Katedra Informatiky

Fakulta Matematiky, Fyziky a Informatiky

Univerzita Komenského, Bratislava

1-FAKTORIZÁCIE NA HYPERKOCKÁCH A STAR GRAFOCH

(Diplomová práca)

Matúš Kókai

Študijný odbor: Informatika

Vedúci: doc. RNDr. Rastislav Královič PhD.

Bratislava. 2010

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry a pod vedením diplomového vedúceho.

.....

Matúš Kókai

Táto práca mohla vzniknúť aj vďaka ústretovému prístupu pedagogického vedúceho doc. RNDr. Rastislava Královiča PhD. Touto cestou sa mu chcem poďakovať za čas venovaný konzultáciám pri tvorbe tejto práce, za poskytnutie materiálov a cenných rád, vedúcich k jej úspešnému zavŕšeniu. Tiež ďakujem svojej rodine a priateľom, a špeciálne svojej priateľke Zuzičke za podporu počas celého štúdia.

Abstrakt

V diplomovej práci sa zaoberáme problematikou generovania neizomorfných 1-faktorizácií (t.j. partícií hrán na perfektné párovania, 1-faktory) na regulárnych grafoch. Práca prináša základný prehľad pojmov, techník a dosiahnutých výsledkov v oblasti. Špeciálne sa zameriavame na hyperkocky a star grafy. Našim hlavným cieľom je navrhnutie algoritmov pre generovanie perfektných a semi-perfektných 1-faktorizácii na hyperkockách a star grafoch nižších dimenzií, ako aj implementácia heuristických metód do navrhnutých algoritmov. Na základe výsledkov testovania rôznych heuristík následne navrhujeme ich optimálnu kombináciu pre uvažované grafy vyšších dimenzií. Ďalším cieľom práce je nájdenie semi-perfektných a perfektných 1-faktorizacii na hyperkockách a star grafoch pomocou našich algoritmov. Súčasťou práce je aj program vhodný na testovanie a vyhodnocovanie účinnosti jednotlivých heuristických metód.

Kľúčové pojmy: 1-faktorizácia, semi-perfektnost, hyperkocka, star graf, izomorfizmus, heuristika

Abstract

In diploma thesis we focus on generating nonisomorphic 1-factorizations (partitions of edges into perfect matchings, 1-factors) on regular graphs. This thesis provides basic overview of terms, techniques and results gained in this field of study. Especially we focus on hypercubes and star graphs. The main goal of our research is the develepment of algorithms for generating perfect and semi-perfect 1-factorizations for hypercubes and stargraphs of lower dimension, as well as implementation of heuristics methods into algorithms. Based on the results of various heuristics we have proposed their optimal combination for graphs of higher dimensions. Additional goal of thesis is enumeration of semi-perfect and perfect 1-facotizations on hypercubes and stargraphs found by our algorithms. The part of this thesis is also useful software application for testing considered heuristic methods.

Keywords: 1-factorization, semi-perfect, hypercube, star graph, isomorphism, heuristic

Obsah

Kapitola 1

Úvod

V bežnom živote sa stretávame s mnohými problémami, ktorých riešenie je aj pri relatívne malej množine vstupných údajov a výpočtovej sile dnešných procesorov v reálnom čase nedosiahnuteľné. Na ilustráciu si uvedieme klasický problém priamo súvisiaci s tematikou našej práce. Uvažujeme turnaj športových mužstiev, v ktorom sa má stretnúť každé s každým. Koľkými spôsobmi je možné navrhnúť systém turnaja, ak rozvrhy, ktoré sa líšia len v poradí hracích dní, prípadne také, v ktorých vymeníme navzájom dva tímy považujeme za totožné? Pre dvanásť mužstiev existuje až 526 915 620 navzájom rôznych rozvrhov, pre štrnásť tímov presné číslo ešte nebolo vypočítané.

Problém izomorfizmu 1-faktorizácii, v našom príklade požadovaných navzájom rôznych rozvrhov, patrí do triedy NP-ťažkých rozhodovacích problémov, t.j. takých, ktorých nájdenie riešenia je zložitejšie ako rozhodnutie problému na nedeterministických Turingových strojoch v polynomiálnom čase. Dôsledkom príslušnosti do tejto triedy je závratný nárast času potrebného na rozhodnutie o izomorfizme vzhľadom na zväčšujúcu sa veľkosť vstupu, konkrétne grafu. Praktická využiteľnosť algoritmov riešiacich problém izomorfizmu je tak obmedzená na grafy s pomerne malým počtom vrcholov a hrán. Napriek veľkému úsiliu, doposiaľ nie je známy všeobecný algoritmus riešiaci problém v polynomiálnom čase.

Hoci je otázka izomorfizmu 1-faktorizacii, či grafov všeobecne NP-ťažká, v určitých prípadoch sme schopný s jej pomocou urýchliť vyriešenie iných zložitých problémov. Ak je tento situovaný do štruktúry umožňujúcej vhodnú klasifikáciu, správnym algoritmom prípadne heuristickou metódou sme schopný obmedziť čas jeho výpočtu. Pri otázke existencie riešenia nám často vzniká potreba prehľadania celej štruktúry. Jednu z možnosti, ako výpočet skrátiť, ponúka práve vlastnosť izomorfizmu. Ňou sa budeme v našej práci podrobnejšie zaoberať. Ukážeme si niektoré algoritmy na špeciálnych triedach grafov ako sú hyperkocky či star grafy, ako aj použitie heuristík na obmedzenie času ich výpočtu. Na základe experimentálnych výsledkov následne navrhneme metódy a heuristiky pre algoritmy na štruktúrach vyšších dimenzií.

Kapitola 2

1-faktorizácie na regulárnych grafoch

V nasledujúcich pasážach si uvedieme základne definície a pojmy súvisiace s našou témou, nevyhnutné pre ďalšie kapitoly. Uvedieme si aj prehľad výsledkov dosiahnutých pre 1-faktorizacie na regulárnych grafoch.

2.1 Základné definície

V práci sme uvažovali len o konečných jednoduchých grafoch G bez cyklov a orientovaných hrán. Základne terminológie a vlastnosti sú uvedené v [1]. Bez ujmy na všeobecnosti môžeme označiť n vrcholov G číslami z postupnosti 0,1, ..,n-1.

Definícia 2.1.1. *1-faktor grafu G je podmnožina jeho hrán pokrývajúca každý vrchol z G práve raz.*

Definícia 2.1.2. *1-faktorizácia G je partícia množiny hrán E na 1-faktory.*

1-faktorizacia $F=(f_1,...,f_n)$ sa nazýva *semi-perfektná*, ak $f_1 \cup f_k$ tvoria v G Hamiltonovskú kružnicu pre všetky $1 < k \leq n$. *Perfektná* sa nazýva taká, ktorej zjednotenie

ľubovoľných dvoch navzájom rôznych 1-faktorov tvorí Hamiltonovskú kružnicu. V ďalšom texte budeme pod pojmami faktorizácia a faktor mať na mysli 1-faktorizáciu resp. 1-faktor.

Ďalej budeme hovoriť o faktorizacii F aj ako o ofarbení hrán grafu G , kde 1-faktory predstavujú jednotlivé farebné partície. Zadefinujeme si teraz lexikografické usporiadanie. Hrana e je usporiadaná dvojica (a, b) kde $0 \leq a < b < n$. Pre ľubovoľné dve hrany $e_1 = (a_1, b_1)$ a $e_2 = (a_2, b_2)$ označme $e_1 < e_2$ ak platí $a_1 < a_2$, alebo platí $a_1 = a_2$ a súčasne $b_1 < b_2$. 1-faktor $f = (e_1,...,e_k)$ zapisujeme ako usporiadanú postupnosť hrán, kde $e_i < e_j$ pre všetky $i < j$. Pre ľubovoľné dva 1-faktory $f_i = (ei_1,...,ei_k)$ a $f_j = (ej_1,...,ej_k)$ označíme $f_i < f_j$ práve vtedy, keď existuje $1 \leq t$ také, že platí $ei_s = ej_s$ a zároveň $ei_t < ej_t$ pre všetky $s < t$. Faktorizácie zapisujeme ako usporiadanú postupnosť faktorov. Pre ľubovoľné dve 1-faktorizácie $F = (f_1,...,f_j)$ a $H = (h_1,...,h_j)$ označíme $F < H$ práve vtedy, keď existuje $1 \leq t$, také že $f_t < h_t$, a $f_s = h_s$ pre všetky $s < t$. Faktrorizáciu, resp. faktor nazveme *kanonickou*, ak je lexikograficky najmenšia. 1-faktor $f_{i=}(e_1,...,e_k,...)$ nazývame *parciálny*, ak existuje t také, že $e_j = Ø$ pre všetky $t < j$. Faktorizáciu $F_i = (f_1,...,f_i)$ označujeme *parciálnou*, ak neobsahuje všetky hrany z E. Index i označuje *rád* parciálnej F.

Definícia 2.1.3. *Dve 1-faktorizácie $F = \{f_1,...,f_j\}$ a $H = \{h_1,...,h_j\}$ sú izomorfné, ak existuje bijektívne zobrazenie Φ medzi vrcholmi grafu G, ktoré mapuje 1-faktory na 1-faktory. Inými slovami $\{f_1\Phi, f_2\Phi, ..., f_n\Phi\} = \{h_1, h_2,...,h_n\}$ kde $f_i\Phi = h_i$ je množina hrán $(a\Phi, b\Phi)$ takých, že (a, b) je hrana v f_i.*

Automorfizmus grafu G je izomorfizmus G na seba samého. *Automorfná stabilizačná grupa* $Aut(G)$ je množina všetkých permutácií γ zobrazujúcich G na G. Formálne, $Aut(G)=\{\gamma \in S_n \mid G^\gamma = G\}$, kde S_n je symetrická grupa o n prvkoch - množina všetkých ich permutácií. Grupa automorfizmu ofarbeného grafu (G, π) je množina všetkých permutácií $p \in S_n$, takých že $G^\gamma = G$ a $\pi^\gamma = \pi$. Hovoríme, že γ zachováva ofarbenie. V ďalšom texte už budeme označením S_n označovať star graf dimenzie n.

Pre partíciu vrcholov $V = \{0,...,n-1\}$ $\pi = (V_1, ..., V_k)$, nech je $c(\pi)$ partícia tvaru

($\{0,1,..,|V_1|-1\},\{|V_1|,..,|V_1|+|V_2|-1\},..,\{n-|V_k|,..,n-1\}$). *c* je teda partícia s rovnako početnými množinami ako π, v rovnakom poradí, ale inak je nezávislá od π. Diskrétnou partíciou nazývame takú, ktorej mohutnosť každej množiny je 1.

Definícia 2.1.4. *Canocical labeling map (kanonická mapovacia funkcia) je funkcia C, kde pre ľubovoľný graf G, partíciu vrcholov V π, a permutáciu γ platí:*
(1) C(G, π) = G na gamu, pre nejakú permutáciu gama takú, že $\pi^{\gamma} = c(\pi)$
(2) $C(G^{\gamma}, \pi^{\gamma}) = C(G, \pi)^{\gamma}$

2.2 Dosiahnuté výsledky pre 1-faktorizácie

Hľadanie 1-faktorizacii na grafoch je predmetom štúdií už viac ako 100 rokov. Problematikou sa zaoberalo veľké množstvo vedcov, výsledkom čoho je dnes obrovský počet publikácií a literatúry. Väčšina z nich sa zaoberá faktorizáciou na kompletných grafoch, nakoľko majú najbližšie k problémom každodenného života. Existencia 1-faktorizacie na kompletnom grafe K_{2n} je ľahko viditeľná. Počet navzájom neizomorfných sa už javí ako zložitý problém, súvisiaci najmä s exponencionálnym nárastom riešení so zvyšujúcim sa počtom vrcholov. Do dnešnej doby boli neizomorfné 1-faktorizacie spočítané len pre K_{2n} kde $n \leq 6$, pre vyššie rady boli vyslovene približne odhady. V K_2, K_4, K_6 existuje len jedna neizomorfná faktorizácia, v K_8 6, v K_{10} už 396. V prípade K_{12} ich existuje až 526 915 620 navzájom neizomorfných [2] [3].

Hypotéza 2.2.1 (A.Kotzig): *Pre každé* $n \geq 2$ *kompletný graf* K_{2n} *umožňuje taký rozklad hrán na n-1 1-faktorov, že ľubovoľné dva z nich tvoria perfektné párovanie.*

V našej práci sa primárne zaoberáme pravé semi-perfektnými a perfektnými 1-faktorizaciami. Z doposiaľ dosiahnutých výsledkov sa zdá, že medzi nimi nie je priama súvislosť.

