Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Umelá inteligencia - Zadanie č. 3

Strojové učenie sa – evolučný algoritmus a evolučné programovanie

Jakub Hajdu

Cvičiaci: Ing. Ivan Kapustík

akad. rok 2020/21

Jakub Hajdu
ID: 105340

Zadanie úlohy

Majme hľadača pokladov, ktorý sa pohybuje vo svete definovanom dvojrozmernou mriežkou (viď. obrázok) a zbiera poklady, ktoré nájde po ceste. Začína na políčku označenom písmenom S a môže sa pohybovať štyrmi rôznymi smermi: hore **H**, dole **D**, doprava **P** a doľava **L**. K dispozícii má konečný počet krokov. Jeho úlohou je nazbierať čo najviac pokladov. Za nájdenie pokladu sa považuje len pozícia, pri ktorej je hľadač aj poklad na tom istom políčku. Susedné políčka sa neberú do úvahy.

Horeuvedenú úlohu riešte prostredníctvom evolučného programovania nad virtuálnym strojom. Tento špecifický spôsob evolučného programovania využíva spoločnú pamäť pre údaje a inštrukcie. Pamäť je na začiatku vynulovaná a naplnená od prvej bunky inštrukciami. Za programom alebo od určeného miesta sú uložené inicializačné údaje (ak sú nejaké potrebné). Po inicializácii sa začne vykonávať program od prvej pamäťovej bunky. (Prvou je samozrejme bunka s adresou 000000.) Inštrukcie modifikujú pamäťové bunky, môžu realizovať vetvenie, programové skoky, čítať nejaké údaje zo vstupu a prípadne aj zapisovať na výstup. Program sa končí inštrukciou na zastavenie, po stanovenom počte krokov, pri chybnej inštrukcii, po úplnom alebo nesprávnom výstupe. Kvalita programu sa ohodnotí na základe vyprodukovaného výstupu alebo, keď program nezapisuje na výstup, podľa výsledného stavu určených pamäťových buniek.

Jakub Hajdu
ID: 105340

Reprezentácia údajov

Na ukladanie jednotlivcov populácie sú využívané objekty triedy **Element**, ktoré obsahujú zoznam svojich pamäťových buniek, cestu, ktorú prešiel v prehľadávanej mape, set súradníc nájdených pokladov a hodnotu fitness. Na uloženie počiatočných súradníc pre hľadača pokladov a tiež veľkosti prehľadávanej mapy, ako aj všetkých súradníc pokladov v mape je využitý objekt triedy **Map**. Populáciu, čiže aktuálnu generáciu, predstavuje zoznam **population** objektov triedy **Element**, ktoré sú s každou novou generáciou aktualizované.

Gény

Gény, s ktorými program pracuje, predstavujú samotné pamäťové bunky jedincov, ktoré spracováva virtuálny stroj. Každý jedinec má týchto buniek podľa zadania 64, ich veľkosť je 1B. Výsledný gén je tak reprezentovaný bunkou o veľkosti 1B, ktorá môže nadobúdať hodnoty 0 až 255. Najvyššie dva bity obsahujú inštrukciu pre virtuálny stroj a zvyšných 6 bitov slúži ako adresa niektorej z buniek a zároveň ako hodnota aktuálnej bunky.

Použitý algoritmus

Pri úvode behu programu sa vstup načítava funkciou **load_map()** z externého textového súboru do objektu triedy **Map**, vytvorí sa takisto na začiatku prázdny zoznam jedincov.

Populácia sa inicializuje vo funkcii **init_population()**. Argumentom tejto funkcie je počet buniek od začiatku pamäte jedinca, ktoré sa majú naplniť hodnotami. V cykle sa do zoznamu populácie priradí potrebný počet jedincov inicializovaných cez konštruktor svojej triedy, následne sa pre každého generuje príslušný počet pamäťových buniek, ktoré sa podľa nastavenia parametra funkcie buď nastavujú na nulu, alebo je do nich zapisovaná náhodná hodnota celého čísla veľkosti 1B, čiže z rozsahu <0, 255>. Na ukladanie bunky o veľkosti 1B je použitý dátový typ uint8 z knižnice **numpy**.

