27,27% | 0,18182 |
| 20 | 18,18% | | | 27,27% | 0,227275 |

Podobne sme to chceli otestovať aj pre zväčšovanie maximálneho počtu iterácií bez vylepšenia. V nasledujúcej tabuľke 3. a grafe 17. je vidieť ako podobnosť narastala pre hľadania, v ktorých maximálne iterácie bez vylepšenia boli väčšie. Toto je vysvetliteľné tým, že do frekvenčnej tabuľky sa počas hľadania zapisujú aj výskyty, ktoré nie sú súčasťou najlepšieho riešenia. Resp. jedná sa o dobeh algoritmu, v ktorom sa už nenájde lepšie riešenie a musia sa zapísať do frekvenčnej pamäte aj hodnoty týchto stavov.

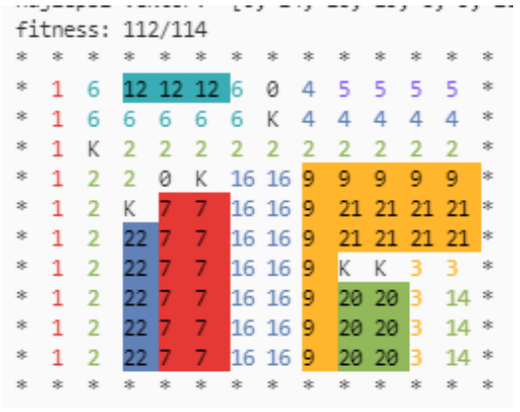



*Obrázok 17 podobnosť vzhľadom na max. počet iter. bez zlepšenia.*

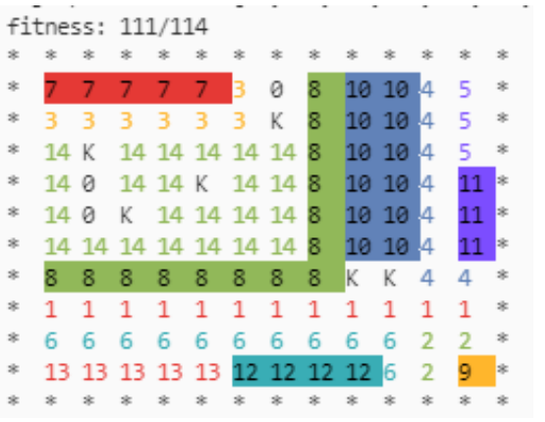
*Tabuľka 3 nárast podobnosti v percentách*

| iterácia | 100 | 300 | 500 | 700 |
|---|---|---|---|---|
| 1 - 5 | 0,00% | 0,00% | 0,00% | 0,00% |
| 6 | 0,00% | 4,55% | 0,00% | 4,55% |
| 7 | 4,55% | 0,00% | 0,00% | 0,00% |
| 8 | 0,00% | 0,00% | 0,00% | 0,00% |
| 9 | 0,00% | 0,00% | 0,00% | 9,09% |
| 10 | 4,55% | 4,55% | 9,09% | 18,18% |
| 11 | 0,00% | 4,55% | 4,55% | 9,09% |
| 12 | 4,55% | 4,55% | 4,55% | 18,18% |
| 13 | 0,00% | 4,55% | 18,18% | 22,73% |
| 14 | 4,55% | 9,09% | 18,18% | 22,73% |
| 15 | 9,09% | 13,64% | 18,18% | 22,73% |
| 16 | 4,55% | 13,64% | 27,27% | 13,64% |
| 17 | 4,55% | 9,09% | 9,09% | 31,82% |
| 18 | 4,55% | 18,18% | 27,27% | 36,36% |
| 19 | 4,55% | 22,73% | 22,73% | 36,36% |
| 20 | 9,09% | 18,18% | 18,18% | 22,73% |

