SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

**Problem raspoređivanja osoblja**

Heurističke metode optimizacija

**Studentski tim:**

Ivan Čapalija

Filip Gobin

Zagreb, Siječanj 2016.

Sadržaj

1. Opis problema 3

2. Opis primijenjenog algoritma 3

2.1. Način dobivanja početnog rješenja 3

2.2. Pseudokod generiranja početne populacije 4

2.3. Genetski algoritam 5

2.4. Operator selekcije 5

2.5. Operator križanja 5

2.6. Operator mutacije 6

2.7. Pseudokodovi operatora 6

3. Opis dobivenih rezultata i diskusija 7

4. Zaključak 8

# Opis problema

Organiziranje poslovnog kalendara i smjena zaposlenika zahvaća velik broj tvrtki sa velikim brojem zaposlenika koji nemaju isto radno vrijeme svaki dan. Problem koji je zadan za projekt inačica je puno poznatijeg problema organiziranja smjena u bolnicama (*eng. Nurse – Doctor scheduling problem*). Inačica koja se rješava u projektu sadrži više tvrdih ograničenja nego originalni problem te je modificirana funkcija za računanje dobrote rješenja korištenjem mekih ograničenja.

Postoji osam tvrdih ograničenja koja je potrebno zadovoljiti:

* Jedna smjena dnevno
* Rotacija smjena iz dana u dan za jednog zaposlenika
* Maksimalni broj smjena bez odmora
* Minimalni broj smjena bez odmora
* Minimalni broj uzastopnih dana odmora
* Minimalno i maksimalno radno vrijeme
* Maksimalni broj radnih vikenda
* Broj dana odmora

Slaba ograničenja na kojima se gleda dobrota rješenja su:

* Zahtjevi za dodjelu smjena
* Zahtjevi protiv dodjela smjena
* Pokrivanje smjena

# Opis primijenjenog algoritma

## Način dobivanja početnog rješenja

Za dobivanje početnog rješenja izgradili smo pohlepni algoritam koji svakom zaposleniku iterativno dodjeljuje smjene u ovisnosti o trenutnom stanju i zadovoljivosti čvrstih ograničenja.

Iznimno je teško bilo izgraditi pohlepni algoritam, jer svako rješenje mora zadovoljiti osam čvrstih ograničenja. U svakom koraku algoritma bilo je potrebno voditi računa smije li iduća smjena doći nakon prethodne, jesmo li dosegli ukupni broj smjena, jesmo li premašili maksimalno radno vrijeme, jesmo li premašili maksimalan broj uzastopnih smjena, jesmo li dodijelili dovoljan broj uzastopnih dana odmora, jesmo li premašili broj radnih vikenda i je li trenutni dan ujedno i dan godišnjeg odmora zaposlenika. Kod generiranja početnog rješenja nismo vodili računa o mekim ograničenjima, jer će ona biti optimirana genetskim algoritmom. U nastavku možete pronaći pseudokod pohlepnog algoritma koji je korišten za generiraje početnog rješenja.

## Pseudokod generiranja početne populacije

#data storage

string[][] data = new string[employeesNum][daysNum];

#iterate employees

for (employee in employees):

#iterate days

for (dayIndex in days):

if (previousDay is not empty):

previousShift = data[employee][day - 1];

else:

dayIndex += employee.minConsecutiveDaysOff;

consecutiveShifts = 0;

#set future shifts number

if (consecutiveShifts == 0):

futureShiftsNum = employee.minConsecutiveShifts;

else if (consecutiveShifts >= employee.minConsectiveShifts &&

consecutiveShifts < employee.maxConsecutiveShifts):

futureShiftsNum = 1;

else:

consecutiveShifts = 0;

continue;

consecutiveShifts += futureShiftsNum;

#suggest a list of valid consecutive shifts

suggestedShifts = suggestShifts(previousShift, futureShiftsNum);

#if there is not a valid list of consecutive shifts skip day

if (suggestedShifts == null):

continue;

#check hard constraints for current state

if (!checkHardConstraints(day)):

continue;

#write suggested shifts to employee

for (shiftIndex in len(suggestedShifts)):

data[employee][dayIndex + shiftIndex] = suggestedShifts[shiftIndex];

#skip days with assigned shifts

dayIndex += len(suggestedShifts) – 1;

## Genetski algoritam

Nakon što je generirana početna populacija, koristi se genetski algoritam za poboljšavanje rješenja. Pseudokod algoritma dan je u nastavku te su opisane operacije selekcije, križanja i mutacije.

