Slovenská technická univerzita

Fakulta informatiky a informačných technológii

Ilkovičova 3, 812 19 Bratislava

Umelá inteligencia

Zadanie č. 3 b

Peter Berta

Cvičiaci: Ing. Ivan Kapustík

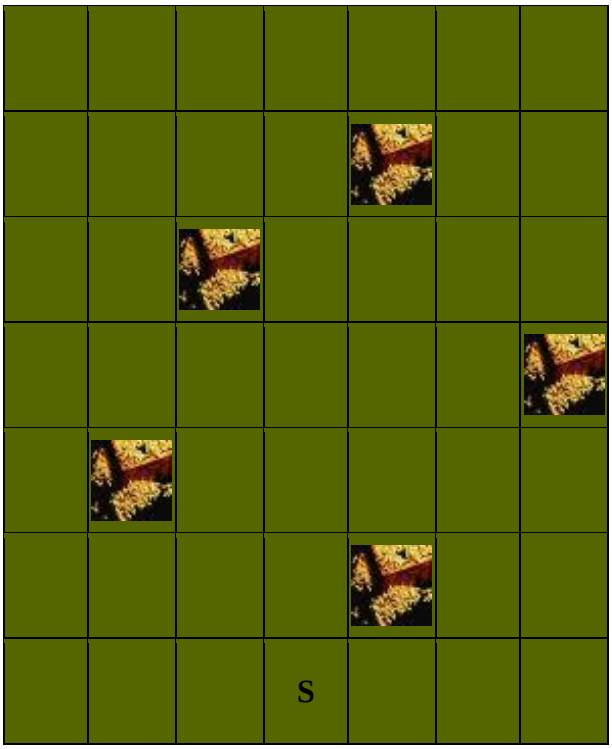
Študijný odbor: Informatika

Ročník: 2. Bc

Akademický rok: 2015/2016

1. Riešený problém

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, “dolu” 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.

2. Virtuálny stroj

Náš stroj bude mať 64 pamäťových buniek o veľkosti 1 byte.

Bude poznať štyri inštrukcie: inkrementáciu hodnoty pamäťovej bunky, dekrementáciu hodnoty pamäťovej bunky, skok na adresu a výpis (H, D, P alebo L) podľa hodnoty pamäťovej bunky.

Program sa zastaví, akonáhle bude splnená niektorá z nasledovných podmienok:

- program našiel všetky poklady
- postupnosť, generovaná programom, vybočila zo stanovenej mriežky
- program vykonal 500 krokov (inštrukcií)

Výber pohybu

V prípade, že virtuálny stroj narazí na inštrukciu výpis, tak daný smer pohybu je získavaný z posledných dvoch bitov bunky:

```
move = cells[act] & 3;
```

3. Reprezentácia údajov problému a použitý algoritmus

Na implementáciu tohto zadania bol použitý programovací jazyk C.

Evolučný algoritmus

Podstatou evolučného algoritmu je vygenerovanie prvej generácie (populácie) a následný výber jedincov do ďalších generácií s cieľom nájsť čo najoptimálnejšie riešenie daného problému.

Spôsob výberu jedincov do ďalšej generácie môže byť rôzny: elitárstvo, mutácie, ruleta, turnaj, náhodná generácia.

Ku generovaniu ďalšej populácie a výberu jedincov však dôjde len v prípade, že sme ešte nenašli správne riešenie. Ak sme našli jedinca, ktorý nájde všetky poklady, algoritmus sa zastaví a vypíše riešenie. V prípade, že by bola požiadavka na nájdenie optimálnejšieho riešenia, je možné jednoducho upraviť podmienku, ktorá cyklus ukončuje.[1][2]

Nový jedinec

Bunky virtuálneho stroja sú predstavované poľom znakov, keďže znak v jazyku C má veľkosť 1 byte, čo nám na reprezentáciu jednej bunky úplne stačí. Celý jedinec je teda reprezentovaný štruktúrou, ktorá v sebe obsahuje pole znakov a hodnotu fitness funkcie daného jedinca:

```
struct dude {
	char *cells;
	double fit;
};
```

