Metajezik za opis evolucijskih algoritama

Marko Đurasević

Sadržaj

- Evolucijski algoritmi
- ECDL
- ECDLC
- Osnove ECDL-a
- Prednosti i nedostaci ECDL-a
- Moguće nadogradnje ECLD-a
- Zaključak

Evolucijski algoritmi

- Zasnovani na prirodnoj evoluciji (selekcija, mutacija, križanje)
- Ne garantiraju pronalaženje najboljeg rješenja, već nekog dovoljno dobrog
- Koriste se pretežito za probleme pretraživanja i optimiziranja

ECDL

- *Evolutionary computation description language*
- Jezik za jednostavan i brz opis evolucijskih algoritama
- Uvođenje veće apstrakcije u razvoj evolucijskih algoritama

ECDLC

- Programski prevoditelj za ECDL
- ECDL program se prevodi se u Java program
- Oslanja se na postojeću razvojnu okolinu za razvoj evolucijskih algoritama ECFJ

Osnove ECDL-a

- Sintaksa slična programskom jeziku C
- Sve varijable predstavljaju multiskupove jedinki (nestaje potreba za deklaracijom)
- Operatori prilagođeni radu nad takvim varijablama
- Operatori rade s referencama jedinki

Osnove ECDL-a

```
if(population.size==0){
    //blok naredbi
}
else if(individual.size==population.size){
    //blok naredbi
}
else{
    //blok naredbi
}
```

Osnove ECDL-a

```
repeat(100){
    //blok naredbi
}

repeat(population.size){
    //blok naredbi
}

while(parents.size<100){
    //blok naredbi
}

for individual in population {
    //blok naredbi
}
```

Osnove ECDL-a

```
individual=population;

individual=population[0];


individual+=population;

individual+=population[0];


individual-=population;

individual-=population[0];
```

Osnove ECDL-a

```
individual==population

individual[0]!=population[0]

individual?population

individual[0]?population
```

Osnove ECDL-a

```
individual=select.random(population);

individual=select.best(population);

individual=select.worst(population);

individual=select.fitnessproportional(population);
```

Osnove ECDL-a

```
crossover(population[1],population[2],population[0]);

individual=crossover(population[0],population[1]);


mutate(individual);

newIndividual=mutate(individual);


evaluate(individual);

evaluate(individual[0]);
```

Osnove ECDL-a

```
repeat(population.size){
      tournament.clear;

      repeat(3){
            tournament+=select.random(population);
      }

      worst=select.worst(tournament);
      tournament-=worst;

      crossover(tournament[0],tournament[1],worst[0]);
      mutate(worst);
      evaluate(worst);
}
```

Prednosti ECDL-a

- Brz i jednostavan način za opis evolucijskih algoritama
- Oslobađa programera brige oko detalja
- Ne zahtjeva poznavanje arhitekture unutar koje se izvode evolucijski algoritmi
- Parametri algoritma ne specificiraju se kroz jezik
- Fast prototyping

Nedostaci ECDL-a

- Jezik je još u fazi razvoja
- Strojno generirani kod je obično sporiji od ručno pisanog koda
- ECDLC ne pruža napredne mogućnosti kao ostali programski prevoditelji

Moguće nadogradnje ECDL-a

- Omogućiti korisnički definirane funkcije
- Omogućiti opis nekih drugih metaheuristika
- Podržati C++ verziju ECF-a
- Omogućiti definiranje paralelnih algoritama

Zaključak

- ECDL omogućuje brzo i jednostavno definiranje osnovnih evolucijskih algoritama
- Nije se potrebno zamarati nepotrebnim detaljima
- Veliki prostor za poboljšanja i proširenja jezika i programskog prevoditelja

HVALA NA PAŽNJI!

