Katedra Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Algoritmy pre vizualizáciu zložených grafov

(Bakalárska práca)

Autor: Miroslav Baláž

Školiteľ: Mgr. Jana Katreniaková

Bratislava, 2007

Čestne prehlasujem, že som bakalársku prácu vypracoval samostatne s použitím uvedenej literatúry. _____

Ďakujem svojej školitelke, Mgr. Jane Katreniakovej, za výber vhodnej témy, poskytnuté materiály a užitočné pripomienky.

Práca obsahuje popis a implementáciu hierarchizačného algoritmu pre kreslenie zložených digrafov, ktoré obsahujú aj inkluzívne aj adjecenčné hrany. Estetickosť grafu formulujeme ako optimalizačné problémy, pre ktoré nájdeme heuristické metódy. Ďalej uvedieme výsledky o aproximovateľnosti a **NP**-úplnosti týchto problémov.

Kľúčové slová: Zložené grafy, Vizualizácia, Feedback Arc Set

This paper contains implementation of hierarchical algorith for drawing of compound digraphs, that contain both inclusion edges and adjecency edges. We identify readability of drawing as optimalizaton problems, and develop heuristic algorithm that solves them. Further we consider results about **NP**-hardness and approximabilty of those problems.

Keywords: Compound digraphs, Graph drawing, Feedback arc set.

Obsah

Kapitola 1

Úvod

Automatické kreslenie grafov je veľmi ťažká úloha. Ak chceme nejakú úlohu riešiť pomocou počítača, musíme ju najprv vedieť formálne popísať. To si však vyžaduje takmer dokonalú znalosť danej tématiky. Kreslením grafov sa zaoberali okrem iných aj Sugiama a Misue. V tejto práci sa venujeme ich algoritmu, ktorý popísali v [10], navyše doplníme výsledky o aproximovatelnosti a ťažkosti problémov, ktoré tento algoritmus používa ako procedúry. Doteraz sa týmto algoritmom zaoberal Raitner, ktorý ale implementoval svoju vlastnú modifikáciu a zameral sa hlavne na navigáciu vo vykreslených zložených grafoch.

1.1 Motivácia

V dnešnej informačnej spoločnosti je dôležité nielen informácie zhromažďovať, ale dôležité je aj vedieť ich efektívne prezentovať. Automatizovaním prezentácie sa šetria finančné prostriedky, vďaka čomu možno za rovnaké náklady využiť viac zdrojov informácií. V tomto duchu vznikla aj potreba využiť počítače na automatické vizualizovanie grafov.

Grafom môžeme reprezentovať napríklad UML diagram, alebo obsah E-learningového systému. Teda ich použitie má význam aj v komerčnej, aj v akademickej sfére. V komerčných projektoch sú programátori postavení pred úlohu vytvoriť popis softvérového riešenia, ktorý obsahuje veľké množstvo UML diagramov. V prípade spolupoužitia reverzného inžinierstva a automa-

tického kreslenia grafov by bolo možné takýto popis vytvoriť za minimálne náklady zo zdrojového kódu riešenia.

Z teoretického hľadiska sme však postavený pred prakticky neriešiteľnú úlohu, za predpokladu $\mathbf{P} \neq \mathbf{NP}$. Preto sa musíme uspokojiť aj s neoptimálnym riešením.

Algoritmus, ktorý popisujeme v tejto práci, je vhodný na kreslenie tak zvaných zložených grafov s hierarchickou štruktúrou. Takúto štruktúru majú ako UML diagramy tak aj obsah E-learningových systémov, preto je pre ich vizualizáciu takýto algoritmus vhodný.

1.2 Členenie práce

Na úvod v druhej kapitole definujeme základné pojmy z teórie grafov a pojmy z oblasti zložených grafov. Tiež povieme pár slov o vizualizácii informacií a grafov a o možnostiach, ktoré sa využívajú. V tretej kapitole popíšeme metódu hierarchizačného kreslenia grafov a jej uplatnenie pri kreslení zložených grafov, postupujeme podľa algoritmu Sugiyama a Misue [10]. V štvrtej rozoberieme konkrétne detaily implementácie tohoto algoritmu. V piatej kapitole zhrnieme súčasné poznatky o aproximovatelnosti a **NP**-úplnosti problémov, ktoré potrebujeme riešiť. V závere zhrnieme, čo sme urobili a navrhujeme ďalšie možné postupy a využitie implementácie.

Kapitola 2

Základné pojmy

2.1 Grafy

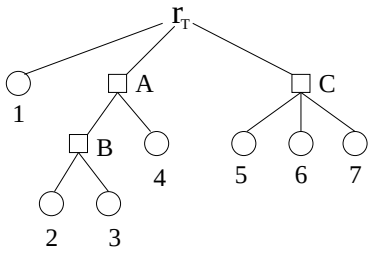
V tejto časti uvedieme definície pojmov, ktoré sa v práci používajú, ide o základné pojmy z teórie grafov. Pojmy z teórie množín, ktoré používame, sú dobre popísané v [8].

Definícia. Digraf je usporiadaná dvojica (V, E), pričom prvky množiny V voláme vrcholy, prvky množiny E voláme hrany, navyše musí platiť $V \cap E = \emptyset$, $E \subseteq V \times V$. Ak $(a, b) \in E$, hovoríme, že z vrcholu a vychádza hrana do vrcholu b, alebo, do vrcholu b vchádza hrana z vrcholu a. Ak G je digraf, potom V(G) označuje množinu jeho vrcholov a E(G) množinu jeho hrán.

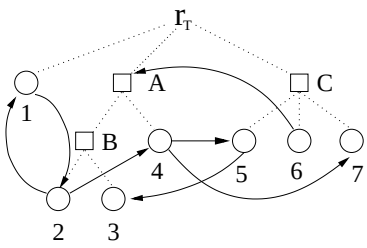
Definícia. Hovoríme, že postupnosť $p = (v_0 v_1 \dots v_{n-1} v_n)$ je $v_0 v_n$cesta v digrafe $G = (V, E)$, ak $(\forall_{0 \leq i \leq n}) v_i \in V$ a $(\forall_{0 \leq i \leq n-1})(v_i v_{i+1}) \in E$. Ďalej hovríme, že cesta p začína vo vrchole v_0, prechádza vrcholmi $v_0 v_1, \dots, v_{n-1} v_n$, končí vo vrchole v_n, a má dĺžku n.

Definícia. Hovoríme, že digraf $G = (V, E)$ je zakorenený strom, ak existuje taký vrchol $x \in V$), že do každého iného vrcholu z V existuje práve jedna cesta, ktorá začína v x. Vrchol x nazývame koreňom stromu G.

Definícia. Ak G je zakorenený strom, s koreňom x, tak $dep(y)$ znamená dĺžku cesty, ktorá začína v x a končí v y. Hovoríme aj, že hĺbka vrcholu y je $dep(y)$.



Obr. 2.1: Inkluzívny podgraf



Obr. 2.2: Adjecenčný podgraf

Nech G je digraf a $v \in \mathrm{V}(G)$, potom:

1. $Desc(v) = \{y \mid$ existuje cesta, ktorá začína vo v a končíi v $y\}$, je množina všetkých potomkov v.

2. $Asc(v) = \{y \mid$ existuje cesta, ktorá začína v y a končí vo $v\}$, je množina všetkých predkov v.

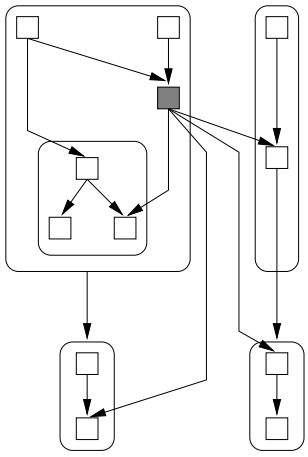
3. $Child(v) = \{y \mid$ existuje hrana $(v, y) \in \mathrm{E}(G)\}$, je množina všetkých detí v.

