Optimalno uparivanje u proizvoljnom grafu
- The blossom algorithm -

Seminarski rad u okviru kursa
Konstrukcija i analiza algoritama 2,
Matematički fakultet

Nevena Mijailović, 1067/2023,
Marija Bogavac, 1068/2023,
nevena.mijailovic000@gmail.com,
marijabogavac001@gmail.com

5. april 2024.

Sažetak

U domenu teorije grafova, koncept optimalnog uparivanja igra ključnu ulogu u različitim događajima u stvarnom svetu, kao što je uparivanje ljudi za ples na žurci, uparivanje mentora sa studentima ili usklađivanje zadataka sa radnicima u distribuiranim sistemima.

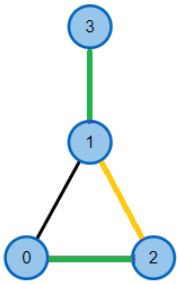
Esej počinje razjašnjavanjem teorijske osnove uparivanja u grafovima, zatim se uvodi *Blossom* algoritam i njegovo poređenje sa algoritmima namenjenim rešavanju istog problema. Štaviše, razmatraju se računarski aspekti *Blossom* algoritma, uključujući njegovu vremensku složenost i praktična razmatranja implementacije.

Konačno, esej se završava naglašavanjem značaja optimalnog uparivanja u teoriji grafova i nezamenljive uloge koju igra *Blossom* algoritam u efikasnom rešavanju takvih problema. Naglašava se svestranost i primenljivost algoritma, naglašavajući njegovu trajnu relevantnost u savremenim računarskim i algoritamskim metodologijama.

Sadržaj

1 Problem optimalnog uparivanja

Za zadati neusmereni graf $G = (V, E)$ uparivanje je skup grana koje nemaju zajedničke čvorove.
Optimalno uparivanje je uparivanje sa maksimalnim brojem grana.[7]
Maksimalno uparivanje je uparivanje koje se ne može proširiti dodavanjem nove grane.
Svako optimalno uparivanje je maksimalno, ali nije svako maksimalno uparivanje optimalno 1.



Slika 1: Primer maksimalnog (žuta boja) i optimalnog uparivanja (zelena boja) u grafu

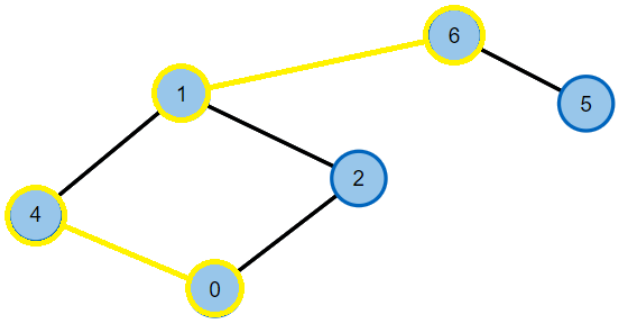
Bipartitni graf je graf čiji se čvorovi mogu podeliti na dva disjunktna podskupa tako da u grafu postoje samo grane između čvorova iz različitih podskupova.

Edmondsov *Blossom* algoritam izračunava optimalno uparivanje u opštem grafu. Za razliku od mnogih drugih algoritama, graf ne mora biti bipartitan. On proširuje ideju Hopkroft-Karp algoritma, koji izračunava optimalno uparivanje za bipartitni graf, tako što tretira neparne cikluse na odgovarajući način.

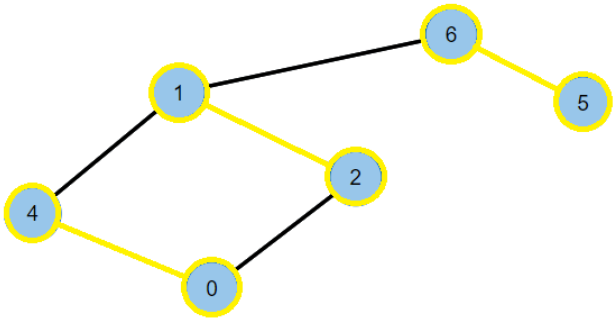
2 Rešenje problema optimalnog uparivanja za bipartitne grafove

Ukratko rekapituliramo osnovne koncepte Hopkroft-Karp algoritma koji su takođe relevantni za Edmondsov *Blossom* algoritam.
Da bismo poboljšali dato uparivanje, pokušavamo da pronađemo alternirajući put.
Alternirajući put je putanja koja počinje slobodnim (neuparenim) čvorom, završava se neuparenim čvorom i naizmenično se smenjuju grane koje nisu i jesu u uparivanju.
Ako smo pronašli alternirajući put, možemo poboljšati trenutno uparivanje tako što ćemo invertovati grane duž putanje: one koje nisu bile u uparivanju, sada će biti u uparivanju, a one koje jesu, neće biti u uparivanju. Time povećavamo kardinalnost uparivanja za 1.