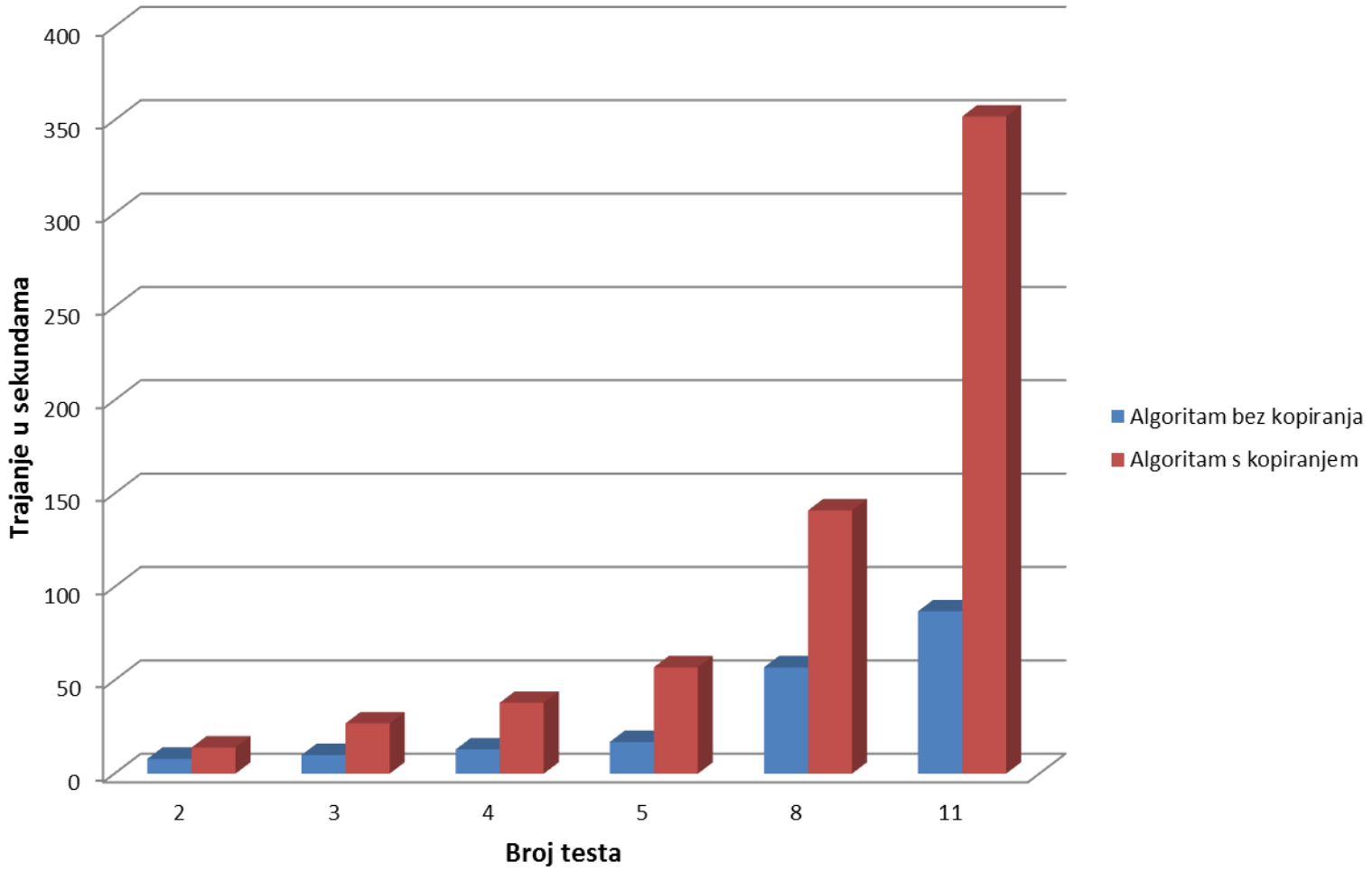
Primjer evolucijskog algoritma

```
repeat(population.size) {
    pool += select.fitnessproportional(population);
}

repeat(population.size * 0.3) {
        r1 = select.random(pool);
        r2 = select.random(pool);

        d1 = crossover(r1[0], r2[0]);
        d2 = crossover(r1[0], r2[0]);

        mutate(d1);
        mutate(d2);

        nova += d1;
        nova += d2;

        pool -= r1;
        pool -= r2;
}

nova += pool;
population = nova;
```

Algoritam sa kopiranjem jedinki

```
repeat(population.size){
tournament.clear;

repeat(3){
tournament+=select.random(population);
}

worst=select.worst(tournament);
tournament-=worst;
population-=worst;

worst=crossover(tournament[0],tournament[1]);
worst=mutate(worst);
evaluate(worst);

population+=worst;
}
```

Usporedba algoritma sa kopiranjem i bez kopiranja



Primjer izlaznog Java koda

```
import java.util.Vector;
import ecf.Deme;
import ecf.Individual;
import ecf.State;
import ecf.selection.SelectionOperator;
import ecf.selection.SelRandomOp;
import ecf.selection.SelWorstOp;

public class SteadyStateTournament extends Algorithm {

	private int nTournament;
	private SelectionOperator selRandomOp;
	private SelectionOperator selWorstOp;

	public SteadyStateTournament(State state) {
		super(state, "SteadyStateTournament");

		selRandomOp = new SelRandomOp();
		selectionOp.add(selRandomOp);

		selWorstOp = new SelWorstOp();
		selectionOp.add(selWorstOp);

	}
```

Primjer izlaznog Java koda

```
	public void initialize() {
		nTournament =
Integer.valueOf(getParameterValue("tsize"));
		if (nTournament < 3) {
			String poruka = "Error: SteadyStateTournament
algorithm requires minimum tournament size of 3!";
			state.getLogger().log(1, poruka);
			throw new IllegalArgumentException(poruka);
		}

		for (int i = 0; i < selectionOp.size(); i++) {
			selectionOp.get(i).initialize();
		}
	}

	public void registerParameters() {
		registerParameter("tsize", "3");
	}
```

Primjer izlaznog Java koda

```
public void advanceGeneration(Deme population) {

      Vector<Individual> helper = new Vector<Individual>();
      Vector<Individual> worst = new Vector<Individual>();
      Vector<Individual> tournament = new Vector<Individual>();


      for (int i0 = 0; i0 < population.size(); i0++) {
            tournament.clear();

            for (int i1 = 0; i1 < 3; i1++) {
                  tournament.add(selRandomOp.select(population));
            }

            worst.clear();
            worst.add(selWorstOp.select(tournament));
```

Primjer izlaznog Java koda

```
if (worst.size() == 1)
                        removeFrom(worst.get(0), tournament);
                  else
                        tournament.removeAll(worst);

                  mate(tournament.get(0), tournament.get(1),
worst.get(0));
                  mutate(worst);

                  for (int i2 = 0; i2 < worst.size(); i2++) {
                        worst.get(i2).evaluate();
                  }
            }

      }

}
```