4. $Pa(v) = y$ také že existuje hrana $(y, v) \in \mathrm{E}(G)$, je rodič v.

2.2 Zložené digrafy

Klasické digrafy sa používajú na modelovanie vzťahov medzi objektami, avšak zachytávajú iba to, či nejaké dva objekty sú v nejakej relácii, alebo nie. Digraf môžeme nakresliť tak, že každý objekt reprezentujeme kružnicou, a tie pospájame podľa toho, či dané objekty sú v relacii, dostaneme takto spleť čiar. Zložené digrafy pridávajú možnosť reprezentácie inkluzívnej relácie, skutočnosť, že nejaký objekt sa skladá z ďalších, nebudeme reprezentovať čiarami, ale intuitívnejšou geometrickou inklúziou (vo veľkom obdĺžniku, predstavujúcom objekt, budú malé obdĺžniky, ktoré predstavujú objekty, z ktorých sa skladá).

Definícia. Zložený digraf je trojica (V, E, F) taká, že (V, E) je zakorenený strom a (V, F) je digraf, navyše požadujeme, že ak u je potomok v, tak $(u, v) \notin F$ a $(v, u) \notin F$.



Obr. 2.3: Príklad zloženého grafu

Definícia. Ak (V, E, F) je zložený digraf, potom hovoríme, že (V, E) je jeho inkluzívny podgraf.

Definícia. Ak (V, E, F) je zložený digraf, potom hovoríme, že (V, F) je jeho adjecenčný podgraf.

Definícia. Koreň zloženého digrafu (V, E, F) je ten vrchol, ktorý je koreňom jeho inkluzívneho podgrafu.

Inkluzívnu reláciu teda reprezentuje inkluzívny podgraf a adjecenčnú adjecenčný podgraf.

2.3 Vizualizácia

Pod vizualizáciou zloženého digrafu rozumieme spôsob zviditeľnenia tak, aby mohol byť vnímaný ľudskými očami. V našom prípade ide o problém nájsť pre každý vrchol jeho umiestnenie na ploche a velkosť, a pre každú hranu nájsť trasu kadiaľ povedie čiara, ktorá ju reprezentuje.

Základné pravidlá, ktoré musí nakreslený graf spĺňať, sa nazývajú *konvencie*. Najčastejšie sa špecifikujú požiadavky na tvar hrán (rektilineárne čiary, úsečky, lomené čiary, krivky), tvar vrcholov (kruh, obdĺžnik) umiestnenie vrcholov (napr. v mrežových bodoch) a všeobecnú štruktúru nákresu (napr.

planarita). K týmto požiadavkám sa ďalej pridávajú ešte *estetické kritéria*, ktoré by sa mali plniť optimálne. Môže ísť napríklad o celkovú plochu grafu, dĺžku najdlhšej hrany (distribúciu dĺžok), distribúciu uhlov medzi nicidentnými hranmi alebo iné kritérium s konečnou popisnou zložitosťou.

Na kreslenie grafov bolo vyvinutých viacero prístupov. O niektorých vieme povedať, že na niektorých typoch grafov sú lepšie ako niektoré iné. Túto kapitolu uzavrieme príkladom rôznych prístupov, ktoré boli doteraz použité.

1. *Spektrálna metóda* využíva na určovanie pozícií vrcholov vlastné vektory Laplacovej derivácie adjecenčnej matice grafu.

2. *Fyzikálne založené metódy* gradientnou metódou minimalizujú energetickú funkciu, ktorá je založená na poznatkoch z molekulárnej mechaniky.

3. *Hierarchizačné metódy* využívajú podobný prístup ako metóda, ktorú využívame v tejto práci. Vrcholom priradia horizontálne vrstvy a na jednotlivých vrstvách ich preusporiadajú tak, aby minimalizovali počet pretínaní hrán. Sú vhodné na kreslenie acyklických grafov.

4. *Augmentačná metóda* v prvom kroku pridaním vrcholov do miesta, kde by sa hrany pretínali, upraví graf na planárny. V druhom kroku doplní graf na trianguláciu, tú potom už vieme vnoriť do roviny tak, aby sa hrany zobrazili na úsečky. Nakoniec vrcholy pridané v prvom kroku odstráni.

5. *Ortogonálne (rektilineárne) metódy* kreslia grafy tak, že hrany sú horizontálne alebo vertikálne (v niektorých prípadoch lomené) čiary, snažia sa minimalizovať napríklad celkovú plochu a počet pretínaní hrán. Sú vhodné pre VLSI dizajn.

6. *Metódy na kreslenie stromov* využívajú jednoduchú štruktúru stromov, ale na druhú stranu sa zaoberajú podrobnejšími vlastnosťami nákresu. Väčšinou ide o metódy typu *rozdeľuj a panuj*.

Kapitola 3

Popis algoritmu

V tejto kapitole najskôr opíšeme vlastnosti algoritmu, ktorý používame. Pod tým máme na mysli, ako bude graf vyzerať po nakreslení. Ďalej opíšeme detailnejšie ako funguje. Algoritmus je snahou o implementovanie algoritmu z [10], a ide o takzvaný algoritmus s hierarchizačným prístupom.

3.1 Základné vlastnosti

Pri návrhu algoritmu bolo najdôležitejšie, aby bolo výsledné nakreslenie zloženého digrafu čo najčitateľnejšie. Jednou dôležitou myšlienkou je, že ľudský mozog ľahšie prijme graf, ktorého štruktúra je pravidelná a rovnomerná. Toto je dosiahnuté vďaka použitiu hierarchizačného prístupu, ktorý také vlastnosti dosahuje. Ďalšie kritéria, ktoré boli zohľadňované, sú počty pretínaní objektov a ich vzdialenosť. Nakoniec boli prijaté nasledovné konvencie:

1. Každý vrchol je znázornený ako obdĺžnik so stranami rovnobežnými so súradnicovými osami.

2. Inkluzívna relácia je znázornená geometrickou inklúziou medzi obdĺžnikmi. Ak a je potomkom b, potom a je celý umiestnený do b. Naopak, ak a nie je potomkom b, a ani b nie je potomkom a. Tak sa obdĺžniky prislúchajúce k a a b nesmú prekrývať.

3. Obdĺžniky sú umiestnené do rovnobežných horizontálnych pásov, ktoré sú nazvané úrovne. Na rozdelenie má vplyv aj inkluzívna aj adjecenčná

relácia. V tejto vlastnosti sa najviac prejavuje hierarchizácia.

4. Adjecenčné hrany sú kreslené ako šípky, ktoré vedú smerom z hora nadol.

Poslednú konvenciu nie je možné vždy dodržať vzhľadom na to, že digraf môže obsahovať cykly. Preto niektoré hrany musíme obrátiť. Zvyšné konvencie sú dodržané vždy. Ďalej sú uvedené pravidlá, ktoré sa metóda snaží dodržať tak dobre, ako to len ide. Sú to:

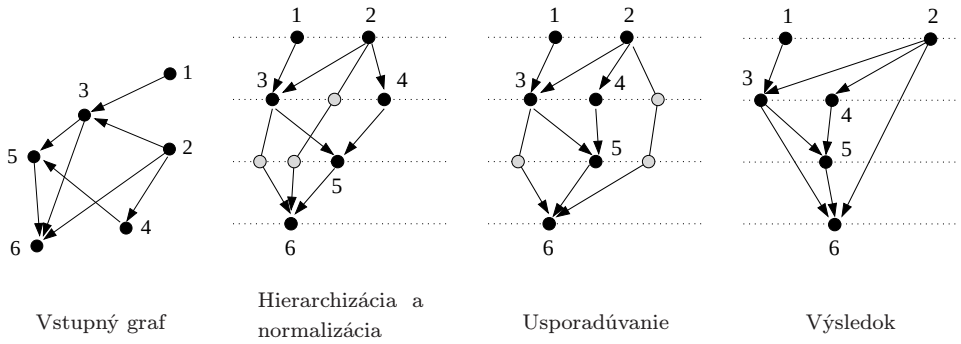
1. Vrcholy, ktoré sú spojené adjecenčnou hranou, by mali byť blízko seba.
2. Čiary, ktoré reprezentujú hrany, by sa mali navzájom čo najmenej pretínať.
3. Čiary, ktoré reprezentujú hrany, by sa mali pretínať s čo najmenším počtom obdĺžnikov.
4. Čiary, ktoré reprezentujú hrany, by mali byť čo najpriamejšie, s malým počtom ohybov a s čo možno najdlhšou vertikálnou časťou.