Po inicializácii a zadaní potrebných parametrov (spôsob výberu rodiča pri krížení, maximálna prípustná generácia) sa v cykle postupne tvoria a analyzujú jednotlivé generácie. Ak sa nájde jedinec so všetkými pokladmi, cyklus skončí a tento jedinec sa vypíše. Ak nie, pokračuje až kým nedosiahne maximálnu stanovenú generáciu. Na spracovanie jednej generácie slúži funkcia **analyze_population()**, ktorá potrebuje ako argument prehľadávanú načítanú mapu a metódu výberu rodiča pre kríženie. V tejto funkcii sa každému jedincovi generácie vypočíta a priradí hodnota fitness v závislosti od počtu nájdených pokladov.

Na hľadanie pokladov slúži zadaním definovaný virtuálny stroj, ktorý je implementovaný vo funkcii **calculate_finess()**, parametrami ktorej je kópie spracovávaného jedinca a prehľadávaná mapa. Cyklus virtuálneho stroja má k dispozícii maximálne 500 krokov, ktoré môže nad kódom jedinca vykonať. Pozná zadané inštrukcie cyklickej inkrementácie, cyklickej dekrementácie, skoku na inú bunku a výpisu pohybu v mape, pričom vypísaný smer pohybu je závislý priamo od hodnoty na posledných šiestich bitoch pamäťovej bunky: hodnota 0 až 15 vykonáva pohyb hore, 16 až 31 znamená pohyb dole, 32 až 47 pohyb vpravo a hodnota 48 až 63 pohyb vľavo. Ak by sa zmenil predvolený počet buniek jedinca 64 na iný počet, rozdelenie bude naďalej funkčné, nakoľko sú dané intervaly (štvrtiny) vypočítavané z premennej. Pri inkrementácii a dekrementácii sa overuje a opravuje pretečenie hodnoty (z 255 + 1 bude 0 a naopak z 0 – 1 bude 255). Podobne sa kontroluje aj index bunky aktuálne vykonávanej inštrukcie (ak by mal presiahnuť index poslednej bunky, vráti sa naspäť na nultú). Pri

pohyboch v inštrukcii výpisu sa vždy overuje, či hľadač pokladov nevyšiel za okraj mapy. Vždy, keď sa hľadač v mape pohne, pridá sa posledný pohyb do reťazca vykonaných pohybov a zmení sa aktuálne súradnica hľadača v mape. Tiež sa overí funkciou **check_field()**, či sa nenašiel nový poklad (aktuálne súradnice sa nachádzajú v sete pokladov mapy, ale ešte sa nenachádzajú v sete nájdených pokladov pri spracovávanom jedincovi). Ak sa našiel nový poklad, pridáva sa do setu nájdených pre daného jedinca a zvyšuje sa hodnota fitness. Ak je počas behu virtuálneho stroja dosiahnutý stav, kedy v sete nájdených pokladov sú všetky existujúce v mape, stroj sa ukončí. Takisto sa ukončí, ak vykonal stanovený, maximálny počet inštrukcií alebo ak hľadač vyšiel za okraj mapy. Vrátený objekt jedinca obsahuje fitness (počet nájdených), nájdené poklady a reťazec s cestou, ktorú prešiel v mape.

Späť vo funkcii **analyze_population()** sa v prípade výberu ruletou sumuje celkový súčet fitness celej generácie. Zároveň sa jedinci porovnávajú s doteraz celkovo najlepším jedincom podľa fitness, aby sa zapamätal najlepší jedinec na výpis aj v prípade, že by sa nenašli všetky poklady. Ak sa v spracovávanej generácii nájde jedinec, ktorý našiel všetky poklady, funkcia sa ukončí, aby mohol byť vypísaný výsledok.

Následne dochádza v cykle ohraničenom počtom jedincov jednej generácie k tvorbe novej generácie. S určitou pravdepodobnosťou dochádza k prežitiu celého zoznamu buniek jedinca a tieto hodnoty sa prenesú priamo do novej generácie. Predchádza sa tak strate celých pôvodných génových sekvencií. Ak tento prípad prežitia nenastane, vyberú sa jednou z metód dvaja jedinci (rodičia), aby sa z nich krížením vytvoril nový jedinec.

Prvým možným spôsobom výberu je ruleta, ktorá vyberie rodiča funkciou **roulette()**. V tejto funkcii sa vygeneruje náhodná hodnota v rozsahu <0, súčet_fitness_generácie). V cykle sa potom postupne sumujú hodnoty fitness jednotlivých jedincov z generácie. Keď bola presiahnutá vyššie spomenutá náhodná hodnota, vráti sa posledný pripočítaný jedinec. Týmto sa zabezpečí výber s vhodnou distribúciou pravdepodobnosti. Ak by došlo k prípadu, že by celkový súčet fitness bol 0, vyberie sa náhodný jedinec (inak by sme generovali náhodné číslo z neplatného rozsahu).

