Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Hľadanie perfektných párení

Bakalárska práca

2020
Filip Novák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Hľadanie perfektných párení

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Ján Mazák, PhD.

Bratislava, 2020
Filip Novák

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Filip Novák
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Hľadanie perfektných párení
Finding perfect matchings

Anotácia: Náplňou práce je implementovať niekoľko rôznych algoritmov, ktoré nájdu všetky perfektné párenia v grafe, a porovnať ich rýchlosť. Na základe experimentálnych poznatkov získaných z výpočtov pre kubické grafy chceme skombinovať tieto algoritmy tak, aby výsledok bol čo najefektívnejši pre zaujímavé konkrétne triedy snarkov.

Cieľ: Cieľom je získať prehľad v existujúcich algoritmoch pre hľadanie perfektných párení v grafoch. Vybrané algoritmy treba implementovať, experimentálne overiť ich výkonnosť na kubických grafoch a prípadne skombinovať do optimalizovaného algoritmu.

Vedúci: RNDr. Ján Mazák, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 25.09.2018

Dátum schválenia: 30.10.2019 doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Chcel by som srdečne poďakovať svojmu školiteľovi, RNDr. Jánovi Mazákovi, PhD. za užitočné a cenné rady, a taktiež za významnú odbornú pomoc, ktorú mi poskytoval počas písania tejto práce. Ďakujem mu, že bol ochotný mi pomáhať pri akomkoľvek probléme, ktorý sa vyskytol. Moje poďakovanie ďalej patrí aj mojej rodine a priateľom, ktorí ma do poslednej chvíle písania práce podporovali.

Abstrakt

Získali sme prehľad o rôznych algoritmoch, ktoré hľadajú perfektné párenia vo všeobecných grafoch. Niektoré z nich sme implementovali a testovali sme ich rýchlosť na kubických grafoch. Najspolahlivejší a najrýchlejší algoritmus sme vybrali a použili vo vlastnej implementácii algoritmu, ktorý hľadá všetky perfektné párenia v grafe. Pomocou vybratého algoritmu sa nám podarila zefektívniť naša implementácia algorimtu, ktorý sme testovali na kubických grafoch. Výsledný algoritmus sme testovali na snarkoch s rôznymi vlastnosťami a porovnávali s rýchlosťou AllSAT solvera.

Kľúčové slová: perfektné párenie, kubický graf, snark, algoritmus, AllSAT solver

Abstract

We have gained an overview of different algorithms which look for perfect matchings in general graphs. We have implemented and tested some of them for speed on cubic graphs. The most reliable and quickest algorithm was selected and used in our own implementation of algorithm which look for perfect matchings in a graph. With the help of said algorithm, we have managed to make our implementation more effective during our test on cubic graphs. The resulting algorithm was tested on snarks with different characteristics and compared to the speed of AIISAT solver.

Keywords: perfect matching, cubic graph, snark, algorithm, AllSAT solver

Obsah

Zoznam tabuliek

Úvod

Perfektné párenie, je zajímavou temou v oblasti teórii grafov. Je veľa rôznych prác a štúdií popisujúcich perfektné párnia v bipartitných grafoch, ale naopak vo všeobecných alebo kubických grafoch ich až tak veľa nie je. Rozhodli sme sa študovať počet perfektných párení v kubických grafoch a neskôr v snarkoch tým, že vytvoríme čo najefektívnejší vlastný algoritmus na hľadanie všetkých perfektných párení. Pomôžu nám pritom rôzne známe algoritmy, ktoré hľadajú v grafe nejaké perfektné párenie. Na konci porovnávame rýchlosť vlastného algoritmu s rýchlosťou AllSAT solvera na snarkoch s rozličnymi vlastnosťami.

Táto práca je rozdelená do piatich častí. Prvá je o základných pojmoch. v druhej popisujeme rôzne známa algoritmy. V tretej tieto algoritmy implementuje a testujeme ich na kubických grafoch. V švrtej časti popisujeme implementáciu vlastného algoritmu. V piatej časti testujeme náš vysledný algoritmus na snarkoch.

Kapitola 1

Základné pojmy a problémy

V tejto kapitole sa oboznámime so základnými pojmami, s ktorými budeme pracovať a spomenieme si rôzne problémy v tejto oblasti.

1.1 Definície

V našej práci budeme používať základný pojem grafu, označeného $G = (V, E)$, ktorý je definovaný pomocou množín V a E. Pod týmto pojmom rozumieme neorientovaný graf, kde V je množina vrcholov a E je množina hrán.

Predtým ako sa začneme zaujímať tým, čo je perfektné párenie v grafe, si musíme zadefinovať párenie v grafe, ktoré veľmi úzko súvisí s naším hlavným pojmom.

Definícia 1.1 *Párenie M v grafe G je množina takých hrán, že žiadne dve nemajú spoločný vrchol. Veľkosť párenia je počet hrán v nej* [8].

Vrchol je pokrytý párením, ak je koncovým bodom jednej z hrán v párení M. Poznáme rôzne pomenovania párení, ako naríklad maximálne a perfektné. Hlavným pojmom tejto práce, teda tým čo nás bude zaujímať, je pojem perfektného párenia. Zadefinujeme si ho a neskôr uvidíme, že aj maximálne párenie v ňom zohráva dôležitú úlohu.

Definícia 1.2 *Perfektné párenie M alebo 1-faktor je párenie, ktoré pokrýva každý vrchol grafu G* [8].

To znamená, že každý vrchol grafu G je incidentný presne s jednou hranou párenia. Grafom $G = (V, E)$ sa znova rozumie neorientovaný graf s množinou vrcholov V a množinou hrán E, ako sme už vyššie spomínali. Veľmi dôležitou podmienkou grafu G je, aby mal párny počet vrcholov, pretože grafy s nepárnym počtom nemôžu mať perfektné párenie. Ďalej vieme, že v perfektnom párení sú všetky vrcholy grafu pokryté, čo nám vyplýva z definície.

Ďaľším dôležitým faktorom grafu je 2-faktor. Spomínaný 2-faktor grafu G je preklenovací podgraf (spanning subgraph). Je to regulárny podgraf, kde všetky vrcholy majú stupeň dva (majú dvoch susedov), čo znamená, že ide o súbor cyklov, ktoré sa spolu dotýkajú každého vrcholu presne jedenkrát.

Dostávame sa k pojmu maximálneho párenia a k jeho prepojeniu s perfektným párením. Toto dôležité prepojenie nám hovorí, že každé perfektné párenie, je zároveň aj maximálne párenie. Znamená to, že obsahuje najväčší možný počet hrán.

Zostáva nám si objasniť poslednú vec. Keďže táto práca je o hľadaní perfektných párení, je dôležité si ujasniť na akých grafoch alebo typoch grafov budeme používať rôzne algoritmy, ktoré budú hľadať párenia. Rozhodli sme sa, že to bude zo začiatku na kubických grafoch a potom prejdeme na grafy nazývané *snarky*, pretože majú zaujímavé vlastnosti a nie sú až tak dobre preskúmané.

Definícia 1.3 *V matematickej oblasti teórie grafov je snark jednoduchý, bez mostov, súvislý kubický graf s chromatickým indexom rovným 4* [20].

Zjednodušene z tejto definície vyplýva, že ak by sme odstránili nejakú hranu, tak graf by to nerozdelilo na viacero komponentov. Každý vrchol grafu má troch susedov, a potrebovali by sme štyri farby na zafarbenie hrán grafu tak, aby všetky susediace hrany mali rôzne farby. Navyše sa dá zistiť aj nepárnosť, čo je minimálne číslo určujúce počet nepárnych cyklov v 2-faktor kubickom bezmostovom grafe G, ktoré sa označuje $\omega(G)$. Keďže každý kubický graf má párny počet vrcholov, tak aj nepárnosť grafu bude vždy párna.

1.2 Hypotézy

Oboznámime sa tu zo známymi hypotézami, ktoré súvisia s perfektnými pároniami. Niektoré sa podarilo dokázať pre určité typy grafov.

