Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Analýza štruktúry snarkov

Diplomová práca

2019
Bc. Rastislav Simeunovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Analýza štruktúry snarkov

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	prof. RNDr. Martin Škoviera, PhD.

Bratislava, 2019
Bc. Rastislav Simeunovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Rastislav Simeunovič
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Analýza štruktúry snarkov
Analysis of the structure of snarks

Anotácia: Cieľom práce je vytvoriť program na skúmanie vlastností snarkov a ich následnú analýzu.

Cieľ: Cieľlom práce je analýza rozličných štrukturálnych vlastností snarkov, čiže kubických grafov bez hranového 3-zafarbenia. Medzi skúmané vlastnosti budú patriť cyklická súvislosť snarkov, klustre cyklov, permutačný charakter, kritickosť, ireducibilita a iné. Na testovanie týchto vlastností bude treba vytvoriť požadované algoritmy napásať príslušné programy. Zistené výsledky sa budeme analyzovať a niektoré sa budeme snažiť zovšeobecniť na všeobecne platné matematické tvrdenia, ktoré dokážeme.

Literatúra: Aktuálna časopisecká literatúra a poznámky vedúceho práce

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

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 16.11.2016

Dátum schválenia: 14.12.2016 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Moje poďakovanie patrí v prvom rade môjmu školiteľovi profesorovi Martinovi Škovierovi, za jeho odborné rady, za jeho ochotu stretnúť sa vždy keď som mal nejaký problém a za mnoho vecí, ktoré som sa pri tejto práci naučil. Tak isto sa chcem poďakovať môjmu zamestnávateľovi, ktorý mi dal dostatok priestoru a voľna na tvorbu práce. Na záver sa chcem poďakovať mojej rodine, blízkym a kamarátom, ktorí ma podporovali pri tvorbe tejto práce.

Abstrakt

Cieľom práce je analýza rozličných štrukturálnych vlastností snarkov, čiže kubických grafov bez hranového 3-zafarbenia. Medzi skúmané vlastnosti budú patriť cyklická súvislosť snarkov, klastre cyklov, permutačný charakter, kritickosť, ireducibilita a iné. Na testovanie týchto vlastností bude treba vytvoriť požadované algoritmy napísať príslušné programy. Zistené výsledky sa budeme analyzovať a niektoré sa budeme snažiť zovšeobecniť na všeobecne platné matematické tvrdenia, ktoré dokážeme.

Kľúčové slová: graf, snark, klaster

Abstract

The aim of this work is analysis of structural properties of snarks, cubic graphs with no 3-edge-colouring. The investigated properties will include clusters of 5-cycles, permutation character, irreducibility and others. It will be necessary to design required algorithms and write the corresponding programs to test this properties. Found properties will be analyzed and we try to generalize them to valid mathematical statements and mathematically prove them.

Keywords: graph, snark, cluster

Obsah

Úvod **1**

1 Základné pojmy z teórie grafov **2**

2 Snarky **8**

2.1 Redukovanie problémov na snarky 8

2.2 História snarkov 9

2.3 Významné snarky 9

2.3.1 Petersenov graf 9

2.3.2 Blanušove snarky 10

2.3.3 Descartesov snark 10

2.3.4 Szekeresov snark a Watkinsov snark 11

2.3.5 Flower snarky 11

2.4 Vlastnosti snarkov 12

2.5 Konštrukcie snarkov 13

2.5.1 Trojuholníková substitúcia 13

2.5.2 4-Súčin (dot product) 14

2.5.3 5-Súčin (star product) 14

2.5.4 Negátorová substitúcia 15

3 Permutačné snarky **17**

3.1 Permutačný snark 17

3.1.1 Vlastnosti permutačných snarkov 18

3.1.2 Skonštruované permutačné snarky 19

3.2 Konštrukcia permutačných snarkov 19

3.2.1 4-Súčin 20

3.2.2 5-Súčin 20

3.2.3 Negátorová substitúcia 22

3.3 Polopermutačné snarky 24

4 Klastre **26**
4.1 Klastre Petersenovho grafu 26
4.1.1 Konštrukcia klastrov Petersenovho grafu 27
4.1.2 Popis a vizualizácia Petersenovských klastrov 27
4.1.3 Nepetersenovské klastre 32
4.2 Dôkaz úplnosti množiny petersenovských klastrov 33

5 Cyklicky 5-súvislé permutačné snarky **41**
5.1 Vizualizácia PERM5.34 42

6 Skúmané vlastnosti **46**
6.1 Existujúce programy 46
6.2 Permutačnosť snarku 46
6.3 Polopermutačnosť snarku 47
6.4 Všetky izomorfné a neizomorfné reprezentácie permutačného 2-faktoru . 47
6.5 Parita permutačných 2-faktorov 47
6.6 Involúcia snarku 48
6.7 Cycle double cover 49
6.8 Klastre v grafe 50
6.8.1 Vypisovanie vrcholov 50

7 Funkcionalita programu **52**
7.1 Grafické prostredie 52
7.2 Vstupný súbor 54
7.3 mainFunction() 54
7.4 Výstupný súbor 67

8 Výsledky **69**

Zoznam obrázkov

2.1 Petersenov graf 10
2.2 Blanušove snarky 10
2.3 Descartesov snark 11
2.4 Szekeresov a Watkinsov snark 11
2.5 Flower snarky 12
2.6 Trojuholníková substitúcia 13
2.7 4-Súčin 14
2.8 5-súčin 15
2.9 Negátorová substitúcia 16

3.1 Permutačný 2-faktor Petersenovho grafu 18
3.2 Permutačný 4-súčin 20
3.3 Cykly permutačného 2-faktoru v 5-cykle 21
3.4 Permutačný 5-súčin 22
3.5 Permutačná negatorová substitúcia 24
3.6 Polopermutačný 2-faktor 25

4.1 Popísaný Petersenov graf 28
4.2 Pentagón 29
4.3 Dvojitý Pentagón 29
4.4 Dyád 30
4.5 Trojitý Pentagón 30
4.6 Trojbunka 31
4.7 Triáda 31
4.8 Isochróm 31
4.9 Heterochróm-1 32
4.10 Heterochróm-1 32
4.11 prvý Blanušov snark 33
4.12 Odstránenie nesusedných vrcholov z Petersenovho grafu 33
4.13 35
4.14 35

4.15 38

5.1 Permutačný 2-faktor PERM5.34-1 42
5.2 PERM5.34 - Typ 1 a 2 42
5.3 Typ 1 a typ 2 43
5.4 PERM5.34 45

6.1 Permutácia permutačného 2-faktoru 48
6.2 B1 involúcia 49

7.1 Grafické prostredie 53

Zoznam tabuliek

3.1 Počty snarkov a permutačných snarkov [19] 19

7.1 Počty bezchordových cyklov v závislosti od rádu grafu 55

Úvod

Cieľom tejto práce je skúmanie a lepšie pochopenie štruktúry snarkov - súvislých bezmostových grafov, ktorých chromatický index je 4. Aj napriek tomu, že existujú nástroje ako vytvárať nové snarky a poznáme niekoľko nekonečných tried snarkov, tak sa jedná o pomerne mladú triedu grafov, ktorá nie je stále dostatočne preskúmaná a nepoznáme štruktúru mnohých z nich. V dnešnej dobe o mnohých grafoch vieme povedať, že patria medzi snarky, avšak nepoznáme množstvo ich ďalších vlastností ako aj ich štruktúru.
Práve preto sme sa rozhodli vytvoriť program, ktorý bude niektoré z vlastností overovať a hľadať. Potreba využitia počítačového programu na overovanie vlastností je zjavná, keďže sa často jedná o veľké grafy a pre niektoré vlastnosti sa jedná o milióny iterácií prehľadávania a počítania, ktoré by človeku pri počítaní na papier zabrali prinajlepšom dni a počítač to zvládne za pár sekúnd. Okrem zistenia vlastností a štruktúry daného grafu program pomôže pri ľahšej vizualizácií grafov.
Okrem toho sa budeme venovať aj podgrafom Petersenovho grafu - klastrom, a ich výskyte v jednotlivých snarkov a priblížia nám súvislosti medzi jednotlivými snarkami. Do tejto časti bude patriť aj dôkaz úplnosti týchto podgrafov.
Samotný program, ktorý je gro tejto práce je napísaný v jazyku C++. Tento jazyk bol zvolený najmä kvôli tomu, že sa jedná o veľmi rýchly jazyk na rozdiel od možno populárnejších a jednoduchších jazykov ako napríklad Python alebo Java.

Kapitola 1

Základné pojmy z teórie grafov

Množstvo pojmov z teórie grafov sa v tejto práci bude opakovať. Aby sme predišli nepochopeniu medzi autorom a čitateľom z dôvodu rôznych interpretácii jednotlivých pojmov, budú v tejto kapitole niektoré pojmy zadefinované a pri všetkých týchto pojmoch použitých ďalej v práci budeme vychádzať z týchto definícií. Pojmy z teórie grafov, ktoré nie sú definované v tejto kapitole budú vychádzať z Diestelovej knihy Teória grafov[3].

Graf

Graf G je usporiadaná dvojica dvoch množín $G = (V, E)$. $V = \{v_1, v_2, \ldots, v_n\}$ je neprázdna množina vrcholov grafu G a $E = \{e_1, e_2, \ldots, e_p\}$ je množina hrán grafu G. Každá hrana $e \in E$ je interpretovaná ako neusporiadaná dvojica vrcholov $e = \{v_i, v_j\}$, kde vrcholy v_i aj v_j sa nazývajú koncovými vrcholmi hrany e a $v_i, v_j \in V$. Takýto graf G sa nazýva neorientovaný graf. Táto práca sa zaoberá výlučne neorientovanými grafmi. O hrane $e = \{v_i, v_j\}$ hovoríme, že je incidentná s vrcholom v_i aj s vrcholom v_j. Označenie hrán písmenom e a vrcholov písmenom v pochádza z anglických slov edge a vertex.

Počet vrcholov

Pod pojmom počet vrcholov grafu G sa bude rozumieť mohutnosť množiny V a označuje sa ako $|G|$. Čitateľ sa môže stretnúť aj s pojmom rád grafu, ktorý je ekvivalentný s pojmom počet vrcholov grafu.

Počet hrán

Počtom hrán grafu G sa myslí mohutnosť množiny E a označuje sa ako $||G||$.

Stupeň vrcholu

Stupňom vrcholu v_i sa označuje počet hrán, ktoré sú s vrcholom v_i incidentné. Stupeň vrcholu sa označuje aj ako $\deg(v_i)$
Pre neorientované grafy je známe $\sum_{v \in V} \deg(v) = 2||G||$

Kubický graf

Graf $G = (V, E)$, pre ktorého všetky vrcholy $v \in V$ platí, že $\deg(v) = 3$ sa nazýva kubický graf. Počet hrán v takomto grafe je $||G|| = (3 * |G|)/2$ a vyplýva z toho, že počet vrcholov grafu G je párny.

Odstránenie hrany

Ak $G = (V, E)$ je graf a e_i je jeho hrana, tak $G - e_i$ označuje graf $(V, E - \{e_i\})$ Pod týmto označením sa myslí odstránenie prvku e_i z množiny E a zníži sa tým stupeň koncových vrcholov danej hrany.

Prerezanie hrany

Keďže v práci sa pracuje s kubickými grafmi a odstránenie hrany vytvára dva vrcholy stupňa 2 budeme často miesto pojmu odstránenie hrany používať pojem prerezanie hrany. Myslí sa tým, že hrana e_i z grafu G sa rozdelí na dve polhrany. Každá polhrana má len jeden koncový vrchol. Tie sa môžu následne prepojiť s inými polhranami, čím opäť vzniknú celé hrany. Aj pri odstránení hrany sa uvažuje, že v grafe ostanú dve polhrany. Označenie visiaca polhrana je ekvivalentná s označením polhrana.

Odstránenie vrcholu

Odstránenie vrcholu $v_i \in V$ z grafu $G = (V, E)$ sa označuje ako $G - v_i$. Myslí sa pod tým odstránenie prvku v_i z množiny V. Zároveň sa spolu s vrcholom v_i odstránia z grafu G aj všetky hrany, ktoré sú s vrcholom v_i incidentné. Platí teda, že ak hrana $e_i = \{v_k, v_l\}$, kde $v_k = v_i$ alebo $v_l = v_i$ bude hrana e_i odstránená z grafu tiež.

Podgraf

Hovoríme, že graf $H = (V', E')$ je podgraf grafu $G = (V, E)$ a píšeme $H \subseteq G$, ak $V' \subseteq V$ a $E' \subseteq E$. Je zrejmé, že každý podgraf $H \subseteq G$ sa dá získať postupným odstraňovaním hrán a vrcholov z grafu G.

Indukovaný podgraf

Indukovaný podgraf $H = (V', E')$ grafu $G = (V, E)$ je taký podgraf, ktorý vznikol odstraňovaním vrcholov z grafu G. Z definície odstraňovania vrcholov vyplýva, že boli odstránené iba hrany, ktoré boli s odstránenými vrcholmi incidentné.

Susedné hrany

Dve hrany v grafe $G = (V, E)$ sa označujú ako susedné, pokiaľ sú existuje vrchol $v \in V$, pre ktorý platí, že obe hrany sú s vrcholom v incidentné.

Susedné vrcholy

Dvojica vrcholov $v_i, v_j \in V$ z grafu $G = (V, E)$ sa nazýva susednými vrcholmi, ak existuje hrana $e \in E$, taká, že $e = \{v_i, v_j\}$. V prípade viacerých vrcholov sa pod pojmom susedné vrcholy rozumie, že medzi každou dvojicou vrcholov z tejto množiny existuje cesta C, ktorá obsahuje iba vrcholy z tejto množiny.

Sled

Sledom sa nazýva postupnosť vrcholov a hrán $v_0, e_1, v_1, e_2, ..., e_n, v_n$ v grafe $G = (V, E)$ a platí, že hrana e_i má koncové vrcholy v_{i-1}, v_i. V slede sa hrany a vrcholy môžu opakovať. Sled pre ktorý platí, že $v_0 = v_n$ sa nazýva uzavretý sled.

Cesta

Sled v ktorom sa žiaden vrchol neopakuje sa nazýva cesta. Z podmienky, že sa žiaden vrchol neopakuje vyplýva dôsledok, že ani žiadna hrana sa neopakuje. Na označenie cesty sa budú zväčša používať iba vrcholy danej cesty a nie hrany. Označenie cesty klm bude teda znamenať cestu z vrcholu k do vrcholu l a odtiaľ do vrcholu m. Dĺžkou cesty sa rozumie počet hrán, ktoré táto cesta obsahuje.

Cyklus

Pokiaľ v grafe $G = (V, E)$ platí, že existuje postupnosť vrcholov a hrán $v_0, e_1, v_1, e_2, ..., e_n, v_n$, pričom platí, že $v_0 = v_n$ a pre ľubovoľné $i > 0$ a ľubovoľné $j > 0$ rôzne od i platí $v_i \neq v_j$, tak sa táto postupnosť nazýva cyklus.
Pojem kružnica je ekvivalentný s pojmom cyklus.

Odstránenie cesty

Ak sa z grafu $G = (V, E)$ odstráni cesta $C = \{v_0, e_1, v_1, .., e_n, v_n\}$ myslí sa tým odstránenie všetkých vrcholov, ktoré sú v ceste C. Z definície odstránenia vrcholov vyplýva, že sa odstránia aj všetky incidentné hrany, čiže aj tie, ktoré nepatria do cesty C.

Vzdialenosť hrán

Pre nesusedné hrany e_i, e_j definujeme vzdialenosť hrán ako dĺžku najkratšej cesty C, kde e_i je hrana, ktorej jeden koncový vrchol je prvý vrchol v C a posledný vrchol v C je jeden z koncových vrcholov hrany e_j, pričom platí, že $e_i, e_j \notin C$.

Vzdialenosť vrcholov

Pre vrcholy v_i, v_j definujeme ich vzdialenosť ako dĺžku najkratšej cesty z v_i do v_j. Z toho vyplýva, že vzdialenosť susedných vrcholov je 1, keďže majú spoločnú hranu.

Dĺžka cyklu

Dĺžkou cyklu sa označuje počet vrcholov $v \in V$, pre ktoré platí, že $v \in C$, kde C je cyklus.

Bezchordový cyklus

Pokiaľ existuje v grafe $G = (V, E)$ cyklus $C = \{v_0, e_1, v_1, e_2, ..., e_n, v_n\}$ a v grafe G neexistuje hrana $e_x = \{v_{x1}, v_{x2}\}$, pre ktorú platí, že $e_x \notin C$ a $v_{x1} \notin C$ alebo $v_{x2} \notin C$ nazýva sa tento cyklus bezchordový. Zjednodušene povedané neexistuje hrana medzi dvoma vrcholmi, ktoré nie sú v cykle C susedné. Dôsledok tohto je, že pre ľubovoľnú podmnožinu vrcholov z cyklu C neexistuje kratší cyklus.
Pokiaľ v grafe existuje takáto hrana nazýva sa chorda. Pojem indukovaný cyklus je ekvivalentný s pojmom bezchordový cyklus.

Odstránenie cyklu

Odstránením cyklu C z grafu $G = (V, E)$ sa myslí odstránenie všetkých vrcholov v z množiny V, pre ktoré platí, že $v \in C$. Spolu s vrcholmi sa odstránia aj incidentné hrany.

Hamiltonovský cyklus

Podgraf grafu $G = (V, E)$, ktorý je cyklus a obsahuje všetky vrcholy z grafu G sa volá hamiltonovský cyklus.

Súvislý graf

Graf $G = (V, E)$, v ktorom pre ľubovoľné vrcholy $v_i, v_j \in V$, pričom $i \neq j$ existuje cesta v_i-v_j sa nazýva súvislý graf.

k-Súvislý graf

Pokiaľ odstránenie ľubovoľných $k - 1$ vrcholov z grafu zachová jeho súvislosť nazýva sa takýto graf $k-$súvislý. Inak povedané, aby z k-súvislého grafu $G = (V, E)$ vznikol nesúvislý graf, treba z neho odstrániť najmenej k vrcholov.

Komponent grafu

Maximálny súvislý podgraf grafu $G = (V, E)$ sa nazýva komponent. Pokiaľ má G práve jeden komponent, tak G je súvislý.

Most

Pokiaľ v grafe $G = (V, E)$ existuje hrana $e \in E$, pre ktorú platí, že $G - e$ má väčší počet komponentov ako G, tak sa hrana e nazýva most. Most e je teda taká hrana, ktorej odstránenie urobí graf nesúvislým.

Obvod grafu

Obvodom grafu $G = (V, E)$ sa označuje dĺžka najkratšieho cyklu v grafe G.

Chromatický index

Chromatický index grafu $\chi'(G)$ je minimálny počet farieb potrebných na to, aby sa každej hrane grafu $G = (V, E)$ priradila práve jedna farba, tak, aby všetky susedné hrany mali priradené rôzne farby.
Pre $\chi'(G) = k$ sa zvykne používať aj vyjadrenie, že graf G je hranovo k-zafarbiteľný.
Pokiaľ sa v práci vyskytne zjednodušené vyjadrenie $k-$zafarbiteľný, bude sa tým myslieť výlučne hranová zafarbiteľnosť.

Faktor grafu

Podgraf $H = (V_H, E_H)$ grafu $G = (V, E)$, ktorého množina vrcholov obsahuje rovnaké prvky ako množina vrcholov grafu G - tj. $V = V_H$, sa nazýva faktorom grafu G.

k-Faktor

Pokiaľ je $H = (V', E')$ faktor grafu $G = (V, E)$ a pre $\forall v \in V'$ platí, že $deg(v) = k$, označuje sa tento faktor ako $k-$faktor.

Hamiltonovský graf

Hamiltonovský graf $G = (V, E)$ je graf, ktorý obsahuje aspoň jeden hamiltonovský cyklus.

Hypohamiltonovský graf

Hypohamiltonovský graf, je taký graf $G = (V, E)$, ktorý nie je hamiltonovský, avšak pre $\forall v \in V$ platí, že podgraf $G - v$ je Hamiltonovský graf.

Planárny graf

Graf $G = (V, E)$ je planárny graf, pokiaľ sa dá reprezentovť v euklidovskej rovine tak, aby sa žiadne dve hrany grafu G nepretínali.

Grafový izomorfizmus

Graf $G = (V, E)$ je izomorfný s grafom $H = (V', E')$, pokiaľ existuje bijektívne zobrazenie $f : V \to V'$, kde pre všetky vrcholy $v_i, v_j \in G$ platí $\{v_i, v_j\} \in G \Leftrightarrow \{f(v_i), f(v_j)\} \in H$. Ak su grafy G a H izomorfné, používa sa označenie $G \cong H$

Automorfizmus

Automorfizmus je taký izomorfizmus v grafe $G = (V, E)$, že existuje bijekcia $f : V \to V$, že pre všetky vrcholy $v_i, v_j \in G$ platí $\{v_i, v_j\} \in G \Leftrightarrow \{f(v_i), f(v_j)\} \in G$.

Cyklická súvislosť

Cyklická súvislosť grafu sa označuje ako λ_C a udáva najmenší počet hrán potrebných na odstránenie z grafu $G = (V, E)$, pričom vzniknutý podgraf H nebude súvislý a každý komponent bude obsahovať aspoň jeden cyklus.

Kapitola 2

Snarky

Celá práca je zameraná na snarky. Táto kapitola vysvetlí, čo sú to snarky, povie krátko o ich histórii a vysvetlí niekoľko spôsobov ako ich konštruovať.

Definícia 1. Nech pre graf $G = (V, E)$ platí:

- G je kubický graf,
- $\chi'(G) = 4$,
- Graf G neobsahuje hranu $e \in E$, ktorá je most.

Potom je graf G snark.

„Napriek tejto pomerne jednoduchej definícii a vyše storočia skúmania snarkov sú ich vlastnosti a štruktúra veľkou neznámou“[8]
Z definície snarku vyplýva a dá sa dokázať množstvo iných vlastností. Napríklad to, že ak je graf snark, tak nie je hamiltonovský graf. Napriek tomu, že žiaden snark nie je hamiltonovský graf, existuje množstvo snarkov, ktoré sú hypohamiltonovské grafy. Táto práca sa bude zaoberať výlučne snarkami a okrem toho, že všetky skúmané grafy budú snarky, budú všetky spĺňať aj iné vlastnosti. Všetky snarky budú mať cyklickú súvislosť $\lambda_C \geq 4$ a obvod všetkých snarkov bude ≥ 5. Takéto snarky sa nazývajú netriviálne.

