FAKULTA MATEMATIKY, FYZIKY
A INFORMATIKY
UNIVERZITA KOMENSKÉHO
BRATISLAVA

Kreslenie grafov

Diplomová práca

Autor : Ľubomír Čech
Vedúci : Prof. RNDr. Branislav Rovan Phd.

Bratislava
apríl 2005

Čestné prehlásenie :

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry.

V Bratislave
29. apríla 2005

_____

Ďakujem svojmu vedúcemu diplomovej práce, Prof. RNDr. Branislavovi Rovanovi Phd. za cenné rady a pripomienky pri písaní tejto práce.

Obsah

Kapitola 1

Úvod

Graf je abstraktná štruktúra, ktorá je často používaná na zobrazenie informácie. Graf môže zobrazovať informácie, ktoré môžu byť modelované ako objekty a spojenia medzi týmito objektami. Preto skoro každá oblasť počítačovej vedy a softvérového priemyslu používa grafy v nejakej forme na reprezentovanie informácií.

Graph drawing sa zaoberá problémom zobrazenia geometrickej reprezentácie grafov. Výskum v graph drawingu pokračuje v niekoľkých rôznych oblastiach, vrátane diskrétnej matematiky (teória grafov, geometrická teória grafov), algoritmov (grafové algoritmy, dátové štruktúry, výpočtová geometria, VLSI) a interakcie človek-počítač (vizuálne jazyky, grafický user interface, softvérová vizualizácia).

Existuje viacero programov, ktoré slúžia na kreslenie, vytváranie a vizualizáciu grafov. Skoro každý z týchto programov používa vlastný formát súborov na pracovanie s grafmi. Práve preto sa na Graph Drawing Symposium 2000 vo Williamsburgu začala práca na novom formáte súborov pre grafy, GraphML[1]. Návrh na štrukturovanú vrstu bol prezentovaný na Graph Drawing Symposium 2001 vo Viedni. GraphML je obsiahly a jednoducho použitelný súborový formát, ktorý podporuje orientované grafy, neorientované grafy, hypergrafy, hierarchické grafy, grafické reprezentácie, referencie na externé dáta, aplikačno špecifické atribútové dáta a jednoduché parsery. Na rozdiel od iných súborových formátov pre grafy, GraphML nepoužíva vlastnú syntax. Namiesto toho je založený na XML a preto je ideálne vhodný ako spoločný menovateľ pre všetky druhy funkčností, ako napr. vytváranie, archivovanie alebo pracovanie s grafmi.

[1]Viac o formáte GraphML je v [22].

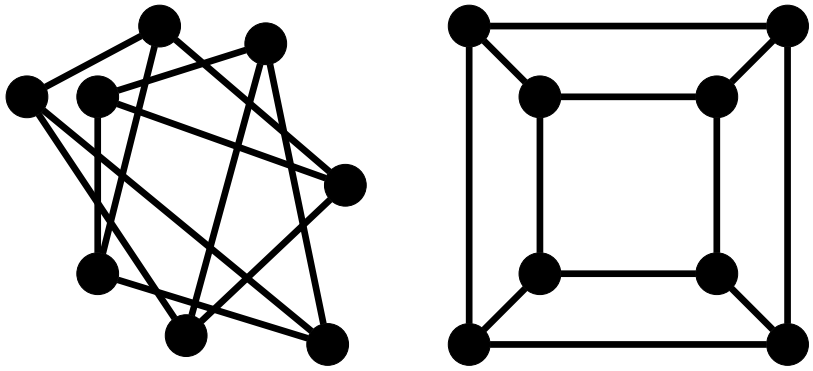
V tejto práci sa budeme zaobrerať kreslením grafov, ktoré majú vrcholy v dvoch skupinách. V tejto kapitole uvedieme základné informácie o kreslení grafov, typy kreslenia a aplikácie kreslenia grafov. V kapitole 2 popíšeme základné definície, pravidlá a kritériá pri kreslení grafov, niektoré algoritmy, NP-ťažké a NP-úplné problémy, niektoré otvorené problémy. V kapitole 3 uvedieme metódy kreslenia základných grafov ako sú stromy, kompletné grafy, planárne grafy ako aj metódy kreslenia všeobecných grafov. V kapitole 4 sa budeme zaoberať dokresľovaním hrán a vrcholov do už nakresleného grafu (neorientovaný graf, straight-line drawing). Uvedieme dve nové heuristiky na dokresľovanie vrcholov do grafu. V kapitole 5 sa budeme venovať kresleniu grafov, ktoré majú vrcholy v dvoch skupinách. Žiadna dostupná literatúra sa problémom kreslenia dvojfarebných grafov nezaoberala. Budú tam uvedené rôzne prístupy na kreslenie takýchto grafov, vrátane dvoch nových algoritmov založených na dokresľovaní vrcholov. V kapitole 6 tieto prístupy experimentálne porovnáme vzhľadom na zvolenú metriku nakreslenia.
V prílohe bude krátky manuál k implementovanému programu na vytváranie a kreslenie dvojfarebných grafov ako aj prehľad najhlavnejších implementovaných funkcií v tomto programe. Budú tam uvedené aj niektoré problémy, ktoré sa objavili počas implementácie nad knižnicou algoritmov `LEDA` spolu s ich riešeniami. V poslednej časti prílohy budú ukážky rôznych nakreslení grafov v implementovanom programe. Táto práca obsahuje aj CD-prílohu, na ktorej nájdeme elektronickú verziu tejto diplomovej práce spolu s jej zdrojovými súbormi, implementovaný program, jeho zdrojové súbory, výsledky testov a obrázky nakreslenia rôznych grafov v tomto programe.

1.1 Typy a aplikácie kreslenia grafov

V kreslení grafov sa vrcholy reprezentujú ako body (alebo ako geometrické útvary napr. kružnica alebo obdĺžnik) a hrany sú reprezentované krivkami tak, že každé dve hrany sa pretínajú najviac v konečnom počte bodov.

Pri kreslení grafov v 2D priestore sa používajú rôzne prístupy kreslenia hrán a vrcholov, podľa toho k čomu slúži výsledné nakreslenie grafu. Niektoré z nich sa dajú medzi sebou aj kombinovať (napr. polyline + planar drawing) a niektoré sa dajú rozšíriť aj na 3D priestor. Najpoužívanejšie typy kreslenia grafov sú :

Straight-line drawing – každá hrana je úsečka



Obrázok 1.1: Dve rôzne nakreslenia toho istého grafu, z druhého nakreslenia je vidieť, že graf je planárny

Polyline drawing – každá hrana je reprezentovaná lomenou čiarou (môže byť aj viac krát lomená)

Orthogonal drawing – každá hrana je zreťazenie horizontálnych a vertikálnych úsečiek

Planar drawing – žiadne dve hrany sa nekrižujú

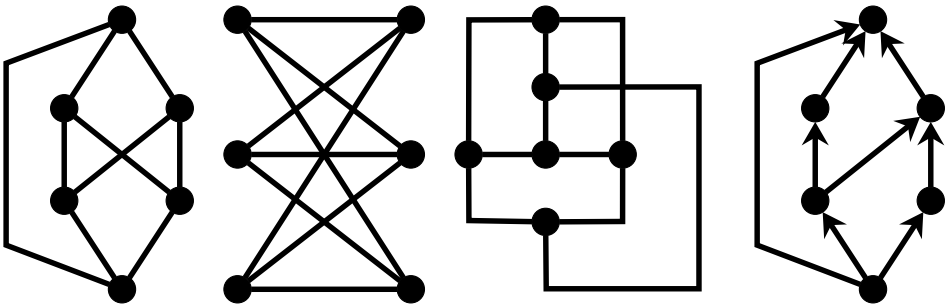
Upward drawing – kreslenie orientovaného grafu, kde každá hrana je monotónne neklesajúca v smere osi y

Layered drawing – kreslenie vrstvového grafu tak, že vrcholy patriace tej istej vrstve ležia na rovnakej horizontálnej čiare (taktiež nazývané hierarchical drawing)

Visibility drawing – kreslenie grafu založené na geometrickej viditeľnosti, napr. vrcholy možu byt kreslené ako horizontálne úseky a hrany sú vertikálne viditeľné časti

Dominance drawing – upward drawing acyklického orientovaného grafu, kde cesta z vrcholu u do vrcholu v existuje vtedy a len vtedy, ak sú súradnice vrchola v väčšie ako súradnice vrchola u (teda $x(u) \leq x(v)$ a $y(u) \leq y(v)$)

Straight-line a orthogonal kreslenia sú špeciálnym typom polyline kreslenia. Polyline kreslenie poskytuje veľkú flexibilitu, pretože môže zobrazovať hrany ako aproximované krivky. Avšak, hrany s viac ako dvoma alebo troma ohybmi



Obrázok 1.2: Typy kreslenia: (a) polyline drawing $K_{3,3}$ (b) straight-line drawing $K_{3,3}$ (c) orthogonal drawing $K_{3,3}$ (d) planar upward drawing acyklického orientovaného grafu

môžu byť ťažké na sledovanie okom. Taktiež systém, ktorý funguje na polyline kreslení je oveľa komplikovanejší ako systém založený na straight-line kreslení. Preto záleží na jednotlivých aplikáciách, či je preferované polyline alebo straight-line kreslenie. Keď sú vrcholy reprezentované bodmi, tak ortogonal kreslenie je možné iba pre grafy s maximálnym stupňom vrchola štyri.

Vizualizácia komplexných štruktúr je kľúčovou zložkou podpory nástrojov pre veľa aplikácií vo vede a inžinierstve. Napríklad softvérové inžinierstvo (call graphs, class hierarchies), databázové systémy (entity-relationship diagramy), digitálne knižnice, World Wide Web browsovanie (hypermediálne dokumenty), distribuované počítanie, VLSI (symbolické výpočty), elektronické systémy (block diagramy, circuit schematics), projekt management (diagramy PERT, organization charts), medicína (concept lattices), telekomunikácie (pokrytie telefónnych sietí), právo (conceptual nets).

Najdôležitejšie aplikácie kreslenia grafov :

- Software engineering (hierarchie classov)
- Databázové systémy (ER-diagramy)
- Projekt management (PERT diagramy)
- Reprezentácia znalostí (isa hierarchie)
- Telekomunikácie (pokrytie telefónov)

- www (browsing history)

Navyše okrem hore zmienených oblastí sa grafy používajú na zobrazovanie informácií v aplikáciách pre kompilátory, inžinierstvo a CAD, paralelné architektúry, informačný manažment, siete a systémový manažment, manažment finančných transakcií, objektovo orientovaná analýza/design, modelovanie sietí chemických reakcií a v mnohých iných aplikáciách.

Kapitola 2

Základné definície a algoritmy

V tejto kapitole si uvedieme základné definície, čo je to graf, podgraf, kompletný graf, strom, komplement grafu, planárna reprezentácia grafu, planárny graf[1]. Ďalej sa budeme venovať základným pravidlám a kritériám, na ktoré sa dbá pri kreslení grafov. Potom si uvedieme najdôležitejšie algoritmy kreslenia grafov, NP-ťažké a NP-úplné problémy súvisiace s kreslením grafov ako aj niektoré otvorené problémy z graph drawingu.

2.1 Definície

Graf sa skladá z vrcholov a hrán, pričom každá hrana spája dva vrcholy.

Definícia 2.1.1 *Graf je dvojica $G = (V, E)$ disjunktných množín, kde $E \subseteq V^2$. Prvky V sú vrcholy grafu G ($V(G)$), prvky E sú jeho hrany ($E(G)$).*

Definícia 2.1.2 *Počet vrcholov grafu G nazývame rád grafu G a označujeme $|G|$. Počet jeho hrán označujeme $||G||$.*

Definícia 2.1.3 *Vrchol v je incidentný s hranou e, ak je jedným z koncových vrcholov (koncov) hrany e. Hovoríme, že hrana e spája vrcholy, ktoré sú s ňou incidentné. Fakt, že hrana e spája vrcholy v_1 a v_2, označujeme $e = (v_1, v_2)$ a hovoríme, že vrcholy v_1 a v_2 sú susedné.*

Podgraf je nejakou podmožinou grafu, obsahuje niektoré vrcholy pôvodného grafu a niektoré hrany vedúce medzi nimi.

Definícia 2.1.4 *Nech $G = (V, E)$. Graf $G' = (V', E')$, kde $V' \subseteq V$ a $E' \subseteq E$, nazývame podgraf grafu G.*

[1] Všetky ostatné definície týkajúce sa grafov sa nachádajú v [3] alebo v [24].

K najdôležitejším triedam grafov patria kompletné grafy a stromy. Kompletný graf má hrany medzi každými dvoma rôznymi vrcholmi grafu. Strom je súvislý graf, ktorý neobsahuje žiaden cyklus. Strom môže byť zakorenený alebo nezakorenený, podľa toho či obsahuje význačný vrchol.

Definícia 2.1.5 *Graf G sa nazýva kompletný ak $E(G) = \{(x, y)|x, y \in V(G), x \neq y\}$, to znamená, že medzi každými dvoma rôznymi vrcholmi existuje hrana. Označujeme ho K^n, kde n je počet vrcholov grafu G.*

Definícia 2.1.6 *Acyklický graf, t.j. taký, ktorý neobsahuje žiadne cykly, nazývame les. Súvislý les je strom. Často je jeden konkrétny vrchol stromu určený ako koreň. Strom, kde je koreň určený sa nazýva zakorenený strom. Strom bez koreňa sa nazýva nezakorenený (voľný) strom.*

Komplement grafu je jeho doplnok vrámci množiny hrán, to znamená, že obsahuje práve tie hrany, ktoré neobsahoval pôvodný graf.

Definícia 2.1.7 *Nech G je graf rádu n. Graf $G' = K^n - G$ nazývame komplement grafu G.*

Z hľadiska vizualizácie grafov je dôležitá množina grafov, ktoré sa nazývajú planárne. Majú tú vlastnosť, že ich môžeme nakresliť do roviny tak, aby sa žiadne dve hrany grafu nepretínali.

