Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívne hľadanie minimálne dominujúcej množiny na reálnych sieťach

Diplomová práca

2015 **Viktor Tomkovič**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívne hľadanie minimálne dominujúcej množiny na reálnych sieťach

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky FMFI
Vedúci práce:	Mgr. Martin Čajági

Bratislava, 2015 Viktor Tomkovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Viktor Tomkovič
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Efektívne hľadanie minimálnej dominujúcej množiny na reálnych sieťach

Cieľ: Prvým z cieľov práce je implementovať aktuálne úplne efektívne algoritmy poskytujúce riešenia pre problém minimálnej dominujúcej množiny s rôznymi obmedzeniami na danú množinu. Ako je napríklad súvislosť, či minimálna vzdialenosť medzi vrcholmi v pokrytí. Tieto budú slúžiť ako benchmark pri menších grafoch. Druhým krokom je implementovať heuristické algoritmy, aproximatívne algoritmy a urobiť vzájomný pomer efektívnosť (čas-priestor) / veľkosť chyby plus porovnanie oproti úplným algoritmom z prvého cieľu.
Tretím cieľom je vyvinúť čo najefektívnejšie algoritmy pracujúce na sieťach v rádoch desiatok až stá tisícov vrcholov a miliónov hrán a otestovať ich na dátach reálnych existujúcich sietí.
Pod najneefektívnejším sa rozumie zlepšenie pre konkrétne typy sietí v čase alebo v chybe. Z nášho pohľadu je prioritnejší čas, pretože cieľom je rýchlo získavať požadované informácie aby sme ich vedeli spracovať a poskytnúť ako vstup pre ďalšie aplikácie.

Vedúci: Mgr. Martin Čajági
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 04.12.2012

Dátum schválenia: 04.12.2012 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Ďakujem.

Abstrakt

Táto práca skúma rôzne algoritmy na riešenie problému hľadania minimálnych dominantných množín (MDS). Konkrétne skúma ich využitie v sieťach malého sveta.
Kľúčové slová: minimálna dominujúca množina, MDS, algoritmy a dátové štruktúry, siete malého sveta.

Abstract

This work is.

Keywords: minimal dominating set, MDS, algorithms and date structures, small-world network.

Obsah

Zoznam tabuliek

Úvod

Toto je úvod do mojej diplomovej práce. Bude doplnený neskôr.

Kapitola 1

Definície

V tejto kapitole sme zaviedli niektoré pojmy, s ktorými sa budeme stretávať počas nasledujúcich kapitol. Sú to prevažne pojmy z teórie grafov. Keďže väčšina článkov, ktorými sme sa zaoberali v ďalších častiach je anglického pôvodu, rozhodli sme sa pre značenie uprednostniť knihu Graph Theory (Diestel 2000) pred knihou Grafové algoritmy (Plesník 1983).

Základným pojmom je pre nás množina, čo je súbor navzájom rôznych objektov. Množinu prirodzených čísel vrátane nuly označujeme $\mathbb{N}$. Množinu celých čísel označujeme $\mathbb{Z}$. Množinu reálnych čísel $\mathbb{R}$. Pre reálne číslo x označujeme hornú celú časť $\lceil x \rceil$ a označuje najmenšie celé číslo väčšie alebo rovné ako x. Podobne dolnú celú časť označujeme $\lfloor x \rfloor$ a označuje najväčšie celé číslo menšie alebo rovné ako x. Základ logaritmov napísaných ako „log“ je 2 a základ logaritmov napísaných ako „ln“ je e. Množina $=\{A_1, \ldots, A_k\}$ navzájom disjunktných podmnožín množina A je *rozdelenie* ak $A = \bigcup_{i=1}^{k} A_i$ a všetky i platí, že $A_i = \emptyset$.

1.1 Grafy

Graf $G = (V, E)$ je usporiadaná dvojica množiny vrcholov a množiny hrán, ktorá má následujúce vlastnosti:

- $V \cap E = \emptyset$ (hrany a vrcholy sú rozlíšiteľné)
- $E \subseteq \{\{u, v\} : u \neq v; u, v \in V\}$ (hrana spája dva vrcholy)

Graf sa zvyčajne znázorňuje nakreslením bodov pre každý vrchol a čiar medzi dvoma bodmi tam, kde existuje hrana. Rozmiestenie bodov a čiar nemá význam.

O grafe s množinou vrcholov V hovoríme, že je grafom *na* V. *Množina vrcholov grafu* G je označovaná $V(G)$ a to aj v prípade, kedy graf G má za množinu vrcholov inú množinu ako V. Napríklad, pre graf $H = (W, F)$ označujeme množinou vrcholov $V(H)$ a platí $V(H) = W$. Podobne označujeme *množinu hrán grafu* $E(G)$ (v hore uvedenom príklade platí $E(H) = F$). Pre jednoduchosť hovoríme, že vrchol (hrana) patrí grafu a nie množine vrcholov (hrán) grafu a preto sa občas vyskytuje označenie $v \in G$ a nie $v \in V(G)$.

Rád grafu je počet vrcholov v grafe a označujeme ho ako $|G|$. *Prázdny graf* $(\emptyset, \emptyset)$ označujeme $\emptyset$. Grafy rádu 0 alebo 1 sa označujeme ako *triviálne*.

Vrchol v je incidentný s hranou e ak platí $v \in e$. Hranu $\{x, y\}$ jednoduchšie označujeme ako xy. *Hranami* $X - Y$ označujeme množinu hrán $E(X, Y) = \{\{x, y\} : x \in X, y \in Y\}$. Namiesto $E(\{x\}, Y)$ píšeme $E(x, Y)$ a podobne aj namiesto $E(X, \{y\})$ píšeme $E(X, y)$. s Zápisom $E(v)$ označujeme $E(v, V(G))$ a hovoríme o *hranách vrchola* v.

Dva vrcholy x, y grafu G sú *susedia*, ak existuje hrana xy v grafe G. Dve hrany $e \neq f$ sú susedné, ak majú spoločný jeden vrchol. Ak sú všetky vrcholy v grafe navzájom susedné, graf je *kompletný* (alebo *úplný*). Kompletný graf s n vrcholmi označujeme K^n.

Majme dva grafy $G = (V, E)$ a $G' = (V', E')$. Potom $G \cup G' := (V \cup V', E \cup E')$. Podobne $G \cap G' := (V \cap V', E \cap E')$. Ak platí $G \cap G' = \emptyset$ tak hovoríme, že grafy sú *disjunktné*. Ak platí $V \subseteq V'$ a $E \subseteq E'$, tak potom je graf G' *podgrafom* grafu G. Zapisujeme $G' \subseteq G$.

Ak $G' \subseteq G$ a G' obsahuje všetky hrany $xy \in E$ pre $x, y \in V'$, tak hovoríme, že graf G' je *indukovaný podgraf* grafu G. Taktiež hovoríme, že V' *indukuje* G' na G a zapisujeme $G' =: G[V']$. Zápis $G[\{v\}]$ skracujeme na $G[v]$.

Ak je U nejaká množina vrcholov, tak zápisom $G - U$ (operátory $-$ a $\setminus$ budeme občas zamieňať) označujeme $G[V(G) \setminus U]$. Inými slovami graf $G - U$ dosiahneme tak, že z grafu G vymažeme všetky vrcholy z množiny U a všetky incidentné hrany k nim. Pre $G - \{u\}$ používame aj zápis $G - u$. Pre graf $G = (V, E)$ a množinu hrán $F = \{xy : x, y \in V\}$ zapisujeme $G + F = (V, E \cup F)$ a $G - F = (V, E \setminus F)$.

Komponent grafu je taký maximálny podgraf, kde medzi každou dvojicou vrcholov existuje cesta.

1.2 Vlastnosti grafu

Uvažujme o grafe $G = (V, E)$. Množinu susedov vrchola v označujeme $N_G(v)$. Pokiaľ to bude z kontextu jasné, tak iba skrátene $N(v)$. Zápis rozšírme na množiny. Pre množinu $U \subseteq V$ zapisujeme $N(U)$ množinu susedov všetkých vrcholov $u \in U$. Množinu $N(U)$ nazývame *susedmi* U. *Susedov vrátane vrchola* označujeme susedov vrchola s vrcholom samotným. Pre vrchol v susedov vrátane vrchola zapisujeme ako $N[v] := N(v) \cup \{v\}$. Podobne môžeme rozšíriť zápis aj na množiny. *Pokrytím* množiny $U \subseteq V$ nazývame množinu $N[U] := N(U) \cup U$.

Číslo $d_G(v) = d(v) := |E_G(v)|$ sa nazýva *stupeň* vrchola. Je to počet susedov vrchola (neplatí pre digrafy, multigrafy a iné zložité grafy, ktorými sa tu však nezaoberáme). Vrchol stupňa 0 je *izolovaný*. Číslo $\delta(G) := min\{d(v), v \in G\}$ je *minimálny stupeň* grafu G. Podobne číslo $\Delta(G) := max\{d(v), v \in G\}$ je *maximálny stupeň* grafu G.

1.3 Cesta a cyklus

Cesta je graf $P = (V, E)$ v tvare:

$$V = \{x_0, x_1, x_2, x_3, \ldots, x_k\} \qquad E = \{x_0x_1, x_1x_2, x_2x_3, \ldots, x_{k-1}x_k\},$$

kde všetky vrcholy x_i sú navzájom rôzne. Vrcholy x_0 a x_k sa nazývajú *konce* cesty a zvyšné vrcholy sú *vnútorné* vrcholy. Cestu zjednodušene označujeme sledom vrcholov: $P = x_0x_1x_2 \cdots x_k$. Aj keď nevieme rozlíšiť medzi cestami $P_1 = x_0x_1x_2 \cdots x_k$ a $P_2 = x_kx_{k-1}x_{k-2} \cdots x_0$, často si zvolíme jednu možnosť a hovoríme o *ceste z* x_0 *do* x_k (v tomto prípade sme si vybrali cestu P_1). *Dĺžka cesty* je číslo k (cesta môže mať dĺžku 0).

Cyklus je cesta $P = (V, E)$ v tvare

$$V = \{x_0, x_1, x_2, x_3, \ldots, x_{k-1}\} \qquad E = \{x_0x_1, x_1x_2, x_2x_3, \ldots, x_{k-2}x_{k-1}, x_{k-1}x_0\}$$

O ceste má zmysel hovoriť iba ak $k \geq 3$. *Dĺžka cyklu* je číslo k. Je to počet vrcholov (a zároveň aj hrán) v grafe.

V nasledujúcej časti si ukážeme dátové štruktúry, s ktorými sme v práci pracovali.

1.4 Strom

Graf, ktorý nemá cyklus, sa nazýva *acyklický*. Taktiež sa nazýva aj *les*. Les, ktorý má iba jeden komponent sa nazýva *strom*. Takže les je graf, ktorého komponenty sú stromy. Často sa nám oplatí poznať základné vlastnosti stromu. Tie najzákladnejšie sú zároveň aj zameniteľné a ú rôznymi obmenami definície stromu.

Následné tvrdenia sú zameniteľné pre graf T:

1. graf T je strom;
2. ľubovoľné dva vrcholy v grafe T sú spojené jedinečnou cestou v grafe T;
3. graf T je minimálne súvislý – graf T je spojitý, ale graf $T - e$ je nespojitý pre všetky hrany $e \in T$;
4. graf T je maxmimálne acyklický – graf T neobsahuje cyklus, ale graf $T + xy$ cyklus obsahuje pre ľubovoľné nesusedné vrcholy $x, y \in T$.

Jedinečnú cestu z vrcholu x do vrcholu y v strome T budeme označovať xTy. Z ekvivalencie bodov 1 a 3 vyplýva, že pre každý spojitý graf platí, že jeho ľubovoľný najmenej spojitý podgraf bude strom.

Vrcholy stupňa jeden sa nazývajú *listy*. Každý netriviálny strom má aspoň dva listy. Napríklad konce najdlhšej cesty. Jeden zaujímavý fakt – ak zo stromu odstránime list, ostane nám strom.

Občas je vhodne označiť jeden vrchol stromu špeciálne. A to ako *koreň*. Koreň potom tvorí základ stromu. Pokiaľ je koreň nemenný, tak hovoríme o *zakorenenom strome*. Vybratím koreňa k v strome T nám dovoľuje spraviť čiastočné usporiadanie na $V(T)$. Nech r je koreň stromu T, $r, x, y \in V(T)$, $x \leq y$ a platí, že $x \in rTy$, potom $\leq$ je čiastočné usporiadanie na množine $V(T)$. Zakorenené stromy sa zvyknú kresliť „po vrstvách“, kde je vidieť čiastočné usporiadanie vrcholov.

1.5 Bipartitné grafy

Nech $r \geq 2$ je prirodzené číslo. Graf $G = (V, E)$ sa nazýva *r-partitný*, ak môžeme V rozdeliť do r skupín tak, že každá hrana má koniec v inej skupine. Z toho vplýva, že vrcholy v jednej skupine nie sú susedné. Ak platí, že $r = 2$, nenazývame graf „2-partitný“ ale *bipartitný* (i keď obe pomenovania sú správne).

1.6 Toky

Veľa vecí z reálneho sveta sa dá modelovať pomocou grafov alebo štruktúr podobných grafom. Ide napríklad o elektrickú rozvodnú sieť, cestnú sieť, vlakovú/dráhovú sieť, komunikačnú sieť. Pri týchto znázorneniach vystupujú vždy dvojice komunikácií a „križovatiek“ (elektrické vedenie s trafostanicami, cesty s mestami, dráhy so zástavkami, linky s prepojovacími stanicami). Každá komunikácia má svoju kapacitu. V týchto štruktúrach má zmysel sa pýtať otázky, ako napríklad koľko veľa prúdu, zásob, dát dokáže prúdiť medzi dvoma vrcholmi. V teórii grafov hovoríme o tokoch.