2.1 Redukovanie problémov na snarky

Dôvod zaoberať sa snarkami, za účelom spoznania ich vlastnosti a lepšieho pochopenia ich štruktúry je motivovaný tým, že množstvo problémov a tvrdení v teórií grafov sa dá zredukovať na snarky. To znamená, že pokiaľ sa dokáže alebo vyvráti, že tvrdenie platí pre snarky, tak platí respektíve neplatí pre všetky grafy. Medzi takéto problémy patrí napríklad hypotéza *5-flow conjecture* od britského matematika W.T.Tutteho, ktorá súvisí s vrcholovým farbením planárnych grafov. Ďalší problém, ktorý sa dá zredukovať na snarky je hypotéza *cycle double cover conjecture*, ktorú vysvetlíme v kapitole 6. Určiť či je graf snark je NP-úplný problém.[7]

2.2 História snarkov

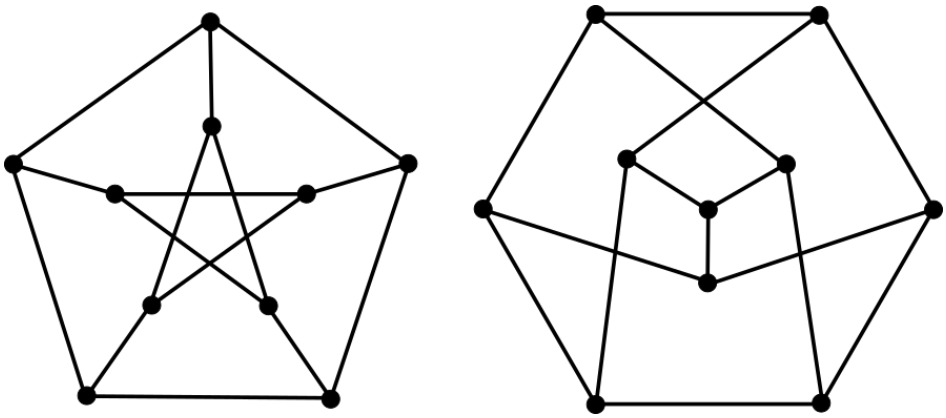
Prvá zmienka o snarkoch pochádza z roku 1880 po tom, ako škótsky matematik P.G.Tait dokázal, že *teoréma o štyroch farbách* je ekvivalentná s tvrdením, že žiaden snark nie je planárny. V tom čase sa pojem snark ešte nepoužíval. Až osemnásť rokov po tomto dôkaze bol objavený vôbec prvý snark. Do roku 1970 bolo dokopy objavených iba 5 netriviálnych snarkov. Výrazný posun nastal v roku 1972, keď bol objavený prvý spôsob ako generovať nekonečné množstvo snarkov.
Označenie snark bolo použité prvýkrát až v roku 1976 americkým matematikom Martinom Gardnerom[5]. Tento názov prebral z básne Lewisa Carolla *The Hunting of the Snark*, ktorej dej zachytáva lovenie záhadnej a nebezpečnej príšery.

2.3 Významné snarky

Dnes už je známe, že snarkov je nekonečne veľa a existuje viacero spôsobov ako generovať a konštruovať nové snarky. V roku 2012 štvorica matematikov z univerzít v Gente a Umeå vygenerovala všetky snarky do rádu tridsaťštyri s obvodom ≥ 4. Dokopy ich je vyše tridsať miliónov. V generovaní väčších grafov pokračovali a dnes sa na portáli House of Graph[19] dajú stiahnuť súbory so všetkými snarkami do tridsaťšesť vrcholov, ktorých obvod je ≥ 4. Takýchto snarkov je takmer päťsto miliónov. Aj napriek tomu, že dnes je už množstvo grafov skonštruovaných a objavených vznikali snarky pomaly a niektoré sú významnejšie ako ostatné.

2.3.1 Petersenov graf

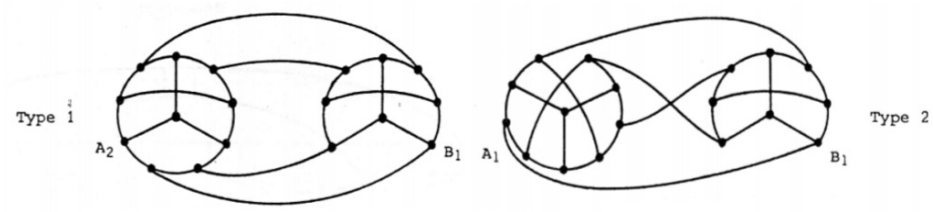
Najznámejší, najmenší a zároveň prvý objavený snark je Petersenov graf, ktorý nesie meno po svojom objaviteľovi dánskom matematikovi Juliusovi Petersenovi. Aj napriek tomu, že zmienka o tomto grafe sa našla v poznámkach britského matematika Sira Alfreda Braya Kempeho z roku 1886 sa jeho objavenie pripisuje Petersenovi a datuje sa na rok 1898.
S desiatimi vrcholmi a pätnástimi hranami sa jedná o najmenší snark. Aj napriek tomu, že má iba desať vrcholov sa v ňom nachádza až dvanásť rôznych cyklov dĺžky päť. Petersenov graf sa môže označiť ako základný stavebný kameň snarkov, keďže veľa dnes známych snarkov vzniklo modifikovaním Petersenovho grafu a jeho podgrafov.
Petersenov graf je symetrický, hypohamiltonovský graf, ktorého cyklická súvislosť je $\lambda_C = 5$ a všetky jeho vrcholy su automorfné.



Obr. 2.1: Petersenov graf - najmenší existujúci snark

2.3.2 Blanušove snarky

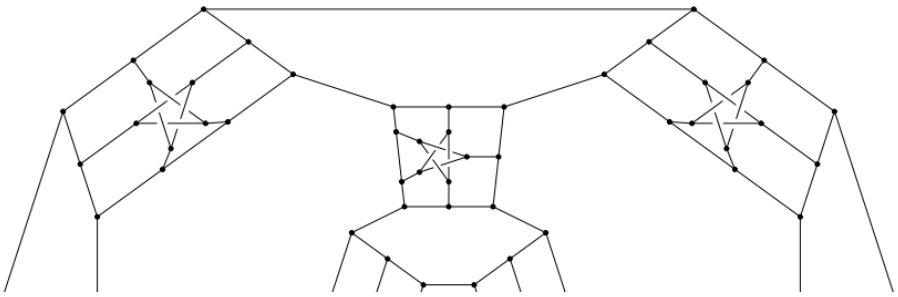
Chorvátsky matematik Danilo Blanuša zostrojil v roku 1946 dva nové snarky. Konštrukcia pozostáva z dvoch Petersenových grafov, pričom sa z každého odstráni ľubovoľný vrchol a následne sa takto vzniknuté podgrafy navzájom prepoja. Od spôsobu vzájomného prepojenia závisí, ktorý z dvoch grafov vznikne, pričom sa v niektorých vlastnostiach líšia. Oba Blanušove snarky majú osemnásť vrcholov a dvadsaťsedem hrán a často sa označujú ako B_1 a B_2. Po Petersenovom grafe sú to najmenšie dva netriviálne snarky.



Obr. 2.2: Blanušove snarky[21] - vzniknú spojením dvoch Petersenových grafov

2.3.3 Descartesov snark

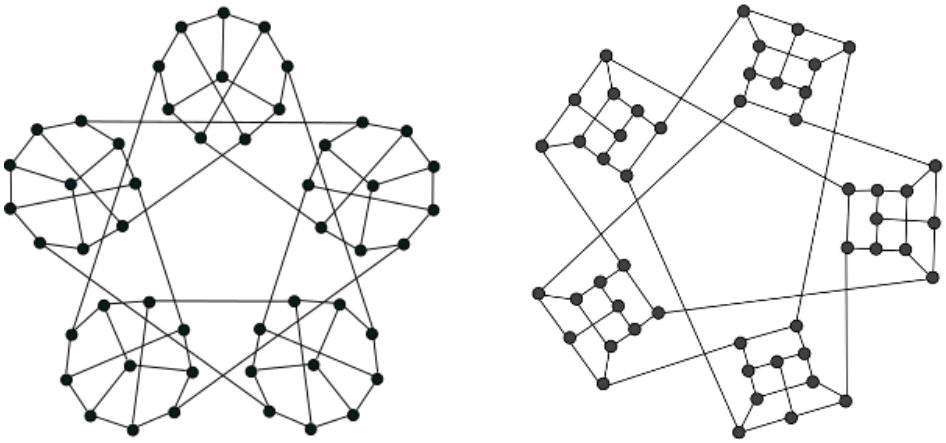
„Ďalší snark bol objavený Blanche Descartesom v roku 1948; má 210 vrcholov a bol zostrojený pridávaním 'hviezdicovej konfigurácie' ku každej hrane Petersenovho grafu.“[20]
Hviezdicová konfigurácia spočíva v tom, že sa hrana nahradí podgrafom obsahujúcim štrnásť vrcholov a vizuálna reprezentácia tohto podgrafu má podobu hviezdy, tak ako je znázornené na obrázku 2.3. Blanche Descartes je pseudonym, ktorý používala štvorica matematikov, medzi ktorými boli aj Rowland Brooks a William Tutte, ktorí patria medzi významných vedcov v teórií grafov.



Obr. 2.3: Výrez z Descartesovho snarku[15]- ukážka hviezdicovej konfigurácie

2.3.4 Szekeresov snark a Watkinsov snark

V roku 1973 zostrojil austrálsky matematik maďarského pôvodu Szekeres nový snark spojením piatich Petersenových grafov. Skonštruovaný graf má 50 vrcholov a 75 hrán. Vhodným spôsobom je možné rozšíriť Szekeresov snark na nekonečnú triedu pridaním ľubovoľného množstva Petersenových grafov. V roku 1989 objavil John Watkins snark, ktorý vznikol rovnako ako Szekeresov snark spojením piatich Petersenových grafov a má 50 vrcholov a 75 hrán.



Obr. 2.4: Szekeresov snark[17] a Watkinsov snark[18] - oba vzniknú spojením piatich Petersenovych grafov

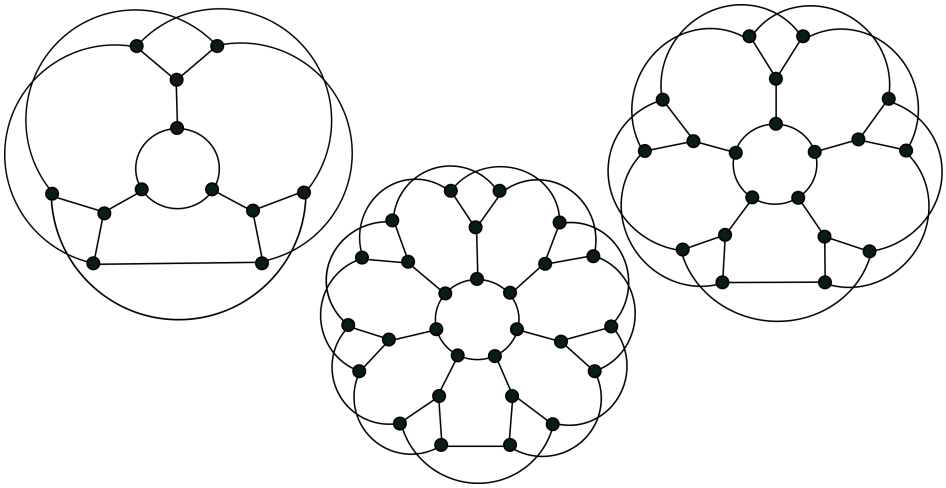
2.3.5 Flower snarky

„Pomalé a pracné objavovanie snarkov prešlo dramatickou zmenou, keď bola objavená nekonečná trieda snarkov, E.Grinbergom v roku 1972, a nezávisle na ňom aj v roku 1974 R.Isaacsom “[20]

Americký matematik Rufus Isaacs objavil v roku 1974 nekonečnú triedu snarkov, ktorá sa nazýva flower snarks. Názov nesie podľa vizuálnej reprezentácie, ktorá pripomína kvet a základ tvorí n-uholník v ktorom sa na každý vrchol sa napojí trojica

vrcholov a navzájom sa tieto trojice prepoja.

Isaacs objavil aj druhú nekonečnú triedu snarkov. Pomocou konštrukčnej operácie zvanej 4-*súčin* zovšeobecnil metódu, ktorú použil Blanuša na konštrukciu svojich snarkov. Táto trieda sa nazýva Blanuša-Descartes-Szekereš, skrátene označovaná aj BDS a ako napovedá názov, patria do nej aj Blanušove snarky, Descartesov snark a Szekerešov snark.



Obr. 2.5: Flower snarky[16] - pripomínajú pohľad na kvet

2.4 Vlastnosti snarkov

Pre množstvo snarkov nás budú zaujímať aj iné vlastnosti ako sú tie, ktoré bude preverovať nami vytvorený program. Jedná sa najmä o nasledujúce štyri vlastnosti, pre ktoré už sú hotové programy, ktoré ich overujú.

Kritický snark je taký snark G, v ktorom pre ľubovoľnú dvojicu susedných vrcholov $u, v \in V$ platí, že podgraf $G - u, v$ je hranovo 3-zafarbiteľný.

Bikritický snark je taký snark G, v ktorom pre ľubovoľnú dvojicu vrcholov $u, v \in V$ kde $u \neq v$, platí, že podgraf $G - u, v$ je hranovo 3-zafarbiteľný.

Bikritickosť je silnejšia vlastnosť ako kritickosť a preto bude platiť, že bikritické snarky sú podmnožinou kritických snarkov.

k-Redukcia $\exists k$ hrán v snarku G, ktoré keď sa prerežú, tak vzniknú dva komponenty. Pokiaľ nie je jeden z týchto komponentov hranovo 3-zafarbiteľný, môže sa k polhrán v tomto komponente navzájom poprepájať, prípadne ak k je nepárne, tak pridať jeden vrchol a poprepájať a takto vzniknutý kubický graf bude snark a bude sa nazývať $k-$redukciou snarku G. Ak platí, že $|G'| < |G|$, kde G' je $k-$redukcia snarku, tak táto redukcia sa označuje ako vlastná redukcia.

Ireducibilita Pokiaľ snark G nemá vlastnú $k-$redukciu pre všetky $k < m$, označuje sa ako $m-$ireducibilný. Pokiaľ je snark $m-$ireducibilný pre $\forall m \geq 1$, označuje sa zjednodušene ako ireducibilný.

Teoréma 1. Ak je graf G snark, potom

1. Pre $1 \leq k \leq 4$ platí G je k-ireducibilný snark $\iff$ G je cyklicky $k-$súvislý snark
2. Pre $5 \leq k \leq 6$ platí G je k-ireducibilný snark $\iff$ G je kritický snark
3. Pre $k \geq 7$ platí G je k-ireducibilný snark $\iff$ G je bikritický snark

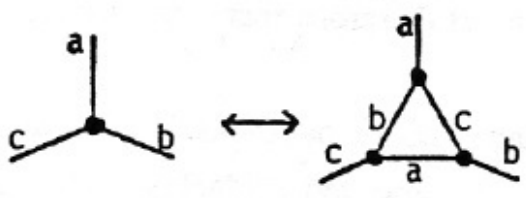
Dôsledok tejto teorémy je, že G je ireducibilný vtedy a len vtedy ak je G bikritický.[10]

2.5 Konštrukcie snarkov

Existuje niekoľko spôsobov ako konštruovať nové snarky. Na tieto konštrukcie treba mať na vstupe jeden alebo viacero snarkov a tie sa následne modifikujú a vznikne z nich nový snark.

2.5.1 Trojuholníková substitúcia

Je to najjednoduchší spôsob ako generovať nové snarky. Pre existujúci snark $G = (V, E)$ sa vezme ľubovoľný vrchol z grafu G a substituuje sa za trojicou vzájomne prepojených vrcholov.[21]Na obrázku 2.5.1 je znázornené, ako treba nastaviť novým hranám správne farbenie, aby bolo zachované, že $\chi'(G) = 4$. Pri trojuholníkovej substitúcii však vzniká v grafe cyklus dĺžky tri a tým sa mení aj obvod grafu. Takto vzniknutý snark s obvodom obvod ≤ 4 je triviálny snark a keďže v tejto práci sa venujeme iba netriviálnym snarkom, tak túto metódu na konštruovanie snarkov nebudeme používať.
Tento spôsob sa dá použiť aj opačným smerom, keď dochádza k redukcii cyklu dĺžky tri na jediný vrchol čím sa môže z triviálneho snarku skonštruovať netriviálny snark.



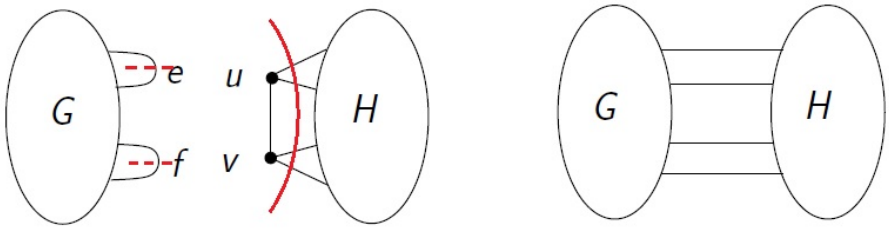
Obr. 2.6: Trojuholníková súbstitúcia[21] - generuje triviálne snarky

2.5.2 4-Súčin (dot product)

4-Súčin, po anglicky dot product, je metóda, ktorú objavil vyššie spomínaný Rufus Isaacs a vytvoril pomocou nej triedu snarkov BDS. 4-*Súčin* sa podrobne venoval John Watkins vo svojich publikáciách *Snarks*[20] a *A Survey of Snarks*[21].
Na túto metódu treba dva snarky $G = (V_G, E_G)$ a $H = (V_H, E_H)$ pričom nie je vylúčené, že $G = H$.

Definícia 2. V snarku G sa vyberú dve ľubovoľné nesusedné hrany $e = ab$ a $f = cd$ a v snarku H sa vyberú ľubovoľné dva susedné vrcholy u a v. Nech a',b' a v sú susedné vrcholy u a nech c',d' a u sú susedné vrcholy v. Následne sa zo snarku G odstránia hrany e a f a zo snarku H vrcholy u a v. Po odstránení sa spoja vrcholy a s a', b s b', c s c' a d s d'. Takto vzniknutý graf je 4-súčinom $G{\cdot}H$.

4-*Súčin* sa štandardne značí ako $I = G{\cdot}H$, kde I je skonštruovaný snark, ktorý vznikne aplikovaním metódy *dot product* na snarky G a H. Pre novovzniknutý snark $I = (V_I, E_I)$ platí, že $|I| = |G|+|H|-2$ a $||I|| = ||G||+||H||-3$ a jedná sa o netriviálny snark.
Je zjavné ako Isaacs vytvoril pomocou tejto metódy nekonečnú triedu netriviálnych snarkov, vzhľadom k tomu, že sa môže 4-*súčin* použiť opäť na novovzniknutý snark, či už so sebou samým, alebo s už známym snarkom, napríklad aj tým z ktorého bol vytvorený.



Obr. 2.7: Ukážka generovania snarkov pomocou 4-súčinu

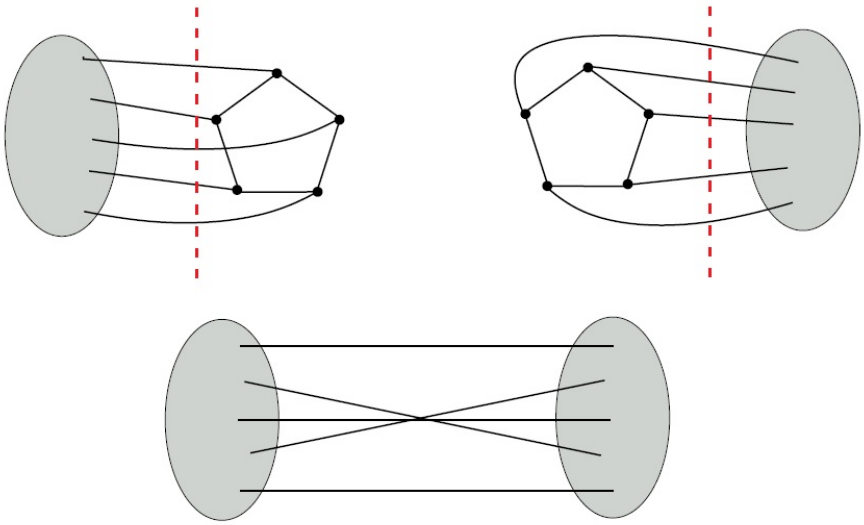
2.5.3 5-Súčin (star product)

5-Súčin, po anglicky 5-product alebo star product, je ďalšia metóda pomocou ktorej je možné generovať nové snarky. Na 5-súčin treba dva grafy $G = (V_G, E_G)$ a $H = (V_H, E_H)$, avšak na rozdiel od 4-súčinu stačí, aby bol iba jeden z týchto grafov snark. Oba tieto grafy však musia obsahovať cyklus dĺžky päť.

Definícia 3. V snarku G a snarku H sa vybere ľubovoľný cyklus C a C' dĺžky 5. Odstránením týchto cyklov vznikne v oboch grafoch päť vrcholov stupňa 2. Nech sú to v snarku G vrcholy a,b,c,d a e a v snarku H vrcholy a',b',c',d' a e'. Prepojením vrcholov

a s a', b s b', c s c', d s d' a e s e' dostaneme nový snark, ktorý je 5-súčinom snarkov G a H.

Na označenie 5-súčinu sa používa zápis $I = G \star H$ a hovorí sa, že I je 5-súčinom grafov G a H. Graf $I = (V_I, E_I)$ je netriviálny snark a platí pre neho, že $|I| = |G| + |H| - 10$ a počet hrán je $||I|| = ||G|| + ||H|| - 15$. Pokiaľ sú oba grafy cyklicky 5-súvislé, tak aj výsledný graf je cyklicky 5-súvislý.



Obr. 2.8: Ukážka generovania snarkov pomocou 5-súčinu

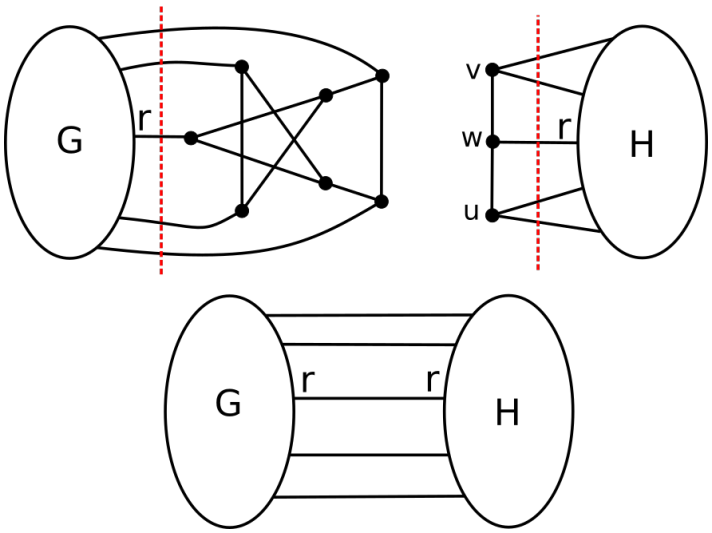
2.5.4 Negátorová substitúcia

Negátor sú tri vrcholy u,w a v a ich incidentné hrany, pre ktoré existuje cesta uwv dĺžky 2.

Petersenovský negátor je podgraf, ktorý vznikne odstránením negátora z Petersenovho grafu. Petersenovský negátor je tvorený dvoma cyklami dĺžky päť. Nech sú tieto cykly C_1 a C_2, existujú práve dve hrany e a f, pre ktoré platí, že $e, f \in C_1$ a zároveň $e, f \in C_2$, pričom existuje vrchol v s ktorým sú hrana e aj hrana f incidentné. Päť vrcholov v tomto podgrafe má hrany idúce von z tohto podgrafu, ktoré ho spájajú so zbytkom grafu. Jeden z týchto vrcholov je práve ten, ktorý je incidentný aj s hranou e aj s hranou f. Nech je tretia hrana incidentná s týmto vrcholom hrana r, takúto polhranu budeme označovať ako reziduálna polhrana. Medzi zvyšnými vrcholmi, ktorých incidentná hrana ide von z Petersenovského negátora existuje automorfizmus. Petersenovský negátor má sedem vrcholov a osem hrán. Viac o Petersenovskom negátore je v kapitole 4.

Na túto metódu treba dva snarky $G = (V_G, E_G)$ a $H = (V_H, E_H)$, pričom graf G musí obsahovať Petersenovsky negátor ako podgraf.