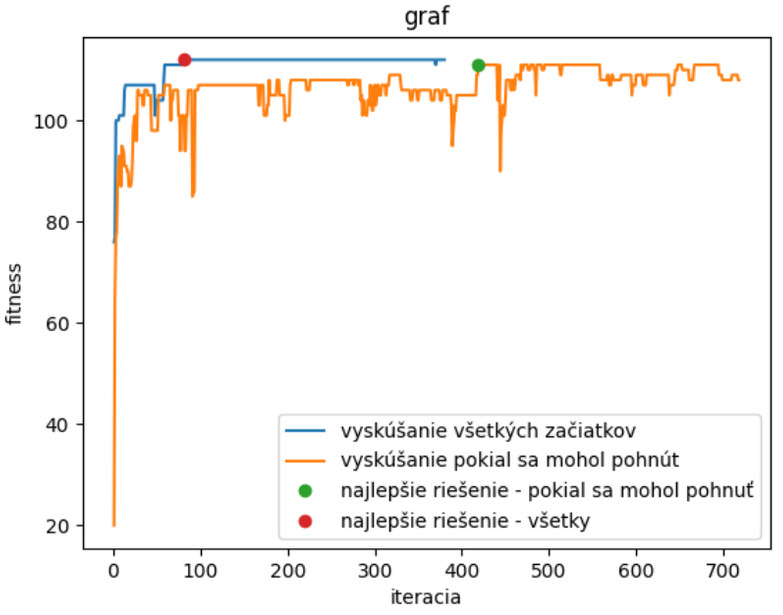
Ďalší scenár je porovnanie preskakovania alebo nepreskakovania počiatočných pozícií s tým, že skúšame rôzne nami definované kombinácie generovania nasledovníkov. Všetky kombinácie sme generovali s veľkosťou tabu listu 100 a maximálnymi iteráciami 300. Na obrázkoch 18. až 20. je riešenie pomocou generovania okolia bez výmen susedov, len za pomoci generovania všetkých chýbajúcich čísel na náhodnom indexe (Metóda 1.). Celkovo toto riešenie generovania malo dobré pre lepšiu diverzifikáciu pri skončení po bez vyskúšania všetkých štartov, pretože si väčšinou nemohol vybrať nasledovníka, ktorý nijako neovplyvnilo výsledok.




*Obrázok 18 preskovanie povolené*



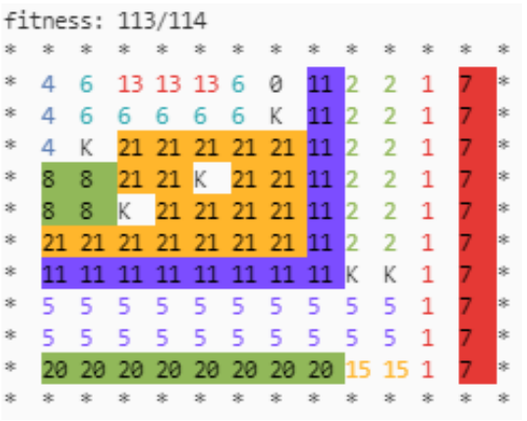

*Obrázok 19 preskakovanie zakázané*



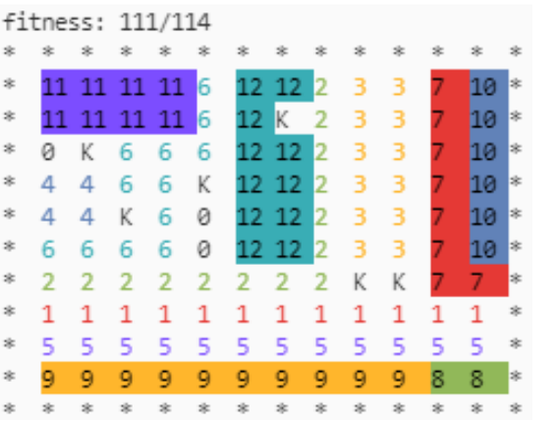

*Obrázok 20 vývoj fitness*

Ďalej na obrázkoch 21. až 22. je použité generovanie kópií, kde na jednej kópií je jeden index nahradený náhodnou hodnotou. Rovnako ako predošlá metóda, tak ani táto nemala pri pohyboch pokiaľ mohol na výber nasledovníka, ktorý nijako neovplyvnil výsledok.