Berge-Fulkersonova hypotéza

Berge a Fulkerson nezáväzne vyslovili tvrdenia, ktoré boli zamerané na bezmostové kubické grafy a o týchto tvrdeniach sa podarilo dokázať že sú ekvivalentné [12]. Berge sa domnieval, že množinu hrán v grafe možno pokryť najviac piatimi perfekntými pároniami. Na druhú stranu Fulkerson predpokladal, že existuje šesť perfektných párení, kde každá hrana v grafe je obsiahnutá v presne dvoch z nich. Z týchto dvoch tvrdení sa stalo jedno a vznikla Berge-Fulkersonova hypotéza, ktorá tvrdí, že pre každý bezmostový kubický graf G existuje šesť perfektných párení $M_1, M_2, M_3, M_4, M_5, M_6$, kde každá hrana v G je obsiahnutá presne v dvoch zo spomínaných perfektných párení

[14]. Ak by mal graf G chromatický index rovný 3, tak by sme si mohli zvoliť tri perfektné párenia M_1, M_2, M_3, kde by každá hrana bola presne v jednom z tých troch. Ak použijeme každú z nich dvakrát, získame 6 perfektných párení s vyššie uvedenými vlastnosťami. Môžeme teda povedať, že vyššie spomínaná hypotéza triviálne platí pre grafy s chromatickým indexom 3.

Na bezmostových kubických grafoch dokážeme pozorovať vlastnosť nazývanú index perfektného párenia (perfect matching index), ktorý označujeme ako $\tau(G)$ grafu G. Je to minimálne číslo určujúce počet perfektných párení pokrývajúcich množinu hrán E grafu G.

Fan-Raspaudova hypotéza

V tejto hypotéze Fan a Raspaund predpokladajú, že každý bezmostový kubický graf obsahuje perfektné párenia M_1, M_2, M_3 také, kde $M_1 \cap M_2 \cap M_3 = \emptyset$ [7]. Môžeme vidieť, že táto hypotéza rovno vyplýva z vyššie spomínanej Berge-Fulkersonovej hypotézy, pretože môžeme zobrať akékoľvek tri perfektné párenia z tých šiestich, ktoré vyhovujú Berge-Fulkersonovej hypotéze.

Lovász–Plummerova hypotéza

V tejto časti budeme množinu perfektnýh párení grafu G označovať ako $\mathcal{M}(G)$.

V sedemdesiatych rokoch sa Lovász a Plummer domnievali, že počet perfektných párení bezmostového kubického grafu G, by mal s jeho počtom vrcholov exponenciálne rásť. Lovász–Plummerova hypotéza [6], nám hovorí nasledovné:

Existuje univerzálna konštanta $\epsilon > 0$ taká, že pre každý bezmostový kubický graf G,

$$2^{\epsilon|V(G)|} \leq |\mathcal{M}(G)| \leq 2^{|V(G)|}.$$

Voorhoeve dokázal túto hypotézu pre bipartitné kubické grafy [19]. Schrijver ju neskôr dokázal rozšíriť na všetky bipartitné regulárne grafy [16]. Pre planárne bezmostové kubické grafy, túto hypotézu dokázali Chudnovsky a Seymour [4].

Pre našu prácu je zaujímavý dôkaz toho, že každý bezmostový kubický graf G má najmenej $2^{|V(G)|/3656}$ perfektných párení [6], čo potvrdzuje Lovász–Plummerova hypotéza, kde to rastie exponenciálne.

1.3 Problémy

V tejto časti si povieme o niektorých problémoch párení vo všeobecnosti. To znamená, že sa pozrieme aj na iné problémy, ktoré budú trochu súvisieť, aj s našou problematikou.

Párenie v bipartitných grafoch

Všeobecný problém perfektného párenia je možné vyriešiť v polynomiálnom čase, ale niektoré súvisiace problémy sú NP-úplné. Dá sa dokázať, že obmedzená forma problému perfektného párenia pre bipartitné grafy je NP-úplná, kde to obmedzenie sa týka rozdelenia vrcholov grafu. Pre stupne vrcholov tri alebo menej, to bude stále NP-úplné. Pre stupne vrcholov, ktoré sú obmedzené na dva alebo menej, existuje polynomiálny algoritmus [15].

Problém zisťovania počtu párení

Zistiť počet párení v grafe je #P-úplné a to aj pre bipartitné grafy. Dokonca je #P-úplné, aj nájdenie počtu perfektných párení v bipartitných grafoch [18]. Na objasnenie toho, prečo je # pred triedou P-úplných problémov, tak je to preto, lebo to označuje triedu zaoberajúcu sa počítaním počtu akceptačných riešení problémov. Kasteleynova veta potom hovorí, že počet perfektných párení v planárnom grafe možno vypočítať pomocou algoritmu FKT v polynomiálnom čase. Taká malá poznámka, počet párení v grafe sa nazýva *Hosoyov index*.

Problém maximálneho párenia

Je to jeden zo základných problémov kombinatorickej optimalizácie, kde cieľom je nájsť párenie najväčšej možnej veľkosti. Tento problém má rôzne algoritmy na rôzne typy grafov. Pri neohodnotených grafoch je tento problém riešený Hopcroft-Karpovým algoritmom v čase $O(\sqrt{|V|}|E|)$ [1]. Existuje pravdepodobnostný (randomizovaný) algoritmus od Muchy a Sankowského, založený na algoritme rýchleho násobenia matíc, ktorý poskytuje zložitosť $O(|V|^{2.376})$ [13].

Ako sme už raz spomínali vyššie, že maximálne a perfektné párenie spolu súvisia, tak nám to potvrdzuje jedno porovnávacie tvrdenie. Toto tvrdenie hovorí, že problém hľadania maximálneho párenia nie je ťažší, ako problém hľadania perfektného párenia [13]. Presnejšie povedané, problém určenia maximálneho párenia možno znížiť v randomizovanom čase $O(n^{\omega})$ na problém nájdenia perfektného párenia, kde ω je exponentom najlepšie známeho algoritmu násobenia matíc. Poznáme algoritmy, ktoré si o trochu zlepšili časovú zložitosť, ale nevýhodou je, že fungujú len na určitých typoch grafov alebo fungujú len pre obmedzený počet vstupov.

Keby sme si to všetko zhrnuli, tak najviac práce a problematík spojených s pármi, sa uskutočnilo už na spomínaných bipartitných grafoch. To sú také grafy, ktorých množinu vrcholov je možné rozdeliť na dve disjunktné množiny tak, že žiadne dva vrcholy z rovnakej množiny nie sú spojené hranou.

Kapitola 2

Prehľad rôznych existujúcich prístupov

Táto kapitola sa zaoberá prehľadom rôznych existujúcich algoritmov na hľadanie perfektných párení.

Riešenie týchto problémov v polynomiálnom čase zostalo dlho nepolapiteľným cieľom, až kým Edmonds neprišiel s prvým algoritmom [5]. Následne bolo potom nájdených niekoľko ďalších algoritmov, ako napríklad od Micaliho a Vaziraniho. Vazirani je jeden z najdôležitejších ľudí v tejto problematike párení. Budeme ho ešte spomínať pri Rabin-Vazirani algoritme, ktorý je jeden zo základných algoritmov.

Rôzne algoritmy, ktoré tu budeme spomínať, tak budú väčšinou teoretické algoritmy, pri ktorých si ukážeme aj ich pseudokódy. Najskôr začneme tým, že si povieme niečo o Tutteho matici.

2.1 Tutteho matica

Predtým ako začneme hľadať nejaké perfektné párenie v grafe, mali by sme si položiť otázku, že či náš graf vôbec obsahuje nejaké perfektné párenie. S odpoveďou na túto otázku prišiel Tutte. K odpovedi nám dopomôže Tutteho matica, ktorú si vytvoríme pomocou vstupného grafu, v ktorom chceme nájsť perfektné párenie.