Definícia 4. Z grafu G sa odstráni Petersenovský negátor N, čo spôsobí, že v $G - N$ ostane päť polhrán. Následne sa z grafu H odstráni negátor uwv. Keďže sa jedná o netriviálne snarky, tak u a v nemôžu byť susedné a táto trojica vrcholov má iba jednu možnú reprezentáciu. Odstránením tejto trojice ostane v grafe H päť polhrán. Polhrana, ktorá ostane v grafe H a jej koncový vrchol bol stredný vrchol z cesty uwv sa bude označovať ako reziduálna polhrana podgrafu $H - N'$ kde N' je negátor. Podgrafy $G - N$ a $H - N'$ a polhrany v nich sa prepoja tak, aby sa reziduálne polhrany v $G - N$ a $H - N'$ prepojili navzájom a $e_{GH} = \{v_G, v_H\}$, kde $v_G \in V_G$ a $v_H \in V_H$.



Obr. 2.9: Ukážka generovania snarkov pomocou negátorovej substitúcie

Pokiaľ by sa nedodržalo prepojenie reziduálnych polhrán, tak by výsledný graf nebol snark, keďže by bol hranovo 3-zafarbiteľný.[4]
Graf $I = (V_I, E_I)$, ktorý je výsledkom *negátorovej substitúcie* grafov G a H je netriviálny snark, ktorého počet vrcholov je $|I| = |G| + |H| - 10$ a počet hrán je $||I|| = ||G|| + ||H|| - 15$. Okrem toho platí, že pokiaľ pre oba snarky H a G $\lambda_C = 5$, tak negátorová substitúcia rovnako ako 5-súčin zachováva cyklickú súvislosť a výsledný graf I bude mať tiež $\lambda_C = 5$.

Kapitola 3

Permutačné snarky

V tejto diplomovej práci sa sústredíme na skúmanie jednej dôležitej triedy snarkov, a to permutačných snarkov. Táto kapitola je venovaná práve tejto triede snarkov a ich konštrukciám.

Definícia 5. Snark, ktorý je tvorený dvoma disjunktnými cyklami rovnakej dĺžky, medzi ktorými existuje perfektné párovanie sa nazýva permutačný snark.

Tvrdenie 1. Snark, ktorý má 2-faktor tvorený dvoma bezchordovými cyklami sa nazýva permutačný snark.

Ľahko si všimnúť, že definícia 5 a tvrdenie 1 tvrdia to isté. Medzi permutačné snarky patrí napríklad Petersenov graf alebo aj oba Blanušove grafy popísané v kapitole 2. Pomocou zovšeobecnenia Petersenovho grafu boli permutačné snarky v roku 1967 zadefinované.[23]

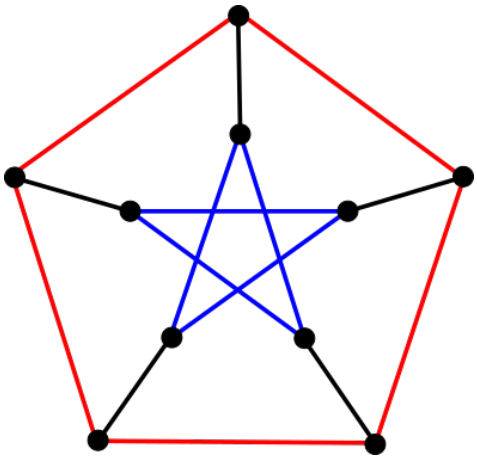
3.1 Permutačný snark

Definícia 6. Nech G je permutačný snark, ktorý má permutačný 2-faktor C skladajúci sa z dvoch cyklov C_1 a C_2. Potom $\{C_1, C_2\}$ budeme označovať ako permutačný 2-faktor a $(G, \{C_1, C_2\})$ bude bicyklická reprezentácia snarku G.

Snark môže obsahovať viacero rôznych permutačných 2-faktorov určujúcich určujúcich bicyklickú reprezentáciu snarku G. Hrany tvoriace perfektné párovanie medzi cyklami permutačného 2-faktoru budú označované ako priečky.

Definícia 7. Nech G je permutačný snark a $(G, \{C_1, C_2\})$ a $(G, \{D_1, D_2\})$ sú dve rôzne bicyklické reprezentácie grafu G, potom $(G, \{C_1, C_2\}) \cong (G, \{D_1, D_2\})$, ak $\exists$ automorfizmus φ grafu G taký, že $\varphi(\{C_1, C_2\{) = \{D_1, D_2\}$.

Petersenov graf je permutačný snark a je to aj najmenší netriviálny snark, ide teda o najmenší permutačný snark. Obsahuje až šesť rôznych bicyklických reprezentácií, pričom všetky sú navzájom izomorfné.



Obr. 3.1: Petersenov graf s permutačnými cyklami označenými červenou a modrou farbou

3.1.1 Vlastnosti permutačných snarkov

Z definície permutačného snarku vyplývajú ďalšie vlastnosti, ktoré nemusia byť na prvý pohľad zjavné.

Tvrdenie 2. Dĺžka permutačných cyklov v permutačnom snarku $G = (V, E)$ je $|G|/2$

Dôkaz 1. Definícia faktoru určuje, že permutačný 2−faktor respektíve permutačné cykly pokrývajú všetky vrcholy v permutačnom snarku $G = (V, E)$. Z definíce 5 a tvrdenia 1 vyplýva, že cykly permutačného 2-faktoru sú bezchordové a existuje medzi nimi perfektné párovanie. Preto musí platiť, že permutačné cykly musia mať rovnakú dĺžku, pričom dĺžka týchto cyklov musí byť $|G|/2$.

Tvrdenie 3. Pre každý permutačný snark $G = (V, E)$ platí $|G| \equiv 2 \pmod 4$

Dôkaz 2. Nech $G = (V, E)$ je permutačný snark a $|G| \equiv 2 \pmod 8$. Vzhľadom k tomu, že počet vrcholov v grafe G je násobok čísla štyri vyplýva z tvrdenia 2, že dĺžka permutačných cyklov by v grafe G bola párna. V takom prípade by nám na ofarbenie hrán oboch permutačných cyklov stačili dve farby a, b, pričom by sa hrany cyklov ofarbili striedavo $a - b - a - ... - b$. Na ofarbenie všetkých priečok by nám stačila jedna farba c a celý snark G by sa dal ofarbiť iba troma farbami, čo je spor s definíciou snarku. Preto musí byť $|G| \equiv 2 \pmod 4$

Doposiaľ všetky permutačné snarky, pre ktoré profesor Škoviera pretestoval bikritickosť túto vlastnosť spĺňali a je preto možné, že plaí, že

$$\text{permutačné snarky} \subseteq \text{ireducibilné snarky}$$

3.1.2 Skonštruované permutačné snarky

Do rádu 34 existuje dokopy 10838 permutačných snarkov. Iba 13 z nich má cyklickú súvislosť $\lambda_C \geq 5$.[19]

Rád grafu	#snarkov	#permutačných	#perm. $\lambda_C \geq 5$
10	1	1	1
14	0	0	0
18	2	2	0
22	31	0	0
26	1297	64	0
30	139854	0	0
34	25286953	10771	12

Tabuľka 3.1: Počty snarkov a permutačných snarkov [19]

Dôsledok 1. Pre všetky $n \geq 0$ existuje permutačný snark rádu $24n + 10$, ktorého cyklická súvislosť je 5

Z tabuľky 7.3.2 vidieť, že permutačné snarky tvoria iba malé percento zo všetkých snarkov. Ide však o zaujímavú triedu snarkov, keďže permutačné snarky súvisia s niekoľkými dôležitými hypotézami ako napríklad hypotéza *cycle double cover*. Tak isto je zaujímavé si všimnúť, že permutačné snarky do rádu 34 existujú len také, že $|G| \equiv 2 \pmod 8$. Aj keď tvrdenie 3 nevylučuje existenciu permutačných snarkov rádu $|G| \equiv 6 \pmod 8$, nebol takýto permutačný snark doposiaľ objavený a existuje predpoklad, že ani neexistuje. Tak isto existuje hypotéza o obvode 5, ktorá tvrdí, že všetky permutačné snarky majú obvod päť. Je známe, že neexistujú permutačné snarky, ktorých obvod by bol ≤ 4 a doposiaľ sa nepodarilo objaviť permutačný snark, ktorého obvod by bol ≥ 6, avšak nie je ani dokázané, že taký snark nemôže existovať.

Do rádu 34 existujú permutačné snarky, pre ktoré platí $\lambda_C \geq 5$ iba pre rády 10 a 34. S týmto súvisí veta, ktorú v roku 2012 dokázala dvojica matematikov Hägglund a Hoffman-Ostenhof, že pre každé $n \geq 0$ existuje permutačný cyklicky 5-súvislý snark rádu $24n + 10$.[6]

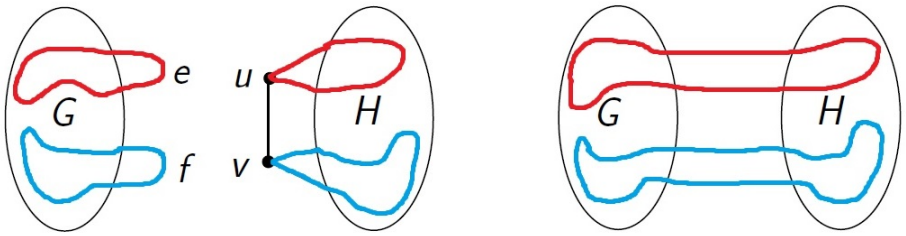
3.2 Konštrukcia permutačných snarkov

Podobne ako pre všeobecné snarky aj pre permutačné snarky existujú metódy ako ich konštruovať. Niektoré spôsoby konštrukciu snarkov ukázané v kapitole 2 sa môžu použiť aj na konštrukciu permutačných snarkov. Platí však, že vstupné grafy v prípade konštrukcie permutačných snarkov musia byť takisto permutačné snarky.

3.2.1 4-Súčin

Nech G aj H sú permutačné snarky a ich permutačné cykly sú C_{G1}, C_{G2}, C_{H1} a C_{H2}, kde $C_{G1}, C_{G2} \subset G$ a $C_{H1}, C_{H2} \subset H$. Konštrukcia bude vychádzať z 2.5.2, avšak vybrané hrany e_{G1}, e_{G2} nemôžu byť priečky a každá musí byť z iného permutačného cyklu a hrana e_H musí byť priečka medzi C_{H1} a C_{H2}. Dva zo štyroch vrcholov stupňa dva, ktoré zostanú po odstránení hrán e_{G1} a e_{G2} sú z permutačného cyklu C_{G1} a zvyšné dva sú z C_{G2}. Toto je určené tým, že e_{G1} ani e_{G2} nie sú priečky a patria do rôznych permutačných cyklov. Tým pádom oba koncové vrcholy hrany patria do rovnakého permutačného cyklu. Rovnako tak odstránenie koncových vrcholov hrany e_H z grafu H spôsobí, že dva zo štyroch vrcholov stupňa dva budú patriť do C_{H1} a zvyšné dva do C_{H2}. Keďže $e_H = \{v_{H1}, v_{H2}\}$ je priečka, koncové vrcholy patria do rôznych permutačných cyklov, tým pádom zvyšné dve hrany vychádzajúce z v_{H1} nie sú priečky a patria spolu s druhým koncovým vrcholom do rovnakého permutačného cyklu ako v_{H1}. Rovnako to platí aj pre v_{H2}. Následne treba tieto dva podgrafy, prepojiť tak, aby dvojica polhrán, ktorá patrila do rovnakého permutačného cyklu grafu G bola napojená na dvojicu polhrán, ktorá je z rovnakého permutačného cyklu grafu H.

Profesor Škoviera spolu s docentkou Máčajovou vyslovili tvrdenie, že každý snark, ktorý má permutačný 2−faktor a jeho $\lambda_C = 4$ je 4-súčinom dvoch permutačných snarkov.[9]



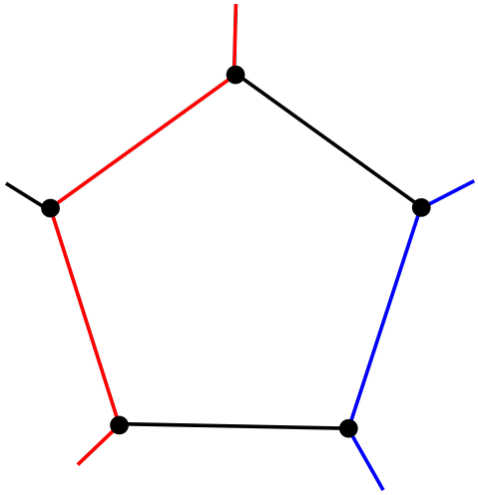
Obr. 3.2: Generovanie permutačných snarkov pomocou 54-súčinu

3.2.2 5-Súčin

Lema 1. Pokiaľ v permutačnom snarku G existuje cyklus dĺžky 5, je presne dané, že dve susedné hrany patria do permutačného cyklu C_{G1}, jedna hrana patrí do permutačného cyklu C_{G2} a zvyšné dve hrany sú priečky permutačných cyklov.

Toto je dané tým, že pokiaľ by boli menej ako dve hrany priečky - či jedna alebo žiadna, musel by existovať vrchol, v ktorom by sa stretli oba permutačné cykly, čo však nemôže nastať. Rovnako tak, ak by viac ako dve hrany boli priečky, musel by existovať vrchol, ktorý by bol incidentný s dvoma priečkami. Keďže priečky nemôžu byť susedné hrany, ich vzdialenosť v cykle dĺžky 5 musí byť 1 a ich koncové vrcholy sú

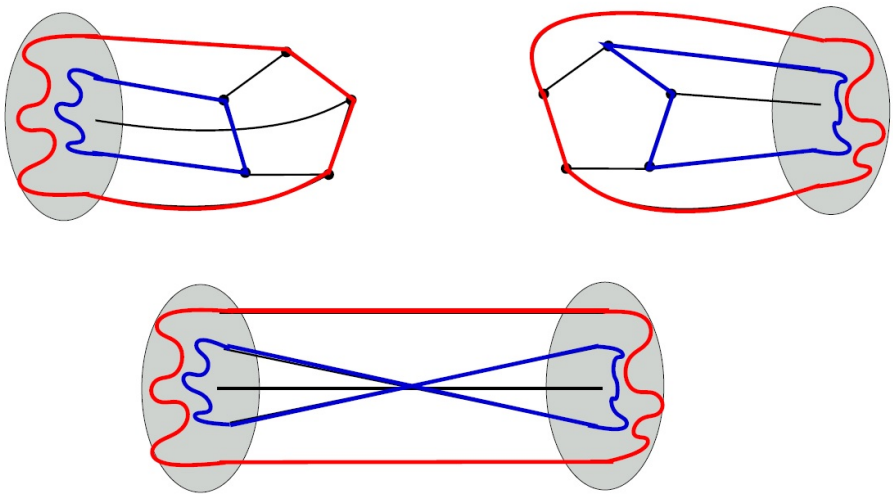
z rôznych permutačných cyklov. Dve hrany, ktoré sú z rovnakého permutačného cyklu musia byť susedné hrany, v opačnom prípade by existovala priečka medzi vrcholmi z rovnakého permutačného cyklu, alebo by cez jeden vrchol prechádzali oba permutačné cykly. Toto platí pre všetky cykly dĺžky 5 a permutačný 2−faktor až na jeden prípad. Jediná výnimka je Petersenov graf, kde dĺžka permutačného cyklu je 5 a teda všetky hrany patria do rovnakého cyklu, tak ako je to znázornené na obrázku 3.1. Avšak pre ľubovoľný pár permutačných cyklov v Petersenovom graf platí, že zvyšných desať cyklov má pokrytie permutačnými cyklami také, ako je vyššie popísané a znázornené na obrázku 3.2.2.



Obr. 3.3: Prechádzanie permutačného 2-faktoru cez 5-cyklus. Jeden cyklus permutačného 2-faktoru je označený červenou farbou a druhý modrou

Dôsledok 2. Dva susedné vrcholy z cyklu dĺžky 5 patria do jedného cyklu permutačného 2-faktoru. Zvyšné tri vrcholy patria do druhého cyklu permutačného 2-faktoru.

5-**Súčin** na generovanie permutačných snarkov vychádza z 2.5.3. Nech $G \star H$ je snark s permutačným 2−faktorom, potom G aj H musia byť permutačné snarky a ich permutačné cykly sú C_{G1}, C_{G2}, C_{H1} a C_{H2}, kde $C_{G1}, C_{G2} \subset G$ a $C_{H1}, C_{H2} \subset H$. Z oboch grafov odstránime cykly dĺžky 5. Vieme, že pre oba cykly je pokrytie permutačným 2−faktorom dané z definície 1. Po odstránení oboch týchto cyklov zostane v oboch grafoch päť polhrán, pričom dve sú z jedného permutačného cyklu, dve z druhého permutačného cyklu a posledná je priečka. Odstránením cyklu sa jeden permutačný cyklus skrátil o dva vrcholy a druhý o tri, toto platí pre oba grafy. Keďže permutačné cykly výsledného grafu musia mať rovnakú dĺžku, je potrebné, aby sa prepojil kratší pozostatok permutačného cyklu z G s dlhším pozostatkom permutačného cyklu z H a rovnako tak dlhší pozostatok z G prepojiť s dlhším pozostatkom z H. Polhrany priečok v oboch grafoch musia byť prepojené navzájom.



Obr. 3.4: Generovanie permutačných snarkov 5-súčinom

Teoréma 2. Nech G a H sú permutačné snarky, pričom pre oba platí $\lambda_C = 5$. Potom $G \star H$ je tiež permutačný snark a $\lambda_C(G \star H) = 5$.

Aj táto teoréma je od profesora Škovieru a docentky Máčajovej [9] a z tejto teorémy vyplýva aj dôsledok 1, ktorý zosilňuje vetu od Hägglunda a Hoffman-Ostenhofa.

Sú známe cyklicky 5-súvislé permutačné snarky rádu 34 a 5-súčin zachováva cyklickú súvislosť, pričom pre snark $I = G \star H$ platí, $|I| = |G| + |H| - 10$. Je očividné, že pokiaľ sa za H zoberie ľubovoľný snark z týchto snarkov rádu 34, tak výsledný graf bude mať o 24 vrcholov viac ako G. Pokiaľ sa novovzniknutý graf I zasubstituuje ako G a opäť sa urobí $G \star H$, je možné vytvoriť nekonečne veľa cyklicky 5-súvislých permutačných snarkov, pričom v každom kroku bude mať výsledný graf o 24 vrcholov viac ako predchádzajúci.

Teoréma 3. Existuje cyklicky 5-súvislý permutačný snark rádu $8n + 2$, pre všetky $n \geq 4$

Aj táto teoréma pochádza od profesora Škovieru a docentky Máčajovej a jej dôkaz, ako aj úplný dôkaz teorémy 2 bol prezentovaný na Bratislavskom seminári z teórie grafov[9] dňa 5.11.2015.

3.2.3 Negátorová substitúcia

Generovanie permutačných snarkov pomocou *negatorovej substitúcie* rovnako ako predošlé dva prípady vychádza z tej konštrukcie, ktorá je vysvetlené v 2.5.4, pričom treba dodržať správne napojenie permutačných snarkov a polhrán z grafu G na permutačné snarky a polhrany z H, aby novovzniknutý snark bol permutačný snark.

Lema 2. Nech G je permutačný snark obsahujúci petersenovský negator N ako svoj podgraf, potom jeden z permutačných cyklov musí prechádzať cez reziduálnu hranu a pokrývať tri vrcholy z N. Druhý permutačný cyklus pokrýva štyri hrany.

Lema aj úplný dôkaz pochádza z článku *Permutation Snarks*[4].

Z automorfizmu Petersenovho grafu sa dá odvodiť, že všetky polhrany okrem reziduálnej polhrany v petersenovskom negatore sú izomorfné. Reziduálna polhrana patrí do jedného z cyklov permutačného 2-faktoru, tak jedna zo zvyšných štyroch hrán musí do neho patriť tiež. Ďalšie dve hrany patria do druhého cyklu permutačného 2-faktoru, tieto hrany budeme nazývať ťažký konektor. Zvyšná hrana je priečka.

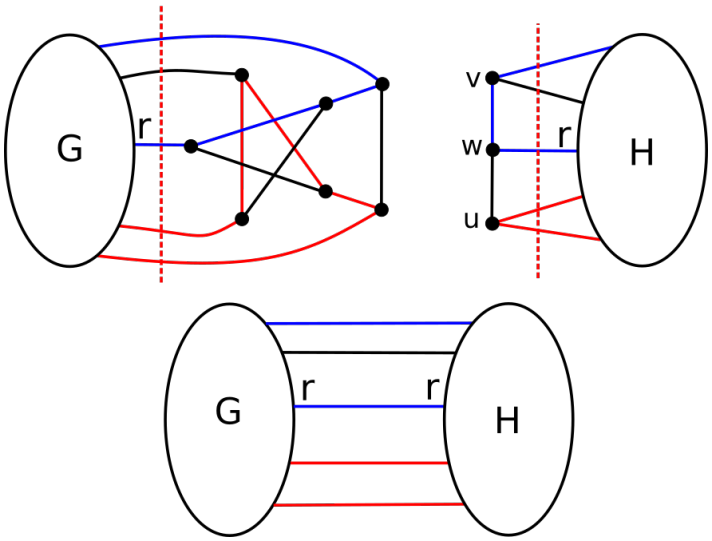
Lema 3. Je presne dané ako prechádzajú cykly permutačného 2-faktoru cez negátor a že reziduálna hrana patrí do jedného z cyklov permutačného 2-faktoru.

Nech H je permutačný snark a C_{H1} a C_{H2} sú permutačné cykly. BUNV hrana $e = u, w$ je priečka a zvyšné dve hrany incidentné s u rovnako ako aj u patria do permutačného cyklu C_{G1}. Hrana $e = w, v$ spolu s vrcholmi w a v patrí do C_{G2}. Reziduálna hrana idúca z vrcholu w musí patriť tiež do C_{G2}, keďže w už je incidentné s jednou priečkou. v je teda incidentné s dvoma hranami z C_{G2} a jednou priečkou, pričom priečka nemôže ísť z v do w. Nech je takýto vrchol u označený ako ťažký konektor a v ľahký konektor.

Následne treba po odstránení petersenovského negatora N z grafu G a negatora N' z grafu H dodržať, aby prepojenie bolo nasledovné

1. Reziduálna polhrana z G sa musí napojiť na reziduálnu polhranu z H
2. Polhrany ťažkého konektora z G sa napoja na polhrany ťažkého konektora z H
3. Polhrany priečok z G a H sa navzájom prepoja
4. Zvyšná hrana z G sa napojí na zvyšnú hranu z H

Tento postup nedefinuje jednoznačne novovzniknutý permutačný snark, keďže v kroku 2 pri napájaní ťažkých konektorov existujú dve možnosti ako polhrany prepojiť.



Obr. 3.5: Ukážka generovania permutačných snarkov pomocou negatorovej substitúcie

Teoréma 4. Každý graf vytvorený pomocou negatorovej substitúcie permutačných snarkov G a H je permutačný snark. Tak isto pokiaľ sú G aj H cyklicky 5-súvislé, aj výsledný graf je cyklicky 5-súvislý.

Teoréma 5. Existuje permutačný snark s cyklickou súvislosťou 5, rádu n, kde $n(mod)2 = 8$, pre každé $n \geq 34$.

Dôkaz týchto teorém ako aj úplný dôkaz negátorovej substitúcie sa nachádza v článku od profesora Škovieru a docentky Máčajovej *Permutation Snarks*[4]

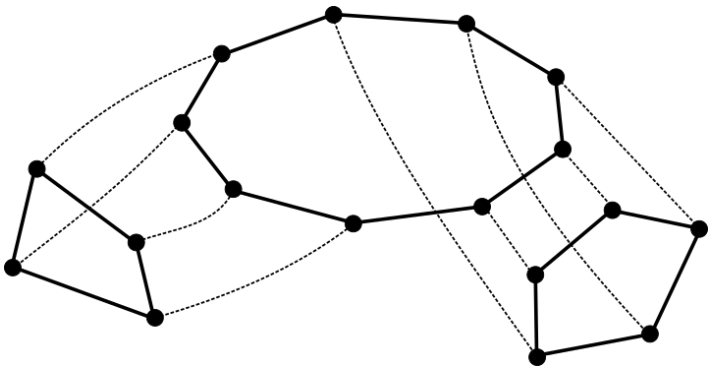
3.3 Polopermutačné snarky

V súvislosti s permutačnými snarkami budeme skúmať ďalšiu triedu snarkov a to polopermutačné snarky.

Definícia 8. Snark G je polopermutačný, ak obsahuje 2-faktor pozostávajúci z indukovaných cyklov, z ktorých jeden má dĺžku $|G|/2$.

Táto definícia je slabšia ako definícia permutačných snarkov a teda bude platiť, že permutačné snarky $\subseteq$ polopermutačné snarky. Pri polopermutačných snarkoch nie je definované, či takto vzniknutý podgraf musí byť súvislý a v prípadoch keď snark bude polopermutačný ale nebude permutačný bude platiť, že podgraf $G - C$ je nesúvislý graf. Je zrejmé, že ak by bol súvislý a nebol by permutačný vznikol by spor, keďže by obsahoval permutačný 2−faktor a zároveň by nebol permutačný. Program najprv zisťuje, či je snark permutačný a iba ak nie je permutačný skúma slabšiu podmienku polopermutačnosti. V prípade polopermutačných snarkov, ktoré nie sú permutačné sa jedná o množinu cyklov, kde jeden cyklus C má dĺžku $|G|/2$ a práve jeden z koncových

vrcholov všetkých priečok patrí do C. Obrázok 7.1 je iba ilustračný obrázok, ako by vyzeral polopermutačný 2−faktor. Nejedná sa však o snark.



Obr. 3.6: Ilustračná ukážka, ako by vyzeral polopermutačný 2-faktor

S použitím nášho programu ukážeme, že polopermutačné snarky maju výrazne odlišné vlastnosti od permutačných snarkov.

Kapitola 4

Klastre

Pre lepšie porozumenie štruktúry snarkov je dôležité vedieť, aké podgrafy dané snarky obsahujú a hľadať súvislosti medzi týmito podgrafmi a vlastnosťami snarkov. Zameranie našej diplomovej práce je na netriviálne snarky a každý takýto snark má obvod aspoň 5. Netriviálnych snarkov s obvodom 6 je veľmi málo. Do rádu 38 je takýchto snarkov iba 42[19], čo je spomedzi netriviálnych snarkov do rádu 38 mizivé percento, pretože ich existuje dokopy vyše sto miliónov a doposiaľ všetky známe permutačné snarky majú obvod 5 a tým pádom obsahujú aspoň jeden cyklus dĺžky 5. Cykly dĺžky 5 preto zohrávajú dôležitú úlohu pri pochopení štruktúry snarkov ako aj permutačných snarkov. Aj preto je zameranie programu na podgrafy, ktoré súvisia s cyklami dĺžky 5. Pod označením 5-cyklus sa myslí cyklus dĺžky 5.

Definícia 9. Maximálny súvislý podgraf G', ktorého každá hrana patrí do cyklu dĺžky 5 prislúchajúcemu grafu G sa nazýva klaster 5-cyklov.

V tejto kapitole sa bude miesto klaster $5-$cyklov používať zjednodušené označenie - klaster (zhluk). V prípade klastrov 5-cyklov sa však nejedná o indukované podgrafy.

4.1 Klastre Petersenovho grafu

Definícia 10. Petersenovský klaster v kubickom grafe G je taký podgraf Petersenovho grafu, kde každá hrana je obsiahnutá v cykle dĺžky 5.

Aj medzi klastrami sa nachádzajú také, ktoré sa v snarkoch vyskytujú častejšie. Ako je spomenuté v 2.3.1, Petersenov graf môže byť označovaný za základný stavebný kameň snarkov a práve klastre Petersenovho grafu sú tie, ktoré sa v snarkoch vyskytujú veľmi často. Keďže obsahuje 12 cyklov dĺžky 5, tak obsahuje ako svoj podgraf viacero rôznych klastrov. Aj samotný Petersenov graf je klaster Petersenovho grafu, keďže každá jeho hrana patrí aspoň do jedného cyklu dĺžky 5, avšak je zrejme, že Petersenov graf sa nemôže vyskytovať ako podgraf v iných snarkoch, pretože by stupne vrcholov

daného snarku museli byť väčšie ako tri a preto budeme ako klastre Petersenovho grafu brať iba tie, ktoré sa môžu vyskytovať ako podgraf v inom grafe.

4.1.1 Konštrukcia klastrov Petersenovho grafu

Pre každý klaster, nie len petersenovské klastre platí, že počet polhrán tohto podgrafu je ≥ 4. Pokiaľ by tomu tak nebolo, znamenalo by to, že by klaster bol prepojený so zbytkom grafu maximálne troma hranami, avšak netriviálne snarky majú $\lambda \geq 4$. Po prerezaní troch hrán, ktoré prepájajú klaster so zbytkom grafu by vznikli dva komponenty čo by znamenalo spor s cyklickou súvislosťou.

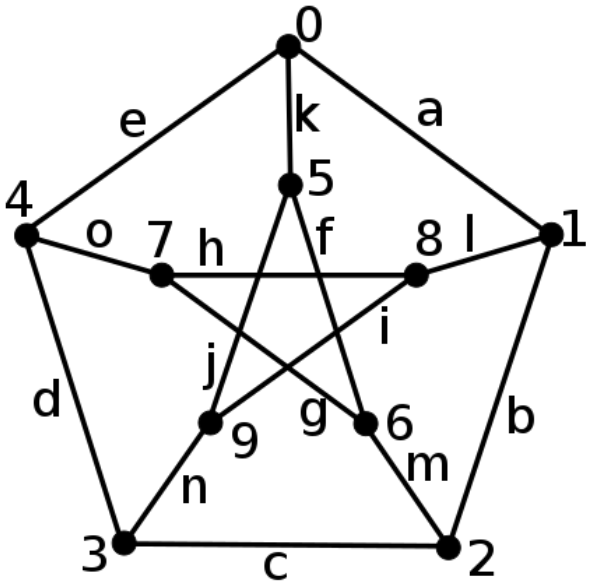
Klastrov Petersenovho grafu je dokopy deväť a líšia sa v počtoch vrcholov, počtoch hrán a polhrán a aj v tom, koľko cyklov dĺžky 5 obsahujú. Až na jednu dvojicu sa dá z trojice počet vrcholov, počet hrán a počet cyklov dĺžky 5 jednoznačne určiť o ktorý z deviatich petersenovských klastrov sa jedná. Samotná konštrukcia týchto klastrov z Petersenovho grafu je pomocou dvoch operácií.

- prerezanie hrany - táto operácie vytvorí dve visiace polhrany a tým zníži počet hrán, zníži počet cyklov dĺžky 5 a zachová počet vrcholov
- odstránenie vrcholu - vytvorí tri visiace polhrany, zníži počet cyklov a zníži aj počet vrcholov

Tieto operácie sa môžu ľubovoľne kombinovať pričom vždy musí byť splnená podmienka, že každá hrana patrí aspoň do jedného cyklu dĺžky 5. Aj prvotné generovanie množiny peteresenovských klastrov bolo pomocou iterovania cez Petersenov graf a už objavené klastre a následného aplikovania týchto operácii.

4.1.2 Popis a vizualizácia Petersenovských klastrov

Každý z deviatich petersenovských klastrov má svoje meno a je neizomorfný s ostatnými petersenovskými klastrami. Pre lepšie pochopenie týchto klastrov boli aj vizualizované, čo pomôže aj pri vizualizácií nových snarkov ak vieme aké petersenovské klastre obsahujú. Pri jednotlivých klastroch je napísane pomocou ktorých operácií vznikli, avšak platí, že tieto operácie sa nemôžu aplikovať ľubovoľne, ale majú určité pravidlá. Podrobnejšie sú tieto pravidlá pre jednotlivé klastre napísane v dôkaze 4.2

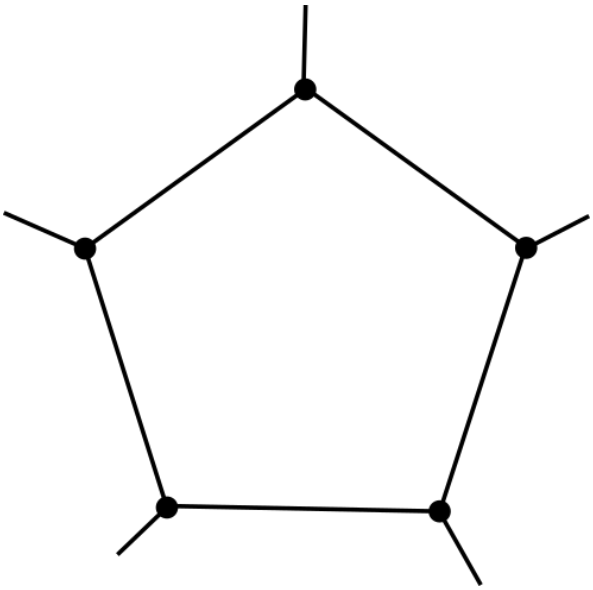


Obr. 4.1: Petersenov graf s označenými vrcholmi a hranami

Tento obrázok budeme používať ako referenciu pre popisovanie vzniku jednotlivých Petersenovských klastrov z Petersenovho grafu. Keď budeme pri popise klastrov spomínať čísla označujúce vrcholy a písmená označujúce hrany bude sa myslieť označenie práve na tomto obrázku.

Pentagón

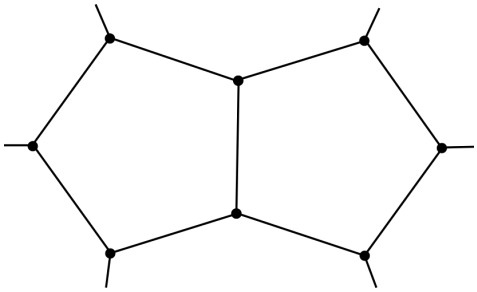
Najmenší a najjednoduchší petersenovský klaster je `pentagon`, ktorý sa skladá z jedného cyklu dĺžky 5 a z Petersenovho grafu sa získa odstránením ľubovoľného z dvanástich 5-cyklov v Petersenovom grafe - napríklad cyklu 5-6-7-8-9. Má päť vrcholov, päť hrán, päť visiacich polhrán a obsahuje jeden 5-cyklus.



Obr. 4.2: Pentagón

Dvojitý Pentagón

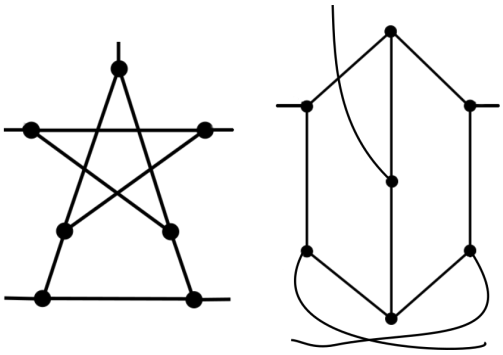
`Dvojitý pentagón` je klaster, ktorý vznikná z Petersenovho grafu pomocou odstránenia dvoch vrcholov a prerezaním jednej hrany. `Dvojitý pentagón` je klaster, ktorý sa skladá z dvoch 5-cyklov, ktoré majú spoločnú hrany. Vznikne napríklad odstránením vrcholov 7 a 8 a prerezaním hrany k. `Dvojitý pentagón` má osem vrcholov, deväť hrán, šesť visiacich polhrán a dva 5-cykly.



Obr. 4.3: Dvojitý Pentagón

Dyád

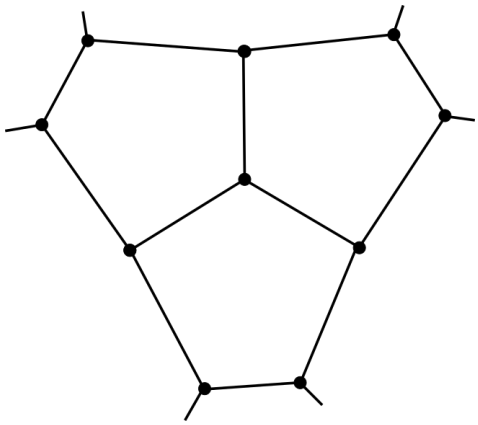
`Dyád` je klaster, ktorý vznikne odstránením troch vrcholov z Petersenovho grafu. Tak isto vznikne aj spojením dvoch 5-cyklov, ktoré majú dve spoločné hrany. Z Petersenovho grafu vznikne napríklad odstránením vrcholov 4,0 a 1. `Dyád` má sedem vrcholov, osem hrán, päť visiacich polhrán a obsahuje dva cykly dĺžky 5.



Obr. 4.4: Dyád - dve možné reprezentácie dyádu

Trojitý Pentagón

Petersenovský klaster `trojitý pentagón` vznikná prerezaním troch hrán v Petersenovom grafe. Názov má podľa toho, že sa jedná o tri spojené `pentagóny`, kde každá dvojica `pentagónov` zdieľa jednu hranu. Platí, že všetky tri hrany, ktoré sú v dvoch `pentagónoch` sú incidentné s rovnakým vrcholom a teda existuje vrchol, ktorý patrí do každého z troch 5-cyklov. Z Petersenovho grafu vzniká prerezaním hrán g,k,l. Má desať vrcholov, dvanásť hrán, šesť visiacich polhrán a tri cykly dĺžky 5.

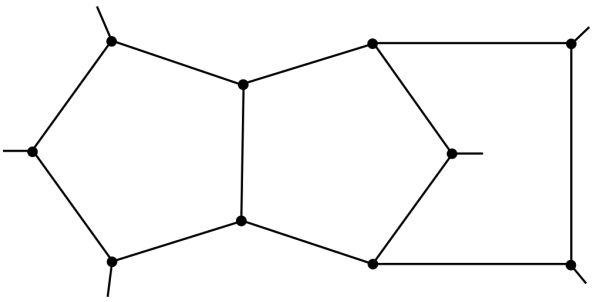


Obr. 4.5: Trojitý Pentagón

Trojbunka

Klaster `trojbunka` sa z Petersenovho grafu dostane rovnako ako `trojitý pentagón`, čiže prerezaním troch hrán. Podľa toho ktoré tri hrany sa prerežú vznikne `trojbunka` alebo `trojitý pentagón`. Práve `trojbunka` a `trojitý pentagón` sú tá dvojica petersenovských klastrov, kde sa nedá jednoznačne určiť z trojice počet vrcholov, počet hrán a počet 5-cyklov o ktorý snark sa jedna. Rozdiel je v nich v tom, že v `trojitý pentagón` zdieľa každá dvojica 5-cyklov jednu hranu a existuje vrchol, ktorý je v každom z týchto cyklov. `Trojbunka` toto nespĺňa a dva z cyklov dĺžky 5 nemajú žiadnu

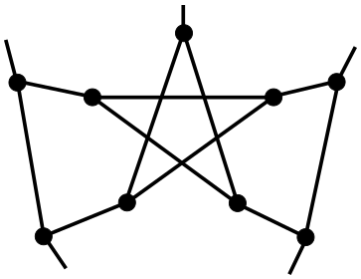
spoločnú hranu. Z Petersenovho grafu ho dostaneme napríklad prerezaním hrán k,l,m. Tento klaster má desať vrcholov, dvanásť hrán, šesť visiacich polhrán a tri 5-cykly.



Obr. 4.6: Trojbunka

Triáda

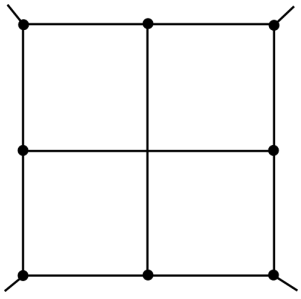
`Triáda` je Petersenovský klaster, ktorý vznikne keď sa z Petersenovho grafu odstráni jeden vrchol a jedna hrana sa prereže vzniká `triáda`. Z Petersenovho grafu ho dostaneme napríklad odstránením vrcholu 0 a prerezaním hrany c. Tento klaster má deväť vrcholov, jedenásť hrán, päť visiacich polhrán a tri 5-cykly.



Obr. 4.7: Triáda

Isochróm

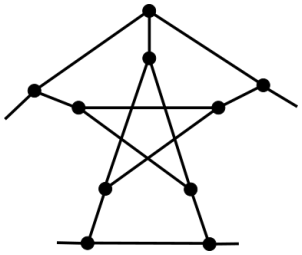
Petersenovsky klaster `isochróm` dostaneme odstránením dvoch vrcholov z Petersenovho grafu. Získame ho napríklad odstránením vrcholov 0 a 5 z Petersenovho grafu. Tento klaster má osem vrcholov, desať hrán, štyri visiace polhrany a štyri cykly dĺžky 5.



Obr. 4.8: Isochróm

Heterochróm-1

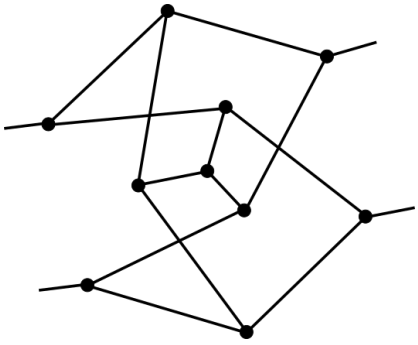
Klaster `heterochróm-1` sa z Petersenovho grafu dostane prerezaním dvoch hrán, ktoré sú vo vzdialenosti 1. Môžu to byť napríklad hrany k a m v Petersenovom grafe. Má desať vrcholov, trinásť hrán, štyri visiace polhrany a obsahuje päť cyklov dĺžky 5.



Obr. 4.9: Heterochróm-1

Heterochróm-2

Petersenovský klaster `heterochróm-2` vznikne rovnako ako `heterochróm-1` avšak prerezané hrany budú vo vzdialenosti 2. Tento rozdiel spôsobí, že sa budú líšiť tieto snarky v počte 5-cyklov a ničom inom. Jedná sa napríklad o hraný k a c v Petersenovom grafe. `Heterochróm-2` má teda desať vrcholov, trinásť hrán, štyri visiace polhrany a štyri 5-cykly.

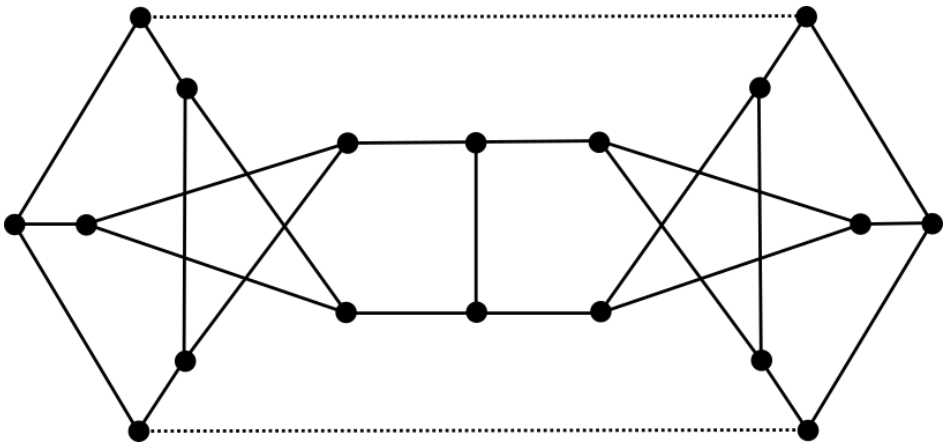


Obr. 4.10: Heterochróm-1

4.1.3 Nepetersenovské klastre

Okrem množiny Petersenovských klastrov, existujú aj nepetersenovské klastre. Jedná sa o klastre, ktoré sa nevyskytujú ako podgraf Petersenovho grafu - je to najmä z dôvodu, že ide o klastre, ktoré maju viac ako 10 vrcholov. Nebola zatiaľ zistená spojitosť medzi výskytom nepetersenovských klastov a petersenovských klastov v jednom snarku. Existujú prípady, kedy graf obsahuje iba nepetersenovské klastre prípadne iba jeden takýto klaster. V niektorých prípadoch existujú snarky, ktorých všetky vrcholy patria do jedného nepetersenovského klastra. Najmenší takýto prípad je prvý Blanušov

snark, v ktorom všetky hrany okrem dvoch patria do nejakého cyklu dĺžky 5 a všetky jeho vrcholy sú v týchto cykloch. Je to teda aj najmenší snark, ktorý obsahuje iba nepetersenovské klastre.

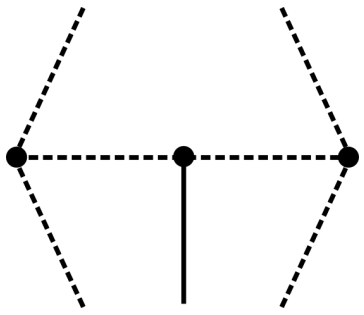


Obr. 4.11: Prvý Blanušov snark je najmenší snark obsahujúci ibe nepetersenovský klaster a zároveň aj najmenší snark, ktorého všetky vrcholy sú v nepetersenovskom klastri

4.2 Dôkaz úplnosti množiny petersenovských klastrov

Lema 4. Pre ľubovoľné dva vrcholy v Petersenovom grafe existuje cesta, ktorej dĺžka je maximálne 2.

Lema 5. Pokiaľ sa z Petersenovho grafu G odstraňuje dva a viac vrcholov, musia byť tieto vrcholy susedné. Pokiaľ by neboli susedné, tak je medzi nimi vrchol, ktorý podľa lemy 4 leží na ceste dĺžky 2. Odstránením koncových vrcholov tejto cesty by v grafe ostal voľne visiaci vrchol s jednou hranou, ktorý by neležal v žiadnom cykle dĺžky 5.



Obr. 4.12: Odstránením dvoch nesusedných vrcholov z Petersenovho grafu by v podgrafe ostal visiaci vrchol

Ako už bolo spomínané v 4.1.1, generovanie Petersenských klastrov sa môže robiť pomocou dvoch operácií. Odstránením vrcholu, respektíve viacerých vrcholov, alebo

prerezaním hrany, respektíve viacerých hrán. Tieto dve operácie sa môžu ľubovoľne kombinovať.
Každý vygenerovaný podgraf však musí spĺňať nasledujúce dve podmienky, aby mohol byť klaster.

Lema 6. Každý klaster musí mať aspoň päť vrcholov

Toto je očividné, keďže pokiaľ by v podgrafe nebolo aspoň päť vrcholov, tak by tam nemohol existovať cyklus dĺžky 5

Lema 7. Každý klaster v netriviálnych snarkoch musí mať aspoň štyri visiace polhrany. Preto musí platiť vždy aspoň jedna z nasledujúcich možností

- Z Petersenovho grafu sa musia odstrániť aspoň dva vrcholy
- V Petersenovom grafe sa musia prerezať aspoň dve hrany
- Z Petersenovho grafu sa musí odstrániť aspoň jeden vrchol a prerezať aspoň jedna hrana

Pokiaľ nebude splnená žiadna z vyššie uvedených podmienok tak počet visiacich polhrán v podgrafe bude ≤ 3 a z 4.1.1 je dané, že takýto prípad nemôže nastať

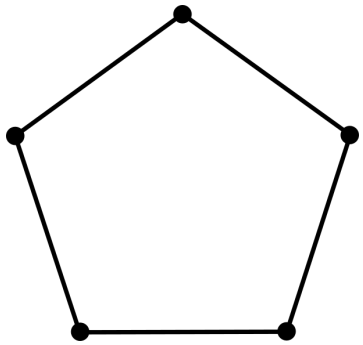
Tvrdenie 4. Množina petersenovských klastrov Pentagón, Dvojitý pentagón, Trojitý pentagón, Trojbunka, Triáda, Isochróm, Heterochróm-1 a Heterochróm-2 je úplná a každý Petersenovský klaster je izomorfný s niektorým prvkom tejto množiny.

Dôkaz

Tento dôkaz pojednáva všetky prípady odstránenia vrcholov a následného prerezávania hrán. Pri odstránení šiestich vrcholov by mal podgraf iba štyri vrcholy a nemohol by tam byť cyklus dĺžky 5. Najväčšie množstvo vrcholov na odstránenie ktoré pripadá v úvahu je teda päť.

4.2.1 Odstránenie piatich vrcholov

Z lemy 5 vyplýva, že vrcholy, ktoré sa odstránia musia byť susedné. Zo symetrie Petersenovho grafu vyplýva, že nie je jednoznačne dané, ktorých päť vrcholov treba odstrániť, avšak treba dodržať, že pre jeden z dvanástich 5-cyklov Petersenovho grafu platí, že obsahuje všetkých päť odstránených vrcholov. Odstránením týchto vrcholov respektíve jedného celého 5-cyklu ostane v podgrafe päť vrcholov, ktoré budú tvoriť najmenší klaster - `pentagon`. Takýto podgraf má päť hrán, päť vrcholov a päť visiacich polhrán.



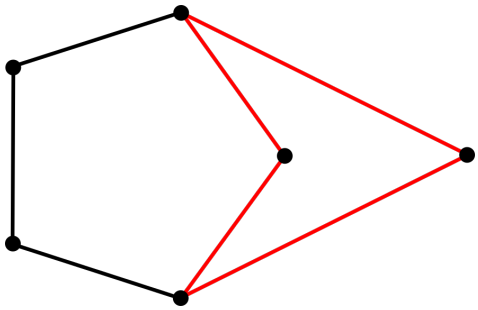
Obr. 4.13: Odstránením piatich vrcholov vznikne `pentagón`

Prerezanie hrany

V podgrafe, ktorý vznikol odstránením piatich vrcholov sa už žiadna hrana nemôže prerezať, keďže takýto podgraf obsahuje päť hrán a prerezaním ľubovoľnej z nich by nebolo možné, aby tam existoval cyklus dĺžky 5.

4.2.2 Odstránenie štyroch vrcholov

Na odstránenie štyroch vrcholov sa dá pozrieť z opačnej strany a to tak, že k `pentagónu` sa napojí jeden vrchol v_i. Aby takéto napojenie vrcholu v_i vytvorilo nejaký nový 5-cyklus treba ho napojiť na cestu, ktorej dĺžka je 3 a dve hrany idúce z v_i vytvoria spolu s cestou dĺžky 3 nový 5-cyklus. Keďže celý cyklus pred napojením vrcholu mal dĺžku 5 a vybraním cesty dĺžky 3 by sa tento vrchol napojil aj na doplnok tejto cesty, ktorej dĺžka je 2, spôsobí to, že zároveň s cyklom dĺžky 5 vznikne aj ďalší cyklus, ktorého dĺžka by bola 4. To je zjavne spor, keďže v Petersenovom grafe neexistuje žiaden cyklus dĺžky 4.



Obr. 4.14: Pripojenie vrcholu k `pentagónu` vytvorí cyklus dĺžky 4 označený červenou farbou

4.2.3 Odstránenie troch vrcholov

Z lemy 5 je potrebné, aby boli v prípade odstraňovania troch vrcholov tieto vrcholy susedné. Zo symetrie Petersenovho grafu vyplýva, že pre ľubovoľnú susednú trojicu

vznikne rovnaký podgraf. Odstránením takýchto vrcholov vznikne podgraf, ktorý má:

- sedem vrcholov
- osem hrán
- dva cykly dĺžky 5
- päť visiacich polhrán

Tento podgraf je ekvivalentný s `dyádom` a teda je to Petersenovský klaster. Ešte treba preveriť prípady prerezania jednej prípadne viacerých hrán v `dyáde`.

Prerezanie jednej hrany

Z definície klastru Petersenovho grafu vychádza, že každá hrana patrí do nejakého cyklu dĺžky 5. V `dyáde` sú dva 5-cykly. Prerezanie ľubovoľnej hrany by aspoň jeden z týchto dvoch cyklov narušilo a zostal by maximálne jeden cyklus dĺžky 5. Tým pádom môže z `dyádu` vzniknúť iba `pentagón`, ku ktorému sú napojené dva voľné vrcholy, čiže by nevznikol žiaden nový klaster a ani tento samotný podgraf by nebol klaster, keďže by jeho dve hrany neboli v žiadnom 5-cykle. Samozrejme platí, že pokiaľ prerezanie jednej hrany v `dyáde` nevygeneruje žiaden nový klaster, tak aj prerezanie viacerých hrán taktiež nevytvorí žiaden ďalší klaster.

4.2.4 Odstránenie dvoch vrcholov

Odstránením dvoch vrcholov, ktoré musia byť susedné a bez prerezania hrany vznikne podgraf, ktorého každá hrana patrí do nejakého cyklu dĺžky 5. Tým pádom patrí aj medzi klastre Petersenovho grafu a jedná sa o klaster `isochróm`. Zo symetrie a lemy 5 vyplýva, že to platí pre ľubovoľnú susednú dvojicu vrcholov. Tento klaster má nasledovné parametre:

- osem vrcholov
- desať hrán
- štyri cykly dĺžky 5
- štyri visiace polhrany

`Isochróm` má podobu cyklu dĺžky 8, kde každý druhý vrchol má iba dvoch susedov. Zvyšné vrcholy sú spojené s protiľahlým vrcholom. A každá hrana patrí aspoň do dvoch cyklov dĺžky 5.

Prerezanie jednej hrany

Vo všeobecnosti platí, že prerezať sa môže iba hrana, ktorej oba koncové vrcholy sú incidentné s troma hranami. Pokiaľ by bol nejaký vrchol incidentný iba s dvoma hranami, tak prerezaním jednej z nich by nám ostal voľný vrchol. V `isochróme` sú takéto hrany iba dve. Keďže `isochróm` je symetrický, tak je jedno, ktorá z týchto dvoch hrán sa prereže, lebo v oboch prípadoch vzniknú navzájom izomorfné grafy. Takýto podgraf má:

- osem vrcholov
- deväť hrán
- dva cykly dĺžky 5
- šesť visiacich polhrán

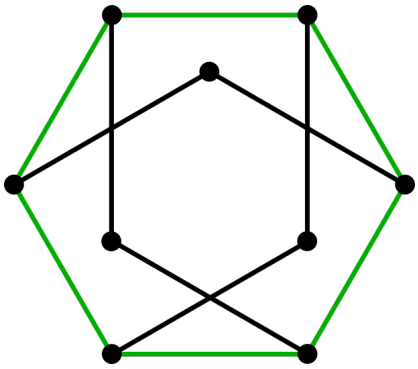
Konkrétne sa jedná o petersenovský klaster `dvojitý pentagón`.

Prerezanie dvoch hrán

`Dvojitý pentagón` má dva cykly dĺžky 5, pričom každá jeho hrana musí patriť aspoň do jedného z týchto cyklov. Prerezaním ľubovoľnej hrany by nastal rovnaký prípad ako v 4.2.3, kde pri odstránení troch vrcholov a následného prerezania hrany ostal iba jeden cyklus dĺžky päť a z takého podgrafu by mohol vzniknúť už iba `pentagón`.

4.2.5 Odstránenie jedného vrcholu

Z lemy 7 je dané, že iba odstránenie jedného vrcholu nestačí, a musí byť spolu s ním prerezaná aspoň jedna hrana. Samotným odstránením jedného vrcholu teda ešte klaster nevznikne. Zo symetrie Petersenovho grafu platí, že je jedno ktorý vrchol sa odstráni, vždy vznikne rovnaký podgraf. Konkrétne odstránením jedného vrcholu vznikne cyklus dĺžky 6, v ktorom sú všetky dvojice protiľahlých vrcholov spojené a na ich spojnici leží ešte jeden vrchol.



Obr. 4.15: Po odstránení jedného vrcholu z Petersenovho grafu ostane takýto podgraf - hrany označené zelenou farbou sa môžu prerezať

Prerezanie jednej hrany

Odstránením jedného vrcholu vznikne podgraf, ktorý má deväť vrcholov a dvanásť hrán. Tri z týchto vrcholov majú stupeň dva, čiže šesť hrán, ktoré sú incidentné s týmito vrcholmi sa nemôžu prerezať. Ostáva na výber zvyšných šesť hrán, pričom zo symetrie opäť vyplýva, že pre ľubovoľnú hranu sa dostanú navzájom izomorfné podgrafy. Takto vzniknutý podgraf má:

- deväť vrcholov
- jedenásť hrán
- tri cykly dĺžky 5
- päť visiacich polhrán

Tento podgraf spĺňa všetky podmienky klastru, keďže každá hrana patrí aspoň do jedného 5-cyklu a je to petersenovský klaster `triád`.

Prerezanie dvoch hrán

Prerezávať sa môžu iba hrany, ktorých koncové vrcholy sú incidentné s troma hranami. Takéto hrany sú v `triáde` iba tri. Avšak pre všetky tri tieto hrany platí, že patria do dvoch cyklov dĺžky 5 a podobne ako v 4.2.3 a 4.2.4 by ich prerezaním vznikol podgraf, ktorý by obsahoval iba jeden 5-cyklus a teda by z neho nemohol už vzniknúť iný podgraf ako `pentagón`.

4.2.6 Bez odstránenia vrcholu

Ostala už iba posledná možnosť a to, že sa z Petersenovho grafu neodstráni žiaden vrchol a budú sa iba prerezávať hrany. Z lemy 7 je určené, že to musia byť aspoň dve hrany.

Prerezanie dvoch hrán

Tieto hrany nemôžu byť susedné, pretože by nastal prípad, ktorý je popísaný v leme 5, keď by vrcholu, ktorý by bol incidentný s oboma týmito hranami ostala už iba jedna hrana a teda by sa jednalo o voľne visiaci vrchol a nemohol by patriť do žiadneho cyklu. Lema 4 určuje, že vzdialenosť medzi každými dvoma hranami v Petersenovom grafe je nanajvýš dva a preto pre nesusedné hrany sú dve možnosti aké hrany vybrať.

- vybrané hrany budú vo vzdialenosti 1
- vybrané hrany budú vo vzdialenosti 2

Zo symetrie Petersenovho grafu opäť vyplýva, že aj v jednom aj v druhom prípade je jedno, ktorá konkrétna dvojica sa vyberie, pre danú vzdialenosť bude vždy podgraf izomorfný s ostatnými. Avšak výber hrán vo vzdialenosti 1 vygeneruje iný podgraf ako výber hrán vo vzdialenosti 2 a teda nebudú izomorfné. Pokiaľ sú vybrané hrany vo vzdialenosti 1 vzniknutý podgraf má:

- desať vrcholov
- trinásť hrán
- päť cyklov dĺžky 5
- dve visiace polhrany

Jedná sa teda o klaster `heterochróm-1`. V prípade, že by boli vybrané hrany vo vzdialenosti 2, mal by tento podgraf rovnaký počet vrcholov a hrán a líšil by sa iba v počte cyklov dĺžky 5 - boli by tam štyri takéto cykly a bol by to teda klaster `heterochróm-2`.

Prerezanie 3 hrán

V článku *Six signed petersen graphs, and their automorphisms*[22] je dokázané, že sú štyri možnosti ako v Petersenovom grafe vybrať respektíve prerezať 3 nesusedné hrany, avšak iba jedna možnosť vygeneruje podgraf, ktorého všetky hrany budú v nejakom cykle dĺžky 5. Konkrétne je to prípad, keď sú všetky hrany vo vzájomnej vzdialenosti 1. Existujú však 2 možnosti ako z Petersenovho grafu vybrať tri hrany vo vzájomnej vzdialenosti 1. Najprv sa vyberú ľubovoľné dve hrany, ktoré sú vo vzdialenosti 1 a prerežú sa. Tieto dve hrany museli mať jednu susednú hranu spoločnú, táto hrana sa označí ako e. Teraz treba vybrať tretiu hranu a to sa dá dvoma spôsobmi. Buďto, že tretia hrana bude od hrany e vo vzdialenosti 1 alebo, že bude vo vzdialenosti 2. Pokiaľ je ich vzdialenosť 1 tak toto prerezanie vygeneruje `trojitý pentagón`. V prípade vzdialenosti 2 vznikne `trojbunka`. Pre oba tieto klastre platí, že majú:

- desať vrcholov
- dvanásť hrán
- tri cykly dĺžky 5
- šesť visiacich polhrán

Tieto dva klastre sa dajú dostať aj z klastrov, ktoré vznikli prerezaním dvoch hrán. `trojitý pentagón` vznikne prerezaním jednej hrany v `heterochróm-1` a `trojbunka` vznikne z `heterochróm-2`

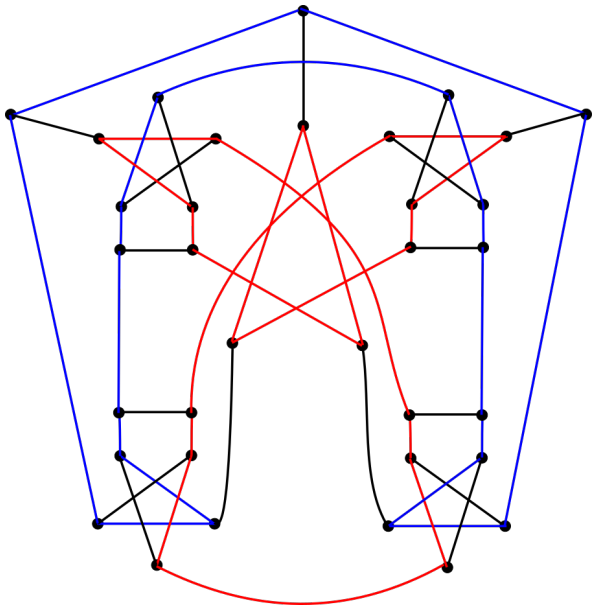
Prerezanie 4 hrán

Pokiaľ by sa z `trojbunky` alebo `trojitého pentagónu` odstránila ešte jedna hrana vznikol by buď podgraf, ktorý by mal iba jeden cyklus dĺžky 5 a rovnako ako v prípade 4.2.4 a 4.2.3 by z neho mohol vzniknúť už iba `pentagón`, alebo by ostali dva 5-cykly a v závislosti od toho, ktorá hrana by sa prerezala by tieto dva 5-cykly mali buď jednu alebo dve spoločné hrany a tým by vznikli `dyád` alebo `dvojitý pentagón` spolu s dvoma voľne visiacimi vrcholmi. A z takýchto podgrafov sa už nedá vygenerovať žiaden ďalší klaster okrem `dyádu` respektíve `dvojitého pentagónu` a z nich následne ešte `pentagón`.

Kapitola 5

Vizualizácia cyklicky 5-súvislých permutačných snarkov rádu 34

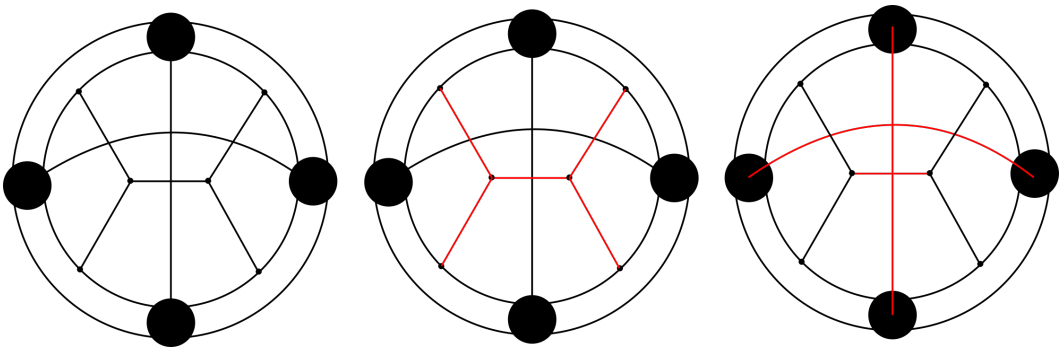
Dlho existovala hypotéza, že jediný cyklicky 5-súvislý permutačný snark je Petersenov graf. To sa zmenilo v roku 2012 po vygenerovaní všetkých snarkov do rádu 36, keď bolo objavených 12 cyklicky 5-súvislých permutačných snarkov rádu 34. Následne článok od Hägglunda a Ostenhofa sa dokázal, že cyklicky 5-súvislé permutačné snarky existujú pre každý rád $24n + 10$, kde $n \geq 0$. Táto podmienka je však pomerne slabá, pretože sú veľké skoky medzi jednotlivými rádmi, keďže najbližší rád po 34 je 58. V roku 2015 profesor Škoviera spolu s docentkou Máčajovou vytvorili cyklicky 5-súvislé permutačné snarky rádu 42 a 50. Následne pomocou týchto snarkov, cyklicky 5-súvislých permutačných snarkov rádu 34 a použitím *star productu* na ne dokázali, že existuje cyklicky 5-súvislý permutačný snark rádu $8n + 2$ pre všetky $n \geq 4$. Ako je spomenuté v 3.1.2 pre všetky doposiaľ známe permutačné snarky platí, že $(|G| mod 8) = 2$ a medzi rádmi 10 a 34 neexistuje žiaden cyklicky 5-súvislý permutačný snark. $8n + 2$ pre $n \geq 4$ teda pokrýva všetky rády vyššie ako 34, pre ktoré sú doposiaľ známe permutačné snarky.
Bez dvanástich snarkov rádu 34, ktoré sú permutačné a cyklicky 5-súvislé, by nebolo možné dokázať existenciu cyklicky 5-súvislých permutačných snarkov, bola súčasťou práce aj ich vizualizácia a lepšie pochopenie ich štruktúry. Všetkých týchto dvanásť snarkov si je veľmi podobných a ich konštrukcia pozostáva zo siedmich *negatorov* a šiestich vrcholov, ktoré nepatria do žiadneho klastra a líšia sa v tom, ako sú jednotlivé klastre a šesť vrcholov medzi sebou prepojené.



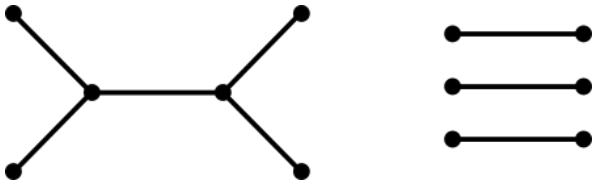
Obr. 5.1: Vyznačený permutačný 2-faktor v snarku PERM5.34-1

5.1 Vizualizácia PERM5.34

Týchto dvanásť snarkov bolo počas práce označovaných ako *PERM5.34* podľa názvu súboru, ktorý tieto snarky obsahoval. Na obrázku 5.1 je vizualizácia všetkých dvanástich snarkov z *PERM5.34*. Aj pomocou tejto vizualizácie sa podarilo zistiť, že sa dajú rozdeliť na dve skupiny po šiestich grafoch. Tieto skupiny sa líšia podľa toho, ako vyzerá podgraf, ktorý vznikne odstránením všetkých štyroch dyádov, ktoré sa nachádzajú v týchto permutačných snarkoch. Existujú práve dva podgrafy, ktoré môžu vzniknúť - typ 1 a typ 2. Typ 1 je súvislý podgraf, ktorý má päť hrán a šesť vrcholov. Typ 2 je nesúvislý a jedná sa o podgraf so šiestimi vrcholmi a 3 hranami.

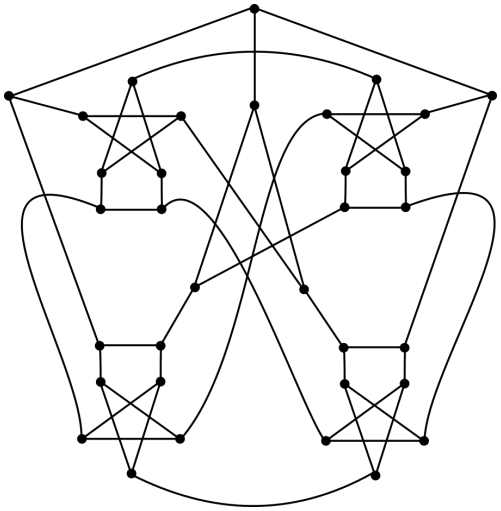


Obr. 5.2: Ukážka toho, ako vyzerajú permutačné snarky PERM5.34 aj s vyznačeným podgrafom typu 1 a 2. Veľké čierne body znázorňujú dyády

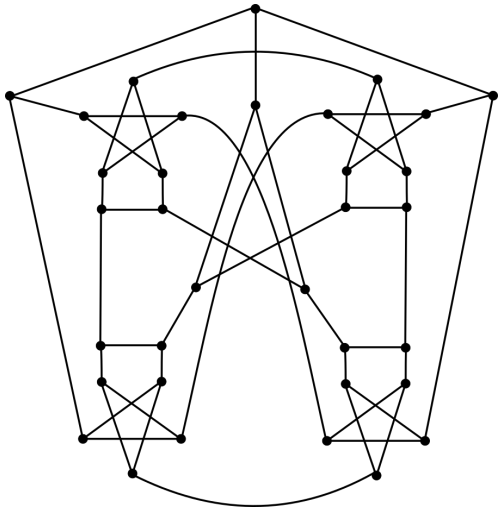


Obr. 5.3: Podgrafy, ktoré môžu vzniknúť odstránením dyád

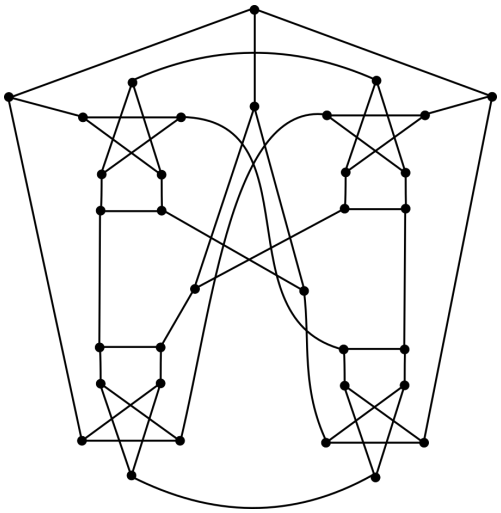
Odstránením dyádov z týchto grafoch môžeme dostať iba 2 rôzne podgrafy v závislosti od toho, ako sú tieto dyády navzájom prepojené. Na nasledujúcom obrázku sú zobrazené podľa toho, či sa jedná o permutačné snarky typu 1 alebo typu 2. Prvých šesť snarkov je typu 1 a ďalších šesť snarkov je typu 2.



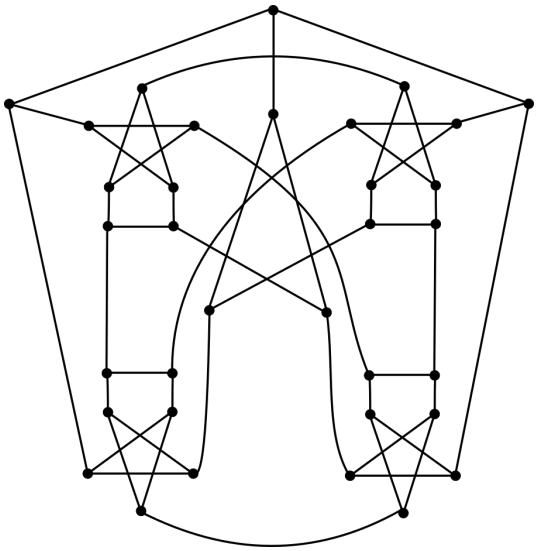
(a) PERM5.34-1



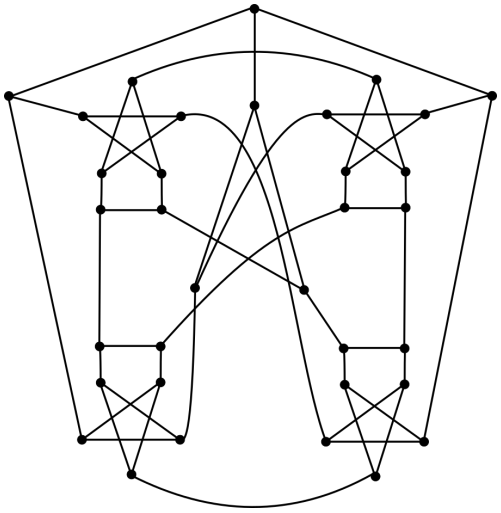
(b) PERM5.34-6



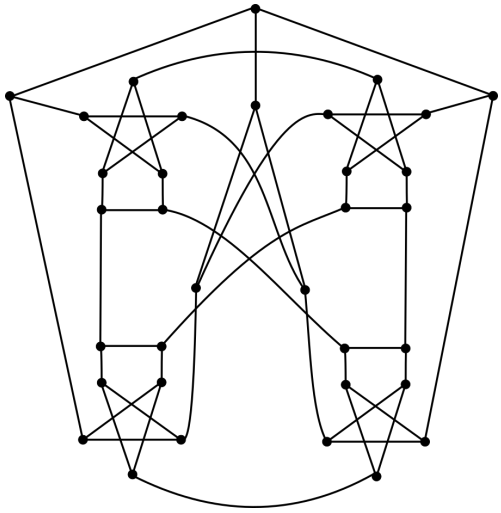
(c) PERM5.34-7



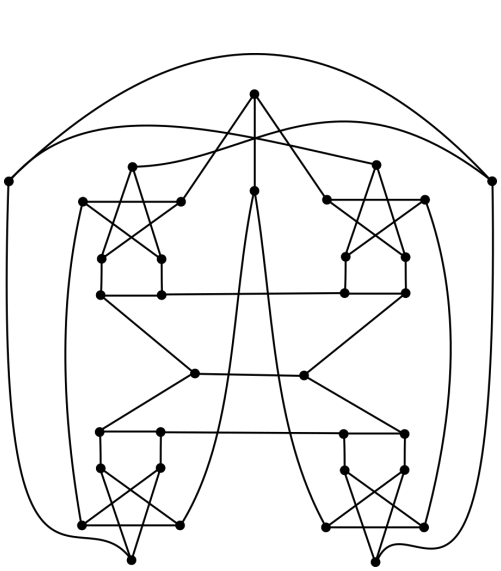
(d) PERM5.34-8



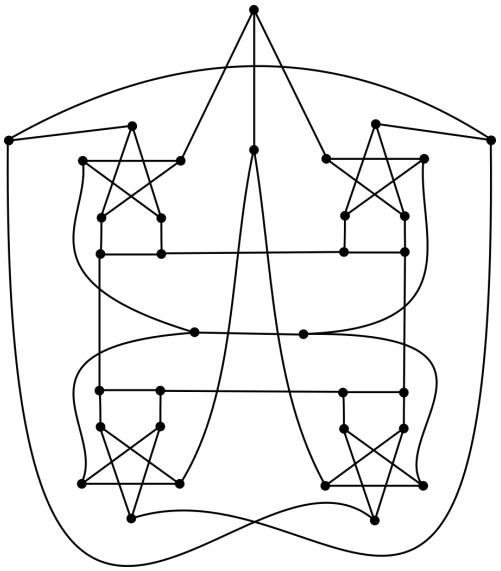
(e) PERM5.34-11



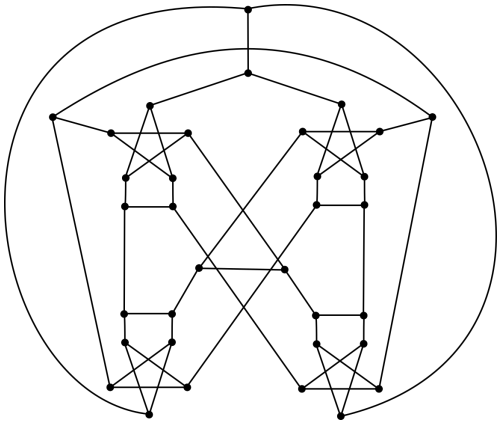
(f) PERM5.34-12



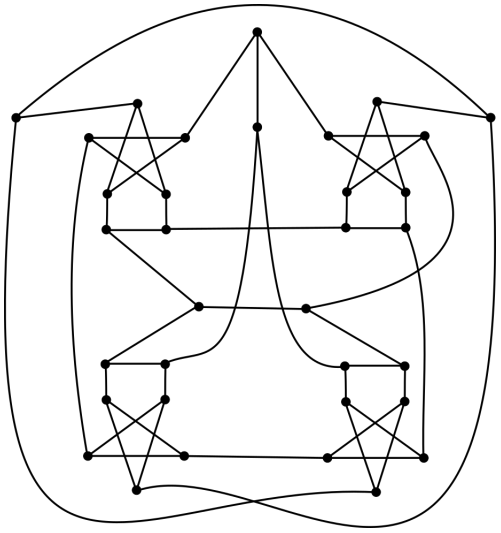
(g) PERM5.34-2



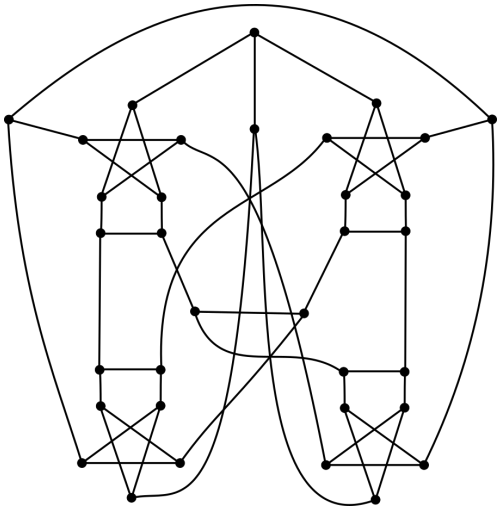
(h) PERM5.34-3



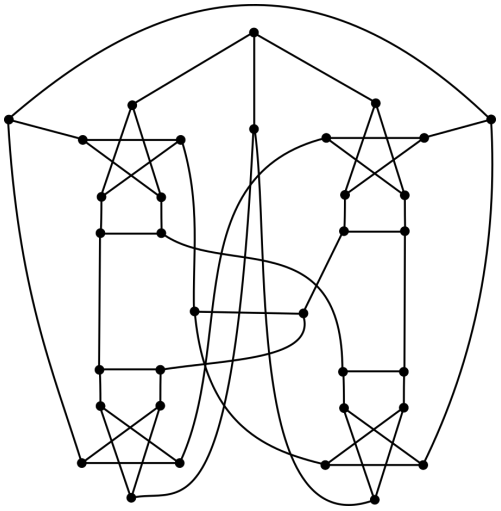
(i) PERM5.34-4



(j) PERM5.34-5



(k) PERM5.34-9



(l) PERM5.34-10

Obr. 5.4: Všetkých 12 cyklicky 5-súvislych permutačných snarkov rádu 34. Prvých šesť je typu 1 a ďalších šesť typu 2.

Kapitola 6

Skúmané vlastnosti

Táto kapitola je venovaná jednotlivým vlastnostiam, ktoré program dokáže skúmať pre snarky. Samozrejme vlastností, ktoré sa v snarkoch môžu overovať je množstvo a tie, ktoré zvládne program sú iba zlomok z nich, ale niektoré vlastnosti sú časovo náročné a na niektoré už existujú programy, ktoré ich zvládnu overovať.

6.1 Existujúce programy

Existuje niekoľko programov a knižníc, ktoré dokážu skúmať grafy a ich vlastnosti. Patrí medzi ne napríklad matematická knižnica *Sage* určená pre jazyk Python, ktorá dokáže napríklad vypočítať chromatický index grafu, ako aj mnohé ďalšie vlastnosti. Tak isto pre Python existuje knižnica *Nauty*, ktorá funguje dobre na overovanie izomorfizmu grafov. Profesor Škoviera má k dispozícii programy, ktoré dokážu zisťovať obvod grafu, cyklickú súvislosť grafu, hranovú 3-zafarbiteľnosť alebo bikritickosť snarku. Na niektoré vlastnosti však programy zatiaľ nie sú a aj to bol jeden z dôvodov, prečo sa overujú práve tieto vlastnosti.

6.2 Permutačnosť snarku

Veľká časť predchádzajúcich kapitol je venovaná permutačným snarkom a je preto zrejmé, že jedna z testovaných vlastností je práve zisťovanie, či sú snarky na vstupe permutačné alebo nie. Viacero ďalších vlastností môžu byť v snarku overené vtedy a len vtedy, pokiaľ je daný snark permutačný. Podrobnosti o tejto vlastnosti sa nachádzajú v kapitole 3, ktorá je celá venované práve permutačným snarkom.

6.3 Polopermutačnosť snarku

Pokiaľ vstupný snark nie je permutačný, tak sa overí, či spĺňa aspoň slabšiu podmienku, a to vlastnosť polopermutačnosti. Rovnako ako permutačnosť snarku aj polopermutačnosť je bližšie popísaná v kapitole 3.

6.4 Všetky izomorfné a neizomorfné reprezentácie permutačného 2-faktoru

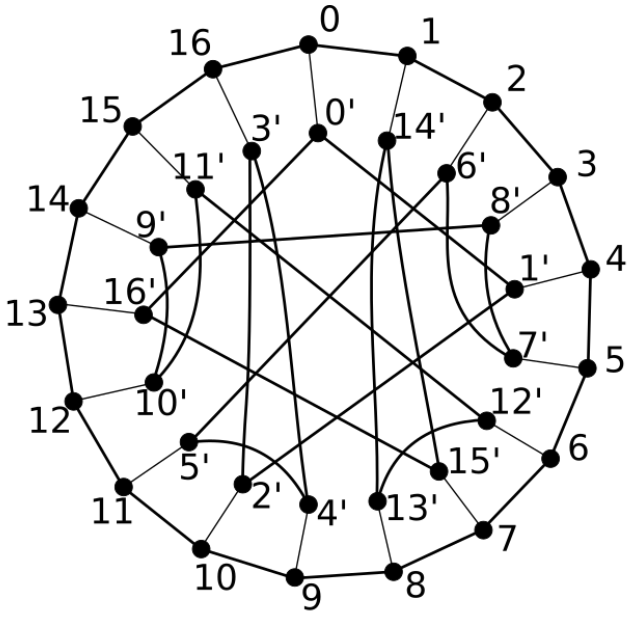
Pre permutačné snarky platí, že nemusia obsahovať iba jeden permutačný 2−faktor. Pokiaľ je teda vstupný snark permutačný, program ďalej zisťuje a určí všetky ďalšie permutačné 2−faktory. Následne po tom čo všetky permutačné reprezentácie grafu vyhľadá určuje, ktoré permutačné 2−faktory sú navzájom izomorfné a ktoré nie. Napríklad Petersenov graf obsahuje šesť rôznych permutačných 2−faktorov, avšak všetky páry permutačných cyklov sú navzájom izomorfné, čo vyplýva aj zo symetrie Petersenovho grafu.

6.5 Parita permutačných 2-faktorov

Táto vlastnosť platí opäť iba pre permutačné snarky. Hovorí viac o tom, ako sú navzájom permutačné 2−faktory prepojené priečkami. Zisťuje sa tak, že sa zoberie jeden z dvoch permutačných cyklov C_1 a vyberie sa jeden jeho vrchol v_i. Počnúc vrcholom v_i sa všetky vrcholy z C_1 postupne označia $0, 1, ..., n/2$ tak, ako idú postupne v cykle C_1. Potom sa zoberie druhý permutačný cyklus C_2, vyberie sa z neho vrchol, ktorý je susedný s v_i a vrcholy sa označia $0', 1', ..., n/2'$ v takom poradí, ako idú v cykle C_2. Na orientácii týchto cyklov ako aj ktorý permutačný 2−faktor sa vyberie nezáleží, pretože výsledná parita bude pre daný graf vždy rovnaká. Keď sú vrcholy označené, vytvoria sa permutácie cyklov.

Tie sa tvoria nasledovne - vyberie sa ľubovoľný vrchol z C_1, jeho označenie sa zapíše do permutácie a pozrie sa aké označenie má jeho sused z cyklu C_2 a vyberie sa jeho prislúchajúce označenie tohto suseda v cykle C_1. Pod prislúchajúcim označením sa myslí, že pokiaľ mal vrchol v C_2 označenie k', tak prislúchajúce označenie v cykle C_1 je vrchol s označením k. Označenie tohto vrcholu sa zapíše do permutácie a tento postup sa opakuje, dokým sa permutácia neuzavrie takým spôsobom, že by sa znova došlo na prvý vrchol z permutácie a následne by sa to iba zacyklilo. Následne sa to opakuje, dokým nie je každý vrchol práve v jeden permutácii. Následne sa na spočítanie parity permutačného 2−faktoru použije vzorec $parity = (\sum(perm - 1))\%2$, kde $perm$ sú jednotlivé dĺžky permutácii - jedna takáto permutácia bude označovaná ako

permutácia permutačného 2−faktoru.



(0) (1 14 9 4)(2 6 12 10)(3 8 13 16)(5 7 15 11)

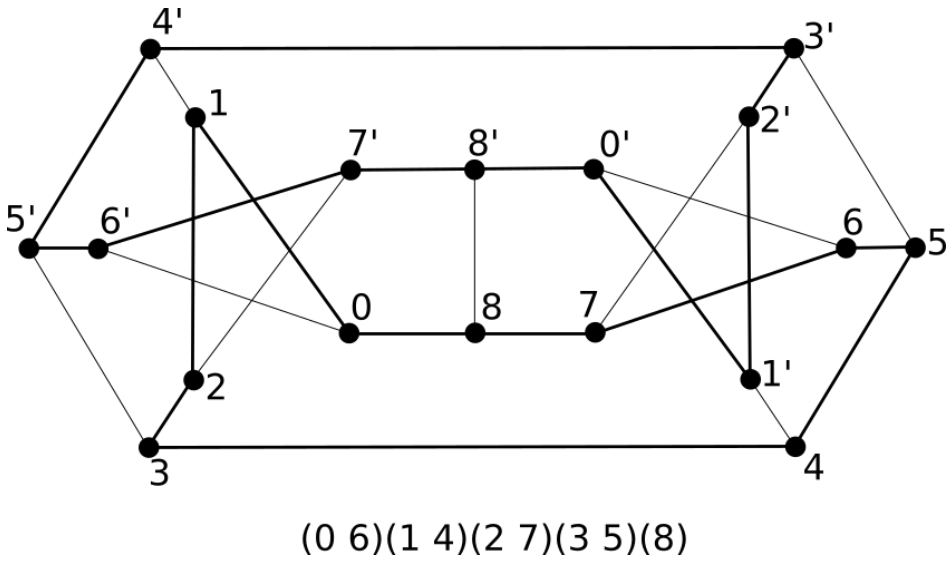
Obr. 6.1: Ukážka permutácie permutačného 2-faktoru pre permutačný snark rádu 32

6.6 Involúcia snarku

Táto vlastnosť súvisí s paritou.

Definícia 11. Graf G je involučný snark, pokiaľ existuje taký permutačný 2−faktor, kde dĺžka každej permutácie permutačného 2−faktoru je ≤ 2.

Najmenší involučný snark je Blanušov snark $B1$. Pre $B2$ táto vlastnosť neplatí. Na overenie involučnosti treba na rozdiel od parity prejsť všetky permutačné 2−faktory a orientácie týchto permutačných cyklov. Aj v prípade $B1$ sa involúcia preukáže iba pre jednu konkrétnu orientáciu permutačných cyklov.



Obr. 6.2: Ukážka involúcie v prvom Blanušovom snarku

6.7 Cycle double cover

Definícia 12. *Dvojité cyklové pokrytie grafu* je taká množina cyklov C, že každá hrana patriaca danému grafu sa nachádza práve v dvoch cykloch z C

Hypotéza 1. Každý hranovo dvojsúvislý graf má dvojité cyklové pokrytie.

Hypotéza 1 - *Cycle double cover conjecture* predpokladá, že každý bezmostový graf má dvojité cyklové pokrytie (CDC). Teda aj grafy, ktoré nie sú snarky, alebo dokonca ani kubické grafy. Z dôvodu, že táto práca je zameraná na snarky, program skúma špecifický prípad dvojitého cyklového pokrytia.

Definícia 13. Permutačné dvojité cyklové pokrytie grafu (permutačné CDC) je také dvojité cyklové pokrytie, kde množina cyklov C obsahuje cykly jedného permutačného $2-$faktoru daného grafu.

CDC podľa definície 1 sa nezaoberá tým, aké cykly sú v množine C. V prípade permutačného CDC je dané, že dva z cyklov v množine C musia byť pár permutačných cyklov z grafu. Keďže každá hrana musí byť pokrytá dvakrát a počet hrán v cykle je $(|G|/2) * 3$, tak súčet dĺžok cyklov z množiny cyklov permutačného CDC je $3|G|$. Permutačný $2-$faktor má súčet dĺžok cyklov $|G|$ a treba k nemu teda nájsť zvyšné cykly, ktorých súčet dĺžok bude $2|G|$.

Hypotéza 2. Žiaden permutačný snark nemá permutačný CDC.

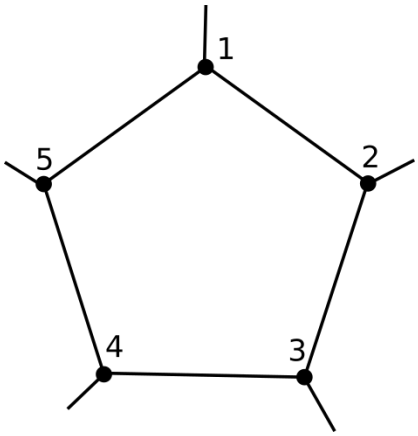
Aj z dôvodu tejto hypotézy program vyhľadáva v permutačných snarkoch permutačný CDC. Keďže existuje predpoklad, že taký snark neexistuje, bol by to veľký objav, ak by sa túto hypotézu pomocou programu podarilo vyvrátiť.

6.8 Klastre v grafe

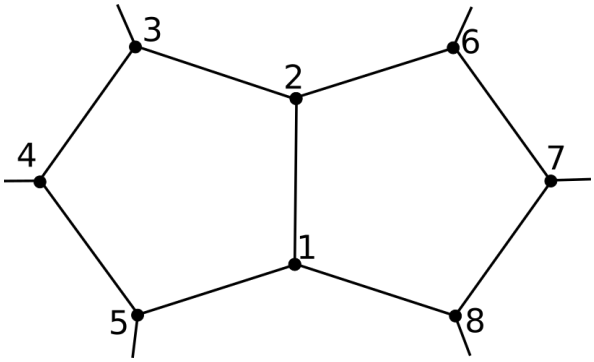
Posledná vec, ktorú program zisťuje sú klastre, ktoré sa v snarku nachádzajú. Program sa zaoberá petersenovskými klastrami. V prípade, že objaví nepetersenovský klaster, tak vypíše počet vrcholov v tomto klastri a pre všetky tieto vrcholy vypíše jeho susedov v danom klastri. Pokiaľ sú v nepetersenovskom klastri K dva susedné vrcholy, ale hrana medzi nimi nepatrí do K, je takýto susedný vrchol vypísaný ako posledný a oddelený bodkočiarkou. V prípade petersenovských klastrov najprv vypíše počet vrcholov a názov klastra a následné vypisovanie jednotlivých vrcholov je trochu špecifickejšie ako v prípade nepetersenovských klastrov. Zisťovanie klastrov v grafe má slúžiť aj na lepšiu vizualizáciu snarkov. Výpis petersenovských klastrov je preto naprogramovaný tak, aby poradie v akom sú vrcholy vypísané jednoznačne určovalo ako klaster vyzerá. Pre každý klaster je presne určené, ktorý vrchol z klastra je na danej pozícii vo výpise.

6.8.1 Vypisovanie vrcholov

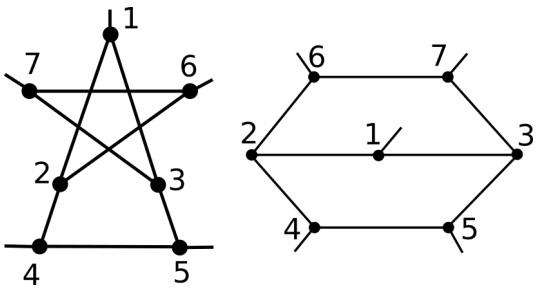
Na nasledujúcich obrázkoch sú znázornené všetky petersenovske klastre a čísla pri jednotlivých vrcholov znamenajú, koľký vo výpise vrcholov klastra je daný vrchol.



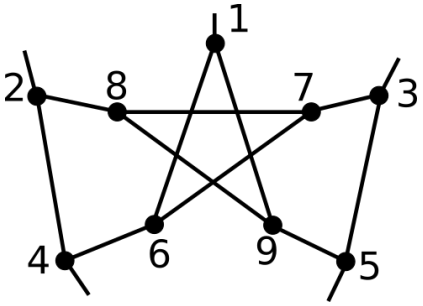
(a) pentagon



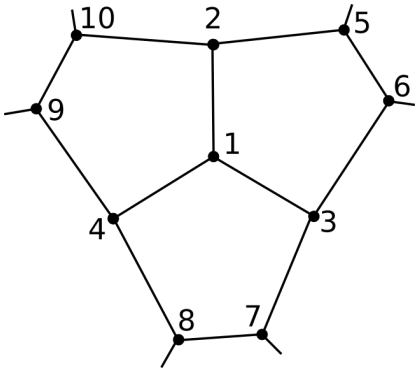
(b) double pentagon



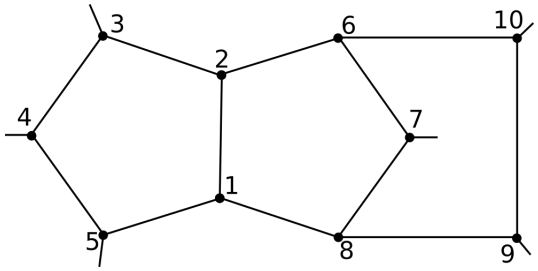
(c) negator



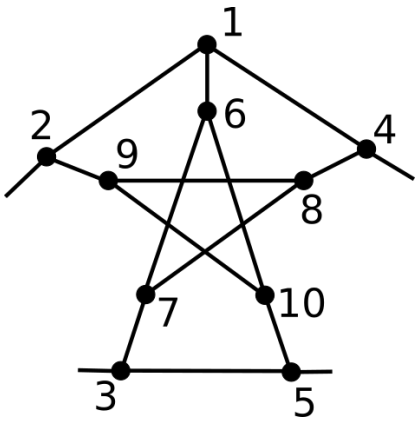
(d) triad



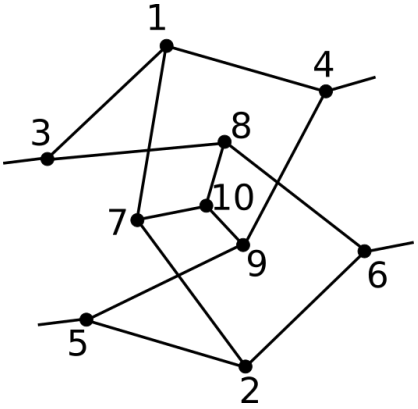
(e) triple pentagon



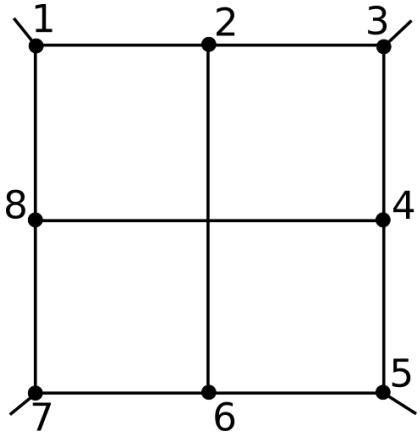
(f) triad



(g) heterochrome distace 1



(h) heterochrome distace 2



(i) isochrome

Kapitola 7

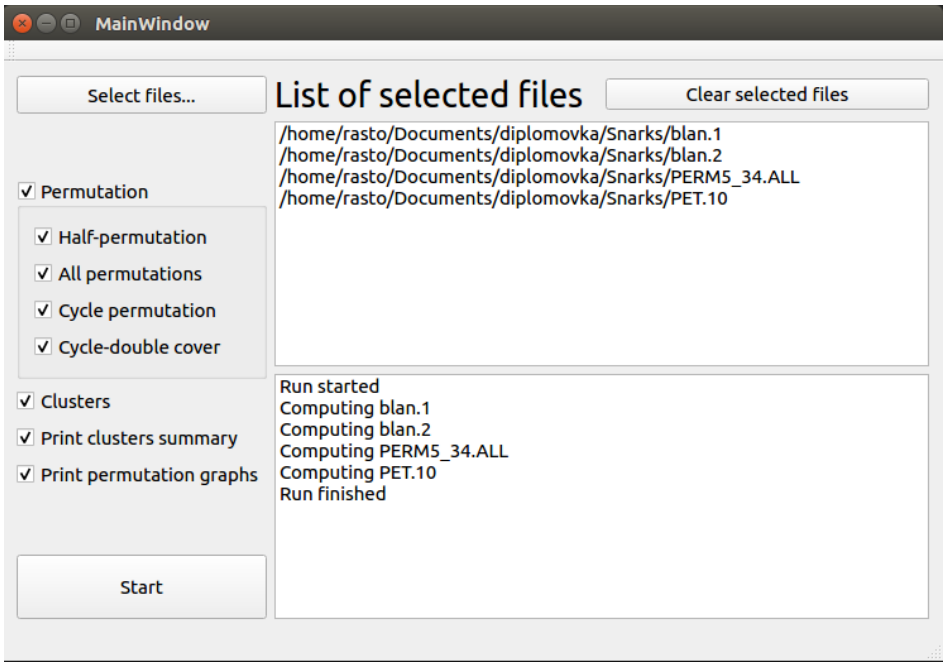
Funkcionalita programu

V tejto kapitole je popis výsledného programu. Je tu vysvetlené aké sú v ňom použité algoritmy, ako jednotlivé funkcie fungujú a ich popis pre potreby, ak by si chcel niekto program upraviť podľa vlastných potrieb. Jednotlivé funkcie a metódy sú popísané v poradí v akom sú volané a vykonávané počas spustenia programu. Celý kód je napísaný v jazyku C++. Okrem štandardných knižníc využíva program cross-platformovú knižnicu *Qt*[11] a v prípade linux based operačných systémov aj externý program *lingeling*[2], čo je SAT solver slúžiaci na rýchle vyhodnocovanie konjuktívnej normálovej formy, pomocou ktorej sa vypočítava, či má snark permutačný CDC. *Lingeling* nemá verziu kompatibilnú s operačným systémom Windows a preto sa na Windowse používa sa časovo náročnejšia heuristika na určovanie tejto vlastnosti. Z tohto dôvodu je odporúčané používať program na počítačoch, kde je linux-based operačný systém a nainštalovať SAT solver *lingeling*. Následne treba vytvoriť symbolický link na tento SAT solver do adresára */usr/bin/lingeling* pomocou príkazu *sudo ln -s /"cela cesta k satsolveru"/lingeling /usr/bin/*. Toto je potrebné z dôvodu, akým spôsobom je v programe urobené volanie externého programu *lingeling*.

7.1 Grafické prostredie

Grafické prostredie je jediná časť programu, s ktorou používateľ interaguje. Knižnica *Qt* je použitá práve na grafické prostredie. Grafické prostredie je urobené čo najjednoduchšie, pričom sa skladá iba z jedného okna. V tomto okne sa nachádzajú dve plochy na vypisovanie, tri tlačidlá a osem checkboxov.
Plochy na vypisovanie sa nachádzajú v strede okna. Horná plocha slúži na vylistovanie súborov, ktoré budú testované. Spodná plocha vypíše po skončení behu programu, že testovanie skončilo a vypíše všetky programy, ktoré boli pretestované.
Jednotlivé vlastnosti, ktoré budú testované zvolí používateľ v ľavej časti okna pomocou zaškrtnutia checkboxov. Pri každom checkboxe je napísané o akú vlastnosť ide.

Checkbox *Cycle permutation* v tomto prípade označuje testovanie parity permutačných cyklov ako aj zisťovanie involúcie snarku. *Print cluster summary* slúži na to, aby sa na konci vypísal sumár počtov všetkých klastrov zo všetkých grafov v súbore do samostatného súboru. Toto je užitočné najmä v prípade testovania súborov, ktoré obsahujú veľké množstvo grafov. *Print permutation graphs* vytvorí samostatný súbor, ktorý bude obsahovať zápis všetkých permutačných snarkov z testovaného súboru. Po zvolení vlastností musí používateľ vybrať súbory na pretestovanie a spustiť program.
Tlačidlo *Select files* slúži na vybranie vstupných súborov, ktoré bude program testovať. Po kliknutí na toto tlačidlo sa otvorí ďalšie okno, v ktorom sa otvorí domovský priečinok používateľa. V tomto okne sa používateľ prekliká do požadovaného adresáru, kde má súbory so snarkami a zvolí, ktoré chce testovať. Po zvolení a potvrdení týchto súborov sa zvolené súbory uložia do vektoru stringov *QFiles*. Všetky súbory uložené v *QFiles* sa následne vylistujú v hornej bielej ploche.
Tlačidlo *Clear selected files* slúži na vymazanie súborov z vektoru *QFiles*. Používateľ najprv v hornej ploche musí požadované súbory označiť a následne sa po stlačení tohto tlačidla vymažú z vektoru *QFiles*.
Posledné tlačidlo je tlačidlo *Start*. Po jeho zmačknutí sa zavolá metóda **computeFiles()**, ktorá následne pre každý súbor z vektoru *QFiles* zavolá funkciu **mainFunction()**, ktorá testuje jednotlivé vlastnosti. Tento program beží na pozadí a používateľ nemôže do neho nijak zasahovať.



Obr. 7.1: Ukážka grafického prostredia programu

7.2 Vstupný súbor

Program je urobený tak, že predpokladá, že všetky vstupné súbory sú validné a grafy v nich sú snarky. Na zápis grafov v súboroch sa používa takzvaný 'bratislavský formát', ktorý najprv obsahuje počet grafov v danom súbore a následne jednotlivé grafy.
Tie sú zapísané ako poradové číslo daného grafu v súbore, komentár k danému grafu, pričom ako komentár sa berie každý riadok ktorý začína symbolom {. Potom ide počet vrcholov grafu a nasleduje toľko riadkov, koľko má graf vrcholov. Každý riadok charakterizuje jeden vrchol a poradie riadku vyjadruje číslo vrcholu, pričom vrcholy sú indexované od 0. V riadku sú tri čísla oddelené medzerou, kde každé z týchto čísiel označuje jedného suseda prislúchajúceho vrcholu.

7.3 mainFunction()

Táto funkcia je základ celého programu a vedela by fungovať aj bez grafického prostredia, ktoré slúži pre jednoduchšie používanie. Funkcia dostane na vstupe názov a adresu súboru, ktorý ma testovať a sadu booleanovských premenných, ktoré určujú, ktoré vlastnosti sa majú testovať. Pre vstupný súbor sa najprv zavolá funkcia **scanner()**

7.3.1 scanner()

Scanner načíta všetky grafy zo súboru a pre každý graf vytvorí premennú typu *Graph*. *Graph* je štruktúra, ktorá uchováva všetky informácie o grafe. Počet vrcholov, vektor susedov pre každý vrchol, komentár k danému grafu a prázdne premenné, do ktorých program ukladá výsledky z testov jednotlivých vlastností.

7.3.2 chordless_cycles()

Po načítaní všetkých grafov **scannerom** sa nainicializujú výstupné programy, do ktorých sa postupne zapisujú výsledky. Potom program prechádza všetky načítané grafy a testuje v nich vlastnosti. Najprv musí v grafe nájsť bezchordové cykly. Tie sa vyhľadávajú pomocou funkcie **chordless_cycles()**, ktorá nájde všetky bezchordové cykly do dĺžky $n/2$, kde n je počet vrcholov grafu. Takéto cykly sú potrebné pre zisťovanie klastrov, pretože tam treba nájsť všetky cykly dĺžky 5 a na zisťovanie permutačnosti, kde treba cykly dĺžky $n/2$. Vyhľadávanie bezchordových cyklov funguje na základe algoritmu prehľadávania do hĺbky. Zvolí sa jeden vrchol v_i a prejdú sa všetky možné dvojice jeho susedov a z nich sa opäť prechádzajú ich susedia. Takto sa vygenerujú cykly, do ktorých v_i patrí, a pre ktoré sa následne preveruje, či sú bezchordové. Potom sa vrchol v_i odstráni a postup sa zopakuje pre ďalší vrchol. Toto sa opakuje dokým nie

je množina vrcholov prázdna. Po vyhľadaní všetkých bezchordových cyklov sa roztriedia na tie, ktoré majú dĺžku 5 a tie ktoré majú dĺžku $n/2$. Všetky ostatné sa zahodia, keďže nie sú potrebné. Táto funkcia je časovo veľmi náročná, keďže počet bezchordových cyklov exponenciálne narastá v závislosti od rádu grafu. Tento algoritmus je naimplementovaný podľa stránky Stack Overflow[12].

Rád grafu	#cyklov	#5-cyklov	$\#\lvert G\rvert/2$-cyklov
10	22	12	12
18	72	10	18
26	262	8	41
34	833	14	161
42	2883	16	535
50	12847	26	1778
58	43966	20	4640
66	541716	18	69787

Tabuľka 7.1: Počty bezchordových cyklov v závislosti od rádu grafu

7.3.3 findPermut()

Funkcia **findPermu()** sa zavolá na testovanie permutačnosti snarku. Keďže permutačný 2−faktor sa skladá z dvoch bezchordových cyklov dĺžky $n/2$ a priečok medzi nimi, odstránením vrcholov jedného z týchto cyklov v grafe ostane jeden bezchordový cyklus dĺžky $n/2$. Aj funkcia **findPermut()** takto funguje. Pre každý bezchordový cyklu dĺžky $n/2$ sa pozrie, ako by vstupný graf vyzeral po odstránení tohto cyklu a prechádza hrany a vrcholy, ktoré zostali v podgrafe a či tvoria dokopy cyklus dĺžky $n/2$. V prípade, že funkcia nájde pre nejaký cyklus permutačný 2−faktor tak skončí a netestuje už ďalšie cykly.

```
void findPermut(Graph &G, vector<vector> &cycles) {
//nainicializuje premenne
    vector<bool> used(G.noOfVerticies);
    size_t currentVert = 0;
    size_t startVert = 0;
    size_t length;
//na vstupe dostane vsetky cykly dlzky n/2
//iteruje cez vsetky z nich
    for (i = 0; i < cycles.size(); ++i) {
        //vytvori si pomocny vector pre druhy cyklus
        vector secondCycle;
        Cycle cycle = cycles[i];
```

```
        length = 0;

        for (j = 0; j < G.noOfVerticies; ++j) {
            used[j] = false;
        }
//pre vsetky vrcholy zo vstupneho cyklu nastavi
//ze uz boli pouzite
        for (k = 0; k < cycle.size(); ++k) {
            used[cycle[k]] = true;
        }

//nasledne iteruje cez vsetky vrcholy
//aby nasiel prvy nepouzity (neobsiahnuty v cykle)
//a ulozi ho do premennej currentVert a startVert
        for (l = 0; l < G.noOfVerticies; ++l) {
            if (!used[l]) {
                currentVert = l;
                startVert = l;
                break;
            }
        }

//zo startVert prechadza cez jeho susedov
//ktori nie su v prvom cykle az dokym sa opat
//nedostane do pociatocneho vrcholu startVert
//ked sa do neho dostane, tak sa ukonci cyklus
//ak ma dlzku n/2 tak sa jedna o druhy cyklus
//premutacneho 2-faktor
        while (true) {
         for (j = 0; j < 3; ++j) {
          neighbor = G.vertices[currentVert]
            .neighbors[j];
          if (!used[neighbor]) {
          secondCycle.push_back(currentVert);
          used[currentVert] = true;
          currentVert = neighbor;
          length++;
          if (length == G.noOfVerticies / 2 - 1) {
           for (k = 0; k < 3; ++k) {
```

```
            if (G.vertices[currentVert]
             .neighbors[k] == startVert) {
            length++;
            secondCycle.push_back(currentVert);
            goto goOut;
            } } }
          if (length == G.noOfVerticies / 2) goto goOut;
           break;
          }
          if (j == 2) goto goOut;
          } }

        goOut:;
        //ak ma cyklus polovicnu dlzku tak nasiel
        //permutacny 2-faktor a skonci, ak nema, tak zoberie
        //dalsi cyklus dlzky n/2
        if (length == G.noOfVerticies / 2) {
            G.permutationCycles.push_back(cycle);
            G.permutationCycles.push_back(secondCycle);
            G.permut = true;
            goto finish;
        } }
    finish:;
}
```

7.3.4 findAllPermut()

Ak je zvolené, že program má nájsť všetky permutačné 2−faktory, tak sa zavolá funkcia **findAllPermut()**, ktorá robí to isté ako **findPermut()**, len s tým rozdielom, že ak nájde nejaký permutačný 2−faktor, tak neskončí, iba pomocou jednoduchej pseudo-hashovacej funkcie overí, či už tento permutačný 2−faktor neobjavila predtým. Toto overovanie je potrebné z dôvodu, že pre dvojicu permutačných cyklov C_1 a C_2 by sa stalo, že by najprv pre C_1 našiel C_2 a následne by pre C_2 našiel C_1 a množina všetkých permutačných 2−faktorov by obsahovala duplikáty. Iná možnosť ako toto vyriešiť by bola, že po nájdení C_2 pre C_1 by sa oba vymazali z množiny cyklov dĺžky $n/2$, to by však bolo časovo náročnejšie.

```
//jedina rozdielna cast medzi findPermut()
//a findAllPermut()
```

```
//pokial najde permutacny 2-faktor tak neskonci
//iba ho pomocou pseudohashovacej prida na zoznam
//a pokracuje na dalsi cyklus
goOut:;
        if (length == G.noOfVerticies / 2) {
            uint64_t sumSecond = 0;
            uint64_t productSecond = 1;
            for(size_t m = 0; m < cycle.size(); ++m){
                sumSecond = sumSecond+secondCycle[m];
                productSecond = productSecond*(secondCycle[m]+1);
            }
            string secondCycleString = to_string(productSecond)
                + "_" +  to_string(sumSecond);

            usedCycles.insert(secondCycleString);
            usedCycles.insert(firstCycleString);

            G.allPermutationCycles.push_back(cycle);
            G.allPermutationCycles.push_back(secondCycle);
        }
```

7.3.5 halfPermut()

Ak graf nie je permutačný, tak funkcia **halfPermut()** testuje, či je aspoň polopermutačný. Aj táto funkcia funguje rovnako ako **findPermut()** a teda, že pre každý cyklus dĺžky $n/2$ sa pozrie, či jeho odstránením z grafu bude vzniknutý podgraf 2−faktor tvorený viacerými cyklami.

```
//prechadza mnozinu vrcholov, ktore nie su
//v cykle dlzky n/2
while (notUsed.size() > 0) {
 Cycle newCycle;
//zacne z vrcholu currentVert a prechadza
//jeho susedov, ktory nie su v cykle dlzky n/2
//dokym sa opat nedostane do pociatocneho vrcholu
//tym sa uzatvori cyklus a zoberie si dalsi nepouzity vrchol
//toto opakuje, dokym nie su vsetky vrcholy pouzite
 currentVert = *notUsed.begin();
 notUsed.erase(currentVert);
```

```
 newCycle.push_back(currentVert);
 bool hasNext = true;
 while (hasNext) {
  hasNext = false;
  for (j = 0; j < 3; ++j) {
  neighbour = G.vertices[currentVert].neighbors[j];
  if (!used[neighbour] &&
  notUsed.find(neighbour)!=notUsed.end()){
   currentVert = neighbour;
   notUsed.erase(currentVert);
   newCycle.push_back(currentVert);
   hasNext = true;
   break;
  } } }
  G.halfPermutationCycles.push_back(newCycle);
}
```

7.3.6 determineIsomorphic()

Táto funkcia slúži na to, aby pre všetky permutačné 2-faktory určila, ktoré z nich sú navzájom izomorfné. Porovnáva medzi sebou všetky dvojice a pozerá sa na permutáciu týchto cyklov a poradie priečok ako sú navzájom prepojené v jednom a druhom permutačnom 2-faktore.

```
void determineIsomorphic(Graph &G){
//na zaciatku si zoberie vsetky permutacne 2-faktory
vector<vector<>> permCycles = G.allPermutationCycles;
//nasledne iteruje cez ne vsetky
for (i = 0; i < permCycles.size(); i=i+2) {
 vector firstCycle = permCycles[i];
 vector secondCycle = permCycles[i+1];

//pre sucasnu dvojicu permutacnych cyklov urci
//permutaciu priecok ako su poprepajane
 vector cyclePerm;
 for (l = 0; l < firstCycle.size(); ++l) {
  currentVertex = firstCycle[l];
  nonCycleNeigh = 0;
  for (j = 0; j < 3; ++j) {
```

```
  curr = G.vertices[currentVertex].neighbors[j];
  if (!elementInVector(&firstCycle, curr)) nonCycleNeigh = curr;
  }
  nonCyclePos = positionOfElement(&secondCycle, nonCycleNeigh);
  cyclePerm.push_back(nonCyclePos);
  }
//nasedne tuto permutaciu priecok porovnava s ostatnymi
//permutacnymi 2-faktormi zavolanim funkcie compareCycles()
  for (k = 0; k < result.size(); k=k+2) {
   compareCycles(G, result[k], result[k+1], cyclePerm))
```

```
bool compareCycles(Graph &G, vector<> currentFirstCycle,
vector<> currentSecondCycle, vector<> cyclePerm){

//na vstupe dostane 2 permutacne cykly
//vytvori cykly s opacnym smerom
vector<> reverseCurrentFirst = reverseCycle(currentFirstCycle);
vector<> reverseCurrentSecond = reverseCycle(currentSecondCycle);

//na zistenie toho, ci existuje rovnaka permutacia
//priecok treba skontrolovat vsetky rotacie permutacnych cyklov
//aj to ktory cyklus je brany ako prvy
//nasledne sa pre vsetky zavola singleOrientationComparsion()
sOC(G, currentFirstCycle, currentSecondCycle, cyclePerm))
sOC(G, currentFirstCycle, reverseCurrentSecond, cyclePerm))
sOC(G, reverseCurrentFirst, currentSecondCycle, cyclePerm))
sOC(G, reverseCurrentFirst, reverseCurrentSecond, cyclePerm))
sOC(G, currentSecondCycle, currentFirstCycle, cyclePerm))
sOC(G, reverseCurrentSecond, currentFirstCycle, cyclePerm))
sOC(G, currentSecondCycle, reverseCurrentFirst, cyclePerm))
sOC(G, reverseCurrentSecond, reverseCurrentFirst, cyclePerm))
```

```
bool singleOrientationComparsion(Graph &G,
vector<> currentFirstCycle, vector<> currentSecondCycle,
vector<> cyclePerm){
//treba skontrolovat vsetky shiftnutia (pociatocne vrcholy)
//cyklov, aby bolo iste, ze permutacie priecok nie su rovnake
for (i = 0; i < currentFirstCycle.size(); ++i) {
 for (j = 0; j < currentSecondCycle.size(); ++j) {
```

```
//pre sucasnu konfiguraciu cyklov vytvori permutaciu priecok
//permutaciu vytvara ako to, na kolky vrchol v druhom cykle
//je prepojeny n-ty vrchol v prvom cykle
  vector<> currentCyclePerm;
  for (k = 0; k < currentFirstCycle.size(); ++k) {
  currentVertex = currentFirstCycle[k];
  nonCycleNeigh = 0;
  for (l = 0; l < 3; ++l) {
  curr = G.vertices[currentVertex].neighbors[l];
  if (!elementInVector(&currentFirstCycle, curr))
    nonCycleNeigh = curr;
  }
  nonCyclePos = positionOfElement(&currentSecondCycle,
   nonCycleNeigh);
  currentCyclePerm.push_back(nonCyclePos);
  }
//pokial su permutacie rovnake, tak sa moze vratit
//lebo nasiel izomorfne permutacne 2-faktory
  if (currentCyclePerm == cyclePerm) {
   return true; }
//pokial nie su rovnake tak shiftne druhy cyklus o jeden prvok
  shiftSecond = currentSecondCycle.at(0);
  currentSecondCycle.erase(currentSecondCycle.begin());
  currentSecondCycle.push_back(shiftSecond);
}
//ak preveril vsetky shiftnutia druheho cyklu
//tak shifne prvy cyklus o prvok a preveruje znova
//vsetky mozne posuny druheho cyklus
//toto opakuje pre vsetky shiftnutia prveho cyklu
 shiftFirst = currentFirstCycle.at(0);
 currentFirstCycle.erase(currentFirstCycle.begin());
 currentFirstCycle.push_back(shiftFirst);
}
return false;
}
```

7.3.7 runCyclePerm()

Na vypočítanie parity permutačných cyklov a následného zistenia involúcie je funkcia **runCyclePerm()**. Táto funkcia prechádza množinu všetkých permutačných 2−faktorov a pre všetky možné orientácie permutačných cyklov a všetky možnosti počiatočných vrcholov zavolá funkciu **cyclePermutationInvolution()**. Táto funkcia je takmer totožná a funguje na rovnakom princípe ako preverovanie izomorfných permutačných 2-faktorov. Jediný rozdiel je, že v prípade determineIsomorphic() prechádza vrcholy cyklu postupne a robí permutáciu priečok, tak v tejto funkcií ak je k-ty vrchol z prvého cyklu prepojený s l-tým vrcholom, tak v ďalšom kroku nepozerá v prvom cykle prvok $k+1$ ale prvok l.

```
//pozera sa na k-ty prvok v prvom cykle a zistuje
//s ktorym prvokm v druhom cykle je prepojeny
currentCycle.push_back(currentVertex);
used[currentVertex] = true;
 diffCycle = -1;
 vector<> neigh = G.vertices[firstCycle[currentVertex]].neighbors;
 for (k = 0; k < 3; ++k) {
  if (!elementInVector(&firstCycle, neigh[k])){
   diffCycle = int(neigh[k]);
 } }
//zisti, ze k-ty prvok v prvom cykle je prepojeny s l-tym
//prvokm v druhom cykle a do dalsej iteracie si miesto k+1
//zoberie poziciu l
currentVertex = positionOfElement(&secondCycle, size_t(diffCycle));
```

7.3.8 cyclePermutationInvolution()

V 6.5 je popísaný postup s označovaním vrcholov $0..n/2$ a $0'..n/2'$. Keďže cykly sú v celom programe zapísané vo vektoroch ako postupnosť vrcholov, tak takéto označovanie netreba robiť a stačí použiť index daného elementu vo vektore prislúchajúceho cyklu. Najprv sa nainicializuje množina vrcholov, ktoré už boli použité. Vyberie sa prvý nepoužitý vrchol z prvého permutačného cyklu a pozrie sa, aké je číslo tohto vrcholu. Toto číslo sa označí ako *currentVertex* a vloží sa do vektoru *currentCycle* reprezentujúceho jednu permutáciu. Potom sa funkcia pozrie na suseda vrcholu *currentVertex*, ktorý patrí do druhého permutačného cyklu, uloží si index tohto vrcholu do premennej *diffCycle*. Funkcia **positionOfElement()** vypočíta pozíciu elementu *diffCycle* vo vektore *secondCycle* a toto číslo uloží do premennej *currentVertex*. Takýto postup, počas ktorého sa postupne skáče medzi jedným a druhým permutačným cyklom sa opakuje,

dokým nie je permutácia konečná. To nastáva vtedy, keď nastane prípad, že sa do *currentVertex* uloží element, ktorý už je vo vektore *currentCycle* - vždy je to element, ktorý je v tomto vektore prvý. Konečná permutácia sa následne uloží do vektoru *result*, ktorý obsahuje všetky zistené permutácie. Toto sa vykonáva dookola, kým nie sú všetky vrcholy z grafu v množine použitých vrcholov. Potom sa pre všetky permutácie vo vektore *result* vypočíta parita podľa vzorca $parita = (\sum perm - 1)\%2$ a tak isto sa pozrie, či nie je dĺžka každej permutácie ≤ 2, čo by určovalo involúciu grafu.