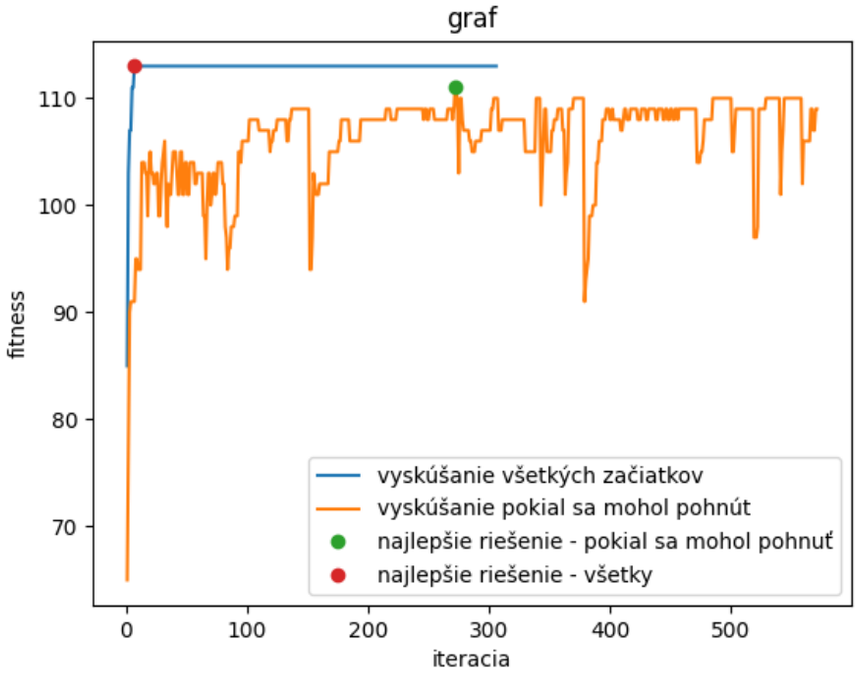
Róbert Junas
102970




*Obrázok 21 preskakovanie povolené*



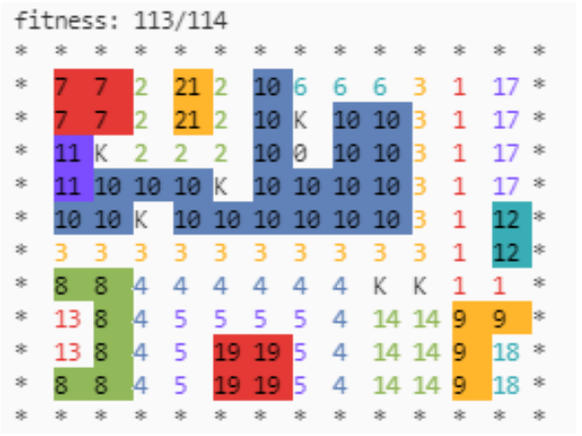

*Obrázok 22 preskakovanie zakázané*



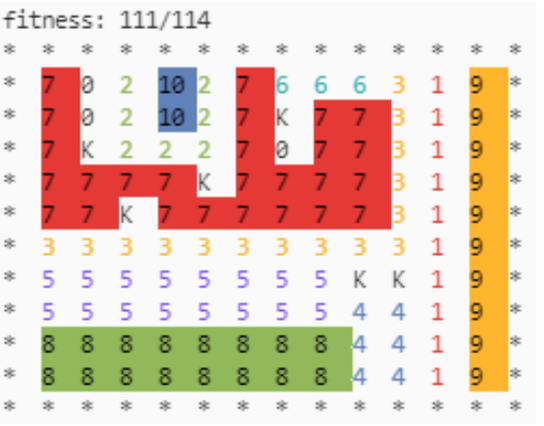

*Obrázok 23 vývoj fitness*

Obidve predošlé metódy boli rýchle, pretože generovali relatívne málo možných nasledovníkov, ale generovanie (obrázky 24. až 26.), kde sme pre každý index vo vektore a pre každé číslo, ktoré sa nedostalo na radu/vo vektore vôbec nebolo, toto vyhľadávanie trvalo razantne dlhšie. Pri tomto generovaní (Metóda 3.) sa našlo lokálne minimum dosť skoro pre obidve metódy vyplňovania mapy, ale počas behu hľadania zostal v tomto lokálnom minime do skončenia hľadania.