Definícia 2.1.8 *Planárna reprezentácia grafu G je jeho zakreslenie na rovinnú plochu tak, že vrcholy nahradíme bodmi a hrany jednoduchými krivkami. Hrany grafu sa pri planárnej reprezentácii nesmú pretínať.*

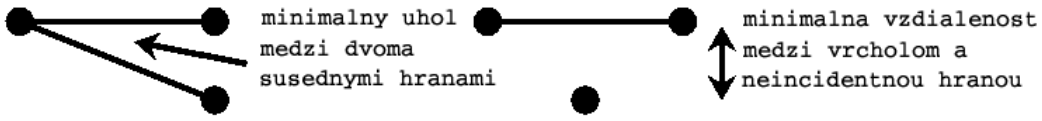
Definícia 2.1.9 *Graf, pre ktorý existuje planárna reprezentácia, sa nazýva planárny graf.*

2.2 Pravidlá a kritériá

Pri kreslení grafov sa dodržujú určité pravidlá, aby výsledné nakreslenie grafu nevypadalo príliš zle a aby bolo lepšie použiteľné. Najdôležitejšie sú celočíselné súradnice vrcholov a aby vzájomné vzdialenosti jednotlivých elementov grafu neboli príliš malé. Pravidlá pri kreslení grafov :

- Celočíselné súradnice pre vrcholy a ohyby
- Predpísaná minimálna vzdialenosť medzi vrcholmi
- Predpísaný minimálny uhol medzi dvoma susednými hranami vychádzajúcimi z toho istého vrcholu (Obrázok 2.1)

- Predpísaná minimálna vzdialenosť medzi vrcholmi a neincidentnými hranami (Obrázok 2.1)



Obrázok 2.1: Pravidlá pri kreslení grafov

Pri kreslení grafov sa takisto dbá na rôzne estetické kritériá, pričom niektoré z nich je veľmi ťažko zmerať (napr. symetria). Ak sa prihliada na viacero kritérií naraz, tak je veľmi ťažké daný graf podľa nich nakresliť, zvyčajne sa darí dodržiavať jedno viac na úkor ostatných. Tak isto záleží aj na tom, k čomu bude výsledné nakreslenie použité, a podľa toho sa zvolí jedno, či viac kritérií, na ktoré sa bude prihliadať. Najpoužívanejšie kritériá :

- Maximálna symetria
- Predchádzanie kríženiu hrán
- Predchádzanie ohybom hrán (polyline, orthogonal)
- Zachovať jednotnú dĺžku hrán
- Rozvrhnúť vrcholy rovnomerne
- Maximálny sklon v ohyboch (polyline)
- Maximálny najmenší uhol (straight-line, polyline)
- Maximálna vzdialenosť medzi vrcholmi pri danej veľkosti nákresu

2.3 Základné algoritmy a problémy

V kreslení grafov je planarita veľmi dôležitá, využíva sa tak pri testovaní či je daný graf planárny, ako aj pri všeobecných metódach kreslenia grafov. Naneštastie veľa problémov spojených s planaritou sú NP-úplné alebo NP-ťažké :

- Rozhodnúť, či pre pevné k sa po zmazaní k-vrcholov stane graf planárnym, je NP-úplný problém (Lewis a Yannakakis, 1980)

- Rozhodnúť, či pre daný graf a pevné k existuje planárny podgraf, ktorý má aspoň k hrán, je NP-úplný problém (Yannakakis, 1978)

- Testovanie upward planarity orientovaných grafov je NP-úplný problém (Garg a Tamassia, 1995)

- Nájdenie maximálneho planárneho podgrafu je NP-ťažký problém[2]

Aj veľa vecí, ktoré sa týkajú kritérií dobrého nakreslenia grafu, ako minimalizácia križení alebo ohybov a ďalšie sú NP-úplné a NP-ťažké problémy :

- Rozhodnúť, či graf G môžeme nakresliť s maximálne k kríženiami, je NP-úplný problém (Garey a Johnson, 1983)

- Nakresliť planárny graf ortogonálne tak, aby mal čo najmenej ohybov je NP-ťažký problém[3]

- Minimalizovanie veľkosti nákresu do mriežky je NP-ťažký problém (Kramer a Van Leeuwen, 1984)

- Minimalizovanie dĺžky najdlhšej hrany je NP-ťažký problém (Miller a Orlin, 1984)

Aj v kreslení grafov existujú otvorené problémy, niektoré z nich môžeme nájsť v [7], [1] alebo v [9]. Existujú aj špecializované www stránky, ktoré sa zaoberajú otvorenými problémami v graph drawingu, napr. [23] alebo [25]. Niektoré zaujímavé otvorené problémy v graph drawingu :

- Daný je neoznačený planárny graf s n vrcholmi a množina n bodov v rovine. Chceme priradiť vrcholy k bodom tak, aby sme vytvorili planárny straight-line nákres. Aká je výpočtová zložitosť tohoto problému?

- Má každý graf s maximálnym stupňom vrchola $\Delta \leq 6$ 3D orthogonal drawing, ktorý nemá viac ako dva ohyby na hranu?

- Ktoré z planárnych grafov majú $O(n)$ veľkosť nákresu pri straight-line drawingu? ($O(n^2)$ majú všetky)

- Dolné ohraničenie pre veľkosť nákresu a ohyby v orthogonal drawingu neplanárnych grafov

[2]Heuristika s dobrými výsledkami na nájdenie maximálneho planárneho podgrafu sa nachádza v [16].

[3]Zdroj pre tento problém je v [6].

Kapitola 3

Kreslenie základných grafov a všeobecné metódy

V tejto kapitole uvedieme metódy kreslenia základných grafov ako sú stromy (zakorenený aj nezakorenený), kompletné grafy, ďalej metódy kreslenia planárnych grafov, ktoré sú veľmi dôležité v graph drawingu. Potom ukážeme najpoužívanejšie metódy kreslenia všeobecných grafov a uvedieme výhody a nevýhody týchto metód.

3.1 Kreslenie stromov

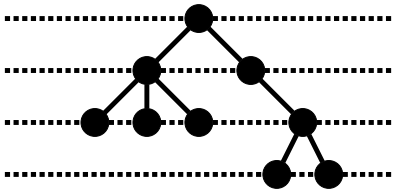
3.1.1 Zakorenený strom

Zakorenený strom obsahuje význačný vrchol, ktorý sa nazýva koreň. Stromová štruktúra je v informatike často používaná, napr. pri ukladaní informácií, prehľadávaní a podobne. Na kreslenie stromov sa používa straight-line drawing alebo ortogonal polyline drawing. Keďže strom neobsahuje cyklus, tak sa dá vždy nakresliť planárne. Zvyčajne sa dbá na nasledujúce estetické kritériá :

1. Vrcholy sú umiestnené pozdĺž horizontálnych čiar prislúchajúcich ich úrovni (vzdialenosť od korena)
2. Je minimálna vzdialenosť medzi dvoma nasledujúcimi vrcholmi na rovnakej úrovni
3. Šírka nákresu je čo najmenšia
4. Otcovský vrchol má synov nakreslených pri sebe

5. Naviac, ak je strom usporiadaný a binárny (ako napr. binárny vyhľadávací strom), tak ľavý a pravý syn vrcholu v sú umiestnené vľavo respektíve vpravo od v

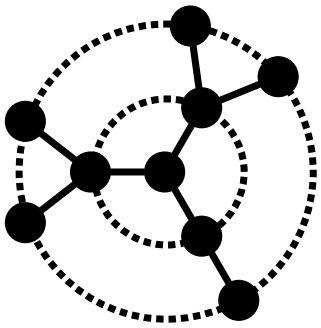
Nakreslenie stromu podľa týchto kritérií je na obrázku 3.1.



Obrázok 3.1: Nakreslenie zakoreneného stromu

3.1.2 Voľný strom

Voľný strom nereprezentuje žiadnu hierarchiu a nemá koreň. Algoritmy pre kreslenie zakorenených stromov možu byť upravené pre nakreslenie voľného stromu napríklad tak, že za koreň zvolíme nejaký vhodný vrchol (napr. vrchol s najmenšou vzdialenosťou od ostatných, alebo centrálny vrchol). Ďalší prístup je ten, že môžeme vrcholy v každej úrovni nakresliť na sústredné kružnice vzhľadom na nejaký zvolený stred grafu (Obrázok 3.2).



Obrázok 3.2: Nakreslenie nezakoreneného stromu pomocou sústredných kružníc

3.2 Kreslenie planárnych grafov

Definície týkajúce sa planárnych grafov a algoritmy na ich kreslenie nájdeme v [10]. Ak máme neohraničenú plochu a chceme do nej nakresliť planárny graf, tak počet všetkých rôznych nakreslení nie je konečný. Tieto nakreslenia sa ale dajú rozdeliť do tried ekvivalencie, podľa toho aké cykly vrcholov sú v planárnom nakreslení. Týchto tried ekvivalenie je preto konečný počet, ale môže ich byť až exponenciálne veľa. Jedna takáto trieda ekvivalencie sa nazýva planárne uloženie.

Definícia 3.2.1 *Dve reprezentácie D_1 a D_2 planárneho grafu G, patria do tej istej triedy ekvivalencie planárneho uloženia (planar embedding) práve vtedy, keď sú splnené nasledujúce podmienky :*

- *Jednoduché cykly vrcholov v G, ktoré viažu oblasti v D_1 sú rovnaké cykly vrcholov, aké viažu oblasti v D_2.*
- *Vonkajšia oblasť v D_1 je viazaná rovnakým cyklom vrcholov z G ako v D_2.*

Planarita grafu je veľmi dôležitá vlastnosť. Na zistenie či je graf planárny potrebujeme zadefinovať pojem subdivízia grafu. Je to graf, ktorý vznikne tak, že niektoré hrany pôvodného grafu sú nahradené cestami, na ktorých je každý vrchol stupňa dva.

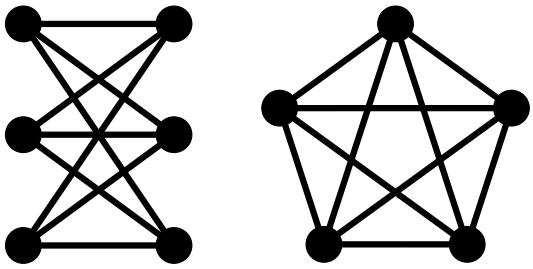
Definícia 3.2.2 *Subdivízia grafu $G = (V, E)$ je graf $G' = (V', E')$, ktorý vznikne z G postupnosťou operácií split, ktorá pozostáva z vloženia nového vrchola u a nahradením hrany $e = (v_1, v_2)$ dvoma hranami $e_1 = (v_1, u)$ a $e_2 = (u, v_2)$.*

Teraz už môžeme planárne grafy presne určiť pomocou nasledujúcej vety.

Veta 3.2.3 (Kuratowski) *Graf G je planárny práve vtedy, keď neobsahuje subdivíziu K_5 ani $K_{3,3}$.*

Dôkaz: Pôvodný Kuratowského dôkaz nájdeme v [17], krátky kombinatorický dôkaz je v [18]. ■

Testovanie planarity je veľmi dôležité pri kreslení grafov a využíva sa tak pri kreslení planárnych grafov ako aj v niektorých metódach kreslenia všeobecných grafov. Prvý algoritmus na testovanie či je graf planárny vymysleli Auslander a Parter (1961) a Goldstein (1963). Hopcroft a Tarjan zlepšili tento výsledok na algoritmus, ktorý beží v lineárnom čase (1974). Ďalší lineárny algoritmus na testovanie planarity vymysleli Lempel, Even a Cederbaum (1967) a Booth a Lueker (1976).



Obrázok 3.3: Dva základné neplanárne grafy : $K_{3,3}$ a K_5

Veta 3.2.4 *Testovanie planarity a zostrojenie planárnej reprezentácie sa dá spraviť v čase* $O(n)$.

Dôkaz: Algoritmus Hopcrofta a Tarjana z roku 1974 nájdeme v [19]. ■

Pri kreslení grafov je často dobré vedieť, aké veľké bude nakreslenie grafu vzhľadom na počet jeho vrcholov. Je dôležité aby sme vedeli, akú veľkú plochu máme pre daný graf rezervovať, alebo pri ohraničenej ploche zistiť, či sa do nej dá daný graf vôbec nakresliť.

Veta 3.2.5 *Každý n-vrcholový planárny graf má straight-line planárny nákres (sieť, mriežka), ktorý je ohraničený obdĺžnikom s rozmermi* $O(n) * O(n)$.

Dôkaz: Konštrukčný dôkaz nájdeme v [5], je tam algoritmus pracujúci v lineárnom čase a kresliaci do obdĺžnika velkosti (2n-4)x(n-2). ■

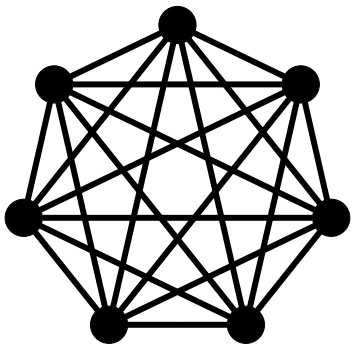
Algoritmy na kreslenie planárnych grafov majú zvyčajne takúto štruktúru :

1. Test planarity – existujú lineárne algoritmy, ale všetky sú zložité na implementáciu a pochopenie
2. Zostrojenie planárnej reprezentácie
3. Použitie planárnej reprezentácie na nakreslenie grafu podľa určitých grafických štandardov

Existujú algoritmy, ktoré spravia kroky 1 a 2 v lineárnom čase (pozri Veta 3.2.4).

3.3 Kreslenie kompletných grafov

Kompletný graf sa obyčajne kreslí tak, že všetky vrcholy ležia na kružnici a sú rozvrhnuté rovnomerne (Obrázok 3.4). Takýto nákres je maximálne symetrický. Pri kreslení kompletného grafu je veľmi ťažké predchádzať kríženiu hrán, keďže už K_5 nie je planárny (Veta 3.2.3) a s každým ďalším vrcholom pribúda viac a viac hrán.



Obrázok 3.4: Nakreslenie K_7 na kružnici

3.4 Všeobecné metódy a ich porovnanie

Na kreslenie všeobecných grafov sa používajú rôzne metódy. Každá metóda je vhodná na kreslenie iných grafov a má iné vlastnosti. Tak isto je pri výbere metódy dôležité, aké kritériá má spĺnať výsledný graf, a aký typ kreslenia je zvolený. Okrem metód popísaných nižšie existuje aj veľa ďalších, napr. metóda založená na genetických algoritmoch[1]. Najpoužívanejšie metódy sú :

1. Planárne metódy[2] — ak graf nie je planárny, tak ho splanárnime (napr. pridaním umelých vrcholov, odobratím hrán, rozdelením vrcholov) a použijeme nejaký algoritmus na kreslenie planárnych grafov, potom spätne vrátime graf do pôvodného stavu

2. Orientované metódy — namiesto každej hrany spravíme dve orientované hrany a použijeme algoritmus na kreslenie orientovaných grafov

[1] Takýto algoritmus a jeho porovnanie so spring algoritmom nájdeme v [14].
[2] Rôznymi metódami splanárňovania grafov sa zaoberá [15].

3. Force-direct metódy — zadefinujeme nejaký systém síl na hranách a vrcholoch a nájdeme stav grafu, v ktorom má graf najmenšiu energetickú hodnotu síl (riešenie diferenciálnych rovníc alebo simulovanie vývoja systému), tieto metódy sa využívajú väčšinou na zobrazovanie stromových sietí, ako napr. počítačové a sociálne siete

Planárna metóda má množstvo výhod, je rýchla, aplikovatelná na straight-line, polyline, ortogonal drawing, je podporovaná teoretickými výsledkami z planar drawingu. Jej nevýhodou je, že je ťažko implementovatelná, a ťažko rozšíritelná na 3D. Oproti tomu je force-direct metóda ľahko implementovatelná, heuristické zlepšenia sa ľahko pridávajú, môže byť rozšíritelná na 3D, pracuje dobre pre malé grafy, ale je pomalá, existuje málo teoretických výsledkov, ťažko ju rozšíriť na orthogonal a polyline drawing. Force-direct metódou nakreslené grafy sú väčšinou symetrické, grafy nakreslené planárnou metódou zasa mávajú menej krížení hrán[3].

Jedna z možností ako splanárniť daný graf, je odobrať čo najmenší počet hrán tak, aby výsledný graf bol planárny. Potom sa pridá hrana tak, aby vzniklo čo najmenej krížení. Tam kde vzniknú kríženia, sa vložia umelé vrcholy a znovu sa môže pridať ďalšia hrana, keďže takto vzniknutý graf je znovu planárny. Takýto postup ale nemusí viesť k nakresleniu, ktoré má minimálny počet krížení. Dôležitý krok je pridanie hrany do plananárneho grafu tak, aby vzniklo čo najmenej krížení, dá sa to spraviť v lineárnom čase :

Veta 3.4.1 *Vloženie hrany (oba koncové vrcholy patria do grafu) do daného planárneho grafu tak, aby vzniklo čo najmenej krížení sa dá v lineárnom čase.*

Dôkaz: Konceptuálne jednoduchý lineárny algoritmus založený na SPQR-stromoch môžeme nájsť v [12]. ■

[3]Porovnanie týchto prístupov kreslenia grafov nájdeme v [11].

Kapitola 4

Dokresľovanie hrán a vrcholov do nakresleného grafu

V tejto kapitole sa budeme venovať problému dokresľovania hrán a vrcholov do už nakresleného grafu. To znamená, že máme nakreslený graf, máme už zafixované kde približne je ktorý vrchol a chceme pridať nové vrcholy alebo nové spojenia medzi vrcholmi ale nechceme pritom hýbať s už existujúcimi vrcholmi (aby sme nestratili prehľad v nakreslenom grafe). Najprv ukážeme že dokresľovanie viacerých hrán naraz je efektívnejšie ako dokresľovanie po jednej hrane z hľadiska počtu krížení hrán. Potom uvedieme dve nové heuristiky na dokresľovanie vrcholov do nakresleného grafu a tieto heuristiky porovnáme, či už z hľadiska časovej a implementačnej zložitosti ako aj z pohľadu efektívnosti (v tomto prípade počtu krížení hrán).

Ak máme daný nakreslený graf G a chceme doňho dokresliť hrany alebo vrcholy, tak vo všeobecnosti je lepšie dokresľovať vrcholy ako hrany, pretože ak dokresľujeme hranu, ktorá je incidentná s novým nedokresleným vrcholom, tak nám stačí určiť polohu tohoto vrchola (alebo oboch vrcholov), a umiestnenie hrany je už určené. Ak dokresľujeme vrchol, tak môžeme prihliadať na viac vecí (ak má veľa susedných vrcholov, tak môžeme prihliadať na všetkých, ale ak dokresľujeme hranu, tak prihliadame najviac na jeden nakreslený vrchol).

4.1 Dokresľovanie hrán do nakresleného grafu

Veta 4.1.1 *Ak k danému nakreslenému grafu pridávame 2 hrany (koncové vrcholy týchto hrán nemusia patriť do grafu), tak vzhľadom na počet krížení nie je efektívnejšie pridáť ich postupne.*

Dôkaz: Problém rozdelíme na viacero situácií podľa toho, ktoré koncové vrcholy dopĺňaných hrán patria do grafu.

1. Koncové vrcholy oboch hrán sú v grafe :
   Tu je jedno či pridávame obe hrany naraz, alebo ich pridávame postupne, nakreslenie týchto hrán je už určené.

2. Jedna hrana má oba koncové vrcholy v grafe, druhá má len jeden :
   Tu je lepšie najprv dokresliť hranu s koncovými vrcholmi v grafe a potom tú s jedným koncovým vrcholom v grafe, pretože pri opačnom postupe by sa mohli tieto dve hrany križovať (Obrázok 4.1 a)).

3. Jedna má oba koncové vrcholy v grafe, druhá nemá ani jeden v grafe :
   Tu nepomôže dokresľovať hrany naraz, jednu dokreslíme do grafu, druhú nakreslíme mimo grafu.

4. Obe hrany majú v grafe jeden koncový vrchol :
   Ak mimografový vrchol týchto hrán je ten istý, tak je veľmi dôležité určiť jeho polohu (Obrázok 4.1 b)). Ak to nie je ten istý vrchol, tak hrany dokreslíme do grafu tak, aby sa navzájom nepretínali.

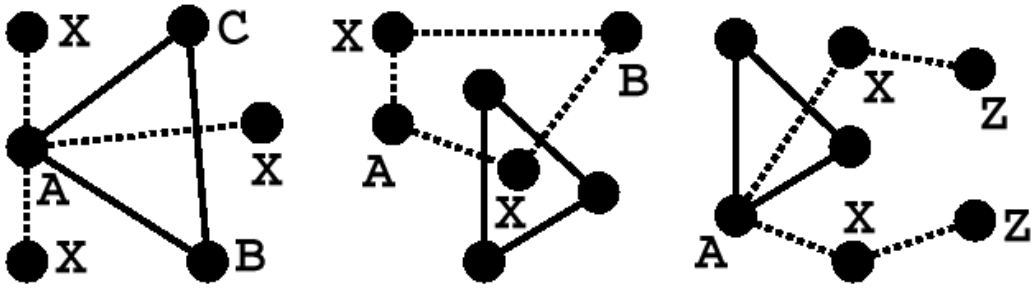
5. Jedna hrana má jeden koncový vrchol v grafe, druhá nemá ani jeden :
   Ak majú tieto 2 hrany spoločný vrchol, tak je lepšie najprv pridať hranu s jedným koncovým vrcholom v grafe, pretože ak by sme dokreslili najprv hranu bez koncových vrcholov v grafe, tak by mohlo vzniknúť viacero krížení (Obrázok 4.1 c)).

6. Žiaden koncový vrchol hrán nie je v grafe :
   Tu nepomôže dokresľovať hrany naraz, hrany nakreslíme mimo grafu, je jedno v akom poradí.

■

4.2 Dokresľovanie vrcholov do nakresleného grafu

Máme daný graf G a vrcholy, ktoré chceme do tohoto grafu dokresliť. Snažíme sa minimalizovať počet križení hrán v grafe. Prvý problém je, či máme dokresľovať vrcholy po jednom alebo viacero naraz, už dokresľovanie jedného vrcholu do nakresleného grafu je zložité. Ďalej treba určiť, v akom poradí sa majú dané vrcholy do grafu dokresľovať. Tu sa môže prihliadať tiež na viacero kritérií, ako napríklad stupeň vrcholu v celom grafe, stupeň vrcholu



Obrázok 4.1: Pridávanie hrán do nakresleného grafu : a) pridávame hrany AX, BC, lepšie je pridať najprv BC b) pridávame hrany AX, BX, treba rozumne nájsť polohu vrcholu X c) pridávame hrany XZ, AX, lepšie je pridať najprv AX

v zatiaľ nakreslenom grafe, počet susedov ktorých má daný vrchol medzi ešte nedokreslenými vrcholmi a ďalšie. Na dokreslenie vybratého vrcholu do nakresleného grafu navrhneme dve nové heuristiky, ktoré neskôr porovnáme.

4.2.1 Heuristika 1

Táto heuristika obsahuje metriku, na základe ktorej určí, kam je vhodné umiestniť daný vrchol. Skladá sa z dvoch častí, v prvej časti určí, v ktorej oblasti grafu umiestnime vrchol a v druhej časti zistí kde v rámci tejto oblasti bude najlepšie daný vrchol umiestniť. Najprv si graf upravíme tak, že kríženia hrán nahradíme umelými (dummy) vrcholmi, a budeme s nimi pracovať ako s normálnymi. Keď už dokresľovaný vrchol umiestnime do grafu, tak môžeme tieto umelé vrcholy zasa odstrániť.

Definícia 4.2.1 *Máme graf G, a jeho zakreslenie do roviny D. Množina bodov $R^2 - D$ je neohraničená a jej súvislé časti sa nazývajú oblasti (faces) z D. Keďže D je ohraničená, tak presne jedna oblasť z D je neohraničená. Túto oblasť nazývame vonkajšia oblasť z D.*

Tým, že sme vložili do grafu umelé vrcholy, tak každá vnútorná oblasť je ohraničená vrcholmi z grafu.

Veta 4.2.2 *Počet oblastí grafu G s n vrcholmi je $O(n^4)$. Ak je graf planárny, tak počet týchto oblastí je $O(n^2)$.*

Dôkaz: Ku každému vrcholu (pôvodnému aj umelému) v grafe zistíme počet oblastí s ktorými susedí a súčet počtu týchto oblastí, ktorý je určite väčší ako

celkový počet oblastí, bude $O(n^4)$. Pôvodných vrcholov je n. Každý z nich susedí maximálne s $n-1$ oblasťami, pretože z neho vychádza maximálne $n-1$ hrán. Súčet počtu susedných oblastí pôvodných vrcholov je teda $O(n^2)$. Ak je graf planárny, tak nepridávame žiadne umelé vrcholy, teda aj počet oblastí je $O(n^2)$. Počet hrán v grafe je $O(n^2)$. Umelý vrchol pridávame na miesto, kde sa krížia hrany, teda ich počet je $O(n^4)$. Ak je umelý vrchol na krížení dvoch hrán, tak susedí so štyrmi oblasťami. Ak je umelý vrchol na krížení m hrán, tak susedí s $2 * m$ oblasťami, ale tento umelý vrchol má v sebe $\frac{m*(m-1)}{2}$ umelých vrcholov (každá dvojica hrán v tomto vrchole). Teda najviac oblastí sa pridá, ak sa každé dve hrany krížia na inom mieste. Vtedy je ale ich počet pre každý umelý vrchol ohraničený konštantou 4, teda ich celkový počet je $O(n^4)$. ■

K metrike, ktorú využíva táto heuristika ešte potrebujeme zadefinovať pojem vrcholová vzdialenosť vrchola v od oblasti U. Je to dĺžka najkratšej cesty z vrcholu v k nejakému vrcholu, ktorý susedí s oblasťou U.

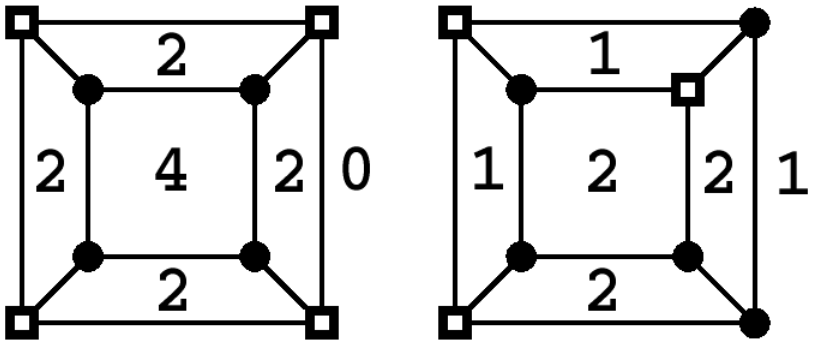
Definícia 4.2.3 *Máme graf G, a jeho zakreslenie do roviny D. Vrcholová vzdialenosť vrchola u_0 od oblasti U je najmenší počet vrcholov $u_1, u_2, \ldots, u_k$ takých, že $(u_{i-1}, u_i) \in E(G)\ \forall i \in \{1..k\}$ a u_k susedí s oblasťou U, označujeme $D_v(U, u_0) = k$.*

Máme nakreslený graf G a chceme k nemu pridať vrchol v. Susedov vrchola v v grafe G označíme $v_1, v_2, \ldots, v_s$. Popis heuristiky :

1. časť :
   Pre každú oblasť U_i určíme nejakú číselnú hodnotu (túto hodnotu označíme $M(U_i)$) podľa zvolenej metriky a potom vyberieme oblasť U_n s najmenšou hodnotou, ak je takých viac, tak pre vopred zvolenú konštantu r vyberieme r náhodných z nich (Obrázok 4.2). Metriku definujeme ako súčet vrcholových vzdialeností všetkých susedných vrcholov vrchola v od oblasti U_i.

   $M(U) = \sum_{i=1}^{s} D_v(U, v_i)$
   $M(U_n) \leq M(U_i)\ \forall i \in \{1..k\}$, kde k je počet oblastí zakreslenia

2. časť :
   Pre vopred zvolenú konštantu k postupne dokresľovaný vrchol umiestnime na k náhodných pozícií vnútri vybratej oblasti a vyberieme umiestnenie, pri ktorom vzniklo najmenej krížení. Ak je takých umiestnení viac, tak vyberieme umiestnenie s najväčšou vzdialenosťou od najbližšieho vrcholu pôvodného grafu.



Obrázok 4.2: Určenie číselných hodnôt pre každú oblasť grafu

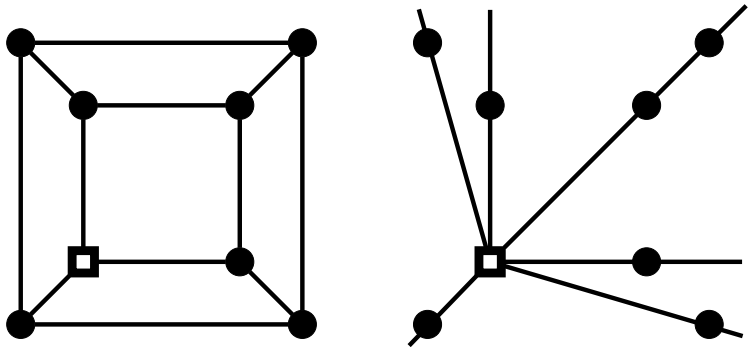
4.2.2 Heuristika 2 (lúčová)

Lúčová heuristika je založená na tom, že pre niekoľkých susedov v_i pridávaného vrchola v, rozdelíme nakreslenie grafu na niekoľko oblastí, pričom pre každú oblasť budeme mať lúčové číslo. Toto číslo nám bude udávať, že keď vrchol v vložíme do tejto oblasti, tak hrana (v, v_i) pridá maximálne takýto počet krížení.
V prvom kroku nakreslený graf rozdelíme na veľké oblasti (ich počet bude menší ako počet vrcholov grafu) a v ďalšom kroku tieto oblasti rozdelíme na niekoľko menších, pričom získame aj lúčové čísla týchto oblastí. Ak máme nakreslenie grafu takto rozdelené pre niekoľkých susedov vrchola v, tak stačí určiť oblasť, ktorá po prieniku oblastí od každého takéhoto suseda má malý súčet lúčových čísel. Vrámci tejto oblasti potom môžeme umiestňovať vrchol tak ako v Heuristike 1 (na k náhodných pozícií) a potom vyberieme najlepšie umiestnenie vzhľadom na počet krížení hrán.

Popíšeme lúčovú heuristiku pre jedného suseda (u) vrchola v, pre ostatných zvolených susedov je postup rovnaký :

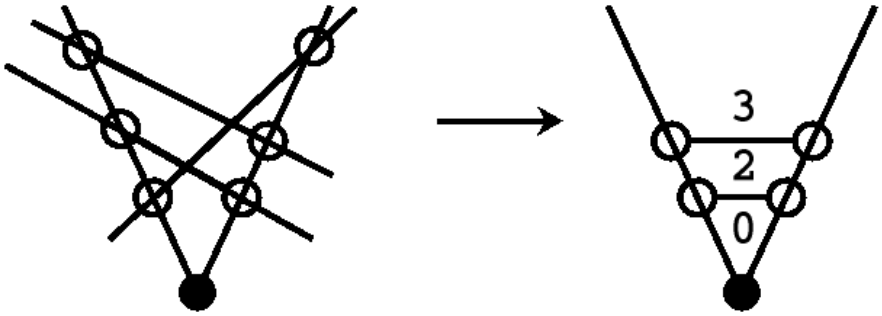
1. časť :
   Z vrchola u vedieme polpriamky (lúče) ku každému vrcholu v grafe. Priestor medzi susednými dvoma polpriamkami bude veľká oblasť (Obrázok 4.3).

2. časť :
   Každú veľkú oblasť rozdelíme na menšie takýmto spôsobom. Ak nejaká hrana zasahuje do tejto oblasti, tak z nej spravíme úsečku s koncovými bodmi na lúčoch ohraničujúcich túto oblasť. Tieto body si označíme ako hraničné body. Každá hrana zasahujúca do oblasti má dva hraničné



Obrázok 4.3: Vytvorenie veľkých oblastí z pôvodného grafu

body, z nich vyberieme ten, ktorý je bližšie k vrcholu u a označíme ho ako dôležitý bod. Potom spravíme pre každý dôležitý bod úsečku, ktorá spája tento bod a bod, ktorý je rovnako vzdialený od vrcholu u a patrí druhému hraničnému lúču. Menšie oblasti budú ohraničené týmito novými úsečkami a hraničnými lúčmi. Lúčové číslo bude rovné počtu nových úsečiek, ktoré sú bližšie k vrcholu u ako daná oblasť (Obrázok 4.4). Do lúčového čísla zarátavame každú úsečku, aj keď ich nakreslenie môže byť totožné.