Na slikama 2 i 3 prikazan je primer pronalaska alternirajućeg puta polazeći od maksimalnog uparivanja. Čvorovi 2 i 5 su neupareni i možemo pronaći alternirajući put između njih (2-1-6-5). Dobili smo novo uparivanje koje ima jednu granu više od prethodnog uparivanja.



Slika 2: Maksimalno uparivanje u grafu



Slika 3: Optimalno uparivanje dobijeno pronalaskom alternativnog puta i invertovanjem grana

Teorema 1. *(O alternirajućem putu) Uparivanje je optimalno ako i samo ako u odnosu na njega ne postoji alternirajući put.*

Prema tome, ako više ne možemo da pronađemo alternirajući put u grafu, uparivanje mora biti optimalno.

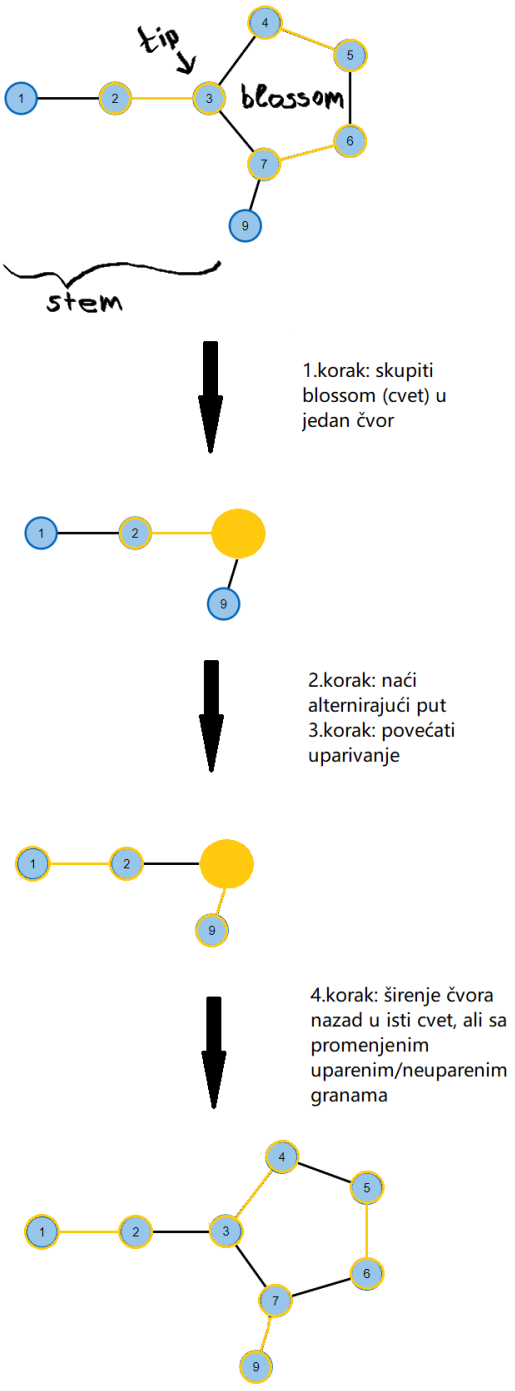
Kako možemo pronaći alternirajuće putanje u grafu?

Prvo biramo proizvoljan slobodni čvor r i odatle pokrećemo modifikovanu pretragu u širinu (BFS). Dok prolazimo kroz graf, konstruišemo slojevito stablo sa korenom r. Grane od parnih do neparnih slojeva su neuparene grane, grane od neparnih prema parnim slojevima su uparene.
Složenost ovog algoritma je $O(|V|(|V| + |E|))$ - za BFS je potreban $O(|V| + |E|)$ i ovaj proces će možda morati da se ponovi $|V|$ puta da bi alternirajuća putanja bila pronađena.
Postoji i poboljšanje ovog algoritma, tako što se traži više alternirajućih puteva odjednom, ali za njih mora važiti da su disjunktni putevi. Tada je složenost $O(\sqrt{V}(|V| + |E|))$.

Za bipartitne grafove važi da nemaju cikluse neparne dužine. **Kako onda naći optimalno uparivanje u grafovima koji sadrže i neparne cikluse?**

3 Opis *Blossom* algoritma

Ideja *Blossom* algoritma je da se ciklus neparne dužine u grafu može kontrakovati (skupiti) u jedan čvor tako da se pretraga za alternirajućim putevima može nastaviti iterativno kroz sada smanjeni graf [3] (slika 4). Ovde se ciklus neparne dužine naziva cvet (***blossom***). Stabljika (***stem***) je put od neuparenog čvora do cveta, a vrh (***tip***) je čvor koji spaja stabljiku i cvet. Poslednja grana stabljike do cveta (ta grana sadrži vrh) je uvek uparena. Zaista, ako bi poslednja grana stabljike bila neuparena, to bi značilo da bi se alternirajući put mogao dalje produžiti dodavanjem još jedne neuparene grane, tako da opet dobijamo da je grana koja povezuje *stem* i *blossom* uparena.
U cvetu postoji $2k + 1$ grana od kojih tačno k pripada uparivanju.[2] Ovo znači da postoji tačno jedan specijalni čvor u ciklusu koji nije uparen ni sa jednim drugim čvorom u ciklusu.



Slika 4: Primer upotrebe *Blossom* algoritma na proizvoljnom grafu

Cilj ovog algoritma je naći optimalno uparivanje u grafu. Koristi se pomoćna struktura podataka *forest* za čuvanje čvorova. Algoritam počinje od neuparenog čvora, prolazi se kroz grane do drugih neuparenih čvorova i grane se dodaju u uparivanje. Kada se dođe do maksimalnog uparivanja, tada se traže alternirajući putevi koji bi uparivanje povećali do optimalnog. Na početku se svaki neupareni čvor dodaje u red za obilazak. Za svaki od tih čvorova se kreira stablo koje se dodaje u šumu. Za svaki čvor v iz reda *nodes_to_check* prolazi se redom kroz svaki njegov susedni čvor w.

Algorithm 1 Nalaženje optimalnog uparivanja

```
function FIND_MAXIMUM_MATCHING(G, M)
    P ← FIND_AUGMENTING_PATH(G, M)
    if P ≠ [] then
        Add alternating edges of P to M
        return FIND_MAXIMUM_MATCHING(G, M)
    else
        return M
    end if
end function
```

Algorithm 2 Nalaženje alternirajućeg puta

```
function FIND_AUGMENTING_PATH(G, M)
    F ← empty forest
    unmark all edges in G, mark all edges in M
    nodes_to_check ← exposed vertices in G
    for v in nodes_to_check do
        create a singleton tree {v} and add the tree to F
        root(v) ← v
    end for
    for v in nodes_to_check do
        while there exists an unmarked edge e = (v, w) do
            if w ∉ F then                                   ▷ Vertex w is in M
                ADD_TO_FOREST(M, F, v, w)
            else
                if dist(w, root(w))%2 == 0 then
                    if root(v) ≠ root(w) then
                        P ← RETURN_AUGMENTING_PATH(F, v, w)
                    else
                        P ← BLOSSOM_RECURSION(G, M, F, v, w)
                        return P
                    end if
                else
                    continue
                end if
            end if
            mark edge e
        end while
    end for
    return []
end function
```

Ako se sused w ne nalazi u šumi, to znači da postoji čvor x sa kojim je on u uparivanju. Tada dodajemo grane (v, w) i (w, x) u drvo čvora v. Čvor x se dodaje u red za obilazak i pretraga će se nastaviti od njega nakon što se obiđu svi susedi čvora v.

Algorithm 3 Dodavanje čvora u forest

function ADD_TO_FOREST(M, F, v, w)
  $x \leftarrow$ vertex adjacent to w in M
  add edges $(v, w), (w, x)$ to tree(v) in F
  add vertex x to *nodes_to_check*
  $root(w) \leftarrow root(v)$
  $root(x) \leftarrow root(v)$
end function

Inače, čvor w se nalazi u šumi. Ako je na parnom rastojanju od korena (tj. broj grana do korena je paran) i ne nalazi se u drvetu čvora v, znači da je nađen alternirajući put i vraća se u glavnu funkciju. Alternirajući put predstavlja put od korena do čvora v u jednom stablu spojen sa putem od korena do čvora w u drugom stablu.

Algorithm 4 Vrati alternirajući put

procedure RETURN_AUGMENTING_PATH(F, v, w)
  $P1 \leftarrow$ SHORTEST_PATH($F, root(v), v$)
  $P2 \leftarrow$ SHORTEST_PATH($F, w, root(w)$)
  return $P1 + P2$
end procedure

Inače, ako je čvor w u šumi, nalazi se parnom rastojanju od korena i u istom stablu je kao i čvor v, znači da je nađen neparan ciklus, tj. *blossom*. On se skupi u jedan čvor i traži se alternirajući put u tako smanjenom grafu. Kada se pronađe ta putanja i poveća uparivanje, *blossom* se otvara i pravilno se rekonstruiše alternirajući put kroz cvet.

Algorithm 5 Kontrakcija *blossom*a

function CONTRACT_BLOSSOM(G, M, v, w)
  $B \leftarrow$ blossom formed by (v, w) and edges $v \rightarrow w$
  $G' \leftarrow G$ with all blossom nodes contracted into w
  $M' \leftarrow M$ with all blossom nodes contracted into w
  $P' \leftarrow$ FIND_AUGMENTING_PATH(G', M')
  $P \leftarrow$ lift P' to G
  return P
end function

Algoritam se zaustavlja kada nema više alternirajućih puteva u grafu, to jest kada uparivanje postane optimalno.

3.1 Vizualizacija algoritma

Animacija i link projekta na github-u.

4 Analiza složenosti *Blossom* algoritma

Vremenska složenost algoritma

Na početku postoji $|V|$ slobodnih čvorova. Za svaki od tih čvorova izvršavamo pretragu u širinu da bismo pronašli alternirajući put, koja radi u vremenskoj složenosti $O(|V| + |E|) = O(|E|)$ za povezane grafove. Pored BFS-a, moramo izvršiti najviše $|V|$ kontrakcija (a samim tim i najviše $|V|$ proširenja). Jedna kontrakcija (proširenje) se može izvršiti u vremenu $O(|E|)$ jer u najgorem slučaju moramo da skupimo sve čvorove i grane u jedan čvor. Nakon što smo pronašli alternirajući put, invertovanje grana u najgorem slučaju zahteva vreme $O(|V|)$. Sve u svemu, ovo dovodi do ukupnog vremena rada $O(|V| \cdot (|E| + |E||V| + |V|)) = O(|V|^2|E|)$. [1]

Kako se algoritam može poboljšati?

Postoji mnogo prostora za poboljšanje performansi algoritma. Neke ideje Mikalijevog i Vaziranijevog algoritma dodatno smanjuju vreme rada na $O(\sqrt{|V|}|E|)$:

- Umesto da izvodimo BFS iz jednog slobodnog čvora, mogli bismo da pokrenemo BFS iz svih slobodnih čvorova istovremeno. Na taj način možemo pronaći nekoliko disjunktnih alternirajućih puteva odjednom i povećati kardinalnost uparivanja za više od 1.
- Ne moraju se svi cvetovi odmah skupiti. Postoji takozvani uslov cvetanja koji određuje da li moramo da sklopimo ciklus neparne dužine ili možemo to da odložimo.
- Posebna tehnika obeležavanja omogućava brzo širenje cvetova.[1]

Štaviše, Edmonds tvrdi da, pod određenim okolnostima, 'superčvorovi' (čvorovi nastali skupljanjem cvetova) mogu biti ostavljeni kontrahovani sve dok njihovo proširenje nije zaista potrebno [6]. U našoj implementaciji, proširujemo sve superčvorove u grafu kada pronađemo putanju za povećanje, ali u mnogim slučajevima to nije neophodno.

5 Poređenje sa drugim algoritmima

Uporedićemo *Blossom* algoritam sa Hopkroft-Karp algoritmom i Ford-Fulkersonovim algoritmom.

Algoritam	**Tip grafa**	**Ključna ideja**	**Vremenska složenost**	**Karakteristike**
Blossom	Opšti	Koristi tehnike skupljanja za iterativno smanjenje ciklusa neparne dužine (cvetanja) dok se podudaranje ne može proširiti	$O(n^3)$	Dokazano je da je optimalan u najgorem slučaju za pronalaženje optimalnog uparivanja u opštim grafovima
Hopkroft-Karp	Bipartitni	U jednoj pretrazi se pronalazi veći broj alternirajućih puteva koji moraju biti međusobno nezavisni (disjunktni)	$O(n^{2.5})$	Brz za bipartitne grafove, ali nije primenljiv na opšte grafove (koji mogu da sadrže neparne cikluse)
Ford-Fulkerson	Opšti	Oslanja se na tehnike maksimizovanja protoka u transportnoj mreži	$O(n^5)$	Nije tako efikasan u praksi zbog proizvoljnog izbora povećavajućeg puta [1].

Ključna razmatranja za izbor algoritma

- Tip grafa: Bipartitni ili opšti?
- Vremenska efikasnost: Koliko je brzina ključna za rad aplikacije?
- Složenost implementacije: Da li su detalji implementacije algoritma zadovoljavajući?
- Opšte preporuke:
  - Za opšte grafove: *Blossom* algoritam je često poželjan izbor zbog njegove efikasnosti i sposobnosti da rukuje bilo kojom strukturom grafa.
  - Za bipartitne grafove: Hopkroft-Karp algoritam je generalno brži i jednostavniji za implementaciju.

[1] U opštem slučaju, kada su kapaciteti grana celobrojni vreme izvršavanja Ford-Fulkersonovog algoritma može biti $O(|E| \cdot f)$, gde je sa f označen maksimalni tok kroz mrežu. Edmonds i Karp su 1972. godine pokazali da ako se među mogućim povećavajućim putevima u rezidualnom grafu uvek bira onaj sa najmanjim brojem grana, onda je broj povećavanja najviše $(|V||E|)$, te je ukupna složenost algoritma $O(|V||E|)$

6 O autoru *Blossom* algoritma

Džek Edmonds rođen je 1934. godine u SAD-u [5]. On je doktor računarskih nauka i dobitnik Džon fon Nojmanove nagrade za izuzetne doprinose.

Slika 5: Džek Edmonds

Ključne oblasti gde se ističe uticaj Edmondsa:

- Kombinatorna Optimizacija: Edmonds se smatra jednim od osnivača ove oblasti, koja se fokusira na pronalaženje "najboljeg"rešenja među velikim brojem mogućnosti. Njegov *Blossom* algoritam za optimalna uparivanja u grafovima, razvijen 1961. godine, ostaje temelj ove oblasti.
- Poliedarska Kombinatorika: Dao je značajan doprinos razvoju poliedarskih metoda, koristeći geometrijske oblike za predstavljanje i rešavanje problema optimizacije.
- Teorija Računarske Složenosti: Ova oblast istražuje inherentnu težinu računarskih problema. Edmondsov rad, posebno njegov članak iz 1965. godine "Paths, Trees and Flowers,"pripisuje se za iniciranje razvoja matematičke teorije efikasnih kombinatornih algoritama.

7 Zaključak

Ovde su navedeni neki primeri iz stvarnog sveta koji ilustruju kako je efektivnost algoritma doprinela rešavanju složenih problema uparivanja u različitim kontekstima.

- *Optimizacija mrežnog toka*
  U optimizaciji mrežnog toka, *Blossom* algoritam se može koristiti za pronalaženje maksimalnog protoka ili minimalnog preseka u mreži. Određivanjem optimalnog protoka resursa kroz mrežu, obezbeđuje se efikasno korišćenje resursa i optimalne performanse mrežnih sistema kao što su transportne mreže, telekomunikacione mreže i mreže lanca snabdevanja.
- *Alokacija resursa u računarskim mrežama*
  Računarske mreže se često suočavaju sa izazovima u vezi sa alokacijom resursa, gde resursi kao što su propusni opseg, procesorska snaga ili skladištenje moraju biti optimalno dodeljeni različitim korisnicima ili zadacima. *Blossom* algoritam može pomoći u maksimalnom korišćenju resursa dok zadovoljava ograničenja, čime se povećava efikasnost i performanse računarskih mreža.
- *Problemi sa stabilnim brakom*
  U kontekstu stabilnih bračnih problema, gde svaki skup muškaraca i žena ima preferencije za potencijalne partnere, *Blossom* algoritam se može koristiti za pronalaženje stabilnih uparivanja koje zadovoljavaju određene kriterijume, kao što je osiguranje da nijedan par pojedinaca ne bi više voleo jedni druge u odnosu na trenutne partnere. Ova aplikacija ima praktične koristi u platformama za pronalaženje provodadžija, dodeljivanju poslova i drugim scenarijima gde preferencije treba uzeti u obzir. [4]

Ovi primeri pokazuju svestranost *Blossom* algoritma u rešavanju različitih problema optimizacije i uparivanja u različitim domenima, pokazujući njegovu praktičnu važnost u efikasnom rešavanju izazova u stvarnom svetu.

Literatura

[1] Blossom algorithm tum,2016. https://algorithms.discrete.ma.tum.de/graph-algorithms/matchings-blossom-algorithm/index_en.html.

[2] brilliant.org. https://brilliant.org/wiki/blossom-algorithm/.

[3] iq.opengenus.org. https://iq.opengenus.org/blossom-maximum-matching-algorithm/.

[4] Stable marriage problem - numberphile. https://www.youtube.com/watch?v=Qcv1IqHWAzg.

[5] wiki jack edmonds. https://en.wikipedia.org/wiki/Jack_Edmonds.

[6] Jack Edmonds. *Paths, Trees and Flowers*. 1965.

[7] Vesna Marinković. *Konstrukcija i analiza algoritama 2*. Matematički fakultet, Beograd, 2023.