Keďže minimalizovať počet pretínaní hrán s hranami a obdĺžnikmi predstavuje **NP**-ťazký optimalizačný problém, je nutné použiť aproximatívne alebo heuristické algoritmy.

3.2 Popis algoritmu

Algoritmus pracuje v štyroch krokoch, ktoré sú nazvané nasledovne: *hierarchizácia*, *normalizácia*, *usporiadanie vrcholov*, *umiestnenie vrcholov*. V *hierarchizácii* sa vrcholom priradia úrovne a vyriešia sa problémy spôsobené prítomnosťou cyklov. Vznikne tak *zložený digraf s priradenými úrovňami*. V *normalizácii* sa do *zloženého digrafu s priradenými úrovňami* pridajú pomocné vrcholy a hrany tak, aby bol v predpísanom normálnom tvare. V *usporiadavacom* kroku sa vrcholom určí poradie vrámci úrovne, od neho bude závisieť x-ová súradnica umiestnenia. Dostaneme tak *usporiadaný zložený digraf*. V poslednom, *umiestňovacom* kroku, sa z *usporiadaného zloženého digrafu* vypočítajú pozície a veľkosti vrcholov a trasy hrán.

Vo zvyšku kapitoly sú detailne popísane všetky štyri kroky.



Obr. 3.5: Príklad priebehu algoritmu

3.3 Hierarchizácia (1.krok)

V tomto kroku sa vrcholom priradia úrovne. Vrchol je priradený do úrovne, v závislosti na adjecenčej a inkluzívnej relácii. Jednotlivé úrovne nebudeme označovat číslami, ale postupnosťami čísel. Poradie úrovní bude potom definované na základe lexikografického usporiadania postupností. Skôr, ako ukážeme ako prideľovanie funguje, potrebujeme zadefinovať niekoľko pojmov a označení.

Nech Σ označuje množinu všetkých prirodzených čísel $\{0, 1, 2, \ldots\}$.

Definícia. Σ^n označuje množinu všetkých postupností prirodzených čísel dĺžky n.

V texte budeme postupnosti značiť gréckymi písmenami, napr. α, β.

Definícia. $tail(\alpha)$ označuje posledný člen postupnosti α.

Definícia. $append(\alpha, x)$ označuje postupnosť, ktorá vznikne z α pridaním x na koniec.

Definícia. Ak α je postupnosť, potom α_i označuje i-ty člen postupnosti.

Definícia. Hovoríme, že postupnosť α je lexikograficky menšia ako β, ak existuje i také, že pre všetky nezáporné j menšie ako i, $\alpha_j = \beta_j$ a $\alpha_i < \beta_i$, túto skutočnosť značíme ako $\alpha < \beta$, alebo $\alpha \leq \beta$ pokiaľ ide o neostrú nerovnosť.

Vo zvyšku práce budeme úroveň, ktorá je priradená vrcholu v označovať ako $clev(v)$ a budeme pracovať s jediným zloženym digrafom $G = (V, E, F)$.

Pre priradenie úrovní požadujeme určité vlastnosti. Nasledujúce vlastnosti závysia od inkluzívnej relácie

$$(\forall v \in V) clev(v) \in \Sigma^{dep(v)} \tag{3.1}$$

$$(\forall v \in V)(\exists s \in \Sigma) clev(v) = append(clev(Pa(v), s) \tag{3.2}$$

Ďalšie vlastnosti závisia aj od adjecenčnej, aj od inkluzívnej relácie. Definujú vzájomné usporiadanie medzi priradenými úrovnami. Pre každú dvojicu vrcholov $(v, w) \in F$ existuje jediný vrchol t, tzv. *najhlbší spoločný predok* taký, že v inkluzívnom podgrafe existujú cesty:

$$P_1 = (t, p_1, p_2, \ldots, p_{m-1}, p_m = v)$$

$$P_2 = (t, q_1, q_2, \ldots, q_{n-1}, q_n = w)$$

Nech $k = min(n, m)$, potom požadujeme, aby

$$clev(p_i) \leq clev(q_i), i < k$$

$$clev(p_k) < clev(q_k)$$

Takéto priradenie nemusí vždy existovať, tento problém ale vyriešime. Algoritmus, ktorý priraďuje vrcholom úrovne, najskôr vytvorí odvodený graf $GD = (V, E, FD, typ)$, $FD \subseteq V \times V$, typ: $FD \to \{<, \leq\}$, ktorý vznikne tak, že v grafe G nahradíme každú adjecenčnú hranu hranami, ktoré reprezentujú usporiadanie úrovní, aké si hrana vynucuje. Tieto hrany budú dvoch typov, jedny pre ostré nerovnosti a druhé pre neostré nerovnosti, to, o aký typ ide, je dané funkciou typ. Ak odvodený graf obsahuje cykly, treba niektoré hrany odstrániť.

Samotné priradenie funguje tak, že sa zoberie vrchol v, ktorý má už priradenú úroveň a priradí sa úroveň všetkým vrcholom z $Ch(v)$. Vieme, že ak $w \in Ch(v)$, potom musí $clev(w) = append(clve(v), s)$. Stačí teda len vhodne zvoliť posledný člen postupnosti $clev(w)$. Najjednoduchšie je za $tail(clev(w))$ dosadiť dĺžku najdlhšej cesty, ktorá končí vo w, a prechádza vrcholmi z $\{x \mid x \in Ch(y) \wedge clev(y) = clev(Pa(v))\}$, dôležité je, že toto

celé sa deje v odvodenom grafe. Nakoniec otočíme všetky hrany (u, v) tak, aby $clev(v) < clev(u)$. Celý algoritmus môžeme zhrnúť v krátkom pseudo programe:

```
procedure Hierarchizácia
vstup:zlozený graf G=(V,E,F)
výstup:priradenie úrovní clev
  vytvor odvodený graf GD:=(V,E,FD,typ)
  vyrieš problemý s cyklami GD -> GD'
  compLevAssign({root},GD')
end
procedure compLevAssign(W,G)
  M := levAssign(W,G)
  for i:=1 to M do
    Z := {x | Pa(x) je z W a tail(clev(x)) = i}
    CompLevAssign(Z)
end
procedure levAssign
  rozloz vrcholy W na vrstvy L(i), i=1...n,
    podla najdlhsej cesty ktora v nich  konci
  for (each y frm L(1) do
    lev(y):=1
    clev(y):= append(clev(Pa(y)),lev(y))
  for i:=2 to n do for (each y from L(i) do
    lev(y):=max{clev(y)+(1 ak je hrana (u,v) ostra) |
      (v,y) je hrana v G, v je z L(j), j<i }
    clev(y):=append(clve(Pa(y)),lev(y))
  return max{lev(y),y je z W}
end
```

3.4 Normalizácia (2.krok)

Normalizácia v podstate slúži na to, aby sa hrany, ktoré vedú cez viacej urovní, mohli ľahko nakresliť ako lomené čiary. Pretože, v pomocných vrcholoch, ktoré pridáme pri normalizácií, môžu byť body ohybu tejto čiary. Vystupom normalizačného kroku je *normalizovaný zložený digraf*.