A.Kotzig vyslovil predpoklad o existencii perfektnej 1-faktorizácie pre kompletný graf

K_{2n} pre všetky $n \geq 2$ [4]. Táto hypotéza je stále otvorená, a hoci bol jej dôkaz pod dlhé roky predmetom mnohých štúdií, podarilo sa ju dokázať len pre n = {16, 28, 36, 40, 50, 52, 126, 170, 244, 344, 730, 1332, 1370, 1850, 2198, 3126, 6860}, pre prvočíselné n a pre prvočísla tvaru 2n-1 [3]. Do dnešnej doby boli perfektné 1-faktorizácie exaktne vyčíslené len pre kompletné grafy malých radov. Pre K_4, K_6, K_8, K_{10} existuje jediná, pre K_{12} 5 [3] pre K_{14} 23 [5]. Hoci sa hypotézu zatiaľ nepodarilo dokázať, nenašiel sa ani taký graf K_{2n}, ktorý by ju vyvracal.

Našu pozornosť sme primárne upriamili na hyperkocky a star grafy. O niektorých ich vlastnostiach je podrobnejšie popísané v kapitole 4. *N-dimenzionálna binárna hyperkocka* Q_n je graf, ktorého vrcholy tvoria binárne vektory dĺžky n. Dva jej vrcholy susedia práve vtedy, keď sa ich vektory líšia práve v jednom bite. Existencia faktorizácie pre všetky regulárne grafy umožňujúcu Hamiltonovskú dekompozíciu, kam patria aj hyperkocky, bola ukázaná v [3].

Hypotéza 2.2.2 (D. Craft): *Pre každé* $n \geq 2$ *existuje semi-perfektná 1-faktorizácia pre* Q_n.

Craftova hypotéza je podobne ako Kotzigova stále otvorená. Bola dokázaná pre nepárne dimenzie n [6] [7], riešenia pre n = 6 je možné nájsť v [7]. Myšlienka dôkazu pre hyperkocky nepárnej dimenzie Q_{2k+1} je založená na známom fakte, že pre každé $k \geq 2$ je Q_{2k} rozložiteľná na k Hamiltonovských kružníc [8] [9] , a že Q_{2k+1} je zložená z dvoch kópií Q_{2k}, ktorých príslušné kópie vrcholov sú navzájom spojené hranami. Vhodnou kombináciou tohto 1-faktora spájajúceho dve hyperkocky Q_{2k} a faktorov obsiahnutých v Hamiltonovských kružniciach v samotných Q_{2k} vznikne semi-perfektná 1-faktorizácia. V [6] bolo tiež ukázané, že semi-perfektné faktorizácie existujú pre všetky K_{2n} a torusi $T_{2n} \times T_{2n}$.

Jedným zo spôsobov ako sa postupne dopracovať k dôkazu silnej Kotzigovej či Craftovej hypotézy je postupne dokázať slabšie tvrdenia, a vytvoriť tak základ pre budúce štúdie. V závere práce voľne vyslovíme tvrdenia založené na výpočtoch na uvažovaných grafoch nižších dimenzií, naznačujúce možnosť či nemožnosť existencie perfektných a semi-perfektných 1-faktorizacii pre vyššie dimenzie.

Kapitola 3

Metódy neizomorfného generovania

Ako sme spomenuli v úvode, otázka izomorfizmu 1-faktorizacii, ako aj grafov, je NP-ťažký problém. Dnes známy všeobecný algoritmus ma časovú zložitosť $exp(O(\log n))$. Rozhodovanie o izomorfizme objektov väčšinou zahŕňa hľadanie generátorov stabilizačnej grupy automorfizmu, jej kanonických (najmenších) zástupcov, či kanonickú mapovaciu funkciu. Na základe týchto faktorov je následne možné zodpovedať otázku izomorfizmu.

3.1 Typy metód generovania neizomorfných výsledkov

Techniky neizomorfného generovania spadajú do viacerých oblasti, v závislosti od typu generovaného objektu. Hlavne pri zložitejších typoch sa často využívajú konverzie na štruktúry čí objekty, pre ktoré sú metódy generovania známe a preskúmané. Grafy v porovnaní s grupami, partíciami či inými jednoduchšími kombinatorickými objektami patria medzi tie zložitejšie, a ich neizomorfné generovanie je až na niektoré špeciálne triedy výpočtovo náročné.

Vo všeobecnosti rozlišujeme dve hlavné fázy : generovanie výsledkov (napr. zástupcov izomorfných tried), a následne priebežne vylučovanie nevyhovujúcich daným podmienkam. Uvedieme si základné typy metód a techník. Rozdiely medzi nimi nie sú vždy jednoznačné,

často sa pri konkrétnej implementácii stretávame s ich kombináciou. Oblasť zahŕňajúca generovacie algoritmy sa rozširuje tak rýchlo, že niektoré najnovšie metódy nie je možné jednoznačne klasifikovať do niektorej z nižšie uvedených tried.

Nižšie uvedené techniky sú založené na výpočte známom ako backtrack (obr 4.2.1), no je možné ich využiť aj na väčších štruktúrach nevhodných pre tento spôsob prehľadávania.

3.2 Globálna štruktúra výsledkov

Všeobecná metóda, ukladajúca si informácie o triedach izomorfných objektov generovaných v priebehu výpočtu. Údaje ako generátory týchto tried, ich kanonických zástupcov, prípadne príslušnú kanonickú funkciu si ukladá do globálnej štruktúry.

Výhody :

- rýchla a jednoduchá implementácia

Nevýhody :

- v prípade distribuovaného výpočtu, k množine globálnych výsledkov potrebuje naraz pristupovať viac procesorov kvôli rozhodovaniu o izomorfizme, prípadne ďalším heuristikám, dôsledkom čoho je náročná paralelizácia
- pri mohutnejších štruktúrach si veľký počet výsledkov vyžaduje veľa pamäte, ako aj vhodnú metódu rýchleho prístupu k nim napr. dobrú hashovaciu funkciu.
- každý jeden potencionálny výsledok je nutné porovnávať s doterajšími uloženými výsledkami, hlavne kvôli možnému orezaniu prehľadávacieho stromu.
- miesto vhodných invariantov metóda hľadá kanonickú mapovaciu funkciu ku každému potencionálnemu výsledku potrebnú pre rozhodnutie o izomorfizme, čo je značne neefektívne

3.3 Metóda kanonických reprezentantov

Metódu nezávisle od seba predstavili Faradžev [10] a Read [11]. Princíp metódy je založený na postupnom (orderly) generovaní objektov. Algoritmus backtracku predpokladá kanonickosť parciálnych výsledkov, pre ktoré následne hľadá ďalšie možné rozšírenie. V prípade nekanonickosti vygenerovaného objektu vzhľadom na jeho triedu izomorfizmu, je tento v ďalšom kroku výpočtu opomenutý. Príklad implementácie metódy pre generovanie neizomorfných 1-faktorizacii v K_{12} je možné nájsť v [2].

Výhody :

- metóda priamo neimplikuje potrebu globálneho ukladania výsledkov
- test izomorfizmu je možné vykonať lokálne, na základe kanonickosti kandidáta, vďaka čomu je možné výpočet distribuovať
- pre navzájom rôzne uzly výpočtového stromu nevzniká potreba vzájomnej komunikácie
- ľahko sa implementujú invarianty, prípadne iné heuristiky na orezanie výpočtového stromu
- ako konečné výsledky sú generované len kanonickí zástupcovia jednotlivých tried izomorfizmu

Nevýhody :

- metóda sa ťažko implementuje pre netriviálne automorfné grupy určujúce izomorfné triedy
- testovanie kandidáta na kanonicitu je výpočtovo náročne

3.4 Metóda kanonického rozširovania

Metóda bola predstavená B.D. McKayom roku 1986, podrobný popis sa nachádza v [12]. Generované sú len neizomorfné objekty, deje sa tak ale spôsobom kanonického rozširovania – na rozdiel od kanonických reprezentantov danej triedy izomorfizmu pri predošlej metóde, je pre každú takúto triedu objekt generovaný kanonickým spôsobom a to za pomoci kanonickej mapovacej funkcie.

Výhody :

- výpočtovo náročnému testu na izomorfizmus môžu predchádzať ľahšie vypočítateľné invarianty, vďaka čomu môže byť tento obmedzený len na malú podmnožinu potencionálnych kandidátov
- otázka kanonickosti rozširovania objektu je rozhodnuteľná v rámci komunikácie s rodičovským uzlom stromu konkrétneho výpočtu, čoho dôsledkom je ľahká paralelizácia
- prípadný automorfizmus objektu sa týka len možných rozšírení z neho, čím nevzniká potreba komunikácie rôznych vetiev výpočtu
- kanonická funkcia môže vylúčiť možné rozšírenia objektu za predpokladu že tieto nie sú z neho kanonicky rozšíriteľné
- rovnako ako pri metóde kanonických zástupcov sú generované len neizomorfné výsledky
- v porovnaní s klasickým generovaním, s použitím prípadných heuristík poskytujú lepšie časy a menšie pamäťové nároky

Nevýhody :

- ťažko sa implementuje
- rozhodovanie o kanonickom rozširovaní je výpočtovo náročné

3.5 Metóda homomorfizmu

Bola predstavená Grunerom, Lauem a Meringerom [13]. Metóda je využívaná hlavne pri generovaní a klasifikovaní špeciálnych štruktúr s vopred známymi stabilizačnými automorfnými grupami objektov. Využíva jednoduché operácie na grupách, na základe ktorých generuje neizomorfné podobjekty rodičovského objektu. Metóda sa často využíva aj v iných oblastiach ako v informatike napr. pri otázke izomorfizmu chemických štruktúr.

Výhody :

- homomorfizmus mapuje objekt na jeho automorfnú grupu, čo poskytuje možnosť rýchlej klasifikácie objektu vzhľadom na jeho automorfnú grupu
- využívajú sa jednoduché grupové operácie nenáročné na výpočet

Nevýhody :

- metóda je vhodná pre štruktúry s vopred známymi stabilizačnými grupami
- nutnosť využitia ďalších metód a heuristík v určitých nerozhodnuteľných situáciách

Kapitola 4

Algoritmy pre Q_n a S_n

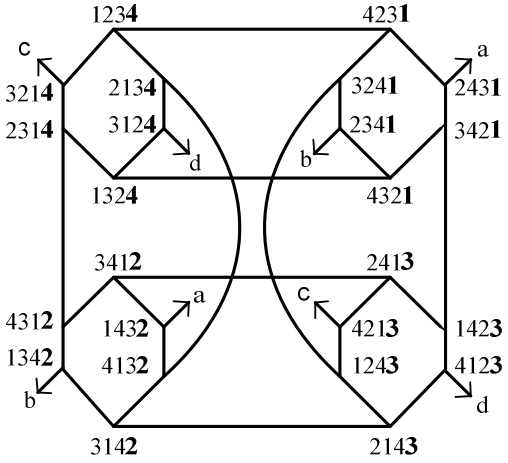
V nasledujúcej kapitole si najskôr ukážeme niektoré zaujímavé vlastnosti hyperkociek a star grafov. Základné terminológie a pojmy z teórie grúp je možné nájsť v [1]. Predstavíme si nami navrhnuté algoritmy pre generovanie neizomorfných 1-faktorizacii na týchto grafoch. Následne vymenujeme a popíšeme uvažované heuristiké metódy ako aj ich implementáciu v algoritmoch. Namerané výsledky uvedieme v kapitole 5.

4.1 Hyperkocky, star grafy a ich vlastnosti

V dnešnej dobe využíva množstvo paralelných výpočtov topológiu hyperkociek umožňujúcu efektívnu komunikáciu medzi procesormi. Zvlášť ak sa dajú naprogramovať spôsobom, ktorý vyžaduje vzájomné vymieňanie informácii len medzi susednými procesormi. S vyšším počtom vrcholov, t.j. procesorov priamo úmerne rastie aj dĺžka komunikačnej cesty. Podrobne sa skúmajú topológie s čo najlepšou komunikačnou cenou t.j. priemerom grafu v závislosti od rastúceho počtu vrcholov. Medzi takéto topológie patria star grafy S_n.

Q_n aj S_n zaraďujeme do množiny Cayleyho grafov, ktoré sú definovane nasledovne: pre dané generátory $g_1,..,g_k$ konečnej grupy H, sú vrcholmi grafu prvky grupy, a hranami sú spojené také vrcholy a a b, že $ag_i = b$ pre $1 \leq i \leq k$. Hrany Cayleyho grafu predstavujú akcie

generátorov na jednotlivých vrcholoch. Ak *H* je symetrická grupa *n* prvkov a generátory sú transpozície prvého a ľubovoľného prvku, ide o star graf S_n.Užitočnými vlastnosťami triedy Cayleyho grafov sú ich vrcholové a hranové symetrie. Q_n a S_n sú navyše silne hierarchické, čo znamená že pre ľubovoľné usporiadané generátory $g_1, ..., g_k$ sú rozložiteľné na grafy nižších dimenzií, pričom *i*-ty podgraf je generovaný všetkými generátormi okrem g_i. Star graf S_n môže byť reprezentovaný ako *n* navzájom spojených grafov S_{n-1} grafov $G_1,..,G_k$, pričom G_i obsahuje vrcholy, ktoré končia symbolom *i*. (obr. 4.1.1) Dekompozíciu Q_n na dve kópie Q_{n-1} sme spomenuli v úvode.



Obr. 4.1.1: Star graf S_4

Hyperkocka Q_n má 2^n vrcholov a $n \cdot 2^{n-1}$ hrán, *stupeň* (degree) aj *priemer* (diameter) Q_n je *n*. Star graf S_n má *n*! vrcholov a $n!\frac{n-1}{2}$ hrán, stupeň grafu je *n*-1 a jeho priemer $\left\lfloor 3\frac{n-1}{2} \right\rfloor$ [14]. Všimnime si, že stupeň vrcholov star grafu aj jeho priemer s rastúcim počtom vrcholov rastie pomalšie ako pre Q_n. S_n má pri *n*! vrcholoch priemer aj stupeň $O(n)$, kým hyperkocka s $O(n)$ vrcholmi má stupeň aj priemer $O(n\log n)$. Star grafy tak dokážu spojiť rovnaký počet vrcholov s menšou komunikačnou cenou a menším priemerom ako hyperkocky, a predstavujú voči nim veľmi atraktívnu alternatívnu topológiu.

Naše algoritmy pre semi-perfektné a perfektné 1-faktorizacie boli spúšťané práve na týchto dvoch grafoch nižších dimenzií.

4.2 Algoritmy pre 1-faktorizácie pre Q_n a S_n

Navrhnuté algoritmy generovania neizomorfných 1-faktorizácií sú založené na klasickom backtracku (obr 4.2.1), ktorý na výpočet využíva hrubú silu pre prehľadanie celého grafu. My sa ho budeme snažiť v našich algoritmoch obmedziť tak, aby nevynechal žiadny výsledok a zároveň sa zrýchlil jeho výpočet. Vo všeobecnosti existujú dva spôsoby generovania 1-faktorizácii grafu *G*, a to spôsobom vrchol za vrcholom, alebo 1-faktor za 1-faktorom. Priblížime si obidva.

```
procedure Backtrack (r1,..,rs : čiastočné riešenie) :
(1): if r1,..,rs je vhodné riešenie then
        Return r1,..,rs
(2): else
(3):    z r1,..,rs vyrátaj množinu ďalších možných riešení M vyhovujúcich problému
(4):    for pre všetky m ∈ M
(5):            Backtrack (r1,..,rs,m)
        end for
 end if
end procedure
```

Obr. 4.2.1: Backtrack

Výpočet našich algoritmov na grafe *G* budeme reprezentovať ako strom. Jeho vrcholy budeme kvôli jednoznačnosti nazývať uzlami. Koreň predstavuje počiatok výpočtu. Listy výpočtového stromu budú predstavovať konečné riešenia, t.j. úplné 1-faktorizácie.

Budeme sa tiež zaoberať implementáciou niekoľkých heuristík v rôznych uzloch výpočtového stromu algoritmu a skúmať ich efekt na dĺžku výpočtu, pamäťové nároky ako aj výslednú množinu výsledkov a nutnosť jej ďalšieho spracovania. Na základe našich výsledkov následne navrhneme vhodné postupy a optimalizácie pre výpočty na hyperkockách a star grafoch vyšších dimenzií.

Algoritmy budú postupne (orderly) prechádzať grafmi a hľadať vyhovujúce riešenia. V rozličnej miere kombinujú metódu ukladania čiastkových výsledkov do globálnych štruktúr

v určitých uzloch výpočtového stromu s modifikovanou metódou kanonických reprezentantov. Zmena oproti metóde kanonických reprezentantov spočíva v hľadaní prvého zástupcu určitej izomorfnej triedy, nie nutne kanonického. Pri vhodnej úprave sú tak výpočty ľahko paralelizovateľné a distribuovateľné medzi viacero procesorov.

Počiatočné podmienky algoritmov:

Vrcholy v Q_n sú označené číslami v desiatkovej sústave prislúchajúcimi ich binárnym vektorom. Hrany označíme číslami z postupnosti 0,...,$n \cdot 2^{n-1}$-1 nasledovným spôsobom: Začneme vo vrchole 0, a hranu ktorú má s najmenším susedom (0,1) označíme ako 0. Ako hranu 1 označíme tú, ktorá patrí vrcholu 0 a jeho druhému najmenšiemu susedovi, čiže (0,2). Podobne označíme všetky hrany incidentné s vrcholom 0, a postupne prejdeme do lexikograficky väčších vrcholov Q_n, pričom označujeme už len hrany, ktoré označené ešte neboli.

Permutácie predstavujúce vrcholy v S_n zoradíme od najmenšej po najväčšiu, a postupne označíme číslami z postupnosti 0,...,n!-1. Hrany v S_n očíslujeme číslami z postupnosti 0, ..., $n!\frac{n-1}{2}-1$ rovnakým spôsobom ako pri hyperkockách.

Faktorizáciu $F = (f_1,..,f_k)$ nazveme *vhodnou* (proper), ak jej i-ty faktor začína i-tou hranou susediacou s vrcholom 0. Pre každú 1-faktorizáciu H, ktorú sme našou myšlienkou vylúčili, existuje nejaká vhodná F z množiny všetkých 1-faktorizácií grafu G, nájdených pri úplnom prehľadaní grafu, pre ktorú platí $F=H^{\gamma}$, kde γ je permutácia 1-faktorov v H. Ďalej budeme uvažovať len o vhodných faktorizáciach, z čoho vyplýva inicializačné ofarbenie hrán incidentných s vrcholom 0.

Prehľadávanie grafov začíname vo vrchole 0, jednotlivé 1-faktory budeme zapisovať ako postupnosť hrán $f = (e_1,...,e_i)$ miesto $f = ((0, s),...,(t, u))$, pokiaľ nebude uvedený inak.
K faktorizacii F budeme pristupovať ako k ofarbeniu hrán grafu G. Ľubovoľná F jednoznačne určuje príslušné ofarbenie. V prípade parciálnej faktorizácie, hrany z G ktoré nepokrýva nazveme neofarbené.

4.2.1 Generovanie metódou faktor po faktore

Algoritmus 1 (obr. 4.2.1.1) generuje parciálne 1-faktorizácie $F = (f_1,..., f_i,..)$ faktor po faktore, pričom strom výpočtu prehľadáva metódou do hĺbky - deep first search, ďalej len DFS. Vstupom procedúry najdi-faktorizaciu je faktorizácia F_i rádu i. Ak je táto úplná (1), predstavuje výsledné riešenie. Ak je len parciálna (2), algoritmus generuje všetky možné jej rozšírenia, t.j. i+1-vé 1-faktory pomocou funkcie najdi_prvu_farbu (obr. 4.2.1.2) (3). Ako náhle algoritmus vygeneruje ľubovoľný úplný faktor f_j vyhovujúci ofarbeniu danom F_i, a prípadnom vykonaní heuristík na požadované vlastnosti (4), zavolá sa procedúra najdi_faktorizaciu pre ďalší rád 1-faktorizácie rozšírenú o 1-faktor f_j $F_{i+1} = (f_1, ..., f_i, f_j)$ (5). Výpočet algoritmu prejde celým grafom a vygeneruje všetky možné faktorizácie vzhľadom na vstupné podmienky a pripadne heuristiky. Metódou DFS sa algoritmus najrýchlejšie dopracuje k prvému riešeniu, nakoľko je procedúra najdi_faktorizaciu pre rád i+1 volaná hneď pre prvé možné rozšírenie F_i.

```
procedure najdi_faktorizaciu(Fi):
(1): if Fi je úplná then
        Return Fi
(2):  else
(3):    for pre všetky vhodné f vygenerované najdi_prvu_farbu(Fi,0) vzhľadom na Fi
(4):            for pre všetky vhodné f vzhľadom na heuristiky
(5):                    najdi_faktorizaciu( Fi ∪ f)
                end for
        end for
 end if
end procedure
```

Obr. 4.2.1.1: Algoritmus 1

Výpočet algoritmu 1 je možné paralelizovať pre jednotlivé faktorizácie F_i rádu i. Ak bola v procedúre najdi_faktorizaciu vykonaná heuristika zaručujúca ich vzájomný izomorfizmus (4), pred jej ďalším volaním (5), pre vetvy následného paralelného výpočtu

platí, že faktorizácie vygenerované jednotlivými navzájom rôznymi vetvami výpočtu nemôžu byť izomorfné, inak by totiž museli byt navzájom izomorfné aj parciálne faktorizácie F_i , čo je v rozpore s predpokladom. Alternatívne, pri neuvažovaní heuristiky pre vylúčenie izomorfných parciálnych faktorizácií počas výpočtu (4) je nutné vykonanie testu na izomorfizmus na konečných výsledkoch. Táto možnosť ale vyžaduje ich globálne porovnanie.

```
procedure najdi_prvu_farbu(F_i, f_j):
(1): if f_j je úplný then
         Return f_j
(2): else
(3):     for pre všetky ďalšie možné hrany e vzhľadom na f_j a F_i
(4):         for pre všetky hrany vyhovujúce heuristikám
(5):             najdi_prvu_farbu(F_i, f_j ∪ e)
         end for
     end for
  end if
end procedure
```

Obr. 4.2.1.2: Procedura najdi_prvu_farbu

Spomenutá funkcia najdi_prvu_farbu (obr. 4.2.1.2) funguje nasledovne: jej vstupom je 1-faktor f_j = (e_1,...,e_k). Ak je tento úplný (1), vráti ho ako vyhovujúce rozšírenie pre F_i. Ak je parciálny (2), najdi_prvu_farbu postupne prehľadá a vyhodnotí všetky hrany e vhodné pre rozšírenie f_j (3), vzhľadom na ofarbenie hrán dané parciálnou F_i , či prípadnou heuristikou pre parciálne 1-faktory (4). Funkcia potom zavolá samú seba pre všetky vyhovujúce hrany metódou DFS (5). Na počiatku je jej vstupom 1-faktor obsahujúci len i+1-vú hranu vrcholu 0.

Prehľadávané grafy Q_n a S_n sú hranovo a vrcholovo symetrické, na počiatku volania funkcie najdi_prvu_farbu sme hranu e_1 prehľadávali len smerom cez jej najmenšieho suseda. Výpočet sa nám značne skrátil, pričom sme žiadne riešenie nevylúčili, nakoľko je funkcia konštruovaná spôsobom prehľadávajúcim všetky možnosti, čím sa dopracuje ku každému možnému faktoru f_j začínajúcim hranou e_1.

Algoritmus 2 (obr. 4.2.1.3) má podobnú štruktúru ako algoritmus 1. Na rozdiel od neho ale prehľadáva graf do šírky - breadth first search , ďalej BFS. Pre svoj vstup, ktorým je parciálna faktorizácia F_i najprv v jednom kroku vygeneruje pomocou najdi_prvu_farbu všetky možné 1-faktory i+1-vé ho rádu f_j (3), a až po ich vyhodnotení, či prípadnom použití heuristických metód (4) sa procedúra najdi_faktorizaciu zavolá pre všetky kombinácie parciálnej F_i rádu i a vyhovujúcich i+1-vých 1-faktorov f_j (6). Výhodu využitia metódy BFS si ukážeme pri vyhodnocovaní experimentálnych výsledkov v nasledujúcej kapitole.

```
procedure najdi_faktorizaciu(F_i):
(1):  if F_i je úplná then
          Return F_i
(2):  else
(3):      pomocou najdi_prvu_farbu(F_i, 0) vygeneruj všetky možné f vzhľadom na F_i
(4):      vylúč f ktoré nevyhovujú heuristikám
(5):      for pre všetky zostávajúce f
(6):              najdi_faktorizaciu( F_i ∪ f)
          end for
 end if
end procedure
```

Obr. 4.2.1.3: Algoritmus 2

Paralelizácia výpočtu je rovnako ako v prvom prípade možná pre jednotlivé rozšírenia parciálnej F_i, za rovnakých podmienok ako pri algoritme 1. Funkcia najdi_prvu_farbu je totožná s funkciou z algoritmu 1, a rovnaká je aj myšlienka generovania faktorov pre parciálny faktor f_j obsahujúci len počiatočnú hranu e_1.

4.2.2 Generovanie metódou vrchol po vrchole

V nasledujúcich algoritmoch budeme pristupovať ku generovaniu 1-faktorizácií iným spôsobom, a to tak, že budeme prehľadávať graf postupne vrchol po vrchole, a ofarbovať im prislúchajúce hrany, následkom čoho bude F_i parciálna, až kým algoritmus nenavštívi posledný vrchol grafu G. Ofarbenie z ľubovoľného vrcholu v budeme reprezentovať

permutáciou farieb $c_1c_2..c_s$, kde farba c na j-tej pozícii pre $1 \leq j \leq s$ znamená ofarbenie hrany prislúchajúcej vrcholu v a jeho j-teho suseda farbou c. Pre Q_n je s rovné n, v prípade S_n je $s = n-1$.

```
procedure najdi_faktorizaciu(F_i ,u):
(1): if F_i je úplná then
        Return F_i
(2):  else
(3):    skoč do ďalšieho vrchola v v lexikografickom poradí po u
(4):    for pre všetky permutácie vygenerované pomocou ofarbi_vrchol(F_i ,v)
(5):            for pre všetky permutácie vyhovujúce heuristikám
(6):                    najdi_faktorizaciu(F_i ∪ permutácia, v)
                end for
        end for
 end if
end procedure
```

Obr. 4.2.2.1: Algoritmus 3

Algoritmus 3 (obr. 4.2.2.1) pracuje nasledovne: Ak je procedúra najdi_faktorizaciu zavolaná pre úplnú 1-faktorizáciou, vráti túto ako hľadané riešenie (1). V prípade nesplnenia podmienky úplnosti (2), algoritmus prejde do ďalšieho vrcholu grafu v v lexikografickom poradí (3). V ňom sa zavolá funkcia ofarbi_vrchol (obr. 4.2.2.2) generujúca všetky možné ofarbenia hrán incidentných s v, vzhľadom na predchádzajúce ofarbenie susedov v dané F_i (4), reprezentované permutáciou. Po vygenerovaní ľubovoľnej permutácie, a po prípadnom vyhodnotení uvažovaných heuristík (5), sa procedúra zavolá pre F_i rozšírenú o príslušnú permutáciu (6). Algoritmus tak prehľadáva graf G metódou do hĺbky, DFS. Nakoľko vrcholmi grafu G prechádzame v lexikografickom poradí, procedúra ofarbí všetky hrany incidentné s v, a strom výpočtu sa rozvetvuje pre všetky možnosti ofarbenia hrán prislúchajúcich v. Je zrejmé, že procedúra nachádzajúca sa v poslednom vrchole v grafu G už nemá čo ofarbiť, nakoľko každá jemu prislúchajúca hrana e bola v priebehu výpočtu ofarbená vo chvíli, keď sa tento nachádzal v lexikograficky menších vrcholoch prislúchajúcich hrane e. Výsledné ofarbenie, resp. 1-faktorizácia F_i je úplná a vhodná, vďaka počiatočným podmienkam algoritmov.

```
procedure ofarbi_vrchol (F_i ,v):
(1): for pre všetky permutácie reprezentujúce ofarbenia v a jemu prislúchajúcich hrán
(2):     vylúč tie, ktore nesúhlasia s F_i
     end for
(3): return zostávajúce permutácie
end procedure
```

Obr. 4.2.2.2: Procedura ofarbi_vrchol

Funkcia ofarbi_vrchol (obr. 4.2.2.2) vygeneruje všetky permutácie *k* prvkov, kde *k* je počet farieb faktorizácie F_i v grafe *G*, a následne vylúči tie, ktoré nemajú na *i*-tej pozícií farbu reprezentujúcu ofarbenie hrany *e* medzi vrcholmi *v* a jeho *i*-tym susedom. Uvažované ofarbenie hrany *e* je určené F_i, inak ju považujeme za neofarbenú.

Algoritmus je možné distribuovať pre jednotlivé faktorizácie F_i rozšírené permutáciami reprezentujúcimi ofarbenia hrán incidentných s vrcholom *v*. V prípade, ak na permutáciách nebola vykonaná heuristika vylučujúca izomorfné parciálne riešenia (5), je nutné konečné výsledky navzájom ešte porovnať.

```
procedure najdi_faktorizaciu(F_i ,u):
(1): if F_i je úplná then
         Return F_i
(2): else
(3):     skoč do ďalšieho vrchola v v lexikografickom poradí po u
(4):     vygeneruj všetky možné permutácie pomocou ofarbi_vrchol(F_i ,v)
(5):     vylúč permutacie vzhľadom na heuristiky
(6):     for pre všetky zostávajúce permutácie
(7):             najdi_faktorizaciu(F_i ∪ permutácia, v)
         end for
 end if
end procedure
```

Obr. 4.2.2.3: Algoritmus 4

Algoritmus 4 (obr. 4.2.2.3) je modifikáciou algoritmu 3, na rozdiel od neho však využíva metódu BFS pri vyhodnocovaní možných permutácii prislúchajúcich vrcholu *v*. Procedúra najdi_faktorizaciu v ňom zavolá funkciu ofarbi_vrchol (4), a až po vygenerovaní všetkých možných permutácii vrcholu *v* vzhľadom k F_i v jednom kroku výpočtu, a prípadnom použití heuristických metód (5), zavolá samu seba pre zostávajúce možné rozšírenia F_i príslušnými permutáciami. Na počiatku výpočtu algoritmov 3 a 4 sa procedúra najdi_faktorizaciu zavolá len s ofarbenými hranami susediacimi s vrcholom 0. Rovnako ako pri algoritme 3, prehľadávame graf *G* v lexikografickom poradí. Úplnosť a vhodnosť vygenerovanej faktorizácie F_i v poslednom vrchole *G*, ako aj podmienky pre prípadnú paralelizáciu sú rovnaké ako v predchádzajúcom prípade.

Všetky štyri predstavené algoritmy volajú backtrackom procedúru najdi_faktorizaciu so vstupom v podobe parciálnej faktorizácie F_i , pripadne aj s vrcholom *v* v ktorom sa výpočet nachádza. Pri jeho paralelizácii tak nevzniká potreba ukladania dát čí parciálnych výsledkov. Tieto sa môžu zapisovať lokálne pre jednotlivé procesy, či procesory. Potreba globálnej štruktúry nastáva len v prípade testovania faktorizácií vzhľadom na izomorfizmus až na konečných výsledkoch, kde je tieto nutne ešte navzájom porovnať.

4.3 Heuristiky a metódy optimalizácie

Ako sme spomenuli v predchádzajúcich kapitolách, otázka izomorfizmu grafov a 1-faktorizacii je NP-ťažka. Jedným zo spôsobov ako urýchliť náročný výpočet problémov z tejto triedy je použitie heuristík. Heuristika, grécky *εύρίσκω,* je preložiteľná ako nachádzanie, či objavovanie. Často ide o metódy nahrádzajúce neexistujúci algoritmus, prípadne nezaručujúce objavenie správneho riešenia, čo ani pri zrýchlení výpočtu nie je veľmi žiaduce. My sa budeme zaoberať heuristikami, ktoré pri generovaní žiadne riešenie nevylúčia, pričom ich

vhodnou impelentáciou sa náš výpočet skráti.

4.3.1 Uvažované heuristiky a ich popis

Uvažované heuristiky:

- vhodné 1-faktorizácie
- test identity
- test (semi)perfektnosti
- d-tabuľka
- paralelizmus
- test izomorfizmu

Vhodné 1-faktorizácie - stanovením počiatočnej podmienky ofarbenia vrcholu 0 sme predišli generovaniu mnohých izomorfných riešení, bližší popis myšlienky sa nachádza v predchádzajúcej časti.

Test identity - elementárna heuristika vylučujúca duplicitné riešenia, v našom prípade pôjde o faktory generovane funkciou najdi_prvu_farbu.

D-tabuľka – invariant 1-faktorizácie $F = (f_1, ..., f_t)$. Zjednotením ľubovoľných jej dvoch 1-faktorov vznikne graf tvorený párnymi cyklami, dĺžky aspoň 4, v prípade star grafov sú dĺžky cyklov aspoň 6. D-vektor faktora f_i tvoria dĺžky jednotlivých cyklov tvorených kombináciou f_i s ostatnými 1-faktormi F. D-tabuľka 1-faktorizácie pozostáva z príslušných d-vektorov jej faktorov. Výhodou d-tabuľky ako invariantu je veľmi ľahká vypočítateľnosť v porovnaní s drahšími heuristikami, napr. testom na izomorfizmus. Detaily sa dajú nájsť v[3]. Nakoľko nás primárne zaujímajú semi-perfektné a perfektné 1-faktorizácie, prvý, respektíve každý riadok d-tabuľky musí obsahovať iba cykly dĺžky Hamiltonovskej kružnice. Všimnime si, že d-tabuľka nedokáže rozhodnúť o izomorfizme perfektných 1-faktorizácii, a dokonca ani o semi-perfektných, ak tieto majú rovnaké kombinácie d-vektorov. Navzájom rôzne d-tabuľky

implikujú izomorfizmus faktorizácií im prislúchajúcim.

Test (semi) perfektnosti – z pohľadu spomenutých d-tabuliek, d-vektor prislúchajúci prvému faktoru f_1 , resp. všetkým faktorom f_i musí obsahovať len čísla rovnajúce sa dĺžke Hamiltonovskej kružnice daného grafu. Test je však možné využiť aj na parciálnych faktorizáciach, prípadne faktoroch. Pri hľadaní semi-perfektných $F = (f_1,..,f_t)$ platí, že ak f_i pre $1 < i \leq t$ v kombinácii s f_1 vytvorí cyklus kratší ako je Hamilonovská kružnica, nemôže isť o semi-perfektnú faktorizáciu, a faktor f_i je možné z ďalšieho výpočtu vylúčiť. V prípade perfektných faktorizacii podmienka ukončenia výpočtu platí pre kombináciu ľubovoľných dvoch jej faktorov.

Paralelizmus – výpočtové nároky algoritmov na grafoch s rastúcim počtom vrcholov rastú závratne rýchlo. Distribuovaný výpočet sa mnohokrát stáva nevyhnutnosťou. Snažili sme sa výpočet paralelizovať vo vhodných uzloch s ohľadom na konečnú efektívnosť, či nutnosť prístupu do globálnych štruktúr.

4.3.2 Test izomorfizmu a program nauty

Posledná uvažovaná heuristika nám umožní generovať navzájom neizomorfné riešenia našich algoritmov. O izomorfizme faktorizacii sú sčasti schopné rozhodnúť aj d-tabuľky, ich možnosti ale končia v bode, kde majú 2 faktorizácie v nich rovnaké kombinácie d-vektorom. My preto v našich algoritmoch využívame jeden z najrýchlejších programov vhodných na rozhodovanie o izomorfizme grafov, nauty. Základné definície a pojmy potrebné na priblíženie jeho výpočtu sú uvedené v úvode. Podrobnejšie informácie je možné nájsť v [16]. Matematické základy algoritmu nautyho sa nachádzajú v [15]. Na rozhodovanie o izomorfizme grafov využíva nasledujúce tvrdenie [15]:

Veta 4.3.2.1: *Nech G a H sú ofarbené grafy s rovnakým počtom vrcholov v každej farbe. Potom $C(G,\pi_1)=C(G,\pi_2)$, práve vtedy, keď $G^\gamma = H$ pre nejakú permutáciu γ zachovávajúcu ofarbenie. (π_1 a π_2 sú partície ich vrcholov na jednotlivé farby, pričom poradie farieb je*

rovnaké)

Nauty dokáže rozhodnúť o izomorfizme grafov G a H na základe porovnania im prislúchajúcich kanonických ekvivalentov. Veta 4.3.2.1 predpokladá permutáciu zachovávajúcu poradie ofarbení. Na poradí vrcholov grafov G a H v jednotlivých triedach partícii π_1 a π_2 nezáleží. Podľa definície izomorfizmu faktorizacii, sú tieto izomorfné pri určitej permutácii svojich faktorov, dôsledkom čoho bolo v našich výpočtoch pri teste na izomorfizmus medzi faktorizáciami F a H často krát nevyhnutné vhodne permutovať ofarbenia v F, vypočítať jej kanonicky graf $C(F)$ a následne porovnať s kanonickým grafom $C(H)$. Aby sme sa podľa možnosti vyhli zbytočnému volaniu programu nauty, používali sme funkciu z jeho knižnice *updatecan*. Je určená na prečíslovanie vrcholov grafu G tak, aby sme z neho získali kanonicky graf $C(G)$. Funkcia *updatecan* na kanonické prečíslovanie potrebuje len "kanonickú" permutáciu vrcholov G. Nakoľko sa ňou len prečíslujú vrcholy grafu G, ide o omnoho lacnejší výpočet ako v prípade volania nautyho, ktorý má exponencionálnu zložitosť. Pre každé vyhovujúce riešenie v určitom uzle výpočtového stromu sme si do lokálnej dátovej štruktúry uložili príslušné "kanonické" permutácie. Pri prvom nájdenom riešení, ktoré nemáme s čím porovnať, sme si uložili všetky jeho "kanonické" permutácie. V prípade ďalšieho potencionálneho riešenia R sme potom funkciou *updatecan* získali kanonické grafy k už uloženým riešeniam, a tieto navzájom porovnali s kanonickým grafom $C(R)$. R aj naše uložené riešenia museli mať rovnaký počet vrcholov v jednotlivých farbách, resp. faktoroch. V prípade, že sa $C(R)$ nezhodoval ani s jedným uvažovaným kanonickým grafom, a teda bol R izomorfný vzhľadom na doterajšie riešenia, bolo nutné si uložiť "kanonické" permutácie prislúchajúce R.

V závislosti od použitých algoritmov tak bolo potrebné vhodne permutovať faktory resp. ofarbenia. Pri algoritmoch 1 a 2 pre nejakú parciálnu faktorizáciu F_i radu i šlo o i faktorov, generovaných postupne vzhľadom na rád i, no pre algoritmy 3 a 4 bolo nutné spermutovať všetky farebne partície, nakoľko sa v priebehu ich výpočtov v každom vrchole grafu G zafarbovali hrany všetkými farbami naraz. Pripomíname, že v prípade Q_n je počet farebných partícii n, v S_n n-1. Neofarbené hrany sme samozrejme nepovažovali za farbu. Dôsledky vyplývajúce z nutnosti permutovať všetky relevantne farby kvôli testu

izomorfizmu, si ukážeme pri vyhodnocovaní experimentálnych výsledkov v nasledujúcej kapitole.

Nauty je pre nekomerčné použitie voľne šíriteľný program. Ide o knižnicu procedúr napísaných v jazyku *C*. Jeho vstupom je vrcholovo ofarbený graf (G,π), a jeho možnými výstupmi sú kanonický graf $C(G,\pi)$ určený kanonickou mapujúcou funkciou, permutácia γ vrcholov G taká, že $G^{\gamma} = C(G,\pi)$, a množina generátorov automorfnej grupy pre G.

Samotný nauty prechádza stromom svojho výpočtu klasickým backtrackom (obr 4.2.1). Koreňom je vstup (G,π) a partícia π' vhodne zvolená nautym vzhľadom na π. Ak je π' diskrétna, stabilizačná (automorfná) grupa pre G je triviálna, a kanonický graf $C(G,\pi)$ je definovaný premenovaním vrcholov V poradí v akom sa nachádzajú v π'. Ak π' nie je diskrétna, obsahuje aspoň jednu nejednoprvkovú podmnožinu M vrcholov z G. Nauty rozvetví strom výpočtu pre všetky vhodné kombinácie vytvorené z π' nahradením M v π' množinami $\{v\}$ a $\{M\}$-$\{v\}$ v poradí, kde v sú vrcholy nachádzajúce sa v M. Každý uzol výpočtového stromu nautyho s diskrétnou partíciou π' predstavuje určité očíslovanie vrcholov. Ak sú dva uzly s diskrétnymi partíciami zhodne očíslované, ich príslušné grafy sú izomorfné. Kanonická mapovacia funkcia, netriviálne vypočítaná nautym, tak zodpovedá nejakému z týchto očíslovaní, presnejšie kanonickému. Sofistikovaný výpočet nautyho následne orezáva strom výpočtu pre vhodne zvolené vetvy, a je relatívne veľmi efektívny pre veľký počet rôznych tried grafov. Pre 1840 rôznych faktorizácií grafu Q_4 bol napríklad schopný rozhodnúť o ich vzájomnom (ne)izomorfizme za ~1.5 sekundy.

Nakoľko nauty pracuje s vrcholovo ofarbenými grafmi, a nami uvažované faktorizácie sú partície hrán, bolo nutne prekonvertovať nami grafy Q_n a S_n na ich hranové ekvivalenty, a až následne bolo možné pre ne zavolať nautyho.

Naše algoritmy, na rozdiel od kanonických zástupcov používaných metódou popísanou v kapitole 3, si pri volaní testu na izomorfizmus v určitom bode výpočtu uložia prvého vygenerovaného zástupcu triedy izomorfizmu, ako aj jeho “kanonické” permutácie, určujúce kanonické grafy pre rôzne permutácie jeho ofarbení resp. faktorov faktorizácie F.

Korektnosť algoritmov: spôsoby prehľadávania grafov našich algoritmov popísané predchádzajúcej časti práce, zaručujú prehľadanie každého vrchola *v* a hranou *e* grafu *G*. V uzloch stromu výpočtu sa priebežne nachádzali všetky možné parciálne riešenia, pokiaľ neboli zamietnuté heuristickými metódami. Správnosť generovania neizomorfných faktorizácií sa zakladá na vete 4.3.2.1, ktorú využíva program nauty, a vhodnej permutácií jednotlivých partícii, t.j. farebných tried vrcholov *G*. Korektnosť vzhľadom na ostatné heuristiky je evidentná z ich popisov.

Kapitola 5

Experimentálne výsledky

V tejto časti prace sa budeme zaoberať postupným vkladaním rôznych kombinácii heuristických metód do príslušných častí algoritmu a následne vyhodnocovať ich dosah na výpočet. Aby sme dosiahli objektívne namerane dáta uvedené v tabuľkách, výpočty boli v priemere 50x zopakované. V prehľadávaných hyperkockách a star grafoch sme žiadnu perfektnú 1-faktorizaciu nenašli, v tabuľkách a ich popisoch sa preto budú nachádzať len experimentálne výsledky pre test na semi-perfektnosť. Na prípady, kedy sme brali do úvahy aj test na perfektnosť, upozornime zvlášť.

Test identity budeme používať len pri algoritmoch 1 a 2. Funkcia najdi_prvu_farbu ktorá je volaná v bode (3) procedúry najdi_faktorizaciu (obr 4.2.1.1 a 4.2.1.3) pri úplnom prehľadávaní grafu *G* generuje aj faktory navzájom totožné. Budeme ju implementovať pred ľubovoľnou inou optimalizačnou metódou. Jediné miesto v ktorom mu bude predchádzať iná heuristika je vo funkcii najdi_prvu_farbu (4), nakoľko sa faktory dajú otestovať na zhodu len pri ich úplnosti.

Ku každému algoritmu si uvedieme dve tabuľky. V prvej budú zhrnuté dosiahnuté priemerné časy výpočtov pre rôzne modifikácie algoritmov, v druhej sa budú nachádzať jednotlivé heuristické metódy v nich uvažované. Budeme pre ne používať nasledujúce označenia: S – test na semi-perfektnosť; N – test na izomorfizmus (program nauty); D – pre d-tabulky; a I - pre počet identických faktorov vygenerovaných počas celého výpočtu. V prípade heuristík S, D a N budeme uvádzať dva údaje. Prvý bude označovať počet ich

volaní počas celého výpočtu a druhý počet potencionálnych riešení, ktoré v priebehu vylúčili. V algoritmoch boli použité v poradí, v akom sú uvedené tabuľkách. V prípade d-tabuliek bude druhý údaj označovať počet takých volaní a zároveň faktorizácií, pre ktoré už nebolo nutne zavolať test na izomorfizmus, ktorému tato heuristika zakaždým predchádza. V kombinácii testu na semi-perfektnosť nasledovaného d-tabuľkami, bolo postačujúce príslušné riadky d-tabuliek vyhodnocovať len pre faktory f_i a f_j, kde $i \neq j \neq 1$, nakoľko d-vektor prislúchajúci faktoru f_1 obsahuje len prvky rovné dĺžke Hamiltonovskej kružnice v G. V nasledujúcich riadkoch pod *priebežnou* heuristikou mame na mysli jej volanie v priebehu výpočtu, na rozdiel od volania metódy až na konečných výsledkoch. Pre algoritmy 1 a 2 tým myslíme volanie v procedúre najdi_faktorizaciu (4), pre algoritmy 3 a 4 v procedúre najdi_faktorizaciu (5). Heuristika volaná vo funkcii najdi_prvu_farbu (4) ktorú používajú prvé dva algoritmy môže byt volaná len priebežne.

5.1 Výsledky pre generovanie metódou faktor po faktore

Tabuľkové označenie kombinácii heuristík pre algoritmy 1 a 2 (obr 4.2.1.1 a 4.2.1.3)

A1 - kompletné prehľadanie grafu, generovanie len nezhodných faktorizacii
A1.1 – priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (4)
A1.2 - test izomorfizmu pre konečné výsledky
A1.3 - d-tabuľky vypočítané na konečných výsledkoch a následný test izomorfizmu
A2 - test na semi-perfektnosť vykonaný vo funkcii najdi_prvu_farbu (4)
A3 – priebežný test na semi-perfektnosť vykonaný v procedúre najdi_faktorizaciu (4)
A4 - test na semi-perfektnosť pre konečné výsledky
A5 - najlepšia heuristika z A2-A4 a následný test izomorfizmu na konečných výsledkoch
A6 - A2 + priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (4)
A7 - A3 + priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (4)
A8 - priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (4) + A4

A9 - najlepšia heuristika z A2-A4 + d-tabuľky na konečných výsledkoch nasledované testom na izomorfizmus.

	Q_3	Q_4	S_4
A1	0.000021	1.374312	0.711306
A1.1	0.000135	9.075259	1.403243
A1.2	0.000081	2.624630	0.778289
A1.3	0.000088	2.729105	0.780835
A2	0.000026	0.613199	0.784601
A3	0.000023	0.660783	0.725943
A4	0.000022	1.395047	0.728365
A5	0.000023	0.676394	0.725943
A6	0.000087	1.156336	1.021723
A7	0.000080	1.179201	1.008063
A8	0.000135	9.083241	1.413967
A9	0.000023	0.638524	0.721067

Tab. 5.1.1 Časové hodnoty pre algorimtus 1

Údaje namerané pre algoritmus 1 sa nachádzajú v tabuľkách 5.1.1 a 5.1.2. Na prvý pohľad si všimneme veľmi zlé výsledky pre priebežný test na izomorfizmus, dosahujúce násobky času výpočtu v prípade jeho volania na konečných výsledkov (A1.1, A8). Rovnako nárast počtu volaní výpočtovo drahého nautyho v týchto metodikách bez predchádzania inou heuristikou je nežiaduci. Metodiky, v ktorých mu predchádzame testom na semi-perfektnosť (A6, A7) majú výsledky lepšie namerané hodnoty, avšak stále podstatne horšie v porovnaní s volaním nautyho na konci (A1.2 A1.3), či s kombináciou rátajúcou d-tabuľky (A9). Tieto rozdiely sú dôsledkom prehľadávania grafu spôsobom DFS, vďaka ktorej sa v procedúre najdi_faktorizaciu výpočtovo drahšie heuristiky volajú aj pre parciálne riešenia, ktoré by v neskoršom výpočte boli zrušené lacnejšou (A8). Pri porovnaní s (A7) si môžme všimnúť dôsledok zmeny v poradí volania nautyho a omnoho lacnejšieho testu na semi-perfektnosť po každom vygenerovanom faktore.

