Problem K Centara

Jelena Živović, Nikola Janković
jzivovic96@gmail.com, nikola_jankovic@tuta.io

Matematički fakultet, Univerzitet u Beogradu
Profesor: Dr Aleksandar Kartelj

15. april 2020.

Sažetak

U ovom seminarskom radu biće obrađen popularni grafovski problem K-centara, kao i neke poznate heuristike za rešavanje ovog problema poznate u literaturi, kao i prilagođene metaheuristike široke upotrebe sa kojima su se autori ovog teksta susreli na fakultetskom kursu Računarska inteligencija. Na kraju će biti prikazani komparativni rezultati svake metode dobijeni eksperimentalnim putem.
Ključne reči: k-centara, heuristike, np-težak, optimizacija

1 Uvod

Problem koji će biti razmatran je izbor najboljih lokacija za postavljanje objekata na nekom prostoru. Ovaj, kao i ostali lokacijski problemi imaju široku praktičnu primenu. Odlučivanje o mestima za gradnju fabrika, bolnica, tržnih centara i ostalih objekata gde je potrebno obezbediti što kvalitetniju uslugu, sa ograničenim resursima, što većem broju korisnika. Formalno, problem se definiše na sledeći način:

Definicija 1 *Neka je $G = (V, E)$ kompletan, neusmeren graf sa težinama koje zadoljavaju nejednakost trougla. I neka je k pozitivan ceo broj ne veći od $|V|$. Za bilo koji skup $S \subseteq V$ i $v \in V$. Definišimo sad $d(v, S)$ kao težinu najkraće grane od čvora v do bilo kog čvora u S. Potrebno je pronaći takav skup $S \subseteq V$ da važi $|S| < k$, koji minimizuje izraz $max_{vinV} d(v, S)$.*

Ovaj problem spada u klasu NP-teških problema [6], tako da egzaktan algoritam u polinomijalnom vremenu, barem za sada, ne postoji. U nastavku će biti prikazani neki od egzaktnih načina za dobijanje suboptimalnih rešenja ovog problema i adapatacije nekih metaheuristika koje su poznate u literaturi.

2 Pregled poznatih heuristika

U ovoj sekciji biće prikazana tri najrasprostranjenija pristupa koja se javljaju u literaturi na ovu temu.

2.1 2-Aproksimativni pohlepni algoritam

Ovo je pojednostavljena verzija aproksimativne heuristike [2] za koju postoji dokaz da predstavlja 2-aproksimativnu heuristiku. P-aproksimativna heuristika zadovoljava svojstvo da rešenje koje ona nudi je u najgorem slučaju p puta lošije od optimalnog rešenja. Takođe [2] pokazuje da ne postoji aproksimativna heuristika sa vrednošću $|p| < 2$.

Algoritam se zasniva na postojanju r radiusa koji se inkrementira sve dok ne bude zadovoljen uslov da se među izabranim čvorovima nalazi manje od k čvorova, gde k predstavlja broj centara koje je potrebno alocirati. Izbor čvorova se svodi na iterativno biranje nasumičnog čvora iz skupa u kom se na početku nalaze svi čvorovi polaznog grafa, a u svakoj poditeraciji se iz skupa eliminišu izabrani čvor i svi čvorovi na razdaljini manjoj od r od izabranog čvora.

```
Ulaz: Graf G(V, E), radius r, k
Izlaz: Skup W koji sadrži K čvorova koji
    predstavljaju rešenje
REPEAT:
    C = skup svih čvorova u grafu;
    W = prazan skup;
    WHILE C nije prazan:
        X proizvoljan čvor iz skupa C;
        dodaj X u W;
        izbaci iz skupa C čvor X i sve
    njegove susede na razdaljini
        manjoj od r;
    IF length(W) < k:
        return W;
    ELSE
        azuriraj r;
        vrati se na početak;
```

Listing 1: 2-aproksimativna heuristika

2.2 Gonzalesov algoritam

Drugi pristup, koji je takođe jedan od poznatijih, predstavljen je u radu meksičko-američkog naučnika Teofila Gonzalesa [12]. Heuristika nije preterano teška za implementaciju. Svodi se na biranje k čvorova iz grafa, tako što se prvi čvor izabere nasumično, a svaki i-ti čvor se bira tako da funkcija razdaljine $d(v_i) = min(w(v_i, v_1), w(v_i, v_2), ..., w(v_i, v_{i-1})$ bude maksimalna. Često u literaturi, ovaj algoritam se naziva još i algoritam najudaljenije tačke, jer se u svakoj iteraciji bira tačka najudaljenija od trenutno izabranih tačaka.

```
Ulaz: Graf G(V, E), k
Izlaz: Lista čvorova R, koja predstavlja reš
    enje

R[0] = nasumičan čvor u grafu
for i in range(1, k):
    najboljiCvor = -1
    najboljiRez = -1
    for j in range(|V|):
        if j nije u R:
            najbližiCvor = nadjiNajblizi(G,
    i)
            if najbližiCvor > najboljiRez:
                najboljiCvor = j
                najboljiRez = najbliziCvor

    R.append(najboljiCvor)

return R
```

Listing 2: Gonzalesov algoritam

Vremenska složenost ove heuristike je $O(kn)$. Složenost je moguće popraviti do složenosti $O(nlogk)$ koristeći delimično komplikovaniju modifikaciju. [13]

2.3 Pristup korišćenjem dominirajućeg skupa grafa

Ovaj pristup se karakteriše time da se problem k centara rešava kao serija uzastopnih rešavanja problema minimalnog dominirajućeg skupa grafa. Problem minimalnog dominirajućeg skupa podrazumeva nalaženje podskupa D, minimalne kardinalnosti, skupa čvorova grafa takvog da je svaki čvor, koji nije u D, susedan najmanje jednom čvoru iz D. Problem minimalnog dominirajućeg skupa je NP-težak[5]. Formalno, dominirajući skup grafa se definiše na sledeći način:

Definicija 2 *Neka je $G = (V, E)$ graf. Skup $D \subseteq V$, takav da je svaki čvor $v \in V \setminus D$ susedan najmanje jednom čvoru iz D se zove dominirajući skup grafa G.*

Algoritam za nalaženje minimalnog dominirajućeg skupa se zasniva na tome da se čvorovi u skup D dodaju što je kasnije moguće, po tzv. "lazy"principu[5]. Za svaki čvor $v \in V$ se čuva broj koliko je puta čvor pokriven od strane preostalih čvorova u grafu (to su oni čvorovi koji nisu dodati u D niti je za njih izbačena mogućnost da mogu da se nađu u D) u nizu $CovCnt$ i inicijalna vrednost je $CovCnt[v] = deg(v) + 1$ gde je $deg(v)$ stepen čvora v. Takođe, za potrebe rangiranja čvorova se koristi strategija bodovanja (scoring strategy) koja podrazumeva da se za svaki čvor čuva broj bodova (score) koji je inicijalno jednak: $Score[v] = deg(v) + 1$. Glavni deo algoritma je petlja koja se ponavlja $|V|$ puta. U svakoj iteraciji se prvo pronalazi čvor x takav da je njegov broj bodova najmanji. Zatim se proverava da li postoji čvor y koji je susedan čvoru x i za koji važi: $CovCnt[y] = 1$, ako se ustanovi da postoji, čvor x se dodaje u skup D jer je x jedini čvor koji može da pokrije y. Takođe, za svakog suseda y čvora x se postavi $CovCnt[y] = 0$, što označava da su ti čvorovi pokriveni. Ako ne postoji čvor y za koji važi da je susedan x i $CovCnt[y] = 1$, onda za svaki čvor y za koji važi $CovCnt[y] > 0$ se uradi sledeće: $CovCnt[y]$ se dekrementira, što znači da ih x ne pokriva više, i $Score[y]$ se inkrementira, što znači da vrede više kao kandidati za članove dominirajućeg skupa. Na kraju svake iteracije vrednost $Score[y]$ se postavi na beskonačnu vrednost.

```
Ulaz: graf G(V, E)
Izlaz: Dominirajući skup D
FOR v in V DO:
    CovCnt[v] := deg(v) + 1;
Score := CovCnt;
D := prazanSkup;

FOR i := 1 to |V| DO:
    x := čvor_sa_najmanjim_brojem_bodova;
    IF postoji y takvo da (x,y) pripada E i
    CovCnt[y]==1 THEN:
        dodaj x u D;
        FOR y : (x, y) pripada E DO:
            CovCnt[y] := 0;
    ELSE:
        FOR y takvo da (x,y) pripada E DO:
            IF CovCnt[y] > 0 THEN:
                CovCnt[y]--;
                Score[y]++;

    Score[x] = inf;
return D;
```

Listing 3: Algoritam za nalaženje dominirajućeg skupa

```
Ulaz: Graf G(V, E), k
Izlaz: Skup centara C

sortirati grane grafa prema težini
u neopadajućem poretku;
FOR e in E DO:
    GTmp := ParametricPrunning(G, e);
    C := prazan_skup;
    best_value = inf;
    IF GTmp povezan graf THEN:
        CTmp := DomSetAlgorithm(GTmp);
        IF |CTmp| <= k i evaluate(C) <
    best_value THEN:
            C := CTmp;
            best_value = evaluate(C);

return C;
```

Listing 4: Algoritam za rešavanje problema k-centara

Vremenska složenost prethodnog algoritma je $O(n^3)$, ali ako se čvorovi uvek obrađuju u istom redosledu, vremenska složenost može bit redukovana na $O(n^2)$ koristeći bektreking[5].

Algoritam za rešavanje problema k-centara koristi prethodno opisani algoritam i parametarsko potkresivanje (parametric pruning). Inicijalno, težine grana su sortirane u neopadajućem poretku. Za svaku granu težine t, graf se potkresuje tako što se eliminišu sve grane čija je težina veća od t i tako se dobija podgraf C i nakon toga se traži minimalni dominirajući skup grafa C ako je C povezan graf[5]. Ako C ima manje ili tačno k čvorova, uzima se u obzir kao jedno od rešenja za problem k-centara.

Ukupna vremenska složenost algortma za rešavanje problema k-centara je $O(n^2)$ računajući i sortiranje grana, potkresivanje kao i algoritam za nalaženje minimalnog dominirajućeg skupa. Vremenska složenost se može popraviti na $O(n^2 logn)$[5].

3 Metaheuristike

Metaheuristike su metode koje koje nisu dizajnirane za rešavanje konkretnog problema, već za rešavanje čitave klase problema, ali iako su metaheuristike opšte metode, mogu biti prilagođene specifičnom problemu[10]. U nastavku teksta je dat pregled nekih metaheuristika koje se mogu koristiti za rešavanje problema k-centara.

3.1 S-metaheuristike

Slovo S u imenu ove metaheuristike označava početno slovo reči singl (eng. *single*). Dakle, S-metaheuristike koriste jedno rešenje koje se popravlja kroz iteracije. Proces se može posmatrati kao „šetnja"kroz okolinu rešenja ili pronalaženje valjane trajektorije kroz prostor rešenja. [3] U radu će biti obrađene tri metaheuristike iz ove grupe i njihove primene na problem k-centara. To su simulirano kaljenje, tabu pretraga i metod promenljivih okolina.

3.1.1 Simulirano kaljenje

Algoritam simuliranog kaljenja je zasnovan na procesu kaljenja čelika u cilju oplemenjivanja metala kako bi on postao čvršći. Prvo je potrebno zagrevati čelik do određene temperature, pa ga onda zadržati neko vreme na toj temperaturi i nakon toga sledi postepeno hlađenje, ali treba voditi računa o brzini hlađenja, da ne bi došlo do pucanja čelika[8].

Glavni princip po kome radi algoritam simuliranog kaljenja je poboljšavanje vrednosti jednog rešenja. Najpre se generiše početno rešenje, u našem slučaju na nasumičan način. Zatim, sve dok nije ispunjen uslov zaustavljanja potrebno je prvo odabrati rešenje u okolini trenutnog rešenja[8]. U našem slučaju, rešenje problema k-centara je predstavljeno nizom boolean vrednosti, pri čemu ako se na i-toj poziciji nalazi vrednost True, to znači da je i-ti čvor trenutno u skupu čvorova koji predstavlja rešenje. Samim tim, rešenje u okolini trenutnog se bira tako što se

nasumično izaberu dve pozicije od kojih se na jednoj nalazi vrednost True, a na jednoj False i te dve vrednosti se zamene. Ako je dobijeno rešenje bolje od trenutnog, trenutno rešenje postaje novodobijeno, inače proveravamo da li je $p > q$ gde je p unapred definisana funkcija neopadajuća na $(0, 1)$, a q je nasumična vrednost iz intervala $(0, 1)$. Ako važi $p > q$ onda trenutna vrednost postaje novodobijena vrednost iako nije ta vrednost nije bolja od trenutne čime se sprečava verovatnoća konvergencije ka lokalnom optimumu, a inače se novodobijeno rešenje odbacuje. Po potrebi se ažurira najbolje moguće rešenje.

```
Ulaz: Graf G(V, E), k
Izlaz: suboptimalno rešenje

trenutno_rešenje = generiši_nasumično_reš
    enje()
trenutna_vrednost = odredi_vrednost(
    trenutno_rešenje)
najbolja_vrednost = trenutna_vrednost

for i in range(1, max_br_iter):
    novo_rešenje = izaberi_reš
    enje_u_okolini_trenutnog()
    nova_vrednost = odredi_vrednost(novo_reš
    enje)
    if nova_vrednost < trenutna_vrednost:
        trenutna vrednost = nova_vrednost
    else:
        p = 1 / i ** 0,5
        q = nasumični_broj(0, 1)
        if p > q:
            trenutna_vrednost =
    nova_vrednost

    if nova_vrednost < najbolja_vrednost:
        najbolja_vrednost = nova_vrednost
return najbolja_vrednost
```

Listing 5: Algoritam simuliranog kaljenja

3.1.2 Tabu pretraga

Pojam Tabu pretrage (eng. *Tabu search*) se prvi put javlja u radu [4], u kom se i prvi put pojavljuje pojam metaheuristike. Glavni princip po kom radi ova metaheuristika zasniva se na pretrazi prostora izvan lokalnog optimuma. Komponenta koja omogućava ovako fleksibilno ponašanje je prilagodljiva memorija. Ova prilagodljivost se ogleda u mogućnosti da se prostor rešenja pregleda ekonomično i efikasno. Pseudo-kod 6 ukazuje da bi u svakoj iteraciji trebalo izabrati nasumično rešenje iz skupa dozvoljenih [9]. Ovaj korak može da se modifikuje i da se bira najbolje rešenje iz okoline trenutno rešenja, ali rezultati oba pristupa biće prikazani u sekciji 4.

```
Ulaz: funkcijaCilja
Izlaz: suboptimalno rešenje

s = generisiPocetnoResenje()
f_star = funkcijaCilja(s)
skupZabranjenih = set()
while not stoppingCriteria():
    p = izaberiRandom(naćiOkolinu(s).
     difference(T))
     if funkcijaCilja(p) < funkcijaCilja(s):
         s = p
     if funkcijaCilja(p) < f_star:
         f_star = funkcijaCilja(p)
     update(T)

return s
```

Listing 6: Pseudo-kod Tabu pretrage

3.1.3 Metoda promenljivih okolina

Metoda promenljivih okolina predstavlja uopštenje lokalne pretrage, gde su definisane okoline po kojima se vrši pretraživanje. Metoda promenljivih okolina se bazira na sledećim principima[7]:

- Lokalni minimum za jedan tip okoline ne mora nužno i da bude lokalni minimum za drugi tip okoline.
- Globalni minimum predstavlja lokalni minimum za sve tipove okolina.
- Za mnoge probleme u praksi, lokalni minimumi više tipova okolina su relativno blizu jedan drugog.

Postoji više varijanti osnovnog algoritma, među kojima se najčešće javljaju redukovana, osnovna i uopštena[7]. U ovom radu je predstavljena redukovana metoda za rešavanje problema k-centara.

Kod redukovane metode promenljivih okolina izostavljena je lokalna pretraga i umesto lokalnog minimuma bira se proizvoljno rešenje iz odgovarajuće okoline trenutnog u cilju traženja poboljšanja[7].

Najpre je potrebno inicijalizovati početno rešenje, što je u našem slucaju lista boolean vrednosti gde vrednost True na i-toj poziciji označava da i-ti čvor pripada rešenju. Zatim, sve dok nije zadovoljen kriterijum zaustavljanja, bira se proizvoljno rešenje iz okoline trenutnog na sličan način kao kod algoritma

simuliranog kaljenja. Proverava se da li je novodobijeno rešenje bolje od trenutnog, ako jeste, trenutno rešenje se ažurira, proverava se da li je trenutno rešenje bolje od trenutnog najboljeg i prelazi se u sledeću iteraciju. Ako novodobijeno rešenje nije bolje od trenutnog, ponovo se bira proizvoljno rešenje iz okoline trenutnog.

```
Ulaz: Graf(G, V), maxEnv
Izlaz: Suboptimalno rešenje
trenutno_rešenje = generiši_nasumično_reš
    enje()
trenutna_vrednost = odredi_vrednost(
    trenutno_rešenje)
najbolja_vrednost = trenutna_vrednost

for i in range(maxIters):
    env = 0
    while env < maxEnv:
        novo_rešenje = izaberi_reš
    enje_u_okolini_trenutnog()
        nova_vrednost = odredi_vrednost(
    novo_rešenje)
        if nova_vrednost < trenutna_vrednost
    :
            trenutna vrednost =
    nova_vrednost
            env = 0
        else:
            env += 1

        if nova_vrednost < najbolja_vrednost
    :
            najbolja_vrednost =
    nova_vrednost
return najbolja_vrednost
```

Listing 7: Metoda promenljivih okolina

3.2 P-metaheuristike

Slovo P u imenu ove grupe metaheuristika označava populaciju. Dakle grupa ovakvih algoritama se u osnovi ne baziraju na jednom rešenju već na populaciji rešenja koja se kroz iteracije menja tako da sva pojedinačna rešenja zajedno konvergiraju ka nekom optimumu. Većina metaheuristika ove grupe su inspirisane pojavama u prirodi[3]. U radu će biti prikazane adaptacije genetskog algoritma i tačkaste pretrage (eng. *scatter search*).

3.2.1 Genetski algoritam

Osnovna verzija ovog pristupa je detaljno razjašnjena [3, 1]. Modifikacija osnovnog procesa se događa na nekoliko podmetoda. To su konkretno: proces mutacije i proces ukrštanja.

Pre nego što objasnimo u čemu je odstupanje kod ova dva koraka genetskog algoritma napomenimo još jednom da je rešenje, to jest jedinke u ovom slučaju, nizovi dužine $|V|$ koji sadrže indikatore da li određeni čvor među izabranim. Proces mutacije se razlikuje jedino u opsegu stope mutacije koju je neophodno povećati u odnosu na standardnih $5-10\%$. Glavni razlog za tu promenu je činjenica da je $k \ll |V|$, pa jednostavna mutacija ne bi uspela da izvuče pretragu iz lokalnog minimuma. Kod procesa ukrštanja situaciju komplikuje dodatno ograničenje da broj uključenih indikatora mora biti jednak k. [1] Zato ne možemo da iskoristimo neko od jednostavnih n-pozicionih ukrštanja jer je verovatnoća generisanja nedopustivih rešenja bila značajna, a postupak korekcije bi zahtevao nepotrebno mnogo vremena. Rešenje je da pored indikatorske reprezentacije rešenja čuvamo i listu uključenih indikatora za svaku jedinku. Iz kojih bi pri procesu ukrštanja bilo nasumično izabrano $x < k$ indeksa iz obe jedinke čije bi se vrednosti međusobno izmenile. Za korektnost procesa neophodno je iz izbora isključiti indekse koji se nalaze u obe jedinke.

```
Ulaz: jedinka1, jedinka2, za_m1, za_m2
Izlaz: ukrštene jedinke
for ind1, ind2 in zip(za_m1, za_m2):
    jedinka1[ind1] = 0
    jedinka2[ind1] = 1

    jedinka2[ind2] = 0
    jedinka1[ind2] = 1
```

Listing 8: Proces ukrštanja na dve pozicije

Ostatak osnovnog oblika ove heuristike je načelno isti i za ovakav problem.

3.2.2 Scatter pretraga

Scatter pretraga je još metod optimizacije koji primenjuje principe evolucije. Ova metaheuristika je formalizovana sedamdesetih godina prošlog veka. Prvi radovi datiraju jednu deceniju ranije. U nastavku će biti izloženi glavni koraci, dok detaljnije informacije o svakom koraku. mogu se pronaći [11].

Osnovi koraci su:

- Generisanje različitih početnih rešenja (eng. *A Diversification Generation Method*)
- Unapređivanje trenutnih rešenja - Od jednog rešenja je moguće dobiti jedan ili više

[1] Iako po definiciji 1 broj može da bude i manji od k zbog jednostavnije implementacije ograničićemo dodatno da broj mora biti tačno k.

unapređenih

- Ažuriranje uskog skupa najboljih rešenja
- Kreirati podskupove skupa najboljih rešenja koji će biti korišćen za kreiranje novih rešenja.
- Rekombinovati rešenja iz kreiranih podskupova i kreirati nova.

Okviran postupak je moguće videti 9

```
diversificationGenerationMethod()
improveMethod()
while not stoppingCriteria():
    azuriajRefSkup()
    while postoji novo rešenje u refSkupu:
        generisiPodskupove()
        rekombinujRešenja()
        improveMethod()
        azurirajRefSkup()


return najbolji iz refSkupa
```

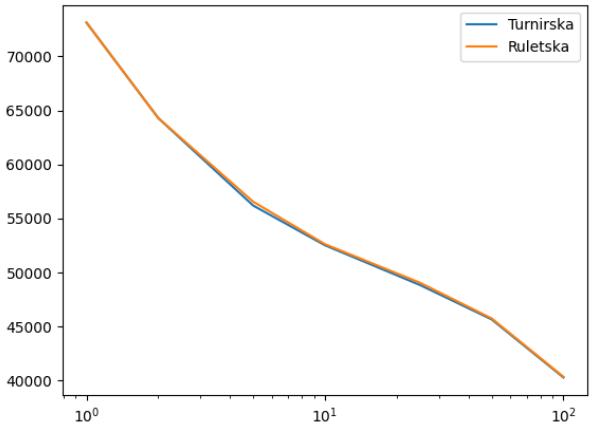
Listing 9: Pseudo-kod Scatter pretrage

Treba posvetiti dodatnu pažnju metodu rekombinovanja rešenja. Ovaj metod je sličan procesu ukrštanja kod genetskih algoritama. Razlika je što se ovde ne očekuje da podskup na kom se primenjuje rekombinacija bude kardinalnosti dva. Pri istaživanju, autori su odlučili da nad n-točlanim skupom primene jednostavnu operaciju bitovske disjunkcije i da naknadno izvrše korigovanje rešenja kako bi održali dopustivost.

4 Rezultati istraživanja

Na početku prikazivanja rezultata osvrnućemo se na neke pojedinosti koje bi trebalo naglasiti. Prvo, definicija 1 navodi da je jedan od uslova problema ta da svake tri grane u kopletnom grafu G zadovoljavaju nejednakost trougla. Pri generisanju grafa autori su odlučili da prednost daju vremenski efikasnom generisanju složenosti $O(|E|)$ ispred velike varijabilnosti težina grana. Naime, pristup koji je korišćen da sve težine budu uniformno birane iz poluotvoreni interval $[A, 2A)$ gde je A proizvoljan element skupa R^+. Ovakav pristup daje znčajno manju raspršenost vrednosti od metoda generisanja nasumčnih tačaka u $2D$ ravni i računanjem njihovih međusobnih rastojanja, ali je vremenska složenost neuporedivo manja.

Genetski algoritam: Pre komparativne analize različitih metaheuristika međusobno, neophodno je za neke algoritme izvršiti internu analizu za različite parametre. Kao predstavnik zavisan od najvećeg broja parametara, najviše prostora za internu analizu je ostavljeno upravo genetskim algoritmima.



Slika 1: Odnos rezultata pri izboru turnirske i ruletske selekcije za n = 500 i promenljivo k

Na slici 1 uočavamo da su ruletska i turnirska selekcija dosta slične, ali da ipak turnirska selekcija pokazuje bolje rezultate. Takođe treba uzeti u obzir da je turnirska selekcija vremenski efikasnija. Ovo bi verovatno bilo još uočljivije da je veća varijabilnost težina postignuta, o čemu smo pričali u paragrafu iznad. Nakon toga izvršeno je testiranje kolika veličina populacije je najbolja uz praćene vremenske zahtevnosti.

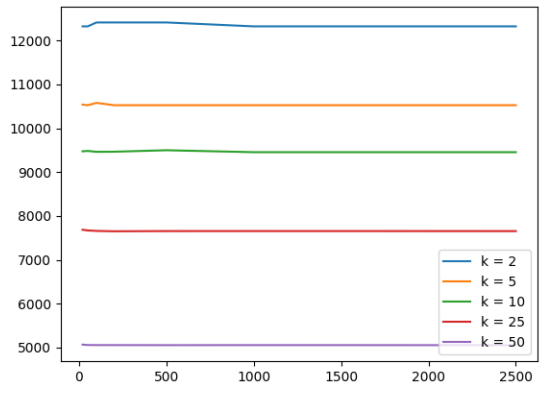
U tabeli 4 je moguće videti rezultate koji nam pokazuju da veće populacije ne pomažu značajno pri poboljšanju rezultata. Ali da značajno utiču na vremensku zauzetost. Treba napomenuti da su ova testiranja rađena sa 20 iteracija po svakom testiranju, pa bi vremenska zauzetost bila značajno veća.

Broj iteracija je poslednji parametar koji je interno testiran. Kao što je moguće videti na slici 2 broj

pop_vel \ k	2	10	25	50
100	(12474.32, 0.19)	(9465.88, 0.31)	(7643.91, 0.58)	(5043.19, 1.05)
200	(12495.00, 0.40)	(9499.61, 0.64)	(7636.24, 1.19)	(5043.60, 2.16)
500	(12428.25, 1.03)	(9482.85, 1.66)	(7640.65, 3.11)	(5042.89, 5.38)
1000	(12428.25, 2.07)	(9439.88, 3.24)	(7643.19, 6.18)	(5041.06, 10.93)
2500	(12428.25, 5.37)	(9436.71, 8.42)	(7631 15.77)	(5040.10, 27.3)

Tabela 1: Zavisnost rezultata i potrebnog vremena (rezultatm, vreme) od veličine populacije

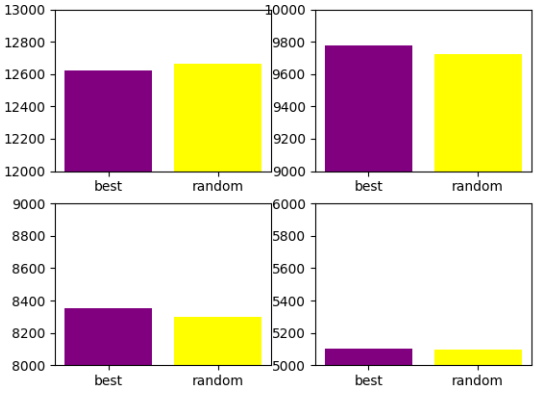
iteracija se pokazao kao bitan samo za malu vrednost k. Tako da će u nastavku istraživanja biti korišćen broj iteracija 500 jer se on pokazao kao kvalitetan za svaku vrednost k.



Slika 2: Kvalitet rezultata u odnosu na broj iteracija

Tabu pretraga je druga metaheuristika nad kojom je izvršeno interno testiranje parametara. Konkretno, izvršena je provera da li način izbora narednog rešenja iz skupa dozvoljenih rešenja utiče značajno na kvalitet rezultata. Isprobano je nasumičnim izborom kao što preporučeno u 6. Drugi pristup, koji je takođe zastupljen u literaturi, preporučuje izbor najboljeg dozvoljenog rešenja. Rezultati koji su dobijeni primenom na problem o kom se govori u radu su vidljivi na slici 3. Vidimo da je nasumičan izbor dao bolje rezultate za manje k, dok za veće vrednosti izbor najboljeg dozvoljenog rešenja daje bolji rezultat.

Završna analiza U tabeli 2 se može videti prob-



Slika 3: Rezultati dobijeni tabu pretragom za vrednosti $k \in \{2, 10, 20, 50\}$

vrsta	rezultat
BruteForce	2191.7802
2-approximation	2727.7366
Greedy	2329.4667
Scatter	2191.7802
Simulated annealing	2192.4801
Dominating set	2401.2494
Variable search neighbourhood	2191.7802
Tabu	2341.0661
EA	2191.7802

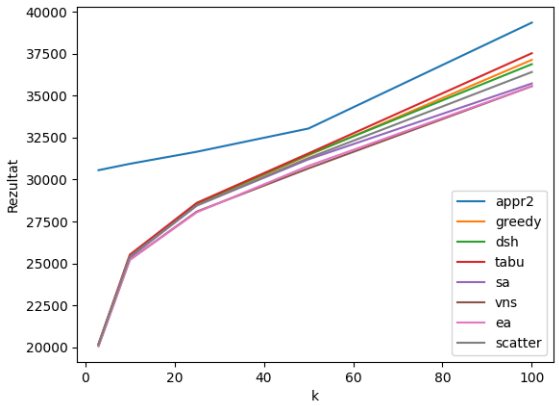
Tabela 2: Pilot testiranje za $n = 20$ i $k = 20$

no testiranje vršeno za male vrednosti n i k kako bi uočili početne odnose i kako se kvalitet rezultata odnosi prema optimalnom rešenju. Kao što je moguće videti čak 3 daju optimalno rešenje. Pristup sa 2-aproksimativnom heuristikom potvrđuje garantovan rezultat koji nije 2 puta lošiji od optimalnog. U nastavku neće biti testiran algoritam grube sile jer vreme izvršavanja prevazilazi razumne okvire. [2]

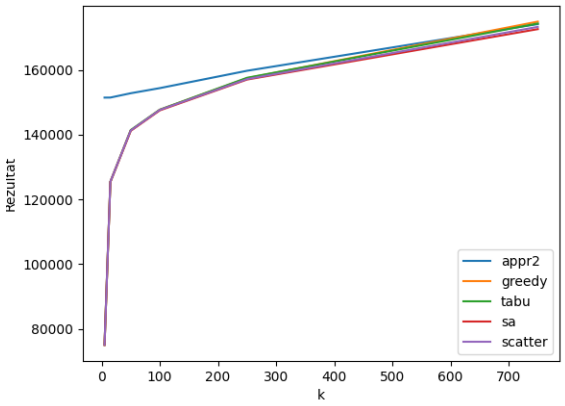
Nakon toga se prešlo na istraživanje kako se algoritmi ponašaju na ulazima srednjih veličina. Rezultate je moguće videti na grafikonima 4 i 5. Jasno se uočava

[2]Procesor na kom je izvršavano testiranje: Intel(R) Core(TM) i3-5005U CPU @ 2.00GHz uz 8gb RAM memorije

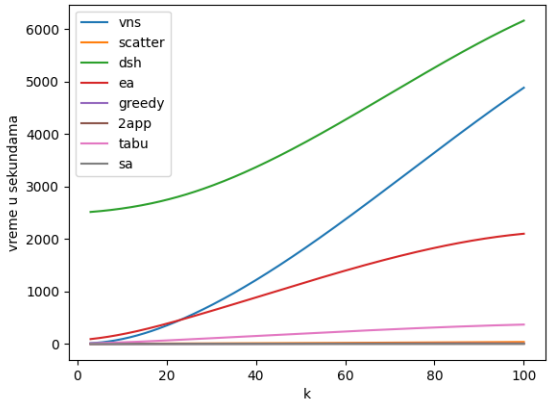
da tek od $k \geq 60$ postoji razlika u kvalitetu rezultata. Takođe, genetski algoritam i vns pokazuju za nijansu bolje rezultate od svih ostalih. U pogledu vremenske zahtevnosti je vidljivo da korišćenje heuristike koja koristi dominirajući skup i VNS metaheuristike, čak i genetskog algoritma, je potpuno nesvrsishodno, te oni neće biti korišćeni u nastavku testiranja za najveće ulaze.



Slika 4: Kvalitet reuzltata za $n = 300$

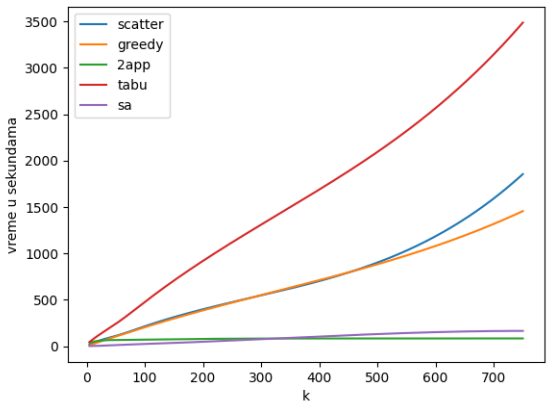


Slika 6: Kvalitet reuzltata za $n = 1500$



Slika 5: Vremenska zahtevnost za $n = 300$

I konačno, za velike ulaze, tačnije ($n = 1500$), na grafiku 7 jasno se uočava vremenska diferencijacija heuristika. 2-aproksimativna heuristika i SA metaheuristika pokazuju se kao jedini razuman pristup za velike ulazne vrednosti. Prednost između ova dva pristupa treba dati SA, na osnovu dobijenih rezultata 6. Koji pokazuje bolje rezultate za manje vrednosti parameta k, dok sa porastom ovog parametra razlika između njih se smanjuje.



Slika 7: Vremenska zahtevnost za $n = 1500$

5 Zaključak

U ovom radu su obrađene tri poznate heuristike, kao i pet mogućih prilagođavanja nekih metaheuristika u cilju rešavanja problema k-centara. Ni za jedan algoritam nije dokazano da daje optimalno rešenje, ali neki algoritmi daju prilično dobro rešenje. Za manje ulaze nije preterano bitno koji pristup se koristi svi će dati dovoljno dobro rešenje, za ulaze iz srednjeg raspona neke pristupe bi trebalo izbeći zbog ogromnog vremenskog utroška. Dok za najveće ulaze, najbolje je iskoristiti pristup Simuliranog kaljenja ili 2-aproksimativnu heuristiku. Zbog manje vremenske zahtevnosti dalje istraživanje bi trebalo okrenuti ka unapređenju parametara Simuliranog kaljenja kao i istraživanje na sličnim heuristikama iz skupa S.

Literatura

[1] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. A John Wiley & Sons, Inc., 2 edition, 2007.

[2] Dorit S. Hochbaum i David B. Shmoys. A Best Possible Heuristic for the k-Center Problem. *Mathematics of Operations Research*, 10:180–184, 1985.

[3] El-Ghazali Talbi. *Metaheuristics, From Desgin To Implementation*. A John Wiley & Sons, Inc., 2009.

[4] Fred Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, pages 533–549, 1986.

[5] Jurij Mihelič, Borut Robic. Solving the k-center Problem Efficiently with a Dominating Set Algorithm. *Journal of Computing and Information Technology*, 2005.

[6] Michael R. Gary i David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1975.

[7] Stefan Mišković. Metoda promenljivih okolina i primena redukovane metodepromenljivih okolina na UFLP, ODABRANA POGLAVLJA OPTIMIZACIJE.

[8] Stefan Mišković. Simulirano kaljenje, RAČUNARSKA INTELIGENCIJA.

[9] Stefan Mišković. Tabu pretraga, ODABRANA POGLAVLJA OPTIMIZACIJE.

[10] Predrag Janičić, Mladen Nikolić. *Veštačka inteligencija*. 2020.

[11] Rafael Marti, Manuel Laguna i Fred Glover. Principles of Scatter Search. 2003.

[12] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, pages 293–306, 1985.

[13] Tomas Feder i Daniel H. Greene. Optimal Algorithms for Approximate Clustering. 1988.