Definícia. Hovoríme, že hrana (v, w) je dobrá, ak

$$\begin{aligned} clev(v) &= (\alpha, s) \\ clev(w) &= (\alpha, s+1) \end{aligned}$$

alebo inými slovami

$$\begin{aligned} clev(Pa(v)) &= clev(Pa(w)) \\ tail(clew(v)) + 1 &= tail(clev(w)) \end{aligned}$$

V normalizačnom korku, sa všetky hrany, ktoré nie sú dobré, nahradia zloženým podgrafom, ktorý má lineárnu štruktúru. Pričom každá novo pridaná hrana už bude dobrá. Pod označčením (β) máme na mysli vrchol s úrovňou β.

Nech (v, w) je hrana, ktorá nie je dobrá,

$$clev(v) = (\alpha, s_1, s_2, \ldots, s_{m-1}, s_m)$$

$$clev(w) = (\alpha, t_1, t_2, \ldots, t_{n-1}, t_n)$$

α je teda časť postupnosti, ktorú majú $clev(v)$ a $clev(w)$ spoločnú, a pre jednoduchosť označenia ju budeme vynechávať. Pridaný zložený podgraf sa skladá z troch častí.

Najskôr je časť, ktorá začína vrcholom $(s_1, s_2, \ldots, s_{m-1}, s_m + 1)$ (do ktorého ide hrana z v, ako je vidno určite je dobrá) a končí vrcholom (s_1). Potom nasleduje postupnosť vrcholov $(s_1 + 1), (s_1 + 2), \ldots, (t_n - 1)$, a nakoniec časť, ktorá začína (t_1) a končí vrcholom $(t_1, t_2, \ldots, t_{n-1}, t_n - 1)$, z ktorého ide hrana do w. V okrajových prípadoch môže byť tvar tohoto zloženého podgrafu trochu deformovaný, napríklad keď n=0, alebo m=0. Nech

$$M_k = \max\{tail(clev(x)) \mid clev(Pa(x)) = clev(Pa^k(v))\}$$

M_k je najväčšie číslo, ktorým končí úroveň nejakého vrcholu, ktorá má prvých $m - k$ členov rovnakých ako v. V prvej časti sa potrebujeme dostať z $(s_1, s_2, \ldots s_{m-1}, s_m)$ do (s_1). Preto začneme s cestou

$$(s_1, s_2, \ldots s_{m-1}, s_m + 1), (s_1, s_2, \ldots s_{m-1}, s_m + 2), \ldots,$$
$$(s_1, s_2, \ldots s_{m-1}, M_1 - 1), (s_1, s_2, \ldots s_{m-1}, M1)$$

ktorú celu zavesíme pod vrchol $(s_1, s_2, \ldots s_{m-1})$, na ktorý nadväzuje cesta

$$(s_1, s_2, \ldots, s_{m-2}, s_{m-1} + 1), (s_1, s_2, \ldots s_{m-2}, s_{m-1} + 2), \ldots,$$
$$(s_1, s_2, \ldots s_{m-2}, M_2 - 1), (s_1, s_2, \ldots s_{m-2}, M_2)$$

ktorú celú zavesíme pod vrchol $(s_1, s_2, \ldots s_{m-2})$, a tak ďalej. Tretia časť je konštruovaná analogicky, s rozdielom, že cesty začínajú v $(\beta, 1)$ a končia v (β, t_i). Stredná časť je veľmi jednoduchá,a je zrejmé z popisu, ako vyzerá.

3.5 Usporiadanie vrcholov (3.krok)

V tejto časti popíšeme najšpecifickejšiu časť algoritmu, ide o spôsob, akým algoritmus dosahuje čo najlepšie rozmiestnenie vrcholov v rámci jednej úrovne.

Definícia. Nech $G = (V, E, F, clev)$ je *zložený digraf s priradenými úrovňami*, potom i-tou vrstvou vrcholu v, označujeme množinu $V_i(v)$

$$V_i(v) = \{x \mid v \in Ch(v) \wedge tail(clev(v)) = i\}$$

a počet vrstiev označujeme $L(v)$

$$L(v) = max_{x \in Ch(v)}(tail(clev(x)))$$

Definícia. Nech $G = (V, E, F, clev)$ je normalizovaný zložený digraf, $n = |V|$ a $V = v_1, v_2, \ldots, v_{n-1}, v_n$. *Usporiadaným zloženým digrafom* nazývame päticu $(V, E, F, clev, \sigma)$, kde

$$\sigma = (\sigma(v_1), \sigma(v_2), \ldots, \sigma(v_{n-1}), \sigma(v_n))$$

$$\sigma(v_j) = (\sigma_1(v_j), \sigma_2(v_j), \ldots, \sigma_{L(v_j)-1}(v_j), \sigma_{L(v_j)}(v_j))$$

a $\sigma_i(v_j)$ je permutácia $V_i(v_j)$.

Vstupom je v tomto prípade *normalizovaný zložený digraf* a výstupom je *usporiadný zložený digraf*. Algoritmus, ktorý sme vyvinuli pracuje lokálne, postupne spracúva všetky množiny $Ch(v_j)$, pričom každú spracúva samostatne. Z tejto črty algoritmu je vidno, prečo bol *usporiadný zložený digraf* zadefinovaný takým spôsobom, $\sigma(v_i)$ nezávisí na inom $\sigma(v_j)$, a $\sigma(v_i)$ je ešte rozdelená na vrstvy, lebo vrcholy patriace do inej vrstvy sú aj na inej úrovni, a preto sa navzájom nepremiešavajú.

Definícia. Nech $G = (V, E, F, clev, \sigma)$ je usporiadaný zložený digraf, potom definujeme lokálnu hierarchiu $H(\sigma(v))$ ako

$$H(\sigma(v)) = (Ch(v), F^d, F^s, \sigma(v), \lambda, \rho)$$

kde F^d je množina hrán, ktoré sú medzi vrchlomi na rôznych vrstvách, F^s je množina hrán, ktoré idú medzi potomkami dvoch vrcholov z rovnakej úrovne. $\lambda(w)$ je počet hrán, ktoré vychádzajú z w, a vchadzájú do vrchola y, ktorý nie je z $Ch(v)$ a zároveň je naľavo od v, ρ je počet hrán, ktoré idú doprava.

Lokálna hierarchia je v podstate orientovaný acyklický graf, ktorý získame tak, že v adjecenčnom podgrfe indukovanom $Desc(v)$ kontrahujeme (viď. [3]) všetky vrcholy z $Desc(w)$, kde $w \in Ch(v)$, do jedného vrcholu w. Vo výslednom grafe sú potom dva druhy hrán tie, čo idú medzi susednými vrstvami a tie, čo idú medzi dvoma vrcholmi z rovnakej úrovne. Vrchol u, ktorý je synom v,alebo potomok u môže byť adjecentný v pôvodnom zloženom digrafe s vrcholom w, ktorý nie je potomkom v. Avšak, vďaka tomu, že graf je normalizovaný, musia sa $Pa(w)$ a u nachádzať na tej istej úrovni. Ak budeme pri usporadúvaní spracovávať vrcholy v poradí podľa hĺbky, budeme už vedieť povedať, či je w napravo, alebo na lavo od u. Môžeme tak vypočítať hodnoty $\lambda(u)$ a $\rho(u)$ pre všetky $u \in Ch(v)$.

Po skonštruovaní lokálnej hierarchie pre vrchol v už len stačí určiť permutáciu vrcholov v každej vrstve tak, aby sme čo najlepšie splnili kritéria, ktoré sme si zadali na začiatku, a teda minimalizovať počet pretínaní hrán, minimalizovať počet pretinaní hrán s obdĺžnikmi a nakoniec, aby vrcholy, ktoré sú spojené hranou, boli blízko seba. Posledné kritérium nazývame aj *Blízkosť*. Najprv tieto kritériá sformulujeme ako optimalizačné problémy, a potom opíšeme heuristické riešenie, ktoré používame na ich riešenie.

Pre jednoduchosť značenia uvažujeme lokálnu hierarchiu H

$$H = (Ch(v), F^d, F^s, \sigma(v), \lambda, \rho)$$

ktorá má len dve vrstvy, teda

$$\sigma = (\sigma_1, \sigma_2), \sigma_1 = (u_1, u_2, \ldots, u_{p-1}, u_p), \sigma_2 = (w_1, w_2, \ldots, w_{q-1}, w_q)$$

1. *Blízkosť:*

$$P1 = \sum_{1 \le k \le p} (k\lambda(u_k) + (p-k+1)\rho(u_k)) + \sum_{1 \le i \le q} (k\lambda(w_i) + (q-i+1)\rho(w_i))$$

2. *Pretínanie hrán:*

$$P2 = |\{\{(u_k, w_\alpha), (u_l, w_\beta)\} \subseteq F^d \mid 1 \le k < l \le p, 1 \le \beta < \alpha \le q\}|$$

3. *Pretíanaie hrana-obdĺžnik:*

$$P3 = \sum_{(u_k, u_l) \in F^s} |k - l| + \sum_{(w_\alpha, w_\beta) \in F^s} |\alpha - \beta|$$

Na minimalizovanie $P1$, stačí utriediť vrcholy podľa $\lambda(v) - \rho(v)$. Vrcholom, ktoré majú $\lambda(v) - \rho(v)$ rôzne od nuly pozíciu zafixujeme a ďalej sa nimi nebudeme zaoberať, tak isto si nevšímame hrany, ktoré sú s nimi incidentné. Minimalizovať $P2$ je **NP**-ťažký problém, podobný ako *minimum feedback edge set*, minimalizovať $P3$ je ťiež **NP**-tažký problém, ekvivalentný s *linear arrangement problem*-om. Aproximovateľnosť týchto problémov je rozobraná v piatej kapitole.

Definícia. Nech $G = (Ch(v), F^d, F^S, \sigma(v), \lambda, \rho)$ je lokálna hierarchia $v \in V_i(v), i < L(v)$, $\sigma_{i+1}(v) = (u_1, u_2, \ldots, u_{q-1}, u_q)$, potom *dolné ťažisko* vrcholu w definujeme ako $\frac{1}{|(w,u_q) \in F^d|} \sum_{(w,u_q) \in F^d} q$

Definícia. Nech $G = (Ch(v), F^d, F^S, \sigma(v), \lambda, \rho)$ je lokálna hierarchia $v \in V_i(v), i > 1$, $\sigma_{i-1}(v) = (u_1, u_2, \ldots, u_{q-1}, u_q)$, potom *horné ťažisko* vrcholu w definujeme ako $\frac{1}{|(u_q,w) \in F^d|} \sum_{(u_q,w) \in F^d} q$

Na minimalizáciu $P2$ a $P3$ sa používa metóda, ktorá opakovane vykonáva jednoduché operácie, ako je utriedenie vrcholov podľa ťažiska a pridavanie pomocných vrcholov. Pomocné vrcholy sa pridávajú do stredu hrán z F^s, a následne preberú úlohu vrcholov z $V_{i+1}(v)$ pri triedení podľa ťažiska, potom sa môžu odstrániť.

Presný postup znázorňuje nasledovný pseudokód. `IDU(j)` a `IDL(j)` predstavujú procedúry, ktoré pridajú pomocné vrcholy na hrany v rámci j-tej vrstvy , a vložia ich do hornej, resp. dolnej vrstvy. Pod pridaním pomocného vrcholu x na hranu (u, v), máme na mysli, že vytvoríme dve nové hrany (u, x) a (v, x). `BOL(j)` a `BOU(j)` predstavujú triedenie vrcholov podľa dolného, resp. horného ťažiska.

```
procedure orderGlobal(v)
  if (Ch(v) = {}) exit
  orderLocal(v)
  for (each w from Ch(v)) do
    OrderGlobal(w)
end
procedure orderLocal(v)
  for i:=1 to numberOfIterations
    IDU(1)
    BOU(1)
    for j:=2 to n do
      BOU(j)
      IDU(j)
      BOU(j)
    IDL(n)
    BOL(n)
    for j:=n-1 downto 1 do
      BOL(j)
      IDL(j)
      BOL(j)
end
```

3.6 Umiestnenie vrcholov (4.krok)

Keď už sú vrcholy vo vhodnom poradí, tak je priradnie súradníc priamočiaré. Použijeme rekurzívny prístup. Rozmery listov závisia od ich obsahu, napríklad, aký dlhý text obsahujú. Rozmery nelistového vrcholu určíme na základe velkostí jeho detí, ktoré umiestnime do neho tak, aby boli v každej vrstve rovnomerné rozostupy. Pozície vrcholov určíme najprv relatívne vzhľadom na rodiča, a na konci ešte raz prejdeme cely graf, aby sme určili absolútne pozície.

Hrany povedú vždy zo stredu spodnej strany vrcholu do stredu hornej strany druhého vrcholu. Hrany, ktoré boli odobrané po normalizácii, opäť rekonštruujeme tak, že ich povedieme ako viac hrán pomedzi pomocné vrcholy, ktorými sme ich nahradzovali. Pomocným vrcholom pridelíme nulovú veľkosť a nekreslíme ich. Druhá možnosť je, že nahradené hrany povedieme priamo, podľa pôvodných vrcholov, medzi ktorými viedli.

Kapitola 4

Implementačné detaily

V tejto kapitole sa venujeme implementovanej verzii algoritmu. Popíšeme formát vstupu a výstupu, dátové štruktúry, efektivitu a celkový návrh softvérového riešenia. Hlavnými požiadavkami na riešenie sú:

1. Správnosť: Požité algoritmy počítajú presne tie funkcie, aké sú popísané v zadaní.

2. Nezávislosť krokov: Kroky algoritmu sa vykonávjú lineárne, ďalší krok začne, až keď predchádzajúci skončí. Každý krok musí mať jasne definovaný vstup a výstup, a to tak aby, bol v čo najkompaktnejšej podobe.

3. Flexiblinosť: Všetky výpočty by mali byť napísané tak, aby ich bolo možné ľahko a transparentne modifikovať. Čo je niekedy aj na úkor efektivity.

4.1 Vstup

Kedže vstupom algoritmu je graf, rozhodli sme sa pouziť formát GraphML [12], ktorý je založený na XML, a je akýmsi štandardným formátom pre reprezentáciu grafov. Dokáže uchovávať nielen jednoduché orientované alebo neorientované grafy, zmiešané grafy, ale aaj zložené grafy a grafy obsahujúce hyperhrany. Existujú viaceré softvérové riešenia, ktoré ho tiež využívaju.

Okrem GraphML sa používajú aj iné formáty založené na XML (GXL, GraphXML, XGML). Formáty, ktoré nie sú založené na XML (Dot, GML),

sú však v súčastnosti výrazne rozšírenejšie.

GraphML má veľmi jednoduchú štruktúru, ale obsahuje mechaniznmy, pomocou ktorých sa dá veľmi efektívne rozširovať. Pre naše účely stačí využívať nasledovné elementy:

1. `graph` – tento element obsahuje popis grafu, ktorý tvoria vnorené elementy, `edge`, `node`, `graph`.

2. `node` – predstavuje jeden vrchol grafu, má atribút `id`, ktorý predstavuje identifikátor vrcholu. Môže obsahovať ďalšie vnorené elementy typu `edge`, `node`, `graph`, vďaka čomu umožňuje jednoducho reprezentovať inkluzívnu reláciu.

3. `edge` – hrana predstavujúca adjecenčnú hranu, krajné vrcholy sú určené atribútmi `source` a `target`, ktoré obsahujú identifikátory vrcholov.