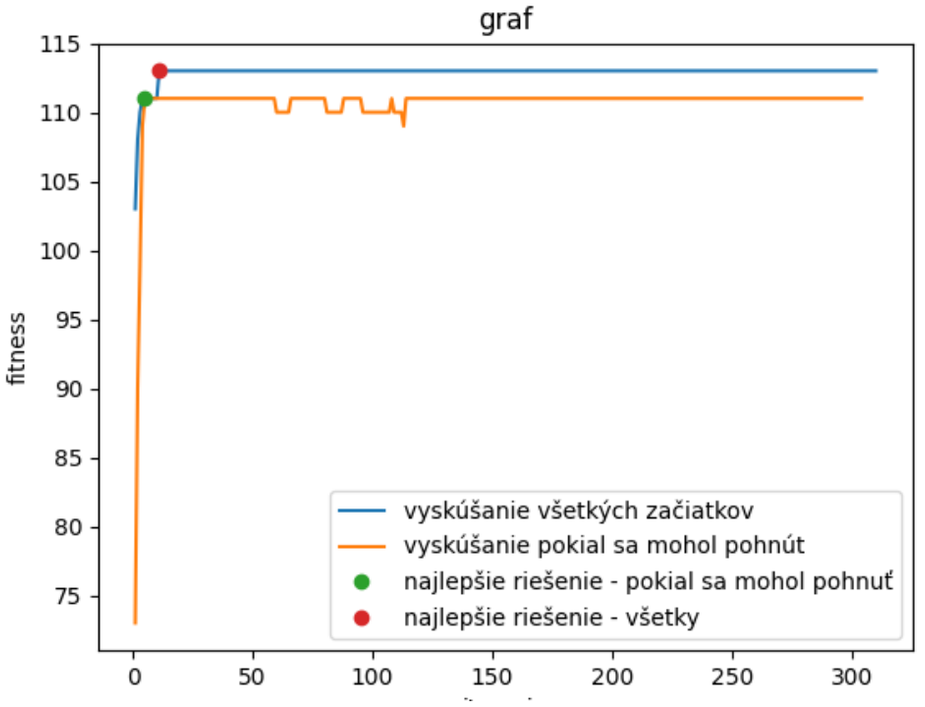
Róbert Junas
102970




*Obrázok 24 preskakovanie povolené*

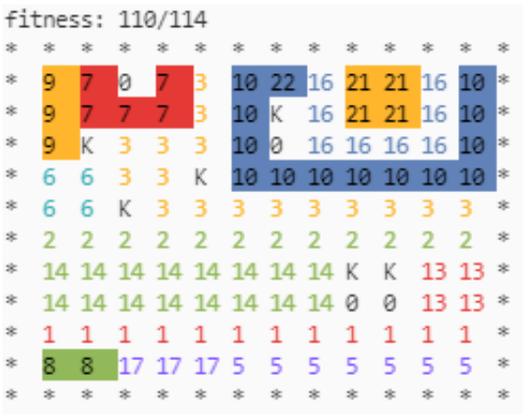



*Obrázok 25 preskakovanie zakázané*

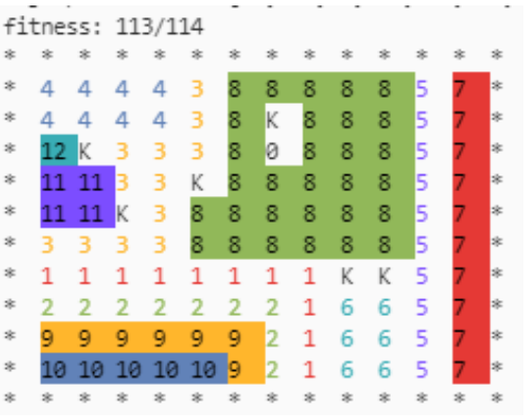



*Obrázok 26 vývoj fitness*

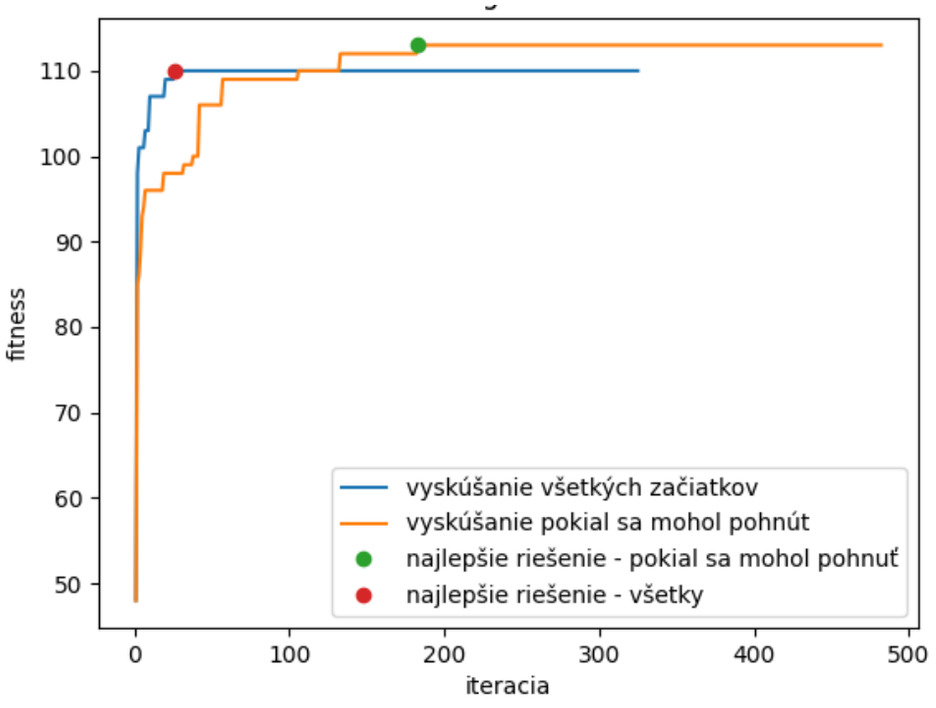
Ďalej sme za pomoci prvej metódy generovania okolia, použili aj výmenu susedov a výmenu každého s každým. Na obrázkoch 27. až 29. je vidieť výsledky výmenou susedných štartov (Metóda 4.) vo vektore po počet vykonaných ciest. Ako je vidieť tak obidve metódy vyplňovania mapy vyberajú takých nasledovníkov, ktorý nie sú horší ako samotné súčasné lokálne maximum, resp. ak existuje taká výmena dvoch štartov, ktoré neovplyvnia vyplnenie mapy a ani nasledovník, ktorý ho nezlepší tak sa zasekne na lokálnom maxime.




*Obrázok 27 preskakovanie povolené*



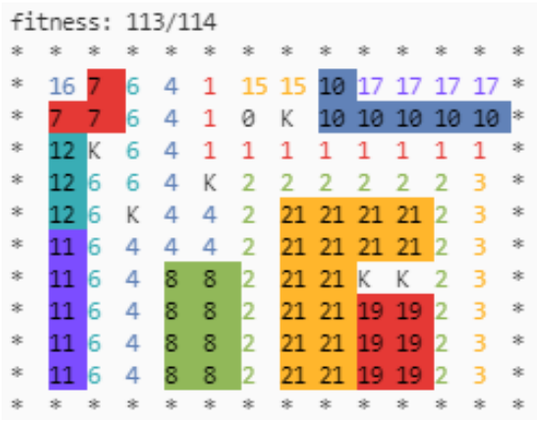

*Obrázok 28 preskakovanie zakázané*




*Obrázok 29 vývoj fitness*

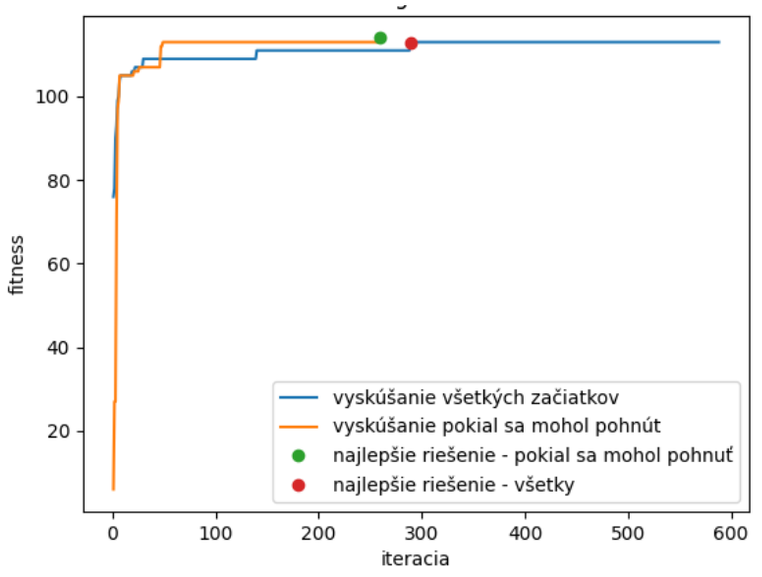
Posledná metóda generovania nasledovníkov je generovanie pomocou metódy 1. s generovaním nasledovníkov výmenou každého s každým po počet vykonaných ciest (Metóda 5.; obrátky 30. až 32.). Podobne ako metóda 3., tak aj toto generovanie trvá dlhšie, podobne ako nájdenie riešenia s metódou 4. V tejto metóde sa podarilo hľadaniu bez preskakovania nájsť riešenie, ktoré kompletne vyplnilo mapu (nemusí vždy platiť). Vývoj fitness rovnako ako v prípade metódy 4., vyberá nasledovníka, ktorý nezhoršil ohodnotenie mapy, resp. vyberá podobné mapy alebo lepšie.

