Elektrotehnički fakultet
Univerzitet u Sarajevu
Odsjek za AiE/Odsjek za RI
Zimski semestar, a.g. 2018/2019.
Predmet: Optimizacija resursa

# Rješavanje problema rasporeda primjenom tabu pretraživanja

Sarajevo, Januar 2019.

Studenti: Lamija Hasanagić (1551/17544)
Tin Vidović (1544/17545)

## Sažetak

Ovaj seminarski rad govori o rješavanju problema rasporeda primjenom tabu pretraživanja. Za problem rasporeda odabran je raspored radnika na poslove i lokacije unutar smjena, za vrijeme tačno određenog vremenskog intervala (sedmice). U ovom radu dat je prikaz osnovnih karakteristika problema rasporeda, kao problema optimizacije, koji uključuje pretraživanje problemskog prostora s ciljem pronalaska optimuma pogodno odabrane kriterijalne funkcije. Kako je za rješavanje problema odabran algoritam tabu pretraživanja, to su u radu date i njegove osobine. Prikazana je implementacija algoritma tabu pretraživanja u *Python* programskom jeziku. Algoritam je testiran za različite veličine problemskog prostora, kao i za slučaj kada je početno rješenje problema unaprijed zadato, te za slučaj kada se početno rješenje bira sasvim slučajno. Rezultati testiranja u radu su prikazani, te su na osnovu toga izvedeni zaključci o radu algoritma kao i o njegovoj efikasnosti kada je u pitanju rješavanje navedenog problema.

## Abstract

This paper considers the solving of the timetabling problem through the use of the taboo search algorithm. A scheduling of a number of workers to a number of jobs and locations inside of a number of shifts (in one week) is considered. The paper presents the principal characteristics of the aforementioned timetabling problem as an optimization problem, consisting of finding the optimum of an adequately selected criteria function within the problem domain. Considering the taboo search algorithm was considered for the purpose of finding the optimum its characteristics are also presented. The implementation of the problem model and an adequate taboo search algorithm is given. The algorithm is tested on varying problem sizes. The effects on the algorithm of a completely random starting solution versus a starting solution chosen through heuristics are investigated. The test results are presented and on the basis of those results certain conclusions on the algorithm and its efficiency on the aforementioned problem are drawn.

# 1. Uvod

Tema ovog seminarskog rada jeste rješavanje problema rasporeda primjenom tabu pretraživanja. Rasporedi se koriste svakodnevno za organizaciju i planiranje. Oni se mogu pojaviti u obliku školskih rasporeda, rasporeda pozorišnih i kino predstava, rasporeda poslova radnicima unutar neke kompanije, voznih redova i sl.. Rasporedi se koriste kako bi olakašali svakodnevni život time što tačno definiraju ko, šta i kada radi, te kako bi optimizirali iskorištavanje resursa. Stoga se može reći da rješavanje problema rasporeda predstavlja optimizacijski problem, koji je kompleksan, i za čije rješavanje se mogu koristiti heuristički i metaheuristički algoritmi. U prvom poglavlju rada dat je opis problema. [1]

Kao problem rasporeda u okviru ovog seminarskog rada odabran je problem raspoređivanja radnika na poslove i lokacije unutar smjena, za vrijeme jedne sedmice.[2] Za rješavanje problema korišten je algoritam tabu pretraživanja koji predstavlja modifikaciju heurističkog algoritma lokalnog pretraživanja s ciljem umanjenja vjerovatnoće njegovog zapadanja u lokalne ekstreme. [3] U okviru prvog poglavlja također je dat pregled literature vezane za ovaj problem, a nevedene su i moguće aplikacije u praksi.

## 1.1. Opis problema

Kako je već navedeno problem rasporeda predstavlja kompleksan optimizacijski problem. Raspored je zapravo alokacija resursa u vremenu i prostoru.[1] Problem rasporeda moguće je matematički modelirati kao problem minimizacije pogodno odabrane funkcije kriterija na problemskom prostoru definsanom skupom ograničenja koja mogu biti tvrda i meka. Tvrda ograničenja su ona koja striktno trebaju biti zadovoljena, dok meka ograničenja nužno ne moraju biti zadovoljena, ali njihovo zadovoljenje bitno utiče na kvalitet rješenja problema. U ovom radu opisano je svođenje problema rasporeda u formu koja je pogodna za primjenu algoritma tabu pretraživanja, a što podrazumjeva definiranje problemskog prostora određivanjem ograničenja (tvdih i mekih) za konkretan problem, te određivanjem matematičke forme kriterijalne funkcije. U suštini problem rasporeda se svodi na traženje ekstremuma (minimuma) tako odabrane funkcije ovisne o više promjenljivih varijabli.

Posmatrajmo problem traženja ekstremuma funkcije više promjenljivih zadat relacijom (1.1.). Neka u relaciji **x** predstavlja tačku n dimenzionalnog prostora, koja se sastoji od n promjenljivih, a $\Omega$ predstavlja posmatrani problemski prostor, koji je u opštem slučaju n dimenzionalan. [3]

$$v^+ = \max_{x \in \Omega} f(x) \qquad (1.1.)$$

Potencijalno rješenje takvog problema **x,** je dato tačkom u posmatranom n dimenzionalnom prostoru, koja je sačinjena od n komponenti, a što je prikazano u relaciji (1.2.). [3]

$$\boldsymbol{x} = \begin{bmatrix} x1 \\ \vdots \\ x_n \end{bmatrix} \qquad (1.2.)$$

### 1.2. Pregled literature vezane za opisani problem

Problem rasporeda predstavlja kompleksan problem traženja rješenja primjenom heurističkih i metaheurističkih algoritama. U radovima na ovu temu date su formulacije samog problema rasporeda u vidu određivanja ograničenja koja se nameću, te u skladu s njima definisanje problemskog prostora koji se pretražuje i odabir pogodne kriterijalne funkcije te odabir prikladnog algoritma za pretraživanje problemskog prostora. S obzirom na različitosti samih algoritama koji se primjenjuju u optimizaciji u smislu njihovog kvaliteta i efikasnosti, te različitosti oblika u kojima se rasporedi mogu pronaći, moguće je naći u literaturi rješenje problema školskog rasporeda primjenom algoritama optimizacije kolonijom mrava [1],rješenje problema rasporeda poslova radnicima na određenim lokacijama i u određenim smjenama u okviru određenog vremenskog perioda primjenom tabu pretraživanja [2], te rješenje problema univerzitetskog rasporeda primjenom metoda lokalnog pretraživanja problemskog prostora [3]. Kako je za rješavanje problema obuhvaćenog ovim seminarskim radom odabrano tabu pretraživanje kao heuristička metoda za optimizaciju, potrebno se bilo upoznati i sa osnovnim karakteristikama algoritma tabu pretraživanja. [4]

### 1.3. Moguće aplikacije u praksi

Problem rasporeda, u opštem slučaju, predstavlja problem svakodnevnice. Raspored, bilo da je u pitanju školski raspored, univerzitetski raspored, raspored pozorišnih i kino predstava, voznih redova, te raspored poslova radnicima na određenim lokacijama, ima za svrhu da pojednostavi svakodnevni život čovjeka i da definiše ko, šta i kada radi, a da pri tome optimizira iskorištavanje resursa. U prošlosti izrada rasporeda bila je uglavnom ručna, međutim u današnje vrijeme sve više se teži automatizaciji izrade istih. Rješavanje problema rasporeda poslova radnicima na određenim lokacijama i u određenim smjenama koje je obuhvaćeno u okviru ovog seminarskog rada, implementirano je na računaru, u okviru programskog jezika Python, pa je samim tim rješavanje problema automatizirano. Moguće je uz neznatne izmjene postojećeg načina rješavanja ovog problema, a i upotrebom neke druge heurističke metode pretraživanje pored tabu pretraživanja, riješiti i probleme rasporeda drugih ovdje navedenih oblika rasporeda.

# 2. Korišteni algoritam

## 2.1. Opis rada korištenog algoritma

### 2.1.1. Razvoj tabu pretraživanja

Tabu pretraživanje se javilo kao modifikacija heurističkog algoritma lokalnog pretraživanja. Lokalno pretraživanje, iako veoma jednostavno, ostavlja mogućnost zapadanja u lokalne ekstreme, te su se zbog toga dobivena rješenja za većinu praktičnih problema pokazala kao vrlo loša. Tabu pretraživanje predložio je Fred Glover 1986. godine, a ideja je ležala u tome da se osnovni algoritam lokalnog pretraživanja modificira na način da mu se omogući pomjeranje iz lokalnog ekstremuma. [3]

Osnovna ideja tabu pretraživanja jeste da se algoritmu lokalnog pretraživanja omogući da se pomjeri u pravcu koji ne dovodi do poboljšanja kriterija u slučaju da nijedan tačka iz okoline trenutnog rješenja ne dovodi do poboljšanja kriterija. Da bi se spriječilo vraćanje u prethodno posjećena potencijalna rješenja koristi se kratkotrajna memorija nazvana tabu lista, u koju se pohranjuje historija pretraživanja. Ni jedna od tačaka iz okoline trenutnog rješenja koja se nalazi u tabu listi nije prihvatljiva kao novo potencijalno rješenje. [3]

### 2.1.2. Tabu i tabu lista

Tabu predstavlja potencijalno rješenje, neku njegovu karakteristiku ili pomak koji vodi u novo potencijalno rješenje, a koji trenutno nije dozvoljen. Tabui se pohranjuju u tabu listu koja se neprekidno osvježava tokom izvršavanja algoritma tabu pretraživanja.

Tabu lista je osnovni element po kome se algoritam tabu pretraživanja razlikuje od algoritma lokalnog pretraživanja. Proglašavanjem jedne ili više tačaka problemskog prostora tabuom omogućava se otklanjanje od trenutno posjećenog dijela problemskog prostora i intenziviranje njegovog istraživanja. U tabu listu se može pohranjivati fiksan broj tabua u vidu cijelih potencijlanih rješenja koja su prethodno posjećena, zatim u tabu listu moguće je pohraniti i nekoliko posljednjih transformacija primjenjenih na potencijalna rješenja čime se zabranjuju obrnute transformacije koje bi pretraživanje ponovo vratile u već posjećene tačke problemskog prostora. Uz pohranjivanje tabua, u tabu listu se može pohranjivati i rok trajanja savkog od tabua, koji se ažurira sa svakom iteracijom algoritma. Nakon što istekne rok trajanja nekog tabua on se uklanja iz tabu liste. Ukoliko je tabu lista fiksne dužine i fiksnog trajanja tabua, moguće je da se pretraživanje nakon određenog broja iteracija ponovo vrati u već posjećene tačke problemskog prostora. Radi toga je od strane više autora predložena implementacija tabu liste promjenljive dužine, ili korištenje promjenljivog trajanja tabua u listi.[3]

### 2.1.3. Prostor pretraživanja i dozvoljena okolina

Dva osnovna elementa heuristike tabu pretraživanja su definicija prostora pretraživanja i strukture okoline potencijalnog rješenja.
Prostor pretraživanja predstavlja skup svih tačaka problemskog prostora koje se mogu posjetiti za vrijeme pretraživanja. Vrlo često nije pogodno ograničiti prostor pretraživanja na skup dozvoljenih vrijednosti, nego se obično dozvoljava da se tokom izvršavanja algoritma posjete i tačke koje leže izvan skupa dozvoljenih vrijednosti.
Neka postoji operator pomaka $\delta(\boldsymbol{x})$takav da se njegovom primjenom na tačku $\boldsymbol{x}$ generira okolina te tačke prikazana u relaciji (2.1.).

$$N(\boldsymbol{x}, \delta) = \{\boldsymbol{x}' \in \Omega \mid \boldsymbol{x}' = \delta(\boldsymbol{x})\} \qquad (2.1.)$$

Pretraživanjem svih ili dijela tačaka iz ove okoline se određuje (ukoliko postoji) tačka koja poboljšava kriteriji, te se ta tačka uzima kao novo potencijalno rješenje. Neka $T(\boldsymbol{x})$ predstavlja tabu listu u tački $\boldsymbol{x}$. Kod tabu pretraživanja se definira dozvoljena okolina tačke $\boldsymbol{x}$ prikazana u relaciji (2.2.).

$$\tilde{N}(\boldsymbol{x}, \delta) = N(\boldsymbol{x}, \delta) \backslash T(\boldsymbol{x}) \qquad (2.2.)$$

i dozvoljava se pomak samo u tačke koje se nalaze u takvoj suženoj okolini trenutnog potencijalnog rješenja. [3]

### 2.1.4. Osnovni algoritam tabu pretraživanja

Osnovni algoritam tabu pretraživanja prikazan je na Slici 2.1.. To je algoritam koji u svakoj iteraciji odabire onu tačku iz dozvoljene okoline tekućeg rješenja, koja dovodi do najvećeg poboljšanja kriterija. Početna tačka $\boldsymbol{x^0}$ se bira slučajno ili korištenjem neke heuristike i evaulira se njena vrijednost pomoću kriterija f. Ova tačka predstavlja teluće najbolje rješenje problema, Inicijalizira se prazna tabu lista. Nakon toga započinje osnovni ciklus algoritma tabu pretraživanja, koji završava nakon ispunjenja uslova zaustavljanja.
U okviru osnovnog ciklusa algoritma se evaluiraju tačke $\boldsymbol{x}'$ iz dozvoljene okoline tekućeg rješenja $\tilde{N}(\boldsymbol{x}, \delta)$ generirane korištenjem nekog operatora pomaka $\delta$. Sve tačke koje su bolje od tekećeg rješenja postaju kandidati da postanu novu tekuće rješenje problema. Pretraživanje okoline tekućeg rješenja se provodi dok se ne ispuni neki uslov. Po završetku pretraživanja okoline tekućeg rješenja se kao novo tekuće rješenje bira jedna od tačaka iz skupa $\boldsymbol{\Omega'}$, ukoliko ovaj skup nije prazan. U skladu sa izabranim novim tekućim rješenjem se ažurira tabu lista. [3]


*Slika 2.1. Osnovni algoritam tabu pretraživanja [3]*

## 2.2. Svođenje opisanog problema u formu korištenog algoritma

Kao što je spomenuto u prvom poglavlju problem rasporeda se može matematički modelirati kao problem minimizacije pogodne odabrane funkcije kriterija na problemskom prostoru definisanom nekim skupom ograničenja. U nastavku će biti opisana odabrana tvrda ograničenja (koja moraju biti zadovoljena), meka ograničenja (koja su poželjna i kao takva utiču na kvalitet rasporeda) te način svođenja problema kreiranja rasporeda u formu pogodnu za primjenu korištenog algoritma.

U prilogu A je data implementacija tabu algoritma za rješavanje problema rasporeda u python programskom jeziku. Na ovom mjestu će se koristiti isječci koda kako bi se objasnilo svođenje problema u formu pogodnu za rješavanje tabu algoritmom.

Raspored je predstavljen kao objekat koji se sastoji od proizvoljnih parametara koji se zadaju u zavisnosti od konkretnog rasporeda, a implementiran je na način prikazan na Slici 2.2.

```
class Raspored:

    def __init__(self, brojZaposlenika, brojSmjena, brojPoslova, brojLokacija , brojSmjenaPoUgovoru, listaBrojaObavljanjaPoslova):
```

*Slika 2.2. : Konstruktor klase raspored*

Dakle jedan raspored se "sastoji" od:

- Broja smjena po ugovoru *b* koji predstavlja koliko radnih dana je radnik ugovorom obavezan da radi
- Liste zaposlenika *zaposlenici* koja predstavlja radnike $z_i$ , i = 1 .. brojZaposlenika koji su u stanju obavljati neki posao $p_i$ na nekoj lokaciji $l_i$ u nekoj smjeni $s_i$.

- Liste smjena *smjene* koja predstavlja sve dostupne smjene $s_i$ , i = 1 .. brojSmjena u kojoj jedan radnik *z* može obavljati jedan posao *p* na jednoj lokaciji *l*
- Liste poslova *poslovi* koji predstavljaju poslove koje treba obaviti određeni broj puta u nekom vremenskom periodu (danu) na lokacijama na kojima se ti poslovi mogu obavljati
- Liste lokacija *lokacije* koja predstavlja sve dostupne lokacije na kojima se mogu obavljati poslovi *p*
- Lista lokacija *lokacije_i*, i = 1 .. brojPoslova, gdje lista *lokacije_i* predstavlja listu lokacija na kojima se može obavljati posao $p_i$
- Lista broja obavljanja poslova *broj_obavljanja_posla_i*, i = 1 .. brojPoslova, gdje lista *broj_obavljanja_posla_i*, predstavlja zahtjevani broj obavljanja posla $p_i$ u nekom vremenskom periodu (danu)
- Lista preferenci zaposlenika *preference_i*, i = 1 .. brojZaposlenika, gdje lista *preference_i* predstavlja preference zaposlenika $z_i$ za svaku od smjena $s_j$, gdje 0 predstavlja da zaposlenik $z_i$ najviše preferira smjenu $s_j$, a 10 predstavlja da zaposlenik $z_i$ najmanje preferira smjenu $s_j$
- Lista kvalifikacija zaposlenika *kvalifikacije_i*, i = 1 .. brojZaposlenika, gdje lista *kvalifikacije_i* predstavlja kvalifikacije zaposlenika $z_i$ za svaki od poslova $p_j$, gdje 0 predstavlja da zaposlenik $z_i$ najviše preferira smjenu $s_j$, a 10 predstavlja da zaposlenik $z_i$ najmanje preferira smjenu $s_j$ i
- Matrice M dimenzija brojZaposlenika x brojSmjena u kojoj M[i][j] = 0 označava da zaposlenik i ne obavlja nijedan posao u smjeni j, dok M[i][j] = (*p*, *l*) označava da zaposlenik i obavlja posao *p* na lokaciji *l*

Uz ovako definisan raspored možemo odrediti čvrsta ograničenja koja moraju biti zadovoljena kako bi se neki raspored smatrao prihvatljivim:

- Niti jedan radnik *z* ne smije obavljati više poslova u jednoj radnoj sedmici od broja smjena zadataih po ugovoru *b*. Ograničenje je implementirano funkcijom prikazanoj na Slici 2.3.:

```
def provjeriZadovoljenostUgovora(self, M):

    M = M


    for i in range(self.getBrojZaposlenika()):
        brojac = 0
        for j in range(self.getBrojSmjena()):

            if M[i][j] != 0:
                brojac += 1


        if brojac > 5:

            return False

    return True
```

*Slika 2.3.: Implementacija ograničenja o zadovoljenosti ugovora*

- Svaki radnik *z* mora imati odmor od najmanje 2 smjene između obavljanja poslova. Ograničenje je implementirano funkcijom prikazanoj na Slici 2.4.:

```
def provjeriOdmorZaposlenika(self, M):

    M = M

    for i in range(self.getBrojZaposlenika()):
        smjeneUKojimaRadi = []
        for j in range(self.getBrojSmjena()):

            if M[i][j] != 0:
                smjeneUKojimaRadi.append(j)

        for j in range(len(smjeneUKojimaRadi) - 1):
            if smjeneUKojimaRadi[j+1] - smjeneUKojimaRadi[j] <= 2:
                return False

    return True
```

*Slika 2.4.: Implementacija ograničenja o obaveznom odmoru zaposlenika*

- Svaki posao $p_i$ se mora se mora obavljati na jednoj od adekvatnih lokacija *lokacije_i*. Ograničenje je implementirano funkcijom prikazanoj na Slici 2.5..

```
def provjeriDaLiJeIspravanParPosaoLokacija(self, M):

    M = M

    for i in range(self.getBrojZaposlenika()):
        for j in range(self.getBrojSmjena()):
            prihvatljiv = False
            if M[i][j] != 0:
                posao = self.getPosao(M[i][j][0])

                lokacija = M[i][j][1]


                for k in range(len(posao[2])):
                    if lokacija == posao[2][k]:
                        prihvatljiv = True
            else:
                prihvatljiv = True


            if not prihvatljiv:
                return False

    return True
```

*Slika 2.5.: Implementacija ograničenja o ispravnosti para posao/lokacija*

- Svaki posao $p_i$ se mora obaviti naznačeni broj puta dat sa *broj_obavljanja_posla_i*. Ograničenje je implementirano funkcijom prikazanoj na slici 2.6..

```
def provjeriPredvidjeniBrojObavljanjaPoslova(self, M):

    M = M


    listaPoslova = self.getPoslovi()

    pomocnaLista = []

    for i in range(len(listaPoslova)):
        pomocnaLista.append(0)

    for j in range(self.getBrojSmjena()):
        for i in range(self.getBrojZaposlenika()):
            if M[i][j] != 0:
                indeksPosla = 0
                for k in range(len(listaPoslova)):
                    if M[i][j][0] == listaPoslova[k][0]:

                        indeksPosla = k
                        pomocnaLista[indeksPosla] += 1

        if (j+1)%3 == 0:
           if pomocnaLista != self.getListaBrojaObavljanjaPoslova():
               return False
           pomocnaLista = []

           for i in range(len(listaPoslova)):
               pomocnaLista.append(0)



    return True
```

*Slika 2.6. Implementacija ograničenja o broju obavljanja poslova*

- U jednoj smjeni *s* se smije obavljati samo jedan posao na nekoj lokaciji *l*. Ograničenje je implementirano funkcijom prikazanoj na slici 2.7...

```
def provjeriZauzetostLokacije(self, M):

    for j in range(self.getBrojSmjena()):
        listaBrojaca = []
        for k in range(len(self.getLokacije())):
            listaBrojaca.append(0)
        for i in range(self.getBrojZaposlenika()):
            if M[i][j] != 0:
                listaBrojaca[M[i][j][1]-1] += 1


        for k in range(len(listaBrojaca)):
            if listaBrojaca[k] > 1:
                return False

    return True
```

*Slika 2.7.: Implementacija ograničenja o zauzetosti lokacije*

Za meka ograničenja su odabrana:

- Preference zaposlenika, povoljniji je onaj raspored u kojem su više zastupljene želje zaposlenika
- Kvalifikacije zaposlenika, povoljniji je onaj raspored u kojem zaposlenici obavljaju poslove za koje su najviše kvalifikovani
- Stabilnost smjene i lokacije, povoljniji su oni rasporedi u kojima zaposlenici rade u istoj smjeni ili na istoj lokaciji većinu radne sedmice

Imajući u vidu meka ograničenja formirana je funkcija kriterija:

$F = M[i][j] * (P_i * c1 + K_i * c2 + stabilnostsmjene_i * c3 + stabilnostlokacije_i * c4)$ (2.1.)

i = 1 .. brojZaposlenika. Gdje su $E_i$ = *lista_preferenci_i*, $K_i$ = *lista_kvalifikacija_i, a* $stabilnostsmjene_i$ i $stabilnostlokacije_i$ 0, ako su lokacija i smjena jednaka prethodnoj, a 1 u suprotnom, c1, c2, c3 te c4 su pogodne odabrane konstante koje određuju važnost pojedinih "dijelova" kriterijalne funkcije.
Računanje kvaliteta rasporeda je implementirano funkcijama prikazanim na Slici 2.8., Slici 2.9. :

```
def dajKvalitetRasporeda(self, c1, c2, c3):

    ## sume za razlicite grupe mekih ogranicenja
    F1 = 0
    F2 = 0
    F3 = 0
    F4 = 0

    preference = self.getPreferenceZaposlenika()
    kompetencije = self.getKompetencijeZaposlenika()
    M = self.getM()

    for i in range(self.getBrojZaposlenika()):
        for j in range(self.getBrojSmjena()):
            if M[i][j] != 0:
                F1 += c1*preference[i][j]
                indeksPosla = self.dajIndeksPosla(M[i][j][0])

                F2 += c2*kompetencije[i][indeksPosla]

    F3 += self.uracunajStabilnostSmjene(c3)
    F4 += self.uracunajStabilnostLokacija(c3)

    return (F1, F2, F3, F4)

def dajUkupniKvalitetRasporeda(self, c1, c2, c3):
    F1, F2, F3, F4 = self.dajKvalitetRasporeda(c1, c2, c3)

    return F1 + F2 + F3 + F4
```

*Slika 2.8.: Implementacija procjene kvaliteta rasporeda*

```
def uracunajStabilnostSmjene(self, c3):

    suma = 0
    prethodnaSmjena = None

    for i in range(self.getBrojZaposlenika()):
        prethodnaSmjena = None
        for j in range(self.getBrojSmjena()):
            if self.getM()[i][j] != 0:
                if prethodnaSmjena != None:
                    if (j - prethodnaSmjena) % 3 != 0:

                        suma += c3
                prethodnaSmjena = j

    return suma

def uracunajStabilnostLokacija(self, c3):

    suma = 0
    prethodnaLokacija = None

    for i in range(self.getBrojZaposlenika()):
        prethodnaLokacija = None
        for j in range(self.getBrojSmjena()):
            if self.getM()[i][j] != 0:
                if prethodnaLokacija != None:
                    if self.getM()[i][j][1] != prethodnaLokacija:

                        suma += c3
                prethodnaLokacija = self.getM()[i][j][1]

    return suma
```

*Slika 2.9.: Pomoćne funkcije za računanje stabilnosti smjene i lokacije*

Kako bi se na ovaj način definisan problem rasporeda mogao primjenjivati algoritam tabu pretraživanja potrebno je definisati i okolinu rasporeda R.

Definirano je nekoliko različitih definicija susjednog rasporeda rasporedu R, performanse kojih su analizirane u poglavlju 3, a zasnivaju se na tri operatora:

1. Raspored R1 i raspored R2 su jednaki ukoliko je M1[i][j] = M2[i][j] ∀i, j. Ukoliko se u rasporedu R2 odabere neki nenulti element matrice M2 i zamijeni sa nekim nultim elementom te matrice iz odgovorajuće kolone tada je primijenjen operator O1. Implementacija je urađena funkcijom prikazanom na Slici 2.10.:

```
def pomakZamjenaRadnika(self, trenutniRaspored):

    i = random.randint(0, self.getBrojZaposlenika()-1)
    j = random.randint(0, self.getBrojSmjena()-1)

    M = trenutniRaspored

    susjedniRaspored = []

    for k in range(len(M)):
        susjedniRaspored.append([])
        for l in range(len(M[0])):
            susjedniRaspored[k].append(M[k][l])

    while susjedniRaspored[i][j] == 0:
        i = random.randint(0, self.getBrojZaposlenika()-1)
        j = random.randint(0, self.getBrojSmjena()-1)

    jGornje = j
    jDonje = j

    while jGornje % 3 != 2:
        jGornje += 1

    while jDonje % 3 != 0:
        jDonje -= 1


    listaNezaposlenih = []
    for k in range(self.getBrojZaposlenika()):
        daLiNeRadi = True
        for l in range(jDonje, jGornje + 1):
            if susjedniRaspored[k][l] != 0:
                daLiNeRadi = False

        if daLiNeRadi:
            listaNezaposlenih.append(k)

    k = random.choice(listaNezaposlenih)

    susjedniRaspored[k][j] = susjedniRaspored[i][j]
    susjedniRaspored[i][j] = 0

    return susjedniRaspored
```

*Slika 2.10.: Implementacija operatora O1*

2. Raspored R1 i raspored R2 su jednaki ukoliko je M1[i][j] = M2[i][j] $\forall$ i, j. Ukoliko se u rasporedu R2 odabere neki nenulti element matrice M2 (p, l), te se lokacija l zamjeni sa novom lokacijom $l_n$

tada je primijenjen operator O2. Implementacija je urađena funkcijom prikazanom na Slici 2.11.:

```
def pomakZamjenaLokacije(self, trenutniRaspored):

    i = random.randint(0, self.getBrojZaposlenika()-1)
    j = random.randint(0, self.getBrojSmjena()-1)

    M = trenutniRaspored

    susjedniRaspored = []

    for k in range(len(M)):
        susjedniRaspored.append([])
        for l in range(len(M[0])):
            susjedniRaspored[k].append(M[k][l])

    while susjedniRaspored[i][j] == 0:
        i = random.randint(0, self.getBrojZaposlenika()-1)
        j = random.randint(0, self.getBrojSmjena()-1)


    posao = self.getPosao(susjedniRaspored[i][j][0])

    novaLokacija = random.choice(posao[2])

    susjedniRaspored[i][j] = (susjedniRaspored[i][j][0], novaLokacija)

    return susjedniRaspored
```

*Slika 2.11.: Implementacija operatora O2*

3. Raspored R1 i raspored R2 su jednaki ukoliko je M1[i][j] = M2[i][j] $\forall$i, j. Ukoliko se u rasporedu R2 odabere neki nenulti element matrice M2 (p, l), te se taj element "premjesti" u neku smjenu *s* unutar odgovarajućeg dana tada je primijenjen operator O3. Implementacija je urađena funkcijom prikazanom na Slici 2.12.:

```
def pomakZamjenaSmjene(self, trenutniRaspored):

    i = random.randint(0, self.getBrojZaposlenika()-1)
    j = random.randint(0, self.getBrojSmjena()-1)

    M = trenutniRaspored

    susjedniRaspored = []

    for k in range(len(M)):
        susjedniRaspored.append([])
        for l in range(len(M[0])):
            susjedniRaspored[k].append(M[k][l])

    while susjedniRaspored[i][j] == 0:
        i = random.randint(0, self.getBrojZaposlenika()-1)
        j = random.randint(0, self.getBrojSmjena()-1)

    posao =  trenutniRaspored[i][j][0]
    lokacija = trenutniRaspored[i][j][1]

    dan = int(j/3)

    novaSmjena = random.randint(dan*3, dan*3 + 2)

    susjedniRaspored[i][j] = 0

    susjedniRaspored[i][novaSmjena] = (posao, lokacija)

    return susjedniRaspored
```

*Slika 2.12.: Implementacija operatora O3*

Dva rasporeda R1 i R2 su susjedna ako je raspored R2 dobijen bilo kojom kombinacijom pojedinačnih prethodno opisanih rasporeda nad rasporedom R1. Npr. R2 je susjedni raspored rasporedu R1 ako:

R2 = (R1 ○ O1) ○ O2
R2 = ((R1 ○ O2) ○ O3) ○ O1

Sada se može primijeniti standardni tabu algoritam pretraživanja čija implementacija je data u prilogu A u vidu funkcije *TabuPretrazivanjeRaspored*.

# 3. Simulacijski rezultati

U ovom poglavlju su prikazani simulacijski rezultati dobiveni primjenom tabu pretraživanja na prethodno opisani problem rasporeda. Prikazane su performanse algoritma za različite dimenzije rasporeda, te za različite parametre algoritma tabu pretraživanja. Konačno su izvedeni zaključci o primjeni tabu pretraživanja na problem rasporeda modeliran na opisani način.

## 3.1. Postavka simulacija

Za testiranje algoritma odabrana su dva rasporeda i to:

- Raspored 1 sa 5 zaposlenika, 18 smjena (3 dnevno), 3 posla, 7 lokacija te maksimumom od 5 radnih dana u sedmici po radniku. Raspored je testiran sa unaprijed zadatim početnim rješenjem, kao i slučajno odabranim početnim rješenjem, koji su predstavljeni na Slici 3.1. i Slici 3.2. respektivno:

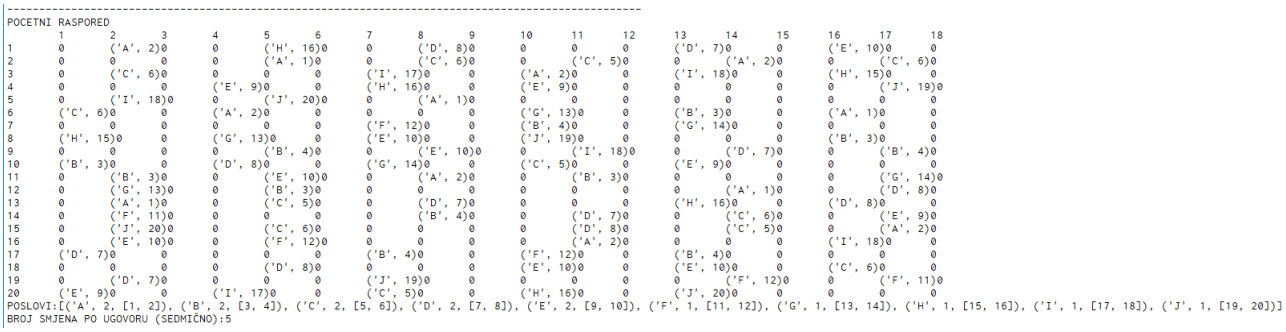
```
POCETNI RASPORED
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
1       0       0       0       0       ('A', 2)0       0       0       ('C', 3)0       0       ('C', 6)0       0       ('A', 1)0       0       0
2       ('A', 1)0       0       0       0       ('B', 4)0       0       ('B', 4)0       0       0       0       ('C', 7)0       0       ('B', 5)0
3       ('A', 2)0       0       ('A', 2)0       0       0       0       0       ('A', 1)0       0       ('A', 2)0       0       0       0       ('C', 6)
4       0       ('B', 2)0       0       ('C', 7)0       0       ('A', 2)0       0       ('B', 2)0       0       0       0       ('A', 1)0       0
5       0       0       ('C', 3)0       0       0       ('A', 1)0       0       ('A', 2)0       0       0       ('B', 5)0       0       ('A', 1)0
POSLOVI:[('A', 2, [1, 2]), ('B', 1, [2, 4, 5]), ('C', 1, [3, 6, 7])]
BROJ SMJENA PO UGOVORU (SEDMIČNO):5
PREFERENCE ZAPOSLENIKA:
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
1       0       5       10      10      0       6       7       10      0       0       8       0       2       1       0       5       10      6
2       9       0       9       10      7       0       4       5       0       10      2       0       9       0       3       5       0       6
3       10      5       1       0       10      4       6       0       9       0       10      8       0       9       4       10      9       0
4       0       4       8       2       0       10      5       0       3       10      0       10      9       1       10      0       10      3
5       2       4       0       0       9       7       0       1       10      0       5       0       2       0       6       0       0       10
KOMPETENCIJE ZAPOSLENIKA:
        1       2       3
1       0       10      10
2       10      10      0
3       0       10      10
4       10      0       10
5       0       10      10
KVALITET RASPOREDA PO TIPOVIMA OGRANICENJA: 230 1200 210 480
UKUPNI KVALITET RASPOREDA: 2120
```

*Slika 3.1.: Početno rješenje odabrano heuristikom za Raspored 1*

```
POCETNI RASPORED
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
1       ('A', 2)0       0       ('B', 2)0       0       0       0       0       ('A', 1)0       0       ('B', 2)0       0       0       ('B', 4)0       0
2       0       ('C', 6)0       0       ('C', 7)0       0       ('A', 1)0       0       0       0       ('C', 7)0       0       ('A', 1)0       0
3       0       0       0       0       0       0       ('B', 5)0       0       ('A', 2)0       0       ('A', 2)0       0       ('C', 6)0
4       0       ('B', 4)0       0       ('A', 1)0       0       ('A', 2)0       0       ('C', 7)0       0       0       0       ('A', 2)0       0
5       ('A', 1)0       0       ('A', 1)0       0       ('C', 6)0       0       ('B', 4)0       0       ('A', 1)0       0       0       0       0
POSLOVI:[('A', 2, [1, 2]), ('B', 1, [2, 4, 5]), ('C', 1, [3, 6, 7])]
BROJ SMJENA PO UGOVORU (SEDMIČNO):5
PREFERENCE ZAPOSLENIKA:
        1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
1       0       5       10      10      0       6       7       10      0       0       8       0       2       1       0       5       10      6
2       9       0       9       10      7       0       4       5       0       10      2       0       9       0       3       5       0       6
3       10      5       1       0       10      4       6       0       9       0       10      8       0       9       4       10      9       0
4       0       4       8       2       0       10      5       0       3       10      0       10      9       1       10      0       10      3
5       2       4       0       0       9       7       0       1       10      0       5       0       2       0       6       0       0       10
KOMPETENCIJE ZAPOSLENIKA:
        1       2       3
1       0       10      10
2       10      10      0
3       0       10      10
4       10      0       10
5       0       10      10
KVALITET RASPOREDA PO TIPOVIMA OGRANICENJA: 790 1300 60 480
UKUPNI KVALITET RASPOREDA: 2630
```

*Slika 3.2.: Slučajno odabrano početno rješenje za Raspored 1*

- Raspored 2 sa 20 zaposlenika, 18 smjena (3 dnevno), 10 poslova, 20 lokacija, te maksimumom od 5 radnih dana u sedmici po radniku. Raspored je testiran sa slučajno odabranim početnim rješenjem predstavljenim na Slici 3.3. :



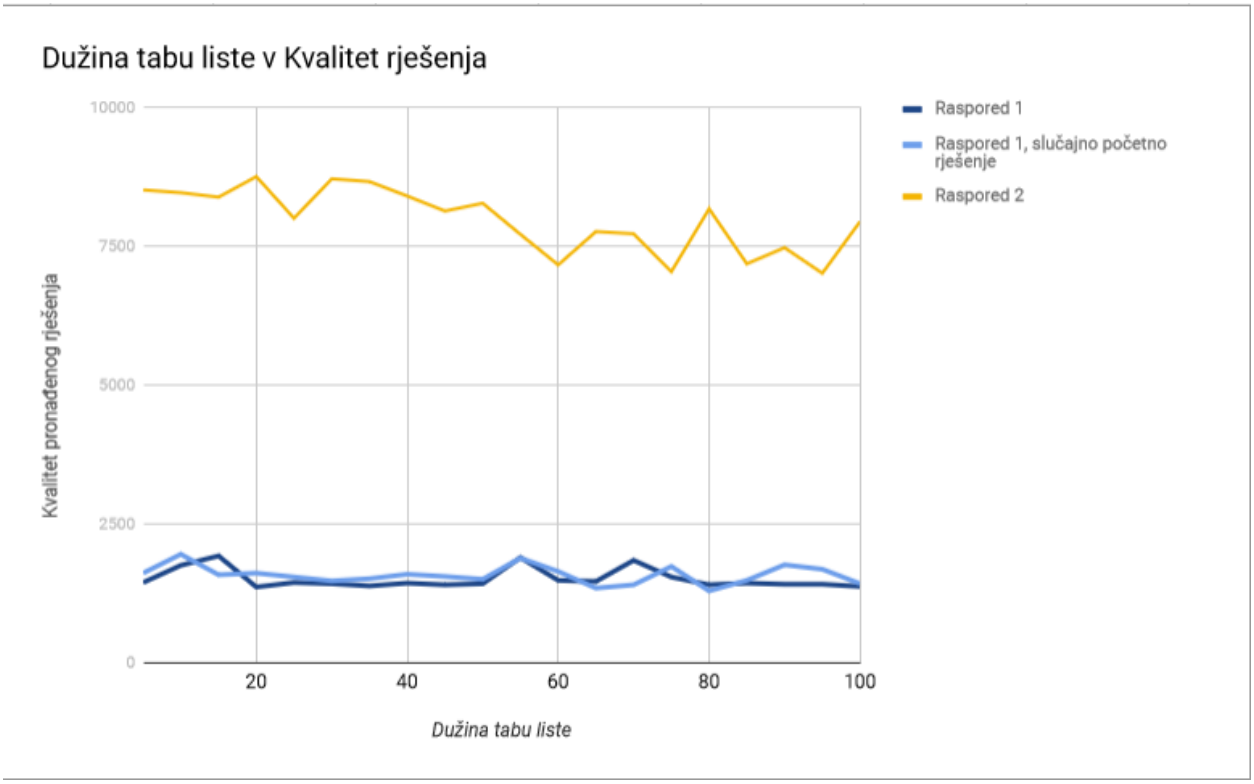
```
--------------------------------------------------------------------------------
POCETNI RASPORED
      1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18
1     0       ('A', 2)0       0       ('H', 16)0      0       ('D', 8)0       0       0       0       ('D', 7)0       0       ('E', 10)0      0
2     0       0       0       0       ('A', 1)0       0       ('C', 6)0       0       ('C', 5)0       0       ('A', 2)0       0       ('C', 6)0
3     0       ('C', 6)0       0       0       0       ('I', 17)0      0       ('A', 2)0       0       ('I', 18)0      0       ('H', 15)0      0
4     0       0       0       ('E', 9)0       0       ('H', 16)0      0       ('E', 9)0       0       0       0       0       0       ('J', 19)0
5     0       ('I', 18)0      0       ('J', 20)0      0       ('A', 1)0       0       0       0       0       0       0       0       0
6     ('C', 6)0       0       ('A', 2)0       0       0       0       ('G', 13)0      0       ('B', 3)0       0       ('A', 1)0       0
7     0       0       0       0       0       ('F', 12)0      0       ('B', 4)0       0       ('G', 14)0      0       0       0       0
8     ('H', 15)0      0       ('G', 13)0      0       ('E', 10)0      0       ('J', 19)0      0       0       0       0       ('B', 3)0       0
9     0       0       0       0       ('B', 4)0       0       ('E', 10)0      0       ('I', 18)0      0       ('D', 7)0       0       ('B', 4)0
10    ('B', 3)0       0       ('D', 8)0       0       ('G', 14)0      0       ('C', 5)0       0       ('E', 9)0       0       0       0       0
11    0       ('B', 3)0       0       ('E', 10)0      0       ('A', 2)0       0       ('B', 3)0       0       0       0       0       ('G', 14)0
12    0       ('G', 13)0      0       ('B', 3)0       0       0       0       0       0       0       ('A', 1)0       0       ('D', 8)0
13    0       ('A', 1)0       0       ('C', 5)0       0       ('D', 7)0       0       0       0       ('H', 16)0      0       ('D', 8)0       0
14    0       ('F', 11)0      0       0       0       ('B', 4)0       0       ('D', 7)0       0       ('C', 6)0       0       ('E', 9)0
15    0       ('J', 20)0      0       ('C', 6)0       0       0       0       ('D', 8)0       0       ('C', 5)0       0       ('A', 2)0
16    0       ('E', 10)0      0       ('F', 12)0      0       0       0       0       ('A', 2)0       0       0       0       ('I', 18)0      0
17    ('D', 7)0       0       0       0       0       ('B', 4)0       0       ('F', 12)0      0       ('B', 4)0       0       0       0       0
18    0       0       0       0       ('D', 8)0       0       0       0       ('E', 10)0      0       ('E', 10)0      0       ('C', 6)0       0
19    0       ('D', 7)0       0       0       0       ('J', 19)0      0       0       0       0       0       ('F', 12)0      0       ('F', 11)0
20    ('E', 9)0       0       ('I', 17)0      0       ('C', 5)0       0       ('H', 16)0      0       ('J', 20)0      0       0       0       0
POSLOVI:[('A', 2, [1, 2]), ('B', 2, [3, 4]), ('C', 2, [5, 6]), ('D', 2, [7, 8]), ('E', 2, [9, 10]), ('F', 1, [11, 12]), ('G', 1, [13, 14]), ('H', 1, [15, 16]), ('I', 1, [17, 18]), ('J', 1, [19, 20])]
BROJ SMJENA PO UGOVORU (SEDMIČNO):5
```

*Slika 3.3.: Slučajno odabrano početno rješenje za Raspored 2*

## 3.2. Rezultati simulacija

### 3.2.1. Uticaj dužine tabu liste na performanse algoritma

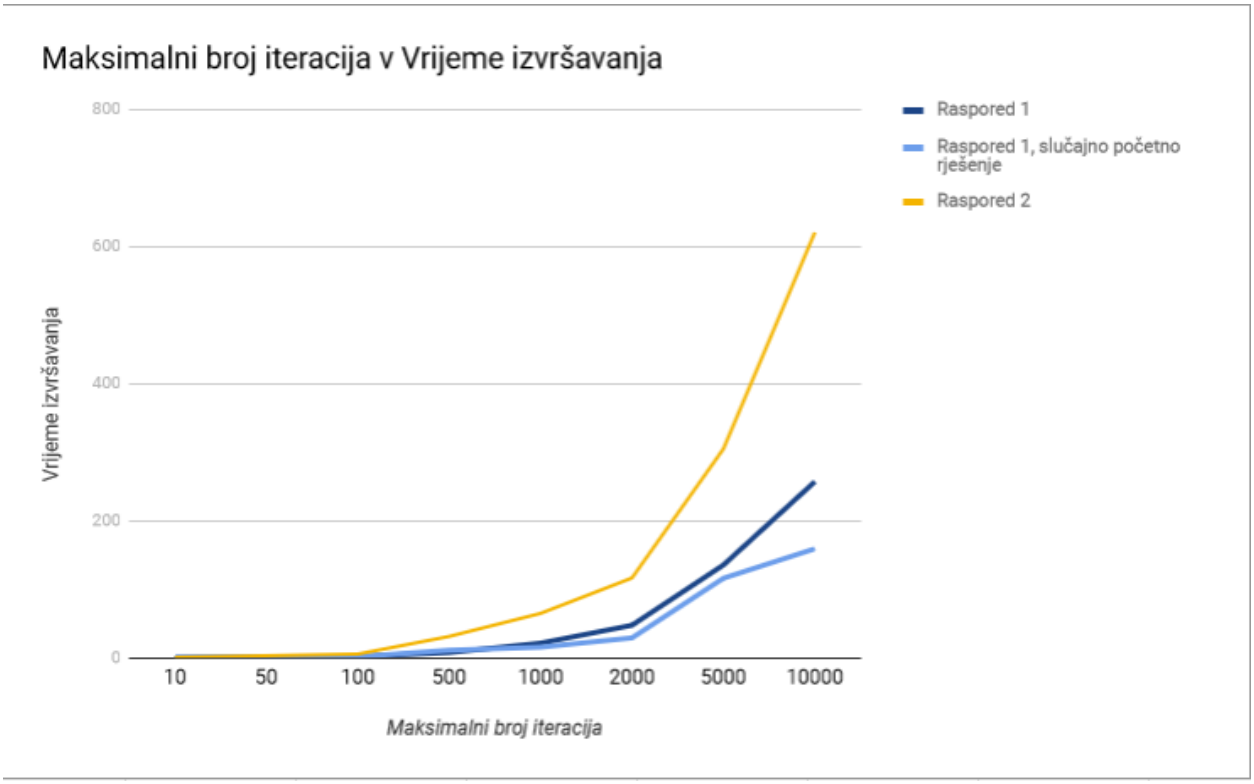
Dužina tabu liste je mijenjana od 5 do 100 sa rezolucijom od 5, pri čemu je parametar maksimalan broj iteracija postavljen na 500 za dužine tabu liste u intervalu [5, 50], a na 1000 za ostale dužine tabu liste. Na Slici 3.4. je prikazan kvalitet najboljeg pronađenog rasporeda u ovisnosti od dužine tabu liste:



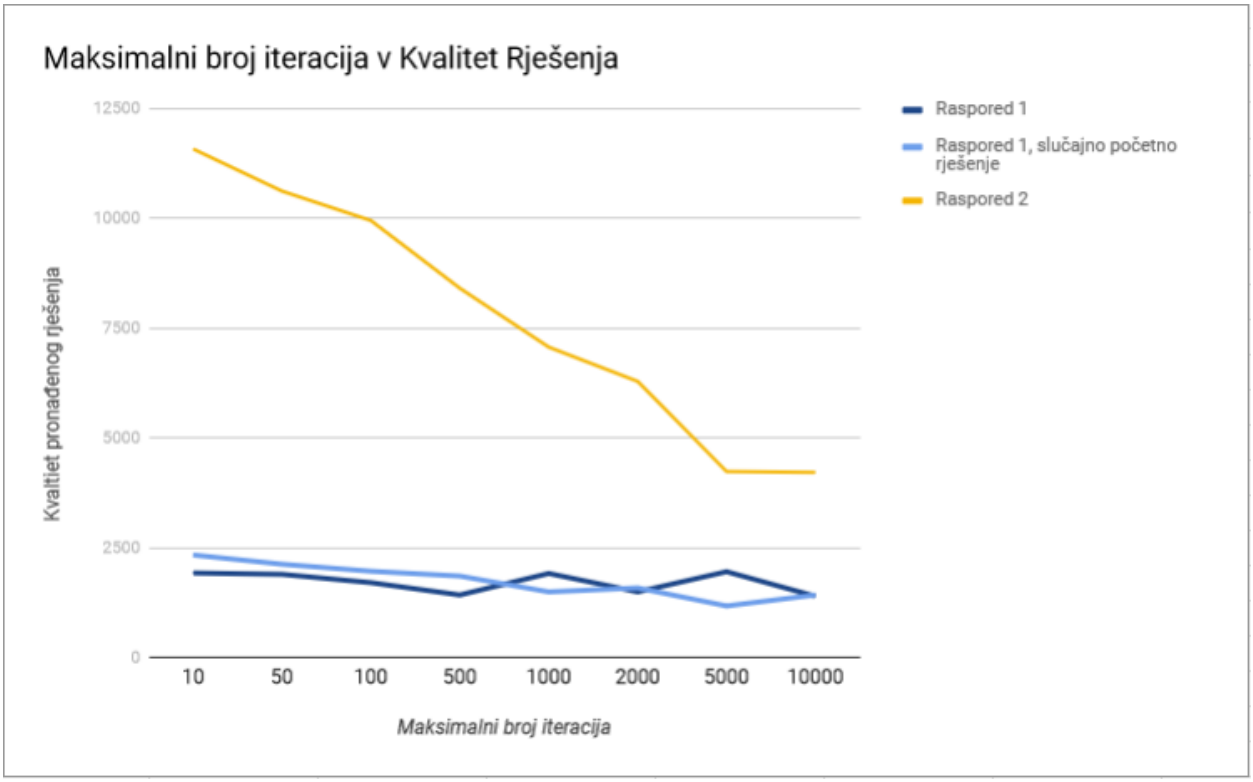

*Slika 3.4.: Uticaj dužine tabu liste na kvalitet pronađenog rješenja*

### 3.2.2. Uticaj maksimalnog broja iteracija na performanse algoritma

Algoritam tabu pretraživanja se proveo sa sljedećim zadanim maksimalnim brojem iteracija: 10, 50, 100, 500, 1000, 2000, 5000, 10000, dok je dužina tabu liste fiksirana na 20 za manji raspored a 75 za veći raspored. Na Slici 3.5. je prikazano vrijeme izvršavanja u ovisnosti od maksimalnog broja iteracija, dok je na Slici 3.6. prikazan kvalitet pronađenog rješenja u ovisnosti od maksimalnog broja iteracija:




*Slika 3.5.: Uticaj maksimalnog broja iteracija na vrijeme izvršavanja*




*Slika 3.6.: Uticaj maksimalnog broja iteracija na kvalitet pronađenog rješenja*

### 3.3. Zaključak

Na osnovu prezentovanih rezultata simulacije je moguće izvesti zaključke o performansama algoritma u zavisnosti od parametara samog algoritma kao i dimenzija problema koji se riješava.

Ovisnost vremena izvršavanja zavisi od veličine problema koji se rješava gdje najveći uticaj ima sam broj zaposlenika i poslova koje im je potrebno dodijelti. Također može se zaključiti da vrijeme izvršavanja eksponencijalno raste sa brojem iteracija što je i bilo očekivano.

Kvalitet pronađenog rješenja također zavisi od maksimalnog broja iteracija, gdje se primjećuju znatna poboljšanja pronađenog rješenja sa većim brojem iteracija. Kvalitet rješenja također zavisi i od dužine tabu liste, gdje određene dužine daju mnogo bolje rezultate za pojedine probleme, pa se može zaključiti da optimalna dužina tabu liste zavisi i od specifičnog problema.

Algoritam koji je polazio od nekog rasporeda zadanog višom heuristikom je u većini slučajeva davao nešto bolje rezultate, od algoritma koji je pretraživao problemski prostor sa slučajno odabranim početnim rješenjem. Ipak oba slučaja su davala približno jednake rezultate i izbor početnog rješenja zavisi od prirode konkretnog problema.

## 4. Zaključak i diskusija

Problem rasporeda predstavlja kompleksan problem u optimizaciji. Moguće ga je riješiti upotrebom poznatih heurističkih i metaheurističkih metoda.

U ovom radu opisan je način rješavanja problema rasporeda primjenom algoritma tabu pretraživanja koji predstavlja heurističku metodu dobivenu modifikacijom osnovnog algoritma lokalnog pretraživanja. U radu je predstavljen opis problema rasporeda i svođenje problema rasporeda u formu pogodnu za primjenu algoritma tabu pretraživanja. Određena su ograničenja potrebna za definisanje problemskog prosotora koji se pretražuje, te je pogodno odabrana funkcija kriterija potrebna za minimizaciju. Algoritam za rješavanje problema je i uspješno implementiran korištenjem programskog jezika Python.

U radu su dati i rezultati dobiveni testiranjem implementiranog algoritma kako bi se utvrdila njegova efikasnost. Došlo se do zaključka da parametri algoritma tabu pretraživanja kao što su maksimalan broj iteracija, dužina tabu liste, te paramteri funkcije kriterija, kao i dimenzionalnost samog problemskog prostora, bitno utiču na kvalitet dobivenog rješenja.

# Prilozi

## Prilog A

U ovom prilogu je dat listing koda.

```
from numpy import random
import copy
import time

c1 = 10
c2 = 10
c3 = 30
c4 = 30

class Raspored:
def __init__(self, brojZaposlenika, brojSmjena, brojPoslova, brojLokacija ,
brojSmjenaPoUgovoru, listaBrojaObavljanjaPoslova):
self._brojZaposlenika = brojZaposlenika
self._brojSmjena = brojSmjena
self._brojPoslova = brojPoslova
self._brojLokacija = brojLokacija
self._brojSmjenaPoUgovoru = brojSmjenaPoUgovoru
self._listaSlova = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
self._listaBrojaObavljanjaPoslova = []
self.setListaBrojaObavljanjaPoslova(listaBrojaObavljanjaPoslova)
self._Zaposlenici = []
self.setZaposlenici(self.getBrojZaposlenika())
self._Smjene = []
self.setSmjene(self.getBrojSmjena())
self._Poslovi = []
self.setPoslovi(self.getBrojPoslova())
self._Lokacije = []
self.setLokacije(self.getBrojLokacija())
self._M = []
self.setM(self.getBrojZaposlenika(), self.getBrojSmjena())
self._preferenceZaposlenika = []
self.setInicijalnePreference()
self._kompetencijeZaposlenika = []
self.setInicijalneKompetencije()
def __eq__(self, drugi):
if isinstance(drugi, self.__class__):
M = self.getM()
drugiM = drugi.getM()
if self.getBrojZaposlenika() != drugi.getBrojZaposlenika() or self.getBrojSmjena() !=
drugi.getBrojSmjena():
return False
```

```
for i in range(self.getBrojZaposlenika()):
for j in range(self.getBrojSmjena()):
if M[i][j] != drugiM[i][j]:
return False
return True
else:
return False
def postaviNaDrugi(self, drugi):
self.postaviRaspored(drugi.getM())
def __str__(self):
ispis = ""
ispis += "\t"
for i in range(self.getBrojSmjena()):
ispis += str(i+1) + "\t"
ispis += "\n"
for i in range(self.getBrojZaposlenika()):
ispis += str(self.getZaposlenici()[i]) + "\t"
for j in range(self.getBrojSmjena()):
if self.getM()[i][j] == 0:
ispis += str(self.getM()[i][j])+ "\t"
else:
ispis += str(self.getM()[i][j])+ ""
ispis += "\n"
ispis += "POSLOVI:" + str(self.getPoslovi()) +"\n"
ispis += "BROJ SMJENA PO UGOVORU (SEDMIČNO):" +
str(self.getBrojSmjenaPoUgovoru()) + "\nPREFERENCE ZAPOSLENIKA: \n"
ispis += "\t"
for i in range(self.getBrojSmjena()):
ispis += str(i+1) + "\t"
ispis += "\n"
for i in range(self.getBrojZaposlenika()):
ispis += str(self.getZaposlenici()[i]) + "\t"
for j in range(self.getBrojSmjena()):
ispis += str(self.getPreferenceZaposlenika()[i][j])+ "\t"
ispis += "\n"
ispis += "KOMPETENCIJE ZAPOSLENIKA:" + "\n"
ispis += "\t"
for i in range(self.getBrojPoslova()):
ispis += str(i+1) + "\t"
ispis += "\n"
for i in range(self.getBrojZaposlenika()):
ispis += str(self.getZaposlenici()[i]) + "\t"
for j in range(self.getBrojPoslova()):
ispis += str(self.getKompetencijeZaposlenika()[i][j])+ "\t"
ispis += "\n"
```

```
F1, F2, F3, F4 = self.dajKvalitetRasporeda(c1, c2, c3, c4)
ispis += "KVALITET RASPOREDA PO TIPOVIMA OGRANICENJA: " + str(F1) + " " +
str(F2) + " " + str(F3) + " " + str(F4) +"\n"
ispis += "UKUPNI KVALITET RASPOREDA: " + str(F1 + F2 + F3 + F4)
return ispis
def getBrojZaposlenika(self):
return self._brojZaposlenika
def getBrojSmjena(self):
return self._brojSmjena
def getBrojPoslova(self):
return self._brojPoslova
def getBrojLokacija(self):
return self._brojLokacija
def getListaBrojaObavljanjaPoslova(self):
return self._listaBrojaObavljanjaPoslova
def setBrojZaposlenika(self, brojZaposlenika):
if type(brojZaposlenika) != int:
raise TypeError("Broj zaposlenika mora biti cijeli broj!")
if brojZaposlenika <= 0:
raise ValueError("Broj zaposlenika mora biti pozitivan broj!")
self._brojZaposlenika = brojZaposlenika
def setBrojSmjena(self, brojSmjena):
if type(brojSmjena) != int:
raise TypeError("Broj smjena mora biti cijeli broj!")
if brojSmjena <= 0:
raise ValueError("Broj smjena mora biti pozitivan broj!")
self._brojSmjena = brojSmjena
def setBrojPoslova(self, brojPoslova):
if type(brojPoslova) != int:
raise TypeError("Broj poslova mora biti cijeli broj!")
if brojPoslova <= 0:
raise ValueError("Broj poslova mora biti pozitivan broj!")
self._brojPoslova = brojPoslova
def setBrojLokacija(self, brojLokacija):
if type(brojLokacija) != int:
raise TypeError("Broj lokacija mora biti cijeli broj!")
if brojLokacija <= 0:
raise ValueError("Broj lokacija mora biti pozitivan broj!")
self._brojLokacija = brojLokacija
def setBrojSmjenaPoUgovoru(self, brojSmjenaPoUgovoru):
if type(brojSmjenaPoUgovoru) != int:
raise TypeError("Broj lokacija mora biti cijeli broj!")
if brojSmjenaPoUgovoru <= 0:
raise ValueError("Broj lokacija mora biti pozitivan broj!")
self._brojSmjenaPoUgovoru = brojSmjenaPoUgovoru
def setListaBrojaObavljanjaPoslova(self, listaBrojaObavljanjaPoslova):
```

```
self._listaBrojaObavljanjaPoslova = []
if len(listaBrojaObavljanjaPoslova) != self.getBrojPoslova():
raise ValueError("Duzina liste broja obavljanja poslova ne odgovara broju poslova!")
for i in range(len(listaBrojaObavljanjaPoslova)):
self._listaBrojaObavljanjaPoslova.append(listaBrojaObavljanjaPoslova[i])
def getZaposlenici(self):
return self._Zaposlenici
def getSmjene(self):
return self._Smjene
def getPoslovi(self):
return self._Poslovi
def getPosao(self, nazivPosla):
for i in range(len(self.getPoslovi())):
if self.getPoslovi()[i][0] == nazivPosla:
return self.getPoslovi()[i]
return None
def getLokacije(self):
return self._Lokacije
def getBrojSmjenaPoUgovoru(self):
return self._brojSmjenaPoUgovoru
def setZaposlenici(self, brojZaposlenika):
self._Zaposlenici = []
for i in range(brojZaposlenika):
self._Zaposlenici.append(i+1)
def setSmjene(self, brojSmjena):
self._Smjene = []
for i in range(brojSmjena):
self._Smjene.append(i+1)
def setPoslovi(self, brojPoslova):
self._Poslovi = []
for i in range(brojPoslova):
self._Poslovi.append((self._listaSlova[i], self._listaBrojaObavljanjaPoslova[i], []))
def setLokacije(self, brojLokacija):
self._Lokacije = []
for i in range(brojLokacija):
self._Lokacije.append(i+1)
def setM(self, brojZaposlenika, brojSmjena):
self._M = []
for i in range(brojZaposlenika):
self._M.append([])
for j in range(brojSmjena):
self._M[i].append(0)
def postaviRaspored(self, raspored):
self._M = []
for i in range(len(raspored)):
self._M.append([])
```

```
for j in range(len(raspored[0])):
self._M[i].append(raspored[i][j])
def getM(self):
return self._M
def setInicijalnePreference(self):
self._preferenceZaposlenika = []
for i in range(self.getBrojZaposlenika()):
self._preferenceZaposlenika.append([])
for j in range(self.getBrojSmjena()):
self._preferenceZaposlenika[i].append(0)
def setInicijalneKompetencije(self):
self._kompetencijeZaposlenika = []
for i in range(self.getBrojZaposlenika()):
self._kompetencijeZaposlenika.append([])
for j in range(self.getBrojPoslova()):
self._kompetencijeZaposlenika[i].append(0)
def getPreferenceZaposlenika(self):
return self._preferenceZaposlenika
def getKompetencijeZaposlenika(self):
return self._kompetencijeZaposlenika
def setPreferenceZaposlenika(self, zaposlenik, listaPreferenci):
preferenceZaposlenika = self.getPreferenceZaposlenika()
for j in range(self.getBrojSmjena()):
if listaPreferenci[j] < 0 or listaPreferenci[j] > 10:
raise ValueError("Preference trebaju biti cijeli brojevi u opsegu [0 (najvisa preferenca), 10
(najmanja preferenca)]")
preferenceZaposlenika[zaposlenik-1][j] = listaPreferenci[j]
def setKompetencijeZaposlenika(self, zaposlenik, listaKompetencija):
kompetencijeZaposlenika = self.getKompetencijeZaposlenika()
for j in range(self.getBrojPoslova()):
if listaKompetencija[j] < 0 or listaKompetencija[j] > 10:
raise ValueError("Preference trebaju biti cijeli brojevi u opsegu [0 (najvisa preferenca), 10
(najmanja preferenca)]")
kompetencijeZaposlenika[zaposlenik-1][j] = listaKompetencija[j]

def setDozvoljeneLokacijeZaPosao(self, posao, listaLokacija):
posaoPostoji = False;
indexPosla = 0
for i in range(len(self.getPoslovi())):
if self.getPoslovi()[i][0] == posao:
posaoPostoji = True
indexPosla = i
if not posaoPostoji:
raise ValueError("Specificirani posao ne postoji!")
self.getPoslovi()[indexPosla] = (self.getPoslovi()[indexPosla][0],
self.getPoslovi()[indexPosla][1],listaLokacija)
```

```
def provjeriZadovoljenostUgovora(self, M):
M = M
for i in range(self.getBrojZaposlenika()):
brojac = 0
for j in range(self.getBrojSmjena()):
if M[i][j] != 0:
brojac += 1
if brojac > 5:
return False
return True
def provjeriOdmorZaposlenika(self, M):
M = M
for i in range(self.getBrojZaposlenika()):
smjeneUKojimaRadi = []
for j in range(self.getBrojSmjena()):
if M[i][j] != 0:
smjeneUKojimaRadi.append(j)
for j in range(len(smjeneUKojimaRadi) - 1):
if smjeneUKojimaRadi[j+1] - smjeneUKojimaRadi[j] <= 2:
return False
return True
def provjeriDaLiJeIspravanParPosaoLokacija(self, M):
M = M
for i in range(self.getBrojZaposlenika()):
for j in range(self.getBrojSmjena()):
prihvatljiv = False
if M[i][j] != 0:
posao = self.getPosao(M[i][j][0])
lokacija = M[i][j][1]
for k in range(len(posao[2])):
if lokacija == posao[2][k]:
prihvatljiv = True
else:
prihvatljiv = True
if not prihvatljiv:
return False
return True
def provjeriPredvidjeniBrojObavljanjaPoslova(self, M):
M = M
listaPoslova = self.getPoslovi()
pomocnaLista = []
for i in range(len(listaPoslova)):
pomocnaLista.append(0)
for j in range(self.getBrojSmjena()):
for i in range(self.getBrojZaposlenika()):
if M[i][j] != 0:
```

```
indeksPosla = 0
for k in range(len(listaPoslova)):
if M[i][j][0] == listaPoslova[k][0]:
indeksPosla = k
pomocnaLista[indeksPosla] += 1
if (j+1)%3 == 0:
if pomocnaLista != self.getListaBrojaObavljanjaPoslova():
return False
pomocnaLista = []
for i in range(len(listaPoslova)):
pomocnaLista.append(0)
return True
def provjeriZauzetostLokacije(self, M):
for j in range(self.getBrojSmjena()):
listaBrojaca = []
for k in range(len(self.getLokacije())):
listaBrojaca.append(0)
for i in range(self.getBrojZaposlenika()):
if M[i][j] != 0:
listaBrojaca[M[i][j][1]-1] += 1
for k in range(len(listaBrojaca)):
if listaBrojaca[k] > 1:
return False
return True
def provjeriPrihvatljivostRasporeda(self, M):
prihvatljiv = True
prihvatljiv = self.provjeriZadovoljenostUgovora(M)
if not prihvatljiv:
return prihvatljiv
prihvatljiv = self.provjeriOdmorZaposlenika(M)
if not prihvatljiv:
return prihvatljiv
prihvatljiv = self.provjeriDaLiJeIspravanParPosaoLokacija(M)
if not prihvatljiv:
return prihvatljiv
prihvatljiv = self.provjeriPredvidjeniBrojObavljanjaPoslova(M)
if not prihvatljiv:
return prihvatljiv
prihvatljiv = self.provjeriZauzetostLokacije(M)
if not prihvatljiv:
return prihvatljiv
return prihvatljiv

def kreirajSlucajniRaspored(self):
poslovi = copy.copy(self.getPoslovi())
M = copy.copy(self.getM())
```

```
zaposlenici = copy.copy(self.getZaposlenici())
brojDana = int(self.getBrojSmjena() / 3)
while True:
for dan in range(brojDana):
brojPosla = 0
random.shuffle(zaposlenici)
listaBrojaObavljanjaPoslova = copy.copy(self.getListaBrojaObavljanjaPoslova())
brojZaposlenika = 0
for i in range(self.getBrojZaposlenika()):
for j in range(3*dan, 3*dan + 2):
M[i][j] = 0
while listaBrojaObavljanjaPoslova[len(listaBrojaObavljanjaPoslova)-1] != 0:
kolikoZaposlenikRadiPoslova = 0
while True:
if brojZaposlenika >= self.getBrojZaposlenika():
brojZaposlenika = 0
for i in range(self.getBrojSmjena()):
if M[zaposlenici[brojZaposlenika]-1][i] != 0:
kolikoZaposlenikRadiPoslova += 1
if kolikoZaposlenikRadiPoslova == 5:
brojZaposlenika += 1
else:
break

if listaBrojaObavljanjaPoslova[brojPosla] == 0:
brojPosla += 1
if dan == 0:
smjena = random.randint(3*dan, 3*dan + 2)
else:
prethodnaSmjena = None
for i in range(3*(dan-1), 3*(dan-1) +2):
if M[zaposlenici[brojZaposlenika]-1][i] != 0:
prethodnaSmjena = i
smjena = prethodnaSmjena + 3
if prethodnaSmjena == None:
smjena = random.randint(3*dan, 3*dan + 2)
lokacija = random.choice(poslovi[brojPosla][2])
M[zaposlenici[brojZaposlenika] -1][smjena] = (poslovi[brojPosla][0], lokacija)
listaBrojaObavljanjaPoslova[brojPosla] -= 1
brojZaposlenika += 1
if self.provjeriPrihvatljivostRasporeda(M):
break
self._M = M
return
def setInicijalniRaspored(self):
Mtemp = self.getM()
```

```
M = []
for i in range(len(Mtemp)):
M.append([])
for j in range(len(Mtemp[0])):
M[i].append(Mtemp[i][j])
M[0][4] = ('A', 2)
M[0][8] = ('C', 3)
M[0][11] = ('C', 6)
M[0][14] = ('A', 1)
M[1][0] = ('A', 1)
M[1][5] = ('B', 4)
M[1][8] = ('B', 4)
M[1][13] = ('C', 7)
M[1][16] = ('B', 5)
M[2][0] = ('A', 2)
M[2][3] = ('A', 2)
M[2][9] = ('A', 1)
M[2][12] = ('A', 2)
M[2][17] = ('C', 6)
M[3][1] = ('B', 2)
M[3][4] = ('C', 7)
M[3][7] = ('A', 2)
M[3][10] = ('B', 2)
M[3][15] = ('A', 1)
M[4][2] = ('C', 3)
M[4][6] = ('A', 1)
M[4][9] = ('A', 2)
M[4][13] = ('B', 5)
M[4][16] = ('A', 1)
if not self.provjeriPrihvatljivostRasporeda(M):
return
else:
self._M = M
def pomakZamjenaRadnika(self, trenutniRaspored):
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
M = trenutniRaspored
susjedniRaspored = []
for k in range(len(M)):
susjedniRaspored.append([])
for l in range(len(M[0])):
susjedniRaspored[k].append(M[k][l])
while susjedniRaspored[i][j] == 0:
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
jGornje = j
```

```
jDonje = j
while jGornje % 3 != 2:
jGornje += 1
while jDonje % 3 != 0:
jDonje -= 1
listaNezaposlenih = []
for k in range(self.getBrojZaposlenika()):
daLiNeRadi = True
for l in range(jDonje, jGornje + 1):
if susjedniRaspored[k][l] != 0:
daLiNeRadi = False
if daLiNeRadi:
listaNezaposlenih.append(k)
k = random.choice(listaNezaposlenih)
susjedniRaspored[k][j] = susjedniRaspored[i][j]
susjedniRaspored[i][j] = 0
return susjedniRaspored
def pomakZamjenaLokacije(self, trenutniRaspored):
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
M = trenutniRaspored
susjedniRaspored = []
for k in range(len(M)):
susjedniRaspored.append([])
for l in range(len(M[0])):
susjedniRaspored[k].append(M[k][l])
while susjedniRaspored[i][j] == 0:
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
posao = self.getPosao(susjedniRaspored[i][j][0])
novaLokacija = random.choice(posao[2]) #probati obje mogucnosti: kada moze ostati ista
lokacija i kada se mora promjeniti u svakoj iteraciji
susjedniRaspored[i][j] = (susjedniRaspored[i][j][0], novaLokacija)
return susjedniRaspored
def pomakZamjenaSmjene(self, trenutniRaspored):
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
M = trenutniRaspored
susjedniRaspored = []
for k in range(len(M)):
susjedniRaspored.append([])
for l in range(len(M[0])):
susjedniRaspored[k].append(M[k][l])
while susjedniRaspored[i][j] == 0:
i = random.randint(0, self.getBrojZaposlenika()-1)
j = random.randint(0, self.getBrojSmjena()-1)
```

```
posao = trenutniRaspored[i][j][0]
lokacija = trenutniRaspored[i][j][1]
dan = int(j/3)
novaSmjena = random.randint(dan*3, dan*3 + 2)
susjedniRaspored[i][j] = 0
susjedniRaspored[i][novaSmjena] = (posao, lokacija)
return susjedniRaspored
def dajSusjedniRaspored(self):
susjedniRaspored = self.pomakZamjenaRadnika(self.getM())
susjedniRaspored = self.pomakZamjenaLokacije(susjedniRaspored)
susjedniRaspored =self.pomakZamjenaSmjene(susjedniRaspored)
return susjedniRaspored
def postaviPrihvatljivogSusjeda(self):
M = self.dajSusjedniRaspored()
while not self.provjeriPrihvatljivostRasporeda(M):
M = self.dajSusjedniRaspored()
self._M = M

def dajPrihvatljivogSusjeda(self):
M = self.dajSusjedniRaspored()
while not self.provjeriPrihvatljivostRasporeda(M):
M = self.dajSusjedniRaspored()
return M
def dajIndeksPosla(self, posao):
poslovi = self.getPoslovi()
for i in range(len(poslovi)):
if poslovi[i][0] == posao:
return i
return None
def uracunajStabilnostSmjene(self, c3):
suma = 0
prethodnaSmjena = None
for i in range(self.getBrojZaposlenika()):
prethodnaSmjena = None
for j in range(self.getBrojSmjena()):
if self.getM()[i][j] != 0:
if prethodnaSmjena != None:
if (j - prethodnaSmjena) % 3 != 0:
suma += c3
prethodnaSmjena = j
return suma
def uracunajStabilnostLokacija(self, c3):
suma = 0
prethodnaLokacija = None
for i in range(self.getBrojZaposlenika()):
prethodnaLokacija = None
```

```
for j in range(self.getBrojSmjena()):
if self.getM()[i][j] != 0:
if prethodnaLokacija != None:
if self.getM()[i][j][1] != prethodnaLokacija:
suma += c4
prethodnaLokacija = self.getM()[i][j][1]
return suma
def dajKvalitetRasporeda(self, c1, c2, c3, c4):
## sume za razlicite grupe mekih ogranicenja
F1 = 0
F2 = 0
F3 = 0
F4 = 0
preference = self.getPreferenceZaposlenika()
kompetencije = self.getKompetencijeZaposlenika()
M = self.getM()
for i in range(self.getBrojZaposlenika()):
for j in range(self.getBrojSmjena()):
if M[i][j] != 0:
F1 += c1*preference[i][j]
indeksPosla = self.dajIndeksPosla(M[i][j][0])
F2 += c2*kompetencije[i][indeksPosla]
F3 += self.uracunajStabilnostSmjene(c3)
F4 += self.uracunajStabilnostLokacija(c4)
return (F1, F2, F3, F4)
def dajUkupniKvalitetRasporeda(self, c1, c2, c3, c4):
F1, F2, F3, F4 = self.dajKvalitetRasporeda(c1, c2, c3, c4)
return F1 + F2 + F3 + F4

def azurirajTabuListu(x, tabuLista, duzinaTabuliste):
tabuLista.append(x)
while len(tabuLista) > duzinaTabuliste:
del tabuLista[0]
return tabuLista

def TabuPretrazivanjeRaspored(duzinaTabuListe, pocetniRaspored, maxBrojIteracija,
maxBrojSusjeda, c1, c2, c3, c4):
start = time.time()
## incijalizacija
x0 = copy.copy(pocetniRaspored)
v0 = x0.dajUkupniKvalitetRasporeda(c1, c2, c3, c4)
xNajbolje = copy.copy(x0)
vNajbolje = v0
## inicijalizacija taboo liste
tabuLista = []
brojacIteracija = 0
```

```
##trenutnoX
x = copy.copy(x0)
## dodajemo pocetno rjesenje u taboo listu
tabuLista = azurirajTabuListu(copy.copy(x), tabuLista, duzinaTabuListe)
brojIteracijaBezPoboljsanjaNajboljegRasporeda = 0
while True:
brojacIteracija += 1
if brojacIteracija > maxBrojIteracija:
break
brojSusjedaBrojac = 0
listaPosjecenihSusjeda = []
brojIteracijaZaSusjeda = 0
while True:
brojIteracijaZaSusjeda += 1
brojSusjedaBrojac += 1
if brojSusjedaBrojac > maxBrojSusjeda or brojIteracijaZaSusjeda > maxBrojSusjeda * 3:
break
x = copy.copy(x)
## eksperiment, ako se previse udaljavamo od optimuma
if brojIteracijaBezPoboljsanjaNajboljegRasporeda > 50:
x = copy.copy(xNajbolje)
## provjeravamo okolinu trenutnog x
x.postaviRaspored(x.dajPrihvatljivogSusjeda())
vecPosjecen = False
for i in range(len(listaPosjecenihSusjeda)):
if listaPosjecenihSusjeda[i] == x:
vecPosjecen = True
brojSusjedaBrojac -= 1
for i in range(len(tabuLista)):
if tabuLista[i] == x:
vecPosjecen = True
brojSusjedaBrojac -= 1
if not vecPosjecen:
listaPosjecenihSusjeda.append(x)
##print("LISTA POSJECENIH SUSJEDA")
##for l in range(len(listaPosjecenihSusjeda)):
##print(listaPosjecenihSusjeda[l])
## nije pronadjed nijedan validan susjed koji vec nije u taboo listi
if len(listaPosjecenihSusjeda) == 0:
print("EXHAUSTED NEIGHBORHOOD")
return (xNajbolje, vNajbolje)
## odredjivanje naboljeg susjeda
najboljiSusjed = copy.copy(listaPosjecenihSusjeda[0])
vrijednostNajboljegSusjeda = najboljiSusjed.dajUkupniKvalitetRasporeda(c1, c2, c3, c4)
for i in range(len(listaPosjecenihSusjeda)):
kvalitetItogRasporeda = listaPosjecenihSusjeda[i].dajUkupniKvalitetRasporeda(c1, c2, c3, c4)
```

```
if kvalitetItogRasporeda < vrijednostNajboljegSusjeda:
vrijednostNajboljegSusjeda = kvalitetItogRasporeda
najboljiSusjed = copy.copy(listaPosjecenihSusjeda[i])
## odredjivanje naboljeg susjeda
x = copy.copy(najboljiSusjed)
##po potrebi azuriramo najbolje rjesenje
brojIteracijaBezPoboljsanjaNajboljegRasporeda += 1
if x.dajUkupniKvalitetRasporeda(c1, c2, c3, c4) < vNajbolje:
vNajbolje = x.dajUkupniKvalitetRasporeda(c1, c2, c3, c4)
xNajbolje = copy.copy(x)
brojIteracijaBezPoboljsanjaNajboljegRasporeda = 0
## dodajemo trenutno rjesenje u taboo listu
tabuLista = azurirajTabuListu(copy.copy(x), tabuLista, duzinaTabuListe)
##print("ITERACIJA ")
##print(brojacIteracija)
##print("TABU LISTA")
##for l in range(len(tabuLista)):
##print(tabuLista[l])
end = time.time()
return (xNajbolje, vNajbolje, end - start)
```

## Reference

[1] Alan Tus, *Heurističke metode lokalne pretrage primijenjene na problem izrade rasporeda sati za škole*, Zagreb, 2012., preuzeto sa: https://bib.irb.hr/datoteka/594933.Diplomski_rad_-_Alan_Tus.pdf

[2] M. Chiarandini, A. Schaerf, F. Tiozzo, *Solving Emplowee Timetabeling Problems with Flexible Workload using Tabu Search*, Italy, preuzeto sa: https://imada.sdu.dk/~marco/Publications/Files/em-timetabling.pdf

[3] Samim Konjicija, *Predavanja na predmetu: Optimizacija resursa*, Elektrotehnički fakultet, Univerzitet u Sarajevu, Š.G. 2018./2019.

[4] O. Rossi-Doria, C. Blum, J. Knowles, M. Sampels, *A local search for the timetabeling problem*, Belgium, preuzeto sa: http://www.metaheuristics.net/downloads/ls_final.pdf