Vylučovanie potencionálnych 1-faktorov pri posudzovaní ich ďalších možných hrán sa ukázalo výhodne už pri Q_4. V prípade väčších grafov sa preto dá predpokladať ešte väčšia efektívnosť. Za pozornosť tiež stojí tempo nárastu výpočtových nárokov z Q_3 na Q_4, ktoré sú v najlepšom prípade ~20 000 krát dlhšie, pričom sa počet rôznych faktorizácii zvýšil zhruba 400-krát (A6, A7, A9).

	Q_3	Q_4	S_4
A1	I 7	I 462605	I 15513
A1.1	N 8/4	N 35254/35167	N 5753/5263
A1.2	N 4/2	N 1840/1805	N 84/75
A1.3	D 4/1 N 3/2	D 1840/26 N 1814/1805	D 84/7 N 77/67
A2	S 13/5	S 532688/161908	S 186632/44753
A3	S 16/7	S 223745/162823	S 9791/9412
A4	S 4/3	S 1840/1808	S 84/84
A5	S 4/3 N 1/0	S 532688/161908 N 32/31	S 9791/ 9779 D 0/0 N 0/0
A6	N 5/2 S 7/3	S 50281/11964 N 12680/12660	S 41052/9760 N 262/142
A7	N 5/2 S 13/4	S 32644/16578 N 12680/12660	S 2494/2166 N 262/142
A8	N 8/4 S 2/1	N 35254/35167 S 35/34	N 5753/5736 S 9/9
A9	D 1/0 N 1/0	D 32/1 N 31/1	D 0/0 N 0/0

Tab. 5.1.2: Počty volaní heuristík pre algoritmus 1

Ani v prípade zakomponovania d-tabuliek v snahe ušetriť volanie nautyho sa nám želaný efekt nepodarilo dosiahnuť, nakoľko sú takmer všetky vhodné a úplné faktorizácie našich grafov navzájom izomorfné (z 1840 je len 35 neizomorfných), a po vypočítaní pre

väčšinu z nich rovnakých d-tabuliek, je nutné ich ďalšie otestovanie nautym.

Metodiky vhodné pre nezávislé distribuované výpočty bez nutnosti porovnávania konečných výsledkov sa ukázali ako výpočtovo neefektívne (A1.1, A6-A8). Ako lepšia alternatíva sa tak pre DFS javí výpočet generujúci izomorfne riešenia, a dodatočne centrálne vyhodnotenie konečných výsledkov (A9). Už pri samotnom generovaní výsledkov (A1) vzniká veľký počet identických faktorov, v priebehu výpočtu na Q_4 až 462 605. Porovnaním rôznych implementácii semi-perfekt testov, ako dôsledok neefektívneho generovania, dosahuje jeho volanie na konci (A4) najhoršie výsledky, hoci ho voláme len 1840 krát oproti 223 745 priebežným volaniam (A3) či dokonca 532 688 (A2). Posledne dve spomínané metodiky dosahujú takmer rovnaké údaje pri vyše dvojnásobnom počte volaní (A3), čím sa prakticky demonštruje výpočtovo lacná cena priebežného volania. Pri kombinácii (A5) sme sa v Q_4 rozhodli pre semi-perfekt test už pri každej hrane faktora volaného vo funkcii najdi_prvu_farbu (A2). Hoci je volaný často, je rýchly, a orezáva strom výpočtu čo najskôr, čo je výhodné najmä pri vyšších dimenziách. V prípade S_4 (A5) nie je nauty volaný ani raz, nakoľko všetky faktorizácie vylúči test na semi-perfektnosť.

Algoritmus 1 spustený v Q_5 sa dopracoval k prvému semi-perfektnému riešeniu už za pár minút, z nameraných údajov pre menšie hyperkocky je ale zrejmé, že prehľadanie celého grafu by mu trvalo týždne. V prípade S_5 sa nám nepodarilo dopracovať k semi-perfektnému riešeniu ani po vyše desiatich dňoch distribuovaného výpočtu využívajúceho metodiku (A6), pričom stihol prehľadať len 2/10 grafu.

Pre vyššie dimenzie sa algoritmus 1 s metódou DFS javí ako celkovo značne neefektívny. Dobré výsledky dosahuje len pri nachádzaní prvých riešení danej vetvy stromu výpočtu. Z uvažovaných kombinácii by sme optimálnou mohli nazvať priebežný semi-perfekt test s nautym na konečných výsledkoch (A9).

	Q_3	Q_4	S_4
A1	0.000017	1.274881	0.686318
A1.1	0.000077	0.123734	0.300457
A1.2	0.000055	2.598596	0.727538
A1.3	0.000063	2.658399	0.722934
A2	0.000005	0.023926	0.755250
A3	0.000005	0.021891	0.680068
A4	0.000008	1.314548	0.696976
A5	0.000005	0.021809	0.680068
A6	0.000022	0.022035	0.328274
A7	0.000018	0.022226	0.299810
A8	0.000057	0.129266	0.307131
A9	0.000010	0.023312	0.680068

Tab. 5.1.3 Časové hodnoty pre algorimtus 2

Výsledné časy výpočtov a počty volaní jednotlivých heuristík pre algoritmus 2, prechádzajúci grafom metódou BFS, sa nachádzajú v tabuľkách 5.1.3 a 5.1.4 Tieto sú vo všeobecnosti lepšie ako v predchádzajúcom prípade, hlavne čo sa týka metodík s priebežným testom na izomorfizmus (A1.1,A6-A8). Je to dané spôsobom prehľadávania grafu, kde sa všetky parciálne výsledky danej F_i rádu i+1 najskôr vyhodnotia v jednom kroku, a až následne je procedúra najdi_faktorizaciu volaná pre zostávajúce. Pripomíname, že jednotlivé heuristiky sú primárne vyhodnocované v poradí podľa ich výpočtovej ceny. Funkcia najdi_prvu_farbu síce nachádza rovnaké množstvo čiastkových identických riešení ako pri algoritme 1 (A1), no ich vylučovanie je omnoho efektívnejšie tým, že test na zhodu pre celý uzol výpočtu predchádza ostatným heuristikám, okrem semi-perfekt testu v priebehu generovania faktorov (A2). Samotné generovanie nezhodných výsledkov (A1) je v algoritme 2 nepatrne rýchlejšie, pri rovnakom počte nájdených identických výsledkov.

Tak ako v prvom algoritme, test na semi-perfektnosť (A2,A3) dosiahol navzájom podobné časové výsledky, počet jeho volaní sa ale značne znížil v prípade testovania až na úplných faktoroch v procedúre najdi_faktorizaciu (A3). V S_4 ich bolo až tisíc krát menej ako pri totožnej heuristike použitej vo funkcii najdi_prvu_farbu, v Q_4 500-násobne menej. V porovnaní s algoritmom 1 sa k výsledkom v Q_4 metódou BFS algoritmus 2 dopracujeme 20-

nasobne rýchlejšie. Volanie nautyho priebežne po faktoroch (A1.1) v porovnaní s testom na konečných výsledkoch (A1.2), ukázalo významné zlepšenie času výpočtu už pri Q_4 a S_4 aj bez použitia ďalšej heuristiky.

	Q_3	Q_4	S_4
A1	I 7	I 462605	I 15513
A1.1	N 6/2	N 303/216	N 63/46
A1.2	N 4/2	N 1840/1805	N 84/75
A1.3	D 4/1 N 3/2	D 1840/26 N 1814/1805	D 84/7 N 77/67
A2	S 18/9	S 476461/154764	S 186690/44538
A3	S 16/7	S 1448/1044	S 130/84
A4	S 4/3	S 1840/1808	S 84/84
A5	N 1/0	S 1448/1044 N 32/21	S 130/84 N 0/0
A6	S 6/3 N 4/2	S 35756/11594 N 96/78	S 48062/10738 N 43/34
A7	N 4/2 S 9/4	S 201/104 N 96/78	S 64/21 N 43/34
A8	N 8/4 S 2/1	N 303/216 S 35/34	N 63/46 S 9/9
A9	D 1/0 N 1/0	D 32/1	D 0/0 N 0/0

Tab. 5.1.4: Počty volaní heuristík pre algoritmus 2

Čas dosiahnutý pre (A3) je takmer totožný s (A2), no pre väčšie grafy predpokladáme väčšiu úsporu času výpočtu pre heuristiku (A2), nakoľko je výpočtovo lacná a oreže strom výpočtu skôr ako ktorákoľvek iná uvažovaná metóda. V kombinácií s nautym na záverečných výsledkoch (A5), je v Q_4 dosiahnutý ešte lepší čas, no len minimálne, zatiaľ čo v S_4 je priebežné volanie nautyho 2x rýchlejšie ako v prípade konečných výsledkov. V Q_3 je

samozrejme vzhľadom na semi-perfekt test najlepšie volať nautyho až na záver (A5, A6, A9), keďže je v podstate nepotrebný. Existuje totiž len jedna vhodná semi-perfektná faktorizácia. Pripomeňme, že v prípade S4 neexistuje žiadna. Metodiky rátajúce d-tabuľky pred volaním konečného nautyho (A1.3, A9) poskytujú takmer totožný čas výpočtu ako tie, ktoré ich nevyužívajú (A5). Spôsobuje to nízky počet semi-perfektných faktorizacii, na ktorých sa ich výhoda neprejaví.

Ako optimálna sa ukázala kombinácia pribežného testu semi-perfektnosti s volaním nautyho, či už priebežne alebo až na záver (A5-A7, A9). Rôzne spôsoby volania testu izomorfizmu môžu vyhovovať v závislosti od želanej distribúcie výpočtu a následnej možnosti porovnávania konečných výsledkov. Vďaka metóde BFS sa algoritmus 2 spustený v Q_5 s priebežným testom semi-perfektnosti ako aj izomorfizmu (A6) ani za týždeň nedopracoval k žiadnemu výsledku. Pri kombinácii s volaním nautyho až na záver (A9), vyžadujúcou si záverečné porovnania výsledkov, takisto žiadnu faktorizáciu nevygeneroval. Modifikovaný a distribuovaný algoritmus 2 nám v Q_5 za približne 20 hodín vygeneroval 117 837 nazvájom rôznych 1-faktorizácií rádu 1, z nich bolo 339 neizomorfných.

5.2 Výsledky pre generovanie metódou vrchol po vrchole

Spôsob, akým sa prehľadávajú grafy pri algoritmoch 3 a .4 neumožňuje vygenerovať zhodné riešenia, a teda test na identitu nebudeme pri nasledujúcich algoritmoch brat do úvahy. Označenie kombinácii heuristík pre algoritmy 3 a 4 (obr 4.2.2.1 a 4.2.2.3).

B1 - kompletné prehľadanie grafu a všetky výsledky
B1.1 – priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (5)
B1.2 - test izomorfizmu pre konečné výsledky
B1.3 - d-tabuľky vypočítané na konečných výsledkoch a následný test izomorfizmu
B2 – priebežný test na semi-perfektnosť vykonaný v procedúre najdi_faktorizaciu (5)

B3 - test na semi-perfektnosť len na konečných výsledkoch

B4 - lepšia heuristika z B2-B3 následná testom izomorfizmu na konečných výsledkov

B5 - B2 + test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (5)

B6 - priebežný test izomorfizmu vykonaný v procedúre najdi_faktorizaciu (5) + B3

B7 - lepšia z B2-B3 + d-tabuľky na konečných výsledkoch nasledované testom na izomorfizmus

	Q_3	Q_4	S_4
B1	0.000018	0.021090	0.0492112
B1.1	0.000612	0.608378	8.012870
B1.2	0.000065	1.524644	0.055032
B1.3	0.000077	1.586347	0.066325
B2	0.000033	0.003315	0.117058
B3	0.000036	0.010883	0.047611
B4	0.000033	0.006571	0.480781
B5	0.000226	0.381163	3.385654
B6	0.000694	0.611503	8.013662
B7	0.000031	0.007320	0.047611

Tab. 5.2.1 Časové hodnoty pre algorimtus 3

V nasledujúcich riadkoch si popíšeme výsledky pre algoritmus 3 uvedené v tabuľkách 5.2.1 a 5.2.2 Namerané údaje sú vo všeobecnosti lepšie ako pre predchádzajúce algoritmy, okrem metodik s priebežným volaním nautyho (B1.1,B5,B6). Pre star grafy sú v týchto prípadoch časy výpočtov veľmi zlé. Samotné generovanie faktorizácii, vďaka spôsobu ofarbovania vrcholu po vrchole, ktorý v nich nepripúšťa identické zafarbenia, je rýchlejšie ako v prípade prvých algoritmov (A1,B1). V tabuľke 5.2.2 je následkom toho príslušný riadok ignorovaný. Hoci je samotné generovanie ofarbení efektívnejšie, veľmi zlé hodnoty zaznamenané pre test na izomorfizmus priebežne v každom vrchole grafu sú dôsledkom nutnosti permutácie všetkých partícii, resp. farieb faktorizácie. Počet premutácii je od prvého momentu výpočtu rovnaký ako na konci – na rozdiel od algoritmov 1 a 2, kde sa ďalšie farby

pridávali až pre ďalšie rady faktorizácie F_i.

