Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Zadanie č. 3

Evolučné algoritmy

Ondrej Harnúšek
ID: 79545

Študijný program: Informatika
Ročník: 2
Krúžok: Po 16.00-17.50 1.30a(LSS1)
Predmet: Umelá inteligencia
Ak. rok: 2016/2017

1 Zadanie - Zenová záhrada

Zenová záhradka je mriežka obsahujúca políčka piesku a políčka kameňov. Našou úlohou je postupne „v pásoch“ prejsť všetkými políčkami piesku.

Každý pás začína a končí na okraji mriežky. Pásy môžu ísť len vodorovne alebo zvislo. Ak narazíme na kameň alebo už pohrabaný piesok je potrebné zmeniť smer. Ak je voľný len jeden smer, otočí sa tam. Ak sú voľné oba, môže sa rozhodnúť.

Ak sa nemá kam otočiť, riešenie je neúspešné. Úspešným riešením je pokrytie všetkých políčok piesku pásmi.

2 Použitý algoritmus

Zadanie je riešené pomocou genetického algoritmu implementovaného v jazyku Java. Hlavná časť algoritmu sa nachádza v triede `Evolution`.

Na začiatku sa vygeneruje prvá populácia jedincov – riešení nášho zadania s náhodne nastavenými génmi. Následne sa začne hľadanie riešenia, ktoré je úspešné.

```
public static int map[][];      //mapa zahradky
public Garden population[];     //pole jedincov v populacii

public Evolution(int width, int height, int[][] stones)
      createMap();//vytvori mapu zahradky
      generateFirstGeneration();//vytvori prvu generaciu nahodnych jedincov
      Garden solution = findSolution();
```

Hľadanie riešenia prebieha vo viacerých generáciách a v každej generácii sa celá populácia najskôr ohodnotí, pričom sa overuje či sa nenašlo úspešné riešenie, a potom sa zo starej populácie vytvorí nová populácia.

```
private Garden findSolution()
      for(int g=0; g<maxGenerations; g++) {
            for(Garden garden : population) {
                  fitnessSum+= garden.fitness();//ohodnotenie jedincov
                  if(garden.isFinal())
                        return garden;
            }
            runReproduction(fitnessSum); //vytvorene novej populacie jedincov
      return null;
```

2.1 Kódovanie génov

Jedinca predstavuje trieda `Garden`. Jedinec má gény dvoch typov:

```
public class Garden {
      public Position entryGenes[]; //geny reprezentujuce poziciu na okraji mapy
      public boolean decisionGenes[];//geny reprezentujuce rozdodnutie odbocenia
```

Trieda génu reprezentujúceho pozíciu na okraji mapy:

```
public class Position {
      public int x, y; //suradnice vstupu na mapu
      public static final int UP = 0, DOWN = 1, RIGHT = 2, LEFT = 3;
      public int direction; //smer akym sa zacne pohybovat
```

2.2 Vyhodnocovacia funkcia

Na vyhodnotenie správnosti riešenia je v prvom rade potrebné prejsť jeho mapu záhradky. Postupne vyberáme gény jedinca, ktoré reprezentujú miesto kde vstupuje na mapu a vytvárame pásy.

V metóde `travel()` prechádzame políčkami piesku v jednom smere a ak v ňom nie je možné pokračovať zmeníme smer. Ak je na výber z dvoch smerov rozhodneme podľa génu reprezentujúceho rozhodnutie odbočenia.

Každým skokom na políčko piesku sa hodnota fitness zvýši o 1 bod. Úspešné riešenie má hodnotu fitness rovnú počtu políčok piesku a zároveň sa dokončili všetky pásy (nezaseklo sa).

```
public int fitness()
	//skopirovanie mapy
	map = new int[Evolution.map.length][];
	for(int i = 0; i < Evolution.map.length; i++) {
		map[i] = Evolution.map[i].clone();
	}
	//vyber genov vstupu
	for(int i=0; i<getEntriesCnt(); i++) {
		travel(entryGenes[i]); //vytvorenie pásu
	}
	return fitnessValue;
```

2.3 Vytvorenie novej populácie

V každej populácii sa nahradí celá populácia novou populáciou. Jedinec novej populácie vzniká skrížením dvoch jedincov vybraných selekčnou metódou a následným zmutovaním.