#start genetic

Chromosomes[] population = generateStartingPopulation();

do while true

Chromosomes[] newPopulation = getSublist(population, elitismSize)

for (i = 1..populationSize \* (1 - elitismSize)):

Chromosome one = selection(population)

Chromosome two = selection(population)

Chromosomes[] results = Crossover(one, two)

for (Chromosome c in results):

mutate(c)

fitness.evaluate(c)

newPopulation.add(c)

population = newPopulation

check if sultion is satistifying:

break #finish if solutions are OK

## Operator selekcije

Operator selekcije je promijenjena klasična turnirska selekcija na način da se vjerojatnost odabira jedinke smanjuje linearno sa koeficijentom koji ovisi o veličini ulazne populacije. Veća je vjerojatnost da će se izabrati jedinka koja ima bolju dobrotu od jedinke koja ima manju.

## Operator križanja

Operator križanja izveden je tako da se ne može dobiti nevaljano riješenje. On radi tako da se zamijene retci rješenja, odnosno djeca nastala od roditelja imaju neke retke zamijenjene. Retci riješenja predstavljaju smjene pojedinog zaposlenika pa njihovom zamjenom ne može nastati nevaljano rješenje pod pretpostavkom da oba roditelja su valjana rješenja problema. Koeficijent križanja je vjerojatnost da će retci biti zamijenjeni.

## Operator mutacije

Operator mutacije je izveden na nekoliko načina kako bi se moglo s njim eksperimentirati. Jedna mutacija zamjenjuje stupce rješenja koji predstavljaju jedan dan, a ako tako rješenje ne zadovoljava tvrda ograničenja, onda se takvo rješenje ignorira. Drugi način je zamjena jednog dana u tjednu sa drugim nasumično izabranim danom. Treći način je generalizacija drugog načina, tako da se slučajno odabere između jednog i sedam dana te se pokuša zamijeniti sa drugim danima. U slučaju da takva mutacija nije zadovoljiva, rješenje se vraća na staro i pokušava se nad drugim zaposlenicima.

## Pseudokodovi operatora

#Selection

function selection(Chromosome[] chromosomes):

double alpha = startingProbability;  
double step = (alpha - 0.005) / chromosomes.size();

for ( i = 0 .. chromosomes.size())

alpha = alpha - step;  
if (random.nextFloat() < alpha)   
 solution = chromosomes.get(i);  
 break;

if solution is null:

solution = chromosomes.get(0)

return solution

#Crossover

function crossover(Chromosome first, Chromosome second):

if(random.nextFloat() < crossoverCoeff):  
 firstRowIndex = random.nextInt() % totalRows

secondRowIndex = random.nextInt() % totalRows

first.setRow(secondRowIndex, second.getRow(secondRowIndex))

second.setRow(firstRowIndex, first.getRow(firstRowIndex))

return [first, second]

#Mutation

function mutate(Chromosome chromosome):

if (random.nextFloat() < this.mutationCofficient):

int days = random.nextInt(6) + 1;  
int cols = chromosome.getColsNum();

for (int i = 0 … chromosome.getRowsNum() ):

int first = 0;  
int second = first + days - 1;  
  
while (Math.*abs*(second - first) < days):  
 first = random.nextInt(cols / days);  
 second = random.nextInt(cols / days);

for (int j = 0; j < days; j++)

firstElement = chromosome.getAt(i, first + j);  
secondElement = chromosome.getAt(i, second + j);  
chromosome.setAt(i, first + j, secondElement);  
chromosome.setAt(i, second + j, firstElement);