Definícia 2.1 (Tutteho matica). *Nech $G = (V, E)$ je neorientovaný jednoduchý graf s $|V| = n$. Potom Tutteho matica z G je $n \times n$ matica T so záznamami:*

$$T[i,j] = \begin{cases} 0, & \text{if } (i,\ j) \notin E \\ x_{ij}, & \text{if } (i,\ j) \in E \text{ and } i < j \\ -x_{ij}, & \text{if } (i,\ j) \in E \text{ and } i > j \end{cases}$$

kde x_{ij} sú formálne premenné (nie sú konkretizované konkrétnou hodnotou).

Ako vidíme z definície, týmto spôsobom si vytvoríme našu maticu T, kde formálne premenné, sú inak povedané neurčité premenné, ktoré sa používajú ako zástupné symboly pre rôzne objekty. V našom prípade sú to polynómy.

Nasledujúce Tutteho tvrdenie, ktoré vyslovíme je jadrom niektorých algoritmov pre perfektné párenie, o ktorých budeme ešte v tejto kapitole počuť. Ukázal v ňom veľmi elegantné spojenie medzi existenciou perfektného párenia a determinantom príslušnej Tutteho matice. Dôkaz si tu ukazovať nebudeme.

Jeho trvdenie znie, že ak máme neorientovaný jednoduchý graf $G = (V, E)$ s párnym $|V|$ a nech T je jeho príslušná Tutteho matica, tak potom $\det(T) \not\equiv 0$ práve vtedy, keď G má perfektné párenie [10]. Inak povedané, keď platí táto bijekcia:

$$det(T) \not\equiv 0 \iff G \text{ obsahuje perfektné párenie.}$$

Musíme si uvedomiť, že matica T nie je maticou čísel, ale dalo by sa povedať, že je to skôr formálna matica kvôli formálnym premenným, ktoré obsahuje. Determinantom Tutteho matice nie je číslo, ale polynóm premenných v matici. $\mathrm{Det}(T) \equiv 0$ sa myslí, že polynóm je identický nulovému polynómu, kde jeho koeficienty sú nula.

V tejto časti sme sa oboznámili s jednoduchým algoritmom, ktorý nám povie, či náš graf má nejaké perfektné párenie. Stačí vypočítať maticu T, vypočítať jej determinant a následne otestovať, či $\det(T)$ je nulový polynóm alebo nie je. Lenže T je formálna matica, takže výpočet $\det(T)$ môže trvať exponencionálny čas. Nemusíme nutne počítať polynóm zodpovedajúci $\det(T)$, my len potrebujeme otestovať, či ten polynóm je identický nule. Existuje na to efektívny spôsob, ale len za predpokladu, že použijeme pravdepodobnoť (randomizáciu). Bol to práve Lovasz [11], ktorý ako prvý navrhol použitie pravdepodobnosti (randomizácie) na tento probém. Hlavnou myšlienkou bolo nahradenie premenných v matici T náhodne vybranými číslami z polynomiálne veľkej množiny celých čísel. Ak na začiatku bola matica T singulárna, čo znamená, že jej determinant je nulový, v našom prípade nulový polynóm, tak aj determinant jej substitúcie s náhodne vybratými celými číslami bude nula. V opačnom prípade, ak matica T nie je singulárna, tak jej substituovaná matica tiež nebude singulárna s veľmi vysokou pravdepodobnosťou. Keďže sme nahradili maticu T celými číslami, tak jej determinant je možné vypočítať v polynomiálnom čase. Formálna pravdepodobnostná analýza tohto algoritmu je hlavne založená na jednom z Schwartzových tvrdení [17]. Ukážeme si preformulované tvrdenie, ktoré hovorí, že pre nenulový $\det(T)$ väčšina substitúcií premenných x_{ij} spôsobí, že $\det(T)$ bude nenulový.

Tvrdenie 2.2 (Schwartz-Zippel). *Predpokladajme, že $det(T) \not\equiv 0$, potom predpokladajme, že každá premenná v T bola rovnomerne náhodne nastavená na niektorý prvok z $\{1, ..., n^2\}$. Potom $\Pr[det(T)(x) = 0] \leq 1/n$, kde je náhodnosť prevzatá z nastavenia každej jednej premennej* [10].

Dôkaz tohto tvrdenia si v našej práci nebudeme uvádzať. O tomto algoritme vieme povedať, že s vysokou pravdepodobnosťou akceptuje grafy, v ktorých sa nachádzajú perfektné párenia. Na druhú stranu nikdy neakceptuje grafy bez perfektných párení. Opakovaním tohto algoritmu sa môže pravdepodobnosť zlyhania exponenciálne znížiť.

V ďalšej časti si ukážeme Rabin-Vaziraniho algoritmus, ktorý využíva, už náš spomínaný algoritmus s Tutteho maticou.

2.2 Rabin-Vaziraniho algoritmus

Konečne sme sa dostali k prvému algoritmu, ktorý nám nájde perfektné párenie. Rabin a Vazirani vyvinuli algoritmus, ktorý využíva rekurzívne Lovászov [11] algoritmus detekcie perfektného párenie. Predpokladajme, že sme zistili, že náš graf G má nejaké perfektné párenie a dokonca sme identifikovali hranu e, ktorá patrí do perfektného párenia. Jedno z kľúčových pozorovaní je, že podgraf G' z G vytvorený odstránením hrany e a všetkých susediacich hrán s e, má tiež perfektné párenie. Z toho dostávame, že akékoľvek perfektné párenie z G' v kombinácii s e vytvorí perfektné párenie pre G. Ak by sme mali nejakú metódu, ktorá by nám povedala, či je hrana v perfektnom párení, tak potom by sme mali rekurzívny algoritmus, ktorý by vytvoril perfektné párenie. Sú to Rabin a Vazirani, ktorí prišli s touto metódou, kde práve inverzia Tutteho matice obsahuje informácie o týchto hranách.

Rabin a Vazirani vyslovili tvrdenie, ktorého obsahom bolo, že nech máme neorientovaný jednoduchý graf G majúci perfektné párenie a T by bola príslušná matica grafu G, tak potom $(T^{-1})_{i,j} \neq 0$ práve vtedy, keď $G - \{i, j\}$ má perfektné párenie. Nakoniec toto tvrdenie aj dokázali [10]. Jediná vec, ktorá nemusí byť jasná je $(T^{-1})_{i,j}$ a ako ju dostaneme. Pre maticu T máme $(T^{-1})_{i,j} = \frac{(adj\ T)_{i,j}}{det\ T}$, kde $(T^{-1})_{i,j}$ sa nazýva adjungovaná matica a jej hodnotou je determinant matice T po odstránení i-teho riadku a j-teho stĺpca.

Musíme pripomenúť, že toto je pravdepodobnostný (randomizovaný) algoritmus, rovnako tak ako Lovászov algoritmus, z ktorého Rabin a Vazirani vychádzali. Rabin-Vaziraniho algoritmus dokáže nájsť nejaké perfektné párenie s konštantnou pravdepodobnosťou. Časová zložitoť algoritmu je $O(n^{\omega+1})$ na nájdenie perfektného párenie, pretože inverzia matice sa vypočíta v čase $O(n^{\omega})$ a vo všetkých ďalších operáciach, v každej z $O(n)$ iterácií vždy dominuje inverzia matice.

Nastal čas ukázať pseudokód algoritmu popísaného vyššie, ktorý rekurzívne odstraňuje hrany v perfektnom párení. Princíp algoritmu je v celku jednoduchý [10].

Existuje aj zrýchlený Rabin-Vaziraniho algoritmus od Muchy a Sankowského, ktorý si ukážeme v nasledujúcej časti.

Algorithm 1 Rabin-Vaziraniho algoritmus

1: $M \leftarrow \emptyset$
2: **while** G nie je prázdne **do**
3: Vypočítajte T_G a vytvorte inštanciu každej premennej s náhodnou hodnotou z $\{1, ..., n^2\}$
4: Vypočítajte T_G^{-1}
5: Nájdite i,j také, že $(v_i, v_j) \in G$ a $(T_G^{-1})_{i,j} \neq 0$
6: $M \leftarrow M \cup \{(v_i, v_j)\}$
7: $G \leftarrow G - \{v_i, v_j\}$
8: **end while**
9: **return** M

2.3 Zrýchlený Rabin-Vaziraniho algoritmus

V tejto časti zrýchlime vyššie spomínaný Rabin-Vaziraniho algoritmus. Bol to Mucha a Sankowski [13], ktorí tento algoritmus dokázali pozmeniť tak, že nakoniec inverzia matice T trvá $O(n^2)$ času a perfektné párenie sa nájde v čase $O(n^3)$.

Mucha a Sankowski si všimli, že Rabin-Vaziraniho algoritmus nie je efektívny, pretože pri každej iterácii opakovane počíta inverziu Tutteho matice, kde sú odstránené dva riadky a dva stĺpce, čo zodpovedá odstráneniu vrcholov i a j z grafu. Každé jedno prepočítanie inverzie matice je časovo nákladné a nevyuživa vzťah medzi Tutteho maticami pri jednotlivých iteráciach, kde $(r + 1)$ Tutteho matica označená ako T_{r+1} je jednoducho T_r s odstránenými dvoma riadkami a dvoma stĺpcami, ktoré zodpovedajú odstránením vrcholom i a j. Mal by existovať nejaký vzťah medzi T_r^{-1} a T_{r+1}^{-1}. Nakoniec tento vzťah existuje a boli to práve Mucha a Sankowski, ktorí ho našli. Ukázali, že T_{r+1}^{-1} sa dá získať z T_r^{-1} použitím Gaussovej eliminačnej metódy, čo je metóda slúžiaca na riešenie sústav lineárnych algebraických rovníc. Pomocou tohto kľúčového poznatku sa nám to podarí zrýchliť. Nasledujúce trvdenie nám k tomu dopomôže.

Tvrdenie 2.3 (Tvrdenie o eliminácii). *Nech A regulárna $n \times n$ matica, potom môžeme napísať $A = \begin{pmatrix} a_{1,1} & v^T \\ u & B \end{pmatrix}$ a $A^{-1} = \begin{pmatrix} \hat{a}_{1,1} & \hat{v}^T \\ \hat{u} & \hat{B} \end{pmatrix}$, $a_{1,1}$, $\hat{a}_{1,1}$ sú čísla, a zároveň u, v, $\hat{u}$, $\hat{v}$ sú vektory dĺžky $n-1$. Kde $\hat{a}_{1,1} \neq 0$, potom $B^{-1} = \hat{B} - \hat{u}\hat{v}^T/\hat{a}_{1,1}$* [13].

Pre upresnenie tvrdíme, že matica A je regulárna, čo znamená, že jej determinant je rôzny od nuly. Nakoniec tvrdenie o eliminácii nám ukazuje ako aktualizovať inverziu matice po odstránení riadku alebo stĺpca bez toho, aby sme ju znova počítali. Opísaná modifikácia $\hat{B}$ je v skutočnosti jediný krok Gaussovej eliminačnej metódy. V tomto prípade, ak sa myslí, že sĺpce sú premenné a riadky sú rovnice, tak to zodpovedá odstráneniu prvej premennej pomocou prvej rovnice. Z dôsledku vyššie spomínaného tvrdenia dostávame algoritmus pre perfektné párenia s časovou zložitosťou $O(n^3)$ [13].

Algorithm 2 *O(n³)* algoritmus na nájdenie perfektného párenia vo všeobecných grafoch

1: $A \leftarrow T_G^{-1}$
2: $M \leftarrow \emptyset$
3: **for** $i = 1$ to n **do**
4: nájdite j také, že $(v_i, v_j) \in G$ a $A_{i,j} \neq 0$
5: $M \leftarrow M \cup \{(v_i, v_j)\}$
6: $G \leftarrow G - \{v_i, v_j\}$
7: odstráňte i-ty riadok a j-ty stĺpec z A
8: odstráňte j-ty riadok a i-ty stĺpec z A
9: **end for**
10: **return** M

Nezabúdajme, že v tomto algoritme vždy, keď odstraňujeme hranu, tak musíme odstrániť všetky hrany susediace s obidvomi jej koncovými vrcholmi, čo znamená, že tento postup vykonávame dvakrát. Pomocou vyššie spomínaného tvrdenia o eliminácii, dokážeme vypočítať inverziu matice v čase $O(n^2)$ a s malou modifikáciou Rabin-Vaziraniho algoritmu, dostávame spomínaný $O(n^3)$ algoritmus na nájdenie perfektného párenia vo všeobecných grafoch.

Tento algoritmus môže byť modifikovaný, aby našiel perfektné párenie v bipartitných grafoch.

Kapitola 3

Testovanie

V tejto kapitole si porovnáme výsledky testovaní na dvoch implementovaných algoritmoch z predošlej kapitoly, z ktorých si vyberieme práve jeden, ktorý použijeme v našej vlastnej implementácii, za účelom jeho zrýchlenia. Prvý z nich bude algoritmus s Tutteho maticou v sekcii 2.1 kapitole 2 a druhý bude zrýchlený Rabin-Vaziraniho algoritmus v sekcii 2.3 kapitole 2. Implementáciu samostatného Rabin-Vaziraniho algoritmu sme nerobili, pretože je zjavne pomalší ako jeho zrýchlená verzia od Muchy a Sankowského. Implementácie týchto dvoch algoritmov sme sústredili do priečinka *zname_ algoritmy* v súbore *main.cpp*.

Testovanie pebieha na neorientovaných kubických grafoch [2] uložených v textovom súbore *26-kubi_ grafov*. Výsledné hodnoty zaznamenávame v mikrosekundách.

3.1 Algoritmus Tutteho matice

Implementovali sme jednoduchý C++ program, ktorého úlohou je zistiť, či existuje nejaké perfektné párenie v grafe. Základným princípom tohto programu je vytvoriť Tutteho maticu z daného grafu, na ktorej zavoláme funkciu, ktorá vypočíta jej determinant. Výsledok determinantu rôzny od nuly znamená existenciu nejakého perfektného párenie v grafe.

Pri testovaní rýchlosti meriame dĺžku trvania behu programu na jednotlivých grafoch z textového sôboru *26-kubi_ grafov*. Zo zozbieraných hodnôt si vyberieme *maximum*, *minimum* a vypočítame *priemer* a *medián* z daných hodnôt.

3.2 Zrýchlený Rabin-Vaziraniho algoritmus

Cieľ tohto algoritmu sa trochu odlišuje od vyššie spomínaného algoritmu, ale podstata zostáva rovnaká. Jeho úlohou je nájsť konkrétne jedno nejaké perfektné párenie nachádzajúce sa v grafe. Môžeme vidieť, že podstata sa nemení, pretože ak sa nájde

MIN	2
MAX	16
PRIEMER	6.53846
MEDIÁN	8

Tabuľka 3.1: Výsledné hodnoty testovania algoritmu Tutteho matice v mikrosekundách

konkrétne perfektné párenie, to znamená, že v grafe nejaké existuje. Týmto tvrdením chceme ukázať, že obidva algoritmy sú pre našu potrebu v zmysle funkčnosti ekvivalenté.

Rovnako sme implementovali tento algoritmus ako C++ program, ktorý si najskôr vytvorí Tutteho maticu T. Na matici zavoláme funkciu, ktorá vytvorí inverznú maticu pomocou jej adjungovanej matice a determinantu Tutteho matice.

$$T^{-1} = adj(T)/det(T)$$

Následne sa spustí prehľadávanie grafu *for* cyklom, kde hľadáma idexy i a j reprezentujúce vrcholy grafu, ktoré spĺňaju zadané podmienky. To znamená, že indexy i a j musia reprezentovať existujúcu hranu v grafe a na tomto políčku v inverznej matici musí byť hodnota rôzna od nuly. Ak sme našli indexy spĺňajúce stanovené podmienky, ich reprezentujúcu hranu pridáme do perfektného párenia. Postupne odstránime túto hranu spolu s jej susedmi z grafu, korý reprezentujeme maticou, čo znamená že z nej odstránime i-ty,j-ty riadok a i-ty, j-ty stĺpec. Rovnako tak odstránime tieto riadky a stĺpce z inverznej matice. Po skončení *for* cyklu sa opýtame, či dvojnásobok počtu hrán v perfektnom párení zodpovedá počtu vrcholov v grafe.