2.3.1 Selekcia

Na základe vstupnej konfigurácie môže výber jedinca prebiehať dvomi spôsobmi:

1. **Turnajová selekcia.** Výber dvoch náhodných jedincov z populácie a vybratie lepšieho do novej populácie. Zložitosť O(1).

```
private Garden tournamentSelect()
	Random r = new Random();
	int indexG1= r.nextInt(populationSize);
	int indexG2= r.nextInt(populationSize);
	if(population[indexG1].fitnessValue > population[indexG2].fitnessValue)
		return population[indexG1];
	else
		return population[indexG2];
```

2. **Proporcionálna selekcia.** Pravdepodobnosť výberu jedinca sa určí ako podiel jeho fitness ku súčtu fitness celej populácie. Na výber jedincov je použitá ruleta. Zložitosť je O(n).

```
private Garden proportionalSelect(int fitnessSum)
	Random r = new Random();
	int exceedance = r.nextInt(fitnessSum); //nahodne cislo
	int cumulation=0;

	for(int i=0;i<populationSize;i++){ //ruleta
		cumulation+= population[i].fitnessValue;
	   if(cumulation > exceedance) {
		return population[i];
	   }
     }
```

2.3.2 Kríženie

Vstupom pre algoritmus kríženia sú dva jedince vybrané algoritmom selekcie a výstupom je nový jedinec. Jeho gény sú spolovice jedného a spolovice druhého rodiča pričom ktoré to budú sa rozhoduje náhodne. Nakoniec nový jedinec ešte zmutuje.

```
private Garden recombination(Garden male, Garden female)
	for(int i=0; i<count/2; i++) {//polovica genov 1.typu z jedneho
		index = r.get(i);
		newGarden.entryGenes[index] = male.getCopyEntryGene(index);
	}
	for(int i=count/2; i<newGarden.getEntriesCnt(); i++) {//polovica z druheho
		index = r.get(i);
		newGarden.entryGenes[index] = female.getCopyEntryGene(index);
	}
	for(int i=0; i<count/2; i++) {//polovica genov 2.typu z jedneho
		index = r.get(i);
		newGarden.decisionGenes[index] = male.decisionGenes[index];
	}
	for(int i=count/2; i<newGarden.getDecisionsCnt(); i++) {//polovica z druheho
		index = r.get(i);
		newGarden.decisionGenes[index] = female.decisionGenes[index];
	}
	newGarden.mutation(); //nakoniec este zmutuje
	return newGarden;
```

2.3.3 Mutácia

S určitou pravdepodobnosťou sa novému jedincovi nastaví každý gén na náhodné hodnoty. Vstupná konfigurácia určí aká je pravdepodobnosť.

```
public void mutation()
	Random r = new Random();
	for(int i=0; i<getEntriesCnt(); i++) {
		if(Evolution.mutationPercentage>r.nextInt(100)) {
			entryGenes[i].randomisation();
		}
	}
```

3 Zhodnotenie

Pri porovnávaní metód selekcie sa neprejavili rozdiely pri počte potrebných generácii na nájdenie úspešného riešenia, avšak hľadanie s využitím turnajovej selekcie bolo značne rýchlejšie oproti proporcionálnej selekcii.

Veľmi významný je faktor mutácie. Pri testovaní sa ukázalo, že bez nej hľadanie riešenia rýchlo spadne do lokálneho maxima.

Počet generácií potrebných na nájdenia úspešného riešenia závisí prirodzene najme od veľkosti populácie. Príklad výsledkov testovania vstupu zo zadania:

veľkosť populácie:	10000	1000	100	10
počet generácií:	29	348	3234	34271

Ukazuje sa, že na nájdenie úspešného riešenia je potrebných rádovo stotisíc jedincov.

3.1 Konfigurácia algoritmu

V súbore config.txt možno nastaviť parametre, ktoré majú vplyv na efektívnosť algoritmu.

```
#TOURNAMENT_SELECTION=0, PROPORTIONAL_SELECTION=1
selectionMetod= 0
populationSize= 1000
maxGenerations = 1000
mutationPercentage = 20
```

Samotný vstup sa nachádza v súbore input.txt

```
12 10   -rozmery záhradky
6       -počet kameňov
5 1     -súradnice prvého kameňa
1 2     -...
4 3     -...
2 4     -...
8 6     -...
9 6     -súradnice šiesteho kameňa
```

Výstup smeruje do kozoly.

```
 10  8  5  4  11 11 11  4 15 15  7  9
  8  8  5  4  11  K 11  4 15 15  7  9
 14  K  5  4  11 11 11  4 15 15  7  9
 14  5  5  4   K 11 11  4 15 15  7  9
  5  5  K  4  11 11 11  4 15 15  7  9
 12 12 12  4  11 11 11  4 15 15  7  9
 12 12 12  4   4  4  4  4  K  K  7  7
  1  1  1  1   1  1  1  1  1  1  1  1
  2  2  2  2   2  2  2  2  2  2  2 13
  6  3  3  3   3  3  3  3  3  3  2 13
```

Grafy hodnôt fitness počas hľadania pri tejto konfigurácii. Riešenie sa našlo v 160.generácii: