Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívne hľadanie minimálne dominujúcej množiny na reálnych sieťach

Magisterská práca

2014 **Viktor Tomkovič**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Efektívne hľadanie minimálne dominujúcej množiny na reálnych sieťach

Magisterská práca

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

Bratislava, 2014 **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

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 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 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.

1.9 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

TODO: Počet hrán je rádovo rovnaký ako počet vrcholov v sieti malého sveta

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 $N/2$.

TODO: cut, prevedenie a tak

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.

TODO: Uviesť zlé typy grafov a vylepšený algoritmus

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).

Kapitola 3

Popis softvéru

V predchádzajúcich kapitolách sme si popísali niektoré dátové štruktúry a vybrané algoritmy. Hlavnou náplňou je ich testovanie.

3.1 Analýza existujúcich riešení

V marci roku 2015 o žiadnych podobných riešeniach nevieme.

3.2 Špecifikácia požiadaviek

TODO: Čo očakávame od softvéru a prečo je viac fajn ako konkurencia.

3.2.1 Požiadavky

TODO: Čo má softvér robiť z hladiska funkčnosti.

3.2.2 Prevádzkové požiadavky

TODO: Čo má softvér robiť z hladiska hardvéru.

3.3 Návrh

TODO: Stručný opis, čo chceme dosiahnuť.

3.3.1 Technické požiadavky

TODO: Self-descripting

3.3.2 Používateľské požiadavky

TODO: Self-descripting

3.3.3 Použité technológie a návrhové vzory atď.

TODO: Self-descripting

3.3.4 Rozhranie aplikácie

TODO: Vstup/výstup – aj pre jednotlivé časti softvéru

Kapitola 4

Implementácia softvéru

V tejto kapitole postupne uvedieme realizáciu návrhu popísaného v kapitole 1. Popíšeme ako sme implementovali jednotlivé algoritmy a ukážeme si štruktúru a ovládanie aplikácie. Na záver uvedieme ako dopadlo testovanie softvéru ().

Kapitola 5

Dosiahnuté výsledky

TODO: Tu budú všetky tie faily, cez ktoré som si prešiel.

	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	4.44	0.001	0.002	0	0.011	0.008	0.049	0.047
ba100	-	0.015	0.004	0.004	0.07	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	0.54	45.781	6.662		
ba20k		8.813	1.294	3.8		22.256		
ba100k		66762	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	1302.94
rnd200		0.032	0.008					
rnd1000		0.072	0.043					
rnd2000		0.114	0.096					
rnd10k		2.484	0.44					
rnd20k		11.281	1.671	4.947				
zly10		0.005	0.003	0.003	0.04	0.018	0.119	0.117
zly20		0.022	0.01	0.022	0.098	0.036	0.809	0.814
zly100		0.09	0.151	0.21	6.213	2.107		

	naive	greedy	greedyQ	ch7alg33	ch7alg34OT	ch7alg35OT	fnaive	fproper	macov
ba10	0.085	0.001	0.002	0	0.003	0.004	0.008	0.009	1
ba18	1.284	0.001	0.002	0	0.013	0.008	0.035	0.035	1
ba20	4.44	0.001	0.002	0	0.011	0.008	0.049	0.047	1
ba100	-	0.015	0.004	0.004	0.07	0.045	0.473	0.506	1
ba200		0.026	0.006	0.009	0.135	0.073	30.515	30.412	1
ba1000		0.087	0.045	0.037	0.883	0.512			2
ba2000		0.142	0.095	0.053	2.524	0.988			
ba10k		1.844	0.362	0.54	45.781	6.662			
ba20k		8.813	1.294	3.8		22.256			
ba100k		66762	68.669	135.197					1
rnd10		0.001	0.002				0.003	0.003	1
rnd15		0.001	0.002				0.048	0.043	2
rnd20		0.001	0.002				0.107	0.112	3
rnd100		0.011	0.004				1293.811	1302.94	3
rnd200		0.032	0.008						5
rnd1000		0.072	0.043						
rnd2000		0.114	0.096						5
rnd10k		2.484	0.44						5
rnd20k		11.281	1.671	4.947					5
zly10		0.005	0.003	0.003	0.04	0.018	0.119	0.117	5
zly20		0.022	0.01	0.022	0.098	0.036	0.809	0.814	5
zly100		0.09	0.151	0.21	6.213	2.107			5

Záver

V práci sme popísali niektoré algoritmy a porovnali ich implementácie. Fomin (Fomin, Grandoni a Kratsch 2005) napísal dobrú prácu... ...

Literatúra

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

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*.