Rovnakým spôsobom testujeme aj tento algoritmus, z ktorého nás zaujíma *maximum*, *minimum*, *priemer* a *medián*.

MIN	42
MAX	923
PRIEMER	273.192
MEDIÁN	199.5

Tabuľka 3.2: Výsledné hodnoty testovania zrýchleného R-V algoritmu v mikrosekundách

Pri niektorých grafoch sa nám nepodarilo nájsť perfektné párenie, aj keď v grafe existovalo.

3.3 Výsledok testovania

Zhrnieme si výsledky testovania oboch algoritmov, z ktorých vyberieme rýchlejší. Vybratý algoritmus následne použijeme v implementácii nášho vlastného algoritmu pri hľadaní perfektných párení.

Pripomíname, že testovanie prebiehalo na noerientovaných kubických grafoch s najviac 14 vrcholmi, s rôznymi dĺžkami najkratších kružníc ≥ 3, až po najkratšie kružnice ≥ 6. Z každého typu najkratšej kružnice s rovnakým počtom vrcholov máme jeden alebo dva grafy.

Pre nás hlavné rozhodujúce údaje, ktoré porovnávame medzi sebou, sú *priemer* a *medián*. Z tabuliek oboch vyššie spomínaných algoritmov je jednoznačne vidieť, že algoritmus Tutteho matice mal omnoho lepšie výsledky *priemeru* a *mediánu*, vzhľadom na rýchlosť behu programu. Bol v celkovom výsledku spolahlivejší, neurobil žiadnu chybu.

Z výsledkou testovania si výberáme algoritmus Tutteho matice, ktorý sa stane našim pomocným algoritmom pri hľadaní všetkých perfektných párení.

Kapitola 4

Návrh a implementácia vlastného algoritmu

V tejto časti si navrhneme a popíšeme vlastnú implementáciu algoritmu, ktorý hľadá všetky perfektné párenia v kubických grafoch. Tento algoritmus potom následne použijeme aj na *snarky*.

4.1 Návrh algoritmu

Našim cieľom a cieľom tejto práce je navrhnúť a implementovať vlastný C++ algoritmus na hľadanie perfektných párení. Bude to rekurzívny program, ktorý pri nájdení hrany patriacej do perfektného párenia, ju pridá do množiny reprezentujúcej hrany perfetkného párenie. Ak postupne nájdeme všetky hrany určujúce nejaké perfektné párenie, zvýšime si počítadlo, ktoré určuje náš počet nájdených perfektných párení. Výstupom algoritmu bude číslo uvádzajúce počet všetkých perfektných párení nachádzajúcich sa v danom grafe.

Zjednodušene a zbežne si popíšeme funkčnosť algoritmu. Zo začiatku bude náš algoritmus jednoduchý rekurzívny program, ktorý keď dostane graf na vstupe, tak z neho vyberie a uloží si všetky jeho hrany. Následne ku každému jednému vrcholu si uložíme jeho susedné vrcholy, aby sme vedeli z ktorými vrcholmi je spojený hranou. Potom sa zavolá dôležitá rekurzívna funkcia, ktorej pošleme množinu hrán a vrcholy s ich susednými vrcholmi. Účelom tejto funkcie je hľadanie perfektných párení a keď sa jej podarí nejaké nájsť, tak sa zvýši počítadlo perfektných párení. Táto funkcia bude fungovať v dvoch fázach:

1. Ak má aktuálne perfektné párenie určitý počet hrán, to znamená, že sme jedno z nich našli, potom môžeme zvýšiť počítadlo. Inak si vyberieme existujúcu hranu z grafu, ktorú pridáme do perfektné párenia a odstránime všetky jej susediace hrany. Po odstránení sa rekurzívne zavoláme na funkciu, v ktorej sa nachádzame.

2. Po vrátení sa z vyššie spomínanej rekurzie v prvej fáze, odstránime vybratú hranu z grafu, čo znamená, že jej susedné hrany vrátime do pôvodného stavu. Túto hranu odstránime z perfektného párenia a znova sa rekurzívne zavoláme na funkciu, v ktorej sa nachádzame.

Týmto spôsobom pomocou rekurzie dokážeme prejsť celý graf, kde nájdeme všetky perfektné párenia. Postupne popridávame lokálne podmienky a prácu s Tutteho maticou, ktoré nám pomôžu zrýchliť výsledný algoritmus.

V programe využijeme aj *ba-graph* knižnicu, ktorá má zaujímavé a užitočné funkcie pre prácu s grafmi. V konečnom dôsledku bude náš algoritmus súčasťou tejto knižnice.

4.2 Používané grafy

Pre našu prácu sú dôležité neorientované kubické grafy a *snarky*. Keďže pracujeme s veľkým množstvom grafov, tak potrebujeme, aby boli uložené jednoduchým a kompaktným spôsobom vo formáte, ktorý nezaberá veľa miesta. Takýmito formátmi sú napríklad *graph6* a *sparse6*, ktoré slúžia na kompaktné ukladanie neorientovaných grafov a používajú iba tlačiteľné ASCII znaky. Súbory v týchto formátoch sú textového typu a obsahujú iba jeden riadok na graf.

- *graph6* – je vhodný pre malé grafy alebo veľké husté grafy
- *sparse6* – je priestorovo efektívnejší pre veľké riedke grafy

Na prácu s grafmi vo vyššie spomínaných formátoch, sme potrebovali vytvoriť niekoľko textových súborov s kubickými grafmi [2] a *snarkami* [3], na ktorých môžeme testovať náš algoritmus. Všetky grafy v týchto súboroch sú vo formáte *graph6*.

Textové súbory:

100-kubi_grafov – obsahuje 100 kubických (nie bipartitných) grafov s najviac až 38 vrcholmi. Nachádzajú sa tu grafy s najkratšími kružnicami ≥ 3 až po grafy s najkratšími kružnicami ≥ 8. Z každého typu najkratšej kružnice s rovnakým počtom vrcholov máme minimálne jeden a najviac tri grafy.

normal_snark_grafy – obsahuje 69 *snarkov* s najviac až 38 vrcholmi. Nachádzajú sa tu grafy s najkratšími kružnicami ≥ 4 až po grafy s najkratšími kružnicami ≥ 7 a zároveň grafy s cyklickou hranovou súvislosťou ≥ 5. Z každého typu najkratšej kružnice a typu s cyklickou hranovou súvislosťou sme vybrali jeden alebo dva grafy s rovnakým počtom vrcholov.

snarks_4-edge-critical – obsahuje 18 *snarkov* s najviac až 36 vrcholmi. Nachádzajú sa tu grafy, ktoré sú 4-hranovo kritické (4-edge-critical). Z grafov, ktoré majú rovnaký počet vrcholov sme zobrali jeden alebo dva grafy.

snarks_4-vertex-critical – obsahuje 18 *snarkov* s najviac až 36 vrcholmi. Nachádzajú sa tu grafy, ktoré sú 4-vrcholovo kritické (4-vertex-critical). Z grafov, ktoré majú rovnaký počet vrcholov sme zobrali jeden alebo dva grafy.

snarks_oddness_4 – obsahuje 10 *snarkov* s počtom vrcholov od 44 až po 52. Nachádzajú sa tu grafy, s nepárnosťou 4. Z grafov, ktoré majú rovnaký počet vrcholov, sme vybrali jeden alebo dva grafy.

Problém nastáva vtedy, keď chceme začať používať graf, ktorý je v jednom zo spomínaných formátov. Najskôr ho potrebujeme premeniť na niečo, s čím môžeme manipulovať a k tomu nám dopomôže *ba-graph* knižnica. Spomínaná knižnica v sebe obsahuje funkciu, pomocou ktorej dokážeme z grafu vo formáte *graph6* alebo *sparse6* vytvoriť objekt triedy *Graph*, ktorý reprezentuje graf v *ba-graph* knižnici.

```
1 std::string s = "GsXPGs";          //príklad grafu vo formáte graph6
2 Graph g(read_graph6_line(s));
```

Funkcia *read_graph6_line* z knižnice *ba-graph* dostane ako parameter string grafu v jednom zo spomínaných formátov, z ktorého vytvorí objekt triedy *Graph* a následne ho vráti. Po tomto procese, už sme schopný manipulovať s grafom *g* a spustiť na ňom algoritmus.

