Fakultet elektrotehnike i računarstva
Zavod za primjenjeno računarstvo

Napredni algoritmi i strukture podataka

2. laboratorijska vježba

Vinko Kolobara 0036475679

Zagreb, 10.12.2016.

1. Zadatak

Zadatci za 11 bodova
Genetskim algoritmom riješiti neki složeniji problem koji će zahtijevati pomniju razradu osnovnih mehanizama (križanje, mutacije, itd.). Poželjno je napraviti više različitih modela (mehanizama) i usporediti rezultate.
Za zadatak je odabrano rješavanje problema trgovačkog putnika pomoću genetskog algoritma.

2. Rješenje zadatka

2.1. Teorijski uvod

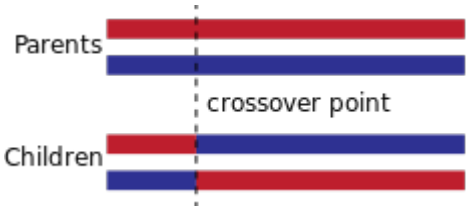
Genetski algoritam je metaheuristika inspirirana prirodnom evolucijom. Osnovna ideja je koristiti populaciju jedinki koje evoluiramo kroz generacije i stvaramo sve uspješnije jedinke. Uspješnost jedinke mjerimo funkcijom dobrote.
Jedinke možemo modelirati na različite načine. Uobičajeni prikazi su:

- Prikaz vektorom bitova
- Prikaz vektorom realnih vrijednost
- Prikaz permutacijskim vektorom

Evoluciju provodimo upotrebom genetskih operatora.
Operatori koji se koriste su:

- Selekcija
- Križanje
- Mutacija

Selekcija vrši odabir nekog broja jedinki iz populacije koje će se dalje koristiti u sljedećim operatorima.
Križanje kombinira dvije jedinke (roditelje) u novu jedinku (dijete) koja je donekle slična roditeljima.



Slika 1 - Primjer križanja

Mutacija uzima jednu jedinku i nasumično je izmijeni.

Slika 2 - Primjer mutacije

2.2. Implementacija

Implementacija laboratorijske vježbe rađena je u programskom jeziku Java, u razvojnom okruženju Eclipse.
Sastoji se od paketa **algorithm, fitness, operator, solution, tsp**. Svaki od tih paketa sadržava implementacije pojedinih dijelova potrebnih za rad genetskog algoritma.

2.2.1. Prikaz rješenja

Za prikaz rješenja je odabran prikaz permutacijskim vektorom. Taj prikaz sadrži n prirodnih brojeva od kojih se svaki smije pojaviti samo jednom. Takav prikaz je dobar zbog prirode problema trgovačkog putnika, jer nijedan grad ne posjećujemo više puta, a svaki moramo posjetiti bar jednom.

2.2.2. Operator selekcije

Za operator selekcije odabrana je eliminacijska k-turnirska selekcija. Ona radi tako što iz populacije odabire k jedinki, od kojih dvije najbolje odabire za roditelje pri križanju, a najlošiju zamijeni rezultatom križanja.

2.2.3. Funkcija dobrote

Funkcija dobrote je implementirana kao suma udaljenosti svih susjednih gradova u kromosomu.

2.2.4. Operatori križanja

Kao operatori križanja odabrani su **pohlepno križanje i PMX križanje.**
Pohlepno križanje radi tako što odabire prvi grad iz prvog roditelja, vidi sljedeći grad iz oba roditelja i odabire onaj koji čini kraću udaljenost. Nakon toga to ponavlja do popunjavanja cijele jedinke. Ako u nekom trenutku oba sljedeća grada budu već iskorištena, uzme nasumično grad od dosad neiskorištenih.
PMX križanje radi tako što uzme nasumični podniz iz prvog roditelja, odredi funkcije preslikavanja svakog elementa iz tog podniza u podniz na istom mjestu drugog roditelja i preostale elemente jedinke preslika u odgovarajući element.

2.2.5. Operatori mutacije

Kao operatori mutacije odabrani su **permutacijska mutacija, shuffle mutacija, swap mutacija.**
Permutacijska mutacija odabire nasumičan podniz djeteta i na tom mjestu promiješa elemente.
Shuffle mutacija radi isto kao i permutacijska, samo što koristi cijelo dijete.
Swap mutacija zamijeni mjesta dva nasumična elementa.

2.2.6. Modeliranje TSP

Zbog bržeg izvođenja, koristi se razred **ProblemDefinition** koji u sebi pohranjuje prethodno izračunate udaljenosti svih gradova po indeksima koje se dalje koriste u izračunu funkcije dobrote.

2.2.7. Genetski algoritam

Implementacija genetskog algoritma nalazi se u razredu **GeneticAlgorithm** i sastoji se od metode **solve(int solutionSize, FitnessFunction fitness)** koja izvodi genetski algoritam kroz generacije i na kraju vraća najbolje rješenje. Članske varijable razreda su:

- crossover (lista operatora križanja koji se koriste)
- mutation (lista operatora mutacije koji se koriste)
- selection (operator selekcije koji se koristi, eliminacijski)
- popSize (veličina populacije)
- maxIter (broj iteracija/generacija genetskog algoritma)
- pC (vjerojatnost križanja)
- pM (vjerojatnost mutacije)

Pseudokod genetskog algoritma:

```
population = randomPop(popSize)

do
     make pC*popSize crossovers
     make pM*popSize mutations
while numIter < maxIter
```

3. Zaključak

Genetski algoritam se pokazuje kao računski vrlo jeftino i dobro rješenje za rješavanje kombinatornih problema. Upotrebom jednostavnih operacija koje se primjenjuju na pojedine jedinke možemo dobiti dobra rješenja. Još jedna zanimljivost je da korištenjem slučajnih brojeva bez ideje gdje se nalazi najbolje rješenje, genetski algoritam ipak nerijetko uspijeva naći neko dobro rješenje. Ipak, zbog korištenja slučajnih brojeva, nikad ne znamo na čemu smo s ovim algoritmom, nekad nas može odvesti do najboljeg rješenja, a nekad može zaglaviti u lokalnom optimumu jako brzo. Kako bi se rad genetskog algoritma popravio moguće je pokrenuti više genetskih algoritama u isto vrijeme i dijeliti dijelove populacija među sobom. Dodatno, potrebno je isprobati i rad drugih operatora selekcije koji su možda bolji za problem trgovačkog putnika.

4. Literatura

[1] Prezentacije iz predmeta Napredni algoritmi i strukture podataka, Nikica Hlupić Mario Brčić
[2] Neizrazito, evolucijsko i neuroračunarstvo, Marko Čupić, Bojana Dalbelo Bašić, Marin Golub
[3] Genetski algoritam, http://zemris.fer.hr/~golub/ga/ga.html, Marin Golub