Druhým spôsobom výberu je turnaj, ktorý má na starosti funkcia **tournament()**. V nej najskôr zo súčasnej generácie vyberieme určitý počet náhodných jedincov (predvolene vo funkcii nastavené na troch), potom sa z týchto vyberie práve jeden s najvyššou hodnotou fitness. Týmto sa aj napriek jednoduchosti princípu výberu zabezpečuje dostatočná šanca na výber jedinca zo slabšej časti populácie, aby sa zachovalo viac génov a predišlo sa priveľkému elitárstvu.

Po výbere dvoch rodičov sa pristupuje k samotnému kríženiu, ktoré je riešené tak, že sa obsah pamäťových buniek kopíruje od oboch rodičov, pričom sa vždy zvolí náhodný stredný bod. Po tento bod (index pamäťovej bunky) sa skopíruje obsah prvého rodiča, ďalej od daného bodu po poslednú bunku sa kopíruje obsah druhého rodiča. Pri určitej pravdepodobnosti však dochádza k mutácii.

Mutácia je v našom prípade stanovená tak, že sa namiesto kopírovania aktuálnej pamäťovej bunky od rodiča vytvorí nová naplnená náhodným číslom ako pri inicializácii, čiže v rozsahu <0, 255> .

Po vytvorení všetkých nových jedincov sa tieto stanú novou generáciou na spracovanie. Výsledným výpisom v hlavnom cykle programu je buď jedinec, ktorý našiel všetky poklady, alebo ten, ktorý ich našiel doposiaľ najviac. Vypisuje sa jeho cesta v mape, počet nájdených pokladov a obsah pamäťových buniek v dekadickom aj binárnom tvare (zarovnanom na 8 bitov). Tento výpis má na starosti funkcia **print_element()**.

Jakub Hajdu
ID: 105340

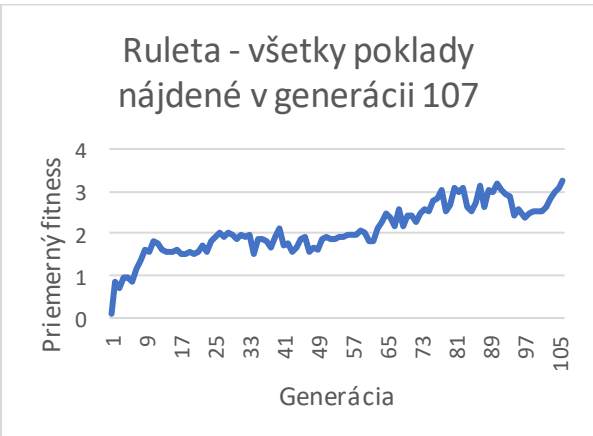
Testovanie

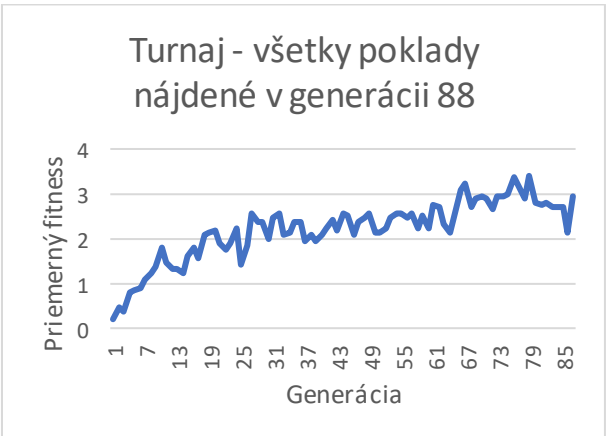
Na testovanie bola v úvodných fázach písania programu použitá vzorová mapa zo zadania (**vzor.txt**), po doladení funkcionality mutácie a hodnôt pravdepodobností bolo testovanie uskutočnené na vstupoch načítavaných z viacerých vlastných vytvorených textových súborov, v ktorých sa nachádza informácia o veľkosti mapy, nasledovaných súradnicami počiatočnej pozície hľadača v mape a súradnicami všetkých pokladov v nej. Pre výstup (výsledok evolučného algoritmu) bol veľakrát najmä na začiatku ručne kontrolovaný každý krok nájdeného riešenia, aby sa overilo správne presúvanie hľadača v mape a hľadanie pokladov.