Konkrétne komunikáciu si môžeme predstaviť ako hranu $e = xy$, ktorá vyjadruje aj smer prúdenia. K usporiadanej dvojici (x, y) môžeme priradiť hodnotu k vyjadrujúcu kapacitu komunikácie. Znamená to, že k jednotiek môže prúdiť z vrcholu x do vrcholu y. Alebo usporiadanej dvojici (x, y) môžeme priradiť zápornú hodnotu $-k$ a to znamená, že k jednotiek prúdi opačným smerom. To znamená, že pre zobrazenie $f : V^2 \to \mathbb{Z}$ (množina V označuje vrcholy), bude platiť, že $f(x, y) = -f(y, x)$, keď x a y sú susedné vrcholy.

Keď už máme vrcholy a komunikácie, musíme mať aj *zdroj* vecí, ktoré po komunikáciach budú prúdiť a taktiež miesta, z ktorých budú tieto veci odchádzať z modelu. Tie sa označujú ako *stoky*. Okrem týchto špeciálnych vrcholov platí, že

$$\sum_{y \in N(x)} f(x, y) = 0$$

Pokiaľ platia pre graf $G := (V, E)$ a zobrazenie $f : V^2 \to \mathbb{Z}$ vlastnosti $f(x, y) = -f(y, x)$ pre susedné vrcholy x a y a pre vrcholy mimo zdrojov a stôk, že $\sum_{y \in N(x)} f(x, y) = 0$, tak budeme hovoriť o *toku* na grafe G.

Graf sme si zadefinovali ako štruktúru, ktorá má hrany *neorientované*, to znamená, že nevieme rozlíšiť, kde hrana „začína“ a kde „končí“. Pri tokoch sme ale začali rozlišovať túto vlastnosť a tým

sa hrany stali *orientovanými*. Grafy sa nazývajú neorientované, pokiaľ sú aj ich hrany neorientované a naopak, ak sú orientované hrany, tak hovoríme aj o orientovanom grafe. V ďalšom texte budeme orientovanosť uvádzať iba vtedy, keď nebude jasne vyplávať z kontextu.

1.7 Komplexné siete

Ďalšie veci, ktoré môžeme pri modelovaní sietí z reálneho sveta skúmať sú veci ohľadne štruktúry. Má zmysel sa pýtať na to, aký je priemer grafu, či vieme určiť hierarchiu vrcholov a jej, aké komunikácie treba prerušiť na to, aby sa sieť rozpadla, kam treba nasadiť obmedzený počet špiónov, aby sme získali čo najviac informácií a podobne.

Pri modeloch reálnych sietí má zmysel zaoberať sa nielen stupňami jednotlivých vrcholov ale aj distribúciou stupňov a priemerným stupňom vrchola. *Priemerný stupeň grafu* $G = (V, E)$ označujeme $\overline{d_G} = \bar{d}$ a vypočítame ako:

$$\overline{d_G} = \bar{d} = \frac{\sum_{v \in V} d_v}{|V|}$$

K zadefinovaniu distribúcie budeme potrebovať ešte jednu funkciu. Táto funkcia vracia hodnotu 1 v bode 0 a vo všetkých ostatných bodoch vracia hodnotu 0. Takže dobre slúži ako filter. Označme ju τ a zadefinujme:

$$\tau(n) = \begin{cases} 1 & \text{ak } n = 1 \\ 0 & \text{inak} \end{cases}$$

Distribúcia stupňov vrcholov grafu G je funkcia $p(d)$ reprezentujúca pravdepodobnosť toho, že vrchol má stupeň d. Platí, že:

$$p(d) = \frac{\sum_{v \in V} \tau(d - d_v)}{|V|}$$

Suma vo funkcii prechádza všetkými vrcholmi a vďaka filtračnej funkcii τ spočíta, koľko je vrcholov stupňa d. Potom sa toto číslo predelí počtom vrcholov.

Zaujímavou vlastnosťou grafu je *klasterizačný koeficient vrchola*. Je definovaný ako pomer počtu hrán medzi susediacimi vrcholmi daného vrchola a všetkými možnými (aj potenciálnymi) hranami medzi susediacimi vrcholmi. Formálne, pre graf $G := (V, E)$ je *klasterizačný koeficient* vrchola v hodnota:

$$c_v = \frac{|E(N[v])|}{\binom{|N[v]|}{2}}$$

Priemerný klasterizačný koeficient $\bar{c}$ grafu G je definovaný ako:

$$\bar{c} = \frac{\sum_{v \in V}^{c_v}}{|V|}$$

Podobne ako distribúciu stupňov vrcholov zadefinujeme aj distribúciu klasterizačných koeficientov. *Distribúcia klasterizačných koeficientov grafu* G je funkcia $c(d)$, ktorá hovorí o tom, aký je priemer

klasterizačných koeficientov pre všetky vrcholy stupňa d. Vypočítame ho ako:

$$c(d) = \frac{\sum_{v \in V} \tau(d - d_v) c_v}{\sum_{v \in V} \tau(d - d_v)}$$

1.8 Vlastnosti komplexných sietí

V predošlej časti sme uviedli definície vlastností, ktorými vieme medzi sebou jednotlivé siete porovnávať a odlíšiť ich od seba.

Prvou vlastnosťou, ktorou môžeme grafy porovnávať a objavuje sa v reálnych sieťach, je to, čomu sa hovorí *malý svet*. Graf je sieťou malého sveta vtedy, keď je jeho priemer malý a zároveň má vysokú klasterizáciu vrcholov.

Veľa z reálnych sietí má ďalšiu spoločnú vlastnosť. Ich distribúcia stupňov vrcholov klesá mocninovo. Môžeme zapísať, že pre distribúciu platí:

$$p(d) = d^{-\alpha}$$

Siete sa líšia v konštante α, ale iba trochu. V drvivej väčšine platí, že $2 \leq \alpha \leq 3$. Keďže v takýchto sieťach by mal náhodný „výsek“ grafu podobné vlastnosti, tak táto vlastnosť sa nazýva aj *bezškálovosť*.

1.9 Modely sietí

V tejto časti si ukážeme spôsob vytvárania grafov s nejakou vlastnosťou. Keďže sa zaoberáme porovnávaním „obyčajných“ grafov so sieťami malého sveta, zameriame sa najmä na tieto dva typy.

Náhodné grafy môžeme vytvoriť napríklad týmito dvoma spôsobmi. Prvým je, že pevne určíme, koľko bude mať graf vrcholov a určíme, koľko hrán má mať graf. Potom náhodne vyberieme počet určených hrán. Druhým spôsobom je, že pevne určíme počet vrcholov grafu a pravdepodobnosť, s ako sa každá hrana vyberie. Grafy vytvorené týmito postupmi nepopisujú (nemajú dostatočné vlastnosti) siete malého sveta dostatočne, preto ich nebudeme považovať za siete malého sveta.

Barabási – Albertov model alebo aj model s preferenčným napájaním je spôsob vytvárania grafov, ktorý zohľadňuje stupeň vrcholov grafu. Na začiatku má graf daný počet náhodne spojených vrcholov a v každom kroku sa pridá jeden vrchol a niekoľko hrán spájajú tento pridaný vrchol s existujúcimi vrcholmi v grafe. Pravdepodobnosť toho, že sa hraná spojí s vrcholom je priamo úmerná stupňu vrchola. Tento postup vedie ku tvorbe grafom podobným vlastnosťami s komplexnými sieťami.

1.10 Asymptotická zložitosť

V tejto práci sa zaoberáme najmä prácou s reálnymi dátami a konkrétnymi algoritmami. Ale aj pri tomto zameraní je dobre, keď vieme približne odhadnúť s akými veľkými množstvami dát algoritmus pracuje, ako zhruba veľa operácií procesor vyžaduje na daný výpočet v závislosti od veľkosti vstupných

dát. Inými slovami, potrebujeme buď vedieť porovnať dve funkcie alebo nejakú funkciu zatriediť medzi ostatné.

Na tieto požiadavky vznikla *„O-notácia“* (Bachmann 1894), ktorá vyjadruje asymptotický rast funkcií. Uplatňuje sa v informatike a matematike. Ide o vytvorenie rôznych tried funkcií. Aj keď v informatike sa zväčša porovnáva rast počtu krokov od veľkosti vstupu a veľkosť vstupu aj počet krokov sú diskrétne údaje, uvedieme tu definíciu pre funkcie, ktoré majú svoj definičný obor aj obor hodnôt v množinách reálnych čísel.

Vyjadrime *asymptotický odhad zhora*. Majme dve funkcie $f, g : \mathbb{R} \to \mathbb{R}$. Potom:

$$\exists c \geq 0\ \exists x_0\ \forall x \geq x_0 : f(x) \leq c \cdot g(x) \iff f(x) \in O(g(x))$$

Veľmi často sa v informatike zamieňa zápis $f(x) \in O(g(x))$ so zápisom $f(x) = O(g(x))$ a hovorí sa, že „$f(x)$ je $O(g(x))$“. Napríklad, ak funkcia $f(x) = 5x^2 + 3x + 7$ a funkcia $g(x) = x^2$, tak sa hovorí, že „$f(x)$ je $O(x^2)$“. Taktiež môžeme povedať, že $f(x)$ rastie kvadraticky.

Vidno, že funkcia $g(x)$ nejakým spôsobom ohraničuje funkciu $f(x)$ zhora. Asymptotický odhad pomocou $O(g(x))$ nám ale nehovorí nič o tom, ako veľmi zhora je funkcia $f(x)$ ohraničená. Zväčša sa však používa dostatočne tesný odhad.

Ak však chceme byť v našom odhade presnejší, existuje aj *asymptoticky tesný odhad* a je definovaný ako:

$$\exists c_1 \geq 0\ \exists c_2 \geq 0\ \exists x_0\ \forall x \geq x_0 : c_1 \cdot g(x) \leq f(x) \wedge f(x) \leq c_2 \cdot g(x) \iff f(x) \in \Theta(g(x))$$

V praxi sa používa menej. Je však vhodný, ak chceme ukázať najtesnejší odhad pre daný algoritmus. Ak existuje. Ak neexistuje, alebo sme ho zatiaľ nenašli, tak neostáva nič iné, ako spraviť tesný horný a dolný odhad. *Asymptotický dolný odhad* je definovaný ako:

$$\exists c \geq 0\ \exists x_0\ \forall x \geq x_0 : c \cdot g(x) \leq f(x) \iff f(x) \in \Omega(g(x))$$

Z tejto definície je vidieť, že *asymptoticky tesný odhad* $\Theta(g(x))$ môžeme vyjadriť aj ako:

$$f(x) \in O(g(x)) \wedge f(x) \in \Omega(g(x)) \iff f(x) \in \Theta(g(x))$$

Tento vzťah dobre vyjadruje reálnu snahu pri určovaní asymptotickej zložitosti algoritmov. Algoritmus sa hrubo odhadne zhora aj zdola a postupne sa tieto odhadu spresňujú.

V matimatike sa používa ešte jeden asymptotický vzťah a to *asymptotická rovnosť*. Je to podobný vzťah ako tesný asymptotický odhad, opäť používa dve funkcie $f(x), g(x)$ na reálnych číslach a je definovaný ako:

$$\forall \varepsilon \geq 0 \exists n_0 \forall n > n_0 : \left| \frac{f(x)}{g(x)} - 1 \right| < \varepsilon \iff f(x) \sim g(x)$$

Z definície je vidieť, že pokiaľ sú dve funkcie asymptoticky rovnaké, tak budú aj navzájom asymptoticky tesné.

1.11 Ostatné definície

V tejto časti uvádzame iné definície, o ktorých si myslíme, že sú užitočné.

Maximum, respektíve *minimum* funkcie je najväčšia, resp. najmenšia hodnota, ktorú funkcia nadobúda. Pre funkciu $f(x)$ platí, že:

$$\max f(x) := f(x) \mid \forall y : f(y) \leq f(x)$$
$$\min f(x) := f(x) \mid \forall y : f(y) \geq f(x)$$

Argumentom maxima, respektíve *argumentom minima* funkcie sú prvky z definičného oboru funkcie, v ktorom funkcia nadobúda maximum, resp. minimum. Pre funkciu $f(x)$ platí, že:

$$\arg\max_x f(x) := \{x \mid \forall y : f(y) \leq f(x)\}$$
$$\arg\min_x f(x) := \{x \mid \forall y : f(y) \geq f(x)\}$$

V následujúcich kapitolách sme sa zamerali a prebrali si jednotlivé existujúce algoritmy, ktoré boli implementované.

Kapitola 2

Prehľad algoritmov

V tejto kapitole si spravíme prehľad algoritmov, ktoré existujú na nájdenie minimálnej dominujúcej množiny. Najprv zadefinujeme minimálnu dominujúcu množinu, neskôr určíme požiadavky na algoritmy a potom popíšeme jednotlivé konkrétne algoritmy.

2.1 Dominujúce množiny

Dominujúca množina S na grafe $G = (V, E)$ je podmnožina množiny vrcholov V grafu G taká, že každý vrchol grafu sa v množine nachádza alebo je susedný s dominujúcou množinou. Pre množinu platí: $N[S] = V$. Z definície je zrejmé, že o dominujúcich množinách má zmysel hovoriť iba pri grafoch s konečným počtom vrcholov.