Keď je v programe vytvorený nový jedinec, tak jeho prvých 32 buniek je náhodne vygenerovaných, pričom ostatné sú inicializované na hodnotu 0:

```
for (i = 0;i < 64;i++){
		if (i < 32) {
			tempDude->cells[i] = rand() % 256;
		}
		else {
			tempDude->cells[i] = 0;
		}
	}
```

Hneď po inicializácii všetkých jeho buniek je vypočítaná jeho fitness funkcia:

```
tempDude->fit = get_fit(field, tempDude);
```

Fitness funkcia

Fitness funkcia je kvantifikácia vhodnosti použitia daného jedinca pre riešenie zadaného problému. Teda čím je hodnota fitness funkcie vyššia, tým je tento jedinec bližšie k riešeniu.

```
double get_fit(char **field, struct dude *dude){…}
```

Jej hodnotu zistíme spustením virtuálneho stroja a počítaním, koľko pokladov je jedinec schopný nájsť. Pre každý nájdený poklad je hodnota fitness funkcie zvýšená o jedna a pre každý krok, ktorý musí jedinec vykonať, je odpočítaná jedna stotina. Takto zaručíme, že najvyššie hodnoty fitness funkcií budú mať jedinci, ktorí navštívia najviac pokladov za čo najmenej krokov. Počet pokladov má však prednosť.

Kríženie jedincov

Pri krížení jedincov vzniká nový jedinec, ktorý obsahuje 32 buniek z prvého jedinca a 32 buniek z druhého jedinca. Následne je vypočítaná hodnota jeho fitness funkcie.

Nová generácia

Každá generácia je v programe reprezentovaná pomocou poľa jedincov:

```
struct dude **generation = NULL;
```

Prvá generácia obsahuje len náhodne vygenerovaných jedincov:

```
generation = new_gen();

for (i = 0;i < DUDE_NUM;i++) generation[i] = new_dude(pole);
```

Všetky nasledujúce generácie sú už kombináciou predchádzajúcej generácie a nových jedincov. Spôsoby výberu jedincov do nasledujúcej generácie sa líšia.

Elitárstvo

Elitárstvo predstavuje spôsob výberu jedincov do novej generácie, pri ktorom do novej generácie postúpia jedinci s najvyššou hodnotou fitness funkcie. Počet týchto jedincov zväčša predstavuje najviac 10% celkového počtu jedincov v generácii.[1]

Turnaj

Počas turnaju sa z našej generácie vyberie 10 náhodných jedincov, z ktorých bude vybraný ten s najvyššou hodnotou fitness funkcie. Tento jedinec predstavuje jedného rodiča. Následne podobným spôsobom nájdeme ďalšieho rodiča, ktorý sa s predchádzajúcim skríži a výsledný jedinec bude zaradený do novej generácie.[1]

4. Spôsob testovania a výsledky experimentov

Testovanie

Počas implementovania tohto programu som na kontrolu správnosti vykonávania inštrukcií vo virtuálnom stroji používal pomocné funkcie a pomocné výpisy.

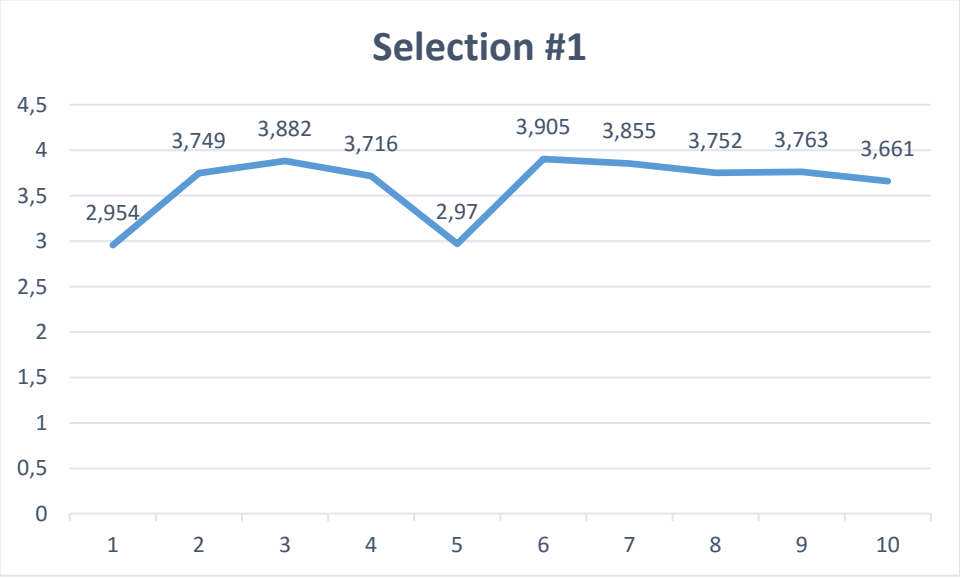
Na testovanie efektivity použitého algoritmu som používal výpis po každom vygenerovaní novej generácie.

