Brzi heuristički algoritam za problem vršnog pokrivača

Tolj, Bruno

Undergraduate thesis / Završni rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:377621

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni preddiplomski studij

BRZI HEURISTIČKI ALGORITAM ZA PROBLEM VRŠNOG POKRIVAČA

Završni rad

Bruno Tolj

Osijek, 2020.

SADRŽAJ

1. UVOD ... 1

1.1. Zadatak završnog rada ... 1

2. TEORIJA GRAFOVA ... 2

2.1. Vršni pokrivač ... 3

2.2. Srodni problemi ... 4

2.3. NP-potpunost ... 5

3. POSTOJEĆI ALGORITMI ... 7

3.1. Brute-force algoritam ... 7

3.2. Pohlepni algoritam ... 8

3.3. Algoritam podudaranja ... 9

3.4. Usporedba točnosti i brzine ... 11

4. OGRANIČENJA I POSEBNI SLUČAJEVI ... 12

4.1. Stabla, ciklički grafovi i bipartitni grafovi ... 12

4.2. Komplementarni graf ... 13

5. BRZI HEURISTIČKI ALGORITAM ... 14

6. ZAKLJUČAK ... 16

SAŽETAK ... 18

ABSTRACT ... 18

ŽIVOTOPIS ... 19

1. UVOD

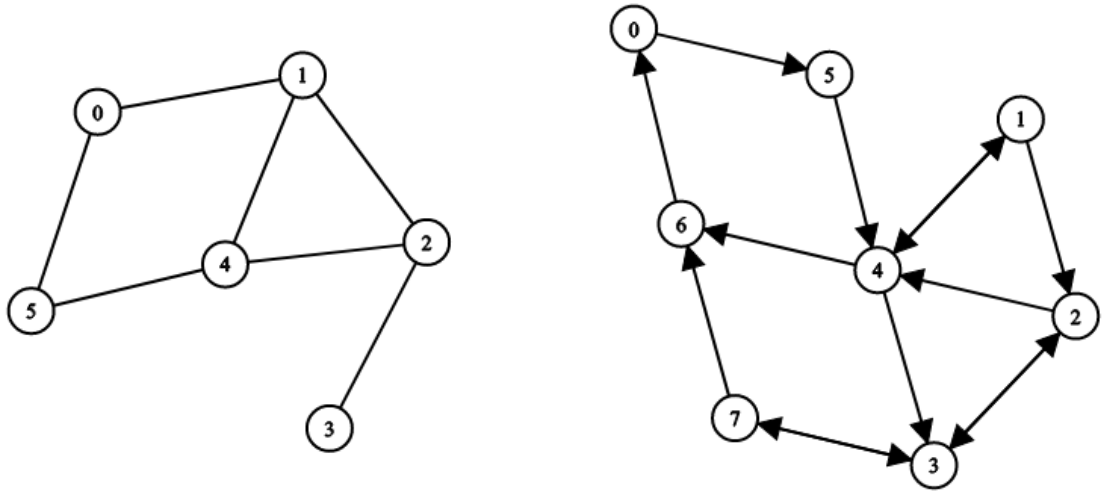
U teoriji grafova, vršni pokrivač (eng. *vertex cover*) je skup čvorova u grafu za koje vrijedi da svaki brid u grafu dodiruje barem jedan čvor vršnog pokrivača. Problem vršnog pokrivača odnosi se na pronalazak pokrivača određene veličine ili pronalazak najmanjeg pokrivača. Rješenja ovog problema imaju primjene u biologiji, kemiji, računalnim mrežama, cestovnoj infrasktrukturi i optimizaciji drugih algoritama. Budući da taj problem spada u klasu NP potpunih problema, što će biti objašnjeno u radu, algoritmi za ovaj problem se mogu primjeniti i na razne druge probleme u području matematike, logike i računarstva. Algoritmi opisani u ovom radu će se odnositi na pronalazak najmanjeg vršnog pokrivača jer se time dolazi do značajnijeg rješenja u većini primjena.

1.1. Zadatak završnog rada

Cilj ovog rada je osmisliti algoritam koji uz razumnu točnost može pronaći što manji vršni pokrivač proizvoljnog grafa. To će biti ostvareno analizom postojećih algoritama i povećanjem učinkovitosti pomoću dokazanih teorema iz područja teorije grafova. Svi algoritmi u radu pisani su u C++ programskom jeziku. Za zapis grafova u kodovima korišten je oblik matrice susjedstva.

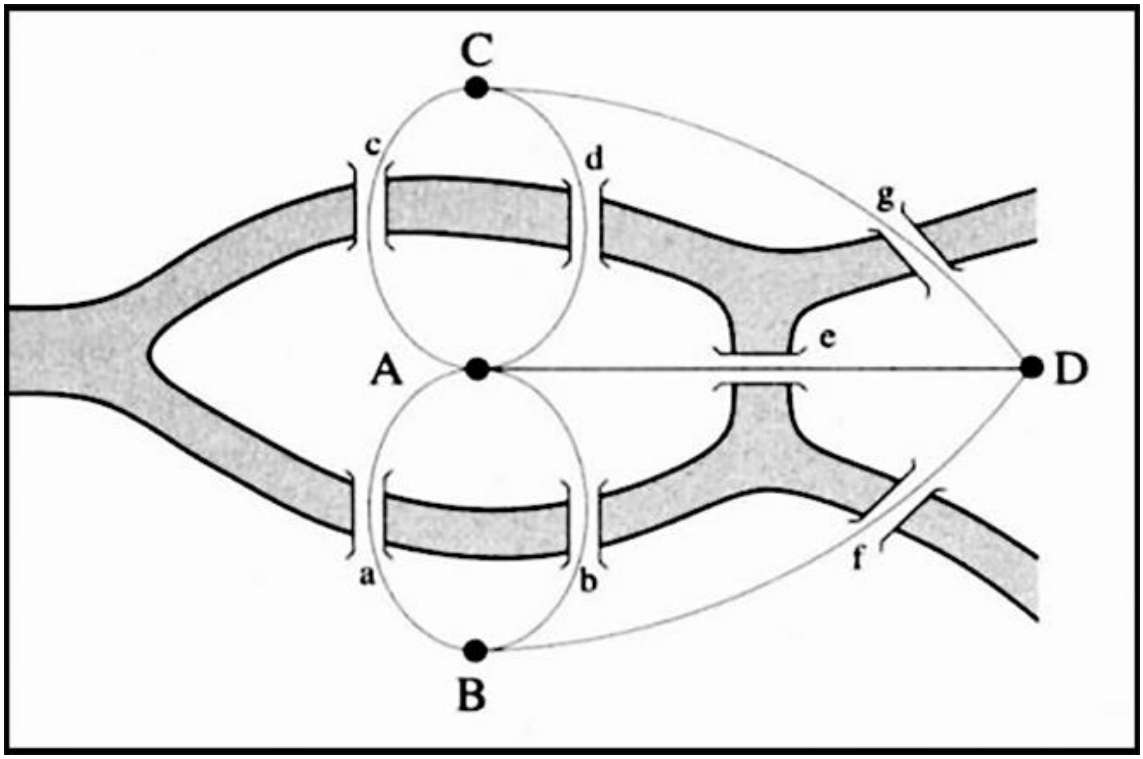
2. TEORIJA GRAFOVA

Teorija grafova bavi se problemima koji uključuju odnose između parova objekata te se mogu modelirati pomoću grafa. Graf je matematička struktura koju čini skup čvorova koji predstavljaju objekte i bridova između parova čvorova koji predstavljaju njihove odnose [1]. Grafovi se dijele na neusmjerene i usmjerene grafove ovisno o simetričnosti bridova, odnosno imaju li bridovi (a,b) i (b,a) jednake vrijednosti. Grafovi također mogu imati bridove koji povezuju čvor sa samim sobom koji se nazivaju petlje. Bridovi grafa mogu imati binarnu (npr. poznaju li se dvije osobe) ili brojčanu vrijednost (npr. udaljenost između dvaju gradova).



Slika 2.1. *– primjer neusmjerenog (lijevo) i usmjerenog (desno) grafa*

Teorija grafova službeno je nastala 1736. kada je švicarski matematičar Leonhard Euler objavio rad o sedam Königsberških mostova [2]. Pruski grad Königsberg (danas Kaliningrad, Rusija) nalazio se na obje strane rijeke Pregel i uključivao je dva otoka na rijeci. Kopnene mase bile su povezane s ukupno sedam mostova. Problem Königsberških mostova bio je odrediti je li moguće u jednoj šetnji preći preko svakog od sedam mostova točno jedanput. Euler je u svojem radu dokazao da je to nemoguće i odredio uvjet za postojanje obilaska grafa koji svaki brid prelazi točno jednom, što se danas naziva Eulerova staza.



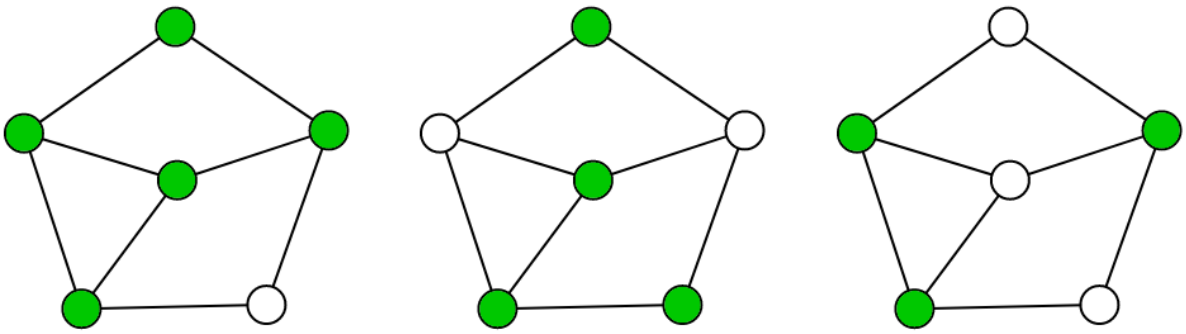
Slika 2.2. *– model Königsberških mostova u obliku grafa – čvorovi predstavljaju kopnene mase, a bridovi mostove*

Grafovi se vrlo često pojavljuju u modernim tehnologijama kao što su internetske mreže, električni strujni krugovi, navigacija putem gps-a, strojno učenje putem neuronskih mreža i slično. Grafovi se također mogu pronaći u modelima molekula, socijalnih mreža, sintakse i semantike jezika, migracijskih puteva životinja i mnogim drugima.

2.1. Vršni pokrivač

Primjer problema koji se riješava grafovima je postavljanje kamera na cestovna raskrižja. U ovom slučaju raskrižja su čvorovi, a ceste između raskrižja bridovi. Kamera može pokriti sve ceste koje vode do raskrižja i potrebno je pokriti svaku cestu, a cilj je postavljanje što manjeg broja kamera radi uštede. U modelu grafa riješenje za ovaj problem je skup čvorova grafa takav da svaki brid dodiruje barem jedan čvor iz skupa, zvan vršni pokrivač.

Vršni pokrivač nema ograničenja na veličinu, stoga svi čvorovi grafa također čine jedan od pokrivača. Takvom pokrivaču moguće je oduzeti jedan čvor tako da skup ostane vršni pokrivač, što nije poželjno u primjeni. Vršni pokrivač kojem nije moguće oduzeti niti jedan čvor naziva se minimalni vršni pokrivač. Do takvog pokrivača je vrlo lako doći, ali to još uvijek može biti daleko od najboljeg rješenja. U svakom grafu postoje jedan ili više najmanjih vršnih pokrivača, odnosno pokrivača s najmanjim mogućim brojem čvorova. Pronalazak takvih pokrivača u velikim grafovima je izuzetno složeno i zahtjeva mnogo procesorskog vremena.



(1) (2)

(3)

Slika 2.3. *– vršni pokrivač sa pet čvorova (1), minimalni vršni pokrivač sa četiri čvora (2) i najmanji vršni pokrivač sa tri čvora (3)*

2.2. Srodni problemi

Analiza problema bliskih izvornom pomažu pri pronalasku učinkovitijih rješenja. Velik broj problema iz teorije grafova ima međusobno povezane rezultate što dodatno povećava korisnost njihovog proučavanja.

Komplement skupa S u grafu G je skup svih čvorova u grafu G koji nisu u skupu S. Komplement vršnog pokrivača tvori nezavisni skup grafa, odnosno skup čvorova takav da nijedan brid grafa ne dodiruje dva čvora tog skupa. Dokaz tvrdnje je jednostavan: ukoliko postoji brid koji povezuje dva čvora komplementa vršnog pokrivača, tada postoji brid koji ne dodiruje niti jedan čvor pokrivača,

što je kontradikcija sa definicijom vršnog pokrivača. Stoga je pronalaženje najmanjeg vršnog pokrivača ekvivalentno pronalasku najvećeg nezavisnog skupa grafa.

Skup čvorova u grafu takav da su svaka dva čvora povezana bridom naziva se klika (engl. *clique*). Komplementaran graf H grafa G čine svi čvorovi grafa G i sadrži bridove takve da za svaki par točaka vrijedi: ako i samo ako je par povezan bridom u grafu H, onda taj par nije povezan bridom u grafu G. Prema tome, nezavisni skup nekog grafa postaje klika u komplementarnom grafu. Stoga se pronalazak najmanjeg vršnog pokrivača može svesti na pronalazak najveće klike u komplementarnom grafu.

Podudaranje (engl. *matching*) je skup bridova grafa bez zajedničkih čvorova. Podudaranje kojem nije moguće dodati niti jedan brid iz grafa naziva se maksimalno podudaranje, a podudaranje čiji bridovi dodiruju sve čvorove naziva se savršeno podudaranje. Po definiciji, u grafu ne postoji brid koji dodiruje dva čvora od kojih bar jedan nije dodirnut bridom iz skupa maksimalnog podudaranja tog grafa. Prema tome, za svako maksimalno podudaranje vrijedi da skup točaka koje dodiruju bridovi iz tog podudaranja čini vršni pokrivač [3].

2.3. NP-potpunost

Odgovaranje na pitanje „Postoji li vršni pokrivač veličine k u grafu G?“ spada u klasu NP potpunih računalnih problema. Problemi u klasi NP su oni za koje trenutno ne postoji algoritam koji može riješiti problem deterministički u polinomijalnom vremenu ovisno o veličini ulaza, ali se mogu riješiti nedeterministički, što nije moguće na današnjim računalima. Poseban slučaj NP problema su NP-potpuni problemi, odnosno svi problemi koji se mogu svesti jedan na drugi u polinomijalnom vremenu. NP-potpunost problema vršnog pokrivača dokazao je 1972. američki računalni znanstvenik Richard Karp [4]. Pronalaskom najmanjeg vršnog pokrivača također se odgovara na spomenuto pitanje pa se modificirani algoritam za pronalaženje najmanjeg pokrivača može koristiti za rješavanje bilo kojeg NP-potpunog problema.

Jedan od najvažnijih NP-potpunih problema je određivanje zadovoljivosti logičkih formula u konjunktivnoj normalnoj formi (KNF), osim ako svaka klauza sadrži samo dva literala jer tada problem postaje rješiv u polinomijalnom vremenu. Za većinu NP-potpunih problema dokaz je ostvaren svođenjem sa problema zadovoljivosti. Prema svojstvima klase vrijedi i da se problem

vršnog pokrivača može svesti na problem zadovoljivosti. Svaki čvor predstavlja jedan literal gdje „istina“ znači da je čvor uključen u vršni pokrivač, a svaki brid predstavlja disjunkciju čvorova koje dodiruje. Konjunkcija svih bridova sa uvjetom da broj uključenih čvorova mora biti manji ili jednak odabranom broju k je logička formula ekvivalentna početnom problemu koja se može svesti na KNF.

Velik broj NP-potpunih problema vezani su za grafove. To uključuje: kromatski broj grafa (podjela grafa na što manji broj nezavisnih skupova), bridni pokrivač (skup bridova koji pokriva sve čvorove) itd. Osim toga, u NP-potpunoj klasi nalaze se razni matematički problemi najčešće vezani za skupove i sume te poneke igre kao što je sudoku.

3. POSTOJEĆI ALGORITMI

Budući da je problem poznat već dugo vremena i pripada jednoj od najpopularnijih kategorija računalnih problema, postoje brojni algoritmi za traženje vršnih pokrivača s raznim optimizacijama. U ovom poglavlju objašnjeni su i uspoređeni najosnovniji koji će pružati osnovu novom heurističkom algoritmu.

3.1. Brute-force algoritam

Najjednostavniji početak rješavanja svakog problema je isprobavanje svih mogućih kombinacija radi otkrivanja uzoraka koji će ubrzati novije inačice algoritma. Na grafu sa n čvorova postoji 2^n podskupova čvorova koje je potrebno provjeriti za pronalazak najmanjeg vršnog pokrivača. Za svaki podskup provjera se vrši tako da se za svaki brid ispita dodiruje li barem jedan čvor iz podskupa. Ukoliko svaki brid u grafu zadovoljava taj uvjet, tada podskup čini vršni pokrivač i obrnuto. Svaki novo pronađeni pokrivač se uspoređuje sa do tada najmanjim čime se u konačnici dolazi do stvarnog najmanjeg.

```
std::set<int> vertexCoverBruteforce(bool** graph, int n) {
	// početno rješenje je skup svih čvorova u grafu
	std::set<int> solution;
	for (int i = 0; i < n; i++)
		solution.insert(i);
	// svaki podskup zapisan je u obliku bitmaska radi lakšeg obilaska u petlji
	for (unsigned long long attempt = 0; attempt < (1ull << n); attempt++) {
		std::set<int> current;
		for (int vertex = 0; vertex < n; vertex++)
			if (attempt & (1ull << vertex))
				current.insert(vertex);
		// preskaču se slučajevi koji nisu manji od trenutnog rješenja
		if (solution.size() <= current.size())
			continue;
		// ispitivanje bridova prema uvjetu za vršni pokrivač
		bool isVertexCover = true;
		for (int i = 0; i < n; i++) {
			for (int j = i + 1; j < n; j++)
				if (graph[i][j] && current.find(i) == current.end() &&
					current.find(j) == current.end()) {
					isVertexCover = false;
					break;
```

```
                        }
                if (!isVertexCover)
                        break;
            }
            // već je poznato da je novi podskup manji pa postaje novo rješenje
            if (isVertexCover)
                solution = current;
        }
        return solution;
}
```

Programski kod 3.1. *– brute-force algoritam za traženje najmanjeg vršnog pokrivača*

Korištenjem bitmaska sa tipom podatka unsigned long long ograničava program na grafove sa maksimalno 64 čvora zbog veličine zapisa u memoriji. U slučaju brute-force algoritma to ne predstavlja problem jer računala ne mogu provjeriti sve podskupove za veće grafove u razumnoj količini vremena zbog eksponencijalne vremenske složenosti. Brzina algoritma se može povećati promjenom načina obilaska svih kombinacija.

3.2. Pohlepni algoritam

Pohlepni algoritmi rade odluke tako da uvijek odabiru ono što će ih u tom trenutku dovesti najbliže cilju. Čvor koji dodiruje najviše bridova, odnosno ima najviši stupanj [5], najviše doprinosti vršnom pokrivaču. Pohlepni algoritam za vršni pokrivač radi tako da ukloni čvor najvišeg stupnja iz grafa i doda ga u drugi skup te ponavlja tu radnju dok postoje bridovi u grafu. Na kraju će dobiveni skup biti vršni pokrivač, ali ne nužno najmanji. Prednost pohlepnog algoritma je polonomijalna vremenska složenost.

```
std::set<int> vertexCoverGreedy(bool** graph, int n) {
        std::set<int> solution;
        bool** temp = new bool*[n];
        for (int i = 0; i < n; i++) {
            temp[i] = new bool[n] {};
            for (int j = 0; j < n; j++) {
                temp[i][j] = graph[i][j];
            }
        }
```

```
        while (true) {
                int maxIndex = n, maxOrder = n;
                // pretraživanje čvorova grafa po stupnju
                for (int i = 0; i < n; i++) {
                        int order = 0;
                        for (int j = 0; j < n; j++)
                                if (i != j && temp[i][j])
                                        order++;
                        if (order > maxOrder) {
                                maxOrder = order;
                                maxIndex = i;
                        }
                }
                // ako nema bridova u grafu, izvođenje se prekida
                if (maxIndex == n)
                        break;
                // bridovi koji dodiruju odabrani čvor se brišu
                for (int i = 0; i < n; i++) {
                        temp[maxIndex][i] = false;
                        temp[i][maxIndex] = false;
                }
                solution.insert(maxIndex);
        }
        for (int i = 0; i < n; i++) {
                delete[] temp[i];
        }
        delete[] temp;
        return solution;
}
```

Programski kod 3.2. *– pohlepni algoritam za traženje vršnog pokrivača*

Iako dani algoritam ne daje uvijek optimalan rezultat, može služiti za brzo računanje gornje granice veličine stvarnog rješenja. Za razliku od prijašnjeg, algoritam nije ograničen veličinom grafa zbog korištenih tipova podataka, što ostavlja količinu memorije računala i mogućnost indeksiranja velikih nizova kao jedina ograničenja.

3.3. Algoritam podudaranja

Kao što je spomenuto u prošlom poglavlju, čvorovi maksimalnog podudaranja u nekog grafu također čine i vršni pokrivač. Osim toga, budući da svaki brid maksimalnog podudaranja mora dodirivati barem jedan čvor pokrivača, najmanji vršni pokrivač ne može sadržavati manje od pola

čvorova maksimalnog podudaranja. Algoritam za traženje podudaranja odabire bilo koji brid grafa i izbacuje čvorove koje dodiruje te ih dodaje u novi skup i proces se ponavlja dok više ne bude bridova za odabrati.

```
std::set<int> vertexCoverMatching(bool** graph, int n) {
	std::set<int> solution;
	int edges = 0;
	bool** temp = new bool*[n];
	for (int i = 0; i < n; i++) {
		temp[i] = new bool[n] {};
		for (int j = 0; j < n; j++) {
			temp[i][j] = graph[i][j];
			edges += temp[i][j];
		}
	}
	while (edges) {
		int a = rand() % n, b = rand() % n;
		if ((a != b) && temp[a][b]) {
			// čvorovi koji dodiruju odabrani brid se uklanjaju iz grafa
			for (int i = 0; i < n; i++) {
				edges -= temp[a][i];
				temp[a][i] = false;
				edges -= temp[i][a];
				temp[i][a] = false;
				edges -= temp[b][i];
				temp[b][i] = false;
				edges -= temp[i][b];
				temp[i][b] = false;
			}
			solution.insert(a);
			solution.insert(b);
		}
	}
	for (int i = 0; i < n; i++) {
		delete[] temp[i];
	}
	delete[] temp;
	return solution;
}
```

Programski kod 3.3. *– algoritam za traženje maksimalnog podudaranja*

U algoritmu je korištena funkcija rand() za nasumičan odabir bridova, ali bridovi mogu biti odabrani na bilo koji način. Izbor bridova ne utječe na brzinu i točnost algoritma.

3.4. Usporedba točnosti i brzine

Spomenuti algoritmi i neke njihove kombinacije testirane su na grafovima raznih veličina. Zbog vrlo velike razlike u vremenima izvođenja, za brute-force algoritam ispisana su vremena izvođenja, a za ostale prosječna odstupanja u veličini rezultata od najmanjeg.

Tablica 3.1. *– rezultati izvođenja programa na malim grafovima*

Veličina grafa	Brute-force	Pohlepni	Podudaranja
9	15.4 ms	0	2
12	142.8 ms	0	2.2
15	1278.4 ms	0	3.4
18	12.62 s	0.4	3.6
21	113.28 s	0.6	4.6

Brute-force algoritam radi presporo za ispitivanje na većim grafovima. Stoga su u sljedećoj tablici ispisane samo razlike u veličini rezultata između druga dva algoritma i njihovo vrijeme izvođenja za znatno veće grafove gdje veličina najmanjeg pokrivača nije poznata.

Tablica 3.2. *– rezultati izvođenja programa na velikim grafovima*

Veličina grafa	Pohlepni	Podudaranja	Razlika rezultata
100	5 ms	2.2 ms	5.2
150	17 ms	6.2 ms	6.4
200	37.6 ms	11.6 ms	7.2
300	147.2 ms	18.8 ms	7.2
400	283.4 ms	31.8 ms	7.4

Algoritam podudaranja pokazao se bržim, dok je pohlepni algoritam davao točnije rezultate. Iako nisu poznate veličine najmanjih pokrivača, bitno je naglasiti da su algoritmi pronalazili rješenja koja obuhvaćaju gotovo sve čvorove u grafu.

4. OGRANIČENJA I POSEBNI SLUČAJEVI

Do sada spomenuti algoritmi rade na općenitim slučajevima i gotovo bez ikakvih optimizacija. Međutim, spoznaje iz teorije grafova pomažu znatnom povećanju brzine i točnosti bez pretjerane dodatne kompliciranosti koda. Na primjer, čvor koji dodiruje samo jedan brid može se sa sigurnšću isključiti iz najmanjeg pokrivača jer drugi čvor na tom bridu pokriva taj i još nekoliko bridova. Zatim, ako graf ima vršni pokrivač veličine k, onda prvih k čvorova poredanih po stupnju moraju imati zbroj stupnjeva veći ili jednak broju bridova u grafu.

4.1. Stabla, ciklički grafovi i bipartitni grafovi

Stabla su vrsta grafova u kojima su čvorovi podjeljeni u razine i svaki čvor ima točno jednog „roditelja“ iz niže razine s kojim je povezan bridom, osim korjenskog čvora koji se nalazi na početnoj razini, najčešće 0 ili 1. Struktura stabla je takva da se može prepoznati programski u polinomijalnom vremenu bez obzira na početni odabir korjenskog čvora. Najmanji vršni pokrivač na stablima se može pronaći u linearnom vremenu tako da se izdvoje čvorovi na svim parnim ili svim neparnim razinama, ovisno o tome koje ukupno imaju manje čvorova.

Ciklički grafovi su grafovi u kojima svi bridovi zajedno čine jednu petlju. Svi čvorovi u ovakvoj vrsti grafa imaju stupanj dva. Ciklički grafovi se također mogu prepoznati u polinomijalnom vremenu. Najmanji vršni pokrivač dobiva se tako da se obilaskom grafa izdvaja svaki drugi posjećeni čvor.

Obje spomenute vrste grafova spadaju u širu kategoriju zvanu bipartitni grafovi. Graf je bipartitni ako se svi njegovi čvorovi mogu podjeliti u dvije particije takve da niti jedan brid ne povezuje dva čvora iz iste particije. Drugim riječima, graf je bipartitni ako postoje dva nezavisna skupa koji zajedno uključuju sve čvorove tog grafa. Particije bipartitnog grafa su vršni pokrivač tog grafa, a manja particija je ujedno i najmanji pokrivač. Bipartitnost grafa može se ispitati s polinomijalnom vremenskom složenošću [6], ali sporije nego kod užih slučajeva kao što su gore navedeni.

4.2. Komplementarni graf

Dodatna ograničenja veličine pokrivača pojavljuju se u komplementarnom grafu zadanome. Kako bi graf mogao imati kliku veličine k mora imati dovoljno bridova da poveže svih k * (k – 1) / 2 parova. Budući da su klike i nezavisni skupovi povezani preko komplementarnih grafova, a nezavisni skupovi i vršni pokrivači su komplementi u istom grafu, Iz navedenog uvjeta može se izraziti uvjet koji povezuje broj bridova u grafu i veličinu pokrivača:

$$2k \geq 2n - 1 - \sqrt{(2n-1)^2 - 8e} \qquad (4-1)$$

k – veličina vršnog pokrivača

n – broj čvorova u grafu

e – broj bridova u grafu

Donja granica za veličinu najveće klike ovisno o broju bridova dana je Turánovim teoremom koji je 1941. dokazao mađarski matematičar Pál Turán [7]. Pretvorbom ovog uvjeta na isti način kao i prošli dobiva se sljedeći izraz:

$$k(n + 2e) \leq 2en \qquad (4-2)$$

Jednadžbe (4-1) i (4-2) određuju interval u kojem se nalazi veličina najmanjeg vršnog pokrivača nekog grafa.

5. BRZI HEURISTIČKI ALGORITAM

Nakon testiranja raznih algoritama pokazalo se da heuristički algoritmi s većom točnosti od pohlepnog algoritma imaju mnogo veće vrijeme izvođenja programa. Osim toga, rješenja algoritama u prosjeku imaju 1-2 čvora manje nego rješenje dobiveno pomoću pohlepnog pa njihovo korištenje nije praktično. Stoga je cilj pronalaska novog algoritma bio dodatno povećanje brzine s minimalnim gubitkom točnosti. Pohlepni algoritam u svakom koraku u vršni pokrivač dodaje čvor s velikim brojem bridova, ali na svakom koraku mora ponovno obići čitav graf. Algoritam podudaranja uzima prvi nađen par čvorova, no tada se čvorovi dodaju u pokrivač bezuvjetno.

Ovaj algoritam obilazi graf u krug i dodaje u pokrivač sve čvorove čiji stupanj prelazi određenu granicu. Granica je u početku jednaka veličini grafa – 1, odnosno u obzir dolaze samo čvorovi koji dijele brid sa svim ostalima. Svaki put kada čvor ne zadovolji uvjet, granica se spušta za 1. To omogućuje da se u pokrivač dodaju čvorovi koji pokrivaju mnogo bridova i da se mogu dodati čvorovi bez obilaska cijelog grafa.

```
std::set<int> vertexCoverTreshold(bool** graph, int n) {
	std::set<int> solution;
	bool** temp = new bool* [n];
	int edges = 0;
	for (int i = 0; i < n; i++) {
		temp[i] = new bool[n] {};
		for (int j = 0; j < n; j++)
			edges += temp[i][j] = graph[i][j];
	}
	int counter = 0, treshold = n - 1;
	while (edges) {
		int order = 0;
		for (int i = 0; i < n; i++)
			if (i != counter && temp[i][counter])
				order++;
		if (order) {
			if (order > treshold) {
				for (int i = 0; i < n; i++) {
					edges -= temp[counter][i];
					temp[counter][i] = false;
					edges -= temp[i][counter];
					temp[i][counter] = false;
```

```
                        }
                        solution.insert(counter);
                    }
                    else
                        treshold--;
                }
                counter++;
                if (counter == n)
                    counter = 0;
            }
            for (int i = 0; i < n; i++) {
                delete[] temp[i];
            }
            delete[] temp;
            return solution;
}
```

Programski kod 5.1. *– algoritam za traženje vršnog pokrivača sa graničnim stupnjevima*

Algoritam je uspoređen sa prijašnja dva algoritma na velikim grafovima, u sljedećoj tablici ispisana su vremena izvođenja i prosječne razlike u veličini rezultata.

Tablica 5.1. *– rezultati izvođenja algoritama na grafovima raznih veličina*

Veličina grafa	Pohlepni	Podudaranje	Granični	Podudaranje - Pohlepni	Granični - Pohlepni
500	651 ms	47.8 ms	16.4 ms	7.2	4.2
700	1695.6 ms	136.2 ms	21.4 ms	8.4	3.6
1000	5.02 s	169.2 ms	54 ms	8	3
1500	16.8 s	591.8 ms	119 ms	7.8	3.8
2000	40.84 s	1165.4 ms	257.2 ms	9	4.2

Algoritam se pokazao znatno uspješnijim od algoritma za pronalaženje maksimalnih podudaranja po vremenu izvođenja i točnosti. Svi algoritmi brzinom omogućuju korištenje na mnogo većim grafovima nego što je moguće osnovnim brute-force algoritmom. Za određivanje točnosti u odnosu na veličinu najmanjeg vršnog pokrivača potrebno je koristiti mnogo sporije algoritme.

6. ZAKLJUČAK

U sklopu ovog završnog rada opisan je problem vršnog pokrivača i osmišljen heuristički algoritam koji ga rješava brzo i relativno učinkovito.

Problem je detaljno opisan i uspoređen sa sličnim problemima te je objašnjena njegova vremenska složenost. Navedeni su parametri grafova koji su međusobno povezani i zajedno su svedeni na isti problem. Objašnjena je NP-potpuna klasa računalnih problema te su navedeni drugi problemi iz te klase.

Zatim su uspoređeni neki od algoritama za njegovo rješavanje. Pokazano je kako se potpuno točno rješenje može dobiti jedino provjerom svih potencijalnih rješenja.Također je pokazano da tim putem nije moguće doći do rješenja u proizvoljno velikim slučajevima, već je nužno osloniti se na heurističke algoritme.

Navedeno je nekoliko slučajeva kod kojih se može brže doći do rješenja zbog specifične strukture grafova. Osim toga, navedena su i ograničenja nekih parametara koja vrijede za sve grafove. Pomoću tih ograničenja određen je interval u kojem se nalazi veličina najmanjeg vršnog pokrivača grafa.

Heuristički algoritam napisan za ovaj zadatak testiran je i uspoređen sa već spomenutim algoritmima. Pokazao se kao vrlo dobra alternativa po pitanju brzine izvođenja i vremenske složenosti. Točnost rješenja usporediva je s ostalim algoritmima.

Svi algoritmi napisani su na relativno jednostavan način i njihova brzina bi se mogla povećati promjenom načina obilaska grafa i spremanja rješenja u memoriju te ostalim optimizacijama.

LITERATURA

[1] R. J. Trudeau, Introduction to Graph Theory, Dover Publications, New York, 1993.

[2] A. Benjamin, G. Chartland, P. Zhang, The Fascinating World of Graph Theory, Princeton University Press, Princeton, 2015.

[3] https://en.wikipedia.org/wiki/Vertex_cover

[4] https://enacademic.com/dic.nsf/enwiki/967781

[5] J. Clark, D. A. Holton, A First Look At Graph Theory, Singapore, 1991.

[6] https://www.geeksforgeeks.org/bipartite-graph/

[7] https://mathworld.wolfram.com/TuransTheorem.html

SAŽETAK

Brzi heuristički algoritam za problem vršnog pokrivača

Cilj ovog završnog rada bio je napisati algoritam za rješavanje problema vršnog pokrivača iz teorije grafova. Algoritam je napisan u programskom jeziku C++. Rad je podjeljen na pet poglavlja. Nakon uvoda u drugom poglavlju problem je stavljen u kontekst s ostalim problemima vezanim uz teoriju grafova i NP-potpunu klasu problema. U trećem poglavlju opisani su i uspoređeni postojeći algoritmi prema brzini izvođenja i točnosti. Četvrto poglavlje bavi se posebnim slučajevima koji se mogu lakše rješiti i pravilima za grafove pomoću kojih se sužava mogući raspon rješenja. U petom poglavlju objašnjen je novi algoritam i uspoređen sa drugim algoritmima navedenima u ovome radu.

Ključne riječi: algoritam, graf, heuristika, vršni pokrivač

ABSTRACT

Fast heuristic algorithm for the vertex cover problem

The goal of this paper was writing an algorithm for solving the vertex cover problem from graph theory. The algorithm is written in the C++ programming language. The paper is split into five chapters. After introduction in the second chapter the problem is put into context with other problems in graph theory and the NP-complete class. In the third chapter existing algorithms are described and compared based on execution speed and accuracy. The forth chapter deals with special cases which can be solved easier and graph rules which help narrow down the range of possible solutions. In the fifth chapter the new algorithm is explained and compared with other algorithms listed in this paper.

Keywords: algorithm, graph, heuristic, vertex cover

ŽIVOTOPIS

Bruno Tolj rođen je 18. veljače 1999. godine u Osijeku. Prva četiri razreda osnovne škole završava u područnoj školi Vladimira Nazora u Bruješću, a osnovnoškolsko obrazovanje završava u osnovnoj školi Vladimira Nazora u Čepinu. 2017. godine završava srednjoškolsko obrazovanje u III. gimnaziji u Osijeku te se upisuje na Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek. Trenutno je redovni student na trećoj godini preddiplomskog sveučilišnog studija računarstva.