Róbert Junas
102970




*Obrázok 30 preskakovanie povolené*

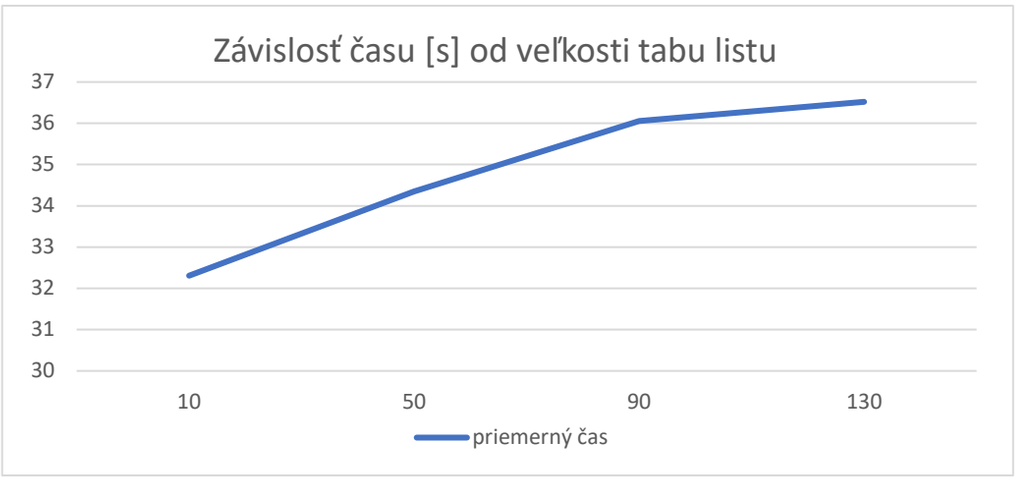

*Obrázok 31 preskakovanie zakázané*



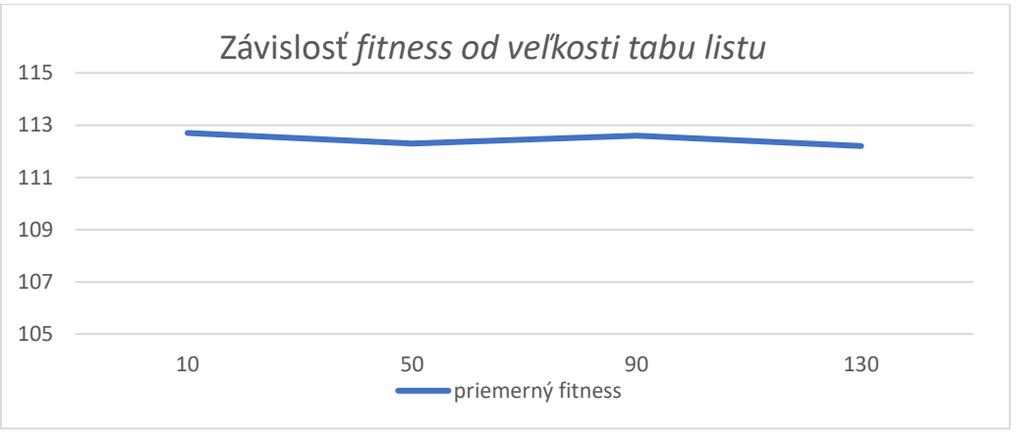

*Obrázok 32 vývoj fitness*

Celkovo sa preskakovanie a nepreskakovanie s výmenami udržujú v rovnakom alebo podobnom vyplnení mapy, ktoré neovplyvňuje vývoj fitness a teda sa ukončí hľadania aj skôr, nakoľko nie vždy existuje v „lokálnom maxime“ stav ktorý zlepší jeho ohodnotenie. Na druhú stranu vyplňovanie bez preskakovania bez výmen je pomalšie, ale vyberá si stavy, ktoré ovplyvňujú fitness aj negatívne.

Posledný scenár, ktorý sme sa rozhodli otestovať je opakované hľadanie riešenia nad rovnakým vektorom tak, že postupne zvyšujeme veľkosť tabu listu. Pre tento test sme museli algoritmus generovania upraviť tak, aby sa správal rovnako pri každej iterácii zväčšenia. Teda všetky náhodne generovane prvky sme upravili tak, aby nemali dopad na priebehy hľadania. V grafoch 33. a 34. je môžné vidieť priemerný čas a priemernú fitness riešení. Priemer sme dostali z behu 10 rôznych počiatočných vektorov. Testované veľkosti boli 10, 50, 90, 130. Maximálny počet iterácií bez vylepšenia bol 600 a na generovanie nasledovníkov sme použili iba generovanie celej výmeny pre náhodný index bez použitia výmeny susedov (Metóda 1. z predchádzajúceho testovania). Celé riešenie prebiehalo na základnej mape zo zadania.