Minimálna dominujúca množina S_M je dominujúca množina s najmenšou kardinalitou.

Dominančé číslo $\gamma(G)$ grafu G je kardinalita minimálnej dominujúcej množiny. Ak je množina S_M minimálnou dominujúcou množinou, tak platí, že $\gamma(G) = |S_M|$.

Na tomto mieste zadefinujeme aj vrcholové pokrytie. Uvádzame ho tu preto, lebo problém nájdenia vrcholového pokrytia súvisí s problémom nájdenia minimálnej dominujúcej množiny. *Vrcholové pokrytie* S' na grafe $G = (V, E)$ je podmnožina množiny vrcholov V grafu G taká, že každá hrana $xy \in E$ je incidentná s vrcholom vrcholového pokrytia. Pre množinu platí: $\forall xy \in E : x \in S' \vee y \in S'$

Podobne ako pri minimálnej dominujúcej množine, existuje aj minimálne vrcholové pokrytie.

Jedným zo spôsobov, ako vyriešiť problém nájdenia minimálnej dominujúcej množiny je previesť ho na problém množinového pokrytia. *Množinové pokrytie* je súbor množín, ktorých zjednotenie obsahuje všetky prvky univerza. Formálne je daná usporiadaná dvojica $(\mathcal{S}, \mathcal{U})$, kde:

- množina množín $\mathcal{S}$ obsahuje množiny $S_1, S_2, S_3, ..., S_n$ také, že $\bigcup_{i=1}^{n} S_i = \mathcal{U}$;
- množina $\mathcal{U}$ je množina všetkých prvkov a nazýva sa *univerzum*.

Množinové pokrytie je podmnožina $\mathcal{C} \subseteq \mathcal{S}$ množín, pre ktorú platí, že $\bigcup_{C \in \mathcal{C}} C = \mathcal{U}$.

2.2 Požiadavky na algoritmy

Táto práca ma za úlohu nájsť vhodný algorimus na hľadanie minimálnej dominujúcej množiny. Ale pre reálne dáta. To znamená, že grafy, na ktorých budeme minimálnu dominujúcu množinu hľadať majú veľa vrcholov. Avšak problém nájdenia minimálnej dominujúcej množiny na grafe sa dá redukovať na problém vrcholového pokrytia, o ktorom vieme, že je NP-ťažký. To znamená, že na vyrátanie minimálnej dominujúcej množiny treba veľmi veľa výpočtového času na súčasných počítačoch. Keďže pre reálne požiadavky je častokrát lepší nejaký, aj keď nie optimálny výsledok, tak sme sa rozhodli, že algoritmus nemusí dávať optimálny výsledok, ale môže dať približný výsledok v rozumnom čase. Rozumný čas však neurčujeme absolútne, keďže počítače sa vyvíjajú a výpočtová sila sa zväčšuje, ale relatívne vzhľadom na ostatné algoritmy.

2.2.1 Výhoda sieti malého sveta

V tejto časti ukážeme dôkaz, že v sieťach malého sveta je počet hrán rádovo rovnaký ako počet vrcholov. A teda siete malého sveta sú riedke grafy.

Budeme vychádzať z toho, že pre distribúciu stupňov vrcholov platí:

$$p(d) = kd^{-a} + q, 2 \leq a \leq 3$$

Pre dostatočne veľké siete môžeme nespojitú funkciu $p : \mathbb{N} \leftarrow \mathbb{N}$ aproximovať spojitou verziou $p : \mathbb{R} \leftarrow \mathbb{R}$. Ďalej o q vieme, že $q \leq 1/V$, kde V je počet vrcholov grafu. Pre počet hrán v grafe $G = (V, E)$ platí, že je to polovica súčtu stupňov grafu. Teda môžeme napísať, že:

$$|E| = \frac{1}{2} \sum_{d=1}^{\Delta(G)} p(d) \sim \frac{1}{2} \int_{1}^{\Delta(G)} p(x)\,\mathrm{d}x$$

Úpravou integrálu dostaneme:

$$\begin{aligned} \frac{1}{2} \int_{1}^{\Delta(G)} p(x)\,\mathrm{d}x &= \frac{1}{2} \int_{1}^{\Delta(G)} kd^{-a} + q\,\mathrm{d}x \\ &= \frac{1}{2} \left[\frac{1}{a-1} \left(-kx^{-a+1} + qx\ (a-1) \right) \right]_{1}^{|V|} \end{aligned}$$

Keďže $2 \leq a \leq 3$ a integrujeme medzi 1 a $\Delta(G)$, tak môžeme ohraničiť výpočet:

$$\frac{1}{2} \left[\frac{1}{a-1} \left(-kx^{-a+1} + qx\ (a-1) \right) \right]_{1}^{\Delta(G)} \leq \frac{1}{2} \left[-\left(kx^{-1} - qx \right) \right]_{1}^{\Delta(G)}$$

Pre konštantu q platí, že $q \leq \frac{1}{\Delta(G)}$. Inak by súčet v distribúcií bol väčší ako počet vrcholov.

$$\frac{1}{2}\left[-\left(kx^{-1}-qx\right)\right]_1^{\Delta(G)} \leq \frac{1}{2}\left[-\left(kx^{-1}-\frac{x}{\Delta(G)}\right)\right]_1^{\Delta(G)}$$

Ďalšou úpravou dostaneme:

$$\begin{aligned}\frac{1}{2}\left[-\left(kx^{-1}-\frac{x}{\Delta(G)}\right)\right]_1^{\Delta(G)} &= \frac{1}{2}\left\{\left[-\left(k\Delta(G)^{-1}-\frac{\Delta(G)}{\Delta(G)}\right)\right]-\left[-\left(k1^{-1}-\frac{1}{\Delta(G)}\right)\right]\right\}\\ &= \frac{1}{2}\left\{\left[-\left(k\Delta(G)^{-1}-1\right)\right]-\left[-\left(k-\frac{1}{\Delta(G)}\right)\right]\right\}\\ &= \frac{1}{2}\left\{(k+1)-\frac{k+1}{\Delta(G)}\right\}\\ &= \frac{1}{2}(k+1)\left(1-\frac{1}{\Delta(G)}\right)\end{aligned}$$

Konštanta k nemôže byť väčšia ako $|V|$. Inak by vrcholov stupňa 1 bolo viac ako vrcholov v grafe. Triviálne platí, že $\Delta(G) \leq |V|$. Takže odhad môžeme ďalej upraviť:

$$\frac{1}{2}(k+1)\left(1-\frac{1}{\Delta(G)}\right) = \frac{1}{2}(|V|+1)\left(1-\frac{1}{|V|}\right)$$

Takže platí, že:

$$|E| \sim \frac{1}{2}(|V|+1)\left(1-\frac{1}{|V|}\right)$$

a teda aj, že počet hrán je $O(V)$.

Táto vlastnosť sietí malého sveta nám pomôže najmä pri voľnejšom výbere heuristík a pravidiel v algoritmoch uvedených ďalej. Keďže algoritmy vo všeobecnosti neuvažujú s týmto faktom, dáva nám možnosť vytvoriť zložitejší a stále rovnako efektívny algoritmus.

2.2.2 Test, či je množina dominujúcou

Keďže graf máme reprezentovaný susednosťou vrcholov, tak priamočiary algoritmus na zistenie, či je množina dominujúcou vyzerá následovne:

1. Pre každý vrchol testovanej množiny pridaj do výslednej množiny všetkých susedov vrchola;
2. porovnaj výslednú množinu s množinou vrcholov grafu.

Nech má graf n vrcholov, m hrán a testovaná množina $s \leq n$ vrcholov. Potom v prvom kroku vykonáme $O(sm)$ operácií a v druhom $O(n^2)$ operácií. Test teda trvá $O(sm+n^2)$ operácií. Keďže

počet vrcholov v testovanej množine môže byť rovnaký ako počet vrcholov grafu, tak odhad môžeme upraviť na $O(nm + n^2)$. Keďže pre siete malého sveta platí $O(m) = O(n)$, tak časový odhad pre siete malého sveta je $O(n^2)$.

2.3 Skúšanie všetkých možností

Prvým algoritmom, ktorý je v prehľade, je najzákladnejší algoritmu vyskúšania všetkých možností. Tento algoritmus budeme volať aj *naivný*. Algoritmus vždy poskytne správny výsledok, ale výpočet bude trvať dlho. Je to však dobrý začiatok k ďalším algoritmom. Pracuje podľa krokov:

1. Vyber podmnožinu grafu;
2. otestuj, či je podmnožina dominujúcou množinou;
3. ak je podmnožina dominujúcou množinou a zároveň má najmenšiu kardinalitu, zapamätaj si ju;
4. opakuj, kým nevyberieš všetky možné podmnožiny práve raz;
5. jednou z minimálnych dominujúcich množín je zapamätaná množina a dominančné číslo grafu je jej kardinalita.

Algoritmus je pomalý hlavne kvôli kroku 4 – všetkých možných podmnožín je 2^n, takže výsledný algoritmus skúšania všetkých možností bude $\Omega(2^n)$. Súčasné počítače zvládnu úlohu v rozumnom čase vyrátať pre $n \leq 40$.

Tam, kde je najväčšia slabina, je zväčša aj najväčší priestor na zlepšenie. Existujú mnohé zlepšenia, ktoré zrýchlia algoritmus nielen v priemernom (resp. reálnom) prípade, ale aj zlepšia teoretický odhad.

2.4 Heuristiky pre algoritmus skúšania všetkých možností

V tejto sekcii si povieme niečo o možných heuristikách pre naivný algoritmus. *Heuristika* v algoritme je nejaký prvok, zväčša zo skúsenosti z reálneho sveta, o ktorom predpokladáme, že nám pomôže zrýchliť výpočet. Aj keď obvykle nezlepšuje asymptotickú zložitosť, heuristiky sa snažia byť navrhnuté tak, aby vo väčšine prípad zrýchlili beh algoritmu.

Častým príkladom a aplikáciou je jedna z heuristík na hľadanie najkratšej cesty. Možná heuristika je, že prehľadávanie bude uprednostňovať cesty smerujúce k hľadanému bodu. Tu si všimnime, že pri heuristike potrebujeme poznať informáciu, ako je daná voľba dobrá. Pokiaľ nie je medzi hľadaným bodom a bodom, z ktorého hľadáme prekážka, algoritmus prehľadá oveľa menej hrán, ako pri bežnom prehľadávaní. Samozrejme, pokiaľ sme v bludisku a najkratšia cesta vedie „opačným“ smerom, tak prehľadáme všetky hrany, kým sa dostaneme k cieľu.

Podobne je to aj pri hľadaní minimálnej dominujúcej množiny. Dobrým odhadom sa javí možnosť vybrať do potenciálnej množiny S ten vrchol v, ktorý vie pokryť čo najviac vrcholov, teda sa javí byť

čo najbližšie k cieľu. Čiže hľadáme $\arg\max_v |N[S \cup v]|$ pre $v \in V$. Spojenie tejto heuristiky s vedomosťou, že siete malého sveta majú veľa klastrov ešte upevňuje predpoklad, že táto heuristika bude dávať na sieťach malého sveta rýchlejšie výsledky a „zlých" prípadov bude málo.

V pôvodnom algoritme, ktorý skúša všetky možnosti to znamená, že si pamätáme dočasný najlepší výsledok a neskúšame tie možnosti, ktoré obsahujú viac vrcholov ako dočasný najlepší výsledok. Samotné vynechanie tých možností, ktoré majú viac prvkov ako momentálny najlepší výsledok je veľké zrýchlenie, keďže pre súvislý graf s N vrcholmi platí, že veľkosť minimálnej dominujúcej množiny je nanajvýš $N/2$.

2.5 Prevedenie na problém množinového pokrytia

Ďalšou možnosťou, ako presne nájsť minimálnu dominujúcu množinu je previesť problém na problém množinového pokrytia, vyriešiť ten a výsledok opäť previesť. Tento spôsob navrhol Grandoni (2004) vo svojej dizertačnej práci. Dôvodom prevodu je fakt, že problém množinového pokrytia bol v minulosti oveľa skúmanejším problémom. Keďže pri exponenciálnych algoritmoch celkom záleží aj na konštante pri exponente, odhad tohto algoritmu spresnil Fomin, Grandoni a Kratsch (2005).

Grandoni (2004) previedol problém minimálnej dominujúcej množiny na hľadanie minimálneho množinového pokrytia na grafe $G := (V, E)$ tak, že množiny predstavovali vrchol a jeho susedov. Univerzom je množina vrcholov grafu. Takže vytvoril usporiadanú dvojicu $(N[v] : v \in V, V)$, čo je vstupný údaj pre hľadanie množinového pokrytia.

2.5.1 Pomocné tvrdenia

V algoritme využijeme následujúce tvrdenia, ktoré platia v každej dvojici $(\mathcal{S}, \mathcal{U})$ problému množinového pokrytia:

1. pre každé dve navzájom odlišné množiny S a R také, že $S, R \in \mathcal{S}, S \subseteq R$, platí, že existuje vrcholové pokrytie, ktoré neobsahuje S;

2. ak existuje prvok $u \in U$, ktorý patrí iba do jednej množiny $S \in \mathcal{S}$, tak táto množina S patrí do každého vrcholového pokrytia.

Zaujímavým pozorovaním je, že každá podmnožina s kardinalitou jeden, spĺňa práve jedno z tvrdení.

V prípade, že všetky podmnožiny $S \in \mathcal{S}$ sú dvojprvkové, problém sa dá redukovať na hľadanie maximálneho párenia. *Párenie* v grafe G je množina hrán M taká, že hrany nemajú spoločný ani jeden vrchol. *Maximálne párenie* je párenie s najväčšou mohutnosťou.