V prípade volania nautyho na záverečných výsledkoch v porovnaní s priebežným v každom bode, sme horšie výsledky zaznamenali len v prípade uvažovania ďalšej heuristiky (B1.2). V tomto prípade bol zaznamenaný aj najväčší počet jeho volaní. Akonáhle sme test na izomorfizmus skombinovali s ľubovoľným semi-perfektným testom (B4,B7), boli namerané údaje len zlomky oproti priebežne volanému nautymu (B5,B6), kde aj počty jeho volania boli výrazne menšie.

	Q_3	Q_4	S_4
B1	-	-	-
B1.1	N 13/2	N 868/205	N 3246/1390
B1.2	N 4/2	N 1840/1805	N 84/75
B1.3	D 4/2 N 2/2	D 1840/26 N 1814/1805	D 84/7 N 77/67
B2	S 12/3	S 2077/618	S 29466/285
B3	S 4/3	S 1840/1808	S 84/84
B4	S 12/3 N 1/0	S 2077/618 N 32/31	S 84/84 N 0/0
B5	S 7/2 N 6/2	S 186/29 N 225/53	S 2547/23 N 2524/1127
B6	N 15/2 S 6/5	N 868/205 S 9/8	N 3246/1390 S 9/9
B7	S 12/3 D 1/0 N 1/0	S 2077/618 D 32/6 N 6/5	S 84/84 D 0/0 N 0/0

Tab. 5.2.2: Počty volaní heuristík pre algoritmi 3 a 4

Metóda umožňujúca paralelizmus bez potreby dodatočnej komunikácie medzi uzlami, t.j. test na izomorfizmus v každom vrchole grafu G sa javí ako značné neefektívna už pri malých dimenziách. Najlepšie výsledky dosahovala v kombinácii s priebežným testom na semi-perfektnost (B5), avšak aj tie sú ~100 násobne horšie ako v prípade volania nautyho až na konečných výsledkoch (B4, B7).

Nárast dĺžky výpočtu z Q_3 na Q_4 bol pre najlepšie metodiky (B4, B7) približne 200-násobný, čo je v porovnaní s predchádzajúcimi algoritmami veľmi dobrý výsledok. Použitie d-tabuliek nám žiadny výpočet výraznejšie nezrýchlilo, naopak ho minimálne predlžilo. Ako veľmi dobra heuristika sa opäť ukázal test na semi-perfektnosť, ktorý vylúčil čiastkové riešenie pri každom treťom volaní (B2, B5, B7). Algoritmus spustený v Q_5 sa dopracoval k prvému semi-perfektnému riešeniu pomocou vrcholového semi-perfekt testu (B2) v porovnateľne rovnakom čase ako v prípade algoritmu 1, a to vďaka spôsobu prehľadávania grafu do hĺbky DFS. Na rozdiel od neho však výpočet algoritmu 3 skončil za pár dni, pričom našiel stovky semi-perfektných faktorizácii, z ktorých ale mnohé boli navzájom izomorfné.

Ideálnou heuristikou pre generovanie neizomorfných semi-perfektných faktorizacii sa ukázala byt kombinácia vrcholového testu semi-perfektnosti s volaním nautyho až pre konečné výsledky (B4), prip. aj s využitím d-tabuliek (B7).

	Q_3	Q_4	S_4
B1	0.000018	0.017978	0.047348
B1.1	0.000548	0.604259	7.687617
B1.2	0.000063	1.484591	0.054838
B1.3	0.000072	1.544731	0.064437
B2	0.000026	0.002923	0.114809
B3	0.000028	0.007871	0.043308
B4	0.000026	0.005997	0.046198
B5	0.000199	0.358060	3.805009
B6	0.000559	0.607393	7.689583
B7	0.000028	0.006321	0.043308

Tab. 5.2.3 Časové hodnoty pre algorimtus 4

V tabuľke 5.2.3 sa nachádzajú namerané údaje dĺžok trvania výpočtov pre náš posledný algoritmus 4. Tento sa od algoritmu 3 líši v podstate len metódou volania procedúry najdi_faktorizaciu princípom BFS. Konkrétne ju volá až po vykonaní uvažovaných heuristík pre všetky možné permutácie ofarbenia vrcholu *v* v nejakom uzle výpočtu v jednom kroku, kým predchádzajúci algoritmus ju zavolá hneď po nájdení prvej takejto možnej permutácie.

Tabuľky počtov volania jednotlivých kombinácii heuristík sú pre oba algoritmy rovnaké 5.2.2. Rozdielne merania, hoci väčšinou len nepatrne, sme zaznamenali len vo vzťahu k dĺžkam trvania výpočtov. Prakticky všetky konečné časy mal algoritmus 4 lepšie, a to vďaka jednorazovému vyhodnoteniu parciálnych riešení, miesto volania backtrackovej procedúry najdi_faktorizaciu, a dodatočného vracania sa z nej pre všetky permutácie pre daný vrchol *v*. Rovnako ako vo všetkých predchádzajúcich meraniach, nám heuristika používajúca d-tabuľky výpočet jemne predĺžila. Optimálna kombinácia heuristík, tak ako v predchádzajúcom prípade sa javí priebežný semi-perfekt check v kombinácii s volaním nautyho na záverečných výsledkoch (B4,B7).

Algoritmus 4 s implementovaným testom na perfektnosť ukončil výpočet v Q_5 do 12 hodín, nevygeneroval však žiadne riešenie. V S_5 sa algoritmy 3 a 4 bez priebežného volania nautyho pohybovali podstatne rýchlejšie ako prvé dva, no za týždeň distribuovaného výpočtu sa ani jednému nepodarilo nájsť čo len jednu semi-perfektnú faktorizáciu. Taktiež sa výpočet nepodarilo ukončiť, aj keď prehľadali viac ako polovicu grafu.

5.3 Zhrnutie experimentálnych výsledkov

V nasledujúcich riadkoch si zhrnieme namerané údaje a vyhodnotíme výsledky pre naše 4 algoritmy. Vo všeobecnosti dosiahli algoritmy 3 a 4 rádovo lepšie časy výpočtov ako 1 a 2, zvlášť čo sa týka volania testu na izomorfizmus až na konečných výsledkoch. V opačnom prípade volania nautyho priebežne, dopadol najlepšie algoritmus 2. Špeciálne na S_4 boli výsledky pri metóde ofarbovania vrcholu po vrchole neprijateľné. V star grafoch nami určeným lexikografickým poradím postupne vrcholy nie sú vždy susedné s už ofarbenými, čo spôsobuje veľkú neefektívnosť volania nautyho hlavne v počiatkoch prechádzania grafu. No aj pre hyperkocky, kde vrcholy v postupnosti vždy susedia s nejakými už ofarbenými, nám metóda priniesla horšie výsledky v porovnaní s algoritmom 2.

Spôsob optimalizovaného generovania neizomorfných faktorizácii metódou BFS je vhodnejší pre výpočty, pri ktorých je po ich prípadnej distribúcii nežiadúce volanie testu na izomorfizmus na konečných výsledkoch. V kombinácii s testom na semi-perfektnost volaným

priebežne ešte pred nautym tak predstavuje ideálne riešenie spomedzi predstavených.

Algoritmus 1 generujúci metódou DFS faktor po faktore sa ukázal ako najmenej efektívny spomedzi všetkých, pravé kvôli prehľadávaniu grafu do hĺbky, umožňujúcemu volanie drahších heuristík na parciálne riešenia, ktoré by inak mohli byt vylúčené skôr. V Q_4 bol tak program nauty priebežne volaný až 100 krát viac v porovnaní s algoritmom 2. Algoritmus 1 je však možné využiť v prípade hľadania prvých riešení z daného uzla výpočtového stromu. Pre algoritmy 1 a 2 je výhodne volať test na semi-perfektnost hneď pri generovaní možnej hrany priamo vo funkcii najdi_prvu_farbu. Nárast počtu jeho volaní rastie veľmi rýchlo so zvyšujúcou sa dimenziou, avšak test je výpočtovo nenáročný, a orezáva vetvy prislúchajúce možným parciálnym riešeniam v najskoršom možnom momente, čím ušetrí používanie náročnejších heuristických metód v ďalších krokoch. V prípade algoritmu 3 a 4 je taktiež veľmi výhodné z pohľadu celkového času trvania volať ho v každom vrchole v ktorý ofarbujú.

Pri teste na izomorfizmus vykonanom až na konečných výsledkoch, sme v kombinácii s priebežnými lačnejšími heuristikami namerali najlepšie výsledky pre algoritmy 3 a 4. Pokiaľ existuje možnosť testovania výsledkov pomocou globálnej štruktúry až na záver výpočtu, ich namerané konečné časy sú niekoľkonásobne menšie ako v prípade algoritmov 1 či 2. Spomínané rozdiely spôsobené rozličnou implementáciou testu izomorfizmu sú spôsobené faktom, že kým pri metóde ofarbovania vrcholu za vrcholom je nutné permutovať všetky farby faktorizácie F_i, pri metóde generovania faktora za faktorom sa tieto permutácie tykajú len i+1 farieb pri priebežne volanom teste pre F_i a jeho rozširujúcom faktore f_j.

V závislosti od typu hľadaných faktorizácii, je vhodné predchádzať testu na izomorfizmus časovo menej náročnými heuristikami, zvlášť test na (semi)perfektnosť, sa ukázal ako veľmi efektívny. d-tabuľky na našich grafoch neukázali žiadne výraznejšie zlepšenie výpočtu, skôr ho nepatrne predlžili. Môžu sa však pozitívne prejaviť pri vyšších dimenziách, hlavne v prípade vyššieho počtu neizomorfných faktorizacii, ktoré sú schopné rozpoznať omnoho rýchlejšie ako program nauty.

Čo sa týka generovania len navzájom rôznych faktorizacii, lepšie výsledky dosahujú metódy ofarbovania vrcholu za vrcholom a im prislúchajúce algoritmy 3 a 4. Pre prvé dva algoritmy počet generovaných identických faktorov na úkor prehľadania celého grafu rastie

rýchlosťou v praxi znemožňujúcou ich využitie bez implementácie lacnejšej priebežnej metódy optimalizácie. V prípade Q_3 je počet zhodných faktorov v priebehu celého výpočtu 7, v Q_4 už 462 605.

Všetky naše 4 algoritmy sú ľahko paralelizovatelné a vhodné pre distribuovaný výpočet. Pre svoje výpočty potrebujú iba lokálne dátove štruktúry. Jediná situácia, pri ktorej je nutné uvažovať globálnu štruktúru, nastáva pri teste izomorfizmu na konečných výsledkoch.

Predstavili sme si 4 algoritmy s rôznym prístupom generovania faktorizácii. Pre prvé dva algoritmy je zrejmé, že spôsob generovania konkrétnych 1-faktorov umožňujúci duplicitné parciálne riešenia je nevhodný pre reálne výpočty na grafoch vyšších dimenzii. Umožnili nám ale aplikovať rôzne kombinácie heuristických metód a demonštrovať tým ich (ne)výhodnosť. Druhé dva algoritmy sa zdajú byť v praxi lepšie aplikovatelné, avšak za predpokladu možnosti vzájomného porovnania výsledkov na izomorfizmus. Modifikácie algoritmov vyžadujúce časte volania testov izomorfizmu, a zvlášť tie, ktoré vyžadujú veľa permutácií vstupných ofarbení pre ktoré sa volá program nauty, nám ukázali silu NP-ťažkých problémov.

5.4 Ďalšie dosiahnuté výsledky

Cieľom práce nebola len implementácia a následne testovanie rôznych kombinácii heuristík, ale aj dopracovanie sa ku konkrétnym semi-pefektným čí iným riešeniam pre prehľadávané grafy nižších dimenzii. Uvedieme si ich súhrn.

