Hľadanie imúnnych podgrafov

BAKALÁRSKA PRÁCA

František Hajnovič

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Katedra informatiky

9.2.1 Informatika

Vedúci: doc. RNDr. Rastislav Královič, PhD.

Bratislava 2010

Čestné prehlásenie

Čestne prehlasujem, že túto bakalársku prácu som vypracoval samostatne, len s použitím citovanej literatúry.

.....

Poďakovanie

Touto cestou by som sa rád poďakoval svojmu školiteľovi doc. RNDr. Rastislavovi Královičovi PhD. za jeho podnety a rady, ktorými ma usmerňoval pri tvorbe tejto bakalárskej práce.

Abstrakt

Práca sa zaoberá hľadaním imúnnych podgrafov v systémoch s väčšinovým hlasovaním. Okrem uvedenia do problematiky a prehľadu najdôležitejších výsledkov práca poskytuje návrhy riešenia daného problému, ich diskusiu a program, ktorý riešenia implementuje.

Kľúčové slová: **väčšinové hlasovanie**, **imúnne podgrafy**

Abstract

This paper talks about searching for immune subgraphs in systems with majority voting. Apart from the introduction to the problematics and overview of the most important results, this paper also proposes solutions of the given problem, the related discussion and a computer program which implements the solutions.

Key words: **majority voting**, **immune subgraphs**

Obsah

Úvod

Prvotným podnetom na tvorbu tejto bakalárskej práce bol problém s ekonomickým podtónom, ktorý má súvis so šírením informácii po grafe a s teóriou grafov. Ide najmä o oblasť väčšinového hlasovania a hľadania minimálnych imúnnych podgrafov v grafoch, čo je problém NP-úplného charakteru.

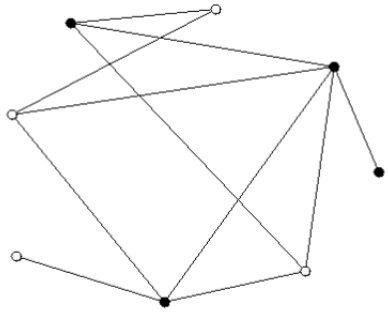
Väčšinové hlasovanie je princíp, ktorý sa vyskytuje v spoločnosti všade tam, kde je rozhodovanie jednotlivca ovplyvňované mienkou okolia. Ako príklad môže poslúžiť situácia na burze, kde "nikto nekupuje to, čo nikto nechce", či rozhodovanie sa koho ísť voliť pri pomerne jednostrannej politickej mienke okolia. V distribuovaných systémoch sa zase hlasovanie používa na zachovanie konzistencie a validity dát.

Uvedené prípady možno zovšeobecniť na model, v ktorom sa počiatočná pluralita názorov (resp. stavov) postupne v závislosti od štruktúry prepojenia v systéme mení na jednostranné zmýšľanie celku. K hlavným otázkam, ktoré sa ponúkajú patrí tá, či sa dá zamedziť prevládnutiu jedného názoru vhodnou počiatočnou konfiguráciou modelu - teda takou, ktorá bude voči takémuto nežiadúcemu správaniu imúnna.

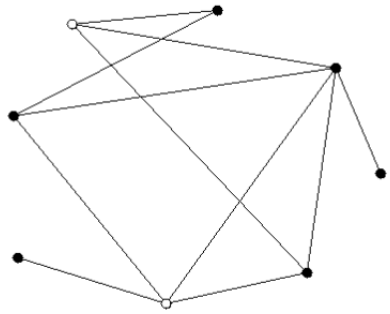
V práci som sa najskôr zaoberal teoretickou stránkou problému, uvedený je dôkaz NP-úplnosti. Následne som uvažoval možné prístupy k riešeniu - od hrubej sily, cez redukciu problému na lineárne programovanie a riešenie cez dostupné solvery až po praktické heuristiky. Všetky prístupy som nakoniec porovnal v rôznych testoch zameraných na rýchlosť výpočtu a presnosť výsledku.

1 Úvod do problematiky

Majme neorientovaný graf G s množinou vrcholov V a množinou hrán E (t.j. G = (V, E)). Vrcholy tohto grafu sú zafarbené buď na čierno, alebo na bielo. Zafarbeniu vrcholov hovoríme konfigurácia. Uvažujme teraz nasledovnú simuláciu na tomto grafe: V každom diskrétnom kroku sa každý vrchol prefarbí/ponechá si farbu podľa toho, ako je zafarbená väčšina jeho susedov. Využije pri tom nejaké pravidlo, napr. "seba nepočítam, v prípade zhody preferujem čiernu farbu". Rozhodovanie o prefarbení pritom robia v danom kroku všetky vrcholy súčasne.



Obr. 1.0.1: Počiatočná konfigurácia grafu



Obr. 1.0.2: Konfigurácia grafu po kroku simulácie

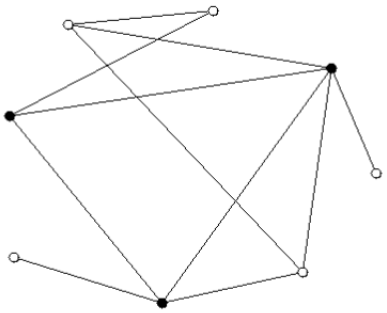
Uvedený typ simulácie (resp. výpočtu) sa volá Väčšinové hlasovanie [1] a spomínané pravidlo sa volá Jednoduchá väčšina [2]. Čierna farba v podstate symbolizuje, že vrchol vykazuje zlé

[1] Ang. Majority voting

[2] Ang. Simple majority, alebo tiež PB/SN (prefer black, self not including) [F]

(chybné) správanie, zatiaľ čo biela naopak.

Takáto simulácia môže skončiť (keď sa zopakuje tá istá konfigurácia dva krát po sebe), alebo sa zacykliť (konfigurácie sa budú periodicky opakovať). V prípade že skončí, zaujíma nás výsledná konfigurácia. Presnejšie, zaujíma nás, či na výstupe nemáme graf zafarbený kompletne na čierno [3]. V takom prípade hovoríme o katastrofickej počiatočnej konfigurácii [4] a o katastrofickom výpočte [5]. Samotná katastrofická konfigurácia však katastrofu ešte neznamená. Znamená len toľko, že sme v počiatočnej konfigurácii zle zvolili biele vrcholy.



Obr. 1.0.3: Ukážka zle zvolených bielych vrcholov. Je ich síce väčšina, no čierne vrcholy majú dominantnejšie postavenie a po 3 krokoch simulácie Majority voting PB/SN dôjde ku celočiernej konfigurácii. Počiatočná konfigurácia je teda katastrofická

Otázka však je, či v danom grafe vôbec možno zvoliť takú konfiguráciu, aby sa výpočet skončil a aby sme sa vyhli katastrofickému výpočtu. Pokiaľ áno, hovoríme, že v grafe G existuje imúnny podgraf G', ktorý je tvorený práve na bielo zafarbenými vrcholmi G.

Samozrejme, každý graf obsahuje imúnny podgraf, keďže vždy vieme zafarbiť na bielo celý graf. No je však žiadúce počet vrcholov minimalizovať.

1.1 Motivácia

Problém, ktorý som doposiaľ stručne popísal sa vyskytuje na pozadí viacerých praktických aplikácii. Samozrejme, my sa budeme zaoberať zovšeobecneným teoretickým modelom, no

[3] Ang. all-black configuration [KR]

[4] Ang. catastrophic configuration

[5] v ang. terminológii complete computation

stojí za to si spomenúť aspoň niektoré príklady reálnych situácii, ktoré sú motiváciou pre hľadanie minimálnych imúnnych podgrafov.

1.1.1 Distribuované systémy

Uvažujme sieť databáz v distribuovanom systéme. Systém si môžeme predstaviť ako graf - databázy budú vrcholy, susedstvo v rámci siete bude reprezentované hranou. Keďže databázy evidujú dáta dôležité pre vykonanie výpočtu v distribuovanom systéme, o validite svojich dát hlasujú v diskrétnych časových intervaloch. Každá databáza sa pýta susedných databáz, aké dáta obsahuje a podľa väčšiny sa rozhodne pozmeniť tie svoje.

Zrejmým problémom je, že chybné dáta sa môžu ľahko rozšíriť do celého systému (podobne ako nákaza), pokiaľ majú v systéme dominantné postavenie. Je teda snaha o nájdenie množiny databáz (resp. vrcholov grafu), ktorá zabezpečí, že systém neskolabuje úplne v dôsledku rozšírenia chybných dát do celého systému. Po pridaní požiadavky na minimálnosť takejto množiny je vidno úzky súvis s problémom hľadania minimálneho imúnneho podgrafu.

1.1.2 Futbalový tím

Predstavme si klasický futbalový (alebo kľudne aj iný) tím. Tréneri často rozmýšľajú, ako hráčov dostatočne dobre pripraviť na zápas. Jedná sa pritom o dvojaké hľadisko - príprava fyzická a psychická. Oveľa problematickejšia býva tá druhá časť. Ako správne vytvoriť atmosféru v šatni? Ako zabezpečiť, aby tým neprestal myslieť pozitívne?

Tím si môžme predstaviť ako graf - jednotliví hráči budú vrcholy a ich vzájomné vzťahy budú predstavovať hrany. Každému hráčovi by sme mohli tiež priradiť farbu podľa toho, či je jeho zmýšľanie pozitívne, alebo negatívne. To záleží nie len na jeho vlastnej povahe, ale tiež na jeho vzťahoch s rodinou, kamarátmi a pod. Tieto vzťahy s okolitým svetom však na teraz zanedbáme a budeme uvažovať len situáciu v tíme.

Naskytá sa teraz otázka, ako sa v čase vyvinie zmýšľanie tímu. Ak zostáva pozitívne, je zrejmé, že v nami definovanom grafe existuje skupina pozitívne farbených (zmýšľajúcich) vrcholov (hráčov), ktoré majú dostatočný vplyv na zvyšok grafu (tímu) a nedovolia tak prevládnutie negatívnej farby (nálady) v grafe (tíme). Inými slovami, uvedená skupina vrcholov v grafe vytvára imúnny podgraf. Ak by sa zmýšľanie tímu vyvinulo negatívnym smerom,

tréner by sa mal zrejme zamyslieť nad štruktúrou vzťahov v tíme a nad jej pozmenením - či už obmenou zostavy, alebo iným spôsobom tak, aby sa v tíme v podstate vytvoril imúnny podgraf.

1.1.3 Šírenie infekcie, poplašnej správy...

Je pravda, že na šírenie infekcie, či poplašnej správy zväčša netreba, aby ich šírila väčšina okolia, no aj tieto praktické problémy majú s problematikou imúnnych podgrafov úzky súvis. Napr. aj v prípade šírenia poplašnej správy je dôležité, aby sa vytvorila väčšia skupina ľudí čo poplašnej správe odoláva a nenechala sa ovplyvniť okolím.

1.2 Základné definície

Definícia 1.1. ***Konfigurácia na grafe G*** [6] *je funkcia priraďujúca každému vrcholu farbu z množiny {biela, čierna}.*

Definícia 1.2. ***Väčšinové hlasovanie***[7] *je nasledujúci výpočet na grafe G s nejakou počiatočnou konfiguráciou: V každom kroku výpočtu sa každý vrchol prefarbí na farbu, akú má väčšina jeho susedov. V prípade zhody preferuje čiernu farbu.* [8].

Poznámka 1.1. *Uvažujme ešte iné pravidlá výpočtu [F]. V prípade zhody môže vrchol preferovať jeho súčasnú farbu (PC/SN* [9]*), alebo môžeme do hlasovania započítavať aj samotný vrchol. Tak vznikajú nové pravidlá - PB/SI* [10]*, PC/SI* [11]*. Väčšinové hlasovanie s iným pravidlom ako so štandardným PB/SN budeme explicitne označovať, napr.* Väčšinové hlasovanie PC/SI

Definícia 1.3. ***Katastrofickým výpočtom*** *budeme označovať výpočet väčšinového hlasovania, ktorý sa skončí v konfigurácii kde sú všetky vrcholy zafarbené na čierno.*

[6] Väčšina definícii je prevzatých z [KR] a z [F].

[7] Ang. Majority voting

[8] Anglicky sa pravidlo volá Simple majority, alebo tiež PB/SN (prefer black, self not including)

[9] Ang. Prefer current, self not including

[10] Ang. Prefer black, self including

[11] Ang. Prefer current, self including

Nasledovať budú 3 definície imúnneho podgrafu (IP), odstupňované od "najmenej prísnej" po "najprísnejšiu".

Definícia 1.4. ***Imúnny podgraf (1) I grafu G*** *je podgraf, ktorý má nasledujúcu vlastnosť: Ak vrcholy I zafarbíme na bielo, ostatné vrcholy G zafarbíme na čierno a spustíme na tejto konfigurácii výpočet väčšinového hlasovania, výpočet neskončí katastrofickým výpočtom.*

Definícia 1.5. ***Imúnny podgraf (2) I grafu G*** *je podgraf, ktorý má nasledujúcu vlastnosť: Ak vrcholy I zafarbíme na bielo, ostatné vrcholy G zafarbíme na čierno a spustíme na tejto konfigurácii výpočet väčšinového hlasovania, výpočet* skončí a zároveň *neskončí katastrofickým výpočtom.*

Definícia 1.6. ***Imúnny podgraf (3) I(V', E') grafu G(V, E)*** *je taký podgraf, pre ktorý platí:*

$$\forall v \in V' : |Neigh(v) \bigcap V'| > |Neigh(v) \bigcap V \setminus V'| \tag{1.2.1}$$

kde $Neigh(v)$ je množina susedov vrcholu v v grafe G.
Takýto podgraf grafu G budeme volať ***stabilný imúnny podgraf****.*

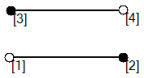
Podmienka imunity (1.2.1) hovorí, že vrcholy v imúnnom podgrafe majú nadpolovičnú väčšinu susedov v samotnom podgrafe. Znamená to teda, že pri zafarbení vrcholov imúnneho podgrafu na bielo sa počas výpočtu väčšinového hlasovania vrcholy imúnneho podgrafu neprefarbia na čierno.

Celkom zrejme definícia IP (2) implikuje definíciu IP (1) (t.j. podgraf spĺňajúci definíciu IP (2) spĺňa aj definíciu IP (1)). Ako je to s implikáciou (3) $\Longrightarrow$ (2) ? Ak zafarbíme vrcholy stabilného IP na bielo (ostatné na čierno), môžu v prvom kroku výpočtu nastať dve možnosti:

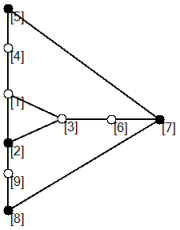
1. Výpočet skončí
2. Na bielo sa zafarbia ďalšie vrcholy. Tie sa však už nikdy neprefarbia späť na čierno, lebo majú nadpolovičnú väčšinu susedov zo stabilného IP (teda "večne biele vrcholy"). Tým pádom tieto vrcholy môžme pridať do stabilného IP a urobiť ďalší krok výpočtu.

Vidíme, že pri druhej možnosti sa musí pridať aspoň jeden vrchol do množiny (stabilne) bielych vrcholov. V konečnom čase sa teda vyskytne aj prvá možnosť, t.j. skončenie výpočtu pri zopakovaní konfigurácie. Teda platí implikácia definícia IP (3) $\Longrightarrow$ definícia IP (2).

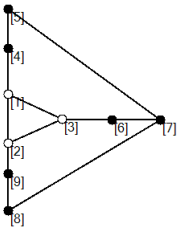
Opačné implikácie však platiť nemusia. Ilustrujú to nasledujúce príklady:



Obr. 1.2.1: Podgraf (na bielo zafarbené vrcholy) spĺňajúci definíciu IP (1). Výpočet sa však nezastaví, nie je teda splnená definícia IP (2)



Obr. 1.2.2: Podgraf spĺňa definíciu IP (2) (a (1)). Nie je to však stabilný IP.



Obr. 1.2.3: Podgraf spĺňa definíciu stabilného IP. Jeho výpočet skončí v prvom kroku.

Definícia 1.7. *Budeme hovoriť, že vrchol v v podgrafe $P(V', E')$ grafu $G(V, E)$* ***nemá imunitu, resp. je chybný*** *ak preň platí*

$$|Neigh(v) \bigcap V| \leq |Neigh(v) \bigcap V \setminus V'| \tag{1.2.2}$$

2 Dôležité výsledky

2.1 NP-úplnosť

Definícia 2.1. ***Trieda P problémov*** *je taká trieda problémov, ktoré sa dajú riešiť deterministickým algoritmom v polynomiálnom čase.*

Definícia 2.2. ***Trieda NP problémov*** *je taká trieda problémov, ktoré sa dajú riešiť nedeterministickým algoritmom v polynomiálnom čase.*

Z uvedeného celkom zrejme vyplýva inklúzia $P \subseteq NP$. To, či platí opačná inklúzia je však otvoreným problémom už desiatky rokov.

Definícia 2.3. *Problém L je* ***NP-ťažký problém*** *ak vieme ľubovoľný problém M z triedy NP polynomiálne redukovať na L. Polynomiálna redukcia znamená, že existuje algoritmus pracujúci v polynomiálnom čase, ktorý vie každý vstup problému M transformovať na vstup problému L tak, že keď vyriešime problém L na danom vstupe, dostaneme riešenie, ktoré vieme opäť polynomiálnym algoritmom transformovať späť na riešenie problému M.*

Je teda jasné, že riešenie akéhokoľvek NP-ťažkého problému v polynomiálnom čase by znamenalo, že vieme v polynomiálnom čase riešiť ľubovoľný problém z triedy NP.

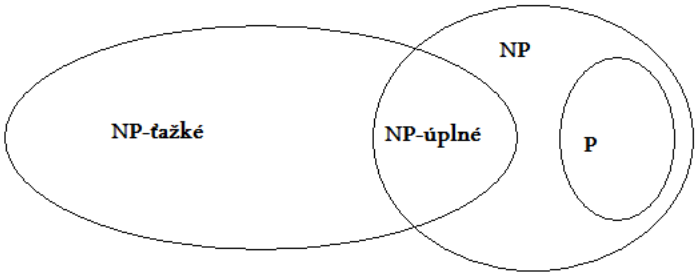
Definícia 2.4. *Problém L je* ***NP-úplný problém****, ak je z triedy NP a je NP-ťažký.*

Najjednoduchší spôsob, ako o nejakom probléme L ukázať, že je NP-úplný je dokázať, že nejaký iný NP-úplný problém M vieme polynomiálne redukovať na L. Potom ľubovoľný NP problém vieme redukovať nielen na M, ale tranzitívne aj na L. [12]

Definícia 2.5. *Problém* ***EC3S*** *(Exact cover by 3-sets*[13]*) je nasledujúci rozhodovací problém: Majme množinu* $A = \{a_1, a_2, \ldots, a_{3m}\}$ *a množinu* $S = \{S_1, S_2, \ldots, S_n\}$ *takú, že* $S_j \subseteq A$ *a* $|S_j| = 3$ *pre* $\forall j \in \{1, 2, \ldots, n\}$*. Existuje množina* $C = \{C_1, \ldots, C_m\}$ *taká, že* $C_i \in S$ *a* $\bigcup_{i=1}^{m} C_i = A$ *?*

[12] bohatšiu diskusiu o vzťahoch P a NP možno nájsť v [HR04]

[13] dokonalé pokrytie množinami mohutnosti 3



Obr. 2.1.1: Diagram znázorňujúci vzťahy tried P a NP

Intuitívne sa problém dokonalého pokrytia týka otázky: Máme čísla od 1 po 3m a množinu obsahujúcu trojice týchto čísel. Vieme vybrať presne m týchto trojíc tak, aby obsahovali všetky čísla od 1 po 3m? Ak áno, množinu $\{1, \ldots, 3m\}$ vieme pokryť dokonale (odtiaľ názov problému).

Napr. ak $A = \{1, 2, 3, 4, 5, 6\}$ a $S = \{\{2, 3, 6\}\{1, 3, 5\}\{2, 4, 5\}\{1, 4, 5\}\}$, vieme za C zvoliť $C = \{\{2, 3, 6\}\{1, 4, 5\}\}$ pričom vidno, že každé číslo z A nájdeme v 3-prvkových množinách C.

Tento problém je NP-úplný. Jeho zovšeobecnením dostaneme známy problém Dokonalého pokrytia [14].

Definícia 2.6. ***m-p-stop-stromom*** *nazveme graf, ktorý sa skladá z koreňa a 6m + p listov.*

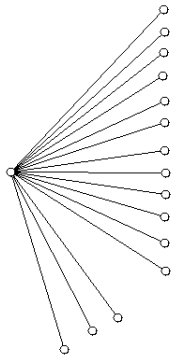
Veta 2.1. *Nasledujúci rozhodovací problém je NP-úplný pre všetky 3 definície IP: Existuje pre daný graf G a dané číslo k imúnny podgraf grafu G veľkosti maximálne k?*

Inšpiráciu pre tento dôkaz som hľadal v [KR].

Dôkaz. Dôkaz je robený redukciou problému EC3S na problém hľadania imúnneho podgrafu.

Majme inštanciu problému EC3S, teda $A = \{a_1, a_2, \ldots, a_{3m}\}$ a $S = \{S_1, S_2, \ldots, S_n\}$. Prípad, keď n < m je triviálny, keďže v takom prípade EC3S nemôže mať riešenie (množín v S nie je dosť aby pokryli A). Uvažujme teda $n \geq m$. Podľa týchto množín zostrojíme následovný graf G:

[14] Ang. Exact cover



Obr. 2.1.2: 2-3-stop-strom. Skladá sa z koreňa a 6m + p listov (v našom prípade m = 2 a p = 3)

1. Pre každú množinu S_j vytvoríme 3 vrcholy $s_{j,i}$ také, že $a_i \in S_j$. Tieto 3 vrcholy navzájom pospájame hranami a množinu S_j budeme chápať ako množinu týchto troch vrcholov. Množinu všetkých vrcholov $s_{j,i}$ budeme označovať ako S.

2. Pre každý prvok a_i množiny A vytvoríme vrchol a_i. Tento vrchol spojíme so všetkými vrcholmi $s_{j,k}$ takými, že k = i a $j \in \{1, \ldots, n\}$. Množinu vrcholov a_i označíme ako A.

3. Nech číslo MAXEDGE označuje maximum z počtu susedov, ktorých majú vrcholy z A v S. Vytvoríme kompletný graf o MAXEDGE vrcholoch. Vrcholy (označíme si ich $b_1, \ldots, b_{MAXEDGE}$) z tohto grafu budú tvoriť množinu B.

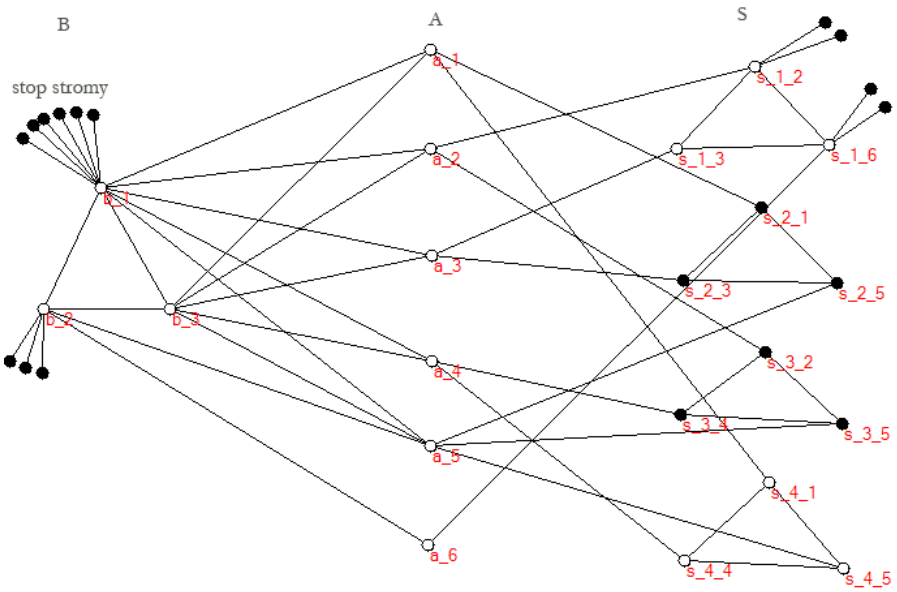
4. Každý vrchol a_i spojíme s toľkými vrcholmi z B (je jedno s ktorými), koľko má daný vrchol susedov v S.

5. Ku každému vrcholu z B a z S pripojíme k - 1 m-MAXEDGE-stop-stromov, kde k je stupeň daného vrchola.

Platí, že graf G má imúnny podgraf veľkosti 6m + MAXEDGE práve vtedy, keď daná inštancia problému EC3S je pozitívna. Poďme sa presvedčiť o platnosti oboch implikácii.

$\Longrightarrow$

G má imúnny podgraf veľkosti 6m + MAXEDGE podľa definície IP (1) (t.j. najslabšia podmienka). Ukážeme, že štruktúra imúnneho podgrafu zároveň predstavuje riešenie EC3S.



Obr. 2.1.3: Príklad zostrojeného grafu G pre inštanciu EC3S $A = \{1, 2, 3, 4, 5, 6\}$ a $S = \{\{2, 3, 6\}\{1, 3, 5\}\{2, 4, 5\}\{1, 4, 5\}\}$. Pre prehľadnosť boli znázornené len korene stop-stromov (aj to len niektorých). Na bielo zafarbené vrcholy tvoria imúnny podgraf veľkosti 6m + MAXEDGE, kde v tomto prípade m = 2 a MAXEDGE = 3.

Keďže v G existuje imúnny podgraf veľkosti 6m + MAXEDGE, existuje tam aj minimálny imúnny podgraf veľkosti $\leq$ 6m + MAXEDGE. Označme ho M. M určite neobsahuje žiaden z vrcholov m-MAXEDGE-stop-stromov, nakoľko takéto vrcholy by sa hneď v nasledujúcom kroku výpočtu prefarbili na čierno a čierne by už ostali (teda by existoval minimálnejší imúnny podgraf). Do úvahy teda prichádzajú len vrcholy z A, B a S.

Uvažujme situáciu, keď nejaký vrchol a_l nepatrí do M. V takom prípade sa však hneď v ďalšom kroku všetky vrcholy z B prefarbia na čierno. Vrchol a_l sa totiž pridá vo väčšinovom hlasovaní na "čiernu stranu" spolu so stop-stromami, s ktorými vrchol susedí, čím sa už nutne vytvorí väčšina. Po prefarbení vrcholov z B však bude mať každý vrchol z A aspoň polovicu susedov čiernych, čím sa prefarbia na čierno aj všetky vrcholy z A. Nasledovať budú vrcholy z S a výpočet sa teda skončí katastroficky.

Vieme teda, že všetky vrcholy z A musia patriť do M. Podobne sú na tom vrcholy z B. Tam opäť stačí prefarbiť jeden na čierno, a v nasledujúcom ťahu už budú čierne všetky. Ďalej

by nasledovali vrcholy z A a výpočet by sa skončil rovnako ako je popísané vyššie.

Keďže $|A| + |B| = 3m + MAXEDGE$, z množiny S môžme vybrať ešte maximálne 3m vrcholov do M tak, aby bol imúnny. Celkom zrejme treba brať vrcholy po trojiciach, v rámci jednej množiny S_j. Ak by sme z nejakej množiny S_j do M vybrali len 1 (alebo 2) vrchol, zvyšné 2 (alebo 1) by sa postarali v následujúcom kroku o prefarbenie vybratého vrchola (vrcholov) na čierno a existoval by menší imúnny podgraf M. Zároveň však treba vybrať vrcholy z S do M tak, aby každý vrchol z A susedil aspoň s jedným vybraným vrcholom v S. V opačnom prípade by mal nejaký vrchol a_l aspoň polovicu susedov (smerom do S) čiernu, t.j. prefarbil by sa na čierno aj on sám a nastal by vyššie spomínaný prípad. Keďže každý vrchol z S má iba jedného suseda v A, treba vybrať presne 3m vrcholov z S do M, t.j. presne m množín S_j. Výber týchto množín tvorí riešenie inštancie EC3S, keďže tieto množiny musia pokrývať A.

$\Longleftarrow$

Za vrcholy imúnneho podgrafu zvolíme všetky vrcholy z B, z A a z $C_i \in C$, kde C je riešenie EC3S inštancie (viď obrázok). Výsledný imúnny podgraf je stabilný, t.j. spĺňa všetky tri definície IP. □

Poznámka 2.1. *V prípade výpočtu väčšinového hlasovania PC/SN by sme vedeli dôkaz jednoducho upraviť. K vrcholom z B a S by sme pripojili o jeden stop strom viac než sa žiada v bode č. 5 zostrojovania grafu G a každý vrchol z A by sme spojili s ešte jedným vrcholom z B. Situácia pri pravidle PB/SI by bola analogická (i keď tieto pravidlá nie sú úplne totožné). Podobne by sa riešil prípad s pravidlom PC/SI. Rozdiely oproti klasickému pravidlu PB/SN by boli tiež v definícii stabilného imúnneho podgrafu. Napr. na stabilitu imúnneho podgrafu pri výpočte s pravidlom PC/SN by stačilo*

$$\forall v \in V' : |Neigh(v) \bigcap V'| \geq |Neigh(v) \bigcap V \setminus V'| \tag{2.1.1}$$

Problém hľadania minimálneho imúnneho podgrafu (MIP) je teda NP úplný pre všetky tri definície imúnneho podgrafu, ktoré boli uvedené. V ďalšom sa však budeme zaoberať, ak nebude uvedené ináč, iba stabilnými imúnnymi podgrafmi. Zároveň budeme uvažovať iba spojité grafy, nakoľko nespojité grafy môžeme rozbiť na komponenty, zistiť na každom MIP a tento IP vrátiť ako výstup.

3 Prístupy k riešeniu

Z predošlej kapitoly vyplýva, že riešiť efektívne a presne problém MIP je vopred prehratý súboj minimálne dokiaľ by sa neukázala platnosť P=NP. V prípade NP-úplných problémov sa preto uvažujú alternatívne formy hľadania riešení - aproximačné algoritmy, ktoré vedia zaručiť odchýlku od optima do nejakej hodnoty, heuristiky ktoré voľbou vhodnej stratégie hľadajú v priestore riešení to najvhodnejšie a niekedy aj pravdepodobnostné algoritmy využívajúce opakovanie výpočtov pre upresnenie riešenia [HR04].

Riešiť problém MIP exaktne je samozrejme možné, no len pre malé grafy. Uvádzam však aj takýto prístup, najmä kvôli porovnaniu rýchlosti výpočtov a presnosti výsledkov na malých grafoch.

V tejto sekcii budem postupne prezentovať jednotlivé možnosti, ako pristupovať k riešeniu problému MIP. Všetky tieto prístupy sú zároveň implementované v programe Immune Subgraph Analyzer (ISA), ktorý popisujem v poslednej kapitole 5 a ktorý je súčasťou tejto bakalárskej práce.

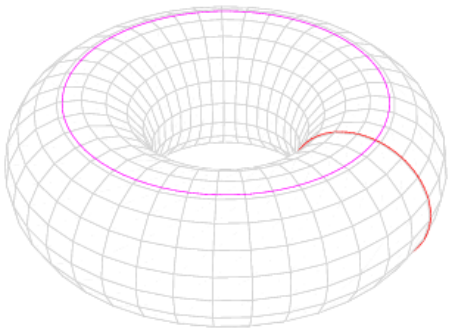
3.1 Štandardná sada testov

Aby som mohol jednotlivé algoritmy navzájom porovnávať, na každom som spúšťal štandardnú sadu testov pozostávajúcu z asi 30 grafov. Ak to nestačilo (napr. na určenie hornej hranice veľkosti grafu, na ktorom výpočet ešte trvá prijateľný čas), púšťal som pre daný algoritmus ďalšie testy.

Problém je, že pre veľké náhodné grafy (rádovo stovky vrcholov) nemáme možnosť zistiť presnú veľkosť minimálneho IP, aby sme mohli porovnať ako veľmi sa vypočítaná hodnota líši od optima. Do testovacej sady som preto zaradil aj niekoľko grafov špeciálnych typov, ktorých optimálna hodnota bola teoreticky zistená, alebo aspoň ohraničená zhora [KR]. Skôr než si teda predstavíme testovaciu sadu, aspoň neformálne načrtnem, ako vyzerajú tieto špeciálne grafy.

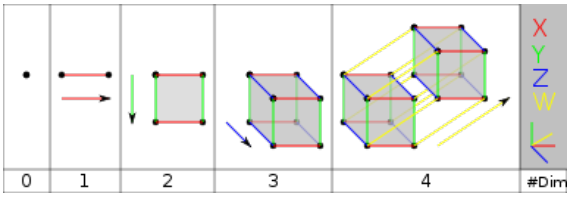
Torus dimenzie d_1 a d_2 je graf, ktorý vyzerá ako koláč s dierou v strede. Jedna dimenzia

udáva počet horizontálnych cyklov, druhá počet vertikálnych cyklov. [15]



Obr. 3.1.1: Torus

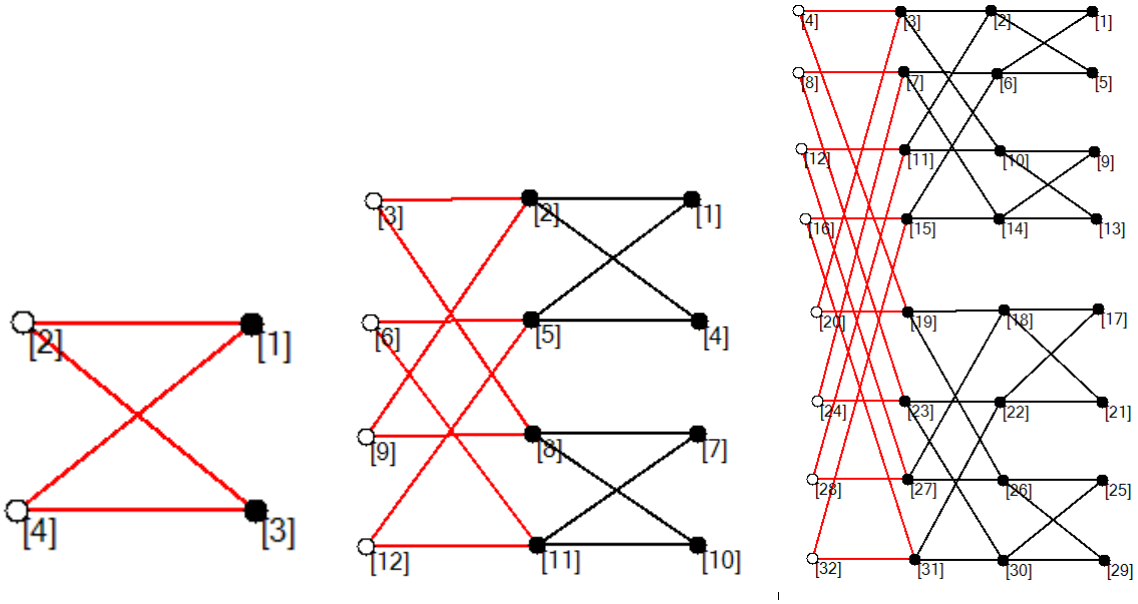
Hypercube (hyperkocka) dimenzie d je graf, ktorý vznikne naklonovaním dvoch hyperkociek dimenzie $d-1$ a ich spojením cez dvojice naklonovaných vrcholov. Hyperkocka dimenzie 0 je bod, dimenzie 1 úsečka, dimenzie 2 štvorec, dimenzie 3 kocka a dimenzie 4 kocka s "vnorenou" podkockou, atď. [16]



Obr. 3.1.2: Proces tvorby hyperkocky

Butterfly (motýľ) dimenzie d je graf, ktorý vznikne naklonovaním dvoch motýľov dimenzie $d-1$, pridaním 2^d vrcholov a špeciálnym spojením týchto motýľov cez pridané vrcholy. Motýľ dimenzie 0 je bod. Princíp spájania by mal byť zrejmý z následujúcich obrázkov. [17]

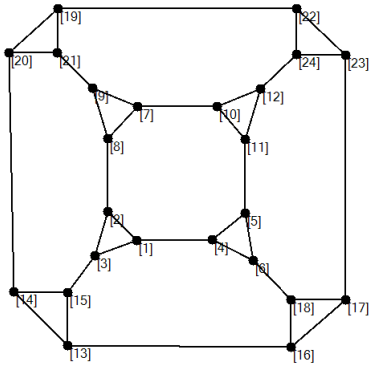
[15] presnejšiu definíciu možno nájsť v [WTR]
[16] presnú definíciu možno nájsť v [WHY1] a [WHY2]
[17] presnejšiu definíciu možno nájsť v [WBF]



(a) Butterfly dimenzie 1 (b) Butterfly dimenzie 2 (c) Butterfly dimenzie 3

Obr. 3.1.3: Spájanie menších motýľov do motýľa väčšej dimenzie. Červenou sú zvýraznené spájajúce hrany, bielou pridané vrcholy

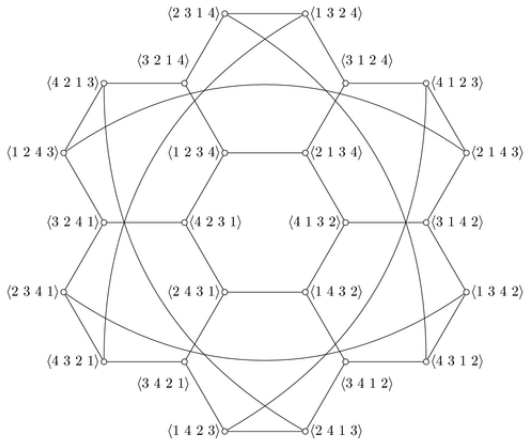
Cube connected cycle (CCC) dimenzie d je hyperkocka dimenzie d, ktorej vrcholy nahradíme cyklom veľkosti d. Pre ilustráciu uvádzam obrázok. [18]



Obr. 3.1.4: CCC dimenzie 3

[18] presnejšiu definíciu možno nájsť v [WCC]

Star (hviezda) dimenzie d je graf ktorého vrcholy sú všetky možné permutácie d prvkov a hrana z jednej permutácie ide do každej, ktorú môžeme získať tak, že prehodíme prvý prvok s ľubovoľným iným. [19]



Obr. 3.1.5: Star dimenzie 4. Autor: Anthony Labarre <http://homepages.ulb.ac.be/ alabarre/home.html>

Veta 3.1. *Na jednotlivých typoch grafov boli teoreticky zistené nasledujúce veľkosti minimálneho IP.* [20]

- ***Torus dimenzie*** d_1**,** d_2 $|minIP| = 2 \cdot min(d_1, d_2)$
- ***Butterfly dimenzie*** d $|minIP| \leq 2 \cdot \lceil \frac{d+1}{2} \rceil$
- ***Hypercube dimenzie*** d $|minIP| = 3 \cdot 2^{\lfloor (d+1)/2 \rfloor}(4 - 2^{dmod2}) - 8$
- ***Cube Connected Cycle dimenzie*** d $|minIP| = 8 \;\; d \geq 8$
- ***Star dimenzie*** d $|minIP| \leq \lceil \frac{d+2}{2} \rceil !$

[19] presnú definíciu možno nájsť v [WST]
[20] dôkaz je uvedený v [KR]

Štandardná sada testov potom vyzerá takto.

Tabuľka 3.1.1: Štandardná sada testov

Testovaný graf		
Typ grafu	**Veľkosť**	**Hustota hrán**
tori, butterfly, random...	-	*perc.*
tori (5, 10)	50	8
tori (7, 15)	135	3
tori (14, 20)	280	1
tori (20, 40)	800	<1
tori (30, 50)	1500	<1
tori (60, 80)	4800	<1
butterfly (4)	80	4
butterfly (6)	448	<1
butterfly (9)	5120	<1
ccc (5)	160	1
ccc (7)	896	<1
ccc (9)	4608	<1
hypercube (5)	32	16
hypercube (7)	128	5
hypercube (9)	512	1
hypercube (12)	4096	<1
star (3)	6	40
star (5)	120	3
star (7)	5040	<1
random	20	20
random	30	80
random	50	40
random	80	90
random	140	30
random	350	50
random	550	60
random	800	40
random	1300	70
random	2000	80

3.2 Riešenie hrubou silou

Za backtrack sa zväčša označuje algoritmus, ktorý prehľadá postupne a systematicky celý priestor riešení a vyberie najlepšie riešenie. Podobne to funguje aj v tomto prípade. Backtrack hľadajúci minimálny (stabilný) imúnny podgraf jednoducho preskúša všetky podmnožiny vrcholov daného grafu a overí, či je podgraf tvorený týmito vrcholmi imúnny, teda skontroluje preň podmienku imunity (1.2.1). Backtrack potom vráti minimálny takýto podgraf. V pseu-

dokóde [21] by algoritmus vyzeral takto.

Algorithm 1 Backtrack

Vstup: graf G

```
I = G
while ((P = ďalšiaPodmnožina(G)) ≠ ∅) do
  if ((jePodgrafImmúnny(P, G)) ∧ (|I| > |P|)) then
    I = P
  end if
end while
```

Výstup: I

V nasledujúcej tabuľke uvádzam výsledky Backtracku - z pohľadu rýchlosti aj presnosti.

Tabuľka 3.2.1: Výsledky Backtracku

Testovaný graf			**Výsledky**		
Typ grafu	**Veľkosť**	**Hustota hrán**	**Veľkosť**	**Čas**	**Úspešnosť**
tori, butterfly, random...	-	*perc.*	-	*h:m:s*	*skutočne imúnny?*
star (3)	6	40	6	00:00:00	OK
random	20	20	8	00:00:01.2950000	OK

Detailnejšie som preskúšal grafy veľkostí od 21 po 28 a bol badateľný exponenciálny nárast času výpočtu. Keďže počet podmnožín grafu sa s rastúcim počtom vrcholov zdvojnásobuje a algoritmus preskúmava všetky podmnožiny, je hlavný faktor vo funkcii času výpočtu 2^n. Samotné preskúšanie imunity podmnožiny je rádovo až n^2. Celkovo sa dá čas Backtraku odhadnúť $O(2^n \cdot n^2)$.

Tabuľka 3.2.2: Detaily výsledkov - exponenciálny nárast

Veľkosť grafu	**Čas**
21	00:00:02.7340000
22	00:00:06.9710000
23	00:00:18.9550000
24	00:00:33.8520000
25	00:01:23.2940000
26	00:02:37.4940000
27	00:04:12.5520000
28	00:08:22.2800000

[21] Vo výpisoch algoritmov, ktoré v tejto práci uvádzam je ako jazyk použitá matematika a niektoré základné riadiace štruktúry. Aby však algoritmy neboli príliš komplikované, vybrané časti algoritmov sú schované do funkcii, ktorých obsah by si čitateľ isto vedel sám predstaviť.

Keďže Backtrack strávil už pri počítaní imúnneho podgrafu grafu veľkosti 28 takmer 8 a pol minúty a v teste sa vyskytovali ešte väčšie grafy, nemalo zmysel strácať hodiny výpočtového času rátania IP hrubou silou. Vhodnou technikou (napr. $\alpha - \beta - osekávanie$) by sme vedeli Backtrack vylepšiť tak, aby zvládal aj väčšie vstupy. Ak však máme použiť hrubú silu, lepšou voľbou sa javí byť konverzia problému do systému lineárnych rovníc a jeho riešenie cez dostupné a prepracované solvery a heuristiky.

3.3 Riešenie cez lineárne programovanie

NP-úplné problémy majú (už z definície) vlastnosť, ktorá hovorí, že vieme jeden NP-úplný problém redukovať na iný. Ide o ten istý princíp, ktorý sme využili napr. pri dôkaze NP-úplnosti problému imúnnych podgrafov (vt. 2.1). Niektoré NP-úplné problémy vystupujú do popredia viac než ostatné. V informatických kruhoch sa nájde len minimum ľudí, ktorí ešte nepočuli o probléme obchodného cestujúceho, alebo o probléme SAT. Známe sú predovšetkým preto, že sa dajú na ne redukovať mnohé iné problémy (alebo sa, naopak, opačná redukcia používa pri dôkazoch NP-úplnosti). Nie každá redukcia medzi dvoma NP-úplnými problémami je však jednoduchá.

Ako vhodný kandidát pre transformáciu problému MIP je lineárne programovanie (LP). V lineárnom programovaní, stručne povedané, ide o maximalizáciu (duálne minimalizáciu) daného cieľa za predpokladu istých obmedzení. Formálne sa vstup lineárneho programovania skladá z dvoch častí:

- výraz, ktorý máme optimalizovať, napr. Minimalizuj $c_1x_1 + c_2x_2$
- obmedzenia v tvare systému lineárnych nerovníc, napr. :

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 &\geq b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 &\geq b_2 \\ a_{3,1}x_1 + a_{3,2}x_2 &\geq b_3 \\ x_i &\geq 0 \ \ \forall i \in \{1,2\} \end{aligned}$$

Ide teda o to zvoliť tzv. *rozhodovacie premenné* x_i čo najvhodnejšie tak, aby sme neporušili obmedzenia a súčasne dosiahli maximum/minimum. Celý proces sa dá taktiež chápať ako hľadanie extrémalneho bodu v priestore ohraničenom daným systémom lineárnych nerovníc.

Pokiaľ ako doménu pre premenné uvažujeme reálne čísla, vieme dokonca riešenie dostať v polynomiálnom čase. Problém nastane ak chceme uvažovať iba celočíselné riešenia. Vtedy sa problém stáva ťažší a dá sa riešiť už len pre malé hodnoty n (počet premenných). Keďže je však tento problém veľmi známy a zovšeobecňujú sa naňho mnohé iné problémy, boli preň urobené výkonné solvery. Pokúsime sa teda riešiť problém MIP prostredníctvom redukcie na celočíselné LP a jeho následné riešenie cez vhodný dostupný solver.

3.3.1 Prepis do systému lineárnych nerovníc

Majme graf $G(V, E)$ kde $V = \{1, 2, ..., n\}$. Pre každý vrchol i zaveďme rozhodovaciu premennú x_i. Každý podgraf P v G potom môžeme chápať ako priradenie hodnoty 0 (vrchol nepatrí do P) alebo 1 (vrchol patrí do P) premenným x_i. Keďže nám ide o minimálny podgraf, chceme minimalizovať:

$$\sum_{i=1}^{n} x_i \tag{3.3.1}$$

pričom obmedzenia zatiaľ činia:

$$0 \leq x_i \leq 1 \ \ \forall i \in \{1, ..., n\} \tag{3.3.2}$$

Treba však ešte zabezpečiť imunitu:

$$\sum_{j \in Neigh(i)} x_j > \sum_{j \in Neigh(i)} (1 - x_j) \ \ \forall i \in \{1, ..., n\} \tag{3.3.3}$$

Takáto podmienka by však požadovala aj imunitu vo vrcholoch mimo imúnny podgraf. Upravíme ju teda na následovnú

$$x_i((\sum_{j \in Neigh(i)} x_j) - 1) \geq x_i(\sum_{j \in Neigh(i)} (1 - x_j)) \ \ \forall i \in \{1, ..., n\} \tag{3.3.4}$$

Stále tu však máme jeden problém. Takouto podmienkou totiž nenútime prítomnosť akéhokoľvek podgrafu. Minimalizovať (3.3.1) by znamenalo nepoužiť do IP ani jeden vrchol. Musíme teda ešte pridať

$$\sum_{i=1}^{n} x_i \geq 1 \tag{3.3.5}$$

Pre každý graf (vstup problému minimálneho IP) teda vieme zostrojiť adekvátny systém lineárnych nerovníc s rozhodovacími premennými x_i ktoré prislúchajú jednotlivým vrcholom

(vstup pre lineárne programovanie). Čisto matematický zápis však ešte treba previesť do syntaxe, ktorej bude rozumieť počítačový solver.

3.3.2 Prepis do syntaxe AMPL

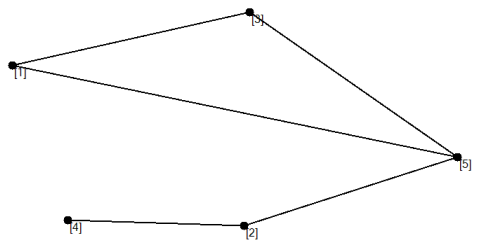
Voľne dostupný na stiahnutie je AMPL/CPLEX solver v študentskej edícii, ktorý je limitovaný na 300 rozhodovacích premenných. Pre naše potreby to celkom stačí. Zadanie problému lineárneho programovania v AMPL sa delí na dve časti

1. **model** - je spoločný pre všetky vstupy a špecifický pre riešený problém. V našom prípade model zabezpečuje imunitu grafu G. To, aký je graf G však model neuvádza.
2. **dáta** - je špecifický pre daný vstup, v našom prípade teda opisuje graf.

AMPL sám o sebe je len jazyk, v ktorom sa napíše model a dátový súbor. C-PLEX je potom solver, ktorý rozumie syntaxi AMPL a hľadá riešenie. O syntaxi AMPL sa tu nebudem zmieňovať, jej popis ako aj praktické ukážky použitia možno nájsť v [F+02]. Uvediem však, ako v našom prípade vyzerá model, a ako vyzerajú dáta:

```
1  set VERTICES;
2
3  param neigh {VERTICES, VERTICES};
4  param needed_deg {VERTICES};
5
6  var Used {VERTICES} integer >= 0, <= 1;
7
8  minimize Number_Vertices: sum {v in VERTICES} Used[v];
9
10 subject to Immunity {v in VERTICES}:
11 sum {n in VERTICES} Used[n] * neigh[v, n] >= needed_deg[v]*Used[v];
12
13 subject to AtLeastOne:
14 sum {v in VERTICES} Used[v] >= 1;
```

Listing 1: Model pre celočíselné riešenie problému MIP v LP



Obr. 3.3.1: Graf, ktorého dátový súbor je na nasledujúcom výpise kódu

```
1  set VERTICES := 1 2 3 4 5
2  ;
3
4  param neigh: 1 2 3 4 5 :=
5  1   0 0 1 0 1
6  2   0 0 0 1 1
7  3   1 0 0 0 1
8  4   0 1 0 0 0
9  5   1 1 1 0 0
10 ;
11
12 param: needed_deg :=
13 1 2
14 2 2
15 3 2
16 4 1
17 5 2
18 ;
```

Listing 2: Dáta grafu z obrázku

Dvojrozmerné pole *neigh* určuje, ktoré dva vrcholy grafu sú prepojené hranami. Pole *needed_deg* zase špecifikuje pre každý vrchol počet susedov, ktoré musia byť zafarbené na bielo aby bol daný vrchol imúnny. Zápis obmedzení cez lineárne nerovnice v syntaxi AMPL teda vyzerá trochu ináč ako v matematickej notácii, no vyjadruje to isté.

Všimnite si v definícii premenných Used kľúčové slovo *integer*. Práve vďaka tomuto slovu sa celý problém stáva výpočtovo tak náročným. Jeho odstránením dostávame problém, ktorý sa rieši rýchlo, v polynomiálnom čase. Dostávame tzv. relaxované riešenie, kde premenné Used (analogicky x_i v matematickej notácii) môžu nadobúdať ľubovoľnú hodnotu od 0 po 1. Naskytuje sa teda otázka, čo takéto riešenie predstavuje, a či aspoň z časti aproximuje skutočné riešenie.

Keďže má prostredie riešiace problémy v jazyku AMPL rozhranie príkazového riadku, nebolo zložité celý proces vytvorenia dát pre daný graf, spustenia výpočtu a extrakcie výsledkov zautomatizovať a integrovať ho tak ako kompaktný algoritmus do programu ISA (sekcia 5).

3.3.3 AMPL Integral

Ako AMPL Integral som nazval prípad, keď uvažujeme iba celočíselné riešenie, teda keď je problém NP-úplný. Ako už bolo povedané, ide o prístup, keď sa snažíme riešiť problém minimálneho IP akoby nepriamo; Najprv ho transformujeme na všeobecnejší problém. Tým však strácame možnosť využiť špecifické črty problému MIP a vybudovať podľa nich presný a rýchly algoritmus (aj tak by sme však samozrejme našli hranicu, za ktorou by už výpočet trval neúnosne dlho). Na druhej strane, nakoľko sa na všeobecnejší problém redukuje mnoho iných, je žiadúce vedieť ho dobre riešiť. Preto sú naň známe výkonné solvery, ktoré vedia túto stratu spôsobenú transformáciou problému dostatočne vykompenzovať.

Solver C-PLEX považujeme za čiernu skrinku, preto je následovný algoritmus skôr zoznamom krokov, ktoré treba spraviť pre riešenie minimálneho IP cez lineárne programovanie.

Algorithm 2 AMPL Integral

Vstup: graf G

vytvor z grafu G dátový súbor
spusti proces AMPL-CML (command line)
načítaj model pre celočíselné riešenie minimálneho IP a dáta grafu G
spusti výpočet solveru C-PLEX na danom modeli a dátach
preparsuj výsledky a zisti imúnny podgraf I

Výstup: I

No a ako na testovacích vstupoch obstál AMPL Integral?

Tabuľka 3.3.1: Výsledky AMPL Integral

Testovaný graf			**Výsledky**		
Typ grafu	**Veľkosť**	**Hustota hrán**	**Veľkosť**	**Čas**	**Úspešnosť**
tori, butterfly, random...	-	*perc.*	-	*h:m:s*	*skutočne imúnny?*
tori (5, 10)	50	8	10	00:00:00.0970000	OK
tori (7, 15)	135	3	14	00:00:00.2640000	OK
tori (14, 20)	280	1	28	00:00:00.8780000	OK
butterfly (4)	80	4	28	00:00:00.1940000	OK
ccc (5)	160	1	5	00:00:00.0860000	OK
hypercube (5)	32	16	8	00:00:00.0580000	OK
hypercube (7)	128	5	16	00:00:00.1590000	OK
star (3)	6	40	6	00:00:00.0940000	OK
star (5)	120	3	24	00:00:00.2740000	OK
random	20	20	8	00:00:00.1660000	OK
random	30	80	15	00:00:00.9260000	OK
random	50	40	22	00:00:05.1500000	OK

Algoritmus som testoval aj detailnejšie, no grafy s väčším počtom vrcholov ako 55 som už neskúšal, nakoľko by výpočet trval príliš dlho. Pri grafe veľkosti 55 sa bola doba výpočtu takmer minútu a 41 sekúnd. Napriek tomu je však AMPL Integral obrovským zlepšením oproti Backtracku, ktorý podobne dlho počítal IP grafu veľkosti pod 30 vrcholov. Uveďme si pre názornosť výsledky pre grafy s veľkosťami do 55.

Tabuľka 3.3.2: Detaily výsledkov AMPL Integral pre grafy do 55 s hustotou hrán 20 - 80 %

Veľkosť grafu	**Čas**
49	00:00:11.5380000
50	00:00:32.5000000
51	00:00:16.8730000
52	00:00:44.1060000
53	00:00:32.1690000
54	00:01:00.0700000
55	00:01:40.4910000

Rovnako ako v prípade Backtracku je možné badať exponenciálny rast, no nie úplne priamočiary - heuristiky v C-PLEX solveri môžu pre niektoré väčšie grafy nájsť riešenie skôr ako u menšieho grafu.

Čo je však zaujímavé je, že AMPL Integral bez problémov rýchlo vypočítal hodnotu optima pre torus dimenzií 14/20, čo je graf veľkosti 280 s veľmi nízkou hustotou hrán. Skúsil som teda uskutočniť ešte jeden test s grafmi s 10 % hustotou hrán, tu sú jeho výsledky.

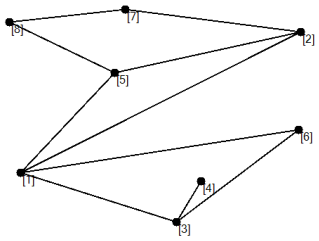
Tabuľka 3.3.3: Detaily výsledkov AMPL Integral pre grafy do 85 s hustotou hrán 10 %

Veľkosť grafu	Čas
45	00:00:00.0920000
50	00:00:00.1840000
55	00:00:00.2690000
60	00:00:00.3630000
65	00:00:01.3810000
70	00:00:04.0670000
75	00:00:06.4890000
80	00:00:17.4090000
85	00:01:32.9570000

Ak by sme hustotu hrán držali blízko pri nule, je možné, že by AMPL Integral počítal minimálny IP celkom rýchlo aj pre väčšie hodnoty. Pre aké však nezistíme, kvôli obmedzeniu danému študentskou verziou solvera C-PLEX.

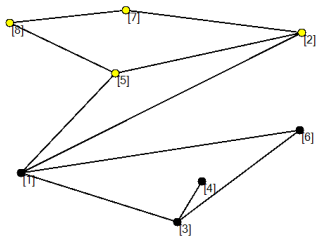
3.3.4 AMPL Relaxed

V AMPL relaxed nedávame požiadavku na celočíselné riešenie. To znamená, že vrcholy do IP sa akoby vyberali s určitou mierou, ktorá sa môže meniť z vrchola na vrchol. Celkovo sa potom AMPL Relaxed algoritmu vždy podarilo minimalizovať podmienku (3.3.1) na hodnotu 1. To však neznamená, že bol vybratý do IP iba jeden vrchol. Za vybraté budeme považovať všetky, ktorých prislúchajúca premenná je nenulová. Uvažujme následovný graf G.



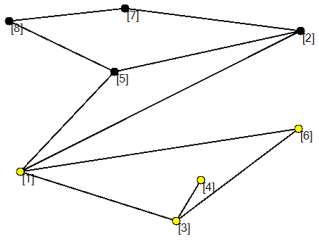
Obr. 3.3.2: Graf G

Jeho minimálny imúnny podgraf je tvorený horným štvoruholníkom.



Obr. 3.3.3: Imúnny podgraf grafu G

Imúnny podgraf, ktorý nájde AMPL Relaxed je však následovný.



Obr. 3.3.4: Imúnny podgraf grafu G, vypočítaný AMPL Relaxed algoritmom

Tento podgraf vznikol preparsovaním výstupu zo solvera C-PLEX a nájdením vrcholov, ktoré majú nenulovú hodnotu.

```
1  ampl: reset; option solver cplex; model ../Models/immune_subgraphs.mod; data ../Data
       /AMPL_ALGORITHM_DAT.dat; solve;
2  CPLEX 11.2.0: optimal solution; objective 1
3  4 dual simplex iterations (0 in phase I)
4  ampl: display Used;
5  Used [*] :=
6  1   0.111111
7  2   0
8  3   0.333333
9  4   0.333333
10 5   0
11 6   0.222222
12 7   0
13 8   0
14 ;
```

Listing 3: Relaxované riešenie za použitia C-PLEXu

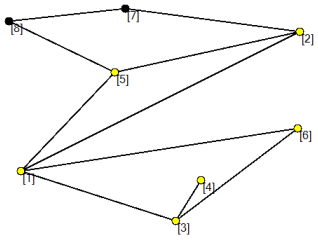
Očividným problémom však je, že graf nie je imúnny. Imunitu porušuje vrchol s číslom 1. Je toto pravidelný jav? Je algoritmus aspoň rýchly? Odpoveď na túto otázku možno nájsť v nasledujúcej tabuľke.

Tabuľka 3.3.4: Výsledky AMPL Relaxed

Testovaný graf			**Výsledky**		
Typ grafu	**Veľkosť**	**Hustota hrán**	**Veľkosť**	**Čas**	**Úspešnosť**
tori, butterfly, random...	-	*perc.*	-	*h:m:s*	*skutočne imúnny?*
tori (5, 10)	50	8	12	00:00:00.0490000	FAIL
tori (7, 15)	135	3	12	00:00:00.0700000	FAIL
tori (14, 20)	280	1	12	00:00:00.1270000	FAIL
butterfly (4)	80	4	11	00:00:00.0520000	FAIL
ccc (5)	160	1	6	00:00:00.1140000	FAIL
hypercube (5)	32	16	8	00:00:00.0470000	OK
hypercube (7)	128	5	16	00:00:00.0650000	OK
star (3)	6	40	6	00:00:00.0860000	OK
star (5)	120	3	29	00:00:00.0720000	OK
random	20	20	10	00:00:00.0520000	FAIL
random	30	80	15	00:00:00.0640000	FAIL
random	50	40	24	00:00:00.0530000	FAIL
random	80	90	40	00:00:00.0670000	FAIL
random	140	30	64	00:00:00.1330000	FAIL

Ako možno vidieť, algoritmus je rýchly, no nedá sa povedať, že by bol presný. Na druhej strane, napríklad podgraf, ktorý našiel algoritmus na obrázku 3.3.4 nemá od imunity ďaleko. Vrcholy sú pokope, väčšina má imunitu no jeden vrchol je chybný.

Ponúka sa teda otázka, či nie je možné množinu nájdenú algoritmom AMPL Relaxed jednoducho doplniť o niekoľko ďalších vrcholov, ktoré ošetria chybné vrcholy tak, aby bol podgraf imúnny. Napr. podgraf z obrázku 3.3.4 obsahuje jeden chybný vrchol, ktorý by sme mohli ošetriť pridaním vrcholov č. 2 a 5 do IP.

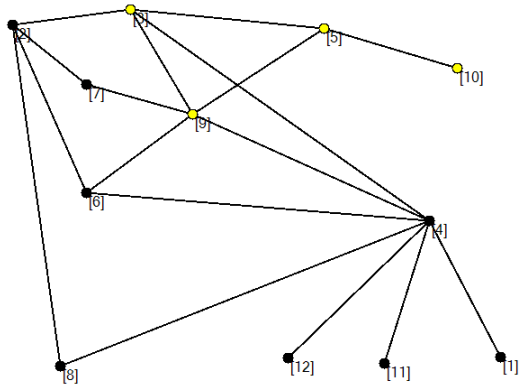


Obr. 3.3.5: Doplnený podgraf z algoritmu AMPL Relaxed, ktorý už je imúnny

Po hrubej sile a riešení cez LP sa teda dostávame k tretiemu prístupu k riešeniu problému MIP - určenie počiatočnej množiny vrcholov, ktorú heuristicky doplníme na IP. Jednou možnosťou bude doplňovanie množiny nájdenej AMPL Relaxed algoritmom. Najprv sa však pozrieme na doplňovacie heuristiky.

3.4 Heuristiky doplňovania IP

Uvažujme následovný graf G.



Obr. 3.4.1: Graf G. Na žlto je zafarbený výstup AMPL Relaxed pre tento graf

Na žlto je v grafe G zafarbený výstup AMPL Relaxed. Tento podgraf však nie je imúnny. Obsahuje totiž hneď dva vrcholy, ktoré nemajú imunitu (čísla 3 a 9). Na tomto grafe si ukážeme tri stratégie dopĺňania vrcholov do existujúcej množiny s cieľom výslednej imunity. Podľa kvality výsledku boli jednotlivé stratégie označené, podobne ako stupeň výbavy automobilov, ako Standard, Premium a Superb.

3.4.1 Complete Standard

Complete Standard (CS) funguje na princípe dopĺňania vrcholov po jednom, až kým nie je podgraf imúnny. Celá stratégia sa teda redukuje na výber čo najvhodnejšieho kandidáta na nový vrchol do podgrafu. V prvom rade treba zistiť, ktorý vrchol z aktuálne vybraného podgrafu nemá imunitu. Nemá totiž zmysel dopĺňať do IP vrcholy, ktoré nesusedia s chybným

vrcholom. V prípade väčšieho počtu chybných vrcholov sa však treba rozhodnúť ktorý z nich pôjdeme ošetriť. Tu sa CS rozhoduje na základe tzv. imúnnej diferencie vrcholov.

Definícia 3.1. ***Imúnna diferencia*** *vrchola v $ID(v)$ v grafe $G(V, E)$ s konfiguráciou K je rozdiel*

$$|Neigh(v) \cap \{w \in V | K(w) = biela\}| - |Neigh(v) \cap \{w \in V | K(w) = čierna\}| \qquad (3.4.1)$$

teda rozdiel počtu bielych a čiernych susedov.

Pri výbere chybného vrchola z aktuálneho podgrafu preferuje CS vrchol, ktorého imúnna diferencia je čo najväčšia, ale zároveň menšia ako 1 (ináč by vrchol nebol chybný). Dôvod je, neformálne povedané, "ošetri to čo ide najľahšie, možno pri tom ošetríš aj niečo iné". No a ako ošetriť chybný vrchol? Vrchol nemá imunitu preto, lebo väčšina (alebo presne polovica) jeho susedov zatiaľ nie sú v IP. Treba teda opäť vybrať jedného zo susedov a prefarbiť ho na bielo. Pri výbere suseda treba opäť dbať na imúnnu diferenciu, aby bola čo najväčšia. Nový vrchol by mal totiž so sebou priniesť čo najmenej problémov. Po pridaní nového vrchola do IP treba ešte aktualizovať hodnoty ID pre jeho susedov - ich ID sa totiž zvýši o 2.

Neformálne povedané, snažíme sa imunizovať (zafarbiť na bielo) vrcholy, ktoré by mohli nakaziť (prefarbiť na čierno) už vyliečené (patriace do IP) vrcholy. Vrchol, ktorý sa rozhodneme imunizovať by mal však mať čo najmenej "stykov" (hrán) s nakazenými (čiernymi) vrcholmi. Imúnny podgraf, ktorý nakoniec dostaneme tak bude predstavovať skupinu vyliečených vrcholov, ktorá sa už nenechá ľahko nakaziť okolím.

Algorithm 3 Complete Standard

Vstup: graf G(V, E), konfigurácia K

```
for all (v ∈ V) do
  computeId(v, G, K)
end for
while (máChybnýVrchol(G, K)) do
  w = chybnýVrcholSMaxId(G, K)
  n = susedSMaxId(w, G, K)
  K(n) = biela
  for all (v ∈ Neigh(n)) do
    ID(v) += 2
  end for
end while
I = {v ∈ V|K(v) = biela}
```

Výstup: I

Ilustrovať si tento proces môžeme na grafe G z obrázku 3.4. Nasledujúca tabuľka udáva imúnne diferencie vrcholov grafu G.

Tabuľka 3.4.1: Imúnne diferencie vrcholov grafu G. Červenou sú označené chybné vrcholy, zelenou imúnne vrcholy

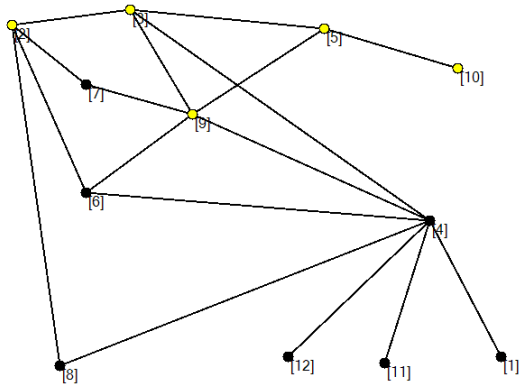
Vrchol	ID
1	-1
2	-2
3	0
4	-3
5	3
6	-1
7	0
8	-2
9	-1
10	1
11	-1
12	-1

V tabuľke sú červenou označené chybné vrcholy. Z nich treba vybrať chybný vrchol s čo najväčším ID. To je v tomto prípade vrchol 3. K tomuto vrcholu chceme nájsť suseda s čo najväčším ID.

Tabuľka 3.4.2: Susedia vrcholu 3. Farebne sú zvýraznené vrcholy čo už v IP sú

Vrchol	ID
2	-2
4	-3
5	3
9	-1

V tomto prípade to bude vrchol 2, ktorý následne aj pridáme do IP.



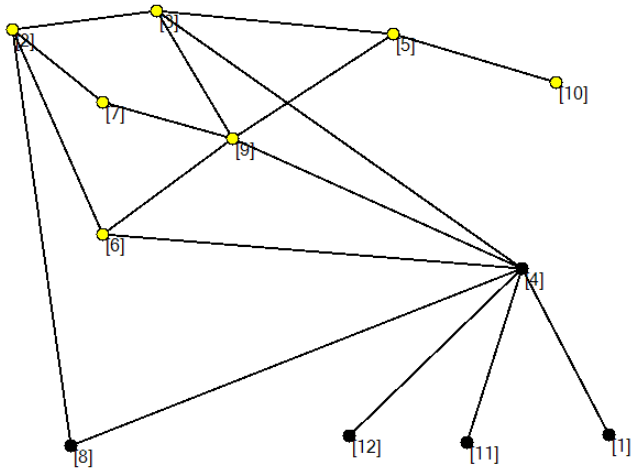
Obr. 3.4.2: Graf G po jednej iterácii Complete Standard

Po pridaní vrchola do IP ešte treba aktualizovať údaje v ID tabuľke.

Tabuľka 3.4.3: Imúnne diferencie vrcholov grafu G po jednej iterácii CS. Červenou sú označené chybné vrcholy, zelenou imúnne vrcholy, modrou aktualizované vrcholy mimo IP

Vrchol	ID
1	-1
2	-2
3	2
4	-3
5	3
6	1
7	2
8	0
9	-1
10	1
11	-1
12	-1

Opakovaním tohto postupu by sme sa dostali až výslednému imúnnemu podgrafu.



Obr. 3.4.3: Graf G po dokončení Complete Standard

Algoritmus na doplnenie IP vždy musí skončiť - v každom kroku totiž do IP pridáva jeden vrchol a overuje, či už nie je IP po tomto pridaní imúnny. V najhoršom prípade skončí tak, že IP bude tvoriť celý graf. Počet krokov CS teda môžeme zhora odhadnúť ako n, keď n je počet vrcholov G. Overenie imunity aktuálneho IP, ako aj aktualizácia ID tabuľky sú tiež operácie lineárne závislé od n, preto časovú zložitosť CS môžeme zhora odhadnúť ako $O(n^2)$.

3.4.2 Complete Premium

Uvažujme ešte raz graf G z obrázku 3.3.4. Vrchol 4 má síce menšiu imúnnu diferenciu ako ktorýkoľvek iný vrchol v grafe, no jeho pridaním do IP by sme vylepšili ID hneď dvoch chybných vrcholov. Ponúka sa teda otázka, či nie je takýto postup výhodnejší a nemáme uprednostňovať opravujúce vrcholy na základe toho, koľkým chybným vrcholom pomôžu.

V Complete Premium (CP) teda postupujeme trochu ináč. Opäť pridávame do IP vrcholy až kým skutočne nie je imúnny, no tento krát v každom kroku vyberáme hneď z celej množiny kandidátov na nový vrchol do IP, a to podľa vlastnosti, ktorú som vyššie spomenul.

Definícia 3.2. ***Fix Gain***[22] *vrchola* v $FG(v)$ *v grafe* $G(V, E)$ *s konfiguráciou* K *je*

$$|\{w \in Neigh(v) | K(w) = biela \wedge ID(w) \leq 0\}| \tag{3.4.2}$$

[22]preklad znamená "koľko získame opravou", v tomto prípade zaradením vrchola do IP

Teda neformálne povedané - koľko susedov vrchola v z IP by ťažilo z pridania v do IP.

Tabuľku ID však musíme naďalej počítať, práve kvôli tomu, že na základe nej budeme počítať Fix Gain tabuľku. V každom kroku algoritmu najskôr overíme, či už nie je podgraf imúnny. Následne vyberieme z čiernych vrcholov ten s najvyšším FG, pridáme ho do IP a aktualizujeme tabuľku ID a FG.

Intuitívna predstava za týmto algoritmom je jasná - snažíme sa akoby imunizovať vrchol, ktorý má veľa spojení s už imunizovaními vrcholmi náchylnými na opätovné nakazenie.

Algorithm 4 Complete Premium

Vstup: graf G(V, E), konfigurácia K

```
for all (v ∈ V) do
  computeId(v, G, K)
end for
for all (v ∈ V) do
  computeFg(v, G, K)
end for
while (máChybnýVrchol(G, K)) do
  w = čiernyVrcholSMaxFg(G, K)
  K(w) = biela
  for all (v ∈ Neigh(w)) do
    ID(v) += 2
  end for
  aktualizujFgOkoloVrchola(w, K, G)
end while
I = {v ∈ V|K(v) = biela}
```

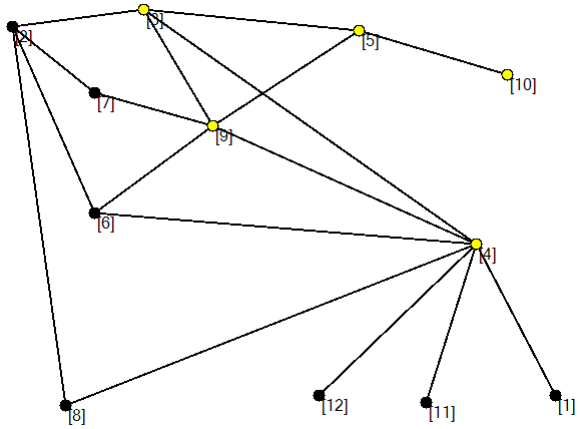
Výstup: I

Detaily aktualizovania FG som skryl do funkcie *aktualizujFgOkoloVrchola*, nakoľko je táto aktualizácia pomerne zložitá. Pridanie nového vrchola totiž môže spôsobiť zmenu FG až u jeho "susedov z druhého kolena", teda susedov susedov. Znázornené to bude na jednom kroku simulácie algoritmu. Tabuľka ID je na začiatku dopĺňania rovnaká ako pri Complete Standard. Tabuľka FG vyzerá takto.

Tabuľka 3.4.4: Fix Gains pre vrcholy z grafu G. Červenou sú označené chybné vrcholy, zelenou imúnne vrcholy

Vrchol	FG
1	0
2	1
3	1
4	2
5	2
6	1
7	1
8	0
9	1
10	0
11	0
12	0

Pre vrcholy z IP síce nemá zmysel počítať Fix Gain, keďže sa už v IP nachádzajú, algoritmu to však nijak neuškodí. Ako kandidáti na prírastok do IP sa ponúkajú vrcholy 2, 4, 6 a 7. Z nich má najväčší Fix Gain vrchol číslo 4. Rovno ho teda pridáme do IP.



Obr. 3.4.4: Graf G po jednej iterácii Complete Premium

Po pridaní treba opäť aktualizovať tabuľku ID.

Tabuľka 3.4.5: Imúnne diferencie vrcholov grafu G po jednej iterácii CP. Červenou sú označené chybné vrcholy, zelenou imúnne vrcholy, modrou aktualizované vrcholy mimo IP

Vrchol	ID
1	1
2	-2
3	2
4	-3
5	3
6	1
7	0
8	0
9	1
10	1
11	1
12	1

Fix Gain najprv aktualizujeme pre susedov vrchola 4. Keďže je tento vrchol chybný, všetkým jeho susedom inkrementujeme hodnotu FG.

Tabuľka 3.4.6: Inkrementácia Fix Gains susedov vrchola číslo 4. Farebne sú označené vrcholy z IP

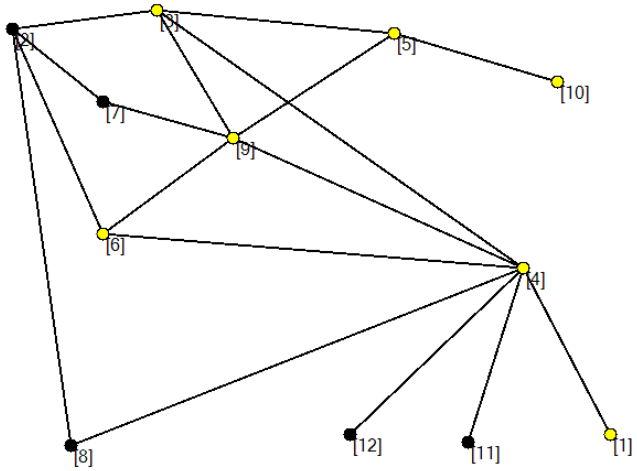
Vrchol	FG
1	0 + 1
3	1 + 1
6	1 + 1
8	0 + 1
9	1 + 1
11	0 + 1
12	0 + 1

Pridaním vrchola 4 do IP sme však ošetrili oba vrcholy - 3 aj 9 - tak, že už nie sú chybné. Preto teda susedom oboch vrcholov musíme (za každý vrchol) dekrementovať Fix Gain.

Tabuľka 3.4.7: Aktualizácia Fix Gains susedov vrcholov 3 a 9. Farebne sú označené vrcholy z IP

Vrchol	FG			
1	0	+ 1		= 1
2	1		- 1	= 0
3	1	+ 1	- 1	= 1
4	2		- 2	= 0
5	2		- 2	= 0
6	1	+ 1	- 1	= 1
7	1		- 1	= 0
8	0	+ 1		= 1
9	1	+ 1	- 1	= 1
10	0			= 0
11	0	+ 1		= 1
12	0	+ 1		= 1

Po istom počte iterácii opäť dostávame imúnny podgraf.



Obr. 3.4.5: Graf G po dokončení Complete Premium

Zamyslime sa teraz nad časovou zložitosťou algoritmu CP. Jediným rozdielom oproti CS (čo sa zložitosti týka) je, že po pridaní nového vrcholu sa musí prepočítavať aj Fix Gain niektorých vrcholov. Okrem priamych susedov vrchola, čo sme pridali do IP sú to aj susedia vrcholov, ktoré *sme pridaním nového vrchola do IP práve zimunizovali*. Naraz však môžeme imunizovať aj viac vrcholov, čo by znamenalo potenciálne až n^2 operácii pri prepočítavaní FG. Máme teda celkovú časovú zložitosť odhadnúť zhora ako $O(n^3)$? Opoveď je nie. Každý vrchol totiž imunizujeme len raz, a tak sa n^2 ako ohraničenie pre prepočítavanie Fix Gain

hodnôt vzťahuje skôr na celú dobu algoritmu ako na jednu jeho iteráciu. Časovú zložitosť Complete Premium teda možno odhadnúť $O(n^2)$

3.4.3 Complete Superb

Problémom CP algoritmu je, že vrchol s väčším FG uprednostní aj v prípade, že má vysokú zápornú imúnnu diferenciu. To znamená, že takýto vrchol bude treba opravovať ešte mnohými ďalšími vrcholmi (teda pridávať do IP jeho susedov). Complete Superb (CSUP) sa preto snaží vyberať vrchol s väčšou rozvahou - kombinuje stratégiu CS a CP a ako ďalší vrchol do IP vyberá ten, ktorého *súčet* FG + $min(1, ID)$ je čo najväčší. Minimum je v súčte zakomponované preto, lebo nemá zmysel vybrať vrchol len kvôli vysokej kladnej ID. ID väčšie ako 0 totiž znamená, že vrchol už nemusíme v budúcnosti ošetrovať a to stačí. Napr. ak vyberáme z dvoch vrcholov v a w keď FG(v) = 4, ID(v) = -1, FG(w) = 2 a ID(w) = 6, je lepšie vybrať vrchol v, lebo vysoká imúnna diferencia pri vrchole w je zbytočná.

Druhým nedostatkom CS a CP je fakt, že pridaný vrchol už z podgrafu nikto neodstráni. CSUP preto po dokončení doplňovania IP prepína do druhej fázy, v ktorej sa snaží z podgrafu vyhodiť čo najviac nepotrebných vrcholov. Pri tomto odstráňovaní uprednostňujeme vrcholy s čo najmenším počtom susedov v IP, aby sme čo najmenej vrcholom z IP znížili imúnnu diferenciu a mohli tak ďalej IP zmenšovať.

Definícia 3.3. ***IPN*** *vrchola v $IPN(v)$ v grafe $G(V, E)$ s konfiguráciou K je*

$$|\{w \in Neigh(v) | K(w) = biela\}| \tag{3.4.3}$$

Teda neformálne povedané - koľko susedov vrchola v je z IP.

Algorithm 5 Complete Superb

Vstup: graf G(V, E), konfigurácia K

Prvá fáza - doplňovanie
for all $(v \in V)$ **do**
computeId(v, G, K)
end for
for all $(v \in V)$ **do**
computeFg(v, G, K)
end for
while (máChybnýVrchol(G, K)) **do**
$w = max_{v \in V}(FG(v) + min(1, ID(v))$
$K(w) = biela$
for all $(v \in Neigh(w))$ **do**
ID(v) += 2
end for
aktualizujFgOkoloVrchola(w, K, G)
end while
Druhá fáza - vyhadzovanie
while (máZbytočnýVrchol(G, K)) **do**
w = minIpnZbytočnýVrchol()
$K(w) = čierna$
for all $(v \in Neigh(w))$ **do**
ID(v) -= 2
end for
aktualizujIpnOkoloVrchola(w, K, G)
end while
$I = \{v \in V | K(v) = biela\}$

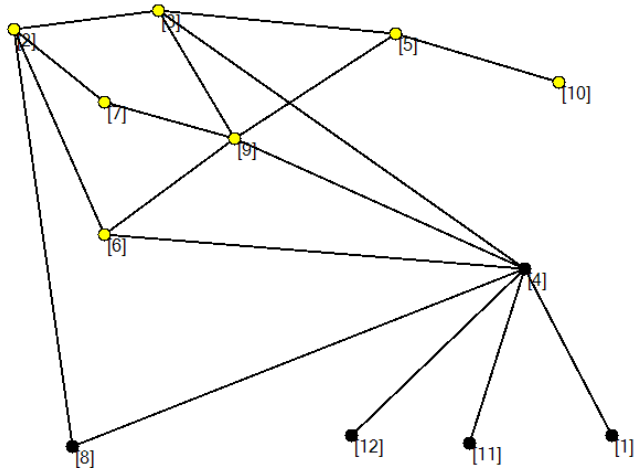
Výstup: I

Začíname teda s rovnakými tabuľkami ID (tabuľka 3.4.1) a FG (tabuľka 3.4.2) ako v algoritme Complete Premium. Uveďme si ešte tabuľku súčtov FG + $min(1, ID)$, keďže to je hodnota, na základe ktorej sa algoritmus rozhoduje.

Tabuľka 3.4.8: Tabuľka súčtu FG(v) + $min(1, ID(v))$ pre $v \in V$ v grafe G(V, E) (obr. 3.3.4). Červenou sú označené chybné vrcholy, zelenou imúnne vrcholy

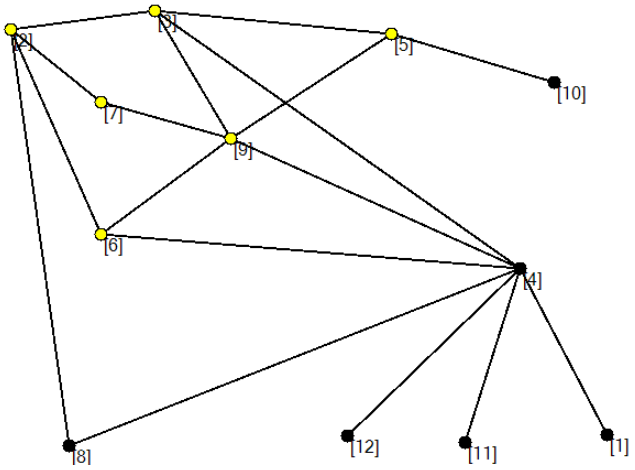
Vrchol v	**FG(v)** + $min(1, ID(v))$
1	0 - 1 = -1
2	1 - 2 = -1
3	1 + 0 = 1
4	2 - 3 = -1
5	2 + 1 = 3
6	1 - 1 = 0
7	1 + 0 = 1
8	0 - 2 = -2
9	1 - 1 = 0
10	0 + 1 = 1
11	0 - 1 = -1
12	0 - 1 = -1

Najlepším kandidátom na nový vrchol do IP je teda vrchol č. 7. Následne sa aktualizuje tabuľka ID a FG - rovnako ako to bolo v CP algoritme. Opakovaním tohto postupu by sme sa dopracovali až k imúnnemu podgrafu.



Obr. 3.4.6: Graf G po dokončení prvej fázy algoritmu CSUP

V druhej fáze algoritmu chceme z IP vyhodiť čo najviac nepotrebných vrcholov. V našom grafe je taký vrchol iba jeden - číslo 10. Preto ho z IP odstránime a dostávame konečnú verziu IP, akú vracia Complete Superb.



Obr. 3.4.7: Graf G po dokončení algoritmu CSUP

Čo sa týka výpočtovej zložitosti, je prvá fáza algoritmu (doplňovanie) rovnako náročná, ako doplňovanie v CP. Výpočet v druhej fáze (vyhadzovanie) opäť nemôže trvať väčší počet krokov ako n. A keďže aktualizácia IPN hodnôt je časovo rovnako náročná ako aktualizácia ID hodnôt, môžeme časovú zložitosť druhej fázy ohraničiť rovnako $O(n^2)$. Tým pádom je celková časová zložitosť $O(n^2)$.

3.5 Riešenie cez heuristiky doplňovania

Na základe doplňovacích heuristík Complete Standard, Premium a Superb som testoval ešte celkom tri trojice algoritmov na hľadanie minimálneho IP. Po trojiciach sa líšia množinou vrcholov, ktoré skúšajú ako základ pre doplňovanie, v rámci trojice sa zase líšia doplňovacou heuristikou. Algoritmy majú teda tú istú jednoduchú štruktúru a princíp.

Algorithm 6 Select Množina Complete Heuristika

Vstup: graf G(V, E)

$K(v) = biela \quad \forall v \in Množina$

$I = completeHeuristika(G, K)$

Výstup: I

Predstavme si ich teda, a uveďme si ich výsledky.

3.5.1 Select Median Complete (SMC)

Tento algoritmus jednoducho za základ pre doplňovanie vyberie vrchol, ktorého stupeň je mediánom (strednou hodnotou) v postupnosti stupňov vrcholov. Následne tento vrchol dopĺňa doplňovacou heuristikou.

Z výsledkov v tabuľke 3.5.4 je vidieť, že algoritmus je veľmi rýchly. Aj grafy veľkosti 2000 vrcholov a viac zvládol do pár sekúnd. Jeho časová zložitosť je rovnaká ako som uvádzal pre doplňovacie heuristiky, teda $O(n^2)$.

3.5.2 Select All Complete (SAC)

Tento algoritmus postupne skúša ako základ pre doplňovanie všetky vrcholy grafu a na každom spustí doplňovanie. Vracia najlepšiu hodnotu (najmenší z nájdených IP).

Časová zložitosť algoritmov je n krát časová zložitosť jednotlivých doplňovacích heuristík, čím sa dostávame k časovej zložitosti $O(n^3)$ a časom výpočtov aj nad 10 minút, viď tabuľka 3.5.4.

3.5.3 AMPL Relaxed Complete (ARC)

Tento algoritmus berie za základ pre doplňovanie podgraf vrátený algoritmom AMPL Relaxed. Bohužiaľ, kvôli obmedzeniu danému študentskou verziou C-PLEX som nemohol otestovať väčšie grafy ako s 300 vrcholmi. Výsledky sú v tabuľke 3.5.4

3.5.4 Výsledky Select-Complete algoritmov

Nasledovať budú tri tabuľky s výsledkami algoritmov SMC, SAC a ARC. Z tabuliek som vypustil stĺpec Úspešnosť, nakoľko všetky algoritmy vždy našli podgraf, ktorý bol skutočne imúnny

Tabuľka 3.5.1: Výsledky SMC

Testovaný graf			**Výsledky SMC standard**		**Výsledky SMC premium**		**Výsledky SMC superb**	
Typ grafu	**Veľkosť**	**Hustota**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**
tori, butterfly, ...	-	*perc.*	-	*h:m:s*	-	*h:m:s*	-	*h:m:s*
tori (5, 10)	50	8	24	00:00:00.0010000	23	0:00:00	22	0:00:00
tori (7, 15)	135	3	48	0:00:00	48	0:00:00	48	0:00:00
tori (14, 20)	280	1	128	00:00:00.0020000	128	00:00:00.0030000	116	00:00:00.0040000
tori (20, 40)	800	<1	368	00:00:00.0180000	368	00:00:00.0190000	319	00:00:00.1020000
tori (30, 50)	1500	<1	708	00:00:00.0620000	708	00:00:00.0680000	590	00:00:00.0690000
tori (60, 80)	4800	<1	2328	00:00:00.6350000	2328	00:00:00.7520000	1859	00:00:00.7090000
butterfly (4)	80	4	28	00:00:00.0040000	49	00:00:00.0010000	36	00:00:00.0010000
butterfly (6)	448	<1	108	00:00:00.0050000	171	00:00:00.0050000	204	00:00:00.0060000
butterfly (9)	5120	<1	1262	00:00:00.5600000	1594	00:00:00.7680000	2176	00:00:00.8780000
ccc (5)	160	1	21	00:00:00.0010000	22	00:00:00.0010000	21	00:00:00.0010000
ccc (7)	896	<1	32	00:00:00.0190000	32	00:00:00.0190000	31	00:00:00.0210000
ccc (9)	4608	<1	42	00:00:00.6140000	42	00:00:00.6260000	41	00:00:00.5990000
hypercube (5)	32	16	12	0:00:00	8	0:00:00	8	0:00:00
hypercube (7)	128	5	36	00:00:00.0010000	16	0:00:00	16	0:00:00
hypercube (9)	512	1	108	00:00:00.0090000	32	00:00:00.0070000	32	00:00:00.0080000
hypercube (12)	4096	<1	828	00:00:00.4700000	128	00:00:00.4050000	128	00:00:00.4730000
star (3)	6	40	6	0:00:00	6	0:00:00	6	0:00:00
star (5)	120	3	24	0:00:00	24	0:00:00	24	0:00:00
star (7)	5040	<1	120	00:00:00.6570000	120	00:00:00.6110000	120	00:00:00.6370000
random	20	20	8	0:00:00	14	0:00:00	12	0:00:00
random	30	80	16	0:00:00	16	0:00:00	16	0:00:00
random	50	40	29	00:00:00.0010000	26	0:00:00	28	00:00:00.0010000
random	80	90	43	00:00:00.0020000	44	00:00:00.0030000	42	00:00:00.0020000
random	140	30	79	00:00:00.0020000	82	00:00:00.0030000	72	00:00:00.0030000
random	350	50	189	00:00:00.0200000	192	00:00:00.0220000	179	00:00:00.0250000
random	550	60	289	00:00:00.0650000	290	00:00:00.0680000	281	00:00:00.0980000
random	800	40	424	00:00:00.1640000	428	00:00:00.1390000	410	00:00:00.2230000
random	1300	70	669	00:00:00.8270000	666	00:00:00.8540000	659	00:00:00.7590000
random	2000	80	1020	00:00:02.2440000	1022	00:00:02.1390000	1011	00:00:02.6170000

Tabuľka 3.5.2: Výsledky SAC

Testovaný graf			**Výsledky SAC standard**		**Výsledky SAC premium**		**Výsledky SAC superb**	
Typ grafu	**Veľkosť**	**Hustota**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**
tori, butterfly, ...	-	*perc.*	-	*h:m:s*	-	*h:m:s*	-	*h:m:s*
tori (5, 10)	50	8	10	00:00:00.0010000	10	00:00:00.0020000	10	00:00:00.0040000
tori (7, 15)	135	3	14	00:00:00.0080000	14	00:00:00.0110000	14	00:00:00.0160000
tori (14, 20)	280	1	28	00:00:00.1000000	28	00:00:00.1690000	28	00:00:00.2570000
tori (20, 40)	800	<1	40	00:00:02.2440000	40	00:00:04.0080000	40	00:00:04.2030000
tori (30, 50)	1500	<1	60	00:00:14.1820000	60	00:00:23.4880000	60	00:00:26.3110000
tori (60, 80)	4800	<1	120	00:07:39.9860000	120	00:12:05.3970000	120	00:14:47.3760000
butterfly (4)	80	4	28	00:00:00.0070000	28	00:00:00.0080000	36	00:00:00.0530000
butterfly (6)	448	<1	72	00:00:00.2790000	128	00:00:00.5890000	104	00:00:01.0760000
butterfly (9)	5120	<1	755	00:04:07.5930000	672	00:08:05.0130000	2172	00:24:58.6720000
ccc (5)	160	1	5	00:00:00.0060000	5	00:00:00.0120000	5	00:00:00.0160000
ccc (7)	896	<1	7	00:00:00.2300000	7	00:00:00.4530000	7	00:00:00.6070000
ccc (9)	4608	<1	9	00:00:07.6740000	9	00:00:14.9300000	9	00:00:20.9450000
hypercube (5)	32	16	8	00:00:00.0010000	8	00:00:00.0010000	8	00:00:00.0010000
hypercube (7)	128	5	16	00:00:00.0080000	16	00:00:00.0100000	16	00:00:00.0150000
hypercube (9)	512	1	32	00:00:00.2060000	32	00:00:00.2130000	32	00:00:00.3060000
hypercube (12)	4096	<1	128	00:00:51.2010000	128	00:00:34.2470000	128	00:00:47.8620000
star (3)	6	40	6	0:00:00	6	0:00:00	6	0:00:00
star (5)	120	3	24	00:00:00.0110000	24	00:00:00.0090000	24	00:00:00.0120000
star (7)	5040	<1	120	00:00:32.7110000	120	00:00:45.8520000	120	00:01:05.5280000
random	20	20	8	0:00:00	12	00:00:00.0010000	8	00:00:00.0010000
random	30	80	16	00:00:00.0020000	16	00:00:00.0020000	16	00:00:00.0030000
random	50	40	28	00:00:00.0030000	24	00:00:00.0040000	25	00:00:00.0060000
random	80	90	41	00:00:00.0210000	41	00:00:00.0310000	41	00:00:00.0340000
random	140	30	73	00:00:00.0530000	69	00:00:00.0750000	68	00:00:00.0970000
random	350	50	181	00:00:01.0070000	179	00:00:01.4990000	176	00:00:02.2200000
random	550	60	283	00:00:04.6280000	279	00:00:07.1910000	278	00:00:08.8450000
random	800	40	415	00:00:11.4730000	407	00:00:16.5160000	402	00:00:20.2640000
random	1300	70	661	00:01:08.2370000	658	00:01:45.8770000	654	00:02:03.3850000
random	2000	80	1013	00:04:47.0660000	1010	00:07:20.0430000	1006	00:08:15.0180000

Tabuľka 3.5.3: Výsledky ARC

Testovaný graf			**Výsledky ARC standard**		**Výsledky ARC premium**		**Výsledky ARC superb**	
Typ grafu	**Veľkosť**	**Hustota**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**
tori, butterfly, ...	-	*perc.*	-	*h:m:s*	-	*h:m:s*	-	*h:m:s*
tori (5, 10)	50	8	20	00:00:00.0500000	20	00:00:00.0490000	20	00:00:00.0570000
tori (7, 15)	135	3	28	00:00:00.0660000	28	00:00:00.0670000	28	00:00:00.0630000
tori (14, 20)	280	1	56	00:00:00.1480000	56	00:00:00.1300000	56	00:00:00.1370000
butterfly (4)	80	4	34	00:00:00.0950000	56	00:00:00.0960000	36	00:00:00.9800000
ccc (5)	160	1	10	00:00:00.3450000	10	00:00:00.0680000	10	00:00:00.0960000
hypercube (5)	32	16	8	00:00:00.0520000	8	00:00:00.0480000	8	00:00:00.0510000
hypercube (7)	128	5	16	00:00:00.0670000	16	00:00:00.0770000	16	00:00:00.1230000
star (3)	6	40	6	00:00:00.0510000	6	00:00:00.0550000	6	00:00:00.0800000
star (5)	120	3	56	00:00:00.0610000	56	00:00:00.0600000	54	00:00:00.0670000
random	20	20	14	00:00:00.0490000	14	00:00:00.0470000	13	00:00:00.0510000
random	30	80	18	00:00:00.0500000	18	00:00:00.0490000	17	00:00:00.0590000
random	50	40	29	00:00:00.0520000	34	00:00:00.0530000	28	00:00:00.0550000
random	80	90	42	00:00:00.0630000	45	00:00:00.0610000	42	00:00:00.0700000
random	140	30	83	00:00:00.1060000	85	00:00:00.0780000	75	00:00:00.1140000

4 Ďalšie testy a porovnanie algoritmov

V tejto sekcii by sa ešte detailnejšie pozrieme na jednotlivé algoritmy a porovnáme ich navzájom.

4.1 Úspešnosť AMPL Relaxed

Tabuľka 4.1.1: Úspešnosť AMPL Relaxed

Test			**Výsledky AMPL Relaxed**		
Grafov	**Veľkosť**	**Hustota hrán**	**Veľkosť IP**	**Čas**	**Úspešnosť**
počet	*od - do*	*od - do*	*priemer*	*priemer, h:m:s*	*percent*
85	2 - 20	50 %	7.047	00:00:00.0851176	23 %
50	200	50 %	91.1	00:00:00.1650600	0 %
99	5 - 297	20 - 80 %	69.636	00:00:00.1475555	2 %

Z uvedených výsledkov jasne vyplýva, že na AMPL Relaxed sa nemožno spoľahnúť, a to ani pre malé grafy.

4.2 AMPL Relaxed vs. ARC Standard

Tabuľka 4.2.1: AMPL Relaxed vs. ARC Standard

Testovacia sada			**Výsledky AMPL Relaxed**			**Výsledky ARC Standard**	
Grafov	**Veľkosť**	**Hustota**	**Veľkosť IP**	**Čas**	**Úspešnosť**	**Veľkosť IP**	**Čas**
počet	*od - do*	*od - do*	*priemer*	*priemer, h:m:s*	*percent*	*priemer*	*priemer, h:m:s*
34	20	20 %	6.941	00:00:00.0828235	0 %	12.765	00:00:00.0798823
50	20	70 %	10.38	00:00:00.0855000	0 %	11.94	00:00:00.0830600
50	50	20 %	17.84	00:00:00.0867800	0 %	30.96	00:00:00.0833400
50	50	70 %	24.4	00:00:00.0932400	0 %	27.8	00:00:00.0861800
50	250	20 %	105.54	00:00:00.1785400	0 %	136.7	00:00:00.1767600
50	250	70 %	120.64	00:00:00.2453600	0 %	135.2	00:00:00.2507800

Pre väčšie grafy je percentuálna odchýlka veľkosti IP AMPL Relaxed od skutočne imúnneho podgrafu čoraz menšia. ARC Standard má vo výsledkoch tiež ešte rezervu - AMPL Integral pre grafy veľkosti 50 s hustotou hrán 20 % (čomu odpovedá tretí test) našiel v priemere IP s veľkosťou 19,3.

4.3 ARC Superb vs. AMPL Integral

Tabuľka 4.3.1: ARC Superb vs. AMPL Integral

Test			Výsledky ARC Superb		AMPL Integral	
Veľkosť	**Poč. hrán**	**Typ**	**Veľkosť IP**	**Čas**	**Veľkosť IP**	**Čas**
-	-	*tori..*	-	*h:m:s*	-	*h:m:s*
5	7	random	4	00:00:00.0780000	4	00:00:00.1710000
7	10	random	5	00:00:00.0790000	5	00:00:00.0830000
9	12	random	3	00:00:00.0790000	3	00:00:00.0810000
12	33	random	8	00:00:00.0800000	6	00:00:00.0910000
14	69	random	8	00:00:00.0830000	8	00:00:00.1100000
16	88	random	10	00:00:00.0780000	9	00:00:00.1100000
19	117	random	11	00:00:00.0790000	9	00:00:00.1140000
21	130	random	12	00:00:00.0790000	11	00:00:00.1240000
23	131	random	14	00:00:00.0780000	10	00:00:00.1260000
26	185	random	14	00:00:00.0800000	12	00:00:00.1390000
28	241	random	15	00:00:00.0850000	13	00:00:00.1340000
31	339	random	17	00:00:00.0960000	16	00:00:00.6670000
33	364	random	18	00:00:00.0800000	16	00:00:00.6140000
35	243	random	19	00:00:00.0820000	15	00:00:00.4970000
38	414	random	21	00:00:00.1010000	18	00:00:01.0910000
40	319	random	22	00:00:00.0850000	18	00:00:01.4490000
42	421	random	23	00:00:00.0830000	20	00:00:02.9970000
45	485	random	26	00:00:00.0820000	20	00:00:03.5470000
47	313	random	26	00:00:00.0810000	20	00:00:02.1870000
50	784	random	27	00:00:00.0860000	24	00:00:42.6000000
27,05	235,25	random	15,15	00:00:00.0827000	12,85	00:00:02.8466000
			Best: 20 %	Best: 95 %	Best: 100 %	Best: 5 %

Na konci tabuľky je zhrnutie výsledkov - priemerné hodnoty v danej testovacej sade. V druhom riadku sú potom percentuálne hodnoty pre čas a veľkosť IP, ktoré udávajú v koľkých testoch bol konkrétny algoritmus najlepší. Tieto hodnoty sa v predošlom teste (AMPL Relaxed vs. ARC Standard) nevyskytovali, nakoľko AMPL Relaxed ani raz nenašiel imúnny podgraf a tieto hodnoty by tak boli zavádzajúce. Hustota hrán grafov sa pohybovala v rozpätí 20 - 80 %.

4.4 Rýchlostný test SMC pre náhodné grafy

Tabuľka 4.4.1: Rýchlostný test SMC, náhodné grafy

Test		SMC Standard		SMC Premium		SMC Superb	
Veľkosť	**Poč. hrán**	**Veľkosť IP**	**Čas**	**Veľkosť IP**	**Čas**	**Veľkosť IP**	**Čas**
-	-	-	*m:s*	-	*m:s*	-	*m:s*
100	1633	54	00:00.21800	53	00:00.04400	51	00:00.01400
363	16425	200	00:00.01300	206	00:00.01300	187	00:00.01400
627	109900	333	00:00.07600	329	00:00.09200	320	00:00.50400
890	241319	461	00:00.83900	475	00:00.66500	454	00:00.53400
1154	518919	594	00:00.78500	596	00:00.79800	586	00:01.11500
1418	502326	746	00:01.11200	733	00:01.13900	718	00:01.26100
1681	1101391	855	00:01.85800	865	00:02.38000	849	00:02.19000
1945	567162	1024	00:01.61400	1020	00:01.35800	988	00:01.35200
2209	877944	1162	00:01.68600	1146	00:01.87200	1120	00:02.09900
2472	1588161	1270	00:02.90400	1296	00:03.98000	1250	00:03.35000
2736	1721080	1415	00:03.36400	1429	00:03.34900	1388	00:04.00100
3000	2519160	1552	00:04.55000	1535	00:04.87900	1517	00:04.39800
1549,583	2703118	805,5	00:01.58491	806,91666	00:01.71408	785,66666	00:01.73600
		Best 0 %	Best 58 %	Best 0 %	Best 16 %	Best 100 %	Best 33 %

Rýchlostne sú teda na tom všetky algoritmy dobre, najlepšie asi SMC Standard. Najmenší IP našiel vo všetkých prípadoch podľa očakávaní algoritmus SMC Superb. Naopak, SMC Premium v priamom porovnaní veľkosti IP s SMC Standard dosiahol skóre 6:6. Hustota hrán grafov sa pohybovala v rozpätí 30 - 70 %.

4.5 Porovnanie SAC algoritmov na náhodných grafoch

Tabuľka 4.5.1: SAC algoritmy na náhodných grafoch

Testovacia sada			SAC Standard		SAC Premium		SAC Superb	
Grafov	**Veľkosť**	**Hustota**	**IP**	**Čas**	**IP**	**Čas**	**IP**	**Čas**
počet	*od - do*	*od - do*	*priemer* *best %*	*priem. sek* *best %*	*priemer* *best %*	*priem. sek* *best %*	*priemer* *best %*	*priem. sek* *best %*
49	20	20 - 80 %	10,449 59 %	00.0005306 61 %	10,306 71 %	00.0005510 48 %	10,082 85 %	00.0007959 44 %
50	150	20 - 80 %	78,3 4 %	00.0859200 100 %	75,4 50 %	00.1272400 0 %	75 72 %	00.1538000 0 %
50	400	20 - 80	207,5 0 %	01.5588600 100 %	202,98 20 %	02.2834200 0 %	201,12 96 %	02.6162800 0 %
10	750	20 - 80 %	388,3 0 %	09.8341000 100 %	380,4 0 %	14.4614000 0 %	377,9 100 %	16.6949000 0 %
5	1000	50 %	515,6 0 %	25.8684000 100 %	507,8 0 %	37.3370000 0 %	504,2 100 %	43.2578000 0 %

Z uvedených výsledkov je vidno, že kvalita i dĺžka výpočtu má vzrastajúcu tendenciu od SAC Standard po SAC Superb.

4.6 SAC vs. SMC pre náhodné grafy od 300 do 700 vrcholov

Grafy v teste mali hustotu od 30 do 70 %. Výsledky (tabuľka 4.7) sú uvedené na samostatnej strane. Na tridsiatich vstupoch sa ukázalo, že SAC algoritmy dávajú vo všeobecnosti lepšie výsledky ako SMC, a že Superb verzie algoritmov sú presnejšie ako Premium a Standard. Zároveň išlo o grafy s pomerne veľkou hustotou, a tak rozdiely medzi SAC a SMC algoritmami neboli až tak veľké (medián stupňov vrcholov bol vhodný ako základ pre doplňovanie).

4.7 SMC a SAC algoritmy na špeciálnych grafoch

V oboch prípadoch je vidno zaujímavý nárast veľkosti IP pre Butterfly od Standard po Superb. SAC algoritmy dávali presné hodnoty pre torusy, hyperkocky a hviezdy, SMC uspeli viac menej len pri hviezdach. Výsledky SMC (tabuľka 4.7) a SAC (tabuľka 4.7) sú uvedené na samostatných stranách.

Tabuľka 4.7.1: SAC vs. SMC pre náhodné grafy

Test		SAC Standard		SMC Standard		SAC Premium		SMC Premium		SAC Superb		SMC Superb	
Veľkosť	**Poč. hrán**	**IP**	**Čas**	**IP**	**Čas**	**IP**	**Čas**	**IP**	**Čas**	**IP**	**Čas**	**IP**	**Čas**
-	-	-	*sek*	-	*sek*	-	*sek*	-	*sek*	-	*sek*	-	*sek*
300	15697	157	00.504	165	00.018	151	00.737	165	00.013	151	00.865	157	00.013
313	19531	164	00.618	169	00.014	160	00.909	174	00.014	158	01.093	161	00.015
327	22919	169	00.758	179	00.015	165	01.156	174	00.016	165	01.309	169	00.016
341	22028	178	00.772	187	00.018	173	01.141	184	00.018	171	01.334	176	00.018
355	43356	182	01.424	187	00.027	180	01.977	196	00.048	179	02.249	182	00.045
368	33088	190	01.231	199	00.022	187	01.723	202	00.023	186	01.930	187	00.022
382	31291	198	01.165	208	00.023	192	01.727	201	00.022	192	02.094	195	00.023
396	43015	204	01.850	214	00.029	201	02.335	217	00.032	199	02.614	202	00.035
410	44437	212	01.641	215	00.033	207	02.456	215	00.035	207	02.792	210	00.034
424	52012	217	02.016	220	00.049	216	03.395	227	00.043	214	04.566	216	00.041
437	48585	227	02.255	234	00.037	221	03.679	241	00.042	220	03.749	221	00.037
451	38560	235	02.126	239	00.028	231	02.796	240	00.030	227	03.137	232	00.032
465	72279	238	02.979	244	00.049	237	05.304	245	00.053	234	05.122	238	00.050
479	76702	246	03.481	253	00.051	243	05.255	250	00.081	241	06.649	244	00.086
493	65490	255	03.870	262	00.052	250	05.400	254	00.073	249	05.865	252	00.080
506	49828	265	02.643	270	00.086	259	05.043	266	00.037	255	05.114	259	00.044
520	83662	267	04.354	274	00.056	265	07.582	275	00.074	262	07.448	266	00.194
534	86809	274	04.221	279	00.142	273	06.554	278	00.103	268	07.533	271	00.156
548	100418	280	05.645	285	00.117	279	08.661	290	00.071	276	08.663	278	00.073
562	91431	289	04.894	302	00.162	285	08.362	294	00.072	283	09.341	287	00.074
575	85813	297	05.574	299	00.064	293	07.896	305	00.066	289	09.158	291	00.101
589	109094	302	06.306	307	00.075	299	09.160	308	00.185	297	10.014	299	00.089
603	74416	312	04.721	328	00.050	308	06.824	308	00.062	304	08.347	309	00.099
617	79815	319	04.819	327	00.147	315	08.096	331	00.066	312	08.797	314	00.068
631	109320	324	06.171	331	00.158	320	10.846	336	00.085	319	11.212	324	00.119
644	105593	333	07.298	344	00.081	327	11.812	334	00.168	325	12.358	329	00.128
658	95107	341	07.057	344	00.165	337	10.414	347	00.159	332	11.942	337	00.087
672	155564	343	10.584	353	00.317	341	16.836	360	00.205	339	17.204	340	00.227
686	70486	361	05.889	377	00.063	348	08.892	370	00.067	345	10.491	354	00.082
700	137004	361	09.697	370	00.154	357	15.658	370	00.208	353	17.543	357	00.194
499,533	68778,33	258	03.885	265,5	00.076	254	06.087	265,233	00.072	251,733	06.684	255,233	00.076
		0 %	0 %	0 %	63 %	13 %	0 %	0 %	36 %	100 %	0 %	0 %	20 %

Tabuľka 4.7.2: SMC algoritmy na špeciálnych grafoch

Testovaný graf			Optimum (vt. 3.1)	SMC standard		SMC premium		SMC superb	
Veľkosť	**Poč. hrán**	**Typ grafu**	**Veľkosť**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**
-	-	*tori, ccc, ..*	-	-	*m:s*	-	*m:s*	-	*m:s*
4	4	butterfly	4	4	0:00:00	4	0:00:00	4	0:00:00
12	16	butterfly	10	10	0:00:00	11	0:00:00	10	0:00:00
32	48	butterfly	16	16	0:00:00	23	00:00.0010000	16	0:00:00
80	128	butterfly	28	28	00:00.0010000	49	0:00:00	36	0:00:00
192	320	butterfly	40	52	00:00.0010000	112	00:00.0010000	85	00:00.0010000
448	768	butterfly	64	108	00:00.0050000	171	00:00.0060000	204	00:00.0100000
1024	1792	butterfly	88	216	00:00.0250000	228	00:00.0330000	447	00:00.0310000
2304	4096	butterfly	136	432	00:00.1110000	771	00:00.1340000	963	00:00.1600000
5120	9216	butterfly	184	1262	00:00.5890000	1594	00:00.6130000	2176	00:00.7500000
2	1	ccc	2	2	0:00:00	2	0:00:00	2	0:00:00
8	8	ccc	8	8	0:00:00	8	0:00:00	8	0:00:00
24	36	ccc	3	3	0:00:00	3	00:00.0010000	3	0:00:00
64	96	ccc	4	10	00:00.0010000	16	0:00:00	10	0:00:00
160	240	ccc	5	21	0:00:00	22	00:00.0010000	21	00:00.0010000
384	576	ccc	6	27	00:00.0030000	27	00:00.0040000	26	00:00.0040000
896	1344	ccc	7	32	00:00.0190000	32	00:00.0190000	31	00:00.0190000
2048	3072	ccc	8	37	00:00.0910000	37	00:00.0990000	36	00:00.1000000
4608	6912	ccc	8	42	00:00.4780000	42	00:00.4840000	41	00:00.4710000
2	1	hypercube	2	2	0:00:00	2	0:00:00	2	0:00:00
4	4	hypercube	4	4	0:00:00	4	0:00:00	4	0:00:00
8	12	hypercube	4	4	0:00:00	4	00:00.0010000	4	0:00:00
16	32	hypercube	8	8	0:00:00	8	0:00:00	8	0:00:00
32	80	hypercube	8	12	0:00:00	8	00:00.0010000	8	0:00:00
64	192	hypercube	16	28	00:00.0010000	16	0:00:00	16	00:00.0010000
128	448	hypercube	16	36	00:00.0010000	16	00:00.0010000	16	00:00.0010000
256	1024	hypercube	32	92	00:00.0020000	32	00:00.0020000	32	00:00.0020000
512	2304	hypercube	32	108	00:00.0080000	32	00:00.0170000	32	00:00.0110000
1024	5120	hypercube	64	276	00:00.0270000	64	00:00.0270000	64	00:00.0420000
2048	11264	hypercube	64	308	00:00.1030000	64	00:00.1050000	64	00:00.1060000
4096	24576	hypercube	128	828	00:00.4260000	128	00:00.4050000	128	00:00.4170000
1	0	star	1	1	0:00:00	1	0:00:00	1	0:00:00
2	1	star	2	2	0:00:00	2	0:00:00	2	0:00:00
6	6	star	6	6	00:00.0010000	6	0:00:00	6	0:00:00
24	36	star	6	6	0:00:00	6	00:00.0100000	6	0:00:00
120	240	star	24	24	00:00.0010000	24	00:00.0010000	24	00:00.0010000
720	1800	star	24	24	00:00.0120000	24	00:00.0130000	24	00:00.0130000
5040	15120	star	120	120	00:00.5760000	120	00:00.5860000	120	00:00.6170000
100	200	torus	20	49	00:00.0010000	48	00:00.0010000	45	0:00:00
255	510	torus	30	125	00:00.0020000	125	00:00.0020000	115	00:00.0020000
525	1050	torus	42	255	00:00.0080000	255	00:00.0090000	222	00:00.0090000
858	1716	torus	52	416	00:00.0210000	416	00:00.0230000	357	00:00.0250000
1312	2624	torus	64	638	00:00.0480000	638	00:00.0530000	529	00:00.0560000
1776	3552	torus	74	848	00:00.0830000	848	0:00:00	698	00:00.0970000
2408	4816	torus	86	1156	00:00.1480000	1156	00:00.1800000	942	00:00.1750000
3072	6144	torus	96	1480	00:00.2520000	1480	00:00.3000000	1199	00:00.2920000
3888	7776	torus	108	1880	00:00.3860000	1880	00:00.4230000	1501	00:00.4560000
4800	9600	torus	120	2328	00:00.6500000	2328	00:00.8400000	1859	00:00.7200000
1074,617	2743		37,361	284,553	00:00.0884255	274,191	00:00.0938085	258,447	00:00.1082127
			Best 100 %	Best 31 %	Best 36 %	Best 42 %	Best 36 %	Best 46 %	Best 31 %

Tabuľka 4.7.3: SAC algoritmy na špeciálnych grafoch

Testovaný graf			Optimum (vt. 3.1)	SAC standard		SAC premium		SAC superb	
Veľkosť	**Poč. hrán**	**Typ grafu**	**Veľkosť**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**	**Veľkosť**	**Čas**
-	-	*tori, ccc, ..*	-	-	*m:s*	-	*m:s*	-	*m:s*
4	4	butterfly	4	4	0:00:00	4	0:00:00	4	0:00:00
12	16	butterfly	10	10	0:00:00	10	0:00:00	10	00:00.0010000
32	48	butterfly	16	16	00:00.0010000	16	00:00.0010000	16	00:00.0010000
80	128	butterfly	28	28	00:00.0050000	28	00:00.0070000	36	00:00.0110000
192	320	butterfly	40	48	00:00.0270000	60	00:00.0620000	48	00:00.0890000
448	768	butterfly	64	72	00:00.2790000	128	00:00.5890000	104	00:01.0760000
1024	1792	butterfly	88	148	00:02.6670000	224	00:05.4870000	443	00:11.9290000
2304	4096	butterfly	136	394	00:27.6390000	396	00:53.5060000	963	02:31.2920000
5120	9216	butterfly	184	755	04:07.5930000	672	08:05.0130000	2172	24:58.6720000
2	1	ccc	2	2	0:00:00	2	0:00:00	2	0:00:00
8	8	ccc	8	8	0:00:00	8	00:00.0010000	8	0:00:00
24	36	ccc	3	3	0:00:00	3	0:00:00	3	0:00:00
64	96	ccc	4	4	00:00.0040000	4	00:00.0020000	4	00:00.0020000
160	240	ccc	5	5	00:00.0060000	5	00:00.0110000	5	00:00.0150000
384	576	ccc	6	6	00:00.0390000	6	00:00.0720000	6	00:00.0960000
896	1344	ccc	7	7	00:00.2260000	7	00:00.4280000	7	00:00.5940000
2048	3072	ccc	8	8	00:01.2840000	8	00:02.4170000	8	00:03.2610000
4608	6912	ccc	8	9	00:07.0880000	9	00:13.4360000	9	00:18.4440000
2	1	hypercube	2	2	0:00:00	2	0:00:00	2	0:00:00
4	4	hypercube	4	4	0:00:00	4	00:00.0010000	4	0:00:00
8	12	hypercube	4	4	0:00:00	4	0:00:00	4	0:00:00
16	32	hypercube	8	8	0:00:00	8	00:00.0010000	8	00:00.0030000
32	80	hypercube	8	8	0:00:00	8	00:00.0010000	8	00:00.0010000
64	192	hypercube	16	16	00:00.0020000	16	00:00.0020000	16	00:00.0030000
128	448	hypercube	16	16	00:00.0070000	16	00:00.0090000	16	00:00.0130000
256	1024	hypercube	32	32	00:00.0500000	32	00:00.0520000	32	00:00.0680000
512	2304	hypercube	32	32	00:00.2120000	32	00:00.2100000	32	00:00.2770000
1024	5120	hypercube	64	64	00:01.3850000	64	00:01.1690000	64	00:01.6270000
2048	11264	hypercube	64	64	00:06.0230000	64	00:04.6580000	64	00:06.5810000
4096	24576	hypercube	128	128	00:47.1910000	128	00:30.5980000	128	00:43.5310000
1	0	star	1	1	0:00:00	1	0:00:00	1	0:00:00
2	1	star	2	2	0:00:00	2	0:00:00	2	0:00:00
6	6	star	6	6	0:00:00	6	0:00:00	6	0:00:00
24	36	star	6	6	0:00:00	6	00:00.0010000	6	0:00:00
120	240	star	24	24	00:00.0070000	24	00:00.0090000	24	00:00.0120000
720	1800	star	24	24	00:00.1950000	24	00:00.3000000	24	00:00.4160000
5040	15120	star	120	120	00:29.4820000	120	00:39.8780000	120	00:57.9260000
100	200	torus	20	20	00:00.0070000	20	00:00.0090000	20	00:00.0120000
255	510	torus	30	30	00:00.0750000	30	00:00.1190000	30	00:00.1510000
525	1050	torus	42	42	00:00.5740000	42	00:00.9470000	42	00:01.1560000
858	1716	torus	52	52	00:02.4840000	52	00:03.8420000	52	00:04.8620000
1312	2624	torus	64	64	00:08.5750000	64	00:13.7380000	64	00:17.4030000
1776	3552	torus	74	74	00:22.5310000	74	00:38.3570000	74	00:49.6960000
2408	4816	torus	86	86	00:55.4500000	86	01:29.3690000	86	01:46.5360000
3072	6144	torus	96	96	01:52.1390000	96	03:06.2760000	96	04:00.0590000
3888	7776	torus	108	108	04:19.3130000	108	06:30.1650000	108	07:57.6270000
4800	9600	torus	120	120	07:35.9450000	120	10:54.5480000	120	15:38.5900000
1074,617	2743		37,361	59,149	00:27.4150000	60,489	00:42.8785319	108,532	01:17.2772978
			100 %	Best 97 %	Best 89 %	Best 91 %	Best 34 %	Best 89 %	Best 27 %