4.3 Implementácia

Implementáciu nášho algoritmu sme sústredili do priečinka s názvom *rekuzia*, kde sa nachádza v súbore *main.cpp*. Prípadné testovania, ktoré sa nachádzajú v tejto časti kapitoly prebiehali na stovke neorientovaných kubický grafoch z textového súboru *100-kubi_grafov*. Hodnoty testovania su zaznamenané v mikrosekundách.

Rozhodli sme sa, že náš vlastný algoritmus bude rekurzívny program napísaný v programovacom jazyku C++, pretože má niekoľko výhod. Jendnou z jeho hlavných výhod je rýchlosť, ktorá je pre nás veľmi dôležitá, pretože používame rekurziu. Výhodou je to, že môžeme využívať spomínanú *ba-graph* knižnicu, ktorá je tiež napísaná v programovacom jazyku C++ a pomôže nám pri našej implementácii. Obsah programu

sa skladá z dvoch hlavných funkcií a niekoľko pomocných funkcií, ktoré si rozoberieme. Hlavný cieľom je pracovanie na neorientovaných kubických grafoch a snarkoch. Tieto grafy sú uložené v *graph6* formáte.

Spracovanie vstupu

Spracovanie vstupu je jeden z najdôležitejších postupov. Náš program dostane ako vstup graf vo formáte *graph6*, z ktorého potrebujeme získať reprezentáciu grafu, na ktorej môžeme pracovať. Na tento prípad využívame užitočnú funkciu z *ba-graph* knižnice, ktorá si ako vstup zoberie graf vo vyššie spomínanom formáte a vráti nám objekt triedy *Graph*, čo je reprezentácia grafu v knižnici.

Získanie informácií z grafu

Jedna z hlavných funkcií programu, ktorej cieľom je získať potrebné informácie z grafu pre chod nášho programu. Funkcia *perfect_matchings_recursion* dostane ako parameter graf, ktorý sme získali spracovaním vstupu. Pre nás je dôležitý počet vrcholov, množina jeho hrán a zapamätať si jenoduchým spôsobom vrcholy a k nim hrany, ktorých sú súčasťou. Reprezentácia týchto informácií v našom programe:

- n – počet vrcholov v grafe
- M – vektor reprezentujúci množinu hrán (perfektné párenie)
- E – utriedená množina (set) reprezentujúca všetky hrany v grafe
- *adj_list* – zoznam reprezentujúci graf, kde ku každému jednému vrcholu existuje príslušny vektor s obsahom jeho susedných vrcholov

Množinu E a zoznam *adj_list* sme získali prejdením grafu po vrcholoch, v ktorých sú informácie o jeho susedoch. Následne potom zavoláme rekurzívnu funkciu.

Hľadanie perfektných párení

V tejto časti popíšeme hlavnú rekurzívnu funkciu *recursion*, ktorej úlohou je nájsť všetky perfektné párenia v grafe. Parametrami sú všetky potrebné informácie, ktoré sme získali v predošlej časti. Vo funkcii využívame algoritmus Tutteho matice, pomocou ktorej sa snažíme zrýchliť náš program. Dokáže nám ušetriť niekoľko zbytočných rekurzívnych vnorení. Perfektné párenia hľadáme rekurzívne v dvoch fázach:

1. Zoberieme hranu z E a pridáme ju do perfektného párenia M, odstránimu ju spolu s jej susednými hranami z E a rekurzívne sa zavoláme na samých seba. To znamená, že hľadáme všetky párenia obsahujúce vybratú hranu.

2. Po vynorení sa z predchádzajúcej fázy máme vybratú tú istú hranu ako predtým. Cieľom je teraz si povedať, že tá hrana nebude v perfektnom párení to znamená, že ju odstránime z M a všetky jej susedné hrany vrátime do E okrem našej vybratej hrany. Následne sa rekurzívne zavoláme na samých seba. To znamená, že sa snažíme nájsť všetky perfektné párenia, ktoré neobsahujú našu vybratú hranu.

Takýmto spôsobom prehľadáme celý graf a pokryjeme všetky možnosti súvisiace s hľadaním všetkých perfektných párení. Rozobereriem si podrobnejšie jednotlivé časti funkcie.

V úvode rekurzívnej funkcie sa pýtame, či veľkosť vektora $|M|\cdot 2 \neq n$ počtu vrcholov v grafe. Ak sa rovnajú, potom sme našli nejaké perfektné párenie a zavoláme funkciu *callback*, ktorej úlohou je zvýšiť počítadlo perfektných párení. Na druhú stranu, ak sa nerovnajú, to znamená, že sme nenašli všetky potrebné hrany. Nasleduje podmienka, ohľadom toho, či má E aspoň jednu hranu. Ak je E prázdna, vo funkcii sme skončili a začíname sa pomaly vynárať z rekurzie. Ak E obsahuje nejakú hranu, potom sa dostávame do najdôležitejšej časti, v ktorej sa nachádzajú aj spomínané dve hlavné fázy funkcie.

Dostávame sa ku kroku vyberania hrany z E. Musíme si uvedomiť, že vybratie správnej hrany dokáže zefektívniť náš program. Hrany uložené v množine (set) E sú zoradené od najmenšej po najväčšiu vzhľadom na čísla vrcholov, ktoré tvoria hranu. Testovali sme tri rôzne postupy vyberania hrán, za účelom optimalizácie programu:

(a) vyberanie hrán zo začiatku E

(b) vyberanie hrán z konca E

(c) vyberanie hrán na preskačku, raz zo začiatku, potom z konca E

Údaje	**(a)**	**(b)**	**(c)**
MIN	36	35	35
MAX	1580977547	1779495794	1511732711
PRIEMER	83700100	96115200	79959600
MEDIÁN	130970	100272.5	107257

Tabuľka 4.1: Porovnávanie rýchlosti vlastného algoritmu pri jednotlivých typoch vyberania hrán

Po dôslednom testovaní vidíme, že najrýchelší postup vyberania hrán je **(c)**, čo zodpovedá vyberaniu hrán na preskačku. Upozorňujeme, že sme testovali náš jednoduchý

rekurzívny program bez použitia Tutteho matice a lokálnych podmienok, ktoré by mali za následok rýchlejší beh programu.

Po vybratí hrany z E, nastáva prvá fáza, kde určenú hranu pridáme do M. Zavoláme funkciu *remove_neighbours*, ktorej ako parametre pošleme čísla vrcholov vybratej hrany, *adj_list* a množinu (set) *edges_to_remove*, do ktorej si uložíme všetkých susedov vybratej hrany. Funkčnosť funkcie *remove_neighbours* spočiva v nájdení a odstránení vybratej hrany a jej susedov z *adj_list* a následne pridaním týchto hrán do množiny (set) *edges_to_remove*.

Nasledujúcim krokom rekurzívnej funkcie je lokálna podmienka, pomocou ktorej zrýchlime hľadanie perfektných párení. Princíp tejto lokálnej podmienky spočíva v tom, že prehľadávame všetky vrcholy v zozname *adj_list* a keď narazíme na vrchol, ktorý susedí len s jedným vrcholom, čo znamená že ide z neho len jedna hrana, tak túto hranu pridáme do M. Keďže vieme že je to jediná hrana, tak určite musí patriť do perfektného párenia, pretože je to jediná cesta k danému vrcholu. Po prejdení zoznamu znova zavoláme funkciu *remove_neighbours*, aby sme upravili náš zoznam *adj_list* a doplnili hrany do množiny (set) *edges_to_remove*.

MIN	35
MAX	323192
PRIEMER	28747.8
MEDIÁN	2549.5

Tabuľka 4.2: Výsledné hodnoty testovania vlastného algoritmu s lokálnou podmienkou

Pomocou lokálnej podmienky sme dokázali zredukovať čas behu programu na 100 kubických grafoch z okolo dvoch hodín na niekoľko sekúnd. Upozorňujeme, že aktuálne testovanie prebiehalo na vlastnom rekurzívnom programe bez použitia Tutteho matice, ale s výberom hrán na preskačku a s použitím lokálnej podmienky.

Po pridaní hrán do množiny (set) *edges_to_remove*, začneme tieto hrany odstraňovať z E. Koniec prvej fázy zakončíme rekurzívnym volaním funkcie *recursion*.

Na začiatku druhej fázy odstráňujeme z M všetky hrany pridané v prvej fáze. Postupne pridáme hrany uložené v *edges_to_remove* naspäť do množiny E a zoznamu *adj_list*, okrem vybratej hrany z prvej fázy. Nezaradením tejto hrany naspäť do E znamená, že v grafe neexistuje a nikdy nebude v žiadnom perfektnom párení. Pokračujeme druhým rekurzívnym volaním funkcie *recursion*. Druhá fáza, ako aj celá funkcia sa zakončí vrátením zoznamu *adj_list* do začiatočného stavu.

Predošlý popis algoritmu neobsahoval prácu s Tutteho maticou. Jeho umiestnenie v programe bude na začiatku funkcie, hneď po podmienke zisťujúcej, či E obsahuje aspoň

jednu hranu. Dokážeme pokryť obidve rekurzívne volanie vo funkcii *recursion*. Potrebujeme zistiť, v akej časti prehľadávania grafu je najvýhodnejšie použiť algoritmus Tutteho matice. Pomocou neho dokážeme, zredukovať niekoľko zbytočných rekurzívnych volaní, ktoré by nenašli žiadne perfektné párenie. Týmto spôsobom zrýchlime náš výsledný algoritmus.

Efektívne zavolanie algoritmu Tutteho matice sme testovali na niekoľkých podmienkach:

(a) použitie vždy, keď sa nachádzame medzi 30% - 70% hrán v M

(b) použitie vždy, keď sa nachádzame v polovici M

(c) použitie vždy, keď sa nachádzame medzi 40% - 60% hrán v M

(d) použitie vždy, keď sa nachádzame medzi 50% - 90% hrán v M

(e) použitie vždy, keď sa nachádzame medzi 10% - 90% hrán v M

(f) použitie vždy, keď to bolo možné (bez podmienky)

(g) použitie vždy, keď aktuálny počet hrán v M bol deliteľný zaokrúhleným číslom znazorňujúcim 10% celkového počtu hrán v M

Ukážeme si 4 najefektívnejšie podmienky použitia Tutteho matice. Všetky ostatné výsledky testovaní sú sustredené v textových súboroch v prečinku *testovanie*.

Údaje	**(a)**	**(e)**	**(f)**	**(g)**
MIN	30	60	51	29
MAX	147407	123753	127861	164947
PRIEMER	16384	12211	11959.3	15230.4
MEDIÁN	2489	1761	1800	1714

Tabuľka 4.3: Porovnávanie rýchlosti vlastného algoritmu pri jednotlivých typoch podmienok na prácu s Tutteho maticou

Pri testovaní sme použili finálnu verziu programu, do ktorej patrila práca s Tutteho maticou a všetky vyššie popisované vlastnosti. Medzi najdôležitejšie zaznamenané hodnoty, podľa ktorých sa rozhodujeme sú *priemer* a *medián*. Pri rôznych testovaniach mali podmienky **(e)** a **(f)** veľmi podobné hodnoty. Rozhodli sme sa vybrať podmienku **(f)**, pretože v *priemere* dokázala zbehnúť na 100 kubických grafoch za menej ako 20 000 mikrosekúnd.

Pri každom úspešnom prejdení začiatočných podmienok vo funkcii *recursion*, sa dostaneme k podmienke, ktorá rozhodne, či pokračujeme v aktuálnej funkcii, alebo v nej končíme. V spomínanej podmienke zavoláme funkciu *has_perfect_matching* pýtajúc sa, či existuje perfektné párenie v aktuálnom stave grafu. Obsahom funkcie je vytvoriť Tutteho maticu, pomocou hrán v množínách M a E. Hrany z M dodajú matici reprezentáciu hrán v perfektnom párení, ktoré nemajú žiadnych susedov a hrany z E dodajú matici reprezentáciu hrán, ktoré sme ešte nenavštívili. Na výslednú maticu zavoláme funkciu *determinant*, ktorá vypočíta jej determinat. Ak je výsledný determinant rôzny od nuly, tak v grafe existuje perfektné párenie a môžeme pokračovať. Naopak nulový determinant znamená, že perfektné párenie neexistuje, vybrali sme zlé hrany do perfektného párenia.

4.4 SAT solver

Problém splnitelnosti booleovskej formuly (SATISFIABILITY alebo SAT) je úloha, ktorá sa snaží zistiť, či existuje splniteľná intrepretácia booleovskej formuly. Inými slovami, či premenné daného booleovského výrazu s premennými (formule) môžu byť nahradené hodnotami TRUE alebo FALSE takým spôsobom, že výsledný výraz bude pravdivý, inak povedané splniteľný. Booleovská fomula môže byť reprezentovaná v konjuktívnej normálnej forme (CNF) alebo v binárnych rozhodovacích diagramoch (BDD) [9].

SAT solvery sú softvéry riešiace splniteľnosť formúl. Existuje veľa rôznych variantov. Jeden z nich je napríklad AllSAT (All solutions SAT) solver, ktorý na rozdiel od normálneho SAT solvera, vygeneruje všetky možné riešenia. Každý SAT solver sa dá zmeniť na AllSAT solver, tým spôsobom, že keď nájdeme jedno riešenie, pridáme na vstupe obmedzujúcu podmienku, že toto riešenie, už nechceme hľadať, a snažíme sa nájsť ďaľšie. Pri veľkých množstvách riešení to prestáva byť efektívne.

Spomínaná *ba-graph* knižnica používa pri niektorý úlohách rôzne SAT solveri. V našom algoritme využívame jeden z jej SAT solverov a to AllSAT solver, pretože dokáže nájsť všetky riešenia, v našom prípade všetky perfektné párenia. Kontrolujeme si pomocou neho správnosť nášho výsledného algoritmu.

4.5 Zdrojové kódy

Zdrojové kódy *ba-graph* knižnice, ako aj zdrojové kódy príslušného AllSAT solvera *bdd_minisat_all-1.0.2*, sú dodané ako príloha, ktorú využívame v našich zdrojových kódoch. Prílohy sme žiadnym spôsobom nemodifikovali. Naše vlastné zdrojové kódy,

pre spomínané algoritmy sú umiestnené v súboroch *main.cpp*. Príslušné textové súbory obsahujúce rôzne typy grafov, sme sami vytvorili.

Kapitola 5

Zhodnotenie výsledného algoritmu

Táto kapitola sa zameriava na hodnotenie nášho výsledneho algoritmu. Budeme tu porovnávať výsledky na rôznych grafoch typu snark, spolu s výsledkami použitého AllSAT solvera.

5.1 Použitie na snarkoch

Porovnávame vlastnú implementáciu algoritmu na hľadanie všetkých perfektných párení, ktorú budeme v tejto kapitole nazývať "Rekurzia", s AllSAT solverom na snarkoch s rôznymi vlastnosťami. Všetky uvedené hodnoty sú myslené v mikrosekundách. Zaznamenávame si *maximum*, *minimum*, *priemer* a *medián* zo zozbieraných hodnôt.

Snarky

Testovanie prebiehalo na snarkoch z textového súboru *normal_snark_grafy*. Nachádzajú sa tu jednoduché bezmostové súvislé kubické grafy s chromatickým indexom rovným 4.

Údaje	**Rekurzia**	**AllSAT solver**
MIN	83	2128
MAX	77337	8852
PRIEMER	17109.2	4513
MEDIÁN	8582	4574

Tabuľka 5.1: Porovnávanie rýchlosti vlastného algoritmu s AllSAT solverom na jednoduchých snarkoch

Z testovania sme spozorovali, že náš algoritmus je efektívnejší pre grafy s menším

počtom vrcholov. Na druhú stranu pri väčších grafoch má jednoznačnú výhodu AllSAT solver.

Kritické snarky

V tejto časti sa budeme zaoberať 4-hranovo kritickým snarkom (4-edge-critical) a 4-vrcholovo kritickým snarkom (4-vertex-critical). Snark je 4-hranovo kritický práve vtedy, keď je kritický (critical). Snark je kritický, ak odstránením akýchkoľvek dvoch susedných vrcholov vznikne 3-hranovo zafarbiteľný graf (3-edge-colourable) graph. Snark je 4-vrcholovo kritický, ak je bikritický (bicritical). To znamená, že ak po odstránení dvoch rôznych vrcholov vznikne 3-hranovo zafarbiteľný graf.

Začneme najskôr na 4-hranovo kritických snarkoch.

Údaje	**Rekurzia**	**AllSAT solver**
MIN	156	2318
MAX	36567	7017
PRIEMER	12298.3	4075.06
MEDIÁN	8241.5	4140

Tabuľka 5.2: Porovnávanie rýchlosti vlastného algoritmu s AllSAT solverom na 4-hranovo kritických snarkoch

Testovanie nám ukázalo, že na snarku z najmenším počtom vrcholov v našom prípade 10, boli Rekurzia a AllSATsolver pomalší oproti jednoduchému 10 vrcholovému snarku z predošlého testovania. Obidva algoritmy, dokázali v priemere zbehnúť na týchto 18 grafoch za menej ako sekundu. Nasledujú 4-vrcholovo kritických snarky.

Údaje	**Rekurzia**	**AllSAT solver**
MIN	332	2585
MAX	44342	7484
PRIEMER	13168.1	4792.11
MEDIÁN	7811.5	4702

Tabuľka 5.3: Porovnávanie rýchlosti vlastného algoritmu s AllSAT solverom na 4-vrcholovo kritických snarkoch

Môžeme pozorovať rovnaké zhoršenie aj na týchto grafoch.

Po otestovaní obidvoch algoritmov na 4-hranovo kritických a 4-vrcholovo kritických snarkov, nám vyplýva, že algoritmy sa zhoršili vo všetkých skúmaných parametroch.

Musíme podotknúť, že výrazne zhoršenie nastalo na 4-vrcholovo kritických snarkoch, oproti testovaniu na 4-hranovo kritických snarkoch. V obidvoch testovaných súboroch bol rovnaký počet grafov s rovnakými počtami vrcholov.

Snarky s nepárnosťou 4

Nepárnosť 4, zodpovedá minimálnemu číslu, ktoré určuje počet nepárnych cyklov v 2-faktor kubickom bezmostovom grafe, v našom prípade snarku. Snarky majú nepárnosť najmenej dva.

Údaje	**Rekurzia**	**AllSAT solver**
MIN	161032	10006
MAX	718633	42398
PRIEMER	383484	23897
MEDIÁN	335914	19851

Tabuľka 5.4: Porovnávanie rýchlosti vlastného algoritmu s AllSAT solverom na snarkoch s nepárnosťou 4

Dôležitým poznatkom, z tohto testovania je, že AllSAT solver bol rýchlejší na grafe s najmenším počtom vrcholov ako náš algoritmus. Dokážeme to odôvodniť tým, že najmenší graf zhľadiska vrcholov mal teraz 44 vrcholov, čo posudzujume už ako jeden z väčších grafov. Obidva algoritmy boli pomalšie, čo je spôsobené aj tým, že sme ich testovali na väčších grafoch.

5.2 Zhodnotenie

Po všetkých vykonaných testoch na snarkoch a aj na kubických grafoch, nám vyšlo, že náš vlastný rekurzívny algoritmus na hľadanie perfektných párení, je efektívnejší na menších grafoch, zhruba do 15 vrcholov. Na druhú stranu AllSAT solveru sa darilo na väčších grafoch.

Z výsledkou testovania na snarkoch, sme zistili, že snarky so špecifickými vlastnosťami sa prehľadávali pomalšie, oproti jednoduchým snarkom.

V práci sme nestihli zodpovedať otázky typu, pre ako veľké grafy sú jednotlivé algoritmy použiteľné. Nestihli sme vyskúšať kombináciu nášho algoritmu s ALLSAT solverom na omnoho väčších grafoch.

Záver

Vytvorili sme vlastnú implementáciu algoritmu, ktorá úspešne hľadá všetky perfektné párenia v grafe. Grafy, ktoré nás v tejto práci zaujímali a na ktorých sme testovali implementované algoritmy, boli kubické grafy a snarky s rozličnými vlastnosťami. Implementovali sme dva známe algoritmy, z ktorých sme vybrali rýchlejší a použili sme ho na zefektívnenie nášho vlastné algoritmu. Z výsledkou testovaní na kubických grafoch a snarkoch, sme zistili, že náš algoritmus je výhodné používať na menších grafoch do 15 vrcholov. Na druhú stranu, pre väčšie grafy je lepšie použiť AllSAT solverom.

Celkové výsledky môžu byť vylepšené o experimentálne skombinovanie nášho výsledného algoritmu s AllSAT solverom. Môžu sa doplniť informácie, pre ako veľké grafy by boli jednotlivé algoritmy ešte použiteľné. Následne z týchto pozorovaní by sa dal vylepšiť výsledný algoritmus.

Literatúra

[1] Norbert Blum. A simplified realization of the hopcroft-karp approach to maximum matching in nonbipartite graphs. Technical report, Citeseer, 1999.

[2] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. Connected cubic graphs. `https://hog.grinvin.org/Cubic`, 2013. [Online; accessed 27-May-2020].

[3] Gunnar Brinkmann, Kris Coolsaet, Jan Goedgebeur, and Hadrien Mélot. Snarks. `https://hog.grinvin.org/Snarks`, 2013. [Online; accessed 27-May-2020].

[4] Maria Chudnovsky and Paul Seymour. Perfect matchings in planar cubic graphs. *Combinatorica*, 32(4):403–424, 2012.

[5] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[6] Louis Esperet, Frantisek Kardos, Andrew King, Daniel Kral, and Serguei Norine. Exponentially many perfect matchings in cubic graphs. *arXiv preprint arXiv:1012.2878*, 2010.

[7] Jean-Luc Fouquet and Jean-Marie Vanherpe. On fan raspaud conjecture. *arXiv preprint arXiv:0809.4821*, 2008.

[8] Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

[9] Sivaram Gopalakrishnan, Vijay Durairaj, and Priyank Kalla. Integrating cnf and bdd based sat solvers. In *Eighth IEEE International High-Level Design Validation and Test Workshop*, pages 51–56. IEEE, 2003.

[10] Ioana Ivan, Madars Virza, and Henry Yuen. Algebraic algorithms for matching. page 2–4, 2011.

[11] László Lovász. On determinants, matchings, and random algorithms. In *FCT*, volume 79, pages 565–574, 1979.

[12] Giuseppe Mazzuoccolo. The equivalence of two conjectures of berge and fulkerson. *Journal of Graph Theory*, 68(2):125–128, 2011.

[13] Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 248–255. IEEE, 2004.

[14] Viresh Patel. Unions of perfect matchings in cubic graphs and implications of the berge-fulkerson conjecture. Technical report, Citeseer, 2006.

[15] David A Plaisted and Samuel Zaks. An np-complete matching problem. *Discrete Applied Mathematics*, 2(1):65–72, 1980.

[16] Alexander Schrijver. Counting 1-factors in regular bipartite graphs. *Journal of Combinatorial Theory, Series B*, 72(1):122–135, 1998.

[17] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[18] Leslie G Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[19] Marc Voorhoeve. A lower bound for the permanents of certain (0, 1)-matrices. In *Indagationes Mathematicae (Proceedings)*, volume 82, pages 83–86. Elsevier, 1979.

[20] Wikipedia. Snark (graph theory) — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Snark%20(graph%20theory)&oldid=905606780`, 2020. [Online; accessed 23-January-2020].