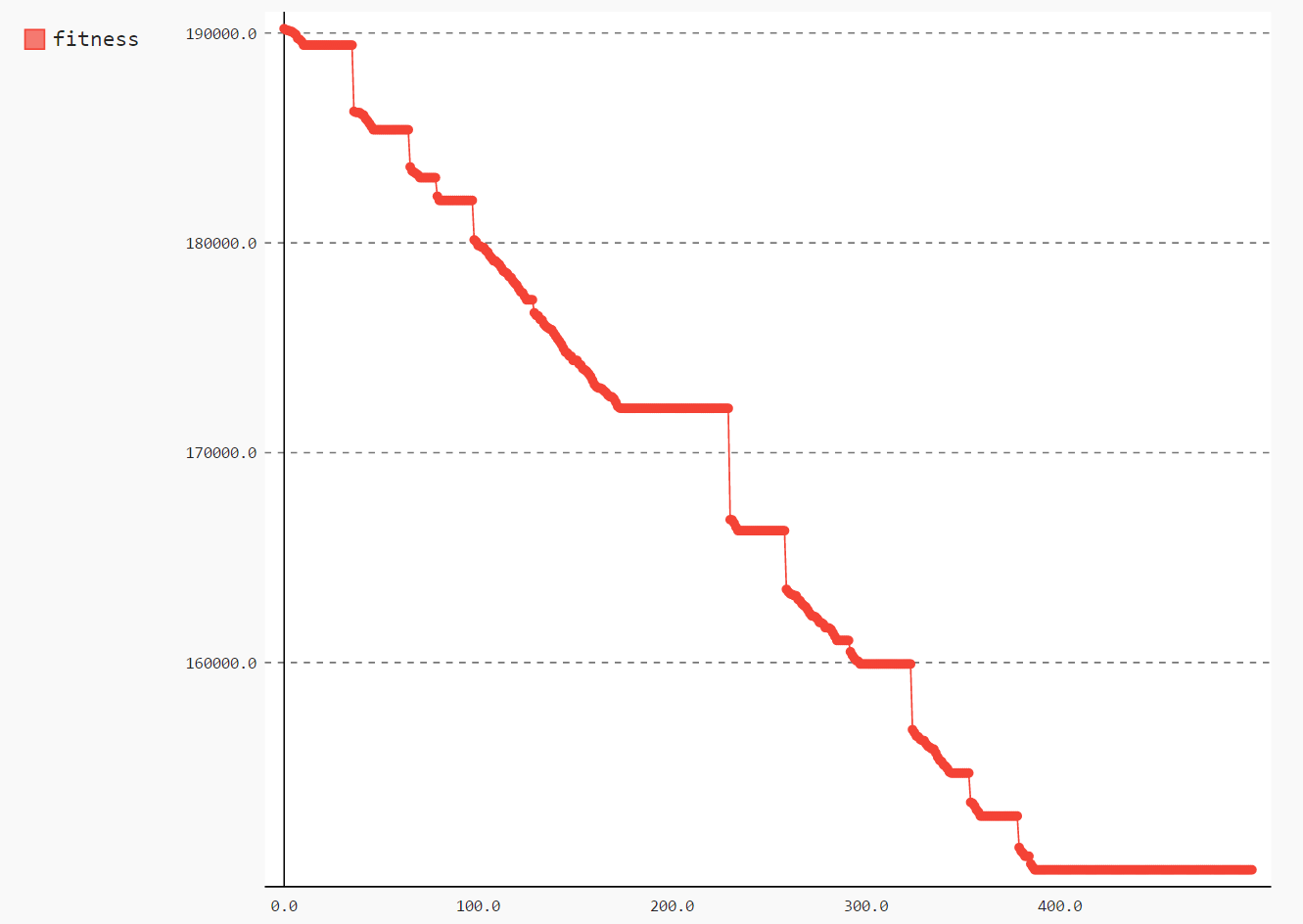
if chromosome doesn’t satisfy hard constraints rollback operations

return chromosome

# Opis dobivenih rezultata i diskusija

Nakon eksperimentiranja konačni algoritam je pokrenut sa sljedecim parametrima:

* Veličina populacije: 200
* Broj epoha: 500
* Koeficijent mutacije: 0.05
* Koeficijent selekcije: 0.2



*Slika 1 – Fitness najboljeg rješenja po epohama*

Fitness 10 najboljih rješenja:

* 150 816
* 150 109
* 150 133
* 150 638
* 152 038
* 150 712
* 150 604
* 150 616
* 150 126
* 151 866

Dobiveni rezultati svi zadovoljavaju tvrda ograničenja. Bolje rezultate bi možda dobili da je bilo više puta pokrenuti eksperiment, ali nakon svakog pokretanja nakon nekog vremena genetski algoritam bi zaglavio u nekom minimumu. Podizanje koeficijenta mutacije nije pomoglo u izvlačenju iz minimuma.

# Zaključak

Genetski algoritam je heuristika koja se često koristi za optimizaciju NP problema. Jedan od takvih problema je i stvaranje rasporeda rada na poslu. To je težak kombinatorički problem i potrebno je napraviti posebne metode za križanje i mutaciju. Zbog velikog broja jakih ograničenja bilo je potrebno napraviti zasebnu metodu za generiranje početne populacije, što je utjecalo na vrijeme razvoja ostatka algoritma. Također trebalo je razviti operatore na način da ne utječu na tvrda ograničenja. To nije bilo moguće za mutaciju pa treba razviti i metodu za oporavak od grešaka.

Nakon razvoja algoritma i dobivenih rezultata nije sigurno da je genetski algoritam jedna od boljih metoda za rješavanje ovakvog problema te je potrebna usporedba s ostalim heurističkim i metaheurističkim metodama poput *tabu pretraživanja* ili *GRASP algoritma* koji su više prilagođeni kombinatoričkim problemima.