Príklad:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <edge source="n0" target="n1"/>
  </graph>
</graphml>
```

Na načítanie XML súboru bola využitá podpora jazyka Java. Konkrétne SAX (Simple API for XML parsing), čo urýchlilo vývoj.

4.2 Výstup

Po vypočítaní výstupu, sa do každého elementu `node` pridaju atribúty, s názvami `X`, `Y`, ktoré určujú polohu a `Width`, `Height`, ktoré určujú rozmery. Názov atribútov možno špecifikovať ako command-line parametre. Zvyšok súboru, ak je to korektný XML súbor, ostane nezmenený, teda je možné, aby obsahoval aj údaje, ktoré náš program nepozná.

4.3 Dátovy model

Vzhľadom na to, že graf sa počas behu algoritmu viackrát transformuje, prehľadáva a vrcholom sa priraďujú rôzne hodnoty, musí byť jeho reprezentácia dynamického charakteru. V našom modeli, sú graf, vrcholy, a hrany samostané objeky. Najväčšiu flexibilitu dosahujeme tak, že každý vrchol obsahuje zoznam hrán, ktoré sú s ním incidentné a samotná orientácia hrany je vrcholu skrytá, prístupná je cez metódy `walk`, (`backWalk`, `undirectedWalk`), ktoré berú ako parameter vrchol a vrátia vrchol, ktorý je v smere hrany (v protismere, ten druhý), alebo `null` ak sa po hrane požadovaným smerom z vrcholu na vstupe prejsť nedá. Vďaka tomu môžeme hrany otáčať, bez toho aby sa menili vlastnosti vo vrcholoch. Na dosiahnutie dynamickosti stačilo použíť *Generics* triedy, ako `Vector` alebo `TreeSet`.

4.4 Časová zložitosť

Na časovú ani na pamäťovú efektivitu implementácie nebol kladený dôraz, preto nie sú všetky algoritmy optimálne. Hlavným dôvodom je fakt, že celková zložitosť je daná najzložitejšou časťou. V prípade, že by sa namiesto niektorej heuristiky použil algoritmus, ktorý má veľkú časovú zložitosť, ale dosahuje lepšie výsledky, by bolo zbytočné mať efektívne spravené ostatné výpočty.

V tejto časti detailnejšie analyzujeme jednotlivé kroky a podáme návod, ako by sa dali zefektívniť. Počet adjecenčných hrán označujeme ako m, počet vrcholov ako n.

4.4.1 Hierarchizácia

Počas tohto kroku prebehne viacero prehľadávaní, ktoré majú dokopy zložitosť $O((m+n)s(n))$, kde $s(n)$ je čas, ktorý trvá dotaz na zistenie, či nejaký vrchol je prvkom množiny W. V implementácii je použitá trieda TreeSet, ktorá má zložitosť dotazu $O(\log n)$. Existuje spôsob ako implentovať algoritmus tak, aby $s(n)$ mala amoritozvanú zložitosť $O(1)$. Kedže každý prvok pridáme do W iba raz, a vymazaváme iba tak, že vymažeme všetky prvky naraz. Stačí, aby si každý vrchol pamätal, či je v množine W, a potom mať

ešte zoznam vrcholov, ktoré sú v množine W , aby sme mohli efektívne odstraňovať prvky.

Ďalším dôležitým aspektom je reprezentácia úrovní, $clev(v)$ môže byť postupnosť s dĺžkou $\Omega(n)$, pre $\Omega(n)$ vrcholov , z čoho vyplýva pamäťová zložitosť $\Omega(n^2)$. V praxi však nemá význam, aby inkluzívny podgraf mal hĺbku väčšiu ako je $O(\log_2 n)$. Ale ani $O(n \log_2 n)$ nie je optimálne. Úrovne sa dajú reprezentovať tak, že každý vrchol v si pamätá len $tail(clev(v))$, a v prípade potreby si $clev(v)$ sknštruuje podľa rekurzívnej rovnice $clev(v) = append(clev(Pa(v)), tail(clev(v)))$ a $clev(root) = (1)$.

Najkritickejším problémom v Hierarchizačnom kroku je odstraňovanie cyklov. Za predpokladu $\mathbf{P} \neq \mathbf{NP}$ neexistuje polynomiálny algoritmus, ktorý by našiel najmenší počet hrán, ktoré na to stačia. V programe používame greedy algoritmus, pracujúci v lineárnom čase.

4.4.2 Normalizácia

V tomto kroku sa do grafu pridávajú nové vrcholy, čo sa premietne do časovej zložitosti neskorších krokov. Počet pridaných vrcholov možno zhora odhadnúť ako $O(n^3)$.

4.4.3 Usporadúvanie vrcholov

Najzložitejšou časťou je konštrukcia lokálnych hierarchií, kde sa viackrát kontrahujú tie isté podstromy. Efektívnejšie riešenie kontrahuje vrcholy v postorder poradí, a pri každom si pamätá hrany, ktoré by boli v danej lokálnej hierarchii.

Dĺžku výpočtu ovplyvňuje aj počet iterácií heuristických metód, ktoré používame na riešenie **NP**-úplných optimalizačných problémov.

4.4.4 Umiestňovanie vrcholov

Umiestňovací algoritmus pozostáva z jedného inorder prejdenia inkluzívneho podgrafu, pričom v každom vrchole sa vykoná $O(\delta(v))$ výpočtov, čo vedie k zložitosti $O(m)$.

Kapitola 5

Viac o optimalizačných problémoch

V tejto kapitole sa bližšie venujeme optimalizačným problémom, ktoré sme v algoritme použili. Uvedieme alternatívne heuristiky a aproximatívne algoritmy, ktoré by sme mohli využiť. A pre úplnosť dodáme výsledky o **NP**-úplnosti a aproximovateľnosti kľúčových problémov. Kapitola má informatívno prehľadový, neformálny charakter.

Už v prvom kroku algoritmu, potrebujeme odstrániť čo najmenší počet hrán tak, aby nám ostal acyklický graf. Tento problém je jeden z klasických **NP**-úplných problémov a má názov Minimum Feedback Arc Set. Ukážeme si dva alternatívne algoritmy, ktoré by sme mohli použiť na jeho približné riešenie a spôsob, ako pomocou neho vyriešiť problém minimalizácie počtu pretínaní hrán v 2-vrstvovej lokálnej hierarchii so zafixovanou jednou vrstvou, čo je tiež **NP**-úplný problém.

Snahu o minimalizovanie pretínaní obdĺžnikov s čiarami sme formulovali ekvivalentne ako známy **NP**-úplný problém s názvom Optimal Linear Arangement Problem.

5.1 Minimum Feedback Arc Set Probmlem

Definícia. Nech $G = (V, E, w)$ je ohodnotený digraf $w: E \to R$,

$$FAS(G) = \min_{S \subseteq E}\{\sum_{e \in S} w(e) \mid G \setminus S \text{ je acyklický}\}$$

Tvrdenie 5.1.1 *Za predpokladu* $\mathbf{P} \neq \mathbf{NP}$, *existuje* $\alpha > 1$ *také, že neexistuje polynomiálny algoritmus, ktorý by pre všetky ohodnotené digrafy* G *našiel takú množinu* $S \subset E$, *že* $G \setminus S$ *je acyjklický a* $\sum_{v \in S} w(v) \leq \alpha FAS(G)$.

