UNIVERZITET U SARAJEVU
ELEKTROTEHNIČKI FAKULTET

Rješavanje problema cjelobrojnog programiranja primjenom tabu pretraživanja

Seminarski rad iz predmeta Optimizacija resursa

Faris Čakarić
Azra Ibrahimović
Ajdin Kahrović

Sarajevo, februar 2015.

Sadržaj

1. Uvod

Problem cjelobrojnog programiranja je problem koji se javlja u cijelom nizu primjena. Efikasno pronalaženje optimuma kriterija u nekom cjelobrojnom problemskom prostoru je preduslov za rješavanje velike skupine problema. U situaciji kada su egzaktne metode nedovoljno brze, a kada dokazivost optimalnosti rješenja nije važna, što je tačno u većini praktičnih primjena, primjenjivost su našle heurističke metode za rješavanje takvih problema. Jedan takav, izuzetno efikasan i efektivan algoritam je algoritam tabu pretraživanja.

Cilj ovog rada je da upravo predloži način rješavanja problema cjelobrojnog programiranja koristeći algoritam tabu pretraživanja. U tom smislu, rad će opisati problem cjelobrojnog programiranja i preldožiti algoritam tabu pretraživanja prilagođen rješavanju ovog problema, na taj način dajući novi pristup u rješavanju ovakvih problema.

Na početku rada, opisan je problem cjelobrojnog linearnog programiranja, a zatim je opisan osnovni koncept tabu algoritama sa nizom koraka za njegovu implemetaciju, te su navedeni načini poboljšanja performasi uvodeći intenziviranje i diverzifikaciju. Nakon toga je predstavljena implementacija tabu algoritma za cjeloborojno programiranje u svom osnovnom obliku i sa predloženim poboljšanjima. Na kraju je demonstrirana primjena algoritma u rješavanju konkretnog problema: problem ranca. Na ovakav način struktuiran rad može u potpunosti ostvariti svoju namjenu.

2. Problem cjelobrojnog programiranja

Problem cjelobrojnog programiranja je problem matematičke optimizacije u kom postoji ograničenje na tip varijabli, odnosno neke ili sve varijable moraju biti cijeli nenegativni brojevi. U opštem slučaju pojam cjelobrojno programiranje se odnosi na cjelobrojno linearno programiranje. Za problem kažemo da je problem cjelobrojnog linearnog programiranja ukoliko funkcija f pripada klasi linearnih funkcija i ukoliko se skup Ω sastoji isključivo od izoliranih tačaka, gdje te tačke formiraju rešetkasti raspored i sve se nalaze unutar nekog poliedarskog skupa.[1] Skup Ω je skup ograničenja.

Cijelobrojno linearno programiranje spada u NP-teške probleme, za čije rješavanje je potrebno eksponencijalno računsko vrijeme. Za razliku od problema linearnog programiranja koji mogu biti rješeni u polinomijalnom vremenu.

Problemi cjelobrojnog programiranja kod kojih postoji ograničenje da sve promjenjive moraju biti cjelobrojne vrijednosti se naziva čisti problem cjelobrojnog programiranja, dok problemi cjelobrojnog programiranja kod kojih postoji ograničenje da neke od promjenjivih moraju biti cjelobrjene vrijednosti se naziva mješoviti problem cjelobrojnog programiranja.

Opći oblik cjelobrojnog linearnog programiranja[1] je dat u nastavku:

$$\arg\max Z = c^T x$$

uz ograničenja:

$$Ax = B$$
$$x \geq 0 \; i \; x \in Z^n$$

gdje je Z^n skup *n*-torki sa cjelobrojnim koordinatama*, n* je dimenzija vektora **x,** matrica **A** = [$a_{i,j}$] je matrica formata *m x n*, dok su dimenzije vektora **b** i **c** respektivno *m* i *n*. Matrica A i vektror b imaju cjelobrojne koordinate. Ukoliko je izostavljeno ograničenje $x \in Z^n$, problem predstavlja problem linearne relaksacije.

Postoje različiti pristupi rješavanja problema cjelobrojnog lienarnog programiranja, neke od ideja za rješavanje su: totalno pretraživanje, zaokruživanje rješenja linearne relaksacije, prevođenje u problem nelinearnog programiranja.[1]

Totalno pretraživanje se odnosi na princip pretraživanja svih tačaka iz okoline tačke, koja je prethodno pronađena, i upoređivanja vrijednosti funkcija kriterija. Totalno pretraživanje je pogodno za probleme manje dimenionalnosti.

Zaokruživanje rješenja linerne relaksacije je princip koji se bazira na rješavanju linearne relakcacije nekim od metoda za rješavanje linearnog programiranja, te se na rezultat primjenjuje zaokruživanje na najbliži cijeli broj, ukoliko rezultat nije cjelobrojna vrijednost.

Prevođenje problema cjelobrojnog programiranja u problem nelinearnog programiranja ima samo teoretski značaj, te u praksi nije primjenjiv.

Laki problemi cjelobrojnog programiranja su problemi čije linearne relaksacije uvijek imaju cjelobrojna optimalna rješenja. Dopustiva oblast je definisana kao cjelobrojni poliedar, te je matrica A totalno unimodularna.

Osnovna poteškoća kod cjelobrojnog linearnog programiranja je što za njega ne postoje kriteriji optimalnosti kojim bi se moglo prepoznati optimalno rješenje.[1] Stoga, rješavanje problema cjelobrojnog programiranja ima eksponencijalnu kompleksnost radi pretraživanja svih tačaka iz okoline, osim u situacijama manjih problema cjelobrojnog linearnog programiranja koji se mogu riješiti egzaktno koristeći metode odsjecanju dijelova dopustivog prostora koji ne sadrže cjelobrojna rješenja, grananja i ograničavanja ili grananja i odsjecanja, kao metod kominacije prva dva. [1]

[1] (Jurić & Mateljan, 2013.)

3. Osnovni koncept tabu algoritma

Tabu pretraživanje je modifikacija heurističkog algoritma lokalnog pretraživanja koji otklanja njegov osnovni nedostatak – zapadanje u lokalne ekstremume. Tabu pretraživanje je iterativni postupak koji počevši od početne tačke postepeno poboljšava potencijalno riješenje, pretražujući njegovu okolinu i pomjerajući se u najbolje riješenje iz te okoline.[2] Ukoliko, trenutna tačka daje najbolju vrijednost funkcije kriterija, algoritam tabu pretraživanja omogućava pomjeranje u pravcu koji ne dovodi do poboljšanja kriterija. Na ovaj način se omogućava pomjeranje iz lokalnog ekstremuma. Tabu pretraživanje se bazira na tri memorije:[2]

1. kratkoročna memorija (tabu lista) u koju se pohranjuje historija pretraživanja. Uloga ove memorije jeste da spriječi algoritam da se vraća u prethodno posjećena potencijalna riješenja. Tabu lista je implementirana na način da se za svaku tačku koja predstavlja potencijalno riješenje pohranjuju već posjećene tačke i njihov rok trajanja. Nakon što rok istekne, posjećene tačke napuštaju tabu listu.
2. memorija srednjeg trajanja koja omogućava povećanje efikasnosti algoritma kroz intenziviranje pretraživanja
3. dugoročna memorija koja omogućava pretraživanje neistraženih dijelova problemskog prostora koristeći diverzifikaciju pretražiavnja

Algoritam se završava nakon što se ispuni neki od kriterija zaustavljanja.
Kriterij asipiracije je uslov pod kojim se pretraživanje može nastaviti u nekoj od tačaka tabu liste ukoliko one daju bolju vrijednost funkcije kriterija od trenutne tačke. Primjenom ovog uslova moguće je uzeti u razmatranje tačke iz tabu liste koje prethodno nisu posjećene.

3.1. Algoritam tabu pretraživanja

Na početku izvršenja algoritma bira se početna tačka x_0. U početnoj tački se izračunava vrijednost funkcije kriterija, te se kreira prazna tabu lista. Odabrana tačka predstavlja najbolje riješenje. Osnovni dio algoritma se sastoji iz tri koraka i ponavlja se sve do ispunjenja nekog od uslova zaustavljanja:

1. za tačku koja predstavlja trenutno najbolje riješenje se kreira njena okolina $\widetilde{N}(x, \delta)$ primjenom operatora pomaka $\delta(x)$, takva da vijedi da je $\widetilde{N}(x, \delta) = N(x, \delta) \backslash T(x)$, gdje je $T(x)$ tabu lista u tački x.
2. Pretraživanje okoline tačke x se izvršava sve do ispunjenja uslova zaustavljanja pri čemu se za svaku tačku iz okoline izračunava vrijednost funkcije kriterija. Sve tačke koje imaju bolju vrijednsot od trenutnog najboljeg rješenja se pohranjuju u privremenu listu potencijalnih rješenja. U slučaju da niti jedna od tačaka iz okoline tačke x ne daje bolju vrijednost funkcije kriterija, algoritam tabu pretraživanja omogućava pomjeranje u neku od tačaka sa lošijom funkcijom kriterija kako bi spriječio zapadanje u lokalne ekstremume.
3. Za novo trenutno rješenje se bira tačka iz liste iz prethodnog koraka sa najboljom vrijednošću funkcije kriterija.

[2] (Konjicija, 2015)

3.2. Povećanje efikasnosti algoritma

Polazeći od nasumično generisane permutacije p, algoritam će naizmjnično izvršavati faze intenziviranja i defazifikacije. Faza intenziviranja koristi memoriju srednjeg trajanja za pohranu brojača. Brojači su vrijednosti koje se inkrementiraju svaki put kada se neka komponenta nalazi u potencijalnom riješenju. Svakom iteracijom se broj uvečava ukoliko je komponenta i dalje u riješenju, a ukoliko nije brojač se postavlja na 0. Komponente se mogu fiksirati nakon parametrom definisanog broja iteracija, nakon kojih komponenta ne može napustiti memoriju srednjeg trajanja. Fiksiranje komponente dovodi do intenziviranja pretraživanja.
Faza diverzifikacije koristi dugotrajnu memoriju za pohranu brojača. Brojači su vrijednosti koje se inkrementiraju svaki put kada se neka komponenta ne nalazi u potencijalnom riješenju. Koristeći komponente koje se ne nalaze u dugoročnoj memoriji omogućeno je istraživanje problemskog prostora koje nije do sada dovoljno istraženo.
Memorije obje faze se pohranjuju kao liste oblika: $X = [x_1, x_2, x_3, \dots x_n]$, gdje svaka od vrijednosti x_i predstavlja brojač.

4. Opis implementacije

Kod se sastoji od sljedećih klasa:

1. `IntegerProblem`
2. `TabuItem`
3. `TabuList`
4. `RecentMemory`
5. `LongTermMemory`
6. `TabuSearch`

Prva klasa, `IntegerProblem`, omogućava definiranje problema cjelobrojnog programiranja. Prilikom kreiranja objekta te klase potrebno je proslijediti string koji opisuje da li je u pitanju minimizacija ili maksimizacija, funkciju cilja, početnu tačku, te ograničenja u vidu liste funkcija. U slučaju da je u pitanju problem minimizacije, funkciju cilja je potrebno negirati.

Klasa `TabuItem` predstavlja element tabu liste, pri čemu konstruktor prima tačku i definira broj iteracija kroz koje će tačka predstavljati tabu. Preklopljeni operator jednakosti omogućava poređenje između dva objekta ove klase, pri čemu su ona jednaka ako i samo ako im je atribut point jednak. Metoda `reduce_time` omogućava postepeno brisanje tačke iz tabu liste, umanjujući broj iteracija koji je potreban da tačka ostane u listi.

Klasa `TabuList` predstavlja tabu listu i sadrži niz objekata klase `TabuItem`. Metoda `add` omogućava dodavanje tačaka u listu. Metodom `contains` se provjerava prisustvo neke tačke u tabu listi, dok metoda `refresh` poziva metodu `reduce_time` za svaki od objekata `TabuItem` klase.

U slučaju da se koristi intenzifikacija, instancira se objekat klase `RecentMemory`. Ova klasa prati promjene komponenti tekućeg rješenja i zapisuje broj iteracija kroz koji je prisutna određena komponenta u rješenju. Ukoliko je potrebno izvršiti diverzifikaciju, tada se koristi dugotrajna memorija, koja je implementirana kroz klasu `LongTermMemory`. Ova klasa spašava podatke o svim izmjenama koje su izvršene nad svakom od komponenti rješenja.

Tabu pretraživanje je implementirano kroz klasu `TabuSearch`. Konstruktor ove klase prima objekat klase `IntegerProblem`, maksimalni broj iteracija i dodatne opcije. Dodatne opcije predstavljaju niz koji može sadržavati string `'diversify'`, nakon čega je potrebno specificirati broj iteracija nakon kojih se započinje diverzifikacija, najmanja frekvencija pojavljivanja koju je određena komponenta potrebno da posjeduje kako bi postala kandidat za ubacivanje u tekuće rješenje, kao i broj komponenti koje se fiksiraju diverzificiranjem. Druga opcija je korištenje intenzifikacije navođenjem stringa `'intensify'`, nakon čega je potrebno specificirati broj iteracija kroz koje je potrebno da je komponenta prisutna u rješenju kako bi se fiksirala.

Metoda `find_neighbourhood` omogućava pronalazak susjednih tačaka u *n*-dimenzionalnom prostoru, pri čemu uzima u obzir moguće korištenje diverzifikacije ili intenzifikacije. U tom slučaju se određene komponente fiksiraju. Nakon toga je potrebno pronaći okolinu tekuće tačke koja se nalazi u dopustivoj oblasti, što se vrši pozivanjem metode `find_accessible_neighbourhood`, koja provjerava da li tačke pronađene prethodnom metodom zadovoljavaju zadana ograničenja.

Glavna metoda klase `TabuSearch` je metoda `next_iteration` koja izvršava jednu iteraciju algoritma. Na početku se pronalaze sve susjedne tačke koje se nalaze u dopustivoj oblasti, a nakon toga se među njima pronalazi tačka za koju je vrijednost funkcije kriterija najveća i koja se pri tome ne nalazi u tabu listi. Ukoliko ova tačka poboljšava vrijednost funkcije kriterija u odnosu na trenutno najbolju tačku, tada se ona postaje `best_point`. U svakom slučaju ona postaje `current_point` i dodaje se u tabu listu, te se tabu lista ažurira.

Metoda `plot` omogućava vizualizaciju izvršenja algoritma ukoliko je problem dvodimenzionalan. Tačke unutar dozvoljene oblasti su prikazane crnom bojom, tekuće rješenje crvenom, a tačke u tabu listi su zaokružene.

Shodno prethodno opisanom, problem cjelobrojnog programiranja se može definirati na sljedeći način:

```
problem = IntegerProblem('max', f, [0,0], [c1, c2, c3])
```

Objekat klase TabuSearch se kreira na sljedeći način:

```
tabu_search = TabuSearch(a,12, ['intensify', 5, 'diversify', 5, 2, 1])
```

Algoritam se izvršava pozivanjem metode next_iteration:

```
while tabu_search.iteration < tabu_search.max_iter:
   tabu_search.next_iteration()
   plt = tabu_search.plot([0, 9, 0, 6], 2, ['delay', 0.5])
```

Posljednja linija omogućava vizualizaciju svake od iteracija, pri čemu će između njih biti pauza u trajanju od 0.5s.

5. Demonstracija implementacije

5.1. Primjer 1

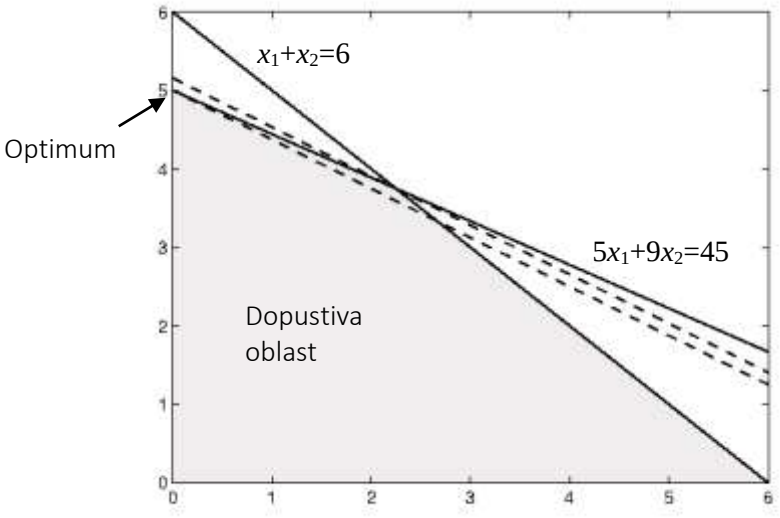
Za demonstraciju izvršenja implementiranog algoritma će biti iskorišten sljedeći primjer:

$$\arg\max Z = 5x_1 + 8x_2$$

sa ograničenjima:

$$x_1 + x_2 \leq 6$$
$$5x_1 + 9x_2 \leq 45$$
$$x_1, x_2 \geq 0$$

Grafički prikaz ovog problema je sljedeći:



Slika 1. Grafički prikaz primjera 1.

U slučaju da je ovo problem linearnog programiranja, rješenje bi bilo u tački (9/4, 15/4). Međutim, globalni optimum je za slučaj problema cjelobrojnog programiranja u tački (0,5). Također, postoji i lokalni optimum u tački (3,3). Algoritam lokalnog pretraživanja bi se zaustavio u ovoj tački.

Koristeći tabu pretraživanje je omogućena zamjena tekućeg rješenja nekim potencijalnim rješenjem koje ne mora nužno poboljšavati funkciju kriterija. Pri tome je, u slučaju da se koristi osnovna varijanta tabu pretrage, potrebno definirati vrijeme trajanja tabua koje će osigurati da se tekuće rješenje ne vrati u lokalni optimum. Ukoliko se odabere 8 iteracija, globalni maksimum će se pronaći nakon 11 iteracija. Broj iteracija potrebnih da se pronađe optimum se može dodatno smanjiti uvođenjem diverzifikacije, pri čemu je prilikom kreiranja objekta klase TabuSearch potrebno proslijediti sljedeće parametre:

```
['diversify', 2, 1, 1]
```

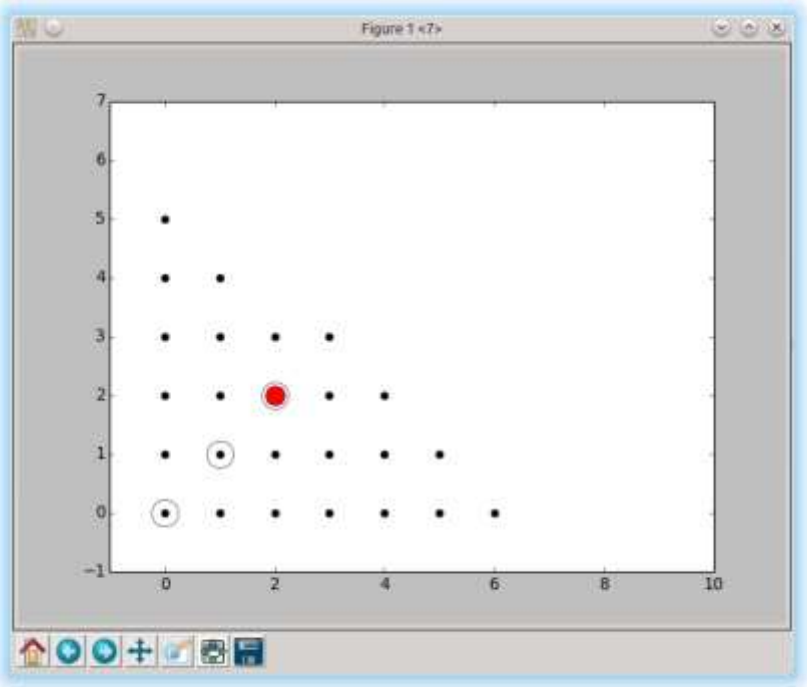
Kao što je prethodno opisano, korištenje diverzifikacije se postiže navođenjem parametra diversify, nakon čega je potrebno navesti nakon koje iteracije će se vršiti diverzifikacija, koliki broj pojavljivanja može imati komponenta kako bi se uzela u razmatranje, kao i broj komponenti koje se diverzificiraju. U ovom primjeru

će nakon druge iteracije započeti diverzifikacije, pri čemu će se u obzir uzeti samo komponente koje su se najviše jedanput pojavile u tekućem rješenju i od svih mogućih komponenti će se odabrati samo jedna. Dakle, s obzirom da je globalni optimum u tački (0,5), a nakon druge iteracije se na prvom mjestu 0 pojavila samo jedanput i u obzir dolazi samo jedna komponenta, to će se fiksirati 0 na prvom mjestu u rješenju i globalni optimum će se dosegnuti nakon 4 iteracije.

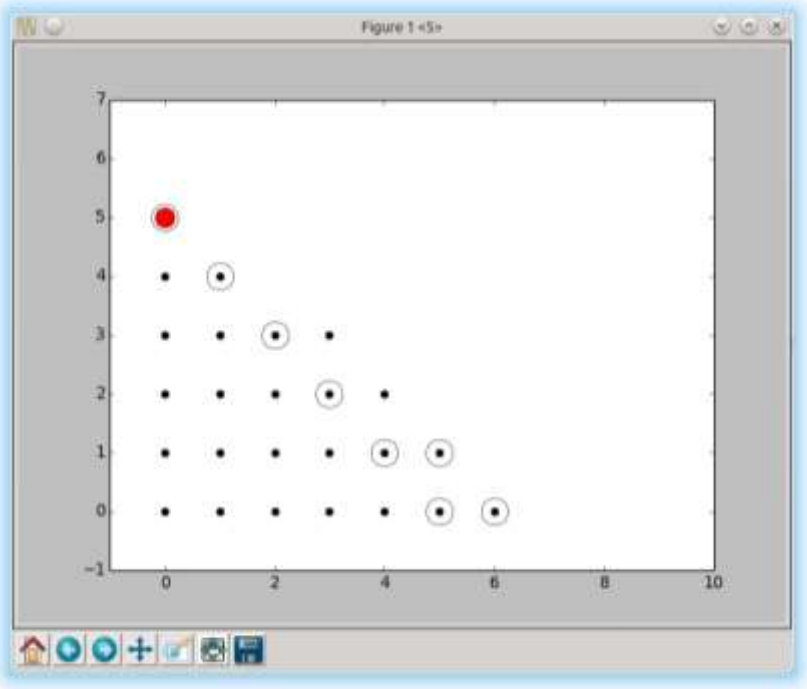
Korištenje intenzifikacije u ovom primjeru nema smisla, s obzirom da se komponente rješenje mijenjaju skoro u svakoj iteraciji, te korištenje srednjoročne memorije ne bi poboljšalo performanse.
S obzirom da je navedeni problem ima dvije dimenzije, to je proces izvršenja algoritma pogodan za vizualizaciju. To se može postići pozivajući plot() funkciju, pri čemu se kao parametar prosljeđuje opseg koordinatnih osa i može se proslijediti vrijeme zadržavanja između prikazivanja susjednih iteracija:

```
while tabu_search.iteration < tabu_search.max_iter:
    tabu_search.next_iteration()
    plt = tabu_search.plot([0, 9, 0, 6], 2, ['delay', 0.5])
```

Nakon treće iteracije se dobije sljedeći prikaz:



a na kraju:



Pri tome je optimum prikazan crvenom tačkom, a tačke koje se nalaze u tabu listi su zaokružene.

5.2. Primjer 2

Problem ranca se može svesti na oblik koji je pogodan za rješavanje implementiranim algoritmom. Neka se količina objekta u rancu označi sa x_j, njegova težina sa w_j, cijena sa c_j i ukupni kapacitet sa z. Tada je funkcija kriterija data izrazom J=sigma c_j * x_j, a ograničenja su oblika sigma w_j * x_j <= z.
Neka je naprimjer dat neograničeni problem ranca, pri čemu su vrijednosti prikazane u sljedećoj tabeli:

j	w_j	c_j
1	2	8
2	5	5
3	1	3
4	4	6
5	3	4

Funkcija kriterija se može u Pythonu definirati kao:

```
def f(x):
    return 8 * x[0] +  5 * x[1] + 3 * x[2] + 6 * x[3] + 4 * x[4]
```

Ograničenja se mogu definirati putem sljedeće funkcije:

```
def c1(x):
    return 2 * x[0] +  5 * x[1] + 1 * x[2] + 4 * x[3] + 3 * x[4] <= 17
```

Potrebno je još osigurati da su rješenja pozitivni brojevi:

```
def c2(x):
    return x[0] >= 0 and x[1] >=0 and x[2] >=0 and x[3] >=0 and x[4] >=0
```

Potom se kreira problem cjelobrojnog programiranja:

```
problem = IntegerProblem('max', f, [0,0,0,0,0], [c1, c2])
```

Rješenje ovog problema je (8,0,1,0,0) i dobije se nakon 16 iteracija. Ukoliko se primijeni intenzifikacija, rješenje se dobije nakon 9 iteracija, u slučaju kada se intenzifikacije počinje vršiti nakon druge iteracije.

Ukoliko se uvedu ograničenja po pitanju količine dostupnih objekata, dati neograničeni problem ranca se transformira u ograničeni. Tada se definira dodatno ograničenje:

```
def c3(x):
      return x[0] <= 4 and  x[1] <= 3 and x[2] <= 4 and x[3] <= 2 and
            x[4] <= 2
```

Rješenje je (4,0,4,1) i dobije se nakon 4. iteracije.

6. Zaključak

Značaj problema cjelobrojnog programiranja je evidentan u nizu primjena. Vođen tom činjenicom, rad je predstavio detaljan opis svih parametara problema cjelobrojnog programiranja, nakon čega je opisan algoritam tabu pretraživanja, sa cilje da se upotrijebi u procesu pronalaska rješenja ovog problema. Nakon osnovnog opisa i opisa potencijalnih poboljšanja algoritma, u smislu optimalnosti rješenja i efikasnosti algoritma, opisana je i konkretna implementacija u Python-u, pri čemu su razmotreni konkretni aspekti implementacije. Na kraju, implementacija je demonstrirana na rješavanju konkretnog problema cjelobrojnog programiranja – tzv. problemu ranca, Pretpohodno opisani problem ranca je jasno pokazao optimalnost i efikasnost predloženog rješenja.

Uzimajući u obzir sve strukturne elemente ovog rada, može se zaključiti da je rad dostigao određene ciljeve. Rad je predstavio konkretnu implementaciju heurističkog algoritma tabu pretraživanja sa elementima koji poboljšavaju efikasnost, te pokazao primjenu tog algoritma na rješavanje širokog skupa problema cjelobrojnog programiranja. Na kraju, može se konstatovati da značaj rada leži u činjenici da je, uzimajući u obzir to da je algoritam tabu pretraživanja relativno nov algoritam, upotreba algoritma tabu pretraživanja u rješavanju problema cjelobrojnog programiranja relativno neistraženo područje i da ovaj rad daje neupitan doprinos tom području.

7. Literatura

1. Jurić, Ž., & Mateljan, T. (2013.). *Radni materijali za kurs "Osnove operacionih istraživanja".* Sarajevo.
2. Konjicija, S. (2015). Tabu pretraživanje. U *Optimizacija resursa.* Sarajevo.
3. Laguna, M., Marti, R., & Campos, V. (1998.). *Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem.*
4. Skorin-Kapov, N. (2014.). Tabu pretraživanje. *Heurističke metode optimizacije.* Zagreb: Zavod za telekomunikacije Sveučilište u Zagrebu.
5. Gendreau, M. (2002.). *An Introduction to Tabu Search.*