Univerzitet u Beogradu
Matematički fakultet

Seminarski rad
Matematičko programiranje i optimizacija

Lokacijski problem sa nadmetanjem

student:
Aida Zolić, 1023/2015

profesor:
dr Zorica Stanimirović

maj 2016.

Sadržaj

Apstrakt

Lokacijski problem sa nadmetanjem predstavlja jednu modifikaciju osnovnog lokacijskog problema dodavanjem uticaja konkurencije na pogodnost neke potencijalne lokacije. Spada u grupu BLP (*Binary Linear Programming*) problema. Iako postoje egzaktne metode za rešavanje ovog tipa problema, u nekim slučajevima su one nepraktične zbog veličine ulaznih podataka. U ovom radu prikazan je pristup rešavanju korišćenjem BCO metaheuristike (*Bee Colony Optimization*), zatim VNS metaheuristike (*Variable Network Search*), kao i njihove hibridizacije. Model oslikava realnu situaciju u nadmetanju za potrošače.
ključne reči: nadmetanje, metaheuristike, BLP, BCO, VNS

1 Lokacijski problem sa nadmetanjem

Lokacijski problem sa nadmentanjem je pod nazivom MAXCAP (*maximum capture problem*) prvi put formulisan u [1], a detaljno razmatran u [3]. Ovaj problem predstavlja pokušaj da se odrede optimalne lokacije za p objekata (snabdevača) firme A ukoliko na tržištu već postoji q konkurentnih objekata neke druge firme B. Cilj je maksimizovati uticaj firme A na tržištu. Pretpostavka je da su lokacije svih uspostavljenih objektata fiksirane i poznate. Jedan čvor predstavlja potrošača sa fiksiranim zahtevima. U svim objektima prodaje se jedan proizvod čija je cena ista bez obzira na vlasnika, tako da smatramo da potrošač bira snabdevača samo na osnovu udaljenosti.

Relaksiranjem nekih uslova iz originalne formulacije problema definisano je još nekoliko verzija. Tako je u [2] obrađen problem MAXRELOG (*maximum capture problem with reallocation*), koji dopušta i realokaciju, a ne samo lokaciju objekata firme A. U [4] razmatran je PMAXCAP (*pricing maximum capture problem*) kada firme A i B imaju različite cene proizvoda, tako da konačna odluka potrošača zavisi i od nje, a ne samo od troškova prevoza.

2 Matematička formulacija MAXCAP problema

Ovde će biti detaljnije opisana samo originalna formulacija data u [1].

Sa tim ciljem uvodimo sledeće parametre:

- I - indeksi lokacija potrošača
- J - indeksi potencijalnih lokacija objekata firme A
- w_i - potreba potrošača i
- P_i - skup indeksa lokacija iz J koje su bliže potrošaču i od njemu najbližeg snabdevača firme B
- T_i - skup indeksa lokacija iz J isto udaljenih od potrošača i koliko i njemu najbliži snabdevač firme B

Promenljive definišemo na sledeći način:

$$y_i = \begin{cases} 1, & \text{ukoliko je potrošač } i \text{ pridružen firmi A} \\ 0, & \text{inače} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{ukoliko je potreba potrošača } i \text{ podeljena među firmama A i B} \\ 0, & \text{inače} \end{cases}$$

$$x_j = \begin{cases} 1, & \text{ukoliko je na lokaciji } j \text{ uspostavljen snabdevač} \\ 0, & \text{inače} \end{cases}$$

$$\max \sum_{i \in I} w_i y_i + \sum_{i \in I} \frac{w_i}{2} z_i \tag{1}$$

$$y_i \leq \sum_{j \in P_i} x_j, \forall i \in I \tag{2}$$

$$z_i \leq \sum_{j \in T_i} x_j, \forall i \in I \tag{3}$$

$$y_i + z_i \leq 1, \forall i \in I \tag{4}$$

$$\sum_{j \in J} x_j = p \tag{5}$$

$$y_i \in \{0, 1\}, \forall i \in I \tag{6}$$

$$z_i \in \{0, 1\}, \forall i \in I \tag{7}$$

$$x_j \in \{0, 1\}, \forall j \in J \tag{8}$$

Funkcija cilja (1) maksimizuje profit firme A, odnosno sumu potražnji onih potrošača koje ona delimično ili potpuno pokriva.

Ograničenje (2) dozvoljava da potrošač i preferira firmu A, tj. da y_i može biti 1, samo ako je firma A uspostavila neki svoj objekat dovoljno blizu, odnosno bliže od bilo kog snabdevača firme B. Slično, zbog ograničenja (3) obe firme imaju podjednak udeo u snabdevanju potrošača i samo ako firma A uspostavi svoj objekat na istoj udaljenosti od potrošača kao što je najbliži objekat firme B. Ograničenje (4) dozvoljava da jednog potrošača i ili u potpunosti snabdeva A ($y_i = 1, z_i = 0$), ili obe firme ($y_i = 0, z_i = 1$), ili samo B ($y_i = 0, z_i = 0$). Ograničenje (5) obezbeđuje da bude uspostavljen traženi broj objekata firme A.

Poslednja tri ograničenja (6), (7) i (8) definišu promenljive kao binarne.

3 Opis instanci

Zbog nemogućnosti pronalaska odgovarajućih instanci za navedeni problem, one su generisane delimičnom transformacijom od instanci za lokacijske probleme sa i bez kapaciteta (*uncapacitated location* i *capacitated location*) i p-centar problem (*phub*) preuzetih sa [12]. Od jedne takve instance dobijeno je više različitih instanci, menjanjem nekih ili svih parametara p, q, $|I|$, $|J|$, kao i skupova P_i i T_i.

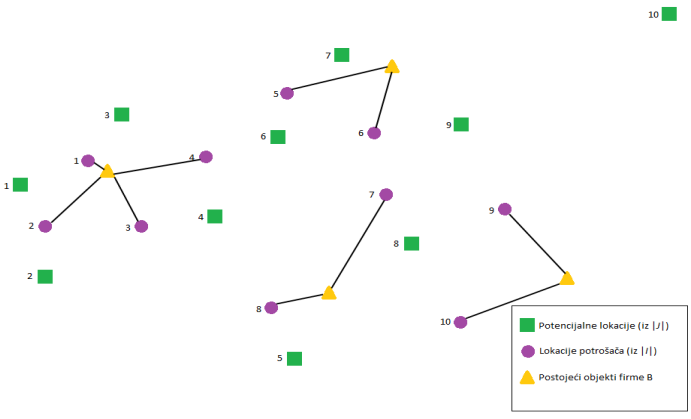
Instance su formatirane tako da su u prvom redu navedeni redom parametri $|I|, |J|, p, q$. U narednom redu su w_i, $i \in I$. Zatim slede redom broj elemenata u P_i i svi indeksi iz P_i, broj elemenata u T_i i svi indeksi iz T_i za svako $i \in I$.

3.1 Primer instance

Sledi primer jedne generisane test instance male dimenzije.