Veľká časť kódu, ktorý bol použitý na testovanie, bola zachovaná v zakomentovanej časti **TESTER**. Okrem štandardných výpisov po skončení celého behu evolučného algoritmu bolo v tejto časti využité zapisovanie do výstupného súboru, kam sa zapisovali buď hodnota fitness a cesta v mape všetkých jedincov každej generácie, alebo sa pre každú generáciu zapísala priemerná hodnota fitness na jedného jedinca generácie. Prvý spomenutý prípad bol užitočný na sledovanie, či je správne nastavená pravdepodobnosť mutácie, t. j. či sa jedinci na seba príliš nepodobajú a nedochádza k uviaznutiu v lokálnych optimách. V prípade výpisu priemerných hodnôt fitness pre každú generáciu sa dal sledovať celkový priebeh hľadania optimálnych riešení a najmä rýchlosť, akou stúpa priemerná hodnota fitness.

Hľadanie bolo na vzorovej mape spúšťané v stovkách behov za účelom doladenia predvolených hodnôt pravdepodobností. Sledované bolo pritom percento prípadov, kedy boli do rovnakej maximálnej generácie nájdené všetky poklady. Ako táto hranica bol zvolený počet 1000 generácií. Odskúšané boli rôzne počty jedincov v generácii (20 - 50), pravdepodobnosti prežitia jedincov (2% - 20%) a pravdepodobnosti mutácie (0,1% – 10%). Do 1000 generácií bola celkovo najviac prijateľná verzia s počtom jedincov 20, pravdepodobnosťou prežitia jedinca 10% a pravdepodobnosťou mutácie 2%, pričom pri týchto nastaveniach našiel program s použitím rulety všetky poklady približne v 65% behov, jeden poklad mu chýbal približne v 30% prípadoch a ostatné možnosti nastali v zvyšných približne 5%. Počas testovania s týmito parametrami nebol zaznamenaný horší výsledok ako tri nájdené poklady z piatich. V prípade turnaja percento behov programu so všetkými nájdenými pokladmi dosahovalo bežne 90%, pričom zo zvyšných 10% behov boli takmer všetky také, že hľadačovi chýbal iba jeden poklad.

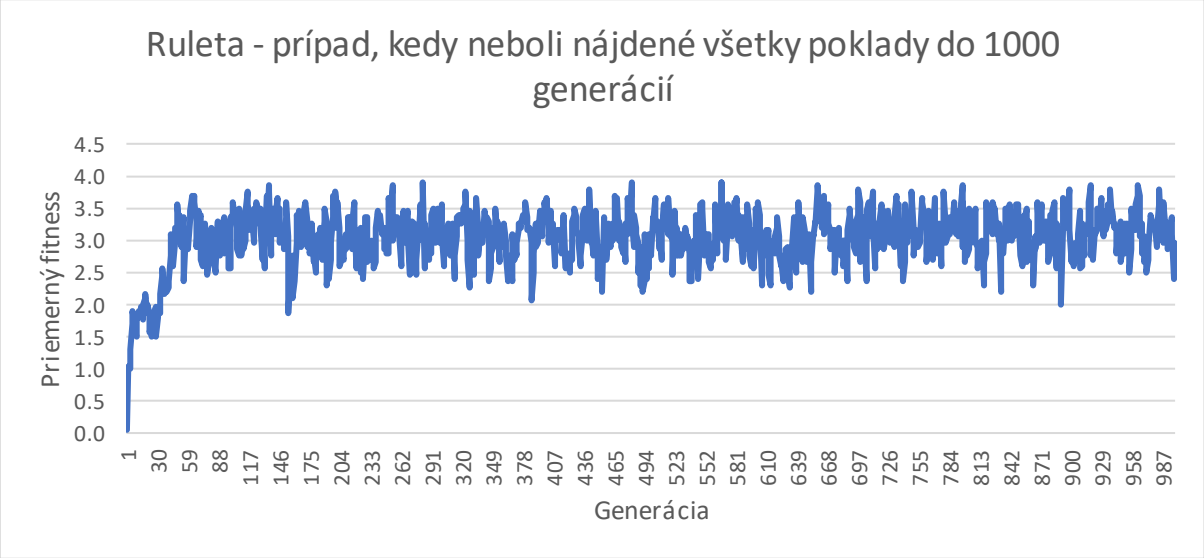
Dosiahnuté výsledky môžeme považovať za akceptovateľné, vzhľadom na charakter evolučného algoritmu a berúc do úvahy polohu piateho pokladu. Tento poklad je menej pravdepodobné nájsť, nakoľko sa nachádza pri okraji mapy. V niektorých prípadoch boli všetky poklady vďaka dobrým počiatočným obsahom buniek jedincov nájdené pomerne skoro. V grafe vidíme vývoj priemernej hodnoty fitness pre jednotlivé generácie pri použití rulety a turnaja v takýchto prípadoch:



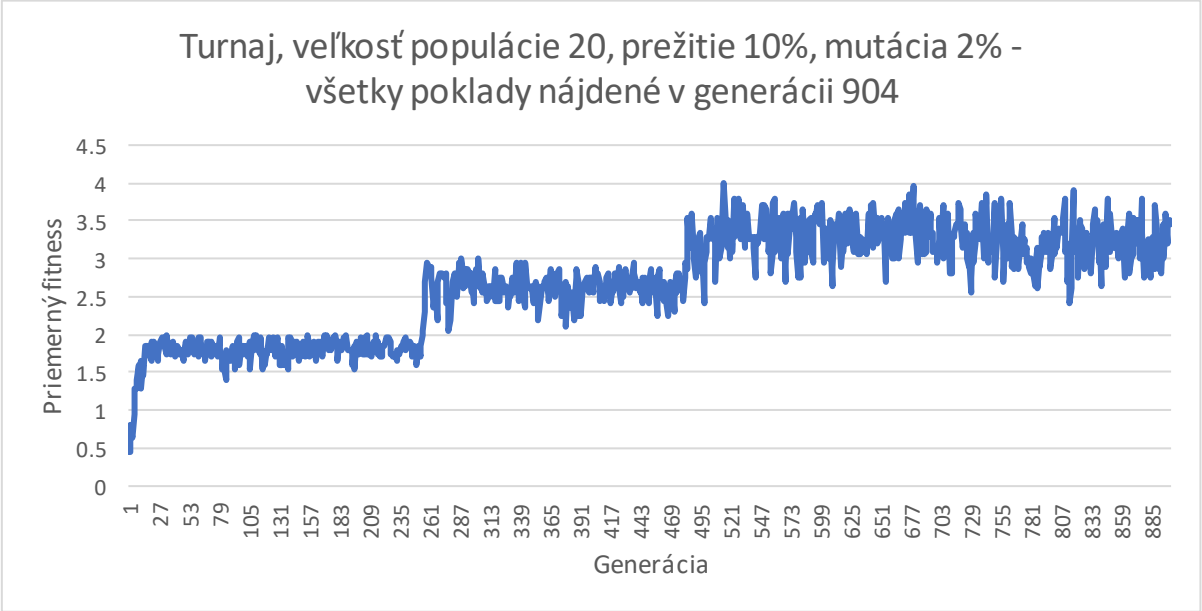


Jakub Hajdu
ID: 105340

Príklad prípadu, kedy sa všetky poklady zo vzorovej mapy nepodarilo nájsť do limitu 1000 generácií je možné vidieť v nasledovnom grafe, zobrazujúcom priemernú hodnotu fitness v závislosti od generácie:



V každom grafe môžeme pozorovať relatívne prudký počiatočný nárast priemernej hodnoty fitness, čo značí, že algoritmus správne konverguje k lepším riešeniam. Takisto je vidno, ako môže často pomôcť mutácia jedincov v prípade, že by malo dôjsť k uviaznutiu v lokálnom optime. Takýto priebeh je výrazne viditeľný v nasledujúcom grafe:



Porovnávaním percentuálneho podielu prípadov, kedy sa našli všetky poklady do 1000 generácií bolo zistené, že v našom konkrétnom prípade fungoval o niečo lepšie zo spôsob výberu rodičov na kríženie turnaj. Parametre pre evolučný algoritmus boli síce do určitej miery testovaním optimalizované, no ruleta bola na tieto parametre vždy o niečo citlivejšia, zatiaľ čo metóda turnaja fungovala zakaždým spoľahlivejšie.

Viac grafov priemerného fitnessu pre parametre navolené po testovaní (20 jedincov, pravdepodobnosť prežitia jedinca 10% a pravdepodobnosť mutácie 2%) je priložených v súbore **grafy.pdf**.