Z inštancie problému množinového pokrytia $(\mathcal{S}, \mathcal{U})$, kde $|S| = 2, S \in \mathcal{S}$, vieme spraviť graf $G' = (V, E)$ tak, že množinou vrcholov bude množina univerza, čiže $V = \mathcal{U}$ a množinu hrán E budú tvoriť podmnožiny $S \in \mathcal{U}$. Minimálnu dominujúcu množinu na grafe reprezentovanú dvojicou $(\mathcal{S}, \mathcal{U})$ vieme určiť pomocou minimálneho hranového pokrytia na grafe G'. Minimálne hranové pokrytie

zase vieme získať pomocou maximáneho párenia. Keďže hranám v grafe G' zodpovedá práve jedna podmnožina $S \in \mathcal{S}$ v dvojici $(\mathcal{S}, \mathcal{U})$, tak vieme určiť maximálne množinové pokrytie.

2.5.2 Algoritmus

Samotný algoritmus hľadania minimálneho množinového pokrytia je založený na princípe rozdeľuj a panuj. Pracujeme s inštanciou $(\mathcal{S}, \mathcal{U})$. Jeho triviálny prípad je, keď $|\mathcal{S}| = 0$. Pred rozdelením sa snaží odstrániť podmnožiny kardinality 1. Ak majú všetky podmnožiny mohutnosť práve dva, problém sa prevedie na problém maximálneho párenia. Ak má nejaká množina kardinalitu väčšiu ako 2 nastáva delenie. Spájanie výsledkov spočíva iba v porovnaní, ktorá z vetiev dala lepší výsledok. Výstupom algoritmu je množina podmnožín, ktorá tvorím minimálne množinové pokrytie. Algoritmus dostáva za vstup iba množinu podmnožín $\mathcal{S}$ a vyzerá následovne (v algoritme sú kvôli prehľadnosti podmnožiny nazývané množinami):

1. ak je množina množín $\mathcal{S}$ prázdna, vráť prázdnu množinu;
2. ak je nejaká množina R podmnožinou inej množiny S, vráť výsledok algoritmu pre $\mathcal{S} \setminus R$;
3. ak existuje jedinečný prvok medzi množinami a ten je obsiahnutý (iba) v množine R, vráť výsledok algoritmu pre $\mathcal{S} \setminus R$ zjednotený s množinou R;
4. ak majú všetky množiny mohutnosť dva, tak vráť výsledok z maximálneho párenia;
5. inak spusti dvakrát algoritmus s vynechaním ľubovoľnej množiny R; raz ju vynechaj s množiny množín $\mathcal{S}$, vtedy sa do výsledku nezaráta; druhý raz odstráň zo všetkých množín prvky množiny R a zarátaj množinu do výsledku; porovnaj, pre ktoré spustenie dal algoritmus lepší výsledok a ten vráť.

Ako vidno, krok 1 je triviálny prípad. Kroky 2, 3 a 4 sú popísané vyššie. V kroku 5 je slovo ľubovoľný. Toto správanie sa dá zameniť pomocou nejakej heuristiky. Prirodzene sa núka skúsiť vyberať množiny s najväčšou mohutnosťou najskôr, keďže tie redukujú ostatné množiny najviac.

2.6 Pažravý algoritmus

V predchádzajúcich častiach sme si popísali algoritmy, ktoré rátajú presné výsledky. Na veľkých sieťach, napríklad na zobrazeniach skutočných sietí, sú však nepoužiteľné. Preto si musíme vystačiť iba s približným výsledkom.

Prvým algoritmom, ktorý uvedieme a bude výsledok určovať iba približne, bude jednoduchý pažravý algoritmus, od ktorého si postupne odvodíme iné a použijeme viacero postupov, na riešenie problému nájdenia minimálnej dominujúcej množiny. Pre pripomenutie – *pokrytie vrchola* je množina všetkých jeho susedov vrátane vrchola a *pokrytie množiny* je prienik pokrytie vrcholov množiny.

Algoritmus vyzerá následovne:

1. na začiatku je výsledná množina prázdna;
2. do výslednej množiny pridaj vrchol, ktorý pokryje čo najviac ešte nepokrytých vrcholov;
3. opakuj predošlý bod, až kým nebude výsledná množina pokrývať všetky vrcholy;
4. vráť výslednú množinu.

Za zmienku stojí, že v bode 3 algoritmus musí určiť, či výsledná množina už pokrýva všetky vrcholy. Toto z tohto algoritmu a jeho variatov robí algoritmy, ktoré sú zložité najmenej kvadraticky od počtu vrcholov.

Ako aj pri algoritmoch s exaktnými výsledkami, aj tu môžeme použiť nejaké heuristiky na druhý krok. O tých si povieme neskôr.

2.7 Distribuovaný algoritmus

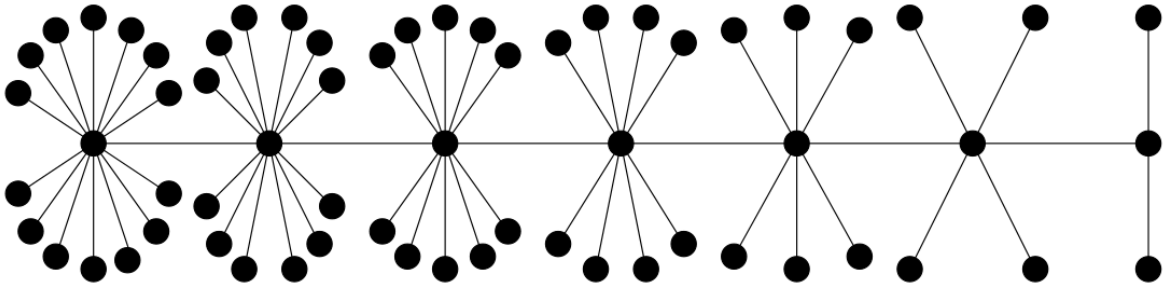
V tejto časti si ukážeme prerobenie pažravého algoritmu na distribuovaný, ktorý navrhol Kuhn (2011). Využijeme pri tom jedno pozorovanie. V bode 2 sa vyberie vrchol, ktorý pokrýva čo najviac ešte nepokrytých vrcholov. Počet ešte nepokrytých vrcholov môže ovplyvniť iba výber vrcholov zo vzdialenosti najviac 2. Preto, ak vrchol môže pokryť najviac vrcholov s pomedzi vrcholov vzdialených najviac 2, tak sa tento vrchol môže vybrať do výslednej množiny pred ostatnými.

Toto pozorovanie vedie ku konštrukcii veľmi jednoduchého algoritmu (v každom vrchole):

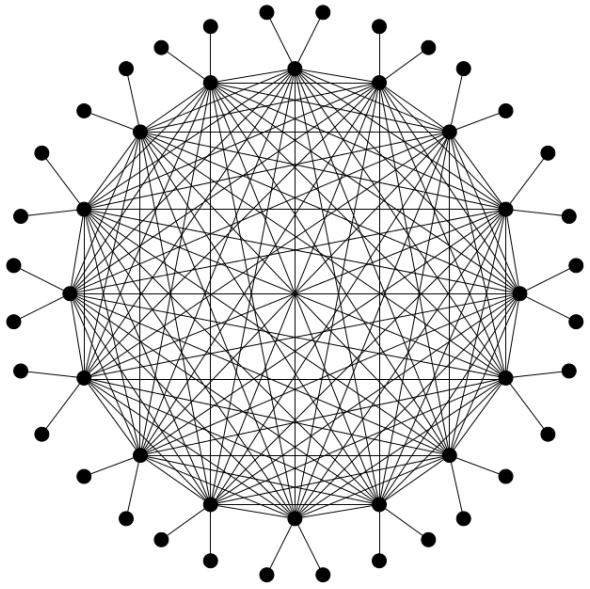
1. pre svoj vrchol vypočítaj počet ešte nepokrytých vrcholov;
2. tento počet pošli algoritmom vo vrcholoch najviac vzdialených dve hrany;
3. ak vrchol pokrýva najväčší počet vrcholov vo vzdialenosti najviac dva, tak vrchol pridaj do dominujúcej množiny (ak je takých vrcholov viac, rozhodni náhodne – napríklad podľa ID);
4. opakuj od bodu 1, až kým vrchol nebude mať všetkých susedov pokrytých;

Tento algoritmus teoreticky funguje veľmi dobre. Počet vykonaných krokov bude lineárne úmerný počtu vrcholov grafu, keďže v každom kroku sa aspoň jeden vrchol vyberie. V skutočnosti má okrem veľa implementačných problémov uvedených v kapitole 5 aj zlý počet krokov výpočtu pre niektoré typy grafov.

Na obrázku 2.1 je vidieť jeden z možných grafov, na ktorom beží algoritmus pomaly. Algoritmus síce vyberie minimálnu dominujúcu množinu presne. Tá pozostáva z vrcholov na jej „osi“. Ale v jednom opakovaní cyklu algoritmus vyberie najviac jeden vrchol, postupne zľava doprava. Je vidno, že algoritmus je $\Theta(N)$, kde N je počet vrcholov grafu. Tento problém môžeme vyriešiť tak, že povolíme vybrať aj vrcholy s menším pokrytím vrchola. Tým síce zhoršíme aproximačný faktor, ale nie o veľa. Konkrétne, ak zhoršíme aproximačný faktor dvojnásobne, môžeme zaokrúhliť pokrytie



Obrázok 2.1: *Graf, na ktorom beží distribuovaný algoritmus pomaly.* Algoritmus v takejto situácií vyberá vrcholy postupne zľava doprava a v každom kroku môže vybrať iba jeden vrchol.



Obrázok 2.2: *Graf, na ktorom beží distribuovaný algoritmus pomaly.* Algoritmus v takejto situácií vyberá vrcholy náhodne, ale v každom kroku iba jeden, pretože všetci potenciálny kandidáti sa blokujú.

vrchola na najbližšiu mocninu dvojky. Pri vyberaní najlepšieho kandidáta môžeme potom vybrať aj vrchol s menším pokrytím.

Druhou zlou možnosťou pre algoritmus je, keď je v grafe veľký klaster s vrcholmi rovnakého stupňa. Možnosť je znázornená na obrázku 2.2. Algoritmus nemôže naraz vybrať všetky vrcholy stupňa 3. Bráni mu v tom podmienka, že sa dá vybrať iba jeden z vrcholov spomedzi okolia vzdialenosti dva. Tu pomôže, ak povolíme vybrať viac vrcholov naraz, pokiaľ si príliš neprekážajú.

Algoritmus s týmito dvoma uvedenými vylepšeniami je zložitejší. Horným ohraničením pokrytia budeme nazývať najbližšiu hornú mocninu dvojky. Podobne, dolným ohraničením pokrytia budeme označovať najbližšiu dolnú mocninu dvojky.

1. pre svoj vrchol vypočítaj počet ešte nepokrytých vrcholov;
2. vypočítaj dolné ohraničenie pokrytia;
3. zober dolné ohraničenia susedov vzdialených najviac 2;
4. ak je dolné ohraničenie také, ako najväčšie dolné ohraničenie spomedzi susedov, tak sa vrchol má šancu stať vybratým;
5. táto šanca je nepriamo úmerná počtu vrcholov s najväčším dolným pokrytím;
6. vyber vrchol s pravdepodobnosťou vypočítanou vyššie;
7. vypočítaj hodnotu c – počet kandidátov na vybratie spomedzi pokrývajúcich vrcholov;
8. ak je vrchol vybratý a súčet hodnôt c pre pokrývajúce vrcholy je menší alebo rovný ako trojnásobok počtu pokrývajúcich vrcholov, tak vrchol pridaj do medzivýsledku;
9. opakuj od bodu 1, až kým vrchol nebude mať všetkých susedov pokrytých;

Takto upravený algoritmus zvláda vypočítať dominujúce množiny o čosi rýchlejšie. Keďže množstvo klastrov a počet vrcholov s rovnakým stupňom je v sieťach malého sveta veľký, tak je predpoklad, že tento upravený algoritmus bude bežať oveľa rýchlejšie na sieťach malého sveta ako neupravený.

Vďalšej časti sa už nezaoberáme distribuovanými algoritmami ale jednoduchým pažravým algoritmom s rôznymi heuristikami.

2.8 Heuristiky pre pažravý algoritmus

V predchádzajúcich častiach sme zhrnuli rôzne postupy, ako riešiť problém minimálnej dominujúcej množiny. V tejto časti uvedieme heuristiky pre pažravý algoritmus spomenutý vyššie. Prvou heuristikou bude výber vrchola, ktorý pokrýva vrchol stupňa najviac jeden. Túto heuristiku rozvinieme a pridáme k nej ďalšie. Potom ich skombinujeme do rôznych algoritmov.

2.8.1 Odstraňovanie výhonkov

Prvá heuristikou, ktorú uvedieme je jednoduchá. V pažravom algoritme, pred tým, ako začneme vyberať vrcholy (krok 2), tak vyberieme všetky vrcholy stupňa nula a vrcholy, ktoré susedia s vrcholmi stupňa jeden (to znamená ich jediných susedov). Tieto vrcholy budeme nazývať *výhonky*.

2.8.2 Odstraňovanie kvetín

Druhá heuristika spočíva v rozšírení prvej – odstraňovaní výhonkov. Vo všeobecnosti sa dá povedať, že sme sa snažili vybrať vrchol s čo najväčším počtom nepokrytých susedov takých, ktoré majú všetkých susedov spoločných s vybratým vrcholom. Teda tvoria akýsi lokálny klaster „na okraji“ grafu.

Vrchol v, ktorý pokrýva n vrcholov, ktoré majú za susedov iba susedov vrchola v, nazývame *kvetinou* rádu n.

2.8.3 Rozhodovanie rovnakých vrcholov

Keď máme viacero rovnakých kandidátov (z pohľadu) na výber, je otázne, či si vybrať náhodný vrchol, vrchol s najmenším/najväčším ID alebo pridať nejaké pravidlo, ktoré rozhodne o priorite výberu. Tu si uvedieme nejaké pravidlá.

V základnom pažravom algoritme na hľadanie sietí malého sveta je často situácia, kde majú rovnaký počet ešte nepokrytých vrcholov vrcholy s úplne iným stupňom. Možno ešte podstatnejšia hodnota ako počet ešte nepokrytých susedov je počet vrcholov, ktoré po vybratí daného vrchola prestanú pokrývať akékoľvek vrcholy. Toto nie je to isté ako hľadanie kvetiny s najväčším rádom, lebo kvetiny sa vyskytujú iba na okrajoch grafu. Vrcholy vybraté takýmto rozhodovaním sa nachádzajú všade a preferujú susedov vzdialených 3 od už vybratých susedov (respektíve vzdialených 2 od pokrytých vrcholov).

Treba si ale uvedomiť, že priemer siete malého sveta je malý a tak skutočná preferencia na „umiestnenie v grafe“ nie je. Taktiež tento spôsob vyberania môže trochu korigovať „slepé“ vyberanie vrcholov s väčšími stupňami.

2.8.4 Výsledné algoritmy

Vyššie uvedené heuristiky ide rôzne kombinovať a rôzne použiť. Na tomto mieste uvedieme algoritmus, ktorý skombinuje všetky pravidlá. Pre odstraňovanie výhonkov a odstraňovanie kvetín platí, že sa vykonajú iba raz – na začiatku algoritmu. Po odstránení sa spustí greedy algoritmus, ktorý vhodného kandidáta vyberie na základe pokrytia vrcholu a odstraňovania najvplyvnejších vrcholov.

Algoritmus vyzerá následovne:

1. odstráň výhonky:
   (a) vyber vrcholy do výsledku;
   (b) pokry susedov;

2. postupne odstraňuj kvety (postupne preto, lebo z niektorý kvetov prestanú byť kvety) – kvety zotrieď podľa rádu a stupňa; potom postupne, až kým nie je množina kvetov prázdna:
   (a) vyber kvet do výsledku;
   (b) pokry susedov;
   (c) odstráň tie kvety, ktoré už nie sú kvetmi;
3. prepočítaj hodnoty pre „vplyvnosť“ vrcholov;
4. opakuj, až kým množina pokrytých vrcholov netvorí celý graf:
   (a) do výsledku vyber najlepšieho kandidáta podľa „vplyvnosti“ a stupňa;
   (b) pokry susedov vybratého vrchola;
   (c) prerátaj „vplyvnosť“ susedov;

Ide o upravený pažravý algoritmus. Upravený o heuristiky a pozorovania uvedené vyššie. Tento algoritmus sa dá rôzne modifikovať upravovaním heuristík a pravidiel.

Ide o posledný uvedený algoritmus. V ďalších kapitolách sa venujeme implementačnej časti, rôznym testom a výsledkom.

Kapitola 3

Popis softvéru

V predchádzajúcich kapitolách sme spravili prehľad problematiky a algoritmov, ktoré slúžia na hľadanie minimálnej dominujúcej množiny. V tejto kapitole sa venujeme popisu a návrhu softvéru, ktorý slúži na porovnanie týchto algoritmov a otestovanie týchto algoritmov v praxi.

3.1 Analýza existujúcich riešení

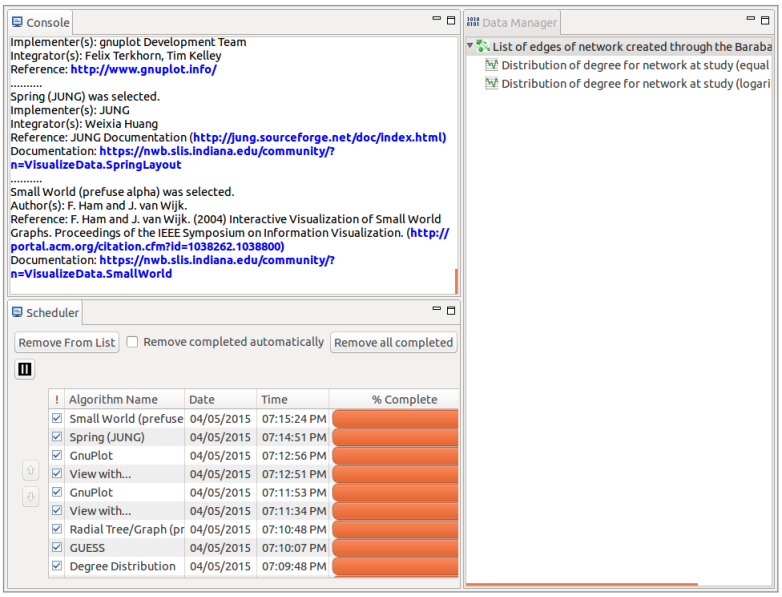
Keďže problém nájdenia minimálnej dominujúcej množiny je podobný problému vrcholového pokrytia je často súčasťou aplikácií venujúcim sa zhromažďovaniu rôznych grafových algoritmov. Veľa nám známych z nich obsahuje iba základy výpočet všetkých možností, alebo jednoduchý aproximačný algoritmus, preto tu uvedieme iba jedného zástupcu tohto druhu. Popri vyvíjaní nášho softvéru sme pracovali aj s nástrojmi pre analýzu grafov. Tieto sa taktiež zväčša vyskytujú v balíkoch s inými nástrojmi pracujúcimi s grafmi (napr. vizualizácia). Uvedieme tiež iba jedného zástupcu.

3.1.1 Network Benchmark

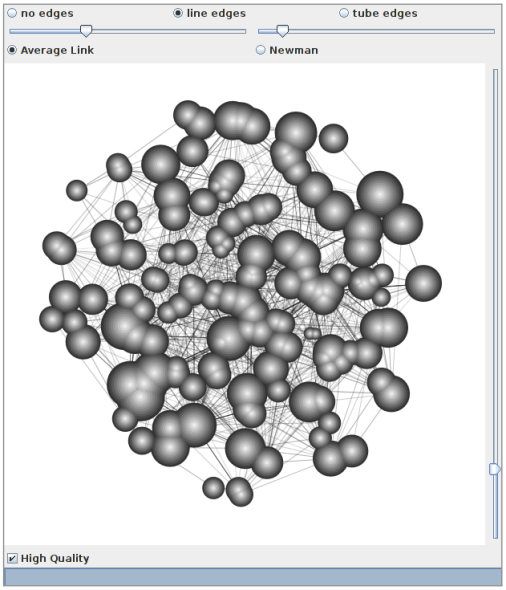
Reprezentantom softvérov na analýzu grafov je Network Benchmark. Ide o softvér, ktorý získal podporu aj Alberta Barabásiho. Bol vyvíjaný v rokoch 2005 až 2011. Obsahuje príklady, dokumentáciu a podklady k práci s analýzou grafu. Je naprogramovaný v jazyku `JAVA` a k jeho silným stránkam patrí relatívna prehľadnosť pri veľkom množstve nastaveniach a možnostiach analýzy.

Je dostupný na adrese: `http://nwb.cns.iu.edu/index.html`

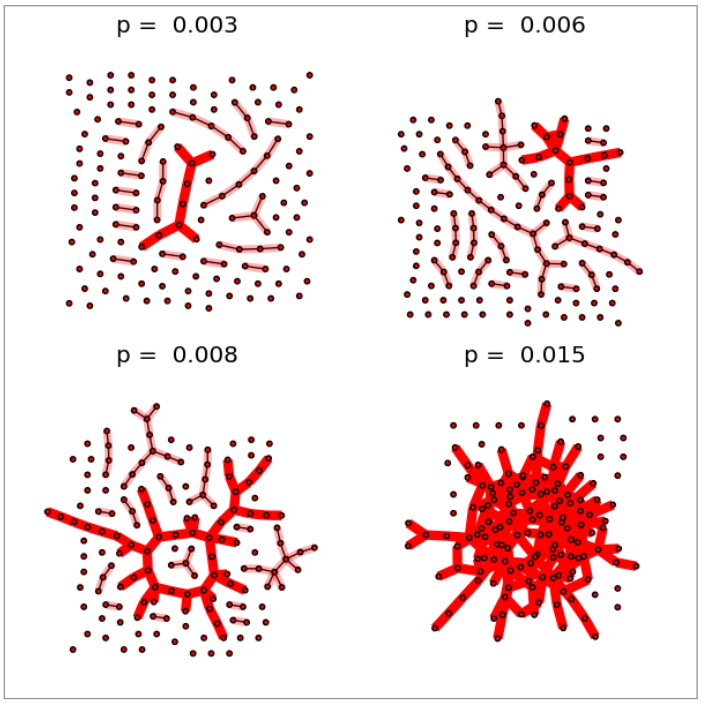
Na obrázku 3.1 je vidno hlavnú obrazovku aplikácie. Práca s aplikáciou prebieha tradične tak, že používateľ načíta nejakú grafovú štruktúru, následne sa mu zobrazí v stĺpci napravo, označí ju a z bohatej ponuky zvolí nejakú analýzu. Vľavo dole, potom vidí priebeh jeho akcií a vľavo hore bližšie popisy toho, čo sa práve vykonáva.



Obrázok 3.1: *Softvér Network Benchmark.* V ľavej hornej časti obrazovky sa nachádzajú výpisy. V dolnej prebiehajúce a ukončené akcie a výpočty. Vpravo sú zhromaždené výsledky výpočtov.



Obrázok 3.2: *Softvér Network Benchmark.* Tu je zobrazená vizualizácia siete malého sveta. Bublinky znázorňujú jednotlivé klastre.



Obrázok 3.3: *Softvér NetworkX*. Tu je zobrazená vizualizácia náhodného grafu s rôznymi pravdepodobnosťami hrán.

3.1.2 NetworkX

Tento softvér je reprezentantom zbierky rôznych algoritmov na grafoch. Je vyvíjaný od roku 2005 až doteraz. Posledná stabilná verzia v čase písania tejto práce je zo septembra roku 2014. Ide o rozšírenie jazyka Python o novú knižnicu.

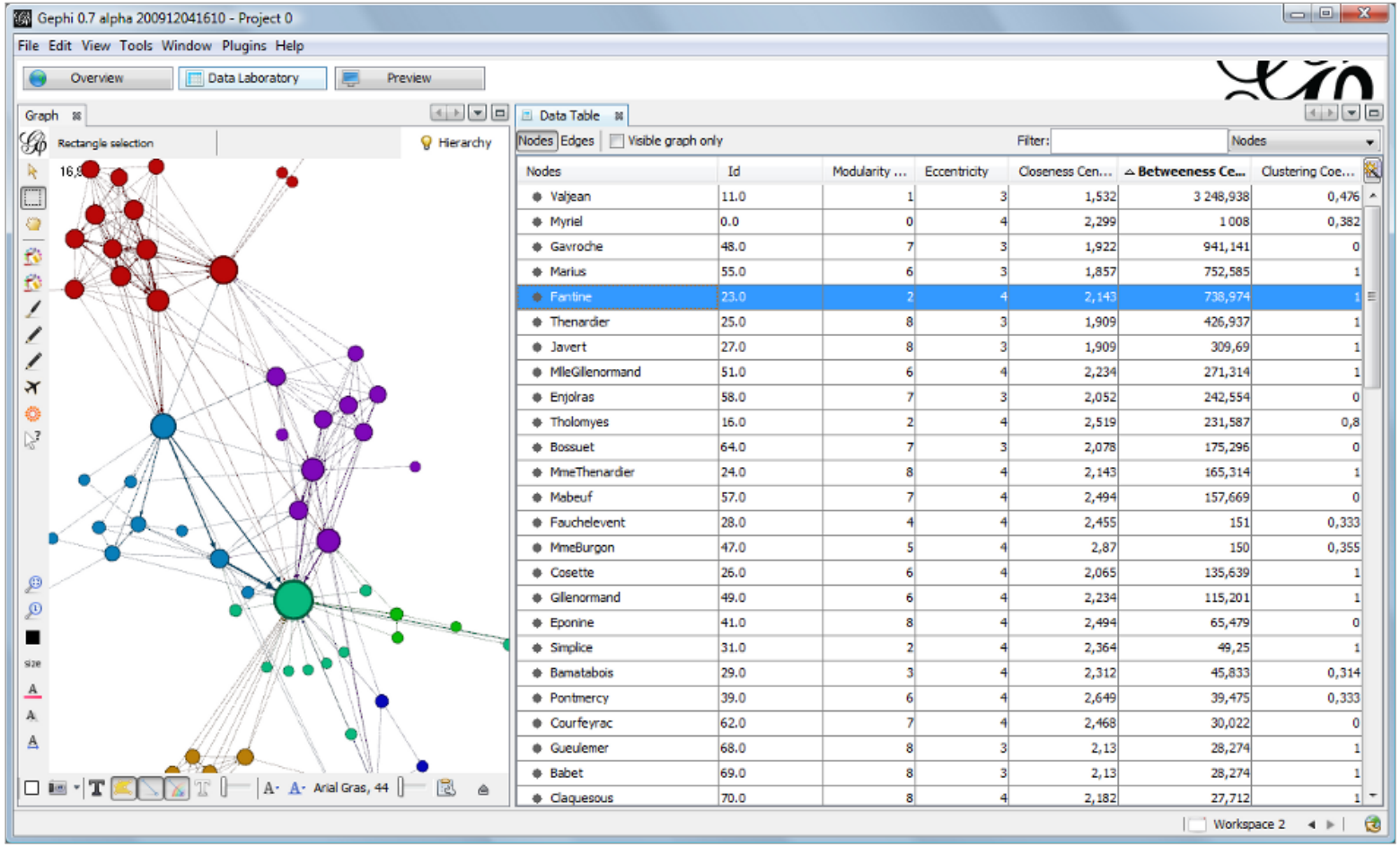
Sídlo projektu je na adrese: `https://networkx.github.io/`

Na obrázku 3.3 je znázornená vizualizácia so softvérom NetworkX. Ide o tvorbu náhodného grafu pomocou určenia pravdepodobnosti vzniku hrany. Vidno, že aj keď je NetworkX hlavne zbierkou algoritmov, poskytuje peknú vizualizáciu. Jeho nedostatkom pre nás je, že na riešenie problému hľadania minimálnej dominujúcej množiny poskytuje iba pažravý algoritmus.

3.1.3 Gephi

Gephi je platforma na vizualizáciu grafov. Zameriava sa na vizualizovanie grafov a počítanie rôznych hodnôt pre analýzu komplexných sietí. Podporuje veľa grafových formátov. Je vo vývoji od roku 2009 (Bastian, Heymann a Jacomy 2009). Zatiaľ posledná verzia vyšla v roku 2013. Platforma je dostupná na internetovej stránke `http://gephi.github.io/`. Na obrázku 3.4 je príklad vizualizácie grafu v programe Gephi.

Aj keď platforma je veľmi silným nástrojom na analýzu komplexných sietí, neposkytuje žiaden algoritmus na nájdenie minimálnej dominujúcej množiny.



Nodes	Id	Modularity ...	Eccentricity	Closeness Cen...	Betweeness Ce...	Clustering Coe...
Valjean	11.0	1	3	1,532	3 248,938	0,476
Myriel	0.0	0	4	2,299	1 008	0,382
Gavroche	48.0	7	3	1,922	941,141	0
Marius	55.0	6	3	1,857	752,585	1
Fantine	23.0	2	4	2,143	738,974	1
Thenardier	25.0	8	3	1,909	426,937	1
Javert	27.0	8	3	1,909	309,69	1
MlleGillenormand	51.0	6	4	2,234	271,314	1
Enjolras	58.0	7	3	2,052	242,554	0
Tholomyes	16.0	2	4	2,519	231,587	0,8
Bossuet	64.0	7	3	2,078	175,296	0
MmeThenardier	24.0	8	4	2,143	165,314	1
Mabeuf	57.0	7	4	2,494	157,669	0
Fauchelevent	28.0	4	4	2,455	151	0,333
MmeBurgon	47.0	5	4	2,87	150	0,355
Cosette	26.0	6	4	2,065	135,639	1
Gilenormand	49.0	6	4	2,234	115,201	1
Eponine	41.0	8	4	2,494	65,479	0
Simplice	31.0	2	4	2,364	49,25	1
Bamatabois	29.0	3	4	2,312	45,833	0,314
Pontmercy	39.0	6	4	2,649	39,475	0,333
Courfeyrac	62.0	7	4	2,468	30,022	0
Gueulemer	68.0	8	3	2,13	28,274	1
Babet	69.0	8	3	2,13	28,274	1
Claquesous	70.0	8	4	2,182	27,712	1

Obrázok 3.4: *Platforma Gephi.* Vľavo je vizualizácia grafu. Vpravo sú uvedené rôzne vlastnosti grafu.

3.1.4 JUNG

Projekt JUNG (Java Universal Network/Graph Framework) je grafová knižnica naprogramovaná v jazyku `JAVA`. Ponúka vlastný jazyk, ktorý poskytuje modelovanie, analýzu a vizualizáciu grafov a sietí.

3.2 Špecifikácia požiadaviek

Ako vidno, súčasné softvéry majú dobrú ponuku rôznych algoritmov a veľa odlišných vizualizácií. Avšak daň za to, že ponúkajú riešenie na veľa problémov je tá, že zväčša je dostupný iba jeden algoritmus na riešenie.

Keďže sa my v práci zaoberáme porovnávaním algoritmov, veľmi nám táto skutočnosť nesedí. Preto by sme mali navrhnúť vlastný softvér, ktorý zrejme nebude poskytovať riešenia na veľa problémov ale veľa riešení na jeden problém.

3.2.1 Požiadavky

Naša aplikácia slúži najmä na porovnávanie relatívnej rýchlosti behu algoritmov a bude používaná zrejme iba úzkym okruhom ľudí.

Preto sa nám zdalo byť vhodné aplikáciu nechať iba v príkazovom riadku. Znížia sa tým odchýlky spôsobené grafickým prostredím. Softvér by mal vedieť spracovať graf zadaný v špecifikovanom tvare. Taktiež by mal správne implementovať všetky algoritmy popísané v kapitole 2 a poskytnúť informácie o behu daného algoritmu.

Implementované algoritmy by mali zahŕňať:

- exaktný algoritmus skúšaním všetkých možností,
- exaktný algoritmus skúšaním všetkých možností s heuristikou,
- exaktný algoritmus riešený pomocou prevedenia na problém množinového pokrytia,
- distribuovaný algoritmus,
- upravený distribuovaný algoritmus,
- jednoduchý pažravý algoritmus,
- pažravý algoritmus s rôznymi heuristikami;

3.2.2 Prevádzkové požiadavky

Keďže ide o softvér, ktorý medzi sebou porovnáva jednotlivé algoritmy relatívne, tak nepotrebuje špecifický operačný systém. Výkon algoritmov je priamo úmerný od hardvéru. To al neznamená, že by hardvér mal byť nejako špecifický. Keďže sa súčasné architektúry z nášho pohľadu veľmi nelíšia, nekladieme po tejto stránke žiadne požiadavky.

3.3 Návrh

Našou hlavnou úlohou je naprogramovať program, ktorý vie spracovať nejaký formát vstupu, spraviť z neho graf, vykonať na ňom zvolený algoritmus. Takže bude obsahovať moduly na spracovanie týchto požiadaviek. V následujúcich častiach popisujeme požiadavky kladené na náš softvér.

3.3.1 Technické požiadavky

Softvér nevyžaduje žiaden špecifické technické veci. Vstupné a výstupné údaje môžu prúdiť cez štandardný systémový vstup a výstup. V prípade lepšej práce môžu byť použité knižnice na prácu s nejakými dátovými štruktúrami.

3.3.2 Používateľské požiadavky

Používateľmi tejto aplikácie sú najmä ľudia, ktorí chcú porovnať jednotlivé algoritmy. Teda stačí, aby bolo poskytnuté dobré vstupno-výstupné prostredie.

Ak chceme porovnávať medzi sebou rôzne algoritmy na nejakom grafe, alebo ak chceme porovnávať jeden algoritmus na rôznych grafoch je dobré mať aj prostredie pre dávkové úlohy.

3.3.3 Použité technológie

Keďže algoritmy porovnávame relatívne na sebe, tak môžeme zvoliť ľubovoľný programovací jazyk za hlavný. Keďže chceme, aby bola aplikácia ľahko spustiteľná na každom operačnom systéme, vhodným kandidátom je programovací jazyk `JAVA`.

`JAVA` poskytuje aj implementáciu základných dátových štruktúr. Avšak túto možno zameniť s nejakými inými externými knižnicami. Na implementáciu rôznych algoritmov na tom istom základe je vhodné použiť návrhový vzor *strategy*.

3.3.4 Rozhranie aplikácie

Ako sme už spomenuli, rozhranie aplikácie tvorí príkazový štandardný vstup a výstup. Na vstupe pre aplikáciu sú dva argumenty. Algoritmus použitý pri behu a cesta k súboru s grafom. Súbor s grafom je textový súbor obsahujúci riadky, v ktorých sú dvojice čísel, označujúcich neorientované hrany medzi vrcholmi. Na vytvorenie vnútornej reprezentácie grafu teda treba meno súboru, ktorý graf obsahuje. výstupom je objekt, s ktorým vie algoritmus pracovať. Algoritmus dostane na vstup tento objekt, vypočíta potrebné pomocné dátové štruktúry, spustí výpočet a vyráta výsledok. Ten potom vypíše na štandardný výstup. Čo je zároveň aj výstupom aplikácie.

Pre algoritmus výpočtu pomocou prevedenia sú potrebné pomocné dátové štruktúry, ale v zásade ide o jednoduchý princíp posielania si rôznych reprezentácií grafu objektami.

Pre dávkový proces je vstupom textový súbor, ktorý obsahuje riadky. V každom riadku je argument pre jeden beh aplikácie. Aplikácia sa spúšťajú za sebou, následujúca až vtedy, keď sa predošlá dokončí.

Kapitola 4

Implementácia softvéru

V tejto kapitole postupne uvedieme realizáciu návrhu popísaného v kapitole 3. Popíšeme ako sme implementovali jednotlivé algoritmy a ukážeme si štruktúru a ovládanie aplikácie.

4.1 Algoritmy

Aj keď sme porovnávali iba 10 algoritmov, dokopy bolo implementovaných algoritmov viac. V nasledujúcich častiach postupne uvedieme jednotlivé algoritmy a ich označenia pre aplikáciu. Algoritmy boli implementované podľa popisu v kapitole 2.

Pred tým, než rozpíšeme implementáciu algoritmov uvedieme štruktúru a implementáciu grafu. Graf je implementovaný ako objekt, ktorý obsahuje zoznam hrán. Pokiaľ je to potrebné (čo zväčša je), vie sa iniciovať a zavolať vypočítanie susedov do vzdialenosti 2 pre každý vrchol (aby sa nemuselo počítať pri každom použití ale iba raz, na začiatku). Taktiež objekt reprezentácie grafu vie skontrolovať, či je nejaká množina dominujúcou. To je spravené pomocou delegácie na algoritmus kontroly dominujúcej množiny.

Nasleduje popis jednotlivých skupín algoritmov s ich označeniami.

4.1.1 Algoritmy skúšajúce všetky možnosti

Prvým a základným algoritmom je algoritmus, ktorý skúša všetky možnosti. Jeho označenie v softvéri je `naive`. Je implementovaný rekurzívnym prehľadávaním. Obdobou sú algoritmy označené `mynaive` a `mynaive2`, ktoré obsahuje základné heuristiky. Prvou je, že algoritmus najprv zoradí pole vrcholov podľa stupňa a až potom začne vyberať do výsledku. Druhou je, že ak je medzivýsledok menšej mohutnosti, tak neskúša vyberať množiny s väčšou mohutnosťou.

4.1.2 Algoritmy prevedenia problému

Ďalšími algoritmami, ktoré sme implementovali, boli algoritmy, ktoré prevádzali hľadanie minimálnej dominujúcej množiny na problém množinového pokrytia. Keďže algoritmus bol reprezentovaný

návrhovým vzorom *strategy*, tak sa implementácia zjednodušila a sprehľadnila. Časť algoritmu sa venovala spracovaniu vstupu a časť výpočtu a implementovaniu náročnejších algoritmov.

Aj keď Fomin, Grandoni a Kratsch (2005) uvádzajú algoritmus jeden, my sme implementovali dve rôzne verzie: `fnaive` a `fproper`. Verzia `fnaive` neobsahuje finálne prevedenie na hľadanie minimálneho hranového pokrytia v grafe. Verzia `fproper` toto prevedenie obsahuje a implementuje ho cez hľadanie maximálneho párenia v grafe. Keďže v čase použitia algoritmu je graf bipartitný, tak sa to dá spraviť.

4.1.3 Distribuované algoritmy

Podobne ako predošlé algoritmy, aj tieto algoritmy mali viac verzií. Konkrétne jednovláknovú a viacvláknovú. Distribuovanosť bola implementovaná cez veľa na sebe nezávislých vlákien. Nedistribuované verzie sme označili príponou `-OT` (one thread).

Algoritmy sme implementovali tak ako v prehľade – dva. Prvý bez riešenia problémových grafov (obrázky 2.1 a 2.2), ale prehľadnejší a zrozumiteľnejší. Druhý s riešením týchto problémov. Prvý je označený ako `ch7alg34` (resp. `ch7alg34OT` pre jednovláknovú verziu) a druhý je označený ako `ch7alg35` (podobne `ch7alg35OT` pre jednovláknovú verziu).

4.1.4 Pažravé algoritmy

Keďže jedným z hlavných cieľov práce bolo spraviť dobrú heuristiku na pažravý algoritmus, tak boli tieto algoritmy implmentované vo viacerých verziách. Okrem rôznych heuristík ide aj o spôsob vyberania vrcholov. Počas behu algoritmu sú niektoré vrcholy vybraté do medzivýsledku, niektoré pokryté a niektoré nevybraté a nepokryté. Tie vrcholy, ktoré sú nevybraté a nepokryté sú označované aj ako *biele*, tie ktoré nie sú vybraté, ale sú pokryté sú označované ako *šedé* a tie, ktoré sú vybraté sú označované ako *čierne*.

Podľa algoritmu, ktorý je uvedený v časti 2.6 sme implementovali algoritmus, ktorý sme označili `greedy`. Podobne sme implementovali aj verziu `ch7alg33`. Avšak tá nevyberá vrcholy, ktoré pokrývajú čo najviac iných spomedzi šedých a bielych, ale iba z bielych vrcholov. Veľmi podobne je implementovaná aj verzia `greedyq`. Jediný rozdiel je, že vrcholy na začiatku behu usporiada podľa stupňa od najväčšieho po najmenší.

Heuristiky na rozhodovanie

Po troch podobných verziách sme implementovali rôzne heuristiky. Prvou bolo odstraňovanie výhonkov. Toto odstraňovanie prebieha na začiatku algoritmu, postupne v iteráciách, až kým nie je čo odstrániť. Motiváciou za týmto algoritmom je efektívne odstraňovanie ciest. Verzia algoritmu s touto heuristikou je označovaná ako `greedysr`.

V ďalšej verzii sme použili opäť základný pažravý algoritmus, ale ako rozhodovací faktor pre výber vrchola sme vybrali počet vrcholov, ktoré po vybratí už nebudú mať bieleho suseda (toto nie je to isté

ako rád kveta) a až potom počet vrcholov, ktorým vrchol dominuje. Motiváciou bolo presvedčenie, že často je lepšie vybrať vrchol, ktorý po svojom vybratí čo najviac zmenší počet potenciálne vybratých vrcholov v ďalších iteráciách, namiesto vyberania vrcholov s najväčším pokrytím. Heuristika by mala byť nejakým kompromisom medzi vyberaním vrcholov „zospodu“ a „zvrchu“. Táto verzia algoritmu je označená ako `greedysw`.

Zložitejšia heuristika

Treťou verziou je `floweru`, ktorá je podobná ako `greedysr`, s tým rozdielom, že na začiatku spraví iba jeden beh odstraňovania výhonkov. Toto je základom pre posledný, zložitejší algoritmus, ktorý je označený `flower`.

Verzia `flower` na začiatku behu zoradí vrcholy podľa stupňa. Následne sa snaží označiť výhonky a kvety. Vyčistí výhonky a upraví graf tak, aby sa mohli vybrať kvety. Kvety sa vyberajú postupne. Najviac záleží na ráde kvetu. Ak majú nejaké kvety rovnaký rád, tak sa rozhoduje podľa toho, koľko vrcholov po vybratí už nebude mať bieleho suseda (podobne ako pri verzii `greedysw`). Ak je aj táto hodnota rovnaká, tak zaváži počet bielych susedov (najväčšie pokrytie). Ak nepomôže ani toto pravidlo, tak sa vyberie náhodný vrchol.

Po označení výhonkov a kvetov sa začnú označovať zvyšné vrcholy podľa jednoduchého pažravého algoritmu. Avšak ako rozhodujúci faktor pri vyberaní slúži najprv počet bielych susedov (pokrytie) a potom počet vrcholov, ktoré po vybratí už nebudú mať bieleho suseda.

4.1.5 Dátové štruktúry a externé knižnice

Aj keď na porovnanie rôznych algoritmov sú základné dátové štruktúry poskytované ako súčasť knižníc jazyka `JAVA`, rozhodli sme sa použiť na implementáciu základných dátových štruktúr knižnicu HPPC od poskytovateľa Carrot Search Labs vo verzii 0.6.0. Je dostupná na webovej adrese `http://labs.carrotsearch.com/hppc.html` a licencovaná Apache License 2.0. hlavnou výhodou knižnice oproti štandardnej knižnici je menšia spotreba pamäte a priamejší prístup k vnútornej reprezentácií (aspoň v danej verzii). To spôsobilo celkové zrýchlenie algoritmov a posunulo hranice testovateľných grafov.

Aj keď medzi analyzovanými existujúcimi riešeniami je veľa grafových knižníc, pripadalo nám jednoduchšie a efektívnejšie implementovať vlastnú reprezentáciu grafov pomocou základných prvkov jazyka `JAVA`a knižnice HPPC.

Kapitola 5

Dosiahnuté výsledky

V predošlých kapitolách sme spravili prehľad existujúcich algoritmov, popísali sme ich implementáciu. V tejto kapitole uvádzame porovnanie jednotlivých implementácií. Najprv uvedieme jednotlivé obmedzenia implementácie, potom popíšeme spôsob testovania a napokon označenie pre jednotlivé testovacie dáta.

5.1 Obmedzenia implementácie

Vývoj softvéru sa začal na staršom počítači, ktorý má procesor Intel Core 2 Duo P7350 (2,0GHz dvojjadrový) a 4GB RAM pamäte. Pamäť nebola obmedzením. Bohužiaľ počítač nebol vhodný na viacvláknové algoritmy, čo spôsobovalo problémy pre algoritmy `ch7alg34` a `ch7alg35`. Preto sme sa zamerali na ich jednovláknové verzie (`ch7alg34OT`, respektíve `ch7alg35OT`). Heuristiky na pažravý algoritmus sa doimplementovali neskôr, preto sa neuvádzajú v starších testoch.

Počas vývoja začal byť dostupný novší počítač, na ktorom sa testovali už všetky heuristiky pre pažravý algoritmus. Počítač má procesor Intel Core i5-4690K CPU (3,50GHz štvorjadrový) a 16GB pamäte. Tento počítač zvláda viacvláknové aplikácie lepšie (o 6 rokov modernejšia architektúra), no vzhľadom na zachovanie kompaktnosti sme opäť viacvláknové verzie algoritmov z testov vynechali.

Neskoršia práca s externými knižnicami HPPC časy zrýchlila, čo bohužiaľ nie je zachované v žiadnej z uvedených tabuliek.

5.2 `JAVA` verzus `C++`

Zo začiatku implementácie nebolo jasné, ktorý programovací jazyk bude použitý. Ako prvý sa zvolil jazyk `JAVA`. Keďže prevláda povedomie, že interpretovaný programovací jazyk (`JAVA`) nemôže byť rýchlejší ako kopmilovaný programovací jazyk (`C++`), tak sme sa v priebehu implementácie rozhodli, že skúsime naprogramovať niektoré algoritmy aj v jazyku `C++`. Ako vidno z porovnania časov behov jednotlivých algoritmov v tabuľkách 5.1 a 5.3, jazyk `C++` neposkytuje žiadne výrazné zrýchlenie. Spolu s faktom, že na porovnanie jednotlivých algoritmov až tak rýchlosť netreba, sme sa rozhodli, že budeme

pokračovať vo vývoji v jazyku `JAVA`.

5.3 Spôsob testovania

Keďže programovací jazyk `JAVA` robí optimalizácie kódu počas behu a triedy inicializuje „lenivo“, tak je potrebné algoritmus spustiť viackrát, aby ukázal relevantné výsledky. Algoritmus teda necháme spustiť trikrát a zoberieme tretí výsledok. Rozhodnutie je vecou pozorovania. Je kompromisom medzi relevantnými výsledkami a dĺžkou skúšania rôznych behov. Z časov nad 20 sekúnd je v tabuľkách uvedený iba čas prvého behu algoritmu.

Rôzne algoritmy sme testovali na rôznych dátach. Najmä podľa toho, aké veľké dáta vedeli spracovať v rozumnom čase. Testovacie dáta uvádzame v ďalšej časti.

5.4 Testovacie dáta

Testovacie dáta boli grafy rôznych veľkostí a štruktúr. Naše testovacie dáta sa dajú rozdeliť do štyroch hlavných skupín. Podľa skupiny majú aj predponu. Sú to tieto:

- grafy tvorené prioritným pripájaním tvoriace Barabási-Albertov model – majú predponu `ba-`;
- grafy tvorené hranami s náhodnou pravdepodobnosťou vzniku tvoriace náhodný model – majú predponu `rnd-`;
- grafy vytvorené špeciálne podľa obrázka 2.1 – majú predponu `zle-`;
- grafy vytvorené z reálnych dát – majú predponu `ca-`.

Grafy tvorené prioritným pripájaním boli vytvorené pomocou programu Network Workbench. Ku grafu sa pridali v každom kroku dve hrany. Grafy tvorené náhodnou pravdepodobnosťou vzniku hrán boli taktiež pomocou programu Network Workbench. Grafy boli vytvorené tak, aby mali približne rovnakú hustotu ako grafy Barabási-Albertoveho modelu rovnakej veľkosti. Reálne dáta boli prevzaté zo stránky `http://snap.stanford.edu/data/`.

Podľa počtu vrcholov majú názvy jednotlivých grafov príponu. Pre grafy vytvorené prioritným pripájaním a náhodné grafy, číslo za predponou vyjadruje počet vrcholov grafu. Pre grafy vytvorené podľa obrázka 2.1 číslo vyjadruje počet vrcholov na „hlavnej“ ceste. Pre grafy reálnych dát boli čísla zvolené umelo a nemajú žiaden význam. Všetky použité reálne dáta boli siete citácií vedeckých publikácií.

5.5 Porovnanie algoritmov

V tejto časti porovnáme medzi sebou jednotlivé skupiny algoritmov. Začneme algoritmami dávajúcimi presné výsledky, budeme pokračovať všeobecným porovnaním a na koniec zhodnotíme jednotlivé heuristiky pažravého algoritmu.

5.5.1 Presné algoritmy

Do tejto kategórie spadajú algoritmy `naive`, `fnaive` a `fproper`. Boli implementované najmä kvôli prehľadu a získaniu veľkostí minimálnych dominujúcich množín. Ako vidno zo všetkých tabuliek (5.3, 5.1 a aj 5.5), pri exaktných algoritmoch nemôžeme rátať s tým, že sa dočkáme výsledkov na reálnych dátach.

Môžeme však vidieť, že použitím vhodných úprav vieme posunúť veľkosť grafu, kedy vieme vypočítať minimálnu dominujúcu množinu v rozumnom čase, zo zhruba 20 vrcholov na zhruba 100 vrcholov. Taktiež je dobré všimnúť si, že aj keď algoritmus `fproper` používa oveľa zložitejšie výpočty, tak čas behu algoritmu to oveľa nezrýchli. Prevedenie na hľadanie maximálneho párenia (respektíve minimálneho hranového pokrytia) je teda viac zlepšením teoretickej zložitosti ako pomoc reálnemu behu algoritmu.

Celkom zaujímavým je pozorovanie, že na neštruktúrovaných dátach (`rnd-` grafy) mali algoritmy `fnaive` a `fproper` oveľa horší čas ako na dátach so štruktúrou (`ba-` a `zle-` grafy).

5.5.2 Všeobecné porovnanie

Skôr ako sa pozrieme na celkové hodnotenie, tak sa môžeme pozrieť na algoritmy `ch7alg340T` a `ch7alg350T`. V tabuľke 5.3 je vidno, žo algoritmus `ch7alg350T` naozaj rieši uvedené problémy v časti 2.7. Škoda, že sa nepodarilo vytvoriť prostredie pre distribuovaný algoritmus.

Celkovo si môžeme všimnúť (tabuľka 5.5), že algoritmy bežia rýchlejšie na štruktúrovaných dátach (`ba-` a `zle-`) okrem algoritmu `ch7alg330T`, ktorý beží lepšie na neštruktúrovaných dátach (`rnd-`).

Keďže štruktúrované aj neštruktúrované dáta majú pomerne rovnakú hustotu hrán, tak sme neporovnávali algoritmy na rôzne hustých grafoch.

5.5.3 Porovnanie hauristík pažravých algoritmov

Zatiaľ sme sa pozerali čisto iba na časy. No pri porovnávaní reálnych časov pažravých algoritmov nám o rýchlosti veľa prezradí aj veľkosť výslednej množiny. Veľkosti výslednej množiny sú uvedené v tabuľke 5.4. Za príklad si môžeme zobrať trebárs základný algoritmus `greedy`. Tam, kde mal menšiu výslednú množinu, tam mal aj nižší čas. Napríklad, pre grafy `ba20k` a `rnd20k` bol rozdiel vo veľkosti nájdených množín 1347. To znamená, že sa základný cyklus musel zopakovať oveľa viac krát, čo vyústilo do horšieho času algoritmu. Keďže všetky verzie pažravého algoritmu až na verziu `flower` mali na začiatku behu algoritmu (pred opakujúcim sa cyklom) takmer nulové predpočítavanie rôznych hodnôt, respektíve vyberanie vrcholov do výsledku, Tak je tento trend badať na všetkých algoritmoch.

Algoritmus `flower` je v tomto smere výnimkou. Keďže si predpočítava susednosť do vzdialenosti tri a grafy tvorené Barabási-Albertovým modelom sú viac spojité ako náhodné grafy, tak sa počet opakovaní následného cyklu prejaví až pri grafoch s veľkým počtom vrcholov (v našom prípade `-20k`).

Keďže v Barabási-Albertovom modely sa každý nový vrchol spojí s práve (v našom prípade) dvoma inými, tak nevznikajú výhonky. Taktiež nie je šanca ani na vytvorenie kvetov. Takže veľa heuristík

založených na odstraňovaní výhonkov a kvetov nič neodstráni a teda výsledkom je taká istá množina ako pri základnom pažravom algoritme.

5.5.4 Porovnanie pažravých algoritmov na reálnych dátach

V reálnych dátach je situácia iná ako v modelových. V reálnych dátach sa tvoria malé lokálne klastre, ktoré sú občas hustejšie prepojené a vytvoria kvet. Taktiež je šanca, že nejaký vrchol bude izolovaný alebo takmer izolovaný. Teda má zmysel vytvárať heuristiky na základe vyberania výhonkov a kvetov. Keďže reálne siete mali nad 5000 vrcholov, iné ako pažravé algoritmy nebolo možné testovať. Výsledky testovania sú uvedené v tabuľke 5.2.

Algoritmy `ch7alg33` a `greedyq` boli navrhnuté s tým, že budú bežať rýchlejšie, ale aj oveľa nepresnejšie, čo sa potvrdilo. Bohužiaľ miera nepresnosti na reálnych dátach je príliš veľká na akékoľvek použitie.

Najjednoduchší pažravý algoritmus (`greedy`) dáva prekvapivo veľmi dobré výsledky. Je prekonaný iba algoritmom `greedysw`, ktorý je rozšírením jednoduchého pažravého algoritmu iba o vhodný spôsob výberu vrchola. Najzložitejší, algoritmus `flower`, síce našiel najmenšiu dominújúcu množinu spomedzi testovaných algoritmov na dvoch grafoch (`ca1` a `ca22`), ale mal horšie časy.

Grafy `ca22` a `ca3` sú hustejšie ako ostatné, čo sa prejavilo aj na pomalom behu algoritmu `greedysw`.

5.6 Tabuľky

V tejto časti sa nachádzajú tabuľky zobrazujúce výsledky z testovania algoritmov.

	naive	greedy	ch7alg33	ch7alg34OT
ba10	0	0	0	0
ba18	1	0	0	0
ba20	4,37	0	0	0
ba100		0	0	0
ba200		0	0,01	0,06
ba1000		0	0	1,28
ba2000		0,01	0,02	5,15
ba10k		0,08	0,51	120,32
ba20k		0,29	2,68	
ba100k		6,52	147,56	
rnd100		0	0	0,03
rnd200		0	0	0,04
rnd1000		0	0	0,21
rnd10k		0,07	0,53	5,6
zly10		0	0	0
zly20		0	0	0,01
zly100		0,02	0,22	4,72

Tabuľka 5.1: Výsledky behov algoritmov v jazyku C++ na staršom počítači. Výsledky sú uvedené v sekundách.

	ca1 - |S|	ca1 - čas	ca2 - |S|	ca2 - čas	ca22 - |S|	ca22 - čas	ca3 - |S|	ca3 - čas
greedy	1176	0,124	2009	0,578	2275	0,330	3694	1,311
ch7alg33	1570	0,070	3243	0,121	3230	0,099	5651	0,392
greedyQ	1586	0,034	3239	0,087	3243	0,080	5625	0,319
greedysr	1803	0,072	3100	0,340	3667	0,118	5553	0,708
greedysw	1159	0,348	1989	1,613	2258	1,022	3649	3,548
floweru	1329	0,111	2131	0,511	2501	0,313	3920	1,196
flower	1157	0,293	2018	2,406	2256	0,591	3774	2,830

Tabuľka 5.2: *Testovanie rôznych heuristík na dátach z reálneho sveta.* V stĺpci |S| je veľkosť nájdenej množiny. Čas je uvedený v sekundách.

	naive	greedy	greedyQ	ch7alg33	ch7alg34OT	ch7alg35OT	fnaive	fproper
ba10	0,085	0,001	0,002	0	0,003	0,004	0,008	0,009
ba18	1,284	0,001	0,002	0	0,013	0,008	0,035	0,035
ba20	4,440	0,001	0,002	0	0,011	0,008	0,049	0,047
ba100		0,015	0,004	0,004	0,070	0,045	0,473	0,506
ba200		0,026	0,006	0,009	0,135	0,073	30,515	30,412
ba1000		0,087	0,045	0,037	0,883	0,512		
ba2000		0,142	0,095	0,053	2,524	0,988		
ba10k		1,844	0,362	0,540	45,781	6,662		
ba20k		8,813	1,294	3,800		22,256		
ba100k		66,762	68,669	135,197				
rnd10		0,001	0,002				0,003	0,003
rnd15		0,001	0,002				0,048	0,043
rnd20		0,001	0,002				0,107	0,112
rnd100		0,011	0,004				1293,811	1302,940
rnd200		0,032	0,008					
rnd1000		0,072	0,043					
rnd2000		0,114	0,096					
rnd10k		2,484	0,440					
rnd20k		11,281	1,671	4,947				
zly10		0,005	0,003	0,003	0,040	0,018	0,119	0,117
zly20		0,022	0,010	0,022	0,098	0,036	0,809	0,814
zly100		0,090	0,151	0,210	6,213	2,107		

Tabuľka 5.3: Výsledky behov algoritmov v jazyku `JAVA` na staršom počítači. Výsledky sú uvedené v sekundách.

	naive	greedy	greedyQ	ch7alg33	greedysr	greedysw	floweru	flower	ch7alg34OT	ch7alg35OT	fnaive	fproper
ba10	2	2	2	2	2	2	2	2	6	6	2	2
ba18	4	4	6	6	5	4	4	4	9	11	4	4
ba20	4	4	7	5	6	4	4	5	10	11	4	4
ba100		17	33	31	17	17	17	18	54	61	15	15
ba200		32	59	56	31	35	31	31	102	113	29	29
ba1000		142	284	290	147	151	147	146	515	558		
ba2000		276	570	577	273	284	273	274	1005	1091		
ba10k		1393	2901	2974	1391	1465	1391	1390	5094	5593		
ba20k		2828	5749	5896	2828	2942	2828	2817		11093		
ba100k		13947	29378	29375	13947	14656	13947	13928				
rnd10	2	2	2	2	2	2	2	2	7		2	2
rnd15	3	4	3	4	3	3	3	3	10		3	3
rnd20	4	4	5	5	4	5	4	4	15		4	4
rnd100		21	28	24	22	22	19	19	62		18	18
rnd200		41	52	53	40	39	40	39	126			
rnd1000		201	284	281	202	210	200	199	633			
rnd2000		411	541	552	414	411	404	399	1272			
rnd10k		2083	2775	2760	2122	2098	2035	2024	6325			
rnd20k		4175	5560	5542	4279	4193	4091	4053				
zly10		10	35	35	10	10	10	10	10	10	10	10
zly20		20	120	120	20	20	20	20	20	20	20	20
zly100		100	2600	2600	100	100	100	100	100	100		

Tabuľka 5.4: Veľkosti nájdených dominujúcich množín pre daný graf a algoritmus.

	naive	greedy	greedyQ	ch7alg33	greedysr	greedysw	floweru	flower	ch7alg34OT	fnaive	fproper
ba10	0,010	0	0	0	0	0	0	0,001	0,001	0,001	0,001
ba18	0,331	0	0	0	0	0	0	0,001	0,001	0,001	0,001
ba20	1,321	0	0	0	0	0	0	0,001	0,001	0,001	0,001
ba100		0,003	0	0,002	0,001	0,003	0,001	0,004	0,016	0,084	0,145
ba200		0,004	0	0,003	0,001	0,006	0,001	0,007	0,043	15,595	15,577
ba1000		0,021	0,003	0,013	0,011	0,040	0,011	0,046	0,358		
ba2000		0,047	0,008	0,025	0,024	0,740	0,025	0,090	1,322		
ba10k		0,308	0,079	0,115	0,251	0,933	0,249	0,952	21,37		
ba20k		0,987	0,290	0,410	0,837	3,097	0,873	3,368			
ba100k		19,602	6,831	19,201	21,315	65,587	20,743	60,146			
rnd10	0,011	0	0	0	0	0	0	0	0	0	0
rnd15	0,058	0	0	0	0	0	0	0	0	0	0
rnd20	1,445	0	0	0	0	0	0	0	0	0,001	0,001
rnd100		0,003	0	0	0	0	0	0,001	0,002	634,429	631,217
rnd200		0,004	0	0,001	0	0,001	0	0,002	0,008		
rnd1000		0,019	0,001	0,007	0,004	0,015	0,004	0,016	0,105		
rnd2000		0,045	0,004	0,020	0,013	0,048	0,013	0,048	0,303		
rnd10k		0,384	0,101	0,120	0,294	0,984	0,313	0,929	2,827		
rnd20k		1,337	0,390	0,478	1,224	3,833	1,260	3,703			
zly10		0	0	0	0	0	0	0	0	0	0
zly20		0	0	0	0	0	0	0,001	0,001	0,009	0,006
zly100		0,027	0,016	0,037	0,003	0,100	0,011	0,088	0,662		

Tabuľka 5.5: Výsledky behov algoritmov v jazyku `JAVA` na novšom počítači. Výsledky sú uvedené v sekundách.

Záver

V práci sme sa snažili naplniť ciele zadania. A to spraviť prehľad algoritmov hľdajúcich minimálne dominujúce množiny a skúsiť vylepšiť niektoré algoritmy, ktoré hľadajú minimálne dominujúce množiny na sieťach malého sveta.

V kapitole 1 sme popísali základné pojmy a uviedli čitateľa do problematiky grafov, komplexných sietí a hľadania minimálnej dominujúcej množiny. Kapitolu 2 sme začali spísaním prehľadu súčasných základných existujúcich algoritmov na hľadanie minimálnej dominujúcej množiny. Ďalej sme v tejto kapitole načrtli výhody v prípade, že vstupným grafom je komplexná sieť. Nakoniec sme navrhli niekoľko heuristík, o ktorých sme predpokladali, že nejako zlepšia čas behu alebo výsledky algoritmov. Keďže sme sa v práci chceli zaoberať dátami z reálneho sveta, zvolili sme si heuristiky na zlepšenie tých algoritmov, ktoré majú šancu spracovať také veľké dáta.

V kapitolách 3 a 4 sme popísali návrh a implementáciu softvéru. Preskúmali sme existujúce riešenia a na základe analýzy navrhli, ako by mala implementačná časť práce vyzerať. Softvér nakonie nenadväzoval na žiadne predošlé práce, aj keď používal pomocnú knižnicu (HPPC) na uchovanie základných dátových štruktúr. V konečnom dôsledku malo použitie externej knižnice pozitívny dopad. Implementačnú časť práce sme sa rozhodli spraviť samostatne a nenadväzovať kvôli úzkej špecifikácií algoritmov a pohodlnosti práce s vlastnými zložitejšími dátovými a programovými štruktúrami oproti existujúcim riešeniam.

V kapitole 5 sme opísali výsledky z implementácie softvéru a porovnali sme algoritmy medzi sebou a na rôznych dátach, včetne reálnych sietí citácií. Z navrhovaných heuristík mala každá svoje slabé a silné miesta. Čo dáva priestor na ďalšie zlepšovania.

V ďalšej činnosti je vhodné zamerať sa podrobnejšie na to, aké heuristiky dávajú lepšie výsledky pre rôzne typy dát ako napríklad hustejšie/redšie grafy, grafy s väčšou/menšou klasterizáciou alebo trebárs rôzne spôsoby vyberania výhonkov. To by umožňovalo vzniku možno lepšej heuristiky (či už z pohľadu času alebo veľkosti nájdenej množiny) alebo kombinovaného algoritmu s využitím viacerých heuristík (podobne ako naša verzia `flower`).

V tejto práci sme nevyužili vedomosť, že siete malého sveta sú síce odolné voči náhodným útokom, no zraniteľné cielenými útokmi v tom zmysle, že ich vieme odobratím malého počtu hrán rozdeliť na viacero komponentov. Je možné, že keby sme vedeli, tieto hrany efektívne určiť, mohli by sme skúmať, ako dôležité sú vrcholy, ktoré sú incidentné s hranou a na základe toho ich prioritne ne/vybrať do výslednej množiny.

Vďaka tomu, že vieme, že siete malého sveta sú riedke grafy, môžeme tieto grafy rozdeliť na mno-

žinu stromov, na ktorých sa minimálna dominujúca množina hľadá ľahšie. Je však otázne, ako vyriešiť vrcholy, v ktorých tieto stromy v pôvodnom grafe susedia. Aj keď sa tento prístup zdá byť prijateľný, tak vieme, že siete malého sveta majú veľa lokálnych klastrov a pridelení týchto klastrov by vznikalo veľa stromov, čo by mohlo viesť ku vybratiu veľa „zbytočných“ vrcholov. Vďaka tejto komplikácií a tomu, že sa tento spôsob riešenia výrazne odlišuje od zvyšných, sme takýto postup hľadania minimálnej dominujúcej množiny nakoniec vynechali.

Veľmi potešujúce však je, že sme našli heuristiku, ktorá na všetkých testovaných reálnych dátach prekonala veľkosťou dominujúcej množiny jednoduchý pažravý algoritmus (verzia `greedysw`). Iba o trochu viac sofistikovaný spôsob vyberania (uprednostňuj vrcholy, ktoré svojím vybratím čo najviac zmenšia množinu vrcholov, ktoré majú nejakého bieleho suseda) dosiahol zlepšenie. Zlepšenie sa nepodarilo vždy pri algoritme so zložitejšou štruktúrou – verzii `flower`. Tento výsledok odráža skutočnosť, že často algoritmy s jednoduchou štruktúrou bývajú tým najšťastnejším riešením.

Literatúra

Bachmann, Paul Gustav Heinrich (1894). *Analytische Zahlentheorie*. Leipzig, Germany.

Bastian, Mathieu, Sebastien Heymann a Mathieu Jacomy (2009). *Gephi: An Open Source Software for Exploring and Manipulating Networks*. URL: `http://www.aaai.org/ocs/index.php/ICWSM/09/paper/view/154`.

Diestel, Reinhard (2000). *Graph theory. 2000*.

Fomin, Fedor V, Fabrizio Grandoni a Dieter Kratsch (2005). "Measure and conquer: domination – a case study". In: *Automata, Languages and Programming*. Springer, str. 191–203.

Grandoni, Fabrizio (2004). "Exact algorithms for hard graph problems". Diz. práca. Taliansko: Università di Roma "Tor Vergata".

Kuhn, Fabian (2011). "Network algorithm".

Plesník, Ján (1983). *Grafové algoritmy*.