10 10 1 4
5 12 7 10 3 8 2 13 20 14

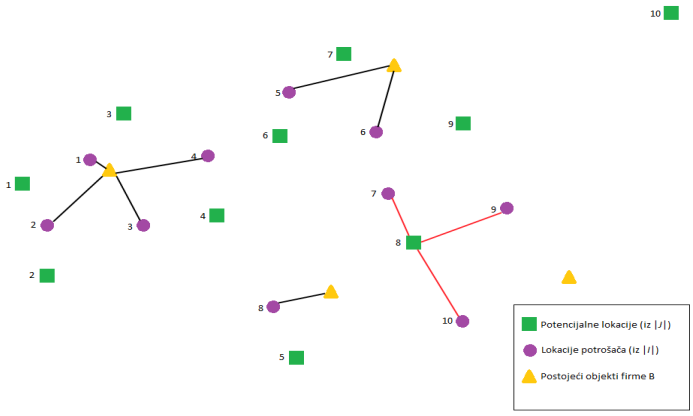
0
0
2 0 1
0
0
1 3
1 5
1 3
3 2 5 6
1 3
1 8
1 5
1 7
0
1 4
0
0
2 7 8
0
1 7



Slika 1: Primer instance pre rešavanja

Na slici 1 prikazana je navedena instanca sa alokacijama potrošača pre ulaska firme A na tržište, odnosno svaki potrošač je povezan linijom sa najbližim objektom firme B. Optimalno rešenje u ovom primeru je 19, koje se dobija za $x_7 = 1$, odakle je $y_6 = 1, z_8 = 1, z_9 = 1$, a ostale promenljive su 0. Ukoliko se uspostavi objekat na lokaciji 8 dolazimo do situacije koja je predstavljena na

slici 2. Neki potrošači su prealocirani na novi objekat i te veze su prikazane crvenom bojom.



Slika 2: Primer instance sa optimalnim rešenjem

4 Opis primenjenih heuristika

U skoro svim oblastima nauke i industrije nailazi se na procese koji se mogu optimizovati. Brojni su realni primeri koji se mogu formulisati kao problemi optimizacije. Međutim, mnogi od njih su previše obimni i komplikovani da bi bili rešivi egzaktnim metodama za razumno vreme. Tada se primenjuju aproksimativne metode, odnosno heuristike i metaheuristike.

4.1 Optimizacija kolonijom pčela

4.1.1 Biološka osnova i opis metode

Optimizacija kolonijom pčela (*Bee Colony Optimization*, BCO) je stohastička P-metaheuristika i spada u metode koje koriste inteligenciju grupe, tj. roja (*Swarm Intelligence*, SI). Postoji više algoritama koji su inspirisani ponašanjem pčela, a ovu metaheuristiku su prvi put predložili Lučić i Teodorović, [6].

Posmatranjem prirodnih pčela ustanovljeno je da prilikom potrage za hranom, neke pčele - izviđači istražuju okolinu košnice, da bi po povratku posebnim plesom ostalim pčelama reklamirale nektar. Brzina i način plesa određuju kvalitet i udaljenost pronađenog nektara. Nakon plesa, neke od neopredeljenih pčela odlučuju da odu po nektar na reklamiranu lokaciju. Po povratku, mogu da ponovo postane neopredeljene, da se vrate na lokaciju ili da pokušaju da regrutuju ostale pčele.

U osnovi ove metode leži ideja da se kreira kolonija veštačkih pčela koja je u stanju da efikasno rešava i teže optimizacione probleme. Inicijalno se svakoj od njih dodeljuje neko rešenje, a u svakom koraku algoritma pčela izvodi let unapred (*forward pass*) i let unazad (*backward pass*). Tokom leta unapred pokušava da popravi svoje rešenje. Tokom leta unazad može ili napustiti svoje rešenje, odnosno postati neopredeljena (*uncommitted*) ili mu ostati lojalna i postati regruter (*loyal*). Parametri su B - broj pčela i NC - broj koraka

tokom leta unapred (broj koraka između razmene informacija između pčela). Pored parametara, potrebno je odrediti i kriterijum zaustavljanja. To može biti ukupan broj koraka, određeno vreme izvršavanja, dovoljan broj poklapanja rešenja... Pravilo ruleta koje se koristi u ovoj metodi se može predstaviti na sledeći način: recimo da treba slučajno izabrati jedan element iz nekog konačnog skupa $S = \{a_1, a_2, ..., a_n\}$, i da svaki od njih ima određenu vrednost v_i, $i = 1, ..., n$. Tada smatramo da element a_i treba da bude izabran sa verovatnoćom $p_i = \frac{v_i}{\sum_{i=1}^{n} v_i}$. Vrednosti v_i se određuju u skladu sa konkretnom primenom.

Ova metoda se može implementirati kao konstruktivna, kada pčele u svakom letu unapred dograđuju rešenje, a počinju od praznog. Druga varijanta, koja je primenjena u radu, je da pčele na početku dobiju kompletno rešenje ali koje u svakoj iteraciji menjaju u pokušaju da nađu bolje (*Improving*, BCOi), [7].

Sledi opšti pseudokod algoritma BCO, tj. BCOi. Neki od koraka, poput inicijalizacije, procene koraka i evaluacije rešenja zavise od problema tako da se moraju prilagoditi konkretnoj primeni. Sa druge strane, postoje formule koje se koriste za izračunavanje lojalnosti i izbor regrutera, [7] i [8]. Glavna razlika između BCO i BCOi to što se u konstruktivnoj varijanti počinje od praznog rešenja koje se dograđuje, dok se pri BCOi sve vreme koristi kompletno rešenje, ali koraci su ipak analogni pa je algoritam za oba pristupa objedinjen u jedan.

begin
// Inicijalizacija
for $b = 1$ **to** B
  Pčeli b dodeliti (prazno) početno rešenje.
STOP = 0
while (STOP == 0)
  // Let unapred
  for $b = 1$ **to** B
    for $k = 1$ **to** NC
      Proceniti sve korake i izabrati naredni korak pravilom ruleta.
  // Let unazad
  for $b = 1$ **to** B
    Odrediti kvalitet (parcijalnog) rešenja.
    Pravilom ruleta odlučiti da li je pčela lojalna ili neopredeljena.
    if (pčela b neopredeljena)
      Pravilom ruleta izabrati regrutera.
  Proceniti sva rešenja i izabrati najbolje.
  Ako je ispunjen uslov zaustavljanja, STOP = 1.
Vratiti korisniku najbolje pronađeno rešenje.
end

Kao što je rečeno, neki koraci algoritma nisu zavisni od konkretnog problema. Tako se određivanje verovatnoće da pčela ostane lojalna svom rešenju nakon u koraka unapred vrši po formuli:

$$p_b^{u+1} = e^{\frac{o_b - o_{max}}{u}}$$

gde o_b predstavlja normalizovanu vrednost (kvalitet) rešenja pčele b, a o_{max} maksimalnu od tih vrednosti. Primetimo da verovatnoća lojanosti raste sa

brojem izvršenih letova unapred. Ako sa f_b označimo vrednost funkcije cilja koju je pčela b izračunala u posmatranoj iteraciji, a f_{min} i f_{max} redom minimalna i maksimalna vrednost funkcije cilja svih pčela u toj iteraciji, onda se normalizacija rešenja za problem maksimizacije izvodi na sledeći način:

$$o_b = \begin{cases} \frac{f_b - f_{max}}{f_{max} - f_{min}}, & \text{ako je } f_{max} \neq f_{min} \\ 1, & \text{ako je } f_{max} = f_{min} \end{cases}$$

Konačno, za svaku neopredeljenu pčelu treba odrediti regrutera čije će rešenje preuzeti. Izbor se vrši pravilom ruleta, tako da je za svaku pčelu koja ostane pri svom rešenju potrebno izračunati verovatnoću da ona bude izabrana. Ako je $R \subseteq B$ skup indeksa lojalnih pčela (regurtera), tražena verovatnoća je:

$$p_r = \frac{o_r}{\sum_{b \in R} o_b}$$

4.1.2 Implementacija i primena na MAXCAP problem

Metoda je implementirana u programskom jeziku C++. Napisana je posebna klasa koja predstavlja pčelu i koja je zadužena za korake koje pčele obavljaju nezavisno jedna od druge (let unapred, izračunavanje verovatnoća lojalnosti i regrutovanja itd.). Rešenje je predstavljeno vektorom $x = (x_0, x_1, ..., x_{|J|-1})$, gde je značenje promenljive x_j kao u definiciji problema. Odtale se mogu izračunati vrednosti ostalih promenljivih y_i i z_i kao i ciljne funkcije. Ipak, kako bi račun bio jednostavniji i kod čitljiviji, svaka pčela čuva informacije o još dva vektora $y = (y_0, ..., y_{|I|-1})$ i $z = (z_0, ..., z_{|I|-1})$, gde je $y_i = |\{j : x_j = 1, j \in P_i\}|$ i $z_i = |\{j : x_j = 1, j \in T_i\}|$, što se razlikuje od promenljivih iz definicije problema u kojoj su te promenljive binarne. Kako se primenjuje se BCOi algoritam, incijalno se svakoj pčeli dodeljuje slučajno generisano kompletno rešenje, tj. neka permutacija vektora koji čine p jedinica i $|J| - p$ nula, a koja je dobijena korišćenjem ugrađene funkcije `random_shuffle`. Određivanje vrednosti funkcije cilja je onda jednostavno:

begin
  $f = 0$;
  for ($i = 0$; $i < |I|$; i++)
    if ($y_i > 0$)
      $f = f + w_i$;
    else
      if ($z_i > 0$)
        $f = f + \frac{w_i}{2}$;
end

Za kriterijum zaustavljanja uzet je broj iteracija bez poboljšanja. Konkretno, testiranja su obavljana sa najviše 20 takvih iteracija. Broj veštačkih pčela je postavljen na $B = 10$. Ako za primer uzmemo instancu navedenu u 3.1, incijalizacija (za jednu pčelu) bi mogla biti $x = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$, odakle je $y = (0, 0, 0, 1, 1, 0, 0, 0, 0, 0)$ i $z = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$, a funkcija cilja je tada $f = 10 + 3 + \frac{8}{2} = 17$.

Nakon inicijalizacije, pčele naizmenično primenjuju let unapred i let unazad. Let unapred podrazumeva izbor slučajnog rešenja na način analogan izboru

početnog rešenja. Dopušteno je da novo rešenje može biti i lošije od trenutnog. U svakom trenutku je za svaku pčelu poznato do tada najbolje pronađeno rešenje, odnosno vrednost ciljne funkcije. Nakon što sve pčele obave let unapred, otpočinju let unazad koji obuhvata procenu pronađenog rešenja i određivanje verovatnoće lojalnosti i regrutovanja (za lojalne pčele), odnosno izbora regrutera (za neopredeljene pčele) na gore navedeni način.

Posle svake iteracije ažurira se globalno najbolje rešenje (vektor x_{best} i vrednost funkcije cilja f_{best}). Takođe, proverava se da li je možda ispunjen uslov zaustavljanja i ako nije prelazi se na sledeću iteraciju. Ukoliko je uslov zadovoljen, pronađeno rešenje se prijavljuje korisniku kao optimalno.

```
begin
iter = 0;
while iter < maxIter
  popravljeno = false;
  // Let unapred i unazad kao u sekciji 4.1.1
  for b in B
    for (k = 1; k < NC; k++)
      b.letUnapred;
      b.letUnazad;
      if (b.popravljeno)
        popravljeno = true;
      if (b.vrednost > f_best)
        f_best = b.vrednost;
        x_best = b.x;
  if (popravljeno == false)
    iter = iter + 1;
end
```

4.2 Metoda promenljivih okolina

Metodu promenljivih okolina (*Variable Neighborhood Search*, VNS) predložili su Hansen i Mladenović još 1997, u [9]. U njenoj osnovi leži lokalna pretraga (*Local search*) ali uz sistematske promene okoline radi veće efikasnosti. Zasniva se na tri činjenice:

1. Lokalni optimum u odnosu na jednu okolinu ne mora biti optimum u odnosu na neku drugu;
2. Lokalni optimum ne mora biti i globalni;
3. Za većinu problema lokalni optimumi u odnosu na razne okoline su relativno blizu.

Ove tri činjenice se mogu koristiti deterministički, stohastički ili kombinovano. Okoline se mogu menjati promenom metrike ili promenom veličine. U osnovnoj varijanti, VNS ima jedan parametar k_{max} - broj okolina. U svakom koraku poznati su najbolje pronađeno rešenje x_{best}, slučajno izabrano x' i lokalno najbolje x''. Osnovni koraci metode su sledeći:

- Nalaženje početnog rešenja.

- Lokalna popravka početnog rešenja.
- Faza razmrdavanja (*shaking phase*) - slučajno pomeranje u tekućoj okolinu do nekog (možda goreg) rešenja.
- Faza lokalne pretrage (*local search phase*) - popravljanje rešenja.
- Pomeranje - ukoliko je pomoću LS pronađeno bolje lokalno rešenje, ostaje se pri njemu, inače se menja okolina za razmrdavanje.

Nemaju sve varijante VNS-a sve faze. Na primer, metoda promenljivog spusta (*Variable Neighborhood Descent*, VND) nema fazu razmrdavanja i potpuno je deterministička. Sastoji se u izboru k_{max} sistematski uređenih okolina, N_1, N_2, ..., $N_{k_{max}}$. Primenjuje se LS u svakoj od okolina, a pri svakom poboljšanju vraća se u prvu okolinu. Staje se kada u poslednjoj okolini više nema poboljšanja.

begin
  //Inicijalizacija - Izabrati početno rešenje x, $x_{best} = x$, $k = 1$;
  while $(k \leq k_{max})$
    Primeniti LS na x u okolini N_k, neka je dobijeni lokalni optimum x'.
    if (x' bolje od x_{best})
      $x_{best} = x'$;
      $k = 1$;
    else
      $k = k + 1$.
end

Redukovana metoda promenljivih okolina (*Reduced Variable Neighborhood Descent*, RVNS) je pak u potpunosti stohastička i nema fazu lokalne pretrage. Sastoji se u izboru k_{max} okolina, $N_1, N_2, ..., N_{k_{max}}$ i sistematskom kretanju kroz njih. Bira se jedno slučajno rešenje u trenutnoj okolini i ukoliko donosi poboljšanje prelazi se u njega i kreće ponovo od N_1, inače se prelazi u sledeću okolinu.

begin
  //Inicijalizacija - Izabrati početno rešenje x, $x_{best} = x$, $k = 1$.
  while $(k \leq k_{max})$
    Izabrati slučajno rešenje x' u okolini N_k.
    if (x' bolje od x_{best})
      $x_{best} = x'$;
      $k = 1$;
    else
      $k = k + 1$;
end

RVNS može biti jako korisna za probleme velikih dimenzija zbog uštede koja se dobija preskakanjem LS faze. Međutim, bolje rezultate daje u kombinaciji sa još nekom varijantom. Kombinovanjem VND i RVNS dobija se (osnovna) metoda promenljivih okolina, VNS. Sastoji se u sistematskoj promeni okolina, slučajnom izboru rešenja u tekućoj okolini i loklanoj pretrazi kojom se ono popravlja. Prilikom lokalne pretrage može se preći u prvo nađeno bolje rešenje

(*first improvement*) ili u najbolje rešenje iz okoline (*best improvement*). Ako se u $N_{k_{max}}$ ne može naći bolje rešenje ili se ispuni neki kriterijum zaustavljanja, staje se sa izvršavanjem. Kriterijum zaustavljanja može biti na primer maksimalan broj iteracija, broj iteracija između dva poboljšanja, procesorsko vreme, ili kombinacija navedenih. Okoline se mogu razlikovati po veličini (odnosno broju transformacija nad rešenjem) ili metrici (odnosno načinu transformacije rešenja). Dodatno, definicija okolina koje se koriste za razmrdavanje i okolina za lokalnu pretragu se mogu, ali ne moraju poklapati.

begin
  //Inicijalizacija - Izabrati početno rešenje x, $x_{best} = x$, $k = 1$.
  while $(k \leq k_{max})$
    Izabrati slučajno rešenje x' u okolini N_k.
    Primeniti LS na x' u okolini N_k, neka je dobijeni lokalni optimum x''.
    if (x'' bolje od x_{best})
      $x_{best} = x''$;
      $k = 1$;
    else
      $k = k + 1$;
end

Još neke varijante koje manje ili više odstupaju od osnovne ideje su Metoda promenljivih okolina sa dekompozicijom (*Variable Neighbourhood Decomposition Search*, VNDS), Adaptivna metoda promenljivih okolina (*Skewed VNS*), Metoda promenljivih formulacija (*Variable Neighborhood Formulation Space Search*), Primal-dual VNS itd.

4.2.1 Implementacija i primena na MAXCAP problem

Programski jezik u kom je metoda implementirana je C++. Analogno kao za BCO metaheuristiku, rešenje je i pri implementaciji VNS metode zapisano u vidu vektora binarnih vrednosti $x = (x_1, x_2, ..., x_{|J|-1})$, gde je $x_j = 1$ ako i samo ako je objekat na poziciji j uspostavljen. Generisanje početnog rešenja se svodi na generisanje vektora sastavljnog od p jedinica i $|J| - p$ nula, na primer preko funkcije `rand`:

begin
  // Inicijalizovati $x = (1, ..., 1, 0, ..., 0)$, $|x| = |J|$
  $t = 0$;
  while $(t < p)$
    $i =$rand()%$|J|$;
    if $(x_i == 0)$
      $x_i = 1$;
      $t = t + 1$;
end

Ostale promenljive y_i i z_i, $i \in I$ se lako mogu izračunati na osnovu x i vrednosti w_i, P_i i T_i koje su prosleđene kao ulazni parametri na način koji je opisan u 3, ukoliko bi to bilo potrebno. Vrednosti koje se odnose na skupove P_i

i T_i za $i \in I$ su interno čuvane u matrici $PT = [p_{ij}]$ dimenzija $|I| \times |J|$, gde važi

$$p[i,j] = \begin{cases} 2, & \text{ako } j \in T_i \\ 1, & \text{ako } j \in P_i \\ 0, & \text{inače} \end{cases}$$

Sledeći pseudokod ilustruje kako pronađu preostale vrednosti promenljivih.

begin
for ($i = 0$; $i < |I|$; i++)
  if (postoji j takav da je $x_j = 1$ i $p_{ij} = 1$)
    $y_i = 1$;
  else if (postoji j takav da je $x_j = 1$ i $p_{ij} = 2$)
      $z_i = 1$;
end

Ispitivanje egzistencije traženog indeksa j se može uraditi iterativno, dakle prolaskom kroz sve vrednosti p_{ij}, za fiksirano i. Druga opcija je iskoristiti ugrađene funkcionalnosti samog programskog jezika.

Međutim, u ovom radu relevantna je bila vrednost funkcije cilja, a ne i indeksi objekata koje treba uspostaviti kako bi se ta vrednost postigla. Imajuću to u vidu, računata je samo ciljna funkcija. Ovo se takođe može obaviti jednostavno koristeći ulazne parametare i vektor x, recimo kao što je opisano pseudokodom koji sledi.

begin
$f = 0$;
for ($i = 0$; $i < |I|$; i++)
  $m = 0$;
  $j = 0$;
  while ($m \neq 1$ **AND** $j < J$)
    if (x_j == **true AND** $p_{ij} \neq 0$)
      $m = \max(m, p_{ij}^{-1})$;
    $j = j + 1$;
  $f = f + m \cdot w_i$;
end

U implementaciji VNS metode za MAXCAP kroz okoline za razmrdavanje prolazi se od najmanje do najveće, dakle metrika je uvek ista. Broj različitih okolina je ograničen parametrom metode $N_{k_{max}}$ koji je postavljen na 50 u svim testovima. Za definiciju okoline N_k, u skladu sa opisanim zapisom rešenja preko vektora x, uzeto je da označava da se od rešenja x_1 dobije $x_2 \in N_k$ promenom nekih k jedinica. Zbog toga je uvedeno dodatno ograničenje da najveća okolina zapravo podrazumeva promenu najviše p jedinica. Dakle, za jednu konkretnu instancu važi da je parametar metode koji predstavlja broj okolina zapravo jednak $N_{k_{max}} = \min(p, 50)$. Implementacija funkcije za razmrdavanje u okolini N_k rešenja se može predstaviti ovako:

begin
  // Neka je J_1 skup onih indeksa $j = 0, ..., |J| - 1$ za koje je $x_j = 1$,

```
    // a J_0 skup onih indeksa j = 0, ..., |J| − 1 za koje je x_j = 0
    Primeniti random_shuffle na oba skupa;
    for (j = 0; j < k; j++)
      x[J_0[j]] = 1;
      x[J_1[j]] = 0;
end
```

Razdvajanje indeksa na dva navedena skupa (preciznije, korišćeni su vektori) se obavlja u jednom prolazu kroz x, a zatim se permutovanjem redosleda indeksa, bez gubljenja opštosti mogu uzeti prvih k elementa iz oba i na odgovarajućim pozicijama u vektoru x zameniti 0 u 1, odnosno 1 u 0.

Okolina za lokalnu pretragu konstruisana je potpuno analogno kao okoline za razmrdavanje. Razlikuju se po tome što je ona fiksirana tako da je uvek u pitanju okolina N_1, odnosno promena samo jedne jedinice. Lokalna pretraga je implementirana tako da staje nakon pronalaska prvog boljeg rešenja (*first improvement*). Funkcija koja obavlja lokalnu pretragu uz to vraća logičku vrednost **tačno** ako uspešno pronadje bolje rešenje od tekućeg, a inače **netačno**. Na osnovu toga se određuje da li je sledeći korak povećavanje okoline ili razmrdavanje.

```
begin
x' = x
for (j = 0; j < |J|; j++)
  if (x'_j == 0)
    continue;
  x'_j = 0;
  for (k = 0; k < |J|; k++)
    if (k ≠ j AND x'_k == 0)
      x'_k = 1;
      if (f(x') > f(x))
        x = x';
        return true;
      x'_k = 0;
  x'_j = 1;
return false;
end
```

Uzmimo ponovo test instancu iz 3.1 za primer. Ako bi početno rešenje bilo $x = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$, $f = 17$, u fazi razmrdavanja bi onda moglo biti izabrano $x' = (0, 1, 0, 0, 0, 0, 0, 0, 0, 0)$, $f = 12$, a lokalna pretraga bi se zaustavila na prvom boljem rešenju od x' što je $x'' = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)$, za koje je $f = 19$ i zapravo je optimalno. Algoritam staje nakon 20 iteracija bez uočene popravke rezultata, zbog čega se nakon svakog koraka ispituje da li je rešenje popravljeno, i ako nije brojač se povećava sve dok se ne dostigne nametnuto ograničenje.

4.3 Hibridna metaheuristika

Sa ciljem rešavanja određenih nedostataka uočenih u primeni na razmatrani MAXCAP problem, konstruisana je nova metoda koja kombinuje navedene dve heuristike. Neke prvobitne odluke su tom prilikom modifikovane. Baza

novonastale meteheuristike je BCO, a uveden je dodatni parametar α koji predstavlja verovatnoću da se pčela odluči da tokom leta unapred primeni VNS. Ukoliko je to slučaj, pčela pokušava sistematično da popravi svoje rešenje. Međutim, obavlja samo jednu iteraciju, tj. u trenutku kada se stigne do poslednje (najveće) okoline za razmrdavanje, korak se završava bez obzira da li je pronađeno bolje rešenje ili ne. Inače se ponaša kao u originalnoj BCO, dakle menja tekuće rešenje nekim slučajno izabranim, potencijalno lošijim. Nakon što sve pčele obave NC letova unapred, nastavlja se poptpuno analogno kao u BCO (procenjuju se sva rešenja, lojalnost i tako dalje).

Ukoliko se parametar α postavi na 0 metoda prelazi u BCO. Ukoliko se postavi na 1, dobija se BCO metoda u kojoj sve pčele uvek primenjuju VNS, čime se umanjuje slučajnost koja odlikuje BCO ali i njena efikasnost.

Druga varijanta kombinovanja BCO i VNS bi mogla biti da se, umesto α uvede parametar koji predstavlja broj pčela koje uvek primenjuju VNS u letu unapred, dok se ostale ponašaju na uobičajen način. Odluka da se iskoristi prvi pristup ima svoje opravdanje. Naime, ukoliko se koristi gore opisani parametar α, tako definisana verovatnoća se u toku rada može menjati, slično kao što recimo verovatnoća da pčela ostane lojalna svom rešenju raste tokom vremena. U ovoj inicijalnoj verziji metode α je fiksiran tokom celog izvršavanja programa, a pronalaženje i ispitivanje različitih funkcija koje bi se na taj parametar mogle primeniti ostaje za buduća istraživanja.

Navedimo pseudokod koji opisuje nastalu hibridnu metaheuristiku.

```
begin
  // Inicijalizacija
  for b = 1 to B
    Pčeli b dodeliti početno rešenje;
  STOP = 0;
  while (STOP == 0)
    // Let unapred
    for b = 1 to B
      for k = 1 to NC
        Izabrati metodu za naredni korak i primeniti je.
        if (BCO)
          // Primeniti korake opisane u sekciji 4.1.1 algoritma BCO
          Proceniti korake pa izabrati naredni pravilom ruleta.
        else // Ukoliko je u pitanju VNS
          Popraviti tekuće rešenje jednim prolazom kroz (sve) okoline.
    // Let unazad
    for b = 1 to B
      Odrediti kvalitet rešenja.
      Pravilom ruleta odlučiti da li je pčela lojalna ili neopredeljena.
      if (pčela b neopredeljena)
        Pravilom ruleta izabrati regrutera.
  Proceniti sva rešenja i izabrati najbolje.
  Ako je ispunjen uslov zaustavljanja STOP = 1.
Vratiti korisniku najbolje pronađeno rešenje.
end
```

4.3.1 Implementacija i primena na MAXCAP problem

Iako značajan deo hibridne heuristike nije izmenjen u odnosu na originalne metode, neke prvobitne odluke vezane za implementaciju su promenjene pod uticajem rezultata dobijenih pri izvršenim testiranjima. Radi jednostavnijeg računa i razumljivijeg dizajna, prilikom implementacije napravljena je nova pomoćna klasa koja predstavlja rešenje, ali je interno ono i dalje kodirano vektorom odgovarajuće dužine $x = (x_1, ..., x_{|J|-1})$. Pored vektora x, rešenje ima informaciju o vektorima y i z, gde je $y_i = |\{j \in P_i : x_j = 1\}|$ odnosno $z_i = |\{j \in T_i : x_j = 1\}|$. U ovoj klasi enkapsulirano je izračunavanje funkcije cilja, procena izmene vektora x, izvršavanje faze razmrdavanja, transformacije u slučajno izabrano rešenje... Dalje, osim matrice PT analogne opisanoj u sekciji 4.2.1, klasa rešenje dodatno čuva informacije i o skupovima $P'_j = \{i \in I : j \in P_i\}$ i $T'_j = \{i \in I : j \in T_i\}$. Klasa koja predstavlja pčelu je skoro potpuno preuzeta iz implementacije BCO uz dodatak mogućnosti primene VNS metode, ali kao što je navedeno uz jedan prolaz kroz sve okoline. Još jedna izmena je što je za hibrid je uzeta varijanta VNS gde lokalna pretraga pronalazi najbolje rešenje iz okoline, a ne prvo bolje (*best improvement*).

Broj pčela je postavljen na $B = 5$. Pri kreiranju svake od pčela dodeljuje joj se slučajno generisano početno rešenje primenom funkcije `random_shuffle` na $x = (1, 1, ..., 1, 0, 0, ..., 0)$, gde je prvih p elemenata postavljeno na 1, a svi ostali na 0. Odatle se mogu odrediti y i z. Inicijalizacija rešenja je onda:

begin
// *Inicijalizovati* $x = (1, ...1, 0, ..., 0)$, $|x| = |J|$
// *Inicijalizovati* $y = (0, 0, ..., 0)$, $|y| = |I|$
// *Inicijalizovati* $z = (0, 0, ..., 0)$, $|z| = |I|$
Primeniti `random_shuffle` na x.
for ($j = 0$; $j < |J|$; j++)
  if ($x_j == 1$)
    for ($i \in P'_j$)
      $y_i = y_i + 1$;
    for ($i \in T'_j$)
      $z_i = z_i + 1$;
end

Prilikom leta unapred, izbor metode koju pčela primenjuje simuliran je generisanjem slučajnog broja iz segmenta $[0, 1]$ i poređenjem sa parametrom $\alpha = 0.3$. Okoline su definisane potpuno isto kao u sekciji 4.2.1. Broj okolina je pak smanjen na 5, pa je $k_{max} = \min\{p, 5\}$. Dakle, let unapred je funkcija koja izgleda ovako:

begin
$r =$ `rand()`%RAND_MAX;
if ($r < \alpha$)
  // *Primeniti VNS*
  for ($k = 1$; $k \leq k_{max}$; k++)
    Neka je razmrdavanjem dobijeno x'.
    Primeniti LS na x' u okolini N_k.
    if (LS-om pronađeno x'' je bolje od x)

$x = x'$;
return;
// Kada je $k = k_{max}$, a nema poboljšanja ne vraća se u N_1, staje se.
else
Primeniti iteraciju BCO (bez izmena, kao u 4.1.1).
end

Navedimo još pseudokod izmenjene lokalne pretrage koja podrazumeva pronalazak najboljeg rešenja iz okoline N_1 rešenja x.

begin
// Neka je J_1 skup onih indeksa $j = 0, ..., |J| - 1$ za koje je $x_j = 1$,
// a J_0 skup onih indeksa $j = 0, ..., |J| - 1$ za koje je $x_j = 0$
$max = 0$;
$x' = x$;

for $(i \in J_1)$
for $(j \in J_0)$
// Proceniti zamenu objekta sa indeksom i objektom sa indeksom j.
Neka je $x'' = x$, osim $x''_i = 0$ i $x''_j = 1$;
$d = f(x'') - f(x)$;
if $(d > max)$
$max = d$;
$x' = x''$;
if $(max==0)$ *// Nema boljeg rešenja u okolini.*
return false;
else
$x = x'$; *// Prelazi se u najbolje rešenje iz okoline.*
return true;
end

Demonstrirajmo rad hibrida takođe na primeru instance iz sekcije 3.1. Na primer, jedna pčela bi mogla krenuti iz rešenja $x = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$, $f = 17$, zatim birajući slučajan broj iz $[0, 1] > \alpha$ odlučiti da primeni BCO i time pređe u rešenje $x = (0, 0, 0, 1, 0, 0, 0, 0, 0, 0)$, $f = 10$ i recimo ostane lojalna. Neka u sledećem koraku primeni VNS i u fazi razmrdavanja na primer prelazi u rešenje $x' = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$, $f = 3$. U fazi lokalne pretrage, kako se traži najbolje rešenje iz okoline i u primeru je $p = 1$ svakako bi morala doći do optimalnog rešenja $x'' = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0)$, $f = 19$. Kriterijum zaustavljanja je izabran da bude maksimalan broj koraka bez poboljšanja i postavljen je na 200.

5 Rezultati

Svaka instanca je testirana 10 puta. U tabelama koje slede su navedeni ulazni parametri $|I|$, $|J|$ i p, optimalno rešenje dobijeno pomoću CPLEX paketa (opt), najbolje rešenje koje je dobijeno heuristikom (sol_{best}), prosečno vreme do prvog pojavljivanja najboljeg rešenja (t_{best}), prosečno ukupno vreme (t_{tot}), prosečno odstupanje ($agap$) u procentima, standardna devijacija (σ) u procentima kao i ušteda ostvarena keširanjem izračunatih vrednosti funkcije cilja ($cache$) u procentima. Ako je $sol_i, i = 1, ..., 10$ rešenje u i-tom izvršavanju, tada se procentualno odstupanje od najboljeg rešenja $sol_{best} = \max(sol_i), i = 1, ..., 10$ računa po formuli $gap_i = 100\frac{|sol_i - sol_{best}|}{|sol_best|}$. Odatle dalje možemo izračunati prosečno odstupanje po formuli $agap = \frac{\sum_{i=1}^{10} gap_i}{10}$, kao i standardnu devijaciju $\sigma = \sqrt{\frac{1}{10}\sum_{i=1}^{10}(gap_i - agap)^2}$.

Testiranja su izvođena na Intel(R) Core i5 procesoru pod Windows 8.1 operativnim sistemom. Korišćena je verzija 12.1 CPLEX paketa uz vremensko ograničenje od 4 sata. Heuristike su implementirane u programskom jeziku C++, uz ograničenje broja iteracija bez popravljanja rešenja.

5.1 Rezultati dobijeni primenom BCO metode

Rezultati ove metode se mogu videti u tabeli 5.1.1. Parametri metode su $B = 10, NC = 10, MaxIter = 20$. Korišćene su četiri instance sa parametrima $|I| = 1000, |J| = 100$, $|I| = 3000, |J = 100$, $|I| = 3000, |J| = 300$ odnosno $|I| = 10000, |J| = 1000$ formirane na način opisan u 3, za koje je zatim variran parametar p.

Tabela 5.1.1: Rezultati BCO metode

ULAZ			CPLEX	BCO					
$\|I\|$	$\|J\|$	p	opt	sol_{best}	t_{best}(s)	$t_{tot}(s)$	agap(%)	σ(%)	cache(%)
1000	100	1	3070.5	3070.5	0.3956	4.3910	0.0000	0.00	99.05
1000	100	5	12402.5	11450	2.1567	5.0828	5.0751	3.50	0.00
1000	100	10	21554.5	18627	3.9020	5.8833	4.3673	2.16	0.00
3000	100	1	6979.5	6979.5	0.5890	3.8872	0.0000	0.00	98.81
3000	100	3	19318.5	18699.5	3.6409	9.5483	4.5279	2.16	4.29
3000	100	5	29698.5	28029	2.9861	7.4065	3.2500	1.48	0.00
3000	100	10	52983.5	47678.5	4.1353	7.9395	2.2192	1.61	0.00
3000	100	20	86302.5	75680.5	4.0958	7.1350	1.2666	0.66	0.00
3000	100	30	105817	96649	3.7995	7.9856	2.8618	1.44	0.00
3000	100	40	117687	108630.5	5.2651	8.1552	1.1746	0.83	0.00
3000	100	50	125856	118272	3.6670	7.7000	1.1895	0.53	0.00
3000	300	1	14986.5	14986.5	4.0245	16.4799	0.5905	1.77	97.41
3000	300	2	27607	27607	10.9058	24.3277	3.9774	2.25	11.87
3000	300	3	39184	37007.5	6.6992	20.3287	2.2733	1.56	0.21
3000	300	4	48064	46426	7.3476	14.3342	2.6681	1.82	0.00
3000	300	5	-	53755	6.7677	17.1433	2.1947	1.50	0.00
3000	300	6	-	60738	6.0606	15.4285	2.0765	1.29	0.00
3000	300	7	-	66617.5	9.3909	17.8234	1.5964	0.89	0.00
3000	300	8	-	71214	7.2809	15.5805	1.1411	1.00	0.00
3000	300	9	-	77538.5	6.4628	16.3929	1.8746	1.41	0.00
3000	300	10	88220	82087.5	4.4617	12.2790	3.5750	1.37	0.00
10000	1000	10	-	13642116.5	11.6359	22.0768	2.4169	1.36	0.00
10000	1000	20	-	25079732	6.4874	17.3393	1.1022	0.68	0.00
10000	1000	30	-	35929262.5	6.6471	13.0340	1.3230	0.52	0.00
10000	1000	40	-	45274068.5	5.3599	11.3026	0.6492	0.37	0.00
10000	1000	50	-	54809865.5	9.4894	14.9536	1.1192	0.59	0.00
10000	1000	60	-	63087858.5	7.3304	14.9342	1.1030	0.53	0.00
10000	1000	70	-	70711177	5.7502	11.8050	0.9359	0.58	0.00
10000	1000	80	-	77527909.5	6.3780	12.0703	0.6643	0.34	0.00
10000	1000	90	-	83990746	8.5727	15.3675	0.2321	0.15	0.00
10000	1000	100	-	90786096.5	8.2727	14.0704	1.0370	0.39	0.00

5.2 Rezultati dobijeni primenom VNS metode

Rezultati VNS metode se mogu videti u tabeli 5.2.1. Za vrednost jedinog parametra metode je izabrano $k_{max} = \min\{p, 50\}$. Ispostavilo se da takav izbor parametra i implementacija nisu optimalni, tako da su testirane samo tri instance manjih dimenzija, dakle sa parametrima $|I| = 1000, |J| = 100$, $|I| = 3000, |J = 100$ i $|I| = 3000, |J| = 300$ formirane na način opisan u 3, za koje je zatim variran parametar p. Dopušteno je $MaxIter = 20$ iteracija bez poboljšanja.