Jakub Hajdu
ID: 105340

Opis používateľského rozhrania

Prostredím používateľského rozhrania je konzola, aplikácia neobsahuje grafické rozhranie. Po spustení sa zobrazí úvodná možnosť pre načítanie súboru alebo ukončenie programu.

```
Nacitat subor: y
Ukoncit program: n
```

Po zadaní „n“ program skončí, zadaním možnosti „y“ sa zobrazí výzva na meno požadovaného súboru.

```
y
Zadajte nazov suboru s mapou (bez pripony)
```

Ak bol zadaný neplatný názov súboru, program na to upozorní:

```
neexistujuci_subor
Zadany subor sa nenasiel
```

Inak pokračuje na voľbu spôsobu výberu rodičov pri krížení.

```
vzor
Zadajte sposob vyberu rodicov pre nove generacie
Ruleta = r
Turnaj = t
```

Následne je potrebné zadať maximálny počet generácií, ktoré sa majú tvoriť a prehľadávať.

```
Zadajte limit poctu generacii pre prehladavanie
```

Pokiaľ sa nepodarí nájsť všetky poklady do stanovenej maximálnej generácie, vypíše sa najlepší doposiaľ dosiahnutý výsledok.

```
500
Do maximalenj generacie neboli najdene vsetky poklady, vypisuje sa najlepsi vysledok
najdene poklady: 4, cesta: L, U, L, U, U, R, L, L, R, R, R, L, L, R, R, R, L, L, R, R, R, L, L, R, R, R, L, L, R
bin: [11111101, 11001111, 01010000, 11111111, 11000011, 11000010, 00011110, 01011011, 10000100, 11011010, 100110
dec: [253, 207, 80, 255, 195, 194, 30, 91, 132, 218, 152, 166, 240, 7, 76, 49, 37, 39, 153, 159, 33, 109, 49, 12
```

Ak sa našli všetky poklady, vypíše sa jedinec, ktorý ich našiel.

```
500
Vsetky poklady najdene v generacii cislo  159
najdene poklady: 5, cesta: U, R, R, U, U, R, L, L, D, L, D, L, U, R, R, U, U, R, L, L, D, L, D, L, U, R, R, U, U, R, L, L, D
bin: [01110001, 01110011, 10011110, 11010000, 00100011, 11111000, 01111110, 00001011, 00110110, 11011010, 10111110, 10010001,
dec: [113, 115, 158, 208, 35, 248, 126, 11, 54, 218, 190, 145, 219, 12, 1, 0, 11, 61, 83, 94, 39, 221, 6, 17, 250, 118, 147,
```

Zakaždým sa aplikácia vracia do úvodného menu, je teda možné spracovávať mapy viackrát po sebe pri jednom spustení aplikácie. Vstupné súbory sa nachádzajú v priečinku **inputs**.

Jakub Hajdu
ID: 105340

Zhodnotenie

Program je schopný správne sa vo virtuálnom stroji pohybovať v zadanej mape, správne rozpoznáva poklady a evolučný algoritmus vďaka funkčným metódam kríženia a mutácie zakaždým konverguje smerom k riešeniam s viac nájdenými pokladmi.

Celková úspešnosť, t. j. prípady so všetkými nájdenými pokladmi, je vzhľadom na implementované metódy výberu rodičov na kríženie postačujúca na nájdenie všetkých pokladov vo vzorovej mape pre približne 65% spustení v prípade rulety a 90% spustení v prípade turnaja, ak hľadáme maximálne do 1000 generácií.

Možným zlepšením je lepšie rozvrhnutie hodnôt fitness, napríklad použitie exponenciálneho ohodnotenia v závislosti od počtu pokladov. Takisto by optimalizovať výsledok pomohlo započítavanie zlomkových záporných hodnôt za každý pohyb hľadača v mape, čo by zabezpečilo uprednostnenie kratších ciest so zachovaním pravdepodobnostnej distribúcie primárne podľa počtu nájdených pokladov. Kód je na toto vylepšenie dobre pripravený, keďže už je v ňom oddelený zoznam nájdených pokladov od samotnej hodnoty fitness, ktorá by teda mohla byť odlišná od presného počtu pokladov (dĺžky zoznamu nájdených pokladov). Rovnako by stálo za vyskúšanie implementovať niektoré ďalšie metódy výberu rodičov na kríženie, poprípade doplniť iné druhy mutácií.