Selection #1

V tomto prípade som vyberal jedincov z predchádzajúcej generácie nasledovne:

- 10% - najlepší jedinci
- 10% - zmutovaní najlepší jedinci
- 30% - kombinácia prvých 60 jedincov
- 50% - náhodne generovaní jedinci

Po 1000 generáciách, pričom každá generácia je tvorená 100 jedincami bola priemerná najlepšia hodnota fitness funkcie z 10 pokusov 3,6207. Z toho vyplýva, že sme ani raz nedosiahli správne riešenie, našli sme len 3 – 4 poklady.



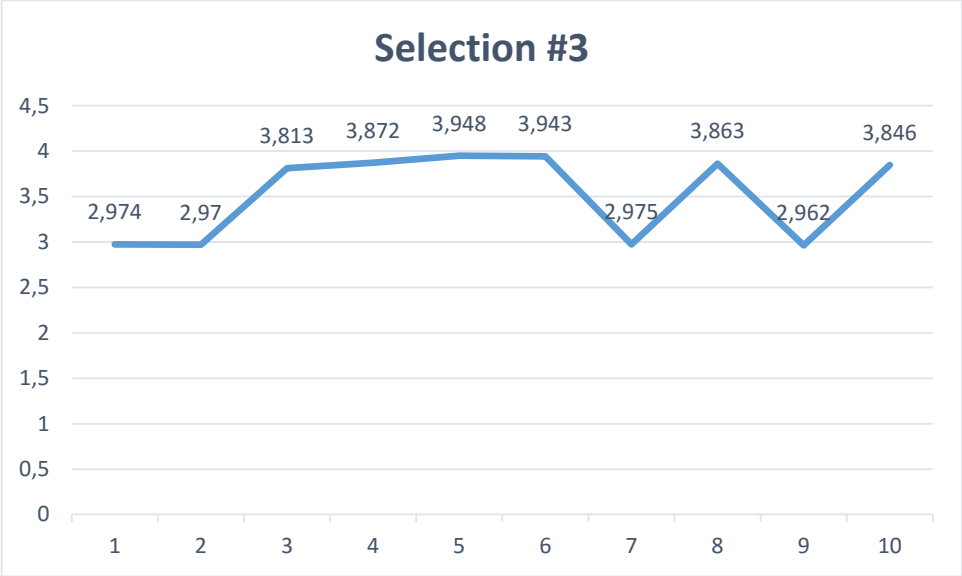
Selection #2

V tomto prípade som vyberal jedincov z predchádzajúcej generácie nasledovne:

- 10% - najlepší jedinci
- 20% - zmutovaní náhodní jedinci

- 20% - kombinácia výberu pomocou turnaju (jedinci sú kombinovaní tak, že prvá polovica pochádza z prvého jedinca a druhá polovica je z druhého jedinca)
- 50% - náhodne generovaní jedinci

Po 1000 generáciách, pričom každá generácia je tvorená 100 jedincami bola priemerná najlepšia hodnota fitness funkcie z 10 pokusov 3,8305. Z toho vyplýva, že sme ani raz nedosiahli správne riešenie, našli sme len 3 – 4 poklady.