Tabela 5.2.1: Rezultati VNS metode

ULAZ			CPLEX	VNS					
$\|I\|$	$\|J\|$	p	opt	sol_{best}	t_{best}(s)	$t_{tot}(s)$	agap(%)	σ(%)	cache(%)
1000	100	1	3070.5	3070.5	2.3009	2.9982	2.0681	4.14	96.19
1000	100	5	12402.5	12216.5	44.9673	100.1697	14.54	5.83	2.12
1000	100	10	21554.5	19265.5	87.3155	163.2805	7.7831	6.46	0.28
3000	100	1	6979.5	6979.5	3.4016	4.2896	3.4472	2.81	95.92
3000	100	3	19318.5	13519.5	36.9624	77.9660	7.1046	4.33	49.54
3000	100	5	29698.5	29043.5	109.4099	193.5759	7.6730	4.17	1.67
3000	100	10	52983.5	48578	100.1143	190.5651	4.4611	2.73	0.20
3000	100	20	86302.5	77969	110.0134	236.8890	4.1967	3.40	0.20
3000	100	30	105817	97472	258.0445	406.4882	2.3502	1.71	0.06
3000	100	40	117687	109640.5	200.9912	384.5252	2.7413	1.64	0.09
3000	100	50	125856	120657.5	323.6848	453.4979	2.5723	1.34	0.10
3000	300	1	14986.5	14986.5	42.9799	47.4328	5.5143	4.03	82.91
3000	300	2	27607	26479	166.7249	261.1927	3.4999	3.95	16.95
3000	300	3	39184	37846	202.3015	726.1096	7.2596	4.14	2.54
3000	300	4	48064	44392	349.3017	735.7316	5.096	3.60	0.57
3000	300	5	-	53675	972.5513	1752.3032	5.9111	3.09	0.43
3000	300	6	-	60290.5	325.6368	794.0744	5.8360	3.77	0.77
3000	300	7	-	63938	2180.3160	2572.7480	4.8111	2.17	0.23
3000	300	8	-	72650	339.5238	793.3408	7.5145	3.76	0.41
3000	300	9	-	74885.5	208.4657	595.3975	4.1286	2.31	0.17
3000	300	10	88220	80655.5	618.7099	950.4041	4.8346	2.43	0.11

5.3 Rezultati dobijeni primenom hibridne metaheuristike

U tabeli 5.3.1 su rezultati primene hibridne metaheuristike kada je broj pčela $B = 5$, broj koraka svake pčele u jednoj iteraciji metode $NC = 5$, broj okolina $k_{max} = 5$, a verovatnoća da će pčela u jednom koraku primeniti VNS je $\alpha = 0.3$. Imajući u vidu da je prilikom testiranja pojedinačnih metaheuristika primećeno da keširanje nije značajno popravilo vreme izvršavanja, taj podatak za ovu metodu nije prikazan. Maksimalni broj iteracija bez popravljanja rešenja je postavljen na 200. Radi poređenja različitih metoda, korišćene su iste četiri instance sa variranjem parametra p kao i ranije.

Tabela 5.3.1: Rezultati hibridne (BCO + VNS) metaheuristike

ULAZ			CPLEX	BCO + VNS				
$\|I\|$	$\|J\|$	p	opt	sol_{best}	t_{best}(s)	$t_{tot}(s)$	agap(%)	σ(%)
1000	100	1	3070.5	3070.5	0.2627	4.8476	0.0000	0.00
1000	100	5	12402.5	11963	9.7821	19.6544	4.1269	2.60
1000	100	10	21554.5	19201.5	15.9505	26.4653	2.4959	1.52
3000	100	1	6979.5	6979.5	0.3526	9.2690	0.0000	0.00
3000	100	3	19318.5	18833	18.0263	35.9934	0.7269	0.56
3000	100	5	29698.5	28721	31.5025	60.5857	2.0929	1.46
3000	100	10	52983.5	48633.5	30.3960	80.7124	1.1811	0.84
3000	100	20	86302.5	77621.5	58.2923	120.4226	1.2116	0.83
3000	100	30	105817	96972	73.4976	159.2226	1.4825	0.93
3000	100	40	117687	109242	96.9511	208.9576	0.8104	0.43
3000	100	50	125856	119053.5	97.8160	249.6978	0.6706	0.40
3000	300	1	14986.5	14986.5	0.9060	14.6003	0.0000	0.00
3000	300	2	27607	27607	15.5635	33.1883	1.2551	1.38
3000	300	3	39184	38409.5	26.2436	47.1399	1.7022	1.03
3000	300	4	48064	46872.5	25.6547	62.6217	1.7160	0.78
3000	300	5	-	54449.5	26.4491	64.5451	1.4568	0.80
3000	300	6	-	62552.5	50.0737	78.6436	2.9000	1.20
3000	300	7	-	67257.5	37.9862	89.4903	1.0477	0.69
3000	300	8	-	72710	35.6418	73.0420	1.1215	0.89
3000	300	9	-	77780	41.7936	97.2115	0.7994	0.60
3000	300	10	88220	81674	64.9074	114.7012	0.8787	0.58
10000	1000	10	-	13778844.5	132.7350	230.4838	1.1911	0.78
10000	1000	20	-	25543402.5	93.7382	246.5262	1.2026	0.75
10000	1000	30	-	36542599	148.0177	265.6374	1.5575	0.86
10000	1000	40	-	45884016.5	119.0698	288.6451	0.6485	0.49
10000	1000	50	-	55181238	180.5431	392.4536	0.7895	0.40
10000	1000	60	-	63496240.5	141.6441	388.5013	0.8342	0.42
10000	1000	70	-	71053886	185.3354	453.6177	0.5139	0.33
10000	1000	80	-	78017338.5	171.4282	448.7873	0.3344	0.25
10000	1000	90	-	84767767.5	166.5828	465.5111	0.5577	0.37
10000	1000	100	-	90915989.5	219.1566	460.4576	0.3793	0.28

6 Analiza rezultata

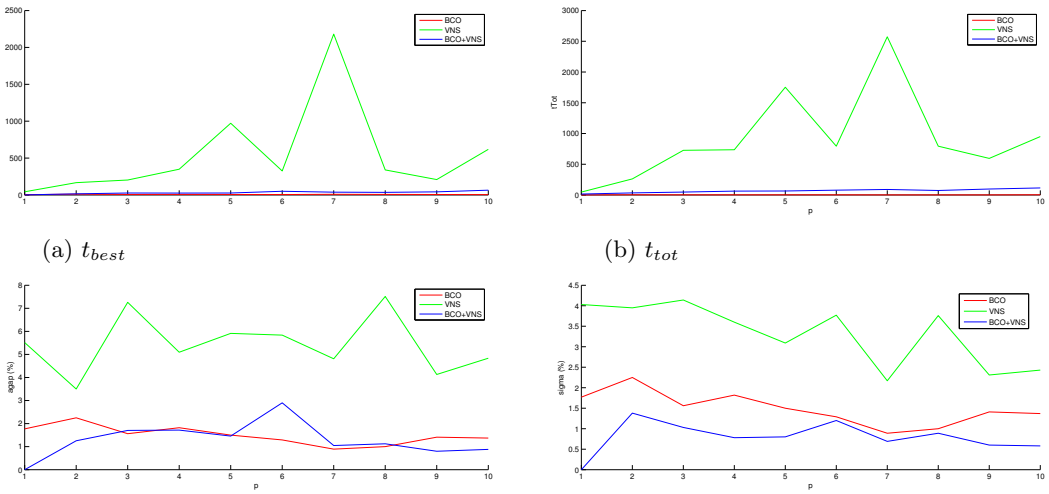
Instance su kreirane tako da se može posmatrati uticaj različitih parametara na izvršavanje programa. Primećeno je sa povećavanjem razlike između p i $|J|$ rastu vreme i broj iteracija egzaktne metode, što znači da za fiksirano $|J|$ postižu se bolji rezultati kada je p veće. Sa druge strane, zbog načina zapisa rešenja i same implementacije metaheuristika, vreme izvršavanja raste sa povećavanjem parametra p zbog velikog porasta u broju mogućih rešenja za ispitivanje. Međutim, metaheuristike su se pokazale dobre u onim slučajevima kada pomoću CPLEX-a nije dobijeno tačno rešenje ili je vreme izvršavanja bilo blizu/preko ograničenja od 4h. Drugačiji izbor parametara bi verovatno doveo i do još boljih rezultata.