Obrázok 4.4: Vytvorenie menších oblastí a ich lúčové čísla

3. časť :
   Teraz musíme určiť nejaký vhodný prienik medzi vzniknutými oblasťami. Ak dokresľovaný vrchol susedí len s dvoma vrcholmi, tak môžeme

pre každú dvojicu menších oblastí zistiť, či majú spoločný prienik a súčet ich lúčových čísel priradíme tomuto prieniku. Potom už len stačí vybrať prieniky oblastí s najmenším takýmto súčtom a umiestnime vrchol sem. Ak dokreslovaný vrchol susedí s viacerými vrcholmi, tak môžeme hľadať spoločný prienik viacerých oblastí (pre každého suseda jedna). Ak je ale počet jeho susedov veľký, tak hľadanie najlepšieho prieniku môže byť až exponencionálne. Aby sa tomu predišlo, tak pre vopred stanovené pevné t vyberieme t susedov a nájdeme najlepší prienik pre týchto susedov (možu to byť napríklad susedia s najväčším stupňom).

4. časť:
   Ak už máme prienik vybratý, tak vrchol doň umiestnime rovnakým spôsobom ako v Heuristike 1. Pre vopred zvolenú konštantu k postupne vrchol umiestnime na k náhodných pozícií vnútri prieniku a vyberieme umiestnenie, pri ktorom vzniklo najmenej krížení. Ak je takých umiestnení viac, tak vyberieme umiestnenie s najväčšou vzdialenosťou od najbližšieho vrchola pôvodného grafu.

Veta 4.2.4 *Počet menších oblastí grafu G s n vrcholmi pre ľubovoľný vrchol u je $O(n^3)$.*

Dôkaz: Kedže máme n vrchlov, tak nám môže vzniknúť maximálne $n-1$ veľkých oblastí. Maximálny počet hrán v grafe okrem hrán s vrcholom u je $\frac{(n-1)*(n-2)}{2}$. Toto je aj maximálny počet hrán, ktoré môžu zasahovať do jednej veľkej oblasti, teda aj maximálny počet dôležitých bodov v tejto oblasti. Maximálny počet menších oblastí pre jednu veľkú oblasť je potom $\frac{(n-1)*(n-2)}{2} + 1$ a celkový maximálny počet menších oblastí v celom grafe je $(\frac{(n-1)*(n-2)}{2} + 1) * (n-1)$, čo je $O(n^3)$. ■

4.2.3 Porovnanie uvedených heuristík

Najprv porovnáme tieto algoritmy z hľadiska časovej zložitosti. Oba algoritmy majú spoločné, že ich časová zložitosť veľmi záleží na tom, akú štruktúru má vstupný graf a aké vrcholy do grafu pridávame. Pre každú heuristiku preto uvedieme horné ohraničenie jej časovej zložitosti.

Časová zložitosť Heuristiky 1 :
Najprv musíme v danom grafe nahradiť kríženia hrán umelými vrcholmi. Zistenie či sa dve hrany pretínajú je $O(1)$. Počet hrán v grafe je $O(n^2)$, preto zisťovanie, či sa križuje niektorá dvojica hrán je $O(n^4)$. Toto je aj horné

ohraničenie pre počet krížení. Pre každú hranu si zároveň pamätáme, ktoré priesečníky na nej ležia. Teraz každú hranu nahradíme novými hranami, ktoré ležia medzi týmito priesečníkmi a koncovými vrcholmi hrany. Hrán je $O(n^2)$, maximálny počet priesečníkov ležiacich na jednej hrane je $O(n^2)$, teda táto transformácia hrán je $O(n^4)$. Máme teraz nový graf, ktorý má $O(n^4)$ vrcholov a $O(n^4)$ hrán.

Teraz musíme nájsť jednotlivé oblasti v tomto upravenom grafe. Každá hrana susedí presne s dvoma oblasťami, teda každú hranu nahradíme dvoma navzájom opačne orientovanými hranami. Oblasti budeme konštruovať tak, že najprv si vyberieme ľubovolnú hranu a ďalšie hrany budeme postupne pridávať. Ďalšiu hranu pridáme tú, ktorá zviera s poslednou pridanou hranou najmenší uhol v smere hodinových ručičiek. Hrany budeme pridávať dovtedy, kým medzi koncovými vrcholmi týchto hrán nevznikne cyklus. Tento cyklus nám ohraničuje nejakú oblasť v grafe, odstránime ho z grafu (iba orientované hrany v jednom smere, po ktorých sme prechádzali) a pokračujeme v pridávaní hrán, dokým nie sú z grafu odstránené všetky hrany. Takto postupne skonštruujeme všetky oblasti. Po každej hrane prechádzame 2-krát, raz v každom smere (hrán je $O(n^4)$), keď vyberáme ďalšiu hranu, tak pozeráme $O(n^4)$ kandidátov, takisto aj keď kontrolujeme, či už vznikol cyklus, tak pozeráme $O(n^4)$ vrcholov. Toto hľadanie oblastí je teda $O(n^8)$.

Pre každú oblasť v grafe musíme zistiť jej vrcholovú vzdialenosť od susedov dokresľovaného vrchola. Hodilo by sa nám poznať vzdialenosť všetkých susedov dokresľovaného vrchola od všetkých vrcholov v grafe. Vzdialenosť vrchola od všetkých ostatných vrcholov v grafe môžeme zistiť Dijkstrovym algoritmom, ten má časovú zložitosť $O(|V|^2)$, čo je pri našom počte vrcholov $O(n^8)$. Kedže toto potrebujeme pre všetkých susedov (tých je $O(n)$), tak je to celkovo $O(n^9)$. Súčet vrcholových vzdialeností susedov od oblasti zistíme tak, že pre každého suseda určíme minimum z jeho vzdialeností cez každý vrchol ohraničujúci túto oblasť. Počet oblastí je $O(n^4)$, počet vrcholov ohraničujúcich jednu oblasť je $O(n^4)$, počet susedov dokresľovaného vrchola je $O(n)$, teda celkovo je to $O(n^9)$.

Keď už pre každú oblasť (je ich $O(n^4)$) máme súčet jej vzdialeností od susedov dokresľovaného vrchola, tak minimum z týchto súčtov nájdeme v čase $O(n^4)$. Teraz pre r oblastí, ktoré majú tento minimálny súčet skúsime hľadať vhodné umiestnenie dokresľovaného vrchola. Budeme skúšať k náhodných umiestnení. Pre každé takéto umiestnenie potrebujeme zistiť koľko nových krížení vznikne, ako aj vzdialenosť od najbližšieho vrchola v pôvodnom grafe. Pre každú hranu, ktorá vznikne pri dokreslení tohoto nového vrchola ($O(n)$) zistíme či pretína nejakú pôvodnú hranu (tých je $O(n^2)$), teda zistenie počtu krížení je $O(n^3)$. Zistenie vzdialenosti od najbližšieho vrcholu pôvodného grafu je $O(n)$. Teda zistenie umiestnenia vrchola je celkovo $O(r * k) * O(n^3)$,

čo je $O(n^3)$.

Časová zložitosť Heuristiky 2 :
Najprv musíme skonštruovať lúče. Skonštruovanie lúčov pre jedného suseda u dokresľovaného vrchola je $O(n)$, pretože lúč je vedený zo suseda cez každý vrchol grafu, lúčov je $O(n)$. Potom lúče usporiadame podľa toho, aký uhol zvierajú s osou x v smere hodinových ručičiek. Usporiadanie $O(n)$ prvkov sa dá spraviť $O(n \log n)$ napríklad quick-sortom. Tu zároveň odstránime duplicitné lúče. Teraz môžeme vytvoriť veľké oblasti, tie budú ohraničené dvojicou susedných lúčov (aj posledný s prvým), pokiaľ tieto lúče zvierajú uhol menší ako π v smere hodinových ručičiek. Vytvorenie veľkých oblastí je teda $O(n \log n)$.
Máme vytvorené veľké oblasti, teraz vytvoríme malé. Najprv určíme dôležité body. Pre každú hranu zistíme či križuje oba hraničné lúče, ak áno, tak ten priesečník ktorý je bližšie k vrcholu u pridáme do dôležitých bodov tejto oblasti. Ak už máme všetky dôležité body, tak ich usporiadame podľa vzdialenosti od vrchola u vzostupne. Počet dôležitých bodov je $O(n^2)$, ich usporiadanie quick-sortom je teda $O(n^2 \log(n^2))$, čo je $O(n^3)$. Teraz už môžeme vytvárať menšie oblasti. Prvá takáto oblasť bude ohraničená bodmi u, prvým dôležitým bodom a jeho obrazom na druhom lúči (oba sú od u rovnako vzdialené), lúčové číslo tejto oblasti bude 0. Ďalšia oblasť bude ohraničená prvým dôležitým bodom, druhým dôležitým bodom a ich obrazmi. Takto postupne konštruujeme oblasti, je ich $O(n^2)$, konštrukcia jedného je $O(1)$, teda celkovo vytvorenie menších oblastí z jednej veľkej je $O(n^3)$. Keďže máme $O(n)$ veľkých oblastí, tak vytvorenie menších oblastí pre jedného suseda dokresľovaného vrchola je $O(n^4)$, pre t susedov $O(n^4)$.
Pre jedného suseda máme $O(n^3)$ menších oblastí, ak chceme pre dvoch susedov dokresľovaného vrchola zistiť ktoré oblasti majú prienik, tak musíme vyskúšať $O(n^3) * O(n^3) = O(n^6)$ dvojíc. Zistiť prienik dvoch takýchto oblastí sa dá v čase $O(1)$, pretože obe sú ohraničené maximálne štyrmi vrcholmi. Ak chceme zistiť spoločné prieniky od t susedov, tak musíme uvažovať $O(n^{3t})$ t-tic.
Ak máme vybraté prieniky, tak vrchol v nich umiestňujeme presne tak ako v Heuristike 1, čo je $O(r * k) * O(n^3) = O(n^3)$.

Obidve heuristiky majú veľkú časovú zložitosť, Heuristika 1 $O(n^9)$, kde najnáročnejšie časti sú hľadanie oblastí ($O(n^8)$) a hľadanie najkratších ciest medzi vrcholmi ($O(n^9)$). Tieto operácie sú preto tak časovo náročné, pretože sú aplikované na graf, ktorý má $O(n^4)$ vrcholov a $O(n^4)$ hrán. Naproti tomu Heuristika 2 aby mala zmysel, tak musí byť $t \geq 2$, pričom ak $t = 2$, tak časová zložitosť je $O(n^6)$. Časovo najnáročnejšia operácia je hľadanie

prienikov medzi každými dvoma menšími oblasťami. Tieto zložitosti sú pri pridávaní jedného vrchola do grafu, ak pridávame viac vrcholov, tak sa dajú upraviť niektoré časti heuristík tak, aby sa zbytočne nerátali tie isté veci 2-krát a tým znížiť ich časovú zložitosť.

Ak má každý z vrcholov kresleného grafu maximálne konštantný počet susedov p, tak časové zložitosti týchto heuristík sú nasledovné :
Heuristika 1 :
Počet hrán v grafe je $O(n)$, počet krížení $O(n^2)$, teda transformácia na planárny graf je $O(n^2)$, vytvorený graf má $O(n^2)$ vrcholov a $O(n^2)$ hrán.
Pri konštruovaní oblastí prejdeme po každej hrane 2-krát ($O(n^2)$), pri pridávaní ďalšej hrany a hľadaní cyklu kontrolujeme $O(n^2)$ hrán/vrcholov, teda celkovo $O(n^4)$.
Pri zisťovaní vrcholovej vzdialenosti zisťujeme vzdialenosť pre $O(p)$ vrcholov od každého vrcholu v grafe, čo je Dijkstrovym algoritmom $O(p) * O(n^2) * O(n^2) = O(n^4)$. Súčet vrcholových vzdialeností oblastí od susedov dokresľovaného vrcholu je $O(n^4)$.
Kedže máme $O(n^2)$ oblastí, tak nájdenie minimálneho súčtu je $O(n^2)$, pri umiestňovaní vrchola do oblasti je zistenie počtu krížení $O(p) * O(n * p) = O(n)$, zistenie vzdialenosti od najbližšieho vrchola je $O(n)$.
Celková zložitosť heuristiky je teda $O(n^4)$, kde znovu najnáročnejšia operácia na čas je zisťovanie vrcholovej vzdialenosti.
Heuristika 2 :
Vytvorenie veľkých oblastí je $O(n \log n)$), tam sa nič nezmení. Počet dôležitých bodov pre jednu veľkú oblasť je $O(n)$, ich usporiadanie je $O(n \log n)$. Vytvorenie malých oblastí z jednej veľkej je $O(n \log n)$, pre $O(n)$ veľkých oblastí to je $O(n^2 \log n)$.
Pre jedného suseda máme $O(n^2)$ malých oblastí, zisťovanie prieniku po dvojiciach pre dvoch susedov je $O(n^2) * O(n^2) = O(n^4)$. Zisťovanie spoločných prienikov pre t susedov je $O(n^{2t})$.
Zisťovanie umiestnenia vrcholu vrámci vybratého prieniku je podobne ako v Heuristike 1 $O(n)$.
Celková časová zložitosť heuristiky je $O(n^4)$ pre $t = 2$, kde opäť časovo najnáročnejšia operácia je zisťovanie prienikov medzi oblasťami od dvoch susedov dokresľovaného vrchola.

Implementačne oveľa ťažšie bolo naprogramovať Heuristiku 1, hlavne pretransformovanie pôvodného grafu na graf s umelými vrcholmi a hľadanie oblastí v tomto grafe.
Porovnaniu týchto dvoch heuristík z hľadiska počtu krížení sa venujeme v **Kapitole 6**, strana 42.

Kapitola 5

Kreslenie dvojfarebných grafov

V tejto kapitole najprv zadefinujeme čo je to dvojfarebný graf a navhneme metriku, ktorá bude slúžiť na určenie, ako dobre je nakreslený daný dvojfarebný graf. Metrika bude postavená na krížení hrán a na váhach týchto krížení. Žiadna dostupná literatúra sa kreslením dvojfarebných grafov nezaoberala. Uvedieme algoritmy, ktoré sme implementovali na kreslenie tejto množiny grafov, či už to budú algoritmy, ktoré nakreslia graf priamo, alebo algoritmy, ktoré sú založené na pridávaní vrcholov do existujúceho nakresleného grafu pomocou heuristík uvedených v predchádzajúcej kapitole, alebo pomocou inej metódy. Všetky tieto algoritmy boli naprogramované nad knižnicou algoritmov `LEDA 5.0`, verzia pre `Mandrake 10.0`, kompilátor `g++ 3.2.2`.

Dvojfarebný graf je graf, ktorého každý vrchol patrí do jednej z dvoch množín vrcholov. Prvá množina obsahuje primárne vrcholy, druhá sekundárne. Nakreslenie dvojfarebného grafu môže byť užitočné, ak chceme zobraziť objekty a spojenia medzi týmito objektami, pričom objekty patria do dvoch skupín. Raz nás môže zaujímať nakreslenie jednotlivých skupín grafu, inokedy zasa celkové nakreslenie grafu, preto by malo byť nakreslenie grafu dobré v rámci jednotlivých skupín ako aj v rámci celého grafu.

Definícia 5.0.1 *Dvojfarebný graf G nazveme taký graf, ktorého vrcholy patria do dvoch disjunktných množín, t.j.* $(V(G) = V_1 \cup V_2) \wedge (V_1 \cap V_2 = \epsilon)$. *Vrcholy z množiny V_1 nazývame primárne, vrcholy z množiny V_2 sekundárne.*

Na určenie, či je nakreslenie dvojfarebného grafu dobré, potrebujeme zadefinovať metriku, podľa ktorej by sme to mohli zmerať. Táto metrika je založená na počte krížení hrán, kde každé kríženie má nejakú váhu, podľa toho, na akých hranách leží. Hrany rozdelíme do troch disjunktných množín, podľa

toho z akých množín sú ich koncové vrcholy. Za každé kríženie na hrane priradíme tejto hrane penalizáciu podľa toho, do ktorej množiny hrán patrí.

Definícia 5.0.2 *Nech G je dvojfarebný graf, funkciu m_{2fh} nazveme hranovou metrikou na grafe G, ak $\forall e \in E(G)$ s koncovými vrcholmi v_1, v_2 platí :*
$m_{2fh}(e) = 3$ *, ak* $v_1, v_2 \in V_1(G)$
$m_{2fh}(e) = 2$ *, ak* $v_1, v_2 \in V_2(G)$
$m_{2fh}(e) = 1$ *, ak* $v_1 \in V_1(G) \Longleftrightarrow v_2 \notin V_1(G)$

Definícia 5.0.3 *Máme daný graf G a jeho zakreslenie do roviny D. Funkciu m_{2fb} nazveme bodovou metrikou na grafe G, ak $\forall v \in D$ platí :*
$m_{2fb}(v) = \sum_{i=1}^{k} m_{2fh}(e_i)$ *, kde $\{e_1, e_2, \ldots, e_k\}$ je množina všetkých hrán, na ktorých bod v leží ($v \in e_i, \forall i = 1..k$).*

Z definície hranovej metriky je jasné, že najviac sa bude prihliadať na hrany, ktoré vedú medzi primárnymi vrcholmi a najmenej na hrany, ktoré vedú medzi vrcholmi z rôznych množín. Teraz zadefinujeme body kríženia grafu, budú to tie body, ktoré vzniknú na krížení hrán.

Definícia 5.0.4 *Máme daný graf G a jeho zakreslenie do roviny D. Množinu bodov $Crp_{G,D} = \{v \in D | \exists e_1, e_2 \in E(G), e_1 \neq e_2, v \in e_1 \wedge v \in e_2 \wedge v \notin V(G)\}$ budeme nazývať body kríženia grafu G pri zakreslení D.*

Teraz už môžeme zadefinovať krížové číslo pre dvojfarebný graf. Toto číslo je určené bodmi kríženia grafu a hranami, na ktorých tieto kríženia ležia. Toto číslo bude zároveň našou metrikou na dobré zakreslenie dvojfarebného grafu.

Definícia 5.0.5 *Krížové číslo dvojfarebného grafu G pri zakreslení D do roviny je $Cr2f_{G,D} = \sum_{i=1}^{k} m_{2fb}(b_i)$, kde $\{b_1, b_2, \ldots, b_k\} = Crp_{G,D}$.*

5.1 Dokresľovanie vrcholov, modifikácia heuristík

Navrhneme modifikácie heuristík uvedených v predchádzajúcej kapitole na dokresľovanie vrcholov do už nakresleného dvojfarebného grafu. Tieto modifikácie by mali slúžiť na to, aby krížové číslo nakresleného dvojfarebného grafu bolo čo najmenšie.

V Heuristike 1 môžeme navrhnúť takéto modifikácie :

- pri rátaní vrcholovej vzdialenosti budeme za každý vrchol z inej skupiny prirátavať penalizáciu

- pri rátaní vrcholovej vzdialenosti bude mať každá hrana inú váhu (napríklad podľa funkcie m_{2fh}, táto funkcia má iba kladné hodnoty, teda môžeme použiť Dijkstrov algoritmus na rátanie vrcholovej vzdialenosti)
- pri dokresľovaní nového vrcholu do grafu budeme uvažovať iba susedov z rovnakej skupiny ako je dokresľovaný vrchol
- pri hľadaní umiestnenia vrchola vo vybratej oblasti môžeme každé kríženie započítavať inak (napríklad podľa funkcie m_{2fb})

V heuristike 2 môžeme navrhnúť takéto modifikácie :

- pri zisťovaní lúčového čísla môžeme každú hranu započítavať inak (napríklad podľa funkcie m_{2fh})
- pri hľadaní vhodného prieniku môžeme uvažovať iba oblasti vytvorené susedmi z rovnakej skupiny ako je dopĺňaný vrchol
- pri hľadaní umiestnenia vrchola vo vybranom prieniku môžeme každé kríženie započítavať inak (napríklad podľa funkcie m_{2fb})

Ďalšia modifikácia heuristík môže byť založená na tom, v akom poradí budeme vrcholy do grafu vkladať. Pre každý vkladaný vrchol môžeme určiť jeho dôležitosť podľa toho, s koľkými vrcholmi z primárnej a sekundárnej skupiny susedí a pomocou toho určiť poradie vrcholov pri vkladaní do grafu.

Otázne ale je, ktoré z týchto modifikácií naozaj pomôžu zmenšiť krížové číslo dvojfarebného grafu. Pri hľadaní umiestnenia vrchola vo vybratej oblasti (prieniku) nám modifikácia naozaj pomôže, pretože vychádza z definície dvojfarebného krížového čísla. Tak isto pomôže aj modifikácia pri rátaní lúčového čísla, tá je tiež založená na definícii krížového čísla dvojfarebného grafu, ostatné modifikácie treba experimentálne overiť. Čím viac budeme pri dokresľovaní vrcholu znevýhodňovať kríženie medzi vrcholmi z tej istej skupiny, tým bude výsledné nakreslenie horšie ale zlepší sa skupinové nakreslenie.

5.2 Kreslenie dvojfarebných grafov existujúcimi algoritmami

Tu uvedieme niektoré základné algoritmy, ktoré sme použili na kreslenie dvojfarebných grafov. Vstupom algoritmov je graf G, množina jeho vrcholov, pre každý vrchol jeho typ (či je primárny alebo sekundárny), množina hrán nad týmito vrcholmi, súradnice ľavého horného a pravého dolného bodu plochy, do ktorej má byť daný graf nakreslený.

- **Náhodné nakreslenie -** každý vrchol umiestnime na náhodnú pozíciu tak, aby bol vnútri plochy určenej na nakreslenie grafu
- **Kruhové nakreslenie -** vrcholy sú rozmiestnené pravidelne po kružnici okolo stredu plochy, do ktorej sa má graf nakresliť. Sú tri varianty tohoto nakreslenia, podľa toho v akom poradí sú vrcholy na kružnici rozmiestnené (náhodné poradie vrcholov, najprv vrcholy z primárnej skupiny potom zo sekundárnej, tretie nakreslenie vrcholov je podľa toho ako sú uložené vo vstupnom grafe).
- **Planárne nakreslenie -** otestuje vstupný graf, ak je planárny, tak vygeneruje jeho planárne nakreslenie, ak nie je planárny, tak zobrazí o tom správu
- **Spring nakreslenie -** medzi susednými vrcholmi sú definované príťažlivé sily, medzi nesusednými odpudivé, algoritmus má ešte ako vstup počet iterácií, v každej iterácii zmení polohu vrchola vzhľadom na ostatné vrcholy a sily medzi nimi
- **Spring 2 nakreslenie -** spring metóda kreslenia grafu, len pred jej spustením odstráni z grafu hrany, ktoré spájajú vrcholy z rôznych skupín, po nakreslení ich do grafu znovu pridá

Pred spustením ľubovolného z týchto algoritmov program najprv otestuje, či vstupný graf je naozaj dvojfarebný, to znamená či neobsahuje slučky, paralelné hrany, či má aspoň jeden vrchol a či každý zo vstupných vrcholov patrí do jednej zo skupín. Po nakreslení grafu sa v ľavom hornom rohu zobrazia dve čísla, znamenajúce krížové číslo dvojfarebného grafu a počet krížení tohoto grafu. Tieto dve čísla sa pravidelne updatujú ak sa graf nejako zmení (pridá/odoberie hrana, pridá/odoberie vrchol, posunie vrchol).

5.3 Kreslenie dvojfarebných grafov dokresľovaním vrcholov

Pri dokresľovaní vrcholov máme na výber štyri základné algoritmy, ktorými môžeme vrchol do grafu dokresľovať. Tieto algoritmy majú rovnaké vstupy ako už uvedené algoritmy a naviac ešte obsahujú aj vstupné parametre, ktoré slúžia na určenie ako daný graf nakresliť.
Algoritmy založené na dokresľovaní vrcholov do grafu :

- **Heuristika 1 -** Do grafu dokreslí vrchol podľa heuristiky uvedenej v predchádzajúcej kapitole, parametre heuristiky sú : $r = 10$, $k = 30$

- **Heuristika 2 -** Do grafu dokreslí vrchol podľa heuristiky uvedenej v predchádzajúcej kapitole, parametre heuristiky sú : $r = 10$, $k = 30$, $t = 2$

- **Spring -** Do grafu dokreslí vrchol spring metódou, už nakreslené vrcholy majú fixovanú polohu

- **Spring naraz -** Do grafu dokreslí všetky chýbajúce vrcholy naraz, už nakreslené vrcholy majú fixovanú polohu

Pred spustením týchto algoritmov musí ešte užívateľ nastaviť parametre nakreslenia :

- **Základné nakreslenie -** Akým algoritmom sa má nakresliť základný graf, sú na výber tri možnosti - náhodné nakreslenie, planárne nakreslenie, spring nakreslenie

- **Maximálny počet krížení -** Koľko maximálne krížení bude mať základný graf

- **Metóda spravenia základu -** Či sa má spraviť základný graf z prázdneho grafu pridávaním vrcholov, alebo z celého grafu G postupným odoberaním jeho vrcholov

- **Priorita vrcholov pri robení základného grafu -** Keď konštruujeme základný graf, tak ktoré vrcholy máme uprednostňovať aby boli v tomto grafe, parametre vrcholov sú : stupeň vrchola v celom grafe G, stupeň vrchola v tomto základnom grafe (nakreslenom), stupeň vrchola v jeho skupine, náhodné číslo vrchola

- **Priorita vrcholov pri dokresľovaní do grafu -** V akom poradí máme pridávať vrcholy do nakresleného grafu, parametre vrcholov sú : stupeň vrchola v celom grafe G, stupeň vrchola v tomto nakreslenom grafe (teda koľko jeho susedov je už nakreslených), stupeň vrchola v jeho skupine, náhodné číslo vrchola

Nakreslenie grafu sa konštruuje nasledovne : Začíname s prázdnym grafom alebo celým grafom G, podľa toho či základné nakreslenie konštruujeme pridávaním alebo odoberaním vrcholov. Ak odoberáme vrcholy, tak vždy odoberieme vrchol s najnižšou prioritou, až dokým nie je splnený parameter na počet krížení. Ak pridávame vrcholy do základného grafu, tak pridávame vrcholy s najvyššou prioritou, pokiaľ základný graf spĺňa parameter na počet krížení. Ak už máme základný graf nakreslený, tak vrcholy doň pridávame podľa ich priority (od najvyššej) a dokresľujeme podľa zvolenej metódy.

Kapitola 6

Porovnanie algoritmov kreslenia dvojfarebných grafov

V tejto kapitole porovnáme niektoré algoritmy uvedené v predchádzajúcej kapitole. V programe sme implementovali aj funkciu na testovanie grafu. Pri funkcii vytváranie dvojfarebného grafu, je možnosť zaškrtnúť políčko otestovať graf kresliacimi algoritmami. Keď sa graf vytvorí, tak sa postupne spustia algoritmy na kreslenie dvojfarebných grafov a nakoniec sa graf nakreslí algoritmom, pri ktorom mal najmenšie dvojfarebné krížové číslo.
Pri komplexnom testovaní sme postupovali nasledovne, náhodne sme vytvárali dvojfarebné grafy a na každom vytvorenom grafe sme spustili testované algoritmy. Výsledok každého algoritmu sme potom zapísali do súborov (kompletný textový protokol o testovaní `test.txt` a súbor určený na ďalšie spracovanie `test.out`). Testované grafy sme potom rozdelili do skupín, podľa počtu ich vrcholov a hustoty hrán :

- **G1 -** počet vrcholov 8-10 hustota hrán $\leq 50\%$
- **G2 -** počet vrcholov 11-14 hustota hrán $\leq 25\%$
- **G3 -** počet vrcholov 11-14 $25\% <$ hustota hrán $\leq 50\%$
- **G4 -** počet vrcholov 15-19 hustota hrán $\leq 25\%$
- **G5 -** počet vrcholov 15-19 $25\% <$ hustota hrán $\leq 50\%$
- **G6 -** počet vrcholov 20-30 hustota hrán $\leq 20\%$
- **G7 -** počet vrcholov 31-50 hustota hrán $\leq 12\%$

Pre ostatné typy grafov sme implementované algoritmy netestovali, pretože ak má graf veľa hrán, tak vzniká veľmi veľa krížení a nakreslenie je neprehľadné. Takisto časová zložitosť niektorých algoritmov (založených na Heuristikách 1,2) je veľmi veľká a testovanie by trvalo príliš dlho.

Každý z testovaných grafov sme otestovali pomocou viacerých algoritmov a nastavení parametrov :

- Dokresľovanie postupne Heuristika 1 (**A1**) - 8 rôznych nastavaní parametrov
- Dokresľovanie postupne Heuristika 2 (**A2**) - 8 rôznych nastavaní parametrov
- Dokresľovanie postupne Spring (**A3**) - 8 rôznych nastavaní parametrov
- Dokresľovanie naraz Spring (**A4**) - 8 rôznych nastavaní parametrov
- Kruhové nakreslenie - v poradí (**A5**), nahodné (**A6**)
- Spring nakreslenie (**A7**), Spring 2 nakreslenie (**A8**)
- Náhodné nakreslenie (**A9**)

Pri dokresľovaní, kde sme pre každý algoritmus testovali 8 rôznych nastavení parametrov, boli tieto parametre rovnaké :

- Základné nakreslenie : Planárne(1) / Spring(2)
- Základné nakreslenie spraviť : Pridávaním(3) / Odoberaním(4) vrcholov
- Pri konštruovaní základného grafu sú priority vrcholov : Deg v celom $>$ Deg v nakreslenom $>$ Deg v skupine (5) / Deg v nakreslenom $>$ Deg v celom $>$ Deg v skupine (6)

Čísla v zátvorkách slúžia na označenie ako boli nastavené parametre pri testovaní. Napríklad algoritmus označený ako $\mathbf{A2}_{145}$ je dokresľovanie vrcholov pomocou Heuristiky 2, základný graf je nakreslený planárne odoberaním vrcholov z grafu G, vrcholy sú v základnom grafe s prioritou Deg v celom grafe, potom Deg v nakreslenom grafe, potom Deg v skupine, potom náhodne.
Ďalšie nastavenie parametrov je nasledovné : maximálny počet krížení je 0, priorita pri dokresľovaní vrcholov je Deg v celom $>$ Deg v nakreslenom $>$ Deg v skupine. Ak bol základný graf kreslený spring metódou, tak sme parameter maximálny počet krížení v základnom nakreslení testovali pre hodnoty 0, 5 a

10, relatívne najlepšie výsledky dosahovali kresliace algoritmy pri nastavení tohoto parametra na hodnotu 0.

Pri testovaní algoritmov pre rôzne parametre sú v nasledujúcich tabuľkách pre každé nastavenie parametrov a každý typ grafu 3 namerané hodnoty. **P%** udáva pre koľko % grafov bolo nakreslenie grafu s týmito parametrami najlepšie spomedzi ostatných nastavení vzhľadom na dvojfarebné krížové číslo (súčet % môže byť väčší ako 100% pretože niektoré nakreslenia môžu mať rovnaké dvojfarebné krížové číslo). **Cr** udáva priemerný počet krížení nameraný na testovaných grafoch, **2F** priemerné dvojfarebné krížové číslo. Najlepšie namerané hodnoty sú v tabuľke označené **bold** písmom.

6.1 Heuristika 1 - testovanie pre rôzne parametre

Pomocou Heuristiky 1 sme testovali nakreslenie dvojfarebných grafov na 8 rôznych nastavení parametrov.

	$\mathbf{A1}_{135}$			$\mathbf{A1}_{136}$			$\mathbf{A1}_{145}$			$\mathbf{A1}_{146}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	87	**0**	**1**	**89**	**0**	**1**	88	**0**	**1**	87	**0**	**1**
$\mathbf{G}_2$	97	**0**	**0**	98	**0**	**0**	97	**0**	**0**	**99**	**0**	**0**
$\mathbf{G}_3$	**47**	**10**	**35**	43	11	39	33	12	39	28	13	43
$\mathbf{G}_4$	59	**4**	**16**	**60**	**4**	17	53	5	18	56	5	18
$\mathbf{G}_5$	31	**65**	**226**	22	71	242	**32**	69	235	11	75	256
$\mathbf{G}_6$	**38**	**54**	**204**	30	55	210	30	56	210	22	64	239
$\mathbf{G}_7$	**32**	125	489	28	**123**	**484**	24	132	514	14	152	592
	$\mathbf{A1}_{235}$			$\mathbf{A1}_{236}$			$\mathbf{A1}_{245}$			$\mathbf{A1}_{246}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	49	1	5	56	1	5	68	1	3	62	1	4
$\mathbf{G}_2$	55	**0**	3	50	1	4	64	**0**	3	59	1	4
$\mathbf{G}_3$	5	19	62	2	19	62	8	16	54	12	15	50
$\mathbf{G}_4$	14	8	31	21	8	30	21	8	30	19	9	32
$\mathbf{G}_5$	0	99	338	2	98	331	6	80	270	4	83	279
$\mathbf{G}_6$	6	70	261	5	71	264	5	75	280	5	77	287
$\mathbf{G}_7$	4	155	602	6	157	613	0	188	749	4	184	709

Z tabuľky je vidieť, že ak základný graf kreslíme planárne, tak je to oveľa efektívnejšie vzhľadom na počet krížení aj dvojfarebné krížové číslo. Tak is-

to sa zdá byť efektívnejšie základný graf konštruovať pridávaním vrcholov do prázdneho grafu. Pri postupnom konštruovaní základného grafu sa nedá povedať, ktorá priorita vrcholov dáva lepšie výsledky.
Relatívne najlepšie výsledky dosahuje algoritmus $\mathbf{A1}_{135}$.

6.2 Heuristika 2 - testovanie pre rôzne parametre

Pomocou Heuristiky 2 sme testovali nakreslenie dvojfarebných grafov na 8 rôznych nastavení parametrov.

	$\mathbf{A2}_{135}$			$\mathbf{A2}_{136}$			$\mathbf{A2}_{145}$			$\mathbf{A2}_{146}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	86	**0**	**1**	86	**0**	**1**	**91**	**0**	**1**	89	**0**	**1**
$\mathbf{G}_2$	**99**	**0**	**0**	96	**0**	**0**	97	**0**	**0**	**99**	**0**	**0**
$\mathbf{G}_3$	**48**	**13**	45	47	**13**	**43**	33	14	49	32	16	54
$\mathbf{G}_4$	61	**4**	**16**	**64**	5	18	54	5	20	47	5	19
$\mathbf{G}_5$	**31**	**75**	**255**	24	80	277	22	**75**	256	17	81	279
$\mathbf{G}_6$	30	**57**	**223**	24	61	229	**38**	62	234	30	67	252
$\mathbf{G}_7$	**32**	135	534	22	**133**	**525**	22	137	537	20	144	573
	$\mathbf{A2}_{235}$			$\mathbf{A2}_{236}$			$\mathbf{A2}_{245}$			$\mathbf{A2}_{246}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	53	2	6	51	1	6	67	1	4	66	1	3
$\mathbf{G}_2$	60	1	3	54	1	4	59	**0**	3	60	**0**	2
$\mathbf{G}_3$	1	22	75	5	23	76	8	19	63	8	17	58
$\mathbf{G}_4$	20	9	32	17	10	37	18	8	29	23	7	27
$\mathbf{G}_5$	1	113	392	0	111	377	5	99	338	5	94	322
$\mathbf{G}_6$	6	80	297	6	80	302	3	79	302	16	71	270
$\mathbf{G}_7$	6	172	670	4	176	686	2	170	662	0	166	651

Z tabuľky je vidieť, že ak základný graf kreslíme planárne, tak je to oveľa efektívnejšie vzhľadom na počet krížení aj dvojfarebné krížové číslo. Ak používame ako základné nakreslenie planárne, tak je lepšie základný graf spraviť pridávaním vrcholov, ak použijeme ako základné nakreslenie spring, tak je ho lepšie konštruovať odoberaním vrcholov. Pri postupnom konštruovaní základného grafu sa nedá povedať, ktorá priorita vrcholov dáva lepšie výsledky.
Relatívne najlepšie výsledky dosahuje algoritmus $\mathbf{A2}_{135}$ a podobné výsledky $\mathbf{A2}_{136}$.

6.3 Spring postupne - testovanie pre rôzne parametre

Pomocou spring metódy postupného dokresľovania vrcholov sme testovali nakreslenie dvojfarebných grafov pre 8 rôznych nastavení parametrov.

	$\mathbf{A3}_{135}$			$\mathbf{A3}_{136}$			$\mathbf{A3}_{145}$			$\mathbf{A3}_{146}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	85	**0**	**2**	**87**	**0**	**2**	86	1	3	84	1	3
$\mathbf{G}_2$	95	**0**	**0**	96	**0**	**0**	**99**	**0**	**0**	95	**0**	**0**
$\mathbf{G}_3$	**38**	**18**	**63**	32	**18**	**63**	21	21	73	22	22	77
$\mathbf{G}_4$	**63**	**7**	**28**	**63**	**7**	29	40	11	43	41	12	48
$\mathbf{G}_5$	**20**	**104**	**368**	**20**	106	375	18	110	389	13	112	394
$\mathbf{G}_6$	**43**	84	325	**43**	**83**	**324**	14	98	385	16	99	391
$\mathbf{G}_7$	**36**	**183**	**733**	28	187	748	10	218	879	2	221	890
	$\mathbf{A3}_{235}$			$\mathbf{A3}_{236}$			$\mathbf{A3}_{245}$			$\mathbf{A3}_{246}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	47	2	8	41	2	8	47	2	7	50	2	7
$\mathbf{G}_2$	45	1	6	36	1	7	45	1	5	44	1	7
$\mathbf{G}_3$	8	25	87	6	25	86	8	24	82	9	25	87
$\mathbf{G}_4$	16	13	49	14	14	53	15	12	47	13	14	54
$\mathbf{G}_5$	9	117	409	7	114	400	9	117	408	8	119	424
$\mathbf{G}_6$	6	101	385	2	99	387	5	92	359	8	99	389
$\mathbf{G}_7$	18	204	799	6	213	864	10	204	822	2	213	863

Relatívne najlepšie výsledky dosahujú algoritmy $\mathbf{A3}_{135}$ a $\mathbf{A3}_{136}$. Ostatné nastavenia parametrov dosiahli približne rovnaké výsledky.

6.4 Spring naraz - testovanie pre rôzne parametre

Pomocou spring metódy dokreslenia vrcholov naraz sme testovali nakreslenie dvojfarebných grafov pre 8 rôznych nastavení parametrov.

	$\mathbf{A4}_{135}$			$\mathbf{A4}_{136}$			$\mathbf{A4}_{145}$			$\mathbf{A4}_{146}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	**87**	**0**	**2**	84	**0**	**2**	82	1	3	80	1	3
$\mathbf{G}_2$	96	**0**	**0**	**97**	**0**	**0**	95	**0**	**0**	**97**	**0**	**0**
$\mathbf{G}_3$	**36**	**17**	**60**	31	18	61	25	18	64	26	19	65
$\mathbf{G}_4$	53	**8**	30	**55**	**8**	**29**	46	9	36	44	9	34
$\mathbf{G}_5$	19	99	351	15	101	357	**20**	**98**	**343**	15	99	347
$\mathbf{G}_6$	22	82	320	**27**	82	320	21	83	323	22	80	313
$\mathbf{G}_7$	16	178	712	12	186	741	12	182	743	18	164	663
	$\mathbf{A4}_{235}$			$\mathbf{A4}_{236}$			$\mathbf{A4}_{245}$			$\mathbf{A4}_{246}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	46	2	7	46	2	7	59	1	5	59	1	5
$\mathbf{G}_2$	44	1	6	37	2	7	52	1	4	50	1	6
$\mathbf{G}_3$	3	23	81	6	23	81	14	20	70	10	20	71
$\mathbf{G}_4$	18	13	49	12	14	52	14	10	39	21	10	37
$\mathbf{G}_5$	4	112	397	4	115	400	12	100	352	16	101	351
$\mathbf{G}_6$	11	87	336	6	96	371	16	**72**	**281**	24	73	282
$\mathbf{G}_7$	0	196	785	4	202	805	**28**	**156**	**633**	24	160	650

Pri tomto testovaní sme dostali veľmi nejednoznačné výsledky. Pre rôzne typy grafov dosiahlo najlepšie výsledky iné nastavenie parametrov. Pre malé grafy to bol algoritmus $\mathbf{A4}_{135}$, pre stredné grafy to boli $\mathbf{A4}_{136}$ a $\mathbf{A4}_{145}$, pre väčšie grafy $\mathbf{A4}_{245}$.

6.5 Porovnanie najlepších algoritmov

V tejto podkapitole vzájomne porovnáme rôzne typy algoritmov, ktoré sme testovali. Tam, kde sme testovali algoritmus pre viacero nastavení parametrov, sme vybrali to nastavenie parametrov, ktoré dávalo relatívne najlepšie výsledky. Pri dokresľovaní vrcholov to bol zvyčajne planárny základ a konštruovanie základného grafu pridávaním vrcholov. Vysvetlenie číselných parametrov pri dokresľovacích algoritmov nájdeme na strane 37.

	$\mathbf{A1}_{135}$			$\mathbf{A2}_{135}$			$\mathbf{A3}_{135}$			$\mathbf{A4}_{245}$		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	**97**	**0**	**1**	91	**0**	**1**	84	**0**	2	57	1	5
$\mathbf{G}_2$	98	**0**	**0**	**99**	**0**	**0**	95	**0**	**0**	52	1	4
$\mathbf{G}_3$	**70**	**10**	**35**	51	13	45	15	18	63	6	20	70
$\mathbf{G}_4$	**76**	**4**	**16**	67	**4**	**16**	39	7	28	14	10	39
$\mathbf{G}_5$	**67**	**65**	**226**	30	75	255	0	104	368	1	100	352
$\mathbf{G}_6$	**65**	**54**	**204**	40	57	223	16	84	325	6	72	281
$\mathbf{G}_7$	**56**	**125**	**489**	30	135	534	2	183	733	8	156	633
	A5			**A7**			**A8**			**A9**		
	P%	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**	**P%**	**Cr**	**2F**
$\mathbf{G}_1$	6	18	54	42	2	8	42	2	7	4	11	35
$\mathbf{G}_2$	0	23	78	39	1	6	36	2	6	1	17	64
$\mathbf{G}_3$	0	108	338	1	24	79	2	29	83	0	67	232
$\mathbf{G}_4$	0	88	296	11	11	43	11	14	43	0	62	233
$\mathbf{G}_5$	0	415	1339	2	109	375	0	134	389	0	251	892
$\mathbf{G}_6$	0	407	1415	0	79	302	2	95	298	0	324	1262
$\mathbf{G}_7$	0	916	3326	0	164	658	10	191	614	0	690	2760

Z tabuľky je vidieť, že najefektívnejší algoritmus z hľadiska počtu krížení ako aj dvojfarebného krížového čísla je jednoznačne algoritmus založený na Heuristike 1. Druhým najlepším algoritmom v týchto kritériách je ten založený na Heuristike 2. Vo všeobecnosti algoritmy založené na dokresľovaní vrcholov dosiahli lepšie výsledky ako tie ostatné. Zaujímavé je, že náhodné nakreslenie dosiahlo lepšie výsledky ako nakreslenie kruhové a pre väčšie grafy bolo spring dokresľovanie vrcholov naraz efektívnejšie ako spring dokresľovanie postupne.

Pri testovaní sme zároveň merali aj časy, za ktoré každý algoritmus testované grafy nakreslil. Všetky testy boli robené na počítači AMD 1GHz, 256 MB RAM, pod systémom `Mandrake Linux 10.0`. Priemerný čas v sekundách na

nakreslenie jedného grafu pre každý algoritmus je v Tabuľke 2. Ako je vidieť, časovo najnáročnejší je algoritmus založený na Heuristike 2, graf z množiny **G7** nakreslil v priemere za 68.20 sekúnd. Časovo náročný je aj algoritmus založený na Heuristike 1, ostatné algoritmy v tomto meraní dosiahli dobré výsledky. To, že algoritmy založené na Heuristikách 1 a 2 boli časovo na tom horšie sa dalo očakávať podľa analýzy ich zložitosti, ktorá sa nachádza na strane 27. Prekvapilo ale, že pri väčších grafoch bol algoritmus založený na Heuristike 1 viac ako 8-krát rýchlejší ako ten založený na Heuristike 2.

	$\mathbf{A1}_{135}$	$\mathbf{A2}_{135}$	$\mathbf{A3}_{135}$	$\mathbf{A4}_{245}$	**A5**	**A7**	**A8**	**A9**
$\mathbf{G}_1$	0.04	0.07	0.03	0.05	0.03	0.07	0.06	0.03
$\mathbf{G}_2$	0.04	0.05	0.04	0.06	0.04	0.09	0.08	0.04
$\mathbf{G}_3$	0.20	0.85	0.09	0.09	0.05	0.13	0.11	0.05
$\mathbf{G}_4$	0.16	0.60	0.08	0.11	0.06	0.14	0.12	0.06
$\mathbf{G}_5$	1.19	6.69	0.21	0.18	0.10	0.19	0.16	0.09
$\mathbf{G}_6$	1.38	11.10	0.28	0.25	0.13	0.24	0.22	0.12
$\mathbf{G}_7$	8.27	68.20	0.61	0.59	0.28	0.41	0.36	0.24

Tabuľka 2: Priemerný čas na nakreslenie jedného grafu v sec.

Celkovo, algoritmus založený na Heuristike 1 dosiahol oveľa lepšie výsledky z hľadiska počtu krížení, dvojfarebného krížového čísla ako aj rýchlosti nakreslenia oproti algoritmu založenému na Heuristike 2.