Dôkaz je uvedený v [11]. Najväčšia známa α je približne 1.36. Najlepší známy aproximačný algoritmus pre MFAS, nájde vždy množinu S, ktorá je maximálne $O(\log n \log\log n)$ krát väčšia, ako najlepšie riešenie [4]. Využaduje si však riešenie lineárneho programu. My si ukážeme algoritmus, ktorý nájde najviac λ krát horšie riešenie, kde $\lambda = \lambda(G)$ je dĺžka najdlhšieho jednoduchého cyklu v grafe G. Druhý algoritmus je založený na greedy metóde, o ňom nevieme povedať, aký je dobrý. Má však lineárnu časovú zložitosť.

5.1.1 Algoritmus λ FAS

Hlavnou myšlienkou algoritmu je rozloženie hodnotovej funkcie w na súčet viacerých funkcií $w = w_1 + w_2 + \ldots + w_k$ takých, aby sme cenu optimálneho riešenia pre $G_i = (V, E, w_i)$, ktorú označíme ako S_i, mohli ľahko nájsť. Cenu optimálneho riešenia pre $G = (V, E, w)$ označujeme ako S. Rozklad budeme konštruovať induktívne, po krokoch. Ak $f((u, v)) = 0$, považujeme to za to isté, ako keby hrana (u, v) v grafe $G_f = (V, E, f)$ nebola. V i-tom kroku máme na vstupe funkciu $r_i = w - \sum_{j<i} w_j$ a skonštruujeme funkciu w_i, pričom množina $T = \bigcup T_i$, označuje hrany, ktoré budú algoritmom vybrané, ako celkové riešenie. Ak $G_i = (V, E, r_i)$ je acyklický, potom $T_i = \emptyset$ a $w_i = r_i$ a môžeme skončiť. Ďalej nech G_i obsahuje cyklus C a ε je cena najlacnejšej hrany na cykle C, podľa funkcie r_i. Položme $T_i = \{e \in C \mid r_i(e) = \varepsilon\}$, $w_i(e) = 0$ ak e neleží na cykle C, a $w_i(e) = \varepsilon$ ak e leží na C.

Cena vybraného riešenia T_i nie je oveľa väčšia, ako optimálne riešenie S_i,

$$\varepsilon = w_i(S_i) \leq w_i(T_i) \leq \lambda\varepsilon$$

Po posledom groku, je graf acyklický a všetky hrany, ktoré sme odstránili sme zahrnuli do $T = \bigcup_{i \leq k} T_i$, teda T zanechá po odstránení acyklický graf.

$$S(w) = \sum_{i \leq k} w_i(S) \geq \sum_{i \leq k} w_i(S_i) \geq \frac{1}{\lambda(G)} \sum_{i \leq k} w(T_i)$$

Pre účely dôkazu hore uvedenej nerovnosti, sa budeme tváriť že hrany si v každom kroku kupujeme. Vždy, keď zmenšíme cenu hrany o ε znamená to, že sme za ňu zaplatili. V každom kroku zaplatíme najviac $\lambda w_i(S_i)$, lebo cena optimálneho riešenia pre w_i, je ε, stačí odobrať najlacnejšiu hranu z C a dĺžka C je podľa definície menšia ako $\lambda(G)$. Ak je cena nejakej hrany 0, tak sme ju kúpili a dáme ju do T_i. Niektoré hrany nakoniec nekúpime, aj keď sme za ne už niečo zaplatili. Peniaze, ktoré sme minuli, sú teda najviac $\lambda \sum_{i \leq k} w_i(S_i)$, čo nám dáva nerovnosť

$$\sum_{i \leq k} w_i(S_i) \leq w(S) \leq \sum_{i \leq k} (w(T_i)) \leq \lambda \sum_{i \leq k} w_i(S_i)$$

z ktorej vyplýva $w(T) \leq \lambda w(S)$.

Ak sú ceny hrán jednotkové, je analýza jednoduchšia. Algoritmus v každom kroku odstráni jeden cyklus, zoberie všetky hrany, ktoré obsahuje. Ak odstránil k cyklov, znamená to, že graf obsahoval k hranovo disjunktných cyklov. A teda $FAS(G) \geq k$, lebo z každého cyklu sa musí odobrať aspoň jedna hrana. Nájdené riešenie T, obsahuje však najviac λk hrán, teda $|T| \leq \lambda k \leq \lambda FAS(G)$. Algoritmus sa dá implementovať s časovou zložitosťou $O(mn)$.

5.1.2 Algoritmus Greedy FAS

V tomto algoritme najprv skonštruujeme permutáciu vrcholov π potom, ak pre nejakú hranu platí (u, v), $\pi(u) > \pi(v)$, tak ju odstránime. Permutáciu konštruujeme v iteráciah. Vždy najprv dáme všetky ústia na koniec postupnosti s_1 a pramene na koniec s_2. Potom nájdeme vrchol, ktorý má najväčší rozdiel vonkajšieho a vnútorného stupňa, a ten dáme na koniec s_2 (alebo vrchol s najmänším rozdielom dáme na koniec s_1). Nakoniec s_2 obrátime a výstup vygenerujeme podľa permutácie $s_1 s_2$.

Algoritmus možno zovšeobecniť pre ohodnotené grafy, ak rozdiel stupňov vrcholov nahradíme rozdielom cien hrán.

5.1.3 NP-úplnosť MFAS

Tvrdenie 5.1.2 *MFAS je* **NP***-uplný [6].*

Dôkaz. Ukážeme redukciu Vrcholového pokrytia na MFAS.

Nech $G = (N, A)$, je inštancia vrcholového pokrytia. Zostrojíme graf

$$H = (N \times \{0, 1\}, \{((p, 0), (p, 1)) \mid p \in N\} \cup \{((u, 1), (v, 0)) \mid (u, v) \in A\}$$

A ukážeme, že veľkosť minimového vrcholového pokrytia je rovnaká, ako veľkosť najmenšej množiny hrán, ktorá z H spraví po odobraní acyklický graf. Z definície vrcholového pokrytia vyplýva, že z každej hrany musíme použiť aspoň jeden vrchol. Odstraňovať má zmysel iba hrany typu $((v, 0), (v, 1))$, lebo každá hrana $((v, 1), (u, 0))$, leží iba na tých cykloch, na ktorých aj $((v, 0), (v, 1))$. Každá hrana $(u, v) \in A$, vytvára cyklus

$$(u, 0), (u, 1), (v, 0), (v, 1), (u, 0)$$

preto podmienka, že z každého cyklu musíme odstrániť aspoň jednu hranu, je ekvivalentná podmienke, že musíme do vrcholového pokrytia vybrať aspoň jeden vrchol z každej hrany. Medzi vrcholmi grafu G a hranami H typu $((u, 0), (u, 1))$, existuje bijekcia φ daná predpisom $\varphi: v \to ((v, 0), (v, 1))$, ktorá každému vrcholovému pokrytiu G priradí všetko-cyklo-rušiacu množinu v H. Teda aj veľkosti minimových množín sú rovnaké.

5.2 Minimalizovanie pretínaní hrán

V našom algoritme riešieme problém minimalizácie počtu pretínaní hrán, medzi dvoma susednými vrstvami. Náš postup je taký, že jednu vrstvu zafixujeme, a druhú utriedime podľa ťažiska. Táto metóda sa nazýva BC-metóda, a nájde optimálne riešenie v prípade, že vrcholy na nezafixovanej vrstve majú stupeň najviac 2 [7]. Pre prípad, keď každý vrchol na nezafixovanej strane má stupeň 4, a vrcholy na zafixovanej strane majú stupeň 1 (takýto graf voláme

les hviezd stupňa 4) je úloha **NP**-úplna[7]. Zaujímavé je, že keď dovolíme hýbať aj zafixovanou vrstvou, je úloha pre lesy hviezd ľubovolného stupňa riešitelná aj BC-metódou, keď zotriedime podľa ťažiska zafixovanú vrstvu.

Teraz si ukážeme postup, ako možno počet pretínaní čiar minimalizovať pomocou riešenia problému MFAS.

Majme graf $G = (L_1 \cup L_2, E)$, kde $E \subset L_1 \times L_2$, $L_1 = (v_1, v_2, \ldots, v_n)$ a $L_2 = (w_1, w_2, \ldots, w_m)$. L_1 je sú vrcholy nezafixovanej vrstvy, a L_2 vrcholy zafixovanej. Úlohou je nájsť takú permutáciu π vrcholov L_1, aby bol počet pretínaní hrán minimálny. Označme počet pretínaní sa hrán incidentných s vrcholmi v_i,v_j, keď $\pi(i) < \pi(j)$ ako $cr(v, w)$.

