Meta-heuristički algoritmi za problem odabira tima

Jungić, Branimir

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Branimir Jungić

META-HEURISTIČKI ALGORITMI ZA PROBLEM ODABIRA TIMA

Diplomski rad

Voditelj rada:
doc. dr. sc. Goranka Nogo

Zagreb, studeni 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Sadržaj

Uvod

U poslovnom se svijetu sve više zahtijeva dobar timski duh i spremnost na timski rad. U skoro svim poslovima i organizacijama je potrebno podijeliti zaposlenike u timove kako bi što efikasnije obavljali zadatke. Kako bi tim uspješno obavio zadatak nije potrebno samo posjedovanje određenih vještina, već i dobra međusobna suradnja i komunikacija. U ovom radu bit će objašnjen problem odabira tima i način na koji komunikacija među osobama u timu utječe na formiranje tima.

Problem odabira tima je NP-težak problem pa ćemo za njegovo rješavanje koristiti meta-heuristike. Meta-heuristika je heuristika čiji je zadatak usmjeriti heuristike specifične za problem prema području u kojem se nalaze dobra rješenja u prostoru svih rješenja. Drugačije rečeno, koristeći meta-heuristike ne pretražujemo sva moguća rješenja nego samo ona koja su dovoljno dobra ili možda i najbolja. Meta-heuristike odabrane u ovom radu su simulirano kaljenje i pčelinji algoritam.

Pregled rada

U poglavlju 1 je objašnjena notacija koja će se koristiti u ovom radu i formalno je definiran problem odabira tima. U poglavlju 2 su opisani egzaktni algoritmi za rješavanje problema odabira tima i prikazani su pseudokodovi te izvedene složenosti algoritama. U poglavlju 3 su opisane meta-heuristike korištene za rješavanje problema i prikazani su pseudokodovi algoritama. U poglavlju 4 slijedi opis implementacije algoritama i rezultati testiranja. Na kraju rada slijedi zaključak.

Poglavlje 1

Matematičko modeliranje problema

U ovom poglavlju bit će opisana matematička formulacija problema. Prvo ćemo uvesti oznake koje će se koristiti dalje u radu.

1.1 Notacija

Pretpostavimo da imamo n osoba (kandidata) kojima možemo dodijeliti zadatak te sa $O = \{o_j; j = 1, 2, ..., n\}$ označimo skup tih osoba. Neka je $\mathcal{V} = \{v_i; i = 1, 2, ..., m\}$ skup vještina. Svaka osoba ima određene vještine koje označimo skupom $X_i \subseteq \mathcal{V}$. Ako je $v_j \in X_i$ kažemo da osoba i posjeduje vještinu j, a inače kažemo da ju ne posjeduje. Kažemo da skup osoba $\mathcal{T} \subseteq O$ posjeduje vještinu v_j ako barem jedna osoba iz $\mathcal{T}$ posjeduje tu vještinu. Sa $S(v)$ se označava skup osoba koji posjeduju vještinu v. Odnosno, $o_i \in S(v)$ ako i samo ako je $v \in X_i$.

Zadatak Z je podskup skupa vještina, tj. $Z = \{v_1, v_1, \ldots, v_k\} \subseteq \mathcal{V}$, za $k \in \{1, \ldots, n\}$. Ako je $v_j \in Z$ kažemo da je vještina v_j potrebna za zadatak Z, inače kažemo da nije potrebna. Definiramo pokrivač skupa osoba $\mathcal{T}$ u odnosu na zadatak Z kao skup vještina koje su potrebne za zadatak Z, a barem jedna osoba iz $\mathcal{T}$ posjeduje tu vještinu. Koristimo oznaku $P(\mathcal{T}, Z)$ i vrijedi $P(\mathcal{T}, Z) = Z \cap (\cup_{o_i \in \mathcal{T}} X_i)$.

S $T \subseteq O$ označavamo tim koji je odabran za zadatak. Sa $t_i = 1$ označavamo da tim posjeduje i-tu vještinu, a sa $t_i = 0$ da ju ne posjeduje. Dakle, $t = (t_1, t_2, ..., t_m)$. Vještina tražena za određeni zadatak je pokrivena ako barem jedan od članova tima posjeduje tu vještinu.

Veze između osoba možemo prikazati neusmjerenim težinskim grafom $G(O, E)$. Svaki vrh grafa G odgovara jednoj osobi iz O, a svaki brid iz E ima nenegativnu težinu koja označava cijenu komunikacije između dvije osobe. Ako je ta vrijednost niža, to znači da te dvije osobe mogu surađivati efikasnije nego dvije osobe povezane bridom s višom vrijednošću. Bez smanjenja općenitosti možemo pretpostaviti da je graf G povezan, inače

ga možemo lagano dodefinirati da bude povezan - dodamo bridove između nepovezanih dijelova grafa s jako velikim težinama (primjerice, brojem većim od sume svih težina u podgrafu).

Definicija 1.1.1. *Za svaka dva vrha $o_i, o_j \in O$ definiramo funkciju udaljenosti $d : E \to \mathbb{R}^+$, $d(o_i, o_j)$ kao sumu težina po najkraćem putu između vrhova o_i i o_j.*
Funkciju d možemo dodefinirati da računa i udaljenost od vrha o_i do nekog skupa vrhova $\mathcal{T} \subseteq O$ i to na ovaj način:

$$d(o_i, \mathcal{T}) = min_{o_j \in \mathcal{T}} d(o_i, o_j).$$

Funkcija d definirana u definiciji 1.1.1 je metrika.

Definicija 1.1.2. *Za svaka dva vrha $o_i, o_j \in O$ definiramo funkciju $Put(o_i, o_j)$ kao skup vrhova po najkraćem putu između vrhova o_i i o_j.*
$Put(o_i, \mathcal{T})$ definiramo kao skup vrhova koji čine najkraći put od vrha o_i do vrha o_j, gdje je

$$o_j = argmin_{o_j \in \mathcal{T}} d(o_i, o_j).$$

Za graf G i $\mathcal{T} \subseteq O$ s $G[\mathcal{T}]$ označavamo podgraf grafa G induciran vrhovima iz skupa $\mathcal{T}$.

1.2 Definicija problema

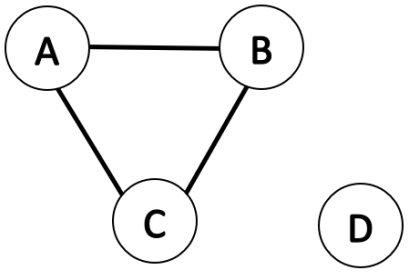
U ovom poglavlju će biti definiran glavni problem kojim se bavi ovaj diplomski rad - problem odabira tima.

Definicija 1.2.1. *Problem odabira tima je za dani skup od n osoba $O = \{o_j; j = 1, ..., n\}$, neusmjereni težinski graf $G(O, E)$ i zadatak Z pronaći $\mathcal{T} \subseteq O$ takav da je $P(\mathcal{T}, Z) = Z$ i da je cijena komunikacije $CK(\mathcal{T})$ minimimalna.*

Cijena komunikacije (CK) može se definirati na više načina, a u ovom radu će biti predstavljeno nekoliko verzija:

- Dijametar (R): Za dani graf $G(O, E)$ i skup osoba $\mathcal{T}$ definiramo cijenu komunikacije kao dijametar pografa $G[\mathcal{T}]$.[1]
- Minimalno razapinjujuće stablo (MRS): Za dani graf $G(O, E)$ i skup osoba $\mathcal{T}$ definiramo cijenu komunikacije kao cijenu minimalnog razapinjujućeg stabla pografa $G[\mathcal{T}]$.[2]

[1] Dijametar grafa je najveća udaljenost najkraćeg puta između bilo koja dva vrha u grafu.
[2] Cijena minimalnog razapinjujućeg stabla je suma težina bridova.



Slika 1.1: Primjer neusmjerenog grafa G.

Primjer 1.2.2. *Promotrimo skup osoba $O = \{A, B, C, D\}$ i skup vještina $\mathcal{V} = \{a, b, c, d, e\}$. Neka je $X_A = \{a, b\}$, $X_B = \{c\}$, $X_C = \{d, e\}$ i $X_D = \{a, d\}$. Na slici 1.1 je prikazan graf $G(O, E)$. Za zadatak $Z = \{a, c\}$ promatramo dva moguća tima: $\mathcal{T}_1 = \{A, B\}$ i $\mathcal{T}_2 = \{B, D\}$. S obzirom da su osobe A i B povezane bridom u grafu G, tim $\mathcal{T}_1$ predstavlja bolje rješenje za zadatak Z, neovisno o težinama.*

Teorem 1.2.3. *Problem odabira tima je NP-težak.*

Dokaz. Dokaz je dan u članku [4]. □

U definiciji problema odabira tima je cilj minimizirati cijenu komunikacije između osoba u timu, ali se može definirati i na drugačiji način. Na primjer, može se zahtijevati da je tim što manji ili kombinirati ova dva zahtjeva.
Također, postoji generalizirana verzija problema u kojoj se svaka vještina pojedinca ocjenjuje nekim brojem, a za zadatak se traži minimalna ocjena koju osoba mora zadovoljavati da može raditi na zadatku. U ovom radu će se razmatrati verzija problema bez ocjenjivanja vještina.

Poglavlje 2

Egzaktni algoritmi

U ovom poglavlju će biti opisani algoritmi za rješavanje problema odabira tima koji ne koriste meta-heuristike. Egzaktni algoritmi uvijek daju optimalno rješenje, ali za NP-teške probleme su primjenjivi samo na relativno male primjerke problema zbog velike računske složenosti. Pokretanje ovih algoritama će svaki put generirati isto rješenje.

2.1 Algoritam koji koristi dijametar

Algoritam 1 Dijametar algoritam

Ulaz: Graf $G(O, E)$; vektor vještina $\{X_1, ..., X_n\}$ i zadatak Z
Izlaz: Tim $\mathcal{T} \subseteq O$ i podgraf $G[\mathcal{T}]$

1: **for** $v \in Z$ **do**
2: $\quad S(v) = \{i \mid v \in X_i\}$
3: $v_{rijedak} \leftarrow \arg\min_{v \in Z} |S(v)|$
4: **for** $i \in S(v_{rijedak})$ **do**
5: $\quad$ **for** $v \in Z \ \ \& \ \ v \neq v_{rijedak}$ **do**
6: $\quad\quad R_{i,v} \leftarrow d(i, S(v))$
7: $\quad R_i \leftarrow min_v R_{i,v}$
8: $i^* \leftarrow \arg\min R_i$
9: $\mathcal{T} = i^* \cup \{Put(i^*, S(v)) \mid v \in Z)\}$

U ovom algoritmu se prvo za svaku vještinu v potrebnu za zadatak Z računa skup indeksa $S(v)$ osoba koje posjeduju tu vještinu. Potom se pronalazi vrijednost $v_{rijedak}$, to jest vještina koju posjeduje najmanje osoba. Za svaki indeks osobe iz $S(v_{rijedak})$ i za svaku vještinu

v potrebnu za zadatak Z koja nije $v_{rijedak}$ se računa udaljenost definirana kao dijametar podgrafa te se bira najmanja vrijednost za svaki indeks osobe.

Na kraju algoritam bira jednu osobu iz skupa $S(v_{rijedak})$ koja će biti dio tima, a koja je po cijeni komunikacije najbliža osobama iz drugih $S(v)$. Pseudokod je prikazan Algoritmom 1.

Složenost ovog algoritma je $O(|S(v_{rijedak})| \cdot k)$. Algoritam je ove složenosti zbog broja računanja udaljenosti od vrha i do skupa $S(v)$ koje se izvršava u dvije ugniježđene *for* petlje (linija 6). Prva *for* petlja se izvršava $|S(v_{rijedak})|$ puta, a druga k puta. U najgorem slučaju (sve osobe posjeduju sve vještine) je $|S(v_{rijedak})| = n$ i $k = n$ te je složenost $O(n^2)$.

2.2 Algoritam koji koristi minimalno razapinjujuće stablo

U ovom algoritmu se koristi funkcija udaljenosti koja koristi minimalno razapinjujuće stablo.

Algoritam PokrivačSteiner dan psudokodom 2 prvo pohlepnim algoritmom[1] računa skup koji pokriva zadatak Z, to jest vraća skup osoba koje imaju vještine potrebne za zadatak. Potom Algoritam SteinerStablo opisan psudokodom 3 računa Steinerovo stablo sa zadanim grafom G i traženim vrhovima O_0. Algoritam SteinerStablo vraća graf (stablo) koji sadrži sve vrhove O_0, a može sadržavati i druge vrhove, i koji ima minimalnu težinu. Algoritam 3 je također pohlepni algoritam koji u Steinerovo stablo dodaje vrh s najmanjom udaljenosti do vrhova koji već čine stablo.

Složenost Algoritma 2 je suma složenosti za algoritme PohlepniPokrivač i SteinerStablo. Algoritam PohlepniPokrivač treba za svaku vještinu $v_i \in Z$ i za svaku osobu $o_j \in O$ provjeriti vrijedi li $o_j \in S(v_i)$ pa je složenost Algoritma PohlepniPokrivač $O(|O| \cdot |Z|)$, a u najgorem slučaju (kada je za zadatak Z potrebno svih m vještina) $O(n \cdot m)$. Algoritam SteinerStablo za svaku osobu $o_i \in O_0 \setminus \mathcal{T}$ prolazi po svim bridovima $e \in E$ pa je složenost Algoritma SteinerStablo $O(|O_0| \cdot |E|)$, a u najgorem slučaju (kada je graf G potpuno povezan) je $|O_0| = n$ i $|E| = n^2$. Konačno, složenost PokrivačSteinerAlgoritma je $O(n^3)$.

Algoritam 2 PokrivačSteinerAlgoritam

Ulaz: Graf $G(O, E)$; vektor vještina $\{X_1, ..., X_n\}$ i zadatak Z

Izlaz: Tim $\mathcal{T} \subseteq O$ i podgraf $G[\mathcal{T}]$

1: $O_0 \leftarrow PohlepniPokrivač(O, Z)$

2: $\mathcal{T} \leftarrow SteinerStablo(G, O_0)$

[1] Pohlepni algoritam tijekom svakog koraka odabire lokalno optimalno rješenje.

Algoritam 3 SteinerStablo

Ulaz: Graf $G(O, E)$; traženi vrhovi $\mathcal{T}$ i Steinerovi vrhovi $O \setminus \mathcal{T}$
Izlaz: Tim $O_0 \subseteq \mathcal{T} \subseteq O$ i podgraf $G[\mathcal{T}]$

1: $\mathcal{T} \leftarrow v$, gdje je v slučajno odabran vrh iz O_0
2: **while** $(O_0 \setminus \mathcal{T}) \neq \emptyset$ **do**
3: $\quad v^* \leftarrow \arg\min_{u \in O_0 \setminus \mathcal{T}} d(u, \mathcal{T})$
4: $\quad$ **if** $Put(v^*, \mathcal{T}) \neq \emptyset$ **then**
5: $\quad\quad \mathcal{T} \leftarrow \mathcal{T} \cup \{Put(v^*, \mathcal{T})\}$

2.3 Poboljšani algoritam koji koristi minimalno razapinjujuće stablo

Ovaj algoritam je sličan prethodnom, s razlikom da graf G proširimo do grafa H na sljedeći način: neka je zadatak $Z = \{z_1, ..., z_k\}$. Grafu G dodamo vrhove Y_j za svaki $z_j \in Z$. Svaki novi vrh Y_j je spojen s vrhovima o_i za koje vrijedi $z_j \in X_i$, to jest sa svakom osobom koja ima tu vještinu. Težinu na svim novim bridovima postavimo na neki veliki broj, primjerice broj veći od sume svih težina na grafu G. Također je i svaki vrh o_i zamijenjen s klikom K_i veličine $|X_i|$. Svaki vrh klike K_i predstavlja jednu vještinu osobe o_i i spojen je sa svim vrhovima s kojima je i o_i spojen u grafu G. Vrijednost težina između vrhova klike K_i je 0. Na tako dobivenom grafu H se poziva Algoritam SteinerStablo te se iz rezultata izbacuju prethodno dodani $\{Y_1, ..., Y_k\}$.

PoboljšaniSteinerAlgoritam je iste složenosti kao Algoritam SteinerStablo, $O(k \cdot |E|)$. U najgorem slučaju (za zadatak Z je potrebno svih m vještina i graf G je potpuno povezan) je složenosti $O(m \cdot n^2)$.

Algoritam 4 PoboljšaniSteinerAlgoritam

Ulaz: Graf $G(O, E)$; vektor vještina $\{X_1, ..., X_n\}$ i zadatak Z
Izlaz: Tim $\mathcal{T} \subseteq O$ i podgraf $G[\mathcal{T}]$

1: $H \leftarrow PoboljsaniGraf(G, Z)$
2: $O_H \leftarrow SteinerStablo(H, \{Y_1, ..., Y_k\})$
3: $\mathcal{T} \leftarrow O_H \setminus \{Y_1, ..., Y_k\}$

Poglavlje 3

Meta-heuristike

S obzirom da je vrednovanje svih mogućih kombinacija timova vremenski eksponencijalno složen problem, morat ćemo na drugi način pronaći optimalno rješenje. Optimalno rješenje je tim koji zadovoljava dani zadatak Z, no nije nužno najbolje moguće, ali je dovoljno blizu najboljeg rješenja. Za pronalazak ovakvog rješenja koristit će se meta-heuristike, i to simulirano kaljenje i pčelinji algoritam, koje će biti opisane u ovom poglavlju.

3.1 Simulirano kaljenje

Simulirano kaljenje je heuristika pogodna za rješavanje optimizacijskih matematičkih problema. Inspiracija za algoritam simuliranog kaljenja je postupak kaljenja metala u metalurgiji. Prilikom kaljenja, metal se zagrijava na visoke temperature te postepeno hladi kako bi se u metalu stvorile pravilne kristalne strukture bez deformacija. Brzim hlađenjem dolazi do stvaranja nepravilnih struktura i brzog pucanja metala. Analogno radi algoritam simuliranog kaljenja koji će biti opisan u ovom poglavlju.

Ukratko ćemo opisati općeniti algoritam, a zatim dijelove algoritma prilagođene rješavanju problema odabira tima. Simulirano kaljenje je dobar algoritam za rješavanje problema s diskretnim prostorom rješenja, poput problema trgovačkog putnika, pa je zato odabran za rješavanje problema odabira tima.

Općenito, simulirano kaljenje se sastoji od sljedećih komponenti:

Kandidat za rješenje je trenutno rješenje koje se kroz iteracije poboljšava.

Funkcija energije rješenja je "funkcija kazne"[1] koja određuje kvalitetu rješenja - veća vrijednost, lošije rješenje. Simulirano kaljenje nastoji naći rješenje sa što manjom energijom.

Funkcija susjedstva je funkcija koja nam daje sljedeće moguće rješenje, a koje ovisi o trenutnom rješenju. Idući kandidat za rješenje se traži u susjedstvu trenutnog rješenja, tj. sa sličnom energijom kao i trenutno rješenje.

Funkcija prijelaza određuje hoće li se iduće rješenje koje daje funkcija susjedstva prihvatiti umjesto trenutnog rješenja. Ako je energija sljedećeg mogućeg rješenja manja, to rješenje se prihvaća kao trenutno i nastavljaju se iteracije. Ako energija nije manja, novo rješenje se ipak može prihvatiti, uz neku vjerojatnost, kako bi se izbjegao lokalni optimum.

Funkcija hlađenja postepeno smanjuje temperaturu kako bi se lošija rješenja prihvaćala s manjom vjerojatnošću. Algoritam započinje s jako visokom temeperaturom.

Problem odabira tima

U slučaju problema odabira tima **kandidat za rješenje** je tim T koji je sastavljen od onoliko osoba koliko je vještina potrebno za zadatak (za svaku vještinu po jedna osoba, čak i ako jedna osoba ima više traženih vještina).

Energija rješenja za tim $\mathcal{T}$ se računa sljedećom formulom:

$$E_{\mathcal{T}} = \sum_{\varepsilon_{\mathcal{T}}} w(e_{ij}) + (\frac{|\mathcal{T}| \cdot (|\mathcal{T}| - 1)}{2} - |\varepsilon_{\mathcal{T}}|) \cdot \alpha \cdot max(w(e))$$

gdje je $\varepsilon_{\mathcal{T}}$ skup svih bridova u podgrafu $G[\mathcal{T}]$, a $max(w(e))$ najveća (najlošija) težina u grafu G. S ciljem minimizacije energije rješenja $E_{\mathcal{T}}$ zbrojimo vrijednosti težina u podgrafu koji čini trenutno rješenje i tome dodamo penale za svaki brid koji ne postoji među osobama u trenutnom timu. Parametrom α kontroliramo koliko će nepostojeći bridovi utjecati na ukupnu energiju rješenja. Ako je α bliži nuli tada će rješenje s više nepostojećih bridova imati sličnu energiju potpuno povezanom podgrafu, a što je α veći to će rješenje koje je potpuno povezan graf imati manju energiju.

[1] U optimizacijskim problemima se koristi izraz "funkcija dobrote" koja određuje kvalitetu rješenja. Funkcija dobrote se nastoji maksimizirati, a u slučaju simuliranog kaljenja se funkcija energije minimizira pa se koristi izraz "funkcija kazne". [6]

Funkcija susjedstva za jednu slučajno odabranu vještinu v bira novu osobu iz skupa $S(v)$ i tako dobiva novo potencijalno rješenje.

Funkcija prijelaza novo rješenje prihvaća ako je energija tog rješenja manja od energije kandidata za rješenje. Lošije rješenje se prihvaća s vjerojatnošću definiranom idućom formulom:

$$p = e^{\frac{\Delta_{energija}}{temp}}$$

gdje $\Delta_{energija}$ predstavlja razliku energija trenutnog i novog rješenja, $temp$ trenutnu temperaturu, a e je Eulerova konstanta. Prijelaz na lošije rješenje je moguć uz visoku temperaturu i malu razliku energija rješenja. Ovakvim pristupom se na početku prostor rješenja pretražuje ugrubo dok je temperatura visoka, a kad se temperatura dovoljno snizi dolazi do fine pretrage u okolici najboljeg trenutnog rješenja.

Funkcija hlađenja ovisi o parametru $r_{hladenja}$ na ovaj način:

$$temp_n = r_{hladenja} \cdot temp_{n-1}.$$

Algoritam 5 prikazuje kako simulirano kaljenje kroz iteracije traži rješenje sa što manjom energijom. Algoritam traži novo rješenje određen broj iteracija zadan varijablom $maxIteracija$ ili dok se temperatura $temp$ nije spustila na 0.

Algoritam 5 Simulirano kaljenje

Ulaz: Graf $G(O, E)$; vektor vještina $\{X_1, ..., X_n\}$ i zadatak Z
Izlaz: Tim $\mathcal{T} \subseteq O$

```
1:  T ← izracunajNovoRjesenje(pocetnaTemp)
2:  temp ← pocetnaTemp
3:  iteracija ← 0
4:  energija ← izracunajEnergiju(T)
5:  while temp > 0 & iteracija < maxIteracija do
6:      novoRjesenje ← izracunajNovoRjesenje(T, temp)
7:      novaEnergija ← izracunajEnergiju(novoRjesenje)
8:      Δ_energija ← energija − novaEnergija
9:      if funkcijaPrijelaza(Δ_energija, novoRjesenje, temp) then
10:         T ← novoRjesenje
11:         energija ← novaEnergija
12:     temp ← izracunajTemperaturu(temp)
13:     iteracija + +
```

3.2 Pčelinji algoritam

Pčelinji algoritam je prirodom inspirirana meta-heuristika koja se koristi kod optimizacijskih problema. Inspiracija dolazi od pčela, njihovog načina traženja izvora hrane te prenošenja informacija o kvaliteti izvora plesom. Algoritam koristi pretraživanje susjedstva rješenja i nasumično pretraživanje cijelog prostora rješenja, kako bi se izbjegao lokalni optimum. Za razliku od simuliranog kaljenja, koje u danom trenutku ima jednog kandidata za rješenje, pčelinji algoritam koristi populaciju rješenja.

Parametar	Oznaka
Veličina populacije rješenja	n_{PA}
Broj najboljih rješenja	m_{PA}
Broj elitnih rješenja	e_{PA}
Broj pčela oko elitnih rješenja	nep
Broj pčela oko neelitnih najboljih rješenja	nsp
Veličina susjedstva	ngh
Broj iteracija	$maxIter$

Tablica 3.1: Tablica parametara pčelinjeg algoritma

Tablicom 3.1 su prikazani parametri koji se koriste u pčelinjem algoritmu. Pseudokod pčelinjeg algoritma je dan Algoritmom 6. U jednoj iteraciji proučava se n_{PA} rješenja. Za m_{PA} najboljih rješenja se u susjedstvu veličine ngh traže slična rješenja te se prihvaćaju ako su bolja. Za e_{PA} elitnih rješenja se traži nep drugih rješenja u susjedstvu, dok se za neelitna rješenja traži nsp rješenja. Rješenja koja nisu najbolja se brišu i na njihova mjesta dolaze nasumično odabrana rješenja iz cijelog prostora pretraživanja. Uvođenje ovakve slučajnosti osigurava izbjegavanje lokalnog optimuma. Na kraju svake iteracije se populacija rješenja sortira po vrijednosti funkcije dobrote, koja se maksimizira. Algoritam se izvršava $maxIter$ iteracija.

U slučaju problema odabira tima rješenje predstavlja tim $\mathcal{T}$ koji je sastavljen od onoliko osoba koliko vještina je potrebno za zadatak. Veličina susjedstva ngh predstavlja za koliko se vještina v_i može osoba iz rješenja koja posjeduje vještinu v_i zamijeniti novom osobom iz $S(v_i)$. Funkcija dobrote se računa po formuli

$$f_{dobrote}(\mathcal{T}) = \sum_{\varepsilon_{\mathcal{T}}} \frac{1}{w(e_{ij})} - (\frac{|\mathcal{T}| \cdot (|\mathcal{T}| - 1)}{2} - |\varepsilon_{\mathcal{T}}|) \cdot \beta \cdot max(\frac{1}{w(e)}),$$

gdje je $\varepsilon_{\mathcal{T}}$ skup svih bridova u podgrafu $G[\mathcal{T}]$, a $max(w(e))$ najveća (najlošija) težina u grafu G. Funkciju dobrote nastojimo maksimizirati pa sumiramo recipročne vrijednosti

Algoritam 6 Pčelinji algoritam

Ulaz: Graf $G(O, E)$; vektor vještina $\{X_1, ..., X_n\}$ i zadatak Z
Izlaz: Tim $\mathcal{T} \subseteq O$

```
1: while iteracija < maxIteracija do
2:     iteracija++
3:     Odredi elitna i neelitna najbolja rješenja za lokalno traženje
4:     Pronađi nova rješenja u susjedstvu elitnih i neelitnih najboljih rješenja
5:     Izračunaj funkciju dobrote za dobivena rješenja
6:     Sortiraj rješenja po dobroti
7:     Pronađi nova slučajna rješenja za rješenja koja nisu najbolja
8:     Izračunaj funkciju dobrote za dobivena rješenja
9:     Sortiraj rješenja po dobroti
```

težina bridova u podgrafu $G[\mathcal{T}]$ i oduzimamo recipročnu vrijednost najveće težine u grafu G onoliko puta koliko bridova nedostaje podgrafu $G[\mathcal{T}]$ da bi bio potpun graf. Parametrom β reguliramo koliko će penalizacija za nepostojeće bridove utjecati na funkciju dobrote. Što je parametar β bliži 0 to manje utječe na funkciju dobrote, to jest, manje se penaliziraju nepostojeći bridovi, a što je β veći to će potpuni graf imati veću vrijednost funkcije dobrote.

Poglavlje 4

Implementacija i testiranje algoritama

Algoritmi iz poglavlja 2 i 3 će biti implementirani u jeziku C++ i testirani na dva skupa podataka. Prvi skup podataka je umjetno generiran na slučajan način, a drugi skup je dobiven interpretacijom stvarnih podataka.

4.1 Implementacija

Za implementaciju grafova koji se koriste u algoritmima se koristi *open source* biblioteka *Boost Graph Library*[1] (BGL), verzije 1.68, objavljena u kolovozu ove godine.
Graf G je predstavljen novodefiniranim tipom `UndirectedGraph` koji je definiran kao `adjacency_list < vecS, vecS, undirectedS, Person, Weight >`.
`adjacency_list` je klasa iz BGL-a, `vecS` označava da se za pohranu vrhova i bridova grafa koristi vektor. Vektor je odabran za pohranu bridova i vrhova jer po dokumentaciji BGL-a takvo pohranjivanje zahtijeva manje prostora te je za inkrementiranje iteratora za bridove i vrhove brže od ostalih odabira spremnika. `undirectedS` označava da je graf neusmjeren. `Person` je struktura koja predstavlja osobu o koja ima varijable `ime` tipa string i `vjestine` tipa vector. `Weight` koristimo za dodjeljivanje težina bridovima.
Za računanje udaljenosti i puta između dva vrha ili između vrha i skupa se koristi Dijkstrin algoritam pozivom funkcije `dijkstra_shortest_paths`. Dijkstrin algoritam vraća udaljenosti do svakog vrha od zadanog vrha v koje koristimo kao vrijednosti funkcije $d(v,\cdot)$ definirane u 1.1.1 te put od svakog vrha do vrha v koji koristimo kao vrijednost funkcije $Put(v,\cdot)$ definirane u 1.1.2.

Testiranja algoritama su provedena na računalu s Intel i5 procesorom frekvencije radnog takta 2.50GHz i 8GB radne memorije.

[1] Biblioteka *Boost Graph Library* je preuzeta sa `https://www.boost.org/doc/libs/1_66_0/libs/graph/doc/index.html`.

4.2 Slučajno generirani graf

Za potrebe testiranja algoritama ćemo generirati graf s 500 osoba i 60 vještina. Na slučajan način ćemo svakoj osobi dodijeliti neke vještine i odabrati susjede. Svakom bridu ćemo pridružiti težinu između 0 i 1, gdje 1 predstavlja lošiju komunikaciju između dvije osobe, a 0 najbolju komunikaciju. Za generiranje slučajnih brojeva će se koristiti Marsenne Twister pseudoslučajni generator brojeva `std::mt19937`.

4.3 StackExchange graf

Za simulaciju timova i vještina iz realnog svijeta ćemo koristiti skup podataka s foruma StackExchange [1] koji je javno dostupan u obliku arhive. StackExchange je forum na kojem ljudi postavljaju pitanja o raznim problemima vezanim uz programiranje ili raspravljaju o rješenjima.

Skup podataka koji je preuzet sa [3] sadrži statuse objavljene u 50 različitih podforuma koji će u našem slučaju predstavljati vještine. Broj korisnika koji su objavljivali na forumu je 54099, a oni će predstavljati skup osoba.

Podaci se nalaze u xml formatu, a za potrebe testiranja su korištene datoteke *Posts.xml* i *Users.xml*. Za otvaranje tih datoteka korištena je *open source* biblioteka *TinyXML-2*[2].

Na forumu je moguće odgovarati na tuđe statuse. Osobu koja je postavila status i osobu koja je odgovorila na taj status ćemo povezati bridom u grafu G. Težina tog brida će ovisiti koliko su si puta te dvije osobe međusobno odgovarale na statuse. S obzirom da koristimo neusmjereni graf jednako ćemo promatrati kada osoba A odgovori osobi B kao i kada osoba B odgovori osobi A. Svakoj osobi ćemo pridužiti one vještine koje odgovaraju podforumima u kojem je objavila status kao pitanje ili kao odgovor. U grafu G ukupno ima 105810 bridova.

4.4 Rezultati testiranja

Svaki algoritam ćemo testirati na slučajno generiranom grafu i na StackExchange grafu. Vrijeme izvršavanja algoritma ćemo mjeriti u sekundama kao prosjek 10 izvršavanja. Tijekom svakog izvođenja algoritmi će tražiti optimalni tim za drugačiji, slučajno generirani, zadatak. Veličina zadatka će varirati između 6 i 15 kako bi se testiralo utječe li veličina zadatka na dobivene rezultate.

[2]Biblioteka *TinyXML-2* je preuzeta sa `http://www.grinninglizard.com/tinyxml2/`.

Kao parametre po kojima ćemo uspoređivati kvalitetu rješenja uzimamo funkciju energije iz Algoritma simuliranog kaljenja, funkciju dobrote iz Pčelinjeg algoritma i sumu težina koja je slična funkciji energije iz Algoritma simuliranog kaljenja, bez penalizacije za nepostojeće bridove ($\alpha = 0$). Sva tri parametra ćemo računati kao prosjek 10 izvršavanja.

Vrijednosti parametara za Algoritam simuliranog kaljenja i Pčelinji algoritam su određeni empirijski. Za Algoritam simuliranog kaljenja su sljedeći:

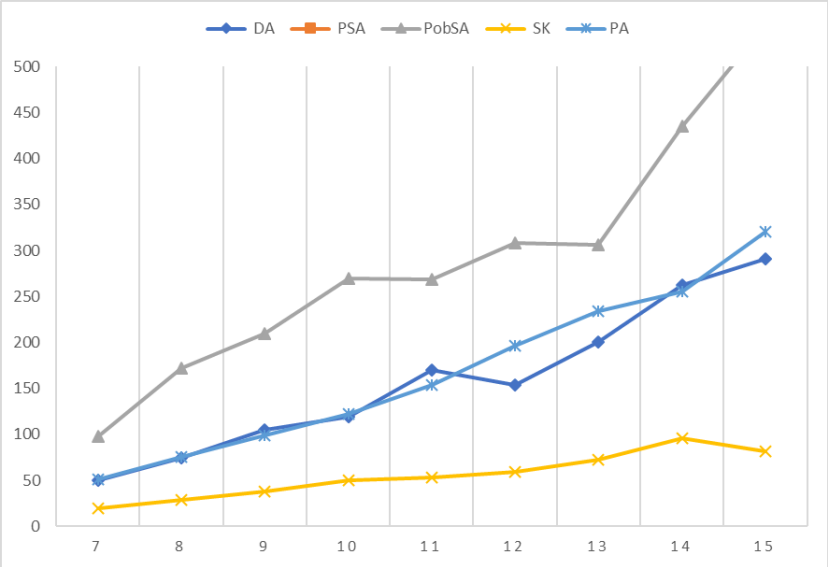
$$maxIteracija = 1000, pocetnaTemp = 1000, \alpha = 4, r_{hladenja} = 0.98;$$

a za Pčelinji algoritam:

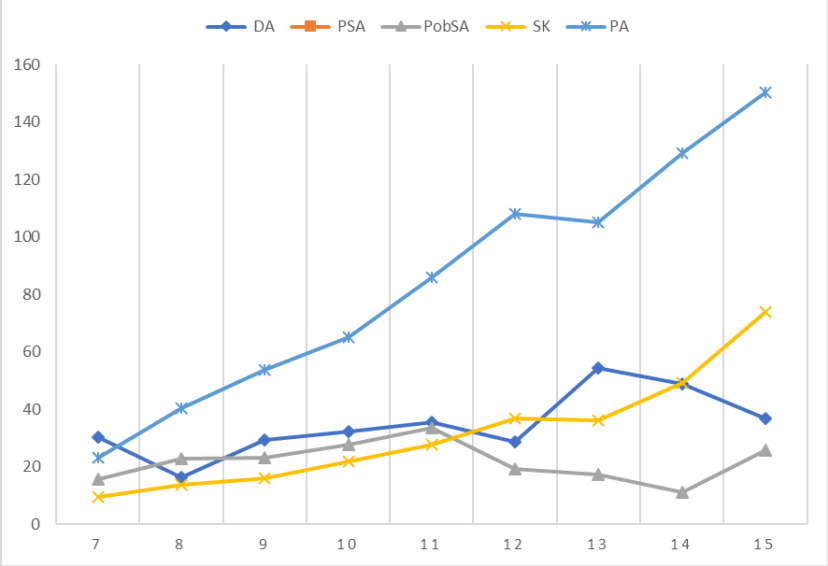
$$maxIteracija = 1000, n_{PA} = 10, m_{PA} = 5, e_{PA} = 2, nep = 5, nsp = 3, ngh = 2, \beta = 3.$$

Grafovi na slikama 4.1[3], 4.2, 4.3 i 4.4 su rezultati testiranja na slučajno generiranom i StockExchange grafu. Prikazani su rezultati testiranja Dijametar algoritma (DA, tamnoplava boja), PokrivačSteinerAlgoritma (PSA, narančasta boja), PoboljšanogSteinerAlgoritma (PobSA, siva boja), Algoritma simuliranog kaljenja (SK, žuta boja) i Pčelinjeg algoritma (PA, svjetloplava boja). Radi preglednosti, odabrali smo grafički, a ne numerički prikaz rezultata testiranja.
Testiranje je ukupno trajalo 6 sati.

[3]Vrijednosti funkcije energije za PokrivačSteinerAlgoritam su jako velike pa su radi preglednosti izostavljene s grafova na slici 4.1.

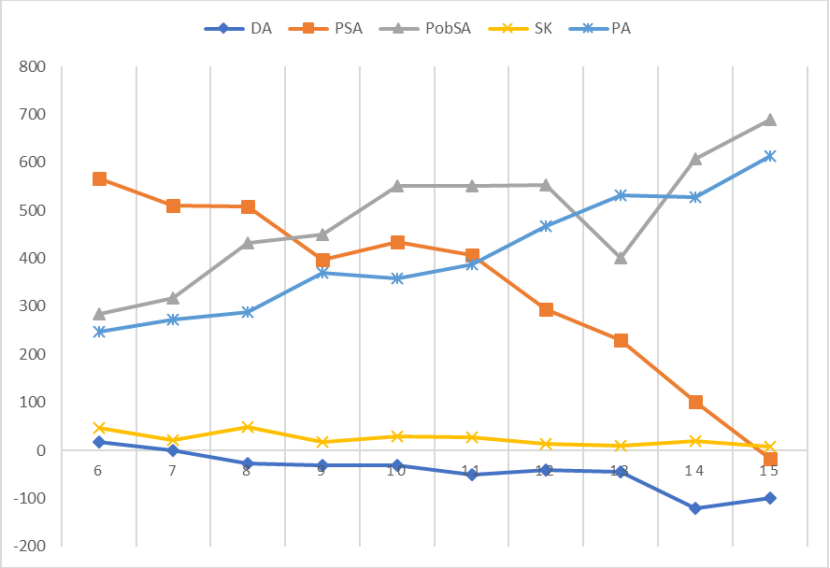


(a) slučajno generirani graf

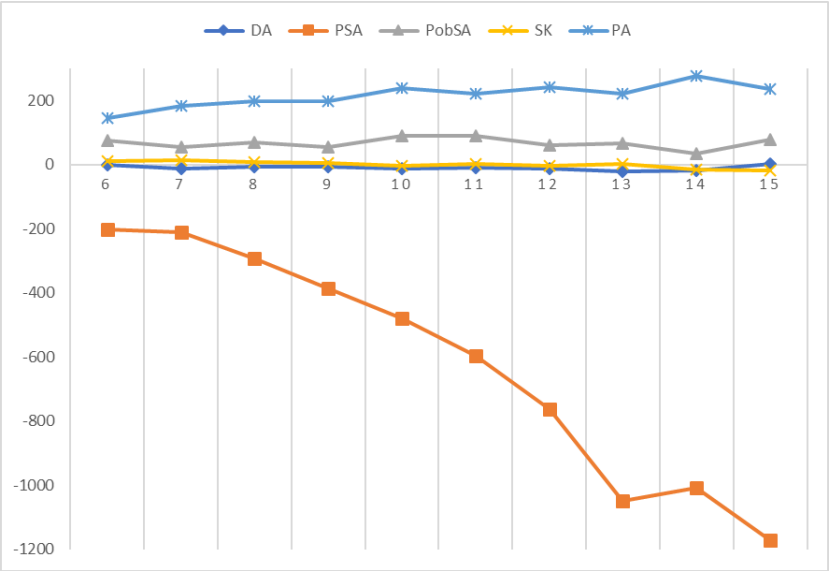


(b) StackExchange graf

Slika 4.1: Grafovi ovisnosti vrijednosti funkcije energije o veličini zadatka.



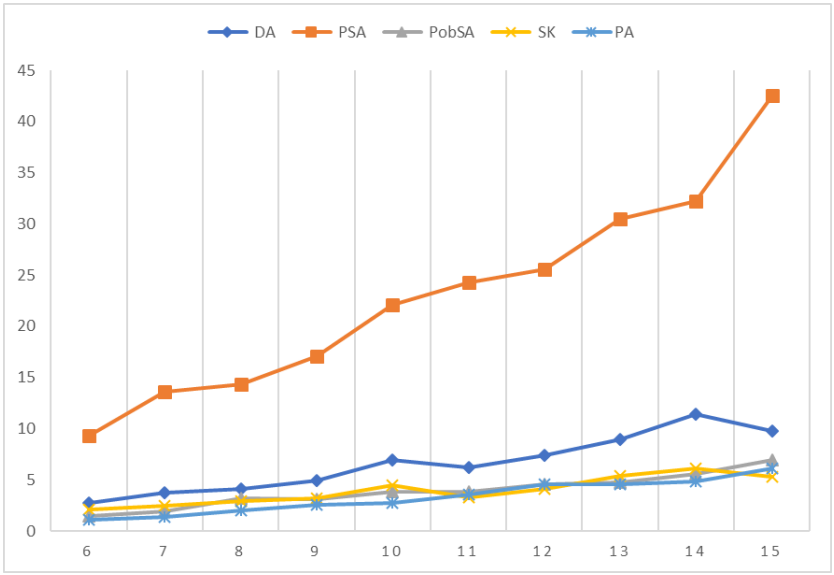
(a) slučajno generirani graf



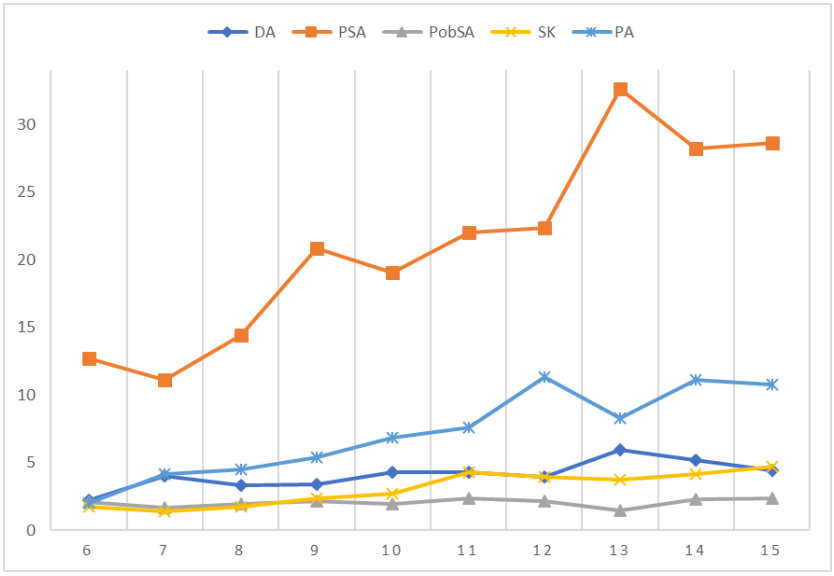
(b) StackExchange graf

Slika 4.2: Grafovi ovisnosti vrijednosti funkcije dobrote o veličini zadatka.

Kao što je vidljivo s grafova na slikama 4.1 i 4.2 algoritam s najboljom (najnižom) vrijednosti funkcije energije je Algoritam simuliranog kaljenja, čiji cilj i je minimiziranje te funkcije, dok je algoritam s najboljom (najvišom) vrijednosti funkcije dobrote Pčelinji algoritam koji ju maksimizira.

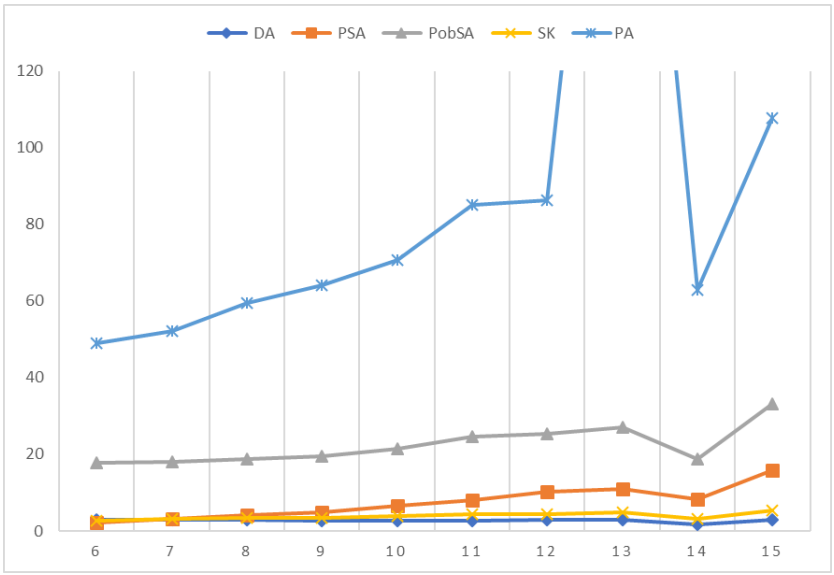


(a) slučajno generirani graf

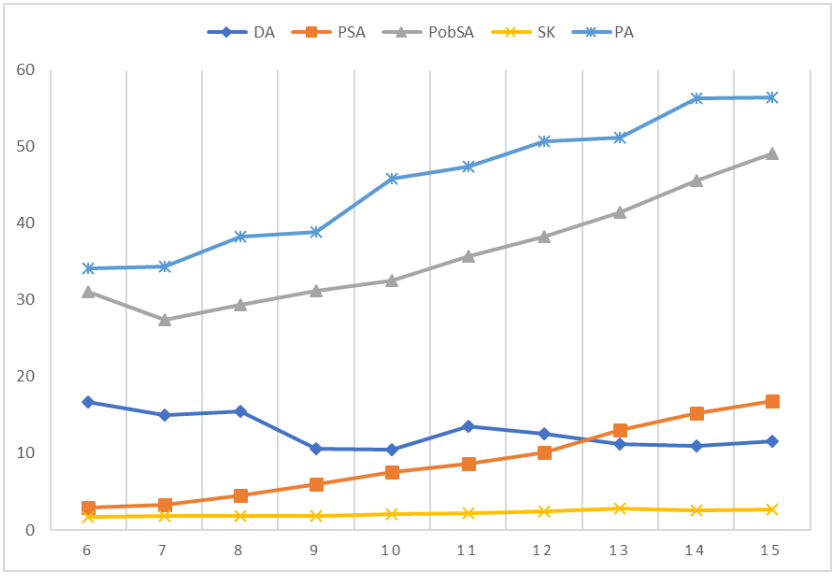


(b) StackExchange graf

Slika 4.3: Grafovi ovisnosti sume težina podgrafa $G[\mathcal{T}]$ o veličini zadatka.



(a) slučajno generirani graf



(b) StackExchange graf

Slika 4.4: Grafovi ovisnosti vremena izvršavanja algoritma o veličini zadatka.

PoboljšaniSteinerAlgoritam se pokazao kao najbolji egzaktni algoritam po sva tri parametra usporedbe, ali se vremenski najduže izvodio. Razlog tome je proširivanje grafa. Velika mana ovog algoritma je što ponekad nije mogao naći rješenje. Još jedna mana ovog algoritma i PokrivačSteinerAlgoritma je ta što pronalaze timove velike kardinalnosti. U pseudokodu se vidi da u tim $\mathcal{T}$ dodaju sve čvorove koji se nalaze na putu od trenutno najbolje osobe v^* do tima $\mathcal{T}$. Samim time što pronalaze velike timove i suma težina raste.

Velika prednost algoritama koji koriste meta-heuristike je ta što pronalaze timove manje kardinalnosti, čak manje i od kardinalnosti zadatka. Razlog tome je što neke osobe imaju više vještina pa će algoritmi imati bolju funkciju energije (odnosno funkciju dobrote) ako dodaju tu osobu u tim. Manjem timu je potrebno manje bridova do potpunog grafa i u konačnici imaju manju penalizaciju za nepostojeće bridove.

S obzirom da Algoritam simuliranog kaljenja i Pčelinji algoritam ne možemo uspoređivati po funkciji energije ili funkciji dobrote, usporedit ćemo ih po sumi težina i vremenu izvođenja. Simulirano kaljenje se kraće izvodi, a suma težina je podjednaka; u slučaju testiranja na StackExchange grafu ide u korist simuliranog kaljenja. Duže vrijeme izvođenja pčelinjeg algoritma se može objasniti načinom na koji algoritam traži rješenje: puno više puta računa funkciju dobrote, tijekom svake iteracije čak $nep \cdot e_{PA} + nsp \cdot (m_{PA} - e_{PA})$ puta.

Poglavlje 5

Zaključak

U ovom radu je predstavljeno 5 algoritama koji rješavaju problem odabira tima. Nakon testiranja na dva skupa podataka može se zaključiti kako je algoritam koji koristi meta-heuristiku simulirano kaljenje najpogodniji za rješavanje ovog problema. U usporedbi s Pčelinjim algoritmom, Algoritam simuliranog kaljenja je lakši za implementaciju jer ima manje parametara i brže se izvodi, no oba algoritma imaju podjednake rezultate.

Kao eventualno poboljšanje algoritma parametri $maxIteracija$ i $pocetnaTemperatura$ mogu se postaviti na više vrijednosti. Također bi se mogla poboljšati funkcija susjedstva. Algoritam predložen u ovom radu koristi slučajno traženje sljedećeg rješenja, a moguće poboljšanje je traženje susjeda sa sličnom energijom.

Algoritmi i pristup problemu prikazani u ovom radu bi se mogli koristiti u stvarnom svijetu, jedino još preostaje odrediti način generiranja grafa G i određivanje težina bridova u grafu. Odluka je na organizaciji koja formira timove, a može ovisiti o hijerarhiji unutar organizacije ili o kvaliteti rada timova na prijašnjim projektima.

Bibliografija

[1] *Stack Exchange*, https://stackexchange.com/.

[2] C. Dorn i S. Dustdar, *Composing Near-Optimal Expert Teams: A Trade-Off between Skills and Connectivity*, On the Move to Meaningful Internet Systems: OTM 2010 (Berlin, Heidelberg), Springer Berlin Heidelberg, 2010, str. 472–489.

[3] Stack Exchange Inc., *Stack Exchange Data Dump*, 2018., https://archive.org/details/stackexchange.

[4] T. Lappas, K. Liu i E. Terzi, *Finding a Team of Experts in Social Networks*, Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA), KDD '09, ACM, 2009, str. 467–476, http://doi.acm.org/10.1145/1557019.1557074.

[5] D. Mallenahalli Shankaralingappa, *Outsource or Train: A team formation problem*, 2016., https://users.ics.aalto.fi/mallend1/Master_Thesis.pdf.

[6] M. Čupić, *Prirodom inspirirani optimizacijski algoritmi. Metaheuristike.*, 2013, http://java.zemris.fer.hr/nastava/pioa/knjiga-0.1.2013-12-30.pdf.

Sažetak

U ovom je radu predstavljen problem odabira tima. Opisani su egzaktni algoritmi kojima se pokušalo naći rješenje problema. S obzirom da je problem NP-težak rješavan je i algoritmima koji koriste meta-heuristike. Meta-heuristike odabrane za rješavanje problema u ovom radu su simulirano kaljenje i pčelinji algoritam. Opisana je implementacija i tijek testiranja algoritama te su prikazani rezultati. Usporedbom rezultata je zaključeno da je simulirano kaljenje najbolji pristup za rješavanje ovog problema, a navedena su i moguća poboljšanja algoritma.

Summary

The team formation problem was presented in this thesis. Exact algorithms used to find a solution to the problem were described. Since the problem is NP-hard, it was also solved with algorithms that use meta-heuristics. Meta-heuristics chosen to solve the problem in this paper are simulated annealing and bees algorithm. The implementation and algorithm testing process were described and their results were shown. By comparison of the results it is concluded that simulated annealing is the best approach to solving this problem and possible algorithm improvements were given.

Životopis

Branimir Jungić rođen je u Sisku 24.1.1995. godine gdje završava osnovnu školu. U Zagrebu upisuje XV. gimnaziju te potom 2013. godine preddiplomski sveučilišni studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Nakon završetka preddiplomskog studija 2016. godine upisuje diplomski sveučilišni studij Računarstvo i matematika na istom fakultetu. Tijekom svog akademskog obrazovanja je bio član-mentor studentske udruge Mladi nadareni matematičari "Marin Getaldić" i volonter-predavač na projektu RADDAR. Tijekom zadnje godine studija je radio kao demonstrator u računalnim praktikumima na fakultetu.