5 Program

Súčasťou tejto bakalárskej práce je aj program, nazvaný Immune Subgraph Analyzer. V tejto sekcii by som ho chcel aspoň stručne predstaviť a priblížiť jeho funkcionalitu.

Program som zhotovil predovšetkým s cieľom testovať jednotlivé heuristiky a algoritmy. V ranných štádiách mojej bakalárskej práce som však potreboval nástroj na vytváranie a editovanie zafarbených grafov, na skúšanie imunity podgrafov či na simuláciu výpočtu väčšinového hlasovania. Základom programu ISA je teda prostredie, kde si môže človek vytvoriť graf, zafarbiť jeho podgraf a analyzovať ho z hľadiska imunity.

Nadstavbou je potom už samotné testovacie prostredie v samostatnom okne, ktoré okrem vytvorenia a spustenia rôznych typov testov vie aj inteligentne podať namerané výsledky, čo mi v neskorších fázach písania bakalárky nesmierne ušetrilo robotu.

Program má GUI rozhranie v angličtine.

5.1 Immune Subgraph Analyzer

V hlavnom okne (obr. 5.3) môžete vytvárať nový graf, editovať a transformovať graf načítaný zo súboru, spúšťať väčšinové hlasovanie, alebo na danom grafe vyskúšať jednu z heuristík. Zároveň je podľa týchto vymenovaných aktivít aj rozdelený bočný panel na jednotlivé oblasti. V spodnej časti okna je logovací mechanizmus, ktorý sprostredkuje komunikáciu informácii užívateľovi.

5.2 Testovacie prostredie

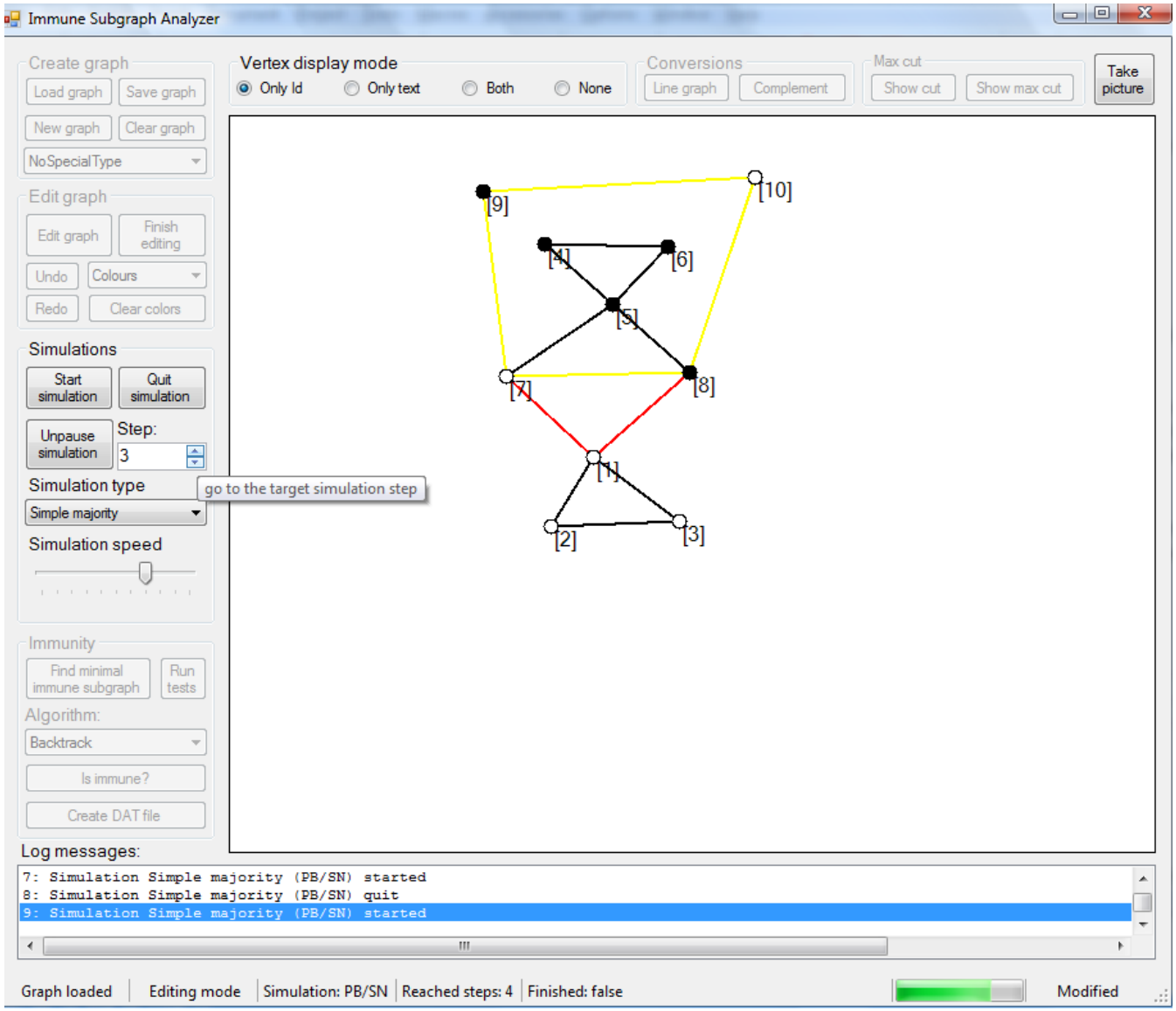
Po kliknutí na "run tests" sa otvorí testovacie prostredie (obr. 5.3). Je možné ich mať otvorených niekoľko naraz a vykonávať tak paralelne viacero testov. V hornej lište naľavo si môžete vytvoriť novú testovaciu sadu - nastavíte, či chcete náhodné grafy, alebo špeciálne, ich veľkosti, prípadne dimenzie, počet grafov, hustotu hrán... Nová testovacia sada sa vytvorí na disku ako priečinok s názvom testu v podadresári *Graphs* adresára spustiteľného súboru. Testovacie sady tak môžte vytvárať aj manuálne v hlavnom okne ISA tak, že budete vytvorené grafy ukladať do nejakého priečinka v spomínanom adresári *Graphs*.

Po výbere algoritmov a dát stačí spustiť test a postupne začnú v tabuľke naskakovať výsledky. Test sa dá zastaviť tlačidlom “Stop”. ktorý nechá dobehnúť aktuálne testovaný graf alebo tlačidlom “Kill”, ktoré zastaví test okamžite [23], ale nezobrazí potom súhrn výsledkov testu. Výsledky možno v tabuľke usporiadať podľa konkrétneho stĺpca, či si zobraziť testovací vstup v hlavnom okne (dvojklik na riadok tabuľky). Po dobehnutí testu sa ešte zobrazia priemerné hodnoty.

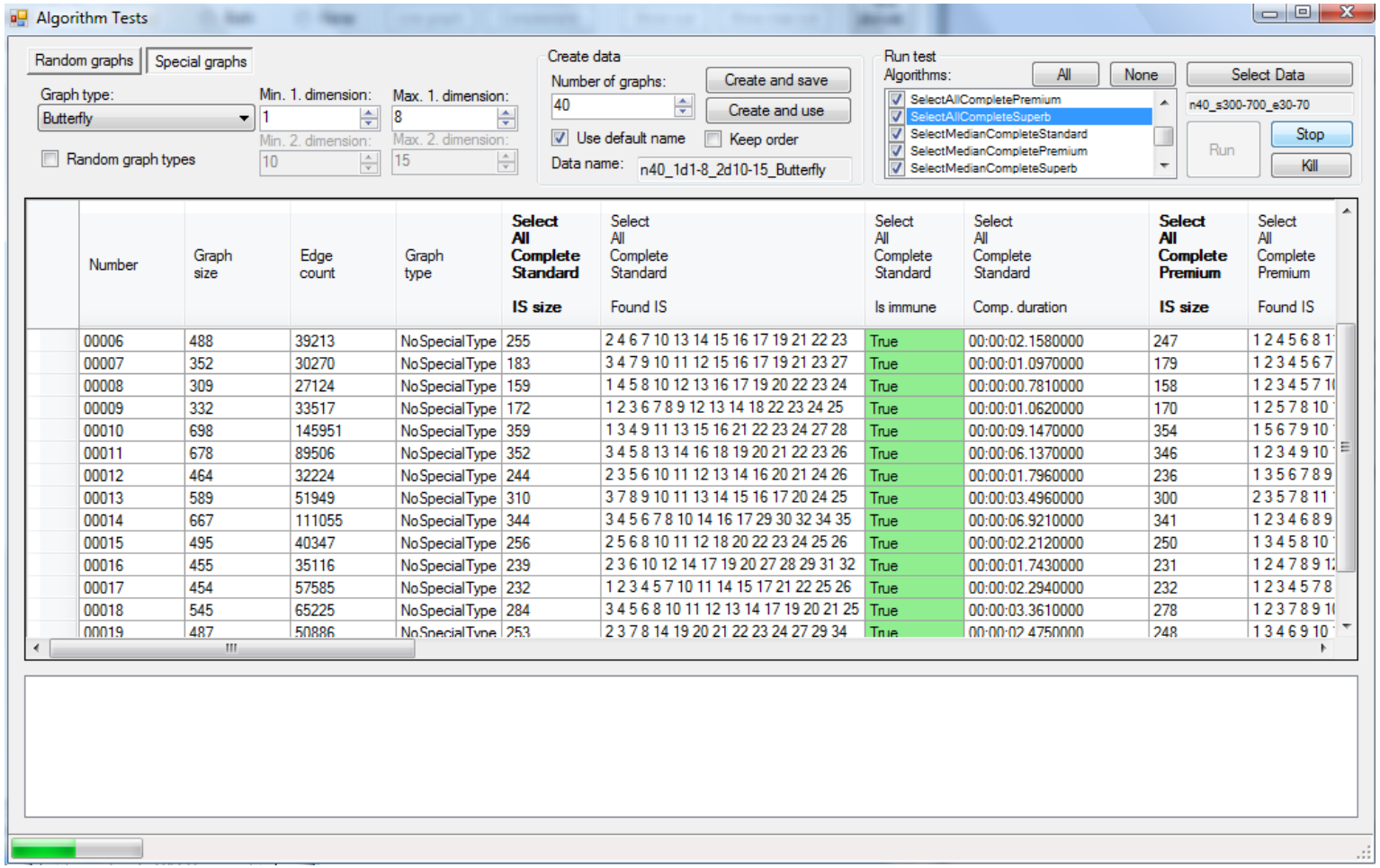
5.3 Rozšíriteľnosť

Program je písaný v objektovo orientovanom C# a dbal som na to, aby bol ľahko rozšíriteľný. Pridanie nového algoritmu je jednoduchá záležitosť tvorby novej triedy ktorá spĺňa špecifické rozhranie.

[23] V prípade algoritmov AMPL Integral a Relaxed je spustený samostatný proces cplex, ktorý treba zastaviť ručne cez task-manager. Tento neduh sa budem snažiť vyriešiť v dohľadnej dobe.



Obr. 5.3.1: Hlavné okno programu Immune Subgraph Analyzer



Obr. 5.3.2: Testovacie prostredie programu - okno sa spustí po kliknutí na “run tests”

Záver

NP-úplný charakter problému hľadania minimálneho imúnneho podgrafu spôsobuje, že ho vieme riešiť presne a rýchlo len pre malé grafy. Potvrdila to aj implementácia a testovanie Backtracku, ktorý s rastúcou veľkosťou vstupu exponenciálne zvyšoval čas výpočtu. Pre grafy s počtom vrcholov okolo 30 sa už doba výpočtu približovala hodine.

Druhým prístupom k riešeniu bolo skúsiť problém prepísať do systému lineárnych nerovníc, a riešiť ho voľne dostupnými solvermi pre lineárne programovanie. S využitím AMPL/CPLEX solveru bolo možné nájsť minimálny imúnny podgraf v relatívne krátkej dobe až pre dvojnásobne väčšie vstupy, ako v prípade Backtracku. Ak sa však uvažovalo neceločíselné lineárne programovanie, algoritmus sa výrazne urýchlil, no daňou boli chybné výsledky - podgrafy, ktoré neboli imúnne.

Tretím prístupom teda bolo nájsť heuristiky, ktoré by vedeli doplniť podgraf čo najmenším počtom vrcholov tak, aby bol imúnny. Celkovo tri typy heuristík odstupňované podľa kvality výsledku (Complete Standard, Premium a Superb) boli implementované v troch algoritmoch líšiacich sa podgrafom, ktorý heuristiky doplňovali: ARC (počiatočný podgraf je výstup z neceločíselného lineárneho programovania), SMC (počiatočný podgraf je vrchol, ktorého stupeň vrchola je mediánom stupňov vrcholov), SAC (za počiatočný podgraf sú vyskúšané všetky vrcholy, vrátená je najlepšia hodnota).

Aj pre veľké grafy dával relatívne presné výsledky SAC s dopĺňaním Superb. Najrýchlejší bol zase algoritmus SMC Standard. Doplnenie výstupu z neceločíselného lineárneho programovania malo dobrý pomer presnosť/rýchlosť.

Každá heuristika má ešte priestor na zlepšenie, prípadne by sa dali uvažovať ďalšie stratégie hľadania minimálneho imúnneho podgrafu. Obohatiť by sa dal aj samotný problém aby presnejšie modeloval realitu - napríklad tak, že by sa vo väčšinovom hlasovaní začal uvažovať aj názor jednotlivca, alebo akási náhodná zložka. Preto sa aj táto práca stáva vhodnou na prípadné budúce rozšírenie do diplomovej práce.

Literatúra

[KR] Rastislav Královič a Peter Ružička: *On immunity and Catastrophic Indices of Graphs*

[F] Paola Flocchini: *Contamination and Decontamination in Majority-Based Systems*

[HR04] Juraj Hromkovič: *Algorithmics for Hard Problems, 2nd Edition*

[F+02] Robert Fourer, David M. Gay, Brian W. Kernighan: *AMPL: A Modeling Language for Mathematical Programming*

[WTR] `http://mathworld.wolfram.com/Torus.html`

[WBF] `http://mathworld.wolfram.com/ButterflyGraph.html`

[WHY1] `http://mathworld.wolfram.com/Hypercube.html`

[WHY2] `http://en.wikipedia.org/wiki/Hypercube`

[WCC] `http://en.wikipedia.org/wiki/Cube-connected_cycles`

[WST] `http://www.texample.net/tikz/examples/star-graph/`