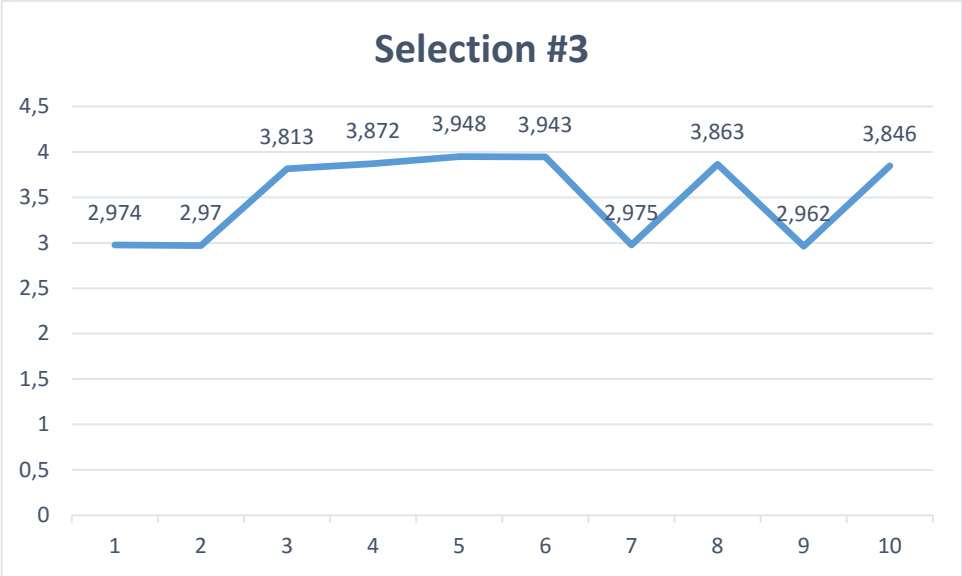
Selection #3

V tomto prípade som vyberal jedincov z predchádzajúcej generácie nasledovne:

- 10% - najlepší jedinci
- 20% - zmutovaní náhodní jedinci
- 20% - kombinácia výberu pomocou turnaju (jedinci sú kombinovaní tak, že každá druhá bunka pochádza z iného jedinca)
- 50% - náhodne generovaní jedinci

Po 1000 generáciách, pričom každá generácia je tvorená 100 jedincami bola priemerná najlepšia hodnota fitness funkcie z 10 pokusov 3,41. Z toho vyplýva, že sme ani raz nedosiahli správne riešenie, našli sme len 3 – 4 poklady.

Rád by som však podotkol, že pomocou tohto spôsobu výberu bol program schopný nájsť riešenie v generácii číslo 9382.



5. Zhodnotenie riešenia

Úspešnosť nájdenia riešenia evolučného algoritmu je z veľkej časti založená na náhode, keďže každou generáciou sú generovaní nový jedinci, ktorí môžu byť vygenerovaní hneď na začiatku veľmi dobre, alebo aj veľmi zle. Jedna vec, ktorú však môžeme ovplyvniť je spôsob selekcie jedincov do novej generácie.

Teoreticky najlepšie riešenie (Selection #3) sa preukázalo ako štatisticky najhoršie. Štatisticky najlepšie sa preukázalo riešenie číslo 2 (Selection #2).

V riešení nebol implementovaný výber pomocou rulety, ktorý by teoreticky dosahoval ešte lepšie výsledky, ako vyššie uvedené riešenia.

6. Používateľská príručka

Vstupný súbor

Pre inicializáciu poľa s pokladmi sa používa súbor, v ktorom sú uložené informácie o počte políčok poľa, súradnice počiatočného bodu, počet pokladov a na koniec súradnice každého z pokladov.

Počet jedincov v generácii a počet generácií

Počet jedincov v každej generácii sa dá nastaviť na začiatku zmenou hodnoty konštanty DUDE_NUM:

```
#define DUDE_NUM    200
```

Počet generácií, po ktorých keď nebude nájdené správne riešenie sa program pozastaví, sa dá podobne nastaviť na začiatku zmenou hodnoty konštanty GEN_NUM:

```
#define GEN_NUM    1000
```

Po pozastavení programu z dôvodu prehľadania určitého počtu generácií je možné pokračovať v prehľadávaní stlačením tlačidla ENTER.

Po každom vygenerovaní generácie sa vypíše určitý počet najlepších jedincov. Tento počet sa dá na začiatku nastaviť zmenou hodnoty konštanty PRINT_NUM:

```
#define PRINT_NUM   10
```

Zdroje:

[1] Ako GA pracujú? http://neuron-ai.tuke.sk/arvayova/bk/algoritmus.html

[2] Evolučné programovanie. http://alife.tuke.sk/kapitola/650/index.html