Kapitola 7

Záver

Problematika kreslenia grafov, ktorej sme sa v diplomovej práci venovali, je značne rozsiahla. V práci sme sa zamerali na kreslenie dvojfarebných grafov, ktorému sa žiadna dostupná literatúra nevenovala.
Pre tieto grafy sme navrhli dva nové algoritmy založené na dokresľovaní vrcholov. Oba algoritmy dosahovali v porovnaní so základnými algoritmami na kreslenie grafov oveľa lepšie výsledky z hľadiska počtu krížení hrán ako aj krížového čísla dvojfarebného grafu, ktoré sme zadefinovali ako metriku. Nevýhodou týchto algoritmov je ich veľká časová zložitosť, preto by bolo vhodné nájsť aj algoritmy s menšou časovou zložitosťou, ktoré dosahujú porovnateľne dobré nakreslenie dvojfarebného grafu. Mohli by to byť algoritmy využívajúce tieto upravené algoritmy, ktoré by boli menej časovo náročné, alebo algoritmy, ktoré by vznikli modifikáciou niektorých existujúcich algoritmov na kreslenie všeobecných grafov.
Tak isto zaujímavý problém je dokresľovanie vrcholov do nakresleného grafu a bolo by zaujímavé sa pozrieť na dokresľovanie viacerých vrcholov do grafu naraz.

Literatúra

[1] R. Tamassia (1997): *Graph Drawing* v J. E. Goodman, J. O'Rourke eds.: *CRC Handbook of Discrete and Computational Geometry*, CRC Press
`http://www.cs.brown.edu/cgc/papers/t-gd-97.ps.gz`

[2] Roberto Tamassia : *Advances in the Theory and Practice of Graph Drawing*, Theoretical Computer Science, Volume 217, Issue 2, 6 April 1999, Pages 235-254

[3] Reinhard Diestel : *Graph theory*, Electronic edition 2000, Springer-Verlag 2000
`ftp://ftp.math.uni-hamburg.de/pub/unihh/math/books/diestel/GraphTheoryII.pdf`

[4] Ashim Garg a Roberto Tamassia : *Upward planarity testing*, Order (12), pp. 109-133, 1995
`http://www.cs.brown.edu/cgc/papers/gt-upt-95.ps.gz`

[5] Peiran LIU : *Design and Analysis of Algorithms*, Seminar Report, University of Ottawa - S.I.T.E. November 10, 2002

[6] Susan Sim : *Automatic Graph Drawing Algorithm*, 1996
`http://www.ics.uci.edu/~ses/papers/grafdraw.pdf`

[7] Prof. Dr. Franz J. Brandenburg : *Graph Drawing: past - present - future*, 2002
`http://www.csse.monash.edu.au/~gfarr/research/GraphDrawing02-Mel.ppt`

[8] Petra Mutzel : *Optimization in graph drawing*, P. Pardalos and M. Resende, editors: *Handbook of Applied Optimization*, pages 967-977, Oxforf University Press, New York, 2002

[9] Giuseppe Di Battista, Peter Eades, Roberto Tamassia a Ioannis G. Tollis : *Algorithms for Drawing Graph: an Annotated Bibliography*, 1994
`http://www.cs.brown.edu/people/rt/papers/gdbiblio.pdf`

[10] René Weiskircher: *Drawing Planar Graphs* v Michael Kaufmann a Dorothea Wagner (Eds.): *Drawing Graphs Methods and Models*, Computer Science Volume 2025 / 2001

[11] Isabel F. Cruz a Roberto Tamassia : *Graph Drawing Tutorial* `http://www.cs.brown.edu/people/rt/papers/gd-tutorial/gd-constraints.pdf`

[12] Carsten Gutwenger, Petra Mutzel a René Weiskircher : *Inserting an Edge Into a Planar Graph*, Proceedings of the Twelfth ACM-SIAM Symposium on Discrete Algorithms, (SODA '2001), Washington, DC, ACM Press, 2001, 246–255

[13] Lila Behzady : *An improved Spring-based Graph Embedding Algorithm and LayoutShow: a Java Environment for Graph Drawing*, 1999, York University, North York, Onratio

[14] Jürgen Branke, Frank Bucher a Hartmut Schmeck : *Using Genetic Algorithms for Drawing Undirected Graphs*, 1996, University of Karlsruhe

[15] Annegret Liebers : *Planarizing Graphs - A Survey and Annotated Bibliography*, Journal of Graph Algorithms and Applications 5(1):1-74, 2001 `http://www.cs.brown.edu/publications/jgaa/accepted/01/Liebers01.5.1.pdf`

[16] Michael Jünger a Petra Mutzel : *Maximum Planar Subgraphs and Nice Embeddings: Practical Layout Tools*, Algorithmica 16, No. 1, 1996, 33–59 `ftp://ftp.zpr.uni-koeln.de/pub/paper/zpr93-145.ps.gz`

[17] Kazimierz Kuratowski : *Sur le problème des courbes gauches en topologie*, Fund. Math., 15:271-283, 1930

[18] Yury Makarychev : *A Short Proof of Kuratowski's Graph Planarity Criterion*, Department of differential geometry, Faculty of mechanics and mathematics, Moscow, State university `http://www.cs.princeton.edu/ ymakaryc/papers/kuratowski.pdf`

[19] John Hopcroft a Robert Tarjan : *Efficient Planarity Testing*, Journal of the ACM (JACM), Volume 21 , Issue 4 (October 1974), Pages: 549 - 568

[20] Tobias Oetiker, Hubert Partl, Irene Hyna a Elisabeth Schlegl : *Nie príliš stručný úvod do systému LaTeX 2ε*, verzia 3.13, 23. február 2000, preklad Ján Buša ml. a Ján Buša st., 17.marec 2002

[21] Algoritmic Solutions : *LEDA 5.0, The LEDA User Manual*

[22] http://graphml.graphdrawing.org

[23] http://graphdrawing.org

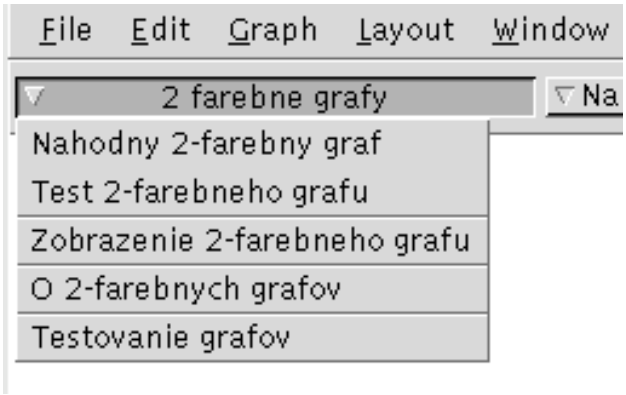
[24] http://planetmath.org

[25] http://maven.smith.edu/~orourke/TOPP/Welcome.html

Dodatok A

Krátky manuál k implementovanému programu

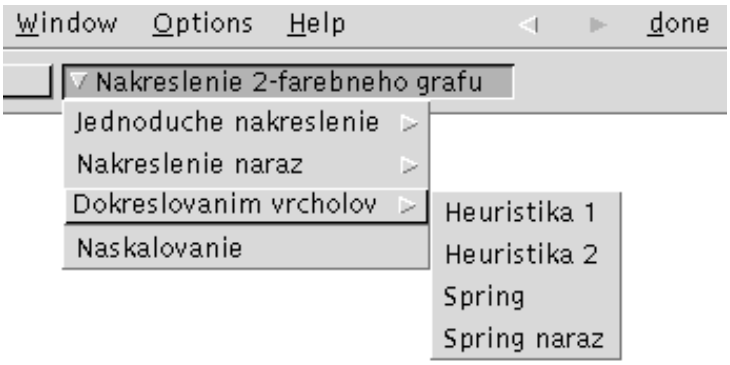
Program bol implementovaný nad knižnicou algoritmov `LEDA 5.0`, verzia pre `Mandrake 10.0`, kompilátor `g++ 3.2.2`. Využíva existujúce prostredie na pracovanie s grafmi `GraphWin`. Implementovali sme do tohoto prostredia funkcie, ktoré slúžia na vytváranie, zobrazovanie, a kreslenie dvojfarebných grafov. Tieto funkcie sme rozdelili do dvoch podmenu, *2 farebné grafy* (obrázok A.1) a *Nakreslenie 2-farebného grafu* (obrázok A.2).



Obrázok A.1: Menu : 2-farebné grafy

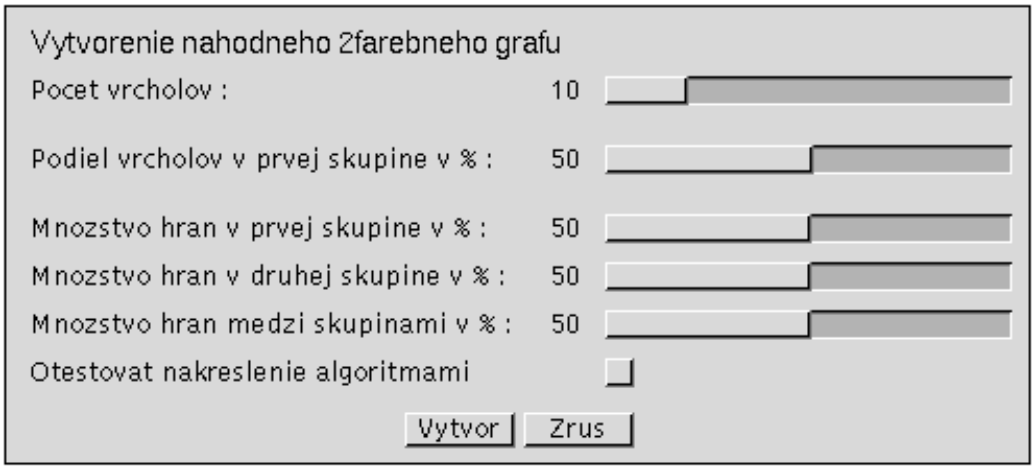
Pri kreslení 2-farebných grafov si môžeme vybrať z viacerých metód. Podmenu *Jednoduché nakreslenie* obsahuje náhodné a kruhové nakreslenie. Podmenu *Nakreslenie naraz* obsahuje planárne nakreslenie, spring nakreslenie a nakreslenie spring 2. Podmenu *Dokresľovaním vrcholov* (obrázok A.2) obsahuje algoritmy založené na dokresľovaní vrcholov. Voľba *Naškálovanie* roz-

tiahne nakreslenie grafu na rozmery otvoreného `GraphWin` okna.



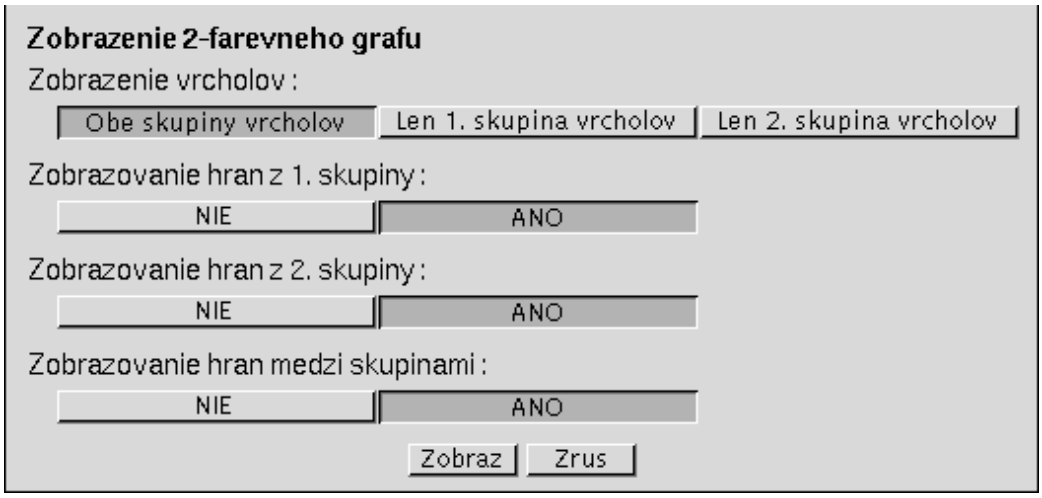
Obrázok A.2: Menu : Nakreslenie 2-farebných grafov

Pri vytváraní náhodného 2-farebného grafu musí užívateľ nastaviť parametre, podľa ktorých sa graf vytvorí. Tieto parametre sú počet vrcholov grafu, percentuálny podiel vrcholov z prvej skupiny, percentuálny podiel hrán medzi vrcholmi z prvej skupiny, medzi vrcholmi z druhej skupiny a percentuálny podiel hrán medzi vrcholmi z rôznych skupín (obrázok A.3). Užívateľ môže zaškrtnúť možnosť *Otestovať nakreslenie algoritmami*, čo spôsobí že vytvorený graf sa postupne nakreslí testovanými algoritmami a nakoniec zobrazí nakreslenie s najmenším dvojfarebným krížovým číslom.



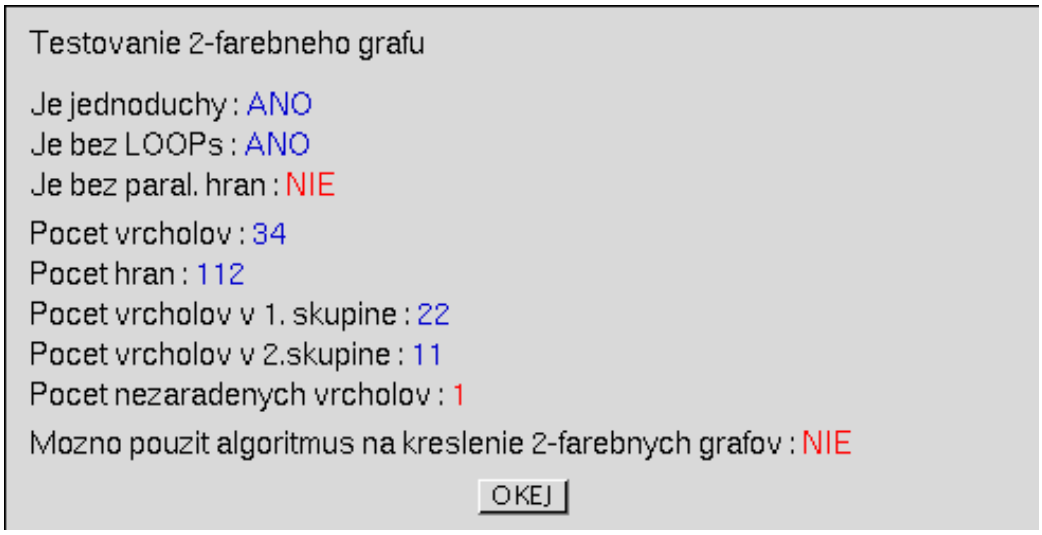
Obrázok A.3: Vytvorenie náhodného 2-farebného grafu

Ak máme 2-farebný graf nakreslený, tak si môžeme nechať zobraziť jeho jednotlivé časti (obrázok A.4). Môžeme zobraziť skupiny vrcholov ktoré chceme, alebo hrany spájajúce len niektoré typy vrcholov. Aby sa hrana zobrazila, tak musia byť zobrazené oba jej koncové vrcholy.