V S_3 a Q_2 čo je vlastne šesťuholník resp. štvorec existuje jedno neizomorfné triviálne riešenie, spĺňajúce podmienku perfektnosti. V S_4 neexistuje žiadna semi-perfektná a tým pádom ani perfektná faktorizácia. S_5 sme po vyše týždni pri prehľadaní viac ako polovice možnosti všetkých faktorizacii taktiež žiadnu semi-perfektnú nenašli. Tento fakt nám naznačuje neexistenciu semi-perfektnej 1-faktorizacie na star grafoch vyšších dimenzií.

V Q_3 existuje len 1 neizomorfná semi-perfektná faktorizácia a to f_1={(0,1),(2,3),(4,5),(6,7)} f_2={(0,2),(1,5),(3,7),(4,6)} f_3={(0,4),(1,3),(2,6),(5,7)}. Nakoľko

Q_3 nie je Hamiltonovsky rozložiteľná, uvedená faktorizácia nemá konštrukčnú vlastnosť použitú pri dôkaze o existencií semi-prefektných 1-faktorizáciach na hyperkockách nepárnych dimenzii Q_n, uvedenú v úvodnej kapitole 2.

V Q_4 nám všetky algoritmy našli zhodne 1840 vhodných neidentických riešení, pričom len 35 z nich bolo navzájom neizomorfných. Zo všetkých vhodných rôznych faktorizácii 32 spĺňalo podmienku semi-perfektnosti, a všetky boli navzájom izomorfné. Z nich kanonická, t.j. najmenšia je nasledujúca

f_1={(0,1), (2,6), (3,11), (4,12), (5,7), (8,10), (9,13), (14,15)}
f_2={(0,2), (1,3), (4,5), (6,7), (8,9), (10,14), (11,15), (12,13)}
f_3={(0,4), (1,5), (2,10), (3,7), (6,14), (8,12), (9,11), (13,15)}
f_4={(0,8), (1,9), (2,3), (4,6), (5,13), (7,15), (10,11) ,(12,14)}

Ak by sme nebrali do úvahy len vhodné faktorizácie, počet všetkých neidentických v Q_4 je 4!·1840 = 44 160. V S_4 existuje 84 rôznych faktorizácii s naším počiatočným ofarbením vrcholu 1234, z nich je 9 navzájom neizomorfných, a ani jedna nespĺňa podmienku semi-perfektnosti.

V Q_5 sme našli desiatky neizomorfných vhodných semi-perfektných 1-faktorizacii, no ani jedna nespĺňala podmienku perfektnosti, napriek tomu, že ich počet výrazne vzrástol oproti Q_4. Neexistencia perfektnej faktorizácie na hyperkockách Q_3, Q_4, a Q_5 nás vedie k presvedčeniu o jej neexistencii aj na hyperkockách vyšších dimenzií. Naše presvedčenie potvrdzuje aj fakt, že sme v Q_6 za 2 týždne výpočtu žiadnu nenašli

Kapitola 6

Implementácia algoritmov a testovací program

Okrem samotného algoritmu a použitých heuristík má vplyv na rýchlosť výpočtu výber programovacieho jazyka ako aj vhodná implementácia. Ako najlepšia možnosť sa ukázal jazyk C++, vzhľadom na jeho rýchlosť, univerzálnosť a modulárnosť. Ďalší dôvod bol samotný program nauty, ktorý je napísaný v jazyku C, a ktorého knižnice a procedúry v programe využívame.

Celočíselné premenné sme podľa možnosti použili typu unsigned (kladné celé čísla). Nemalou výhodou je tiež zapísať si často používané premenné do lokálnych statických polí, namiesto dynamického vyhodnocovania, napr. čo sa týka susedných vrcholov grafu. Pri grafoch s vyšším počtom vrcholov, samozrejme vzniká potreba voľby vhodnej hashovacej funkcie. V prípade príliš veľkého vstupného grafu G, sa javí ako najlepšia možnosť kombinácia lokálnych premenných s vhodnou voľbou dynamickej rozhodovacej funkcie, samozrejme s ohľadom na distribuovaný výpočet. V samotnom algoritme je kvôli optimalizácii takmer nevyhnutná vhodná hierarchia a prioritizácia cyklov či rozhodovacích podmienok. Pri teste na zhodu celočíselných prvkov je najrýchlejšie porovnanie na 0, čomu by

sa podľa možnosti mali prispôsobiť dátové štruktúry programu.

Kompilátor zdrojového kódu, v našom prípade gpp a jeho možnosti hrajú tiež svoju úlohu pri zrýchlení výpočtu. Program je možné skompilovať spôsobom zameraným práve na rýchlosť, na úkor veľkosti jeho kódu v cache pamäti procesora. Výpočty uvedené v tabuľkách v kapitole 5 boli vykonané na 2 jadrovom 64 bitovom procesore s frekvenciou 2.1 GHz. Zložitejšie výpočty pre vyššie dimenzie grafov boli vykonané distribuované na viacerých procesoroch.

Prílohou diplomovej práce je softvérový program Faktorizacia, primárne určený na testovanie a vyhodnocovanie použitia nami uvažovaných heuristík na grafoch Q_4 a S_4. Výsledky pre vyššie dimenzie ako 4 sú uvedené v kapitole 5, a tieto nie sú súčasťou testovacieho programu, nakoľko dĺžky výpočtov pre ne exponencionálne rastú, a trvajú niekoľko hodín až dní. Samotný testovací program beží v systémoch unix/linux. Je mozne ho spustiť príkazom ./faktorizacia v adresari /Faktorizacia. Vstup aj vystup je smerovaný priamo cez konzolu. Menu programu Faktorizacia je intuitívne, a nevyžaduje si takmer žiadne opisovanie. Užívateľ má možnosť si vybrať každý z predstavených algoritmov, ako aj heurustík uvažovaných v práci. Zdrojové kódy obsahujú samostatné knižnice pre každý algoritmus ako aj pre každú testovanú topológiu Q_4 a S_4. Taktiež je v nich zakomponovaný skompilovaný program nauty ver.2.2, bližšie informácie o ňom je možné nájsť v [16]. Uvedieme si krátky popis funckií k triedam algoritmov 1 a 2 (obr. 4.2.1.1 a 4.2.1.3), pre ostatné algoritmy sú popisi a využitie procedúr vzhľadom na ich pseudokódy a heuristiky intuitívne.

- *void najdi_faktorizaciu* - hlavná procedúra backtracku
- *void najdi_prvu_farbu* – procedúra volaná v (3) procedúrou najdi_faktorizaciu, generuje príslušné 1-faktory
- *int check_P1F*- funkcia rátajuca semi-perfekt test priebežne
- *int check_P1F_koniec* - funkcia rátajúca semi-perfekt test na konečných výsledkoch
- *void vyrataj_D_tabulku* – procedúra ráta d-tabuľku príslušnej 1-faktorizacie
- *void NaplnDefault()* – generuje počiatočné podmienky výpočtu, ako sú uvažované

grafy, ich počiatočné ofarbenia, tabuľky susedných vrcholov a hrán, inicializuje hranové grafy k nami uvažovanými (pre potreby volania nautyho) a iné

- *int RunNautyPocas* – funkcia volajúca priebežný test izomorfizmu
- *int RunNautyKoniec* – funkcia volajúca test izomorfizmu na konečných výsledkoch

Kapitola 7

Záver

1-faktorizácie sú témou zasluhujúcu si nemalú pozornosť. Presviedčajú nás o tom mnohé jej využitia v praxi, ako aj často netušená blízkosť k problémom a otázkam denného života. Hoci sú mohutne študované už pár dekád, poskytujú čoraz viac priestoru pre ďalšie otázky, na ktoré sa stále ťažšie hľadajú odpovede.

Jedným z cieľov našej diplomovej práce bola aj snaha priblížiť čitateľovi práve tematiku 1-faktorizacii na grafoch. Zaoberali sme sa otázkou ich vzájomného izomorfizmu, ako aj zložitosťou a výpočtovou náročnosťou algoritmov súvisiacich s ich generovaním. Dúfame, že sme v ňom vzbudili aspoň minimálny záujem o tuto atraktívnu a zároveň veľmi aktuálnu tému vedy zvanej informatika.

V práci sme poskytli všeobecný prehľad metód a dosiahnutých výsledkov v oblasti 1-faktorizacii na regulárnych grafoch, pričom sme sa zamerali hlavne na semi-perfektné a perfektné faktorizácie. Ukázali sme, že ani prípadne veľké množstvo prvých nám nezaručuje existenciu druhých. Čitateľovi sme tiež priblížili techniky a spôsoby neizomorfného generovania objektov.

Hlavným cieľom našej práce bola ale snaha navrhnúť algoritmy pre generovanie neizomorfných 1-faktorizacii v hyperkockách a star grafoch. Predstavili sme si 4 algoritmy

využívajúce rôzne metódy prehľadávania grafov a generovania faktorizácií. Do nich sme postupne vkladali rôzne heuristiky, a následne vyhodnocovali ich efekt na výpočet. Na základe nameraných hodnôt sme sa pokúsili navrhnúť vhodne kombinácie jednotlivých optimalizanych metód pre uvažované grafy vyšších dimenzii.

Ďalším cieľom nášho úsilia boli odpovede na otázky existencie semi-perfektných a perfektných 1-faktorizacií na hyperkockách a star grafoch nižších dimenzii, ktoré sme sčasti aj úspešne zodpovedali. Na základe našich zistení, sme vyslovili predpoklady o prípadnej (ne)existencii hľadaných objektov na uvažovaných grafoch vyšších dimenzií. Vedľajším efektom nasej snahy je príloha diplomovej práce, testovací program vhodný na demonštráciu efektu jednotlivých heuristických metód. Dúfame, že v budúcnosti poslúži minimálne tomuto účelu.

Zoznam obrázkov a tabuliek

Zoznam použitej literatúry

[1] Reinhard Diestel, Graph Theory, 2nd ed., Springer-Verlag, New York, 2000

[2] H. Dinitz, D. K. Garnick, and B. D. McKay, There are 526,915,620 nonisomorphic one-factorizations of K12, J. Combin. Des. 2 (1994), 273–285.

[3] E. Mendelsohn, and A. Rosa, One-factorizations of the complete graph—a survey, J. Graph Theory 9 (1985), no. 1, 43–65.

[4] A. Kotzig: Hamilton graphs and Hamilton circuits Theory of Graphs and its Applications Proceedings of the Symposium of Smolenice, 1963 Nakl. CSAV Praha, 1964, pp. 62–82

[5] J. H. Dinitz, D. K. Garnick There are 23 Nonisomorphic Perfect One-Factorizations of K14,
Journal of Combinatorial Designs, Vol. 4, No. 1 (1996)

[6] Rastislav Královič and Richard Královič, On Semi-perfect 1- factorizations,
Structural information and communication complexity, 216–230, Lecture Notes in Comput. Sci., 3499, Springer, Berlin, 2005.

[7] Vasil S. Gochev and Ivan S. Gotchev - On K-Semi-Perfect 1-Factorizations Of Qn And Craft's Conjecture, 2000 Mathematics Subject Classification. Primary 05C38; Secondary 05C45, 05C70, 68R10, 68M15.

[8] Douglas W. Bass, I. Hal Sudborough: Hamilton Decompositions and (n/2)- Factorizations of Hypercubes, Journal of Graph Algorithms and Applications,
7(2003)79–98

[9] D. A. Pike, Hamiltonian decompositions of line graphs of perfectly 1- factorizable graphs of even degree, Australian Journal of Combinatorics 12 (1995), 291-294.

[10] I. A. Faradžev, Generation of nonisomorphic graphs with a given degree sequence (Russian), in “Algorithmic Studies in Combinatorics” (Nauka, Moscow, 1978) 11–19

[11] R. C. Read, Every one a winner, Annals Discrete Math., 2 (1978) 107–120.

[12] B. D. McKay, Isomorph-free exhaustive generation, J Algorithms, 26 (1998) 306-324.

[13] T. Gruner, R. Laue and M. Meringer, Algorithms for group actions: homomorphism principle and orderly generation applied to graphs, preprint 1995

[14] K. Day A. Tripathi, "A Comparative Study of Topological Properties of Hypercubes and Star Graphs" IEEE Transactions on Parallel and Distributed Systems, Vol. 5 , No. 1 (1994) 31 - 38

[15] B. D. McKay, Practical graph isomorphism, Congressus Numerantium 30 (1981) 45–87.

[16] B. D. McKay, nauty user’s guide (version 2.2), dostupne na http://cs.anu.edu.au/~bdm/nauty/