*Obrázok 33 Závislosť času [s] od veľkosti tabu listu*



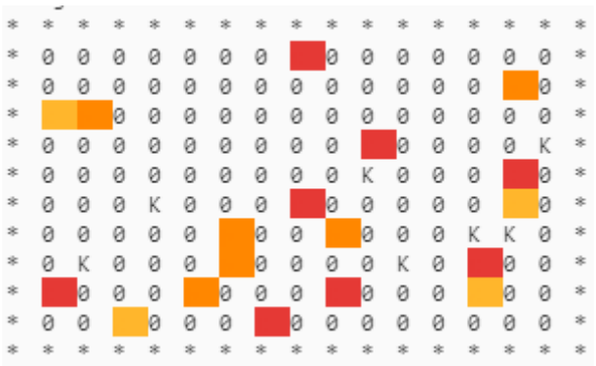

*Obrázok 34 Závislosť fitness od veľkosti tabu listu*

*Tabuľka 4 riešenia vzhľadom na veľkosť tabu listu*

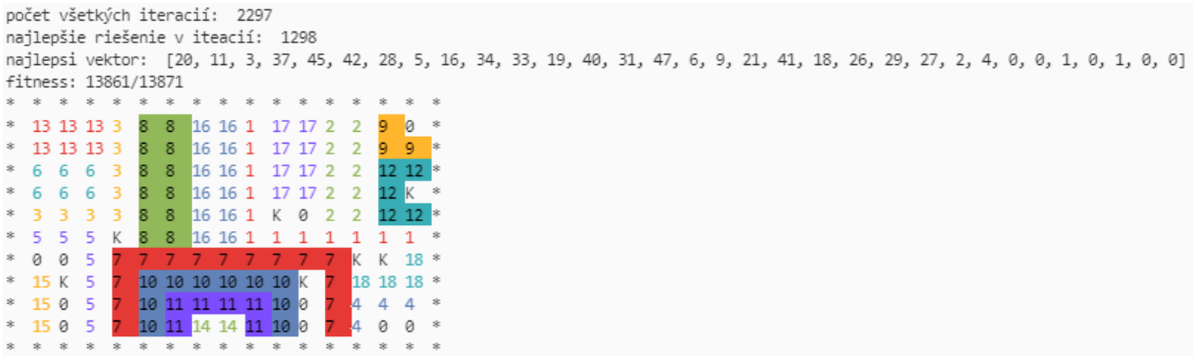
| | Veľkosť tabu listu | | | |
|---|---|---|---|---|
| vektor: | 10 | 50 | 90 | 130 |
| 1 | 113 | 113 | 113 | 113 |
| 2 | 113 | 113 | 113 | 113 |
| 3 | 113 | 112 | 112 | 112 |
| 4 | 113 | 110 | 113 | 112 |
| 5 | 113 | 113 | 112 | 113 |
| 6 | 113 | 112 | 114 | 113 |
| 7 | 112 | 113 | 111 | 110 |
| 8 | 113 | 112 | 113 | 110 |
| 9 | 111 | 113 | 112 | 113 |
| 10 | 113 | 112 | 113 | 113 |

Ako je na výsledkoch v predchádzajúcich grafoch a tabuľke vidieť, tak čas prehľadávania narastal, ale výsledky boli porovnateľne rovnaké. Jediné kompletné riešenie sa našlo pri iterácií 6 na veľkosti tabuľky 90. Ďalej, sme sa rozhodli vykonať rovnaký test aj pre mapu 15x10 (obrázok 35), ale s tým rozdielom, že sme sa rozhodli použiť veľkosti tabu listu

od 100 po 300 s tým že ho zväčšujeme po 100 prvkoch. Ďalej sme zväčšili počet maximálnych iterácií bez zlepšenia zo 600 na 1000.



*Obrázok 35 mapa pre väčší test*



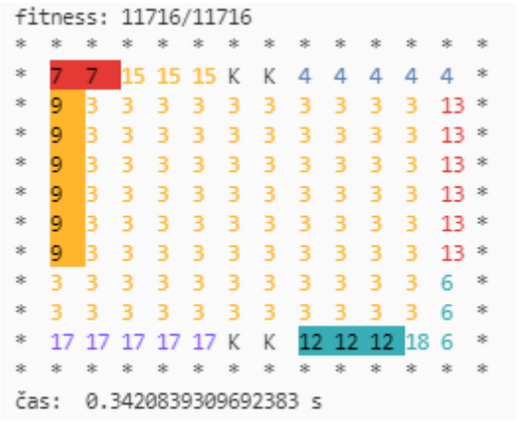

*Obrázok 36 príklad výsledku s fitness 13861 z 13871*

Ako je možné vidieť v tabuľke 5. tak veľkosť tabuľky väčšia ako 100 nemala žiadny vplyv na vývin fitness (zobrazené v percentách), dokonca mala rovnaké výsledky naprieč všetkými hľadaniami s počiatočným vektorom. Teda môžeme predpokladať že najlepšia veľkosť tabu listu pre naše zvolené generovanie okolia je rádovo v desiatkach záznamov. Je možné, že táto veľkosť zodpovedá iba nášmu testovanému generovaniu okolia a pre inú kombináciu by sme ju mali zväčšiť.

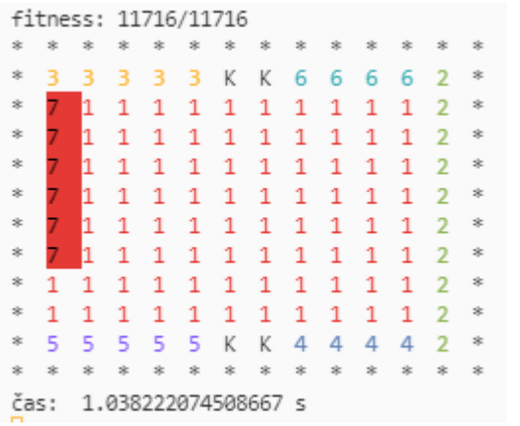
*Tabuľka 5 vývoj fitness v percentách pre tabu list veľkosti 100 a viac*

| | Veľkosť tabu listu | | |
|---|---|---|---|
| vektor: | 100 | 200 | 300 |
| 1 | 91,63 | 91,63 | 91,63 |
| 2 | 99,91 | 99,91 | 99,91 |
| 3 | 99,75 | 99,75 | 99,75 |
| 4 | 99,96 | 99,96 | 99,96 |
| 5 | 99,83 | 99,83 | 99,83 |
| 6 | 99,93 | 99,93 | 99,93 |
| 7 | 99,84 | 99,84 | 99,84 |
| 8 | 91,67 | 91,67 | 91,67 |
| 9 | 99,93 | 99,93 | 99,93 |
| 10 | 99,95 | 99,95 | 99,95 |