$$cr(v_i, v_j) = |\{(w_\alpha, w_\beta) \mid (v_i, w_\alpha) \in E \wedge (v_j, w_\beta) \in E \wedge \alpha > \beta\}|$$

počet pretínaní pre danú permutáciu π je

$$cr(\pi, G) = \sum_{\pi(i)<\pi(j)} cr(v_i, v_j)$$

dolný odhad pre $cr(\pi, G)$ je zjavne

$$LB(G) = \sum_{1\leq i<j\leq n} \min\{cr(v_i, v_j), cr(v_j, v_i)\}$$

Zostrojme ohodnotený digraf

$$G_R = (L_1, \{(v_i, v_j) \mid cr(v_j, v_i) < cr(v_i, v_j)\}, w((v_i, v_j)) = cr(v_i, v_j) - cr(v_j, v_i)))$$

Naším cieľom je dokázť nasledovnú rovnosť

$$FAS(G_R) + LB(G) = \min_{\pi \in S_n} cr(\pi, G)$$

každá permutácia vrcholov π indukuje množinu hrán

$$F = \{(v_i, v_j) \in E(G_R) \mid \pi(j) > \pi(i)\}$$

Po odobraní F je graf G_R čiastočným usporiadaním, a teda je acyklický. Cena S je

$$\begin{aligned} cost(S) &= \sum_{\pi(i)<\pi(j)} (cr(v_i, v_j) - cr(v_j, v_i)) \\ &= cr(\pi, G) - \sum_{\pi(i)<\pi(j)} \min\{cr(v_j, v_i), cr(v_i, v_j)\} \\ &= cr(\pi, G) - LB(G) \end{aligned}$$

$FAS(G_R)$ je dolný odhad pre $cost(S)$, z čoho vyplýva nerovnosť

$$FAS(G_R) + LB(G) \leq \min_{\pi \in S_n} cr(\pi, G)$$

Nech, F je najlacnejšia množina taká, že po odstránení zanechá acyklický podgraf. Definuje permutáciu π_F danú čiastočným usporiadním $G_R \setminus F$. Permutácia π_F indukuje množinu hrán F_2, ktorá je podmnožinou F. Z minimality $cost(F)$ vyplýva, že $cost(F_2) = cost(F) = FAS(G_R)$. Keď k výsledku pridáme vyššie dokázanú rovnosť, dostaneme $cost(F_2) = cr(\pi_F, G_R) - LB(G)$, a následne $FAS(G_R) + LB(G)) \geq \min_{\pi \in S_n} cr(\pi, G)$.

Na minimalizovanie počtu pretínaní stačí teda nájsť v G_R najlacnejšiu množinu F, ktorá zanechá acyklický graf a vrcholy usporiadať ľubovoľne podľa čiastočného usporiadania, ktoré je dané grafom $G_R \setminus F$. Konštrukciu možno zovšeobecniť s rovnakým výsledkom pre ohodnotené grafy, čo by mohla byť výhoda oproti BC-metóde, ktorá takúto možnosť neposkytuje.

5.3 Optimal Linear Arrangement Problem

Tento problém je o tom, že treba pre graf $G = (V, E)$ nájsť takú permutáciu vrcholov π, aby $\sum_{(u,v) \in E} |\pi(u) - \pi(v)|$ bolo čo najmenšie.

Dôkaz **NP**-úplnosti, dokonca aj pre intervalové grafy môže čitateľ nájsť v [2]. Pre stromy je úloha efektívne riešiteľná[5]. Pre husté grafy, kde $|E| = \Theta(|V|^2)$, pre ľubovolné α existuje polynomiálny algoritmus, ktorý nájde riešenie, ktoré je najviac α krát horšie[1].

Pre všeobecné grafy je doteraz najlepší známy algoritmus, taký že nájde najviac $O(\log n)$ krát horšie riešenie [9].

My na riešenie tohoto problému používame triedenie podľa ťažiska s vkladaním pomocných vrcholov. Takýto algoritmus je rýchly, a experimentálne bolo overené, že pre potreby kreslenia grafov funguje dostatočne dobre. Na jeho opodstatnenie môžeme poznamenať, že redukcia zachovavájuca aproximáciu, z OLAP na problém minimalizácie počtu pretínaní hrán grafu, v ktorom má každý vrhol priradenú jednu z dvoch vrstiev, pozostáva iba z toho, že pridáme pomocné vrcholy tak, ako to robíme v použitom algoritme.

Na záver povieme, ako funguje alternatívny algoritmus, ktorý by sme mohli použiť.

5.3.1 Algoritmus Greedy LA

Na začiatku, nech sú vrcholy v ľubovolnom poradí. Potom niekoľkokrát zopakujeme nasledovnú operáciu. Vyberme náhodný vrchol v, a vložme ho na novú pozíciu tak, aby tvoril medián medzi jeho susedmi.

Ak môžeme hýbať len jedným vrcholom, tak jeho umiestnenie ako medián jeho susedov minimalizuje sumu vzdialeností. Princíp je teda podobný, ako keď triedime podľa ťažiska, lebo ťažisko a medián sú väčšinou dosť podobné hodnoty.

Kapitola 6

Záver

Implementovali sme hierarchizačný algoritmus na kreslenie zložených digrafov, ktorý sa dá ľahko integrovať s už existujúcimi aplikáciami. Zistili sme, že veľa problémov, ktoré sú spojené s dobrým nakreslením grafu sú **NP**-úplné. Teoreticky efektívne aproximatívne algoritmy nie sú v tomto prípade pre prax použiteľné, preto sme použili heuristiky, ktoré sa javia ako dostatočne dobré. Na prácu možno nadviazať preskúmaním ďalších prístupov ku kresleniu grafov, alebo nájdením lepších aproximatívnych algoritmov pre dôležité problémy.

Dodatok A

Príloha

K práci je priložené CD-médium, na ktorom sa nachádza zdrojový kód softvérového riešenia.

Literatúra

[1] Frieze A. Arora S. and Kaplan H. A new rounding procedure for assignment problem with application to dense graph arrangement problems. In *37th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 21–30, 1996.

[2] Johanne Cohen, Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, and Gregory Kucherov. Optimal linear arrangement of interval graphs. In *MFCS*, pages 267–279, 2006.

[3] R. Diestel. *Graph Theory*. Springer-Verlag, 2006.

[4] Schieber B. Even G., Naar J. and Sudan M. Approximating minimum feedback sets and multi-cuts in directed graphs. In *4th Int. Coonf. on Integer Prog. and Combinatorial Optimization*, pages 14–28, 1995.

[5] M.A. Goldberg and I.A. Klipker. Minimal placing of trees on a line. Technical report, Physico-Technical Institute of Low Temperatures, 1976.

[6] R.M. Karp. *Complexity of Computer Computations*, chapter Reducibility among combinatorial problems. Plenum press, 1972.

[7] Xavier Muoz, Walter Unger, and Imrich Vrto. One sided crossing minimization is np-hard for sparse graphs. In *GD '01: Revised Papers from the 9th International Symposium on Graph Drawing*, pages 115–123, London, UK, 2002. Springer-Verlag.

[8] Škoviera M. Olejár D. Úvod do diskrétnych matematických štruktúr. 2004.

[9] S. Rao and Richa. New approximation techniques fo some ordering problems. In *ACM-SIAM Symp. on Discrete Algorithms*, pages 211–218, 1998.

[10] K. Sugiama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, 21(4):876–892, July/August 1991.

[11] Kann V. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology,Stockholm, 1992.

[12] Graphml file format. http://grpahml.graphdrawing.org.