U slučaju VNS metaheuristike, prilikom LS je korišćen first improvement pristup, a broj okolina za pretragu je bio ograničen odozgo parametrom p. Prirodno je očekivati da bi izmena u definisanju okolina i zahtevanje najboljeg, a ne uzimanje prvog poboljšanja u okviru lokalne pretrage moglo rezultovati vrednostima funkcije cilja koje manje odstupaju od stvarnih optimuma. Keširane vrednosti nisu donele veliku uštedu sem za malo p.

Što se tiče BCO metode, radi jednostavnosti je procena mogućih koraka pri jednom letu unapred implementirana dosta stohastički. Kao rezultat, metoda radi jako brzo ali često daje suboptimalna rešenja. U planu je razrada efikasanog ali i dovoljno tačnog načina procene narednog koraka. Za isti broj MaxIter, BCO se pokazala dosta bržom od VNS, upravo jer problemu pristupa stohastički.

Metoda koja je nastala hibridizacijom VNS i BCO metaheuristika brzinu optimizacije kolonijom pčela sa detaljnošću koju donosi lokalna pretraga iz metode promenljivih okolina.

Na slici 3b se može videti zavisnost vremena izvršavanja od p (koji najviše utiče na kompleksnost ulaza) za svaku od metoda, a na slici 3a slično za vreme do prvog pronalaska najboljeg rešenja. Na slikama 3d i 3c vidi uticaj istog parametra na standardnu devijaciju i odstupanje, takođe za sve tri metode.



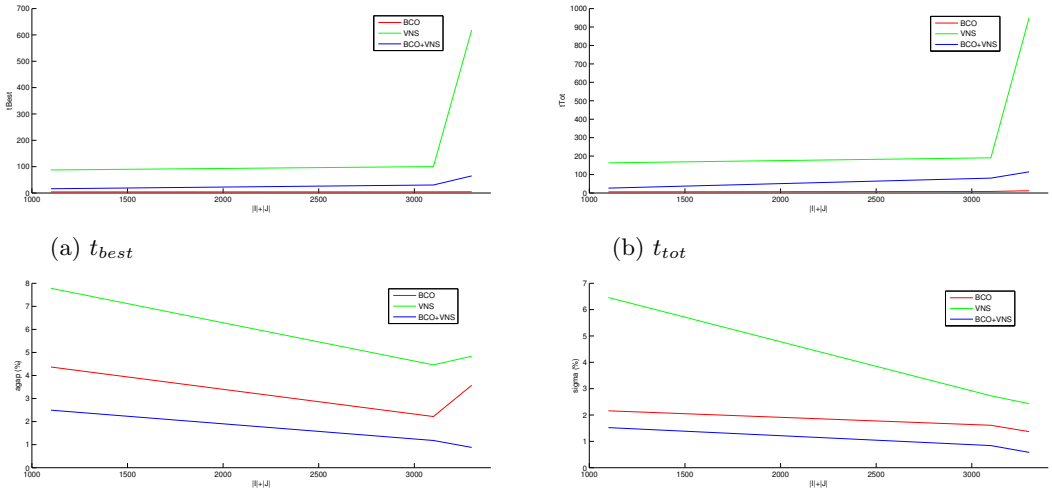
(a) t_{best} (b) t_{tot}

(c) *agap* (d) σ

Slika 3: Uticaj parametra p na performanse metoda

Sa ovih grafika se može primetiti da je VNS metoda znatno lošija kada se poredi prosečno vreme do pronalaska optimuma, dok se BCO i nova metoda tu ne razlikuju bitno. Takođe, očigledno je odstupanje najmanje kod hibridne metaheuristike.

Može se posmatrati i kako vreme, standardno odstupanje i devijacija zavise od veličine ulaza, odnosno od parametara $|I|$ i $|J|$, što je prikazano na na slikama 4a,4b,4c i 4d.

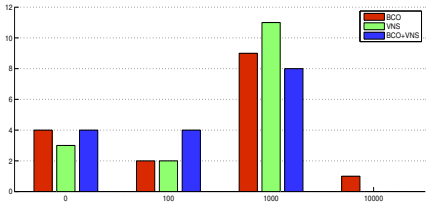


(a) t_{best} (b) t_{tot}

(c) $agap$ (d) σ

Slika 4: Uticaj parametara $|I|$ i $|J|$ na performanse metoda

Ako bismo uspešnost primenjene heuristike merili brojem onih instanci u kojima je dobijena tačna optimalna vrednost (za 16 instanci na kojima su te vrednosti poznate, odnosno na koje se CPLEX mogao primeniti), dolazimo do podataka zabeleženih na slici 5.



Slika 5: Odstupanje od optimalnih vrednosti

Do stvarnog maksimuma BCO i hibridna metaheuristika su došle u po četiri od šesnaest instanci, a VNS u tri. Najviše je bilo testova gde je odstupanje od optimalne vrednosti između 1000 i 10000, a do odstupanja od preko 10000 došlo je samo jednom i to kod BCO metode.

7 Zaključak

U ovom radu predstavljen je jedan apstraktan model problema koji se lako može prepoznati u raznim konkretnim situacijama. Pored MAXCAP problema, predstavljene su dve metaheuristike i jedan predlog njihove kombinacije koja bi mogla biti pogodna za razmatrani problem. Zbog relativno malog broja testova, optimalna podešavanja parametara verovatno nisu pronađena. Na kraju navedimo još da velika razlika u vrednosti funkcije cilja može nastati i pri maloj razlici u vektoru x. Naime, primetimo da je za jednu od većih instanci sa $|I| = 3000, |J| = 300, p = 1$ bio dovoljan jedan uspostavljen centar da bi funkcija cilja imala vrednost 14986.5. U najgorem slučaju, a teorijski mogućem, ukoliko on jedini koji pokriva te potrošače, svako rešenje koje ga ne uključuje bilo za čak 14986.5 manje od optimalnog. Ovo svakako može poslužiti kao podstrek za dalju razradu predloženog pristupa.

8 Literatura

[1] C. ReVelle: The maximum capture or sphere of influence problem: hotelling revised on a network. J. Reg. Sci. 26, 343–357 (1986)

[2] C. ReVelle, D. Serra: The maximum capture problem including reallocation. Inf. Oper. Res. 29, 130–138 (1991)

[3] D. Serra, C. ReVelle: Competitive location in discrete space, in Facility Location. A Survey of Applications and Methods, ed. by Z. Drezner (Spribger, Berlin, 1995), pp. 367–386

[4] D. Serra, C. ReVelle: Competitive location and pricing on networks. Geogr. Anal. 31, 109–129 (1999)

[5] Athanasia Karakitsiou: Modeling Discrete Competitive Facility Location (Springer, 2015)

[6] Lučić, Teodorović: Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. In: Preprints of the TRISTAN IV Triennial Symposium on Transportation Analysis, Sao Miguel, Azores Islands, Portugal (2001) 441–445

[7] Šelmić, Teodorović, Davidović: Bee Colony Optimization part I: The algorithm overview, Yugoslav Journal of Operations Research 25 (2015), Number 1, 33-56

[8] Šelmić, Teodorović, Davidović: Bee Colony Optimization part II: Application, Yugoslav Journal of Operations Research 25 (2015), Number 2, 185–219

[9] N. Mladenović, P. Hansen : Variable neighborhood search, Computers and Operations Research (1997)

[10] Glover, Kochenberger: Handbook of Metaheuristics, Kluwer Academic Publishers (2003)

[11] El-Ghazali Talbi: Metaheuristics from design to implementation (2009)

[12] www.people.brunel.ac.uk