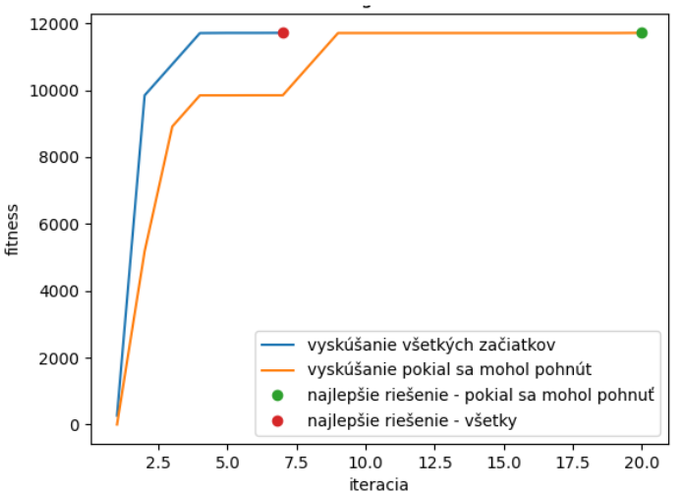
Na koniec zobrazujeme nájdené riešenia pre mapu s listami zo zadania (Obrázok 37. až 39.). Na vygenerovanie sme použili rovnaké nastavenia ako pri testovaní generovania nasledovníkov a generovanie podľa prvej metódy. Obidva spôsoby vypĺňania mapy nemali problém s nájdením riešenia. Preskakovanie ho našlo rýchlejšie už v 7. iterácií, ale rovnako ho našlo aj nepreskakovanie aj keď pomalšie v 20. iteranceí.




*Obrázok 37 preskakovanie povolené*

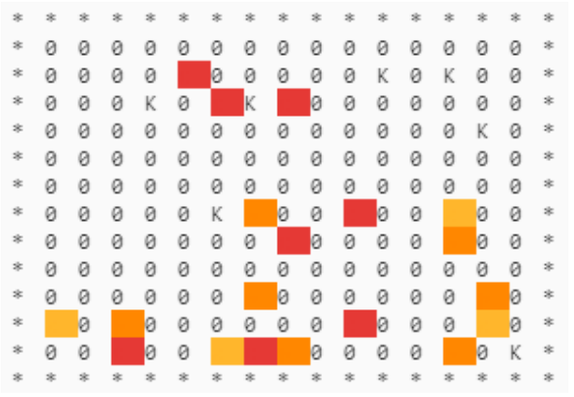



*Obrázok 38 preskakovanie zakázané*




*Obrázok 39 vývoj fitness*

Komplikovanejší problém, parametre a najlepšie riešenie:



```
iteracia c.5
podobnost pociatocneho stavu s predchadzajucimi: 0.000%
pociatocny stav:  [16, 29, 52, 1, 18, 12, 39, 27, 4, 5, 6,
počet všetkých iteracií:  988
najlepšie riešenie v iteracií:  689
najlepsi vektor:  [45, 40, 38, 27, 32, 31, 24, 22, 20, 34,
fitness: 17456/17473
cas :  78.72247648239136 s
```

# 4. Záver

Z nášho pohľadu je naše riešenie úspešné. Používateľovi ponúka mnoho parametrov na nastavenia prehľadávania pomocou tabu search. Celkovo by sme na tento problém odporúčali iný algoritmus, napr. genetický algoritmus, ktorý by pravdepodobne lepšie preskúmal okolie, nakoľko si „nevyberá“ iba jedného nasledovníka, ale má ich niekoľko a teda môže lepšie prehľadať všetky možné počiatočné pozície a vyskladať z nich lepšie riešenia. Na druhú stranu naše riešenia za použitia všetkých nami definovaných a testovaných metód generácie okolia dokázali nájsť semi-optimálne až kompletné riešenia. Veľkosť tabu listu sme určili, že by mala byť rádovo v desiatkach maximálne iba 100 prvkov, na ktorých sme testovali my. Úspešne sme vytvorili riešenie, ktoré nepoužívalo len krátkodobú pamäť, ale aj riešenie s dlhodobou frekvenčnou pamäťou na diverzifikáciu riešení.

Možné zlepšenia nášho projektu by bolo otestovanie ostatných kombinácií generovania nasledovníkov ako aj zistenie najlepších veľkostí tabu listu pre ostatné metódy generovania. Ďalej vyladenie pravdepodobnosti a používanie výberu horšieho nasledovníka v prípade, že ako pri výmenách susedov nastane, že sa „zasekne“ na rovnakom vyplnení máp. Túto funkcionalitu naša implementácia podporuje, ale pri testovaní ho nevyužívame.