7.3.9 Cycle double cover - Windows

Cycle double cover má dve rôzne implementácie v závislosti od toho, či je program spúšťaný na Windowse alebo na Linuxe. V prípade Windowsu sa volá funkcia **signedRotationCycles()**, na linuxe sa volá funkcia **createSATfile()**.
V prípade Windowsu sa nastaví pre graf signovaná rotácia. Táto časť kódu je pravdepodobne najnáročnejšia na modifikovanie ako aj pochopenie. Ide o systém, kde ma každý vrchol nastavený orientáciu v akom poradí idú jeho incidentné hrany. Tak isto má každá hrana nastavený svoj smer a priradí sa jej či je kladná alebo negatívna a tak isto je nastavený súčasný stav prechádzania, ktorý je tiež pozitívny alebo negatívny. Následne sa v takomto grafe prechádza po hranách a pamätá sa stav a smer v akom sa hrana prechádzala - kladný alebo negatívny a po smere alebo protismeru orientácie hrany. Ak je hrana negatívna tak sa stav prechádzania zmení na opačný. Keď sa dojde do vrcholu, tak v kladnom stave ide ďalej po hrane, ktorá je po smere orientácie vrcholu, v zápornom stave sa vyberie do protismeru. Keď sa pri prechádzaní dostane znovu do stavu, že sa prejde po hrane po ktorej už sa šlo v rovnakom smere a v rovnakom stave, tak sa to zacyklí a toto definuje jeden cyklus z permutačného CDC. Ak existuje taká signovaná rotácia, že cez každú hrana sa prejde v obidvoch smeroch práve raz, tak vytvorené cykly budú tvoriť permutačné dvojité cyklové pokrytie. Keďže na začiatku sú dané dva permutačné cykly, tak všetkým vrcholom a hranám týchto cyklov je daná orientácia a smer hrán podľa orientácie permutačného cyklu. Následne treba preveriť všetky možnosti nastavenia priečok, smer týchto priečok môže byť ľubovoľný, avšak treba preveriť každý prípad, pre každú priečku či je jej stav kladný alebo záporný. Keďže priečok je v grafe $|G|/2$ tak pre veľké grafy to bude činiť obrovské množstvo možností, ktoré narastá exponenciálne v závislosti od počtu vrcholov a tým narastá aj doba výpočtu tejto vlastnosti. Platí, že možností nastavaní kladnosti respektíve zápornosti priečok je v grafe $2^{|G|/2}$.
Signovaná rotácia sa môže používať aj na zafarbovanie planárnych grafov. Podrobné vysvetlenie toho ako funguje sa nachádza v článkoch *The combinatorial map color theorem*[13] a *Generalized embedding schemes*[14]

signedRotationCycles()

Funkcia **signedRotationCycles()** pre každý pár permutačných cyklov nastaví smer týchto cyklov a zavolá funkciu **initSignedRotation()**.

initSignedRotation()

Volanie tejto funkcie nainicializuje signovanú rotáciu podľa vstupných orientácii permutačných cyklov. Každému vrcholu nastaví orientáciu a hranám permutačných cyklov smer, pričom všetky sú kladné. Všetkým priečkam nastaví smer a zavolá funkciu **signedSystemAllPossibilites()**

signedSystemAllPossibilites()

signedSystemAllPossibilites() vytvorí niekoľko pomocných polí, ktoré budú potrebné na prechádzanie signovanej rotácie a zavolá rekurzívne volanie **switchChords()**.

switchChords()

Táto rekurzívna funkcia sa volá do hĺbky $|G|/2$, pričom v hĺbke n nastaví priečke s indexom n obe možnosti - či je kladná alebo záporná a pre obidve prepnutia zavolá **swtichChords()** pre hĺbku $n+1$. Pokiaľ je hĺbka $|G|/2$, znamená to, že všetky priečky už majú nastavený stav a zavolá sa **computeCycleDouble()**.

computeCycleDouble()

Na vstupe dostane signovanú rotáciu, kde už sú nastavené všetky potrebné stavy hrán a orientácie vrcholov a prechádza cez tieto hrany a overuje, či prejde cez každú hranu práve dvakrát. Ak sa tak udeje, tak sa funkcia ukončí, uloží cykly permutačného CDC a nepreveruje už ďalšie možnosti.

7.3.10 Cycle double cover - Linux

Testovanie permutačného CDC v linuxe je podstatne jednoduchšie. Používa sa na to SAT solver *lingeling*, ktorému treba dať jeden vstupný súbor. Tento súbor musí v prvom riadku obsahovať informácie o tom, koľko obsahuje formúl a premenných. Následne každý riadok obsahuje jednu formulu v konjuktívnej normálovej forme ukončený znakom 0, pre formulu platí, že ak má na začiatku premenná znak mínus, znamená to negáciu danej premennej. Na vytvorenie tohto súboru sa používa funkcia **createSATfile()**.

createSATfile()

Na to aby permutačný graf mal permutačné CDC treba v jazyku SAT solvera splniť určité podmienky. Pomocou týchto podmienok sa pre vstupné hrany hľadajú cykly. Pre každú podmienku je vytvorená samostatná funkcia, ktorá zapisuje formuly v konjuktívnej normálovej forme do súboru, pre ktorý sa následne spustí *lingeling*

- **symmetry()** - ak je v cykle hrana idúca z vrcholu v_i do v_j, musí v ňom byť aj hrana idúca z v_j do v_i
- **inCycle()** - každá hrana v grafe musí byť v nejakom cykle
- **minTwoEdges()** - pre každý vrchol v cykle platí, že minimálne dve hrany s ktorými je incidentný sú v danom cykle tiež
- **maxTwoEdges()** - pre každý vrchol v cykle platí, že maximálne dve hrany s ktorými je incidentný sú v danom cykle tiež
- **containPermutCycles()** - permutačné cykly musia byť tiež vo výsledných cykloch
- **minTwoCycles()** - každá hrana v cykle musí byť aspoň v dvoch cykloch
- **maxTwoCycles()** - každá hrana v cykle musí byť maximálne v dvoch cykloch

Tieto podmienky sa zapíšu do súboru *satfile.txt* pomocou volania **system()** sa spustí *lingeling* a ako vstupný súbor sa mu zadá *satfile.txt*. Pre potreby tohto volania je potrebné, aby bol vytvorený symbolický link na *lingeling* v */usr/bin/* Z výsledného súboru z *lingelingu* sa následne sparsuje výsledok, či graf má alebo nemá permutačné dvojité cyklické pokrytie. Beh *lingelingu* je veľmi rýchly, rádovo sú to sekundy čo je veľký rozdiel oproti testovania vo Windowse.

7.3.11 separate_cycles_into_graphs()

Funkcia slúži na zisťovanie klastrov 5-cyklov. Množinu cyklov dĺžky 5 pomocou algoritmu Union-Find a funkcie **separate_graph_vertices()** rozdelí podľa vrcholov na vrcholovo disjunktné množiny a tým zadefinuje, ktoré 5-cykly patria do rovnakého klastru. Po rozdelení výsledok uloží do premennej *klaster* v ktorej sú uložené všetky klastre ako množina cyklov dĺžky 5. Pre každý prvok z premennej *klaster* zavolá funkciu **clusterType()**, ktorá určí podľa trojice počet vrcholov, počet cyklov a počet hrán o ktorý klaster sa jedná. Pri klastroch *triple pentagon* a *tricell* ešte overuje, či existuje vrchol, ktorý je vo všetkých troch cykloch dĺžky 5.
Následne je pre každý typ petersenovského klastru vytvorená samostatná funkcia, ktorá

daný klaster vypíše tak, ako je to popísané v 6.8.1. Tieto funkcie fungujú na základe vyhľadávania špecifických vrcholov v klastroch a ich susedov a podľa toho určujú, ktorý vrchol je ktorý a následne ich vypíšu. Všetky tieto funkcie sa nachádzajú v súbore *printclusters.cpp*.

```
separate_cycles_into_graphs(const vector<Cycle> &cycles) {
auto cycledGraphs = <vector<CycledGraph>>(new vector<CycledGraph>());

//rozdeli vrcholy na jednotlive podgrafy(klastre)
//pomocou funkcie separate_...()
 vector<GraphVertices> graphs_vertices;
 separate_graph_vertices(cycles, graphs_vertices);

for (l = 0; l < graphs_vertices.size(); ++l) {
 CycledGraph emptyGraph;
 cycledGraphs->push_back(emptyGraph);
}
//vsetkym cyklom priradi, do ktoreho podgrafu patria na
//zaklade vrcholov v tomto cykle
 for (i = 0; i < cycles.size(); ++i) {
 firstElement = cycles[i][0];
 for (j = 0; j < graphs_vertices.size(); ++j) {
 unordered_set<size_t> currentVertices = graphs_vertices[j];
 bool graphHasThisVertex = currentVertices
  .find(firstElement) != currentVertices.end();
 if (graphHasThisVertex){
 (*cycledGraphs)[j].push_back(cycles[i]);
  break;
 } } }
```

Táto funkcia v sebe volá funkciu separate_graph_vertices(), ktorá rozdeluje vrcholy do jednotlivých podgrafov na základe cyklov dĺžky 5. Implementácia algoritmu union-find.

```
//iteruje cez vsetky cykly dlzky 5
for (i = 0; i < cycles.size(); ++i) {
Cycle cycle = cycles[i];
//vytvori novy podgraf, skladajuci sa z jedneho cyklu
GraphVertices current_cycle_graph(cycle.begin(), cycle.end());
//nasledne pozera, ci moze niektore podgrafy spojit
//do jedneho podgrafu na zaklade spolocnych vrcholov
```

```
for (k = (graphs_vertices.size()) - 1; k >= 0; --k) {
GraphVertices graph = graphs_vertices[size_t(k)];
//iteruje cez vsetky vrcholy aktualneho podgrafu a pozera
//ci sa vrchol nevyskytuje v oboch tychto podgrafoch
 for (j = 0; j < cycle.size(); ++j) {
 vertex = cycle[j];
 bool graphHasThisVertex = graph.find(vertex) != graph.end();
 if (graphHasThisVertex) {
//ak sa nejaky vrchol vyskytuje v 2 podgrafoch, tak tieto 2
//podgrafy spoji do jedneho vacsieho
 merge_graph_vertices(graph, current_cycle_graph);
//po spojeni podgrafov vymaze predosle
 graphs_vertices.erase(remove(graphs_vertices.begin(),
  graphs_vertices.end(), graph), graphs_vertices.end());
break;
 } } }
//vrati maximalne podgrafy
graphs_vertices.push_back(current_cycle_graph);
}
```

7.4 Výstupný súbor

Súbor s výsledkami testovania má názov ako pôvodný súbor, len pred príponu ešte pridá *_results*. Defaultne je to nastavené, že sa v samostatnom priečinku *results/*, ktorý je v adresári odkiaľ bol program spúšťaný - túto časť rieši knižnica Qt a na niektorých operačných systémoch sa to môže líšiť. Výstupný súbor obsahuje informáciu o počte testovaných grafoch, celkový počet permutačných a polopermutačných grafov v súbore a výsledky pre jednotlivé grafy.

Pre grafy je najprv uvedené číslo grafu vo vstupnom súbore, následne je na výstupe vypísaný počet vrcholov grafu. Nasleduje informácia o tom či je daný graf permutačný a ak je, tak vypíše počet permutačných 2−faktorov a za tým vypíše všetky bicyklické reprezentácie daného grafu. Bicyklické reprezentácie vypíše vo formáte dĺžka cyklu a následne všetky vrcholy cyklu tak, ako idú postupne v cykle. Následne vypíše počet neizomorfných bicyklických reprezentácií grafu a vypíše všetky tieto bicyklické reprezentácie v rovnakom formáte ako všetky bicyklické reprezentácie.

Ak graf obsahuje permutačný CDC tak túto informáciu vypíše a za ňou idú všetky cykly z množiny permutačného CDC, pričom najprv idú dva permutačné cykly až potom ostatné. Tvar výpisu cyklov je počet vrcholov a následne všetky jeho vrcholy v

poradí v akom idú v cykle.
Na výstupe nasleduje informácia o parite permutácie permutačných cyklov a ak je graf involučný, tak vypíše aj konkrétnu permutáciu permutačných cyklov, ktorá dáva involúciu. Ak nie je involučný nevypíše nič. Ako posledné vypíše informácie o klastroch v grafe. Ich počet, následne pre každý klaster názov tohto klastru, počet jeho vrcholov a potom vrcholy v takom poradí ako je to popísane v 6.8.1. Posledný riadok je informácia o vrcholoch, ktoré nie sú v žiadnom klastri.

Kapitola 8

Výsledky

Hlavným cieľom práce bolo vytvoriť program, ktorý zvládne overovať určité vlastnosti v snarkoch. Program funguje spoľahlivo s tým, že pri väčšine funkcii bol dôraz kladený na to, aby bola časová zložitosť výpočtu čo najefektívnejšia a tým pádom aj samotný program čo najrýchlejší. Tak isto je program štrukturovaný tak, aby pridávanie nového kódu a testovanie ďalších vlastností v budúcnosti čo najjednoduchšie. Funkcie sú naprogramované tak, aby v boli jedna od druhej čo najmenej závislé, aby úprava jednej funkcie súčasného kódu nespôsobila znefunkčnenie ďalších vlastností.
Popri hlavnom programe boli v rámci tejto práce naprogramované aj ďalšie tri programy. Jeden je modifikácia hlavného programu, ktorá slúži iba na vyhľadávanie klastrov v grafe, s tým, že niektoré funkcie sú upravené a zefektívnené pre potreby tohto konkrétneho účelu. Jedna z takýchto funkcii je napríklad funkcia **chordless_cycles()**, ktorá v hlavnom programe musí vyhľadať všetky bezchordové cykly do veľkosti $|G|/2$. V prípade klastrov stačia iba bezchordové cykly do dĺžky 5 a teda je jej beh podstatne rýchlejší.
Zvyšné dva programy sú implementácie dvoch spôsobov generovania permutačných snarkov. Jedno je implementácia *star productu* a druhé je *negatorová substitúcia*. Tieto programy boli naprogramované z dôvodu generovania permutačných cyklicky 5-súvislých permutačných snarkov. Takto vygenerované snarky boli následne testované hlavným programom, aby mohli byť analyzované ich vlastnosti.
Pomocou týchto dvoch programov bolo vygenerovaných a pretestovaných približne tritisíc cyklicky 5-súvislých permutačných snarkov. Väčšina z nich bola vygenerovaná zo snarkov PERM5.34 medzi sebou, ale niekoľko stoviek bolo skonštruovaných aj zo 42 a 50 vrcholových grafov. Tieto grafy boli vstupy do programov pre *star product* a *negatorovú substitúciu* a boli medzi sebou rôzne kombinované. Vygenerované grafy sa teda pohybujú od 58 vrcholových až po 90 vrcholové permutačné 5-súvislé snarky.
Samotné testovanie hlavným programom neprinieslo veľmi prekvapivé výsledky. Žiaden z grafov nemal permutačné dvojité cyklové pokrytie, čo bol očakávaný výsledok.

Pokiaľ by sa objavil graf s permutačným CDC išlo by o veľký objav, keďže by sa podarilo vyvrátiť hypotézu 2. To sa žiaľ neudialo. V prípade cyklických permutačných 5-súvislých snarkov bol najčastejší klaster *negator*, ktorý tvoril približne 65% všetkých klastrov v týchto grafoch. Po ňom nasledovali klastre *pentagon* a *triad*, občas sa tiež vyskytol *double pentagon*. Ostatné klastre sa v nich nevyskytli ani raz.
V prípade nepermutačných snarkov sa nedá povedať, že by niektorý klaster výrazne svojim počtom prevyšoval ostatné. Zastúpenie v grafoch mali všetky klastre, avšak klastrov, ktoré vzniknú z Petersenovho grafu iba prerezávaním hrán, bolo výrazne menej ako ostatných. Konkrétne sa jedná o klastre *tricell*, *triple pentagon*, *heterochrome 1* a *heterochrome 2*. Nepetersenovské klastre sa tak isto vyskytovali bežne a objavili sa aj prípady, keď bol v grafe iba nepetersenovský klaster a nebol v ňom žiaden klaster odvodený z Petersenovho grafu.
V prípade permutačných snarkov sa nepodarilo objaviť žiaden taký, pre ktorý by platilo ($|G| mod 8 = 6$).
Okrem testovania grafov sa podarilo odvodiť všetky petersenovské klastra a dokázať úplnosť tejto množiny. Tak isto boli vizualizované grafy PERM5.34 čo pomohlo k poznatku, že z dvoch z nich sa dá skonštruovať zvyšných desať poprepájaním dvoch hrán.
Zdrojový kód programu, ako aj programov na testovanie klastrov a generovanie snarkov pomocou a *negatorovej substitúcie* sa nachádzajú na disku priloženom k výtlačku práce ako aj v repozitári *https://github.com/kalerab/diplomova_praca* na *GitHube*.

Záver

V našej práci sme sa zaoberali štruktúrou snarkov a skúmaniu ich vlastností. Pre tieto účely sa nám podarilo vytvoriť program na počítanie vlastností a vyhľadávanie klastrov, ako aj ich generovanie. Niektoré snarky a podgrafy boli vizualizované a bola dokázaná aj úplnosť množiny petersenovských klastrov.

Aj napriek pretestovaniu desaťtisícok grafov, vygenerovaniu tisícov nových snarkov a všetkým poznatkom zhrnutých v kapitole 8, je množstvo otázok v teórii grafov týkajúcich sa snarkov, na ktoré nie sú známe odpovede.
Existuje permutačný snark rádu $6(mod 8)$? Má každý permutačný snark obvod 5? Existuje permutačný snark s cyklickou súvislosťou 6? Existuje permutačný snark, ktorý má permutačné dvojité cyklové pokrytie?

Je možné, že na niektoré z týchto otázok nebude odpoveď známa ešte mnoho rokov, treba však pokračovať v zisťovaní vlastností snarkov a snažiť sa pochopiť ich štruktúru. Je to potrebné najmä z dôvodu, že existuje množstvo problémov v teórii grafov, ktoré sa dajú zredukovať iba na snarky. Súčasná výpočtová sila počítačov umožňuje počítať a testovať množstvo vlastností aj pre veľké grafy, čo bolo pred rokmi nemožné. Iba postupným zisťovaním ďalších vlastností čo najväčšieho množstva grafov, sa podarí nájsť súvislosti medzi jednotlivými vlastnosťami.
Treba iba dúfať, že sa to podarí v čo najkratšom čase.

Literatúra

[1] A.G.Chetwynd and R.J.Wilson. Snarks and supersnarks. *The Theory and Applications of Graphs*, pages 215–241, 1981.

[2] A. Biere. Lingeling, 2018.

[3] Reinhard Diestel. *Graph Theory*. Springer-Verlag Heidelberg, 4 edition, 2005.

[4] Martin Škoviera Edita Máčajová. Permutation snarks. 2019.

[5] Martin Gardner. *Mathematical Games*. 1976.

[6] Jonas Hägglund and Arthur Hoffmann-Ostenhof. Construction of permutation snarks. *Combinatorics(math.CO)*, 2012.

[7] I.Holyer. The complexity of graph theory problems. 1980.

[8] Chladný Miroslav and Škoviera Martin. Factorisation of snarks. *The Electronic Journal of Combinatorics*, 17(R32), 2010.

[9] Edita Máčajová and Martin Škoviera. Construction of permutation snarks of order 2 (mod 8). *Department of Computer Science, Comenius University, Bratislava, Slovakia.*

[10] Roman Nedela and Martin Škoviera. Decompositions and reductions of snarks. *Journal of Graph Theory*, Vol. 22, No. 3:253–279, 2013.

[11] Haavard Nord and Eirik Chambe-Eng. Qt, 1995.

[12] Stack Overflow. Find all chordless cycles, 2010.

[13] Gerhard Ringel. The combinatorial map color theorem. *Journal of Graph Theory*, 1977.

[14] Saul Stahl. Generalized embedding schemes. *Journal of Graph Theory*, 1978.

[15] Wikipedia the free encyclopedia. Descartes snark, 2010.

[16] Wikipedia the free encyclopedia. Flower snarks, 2010.

[17] Wikipedia the free encyclopedia. Szekeres snark, 2010.

[18] Wikipedia the free encyclopedia. Watkins snark, 2010.

[19] Ghent University. House of graphs. 2010.

[20] John J. Watkins. Snarks. *Ann. New York Acad. Sci.*, 576:606–622, 1989.

[21] John J. Watkins and Robin J. Wilson. A survey of snarks. *Graph Theory, Combinatorics and Applications*, Vol 2:1129–1144, 1991.

[22] Thomas Zaslavsky. Six signed petersen graphs, and their automorphisms. *Discrete Mathematics*, 312:1558–1583, 2012.

[23] Martin Škoviera and Edita Máčajová. Permutation snarks. Bratislavský seminár z teórie grafov, 2015.