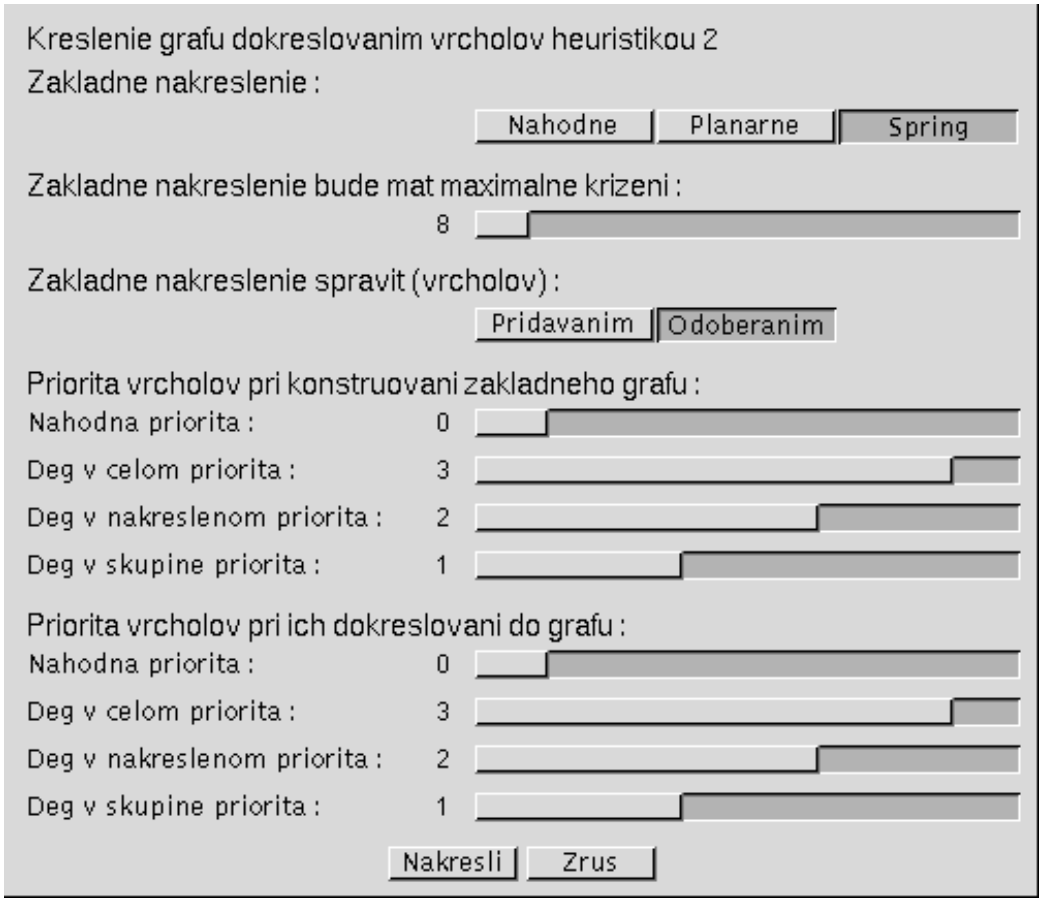
Obrázok A.4: Zobrazenie 2-farebného grafu

Ak máme nakreslený graf a nevieme, či je 2-farebný, tak si môžeme otestovať, ktoré podmienky 2-farebnosti spĺňa alebo nespĺňa (obrázok A.5). 2-farebný graf nesmie mať paralelné hrany, slučky, musí mať aspoň jeden vrchol a každý nakreslený vrchol musí patriť do jednej z dvoch skupín, čo znamená, že vrchol musí byť červenej (prvá skupina) alebo žltej (druhá skupina) farby.



Obrázok A.5: Testovanie či je graf 2-farebný

Ak chceme 2-farebný graf nakresliť algoritmom založeným na dokresľovaní vrcholov, tak musíme najprv nastaviť parametre, podľa ktorých sa bude daný graf kresliť. Nastavovanie týchto parametrov je na obrázku A.6.



Obrázok A.6: Kreslenie 2-farebných grafov dokresľovaním vrcholov.

Dodatok B

Prehľad niektorých implementovaných funkcií

Program bol implementovaný pomocou viacerých zdrojových súborov :

`2f_grafy.cpp` : je to základný zdrojový súbor, v ňom sú implementované základné funkcie a includované ostatné zdrojové súbory

`_create.c` : obsahuje funkciu na vytvorenie náhodného grafu podla zvolených parametrov

`_drawing.c` : obsahuje všetky implementované funkcie na kreslenie dvojfarebných grafov a k nim pomocné funkcie

Ešte boli implementované aj ďalšie programy slúžiace na vytváranie tabuliek z nameraných údajov :

`udaje.c` : načíta otestované údaje zo súboru `test.out` a podľa hodnoty konštanty `ALGORITMUS` vytvorí z nameraných údajov tabuľku a zapíše ju do súboru (podľa konštanty `ALGORITMUS` : `heu1.tex`, `heu2.tex`, `spring.tex` alebo `spring_nar.tex`)

`udaje_naj.c` : načíta otestované údaje zo súboru `test.out` a podľa konštánt `ALG_1`, `ALG_2`, ... `ALG_8` vytvorí z nameraných údajov tabuľku a zapíše ju do súboru `spolu.tex`

Niektoré vybraté funkcie z `2f_grafy.cpp` :

`int crossing_number(GraphWin& GW);`
vráti dvojfarebné krížové číslo grafu nakresleného na ploche `GW`

`int crossings(GraphWin& GW);`
vráti počet krížení grafu nakresleného na ploche `GW`

`void zobraz_graf(GraphWin& GW, int vrcholy, int hrany1, int hrany2, int hrany3);`
zobrazí dvojfarebný graf na ploche `GW` podľa zvolených parametrov (ktoré vrcholy/hrany sú zobrazené alebo skryté)

`void vypis_cr_number(GraphWin& GW);`
na plochu `GW` vľavo hore vypíše dvojfarebné krížové číslo a počet krížení grafu

`int testuj_2f(GraphWin& GW,int tabulka);`
otestuje, či graf na ploche `GW` je dvojfarebný, podľa hodnoty parametra vypíše o tom info do tabuľky

`void change_node(GraphWin& GW, const point&);`
zmení typ vrchola, ktorý obsahuje bod `point`

`void zobraz_graf_2f(GraphWin& GW);`
vykreslí tabuľku, v ktorej si užívateľ navolí parametre pre zobrazenie dvojfarebného grafu, následne graf podľa nastavených parametrov vykreslí

`void random2f(GraphWin& GW);`
vykreslí tabuľku, v ktorej si užívateľ navolí parametre pre náhodný dvojfarebný graf, následne takýto graf vytvorí

`int nakresli_graf(GraphWin& GW, int metoda, ...);`
graf na ploche `GW` vykreslí podľa zvolenej metódy a podľa daných parametrov

`int velke_testovanie(GraphWin& GW, int poc_vrcholov, int poc_hran, int vrch1, int hran1, int hran2, int hran3);`
na grafe z plochy `GW` spustí postupné testovanie všetkých testovaných algoritmov

`void pustaj_testy(GraphWin& GW,int random);`
na ploche `GW` postupne vytvára náhodné dvojfarebné grafy a testuje na nich algoritmy

Niektoré vybraté funkcie z `_create.c` :

`void create_2f_graph(ugraph &G1, int poc_vrcholov, int poc_hran, int vrcholy_1, int hrany_1, int hrany_2);`
podľa parametrov do objektu `G1` vytvorí nahodný dvojfarebný graf

Niektoré vybraté funkcie z `_drawing.c` :

```
void RANDOM(graph& G, node_array<double>& xpos,
    node_array<double>& ypos, ...);

void CIRCLE(graph& G, node_array<double>& xpos,
    node_array<double>& ypos, ...);

void PLANAR_NAK(graph& G, node_array<double>& xpos,
    node_array<double>& ypos, ...);

void VYTVOR_DOKRESLOVANIM(graph& G, list<node>& fixed,
    list<node>& nonfixed, node_array<double>& xpos,
    node_array<double>& ypos, ...);
```

Podľa metódy a parametrov vytvorí nakreslenie grafu `G` a súradnice jeho vrcholov zapíše do objektov `xpos`, `ypos`. Vo funkcii `VYTVOR_DOKRESLOVANIM` sú implementované všetky kresliace algoritmy založené na dokresľovaní vrcholov, to ktorou metódou ho nakreslí záleží na hodnote jeho parametrov.

```
void HEURISTIKA_1(graph &G,list<node> fixed, node& vrch,
    node_array<double>& xpos, node_array<double>& ypos, ...,
    double &pos_x, double &pos_y, ...);

void HEURISTIKA_1(graph &G,list<node> fixed, node& vrch,
    node_array<double>& xpos, node_array<double>& ypos, ...,
    double &pos_x, double &pos_y, ...);
```

Do grafu `G`, ktorý má nakreslené vrcholy z `fixed` na pozíciách z `xpos`, `ypos` vypočíta podľa zvolenej heuristiky pozíciu pre vrchol `vrch` a zapíše ju do premenných `pos_x`, `pos_y`.

```
void NASKALUJ(graph &G,node_array<double>& xpos,
    node_array<double>& ypos, double xleft, double xright,
    double ybottom, double ytop);
```

Nakreslenie grafu `G`, ktorého vrcholy sú na pozíciách z `xpos`, `ypos` roztiahne do obdĺžnika ohraničeného premennými `xleft`, `ytop`, `xright`, `ybottom`.

Dodatok C

Problémy pri implementácii v LEDA 5.0

V tejto časti prílohy uvedieme niektoré problémy, ktoré nastali pri implementovaní nad knižnicou `LEDA`. Časť z nich bolo veľmi ťažké odhaliť a nájsť pre ne uspokojujúce riešenie. Táto sekcia by mala pomôcť ďalším ľuďom, ktorí budú implementovať nad knižnicou `LEDA`, ušetriť čas stratený pri hľadaní a odstraňovaní chýb.

Problém : pri vytváraní objektu `Polygon` pomocou množiny bodov môže byť vytvorený polygón presne opačný ako sme chceli vytvoriť, to znamená že bude jeho doplnkom (metódy `.inside` a `.outside` fungujú presne opačne)

Riešenie : zistíme či vzdialený bod patrí vytvorenému polygónu, ak áno, tak polygónu priradíme jeho komplement :
`if (P.inside(point(-10000,-10000))) P=P.complement();`

Problém : niekedy pri vytváraní objektu `Polygon` pomocou množiny bodov vyhlási kritickú chybu **Segmentation fault** a ukončí program

Riešenie : pred vytvorením polygónu body otestujeme tak, že cyklicky medzi každými dvoma vytvoríme úsečku (segment), zistíme prieniky týchto úsečiek, ak sa množina bodov z prieniku rovná množine bodov, pomocou ktorých vytvárame polygon, tak ho môžeme vytvoriť

Problém : pri využívaní funkcií `LEDA` na rátanie priesečníkov medzi geometrickými útvarmi (ray's, segment's, line's) vznikajú vo výsledku odchýlky

Riešenie : treba dať pozor k čomu je výsledok určený, ak chceme zistiť či nejaký priesečník leží na najekom geometrickom útvare (alebo či je totožný s bodom) tak použijeme metódu `.distance()` na zistenie vzdialenosti od útvaru a porovnáme s nami zvolenou geometrickou odchýlkou (my sme zvolili odchýlku 0.002)

Problém : pri volaní funkcií, kde parametrom sú `LEDA` objekty ako `Graph` alebo `list<type>` musia byť tieto parametre referencované cez ich adresu, (teda cez operátor `&`), v opačnom prípade funkcia dostane pri jej volaní prázdne objekty (napríklad funkcia bude definovaná takto : `int testuj(Graph &G, list<node> &vrcholy, int pocet);`)

Riešenie : ak chceme vo volanej funkcii kópiu objektu modifikovať, tak si ju musíme vytvoriť a pracovať s ňou (priraďenie typu `Graph G1=G;` nefunguje !)

Problém : ak sú niektoré elementy grafu (vrcholy, hrany) skryté (hidden), tak pri použití iterátorov `for_all_nodes` (`for_all_edges`) sú dodávané aj niektoré skryté vrcholy namiesto neskrytých

Riešenie : ak chceme použiť iterátor cez všetky neskryté vrcholy (hrany) grafu, tak použijeme takúto konštrukciu :

```
node v=G.first_node();
while(v!=nil)
{
...
v=G.succ_node(v);
}
```

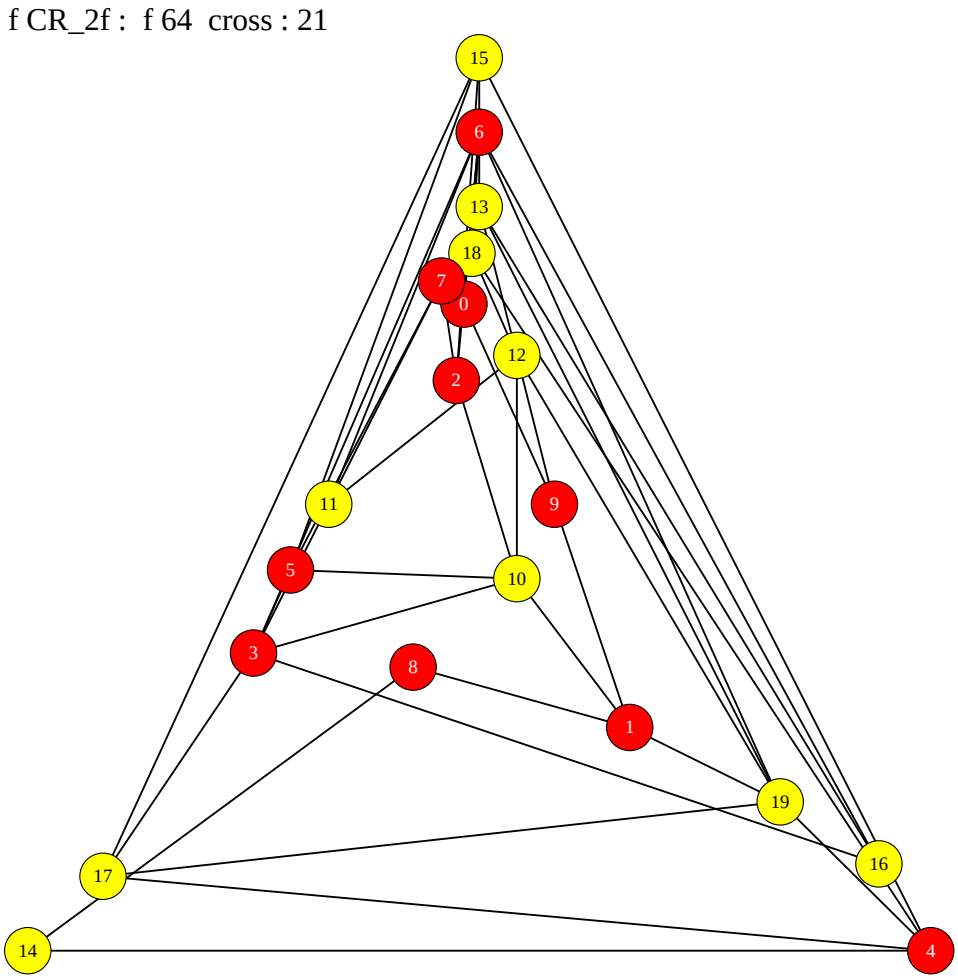
Problém : pri volaní funkcie `STRAIGHT_LINE_EMBEDDING`, ktorá slúži na planárne nakreslenie grafu, musí byť vstupný graf orientovaný, v opačnom prípade program vyhlási chybu

Riešenie : pred volaním funkcie použijeme na vstupný graf metódu `G.make_directed();`, po skončení funkcie `STRAIGHT_LINE_EMBEDDING` použijeme na graf metódu `G.make_undirected();`

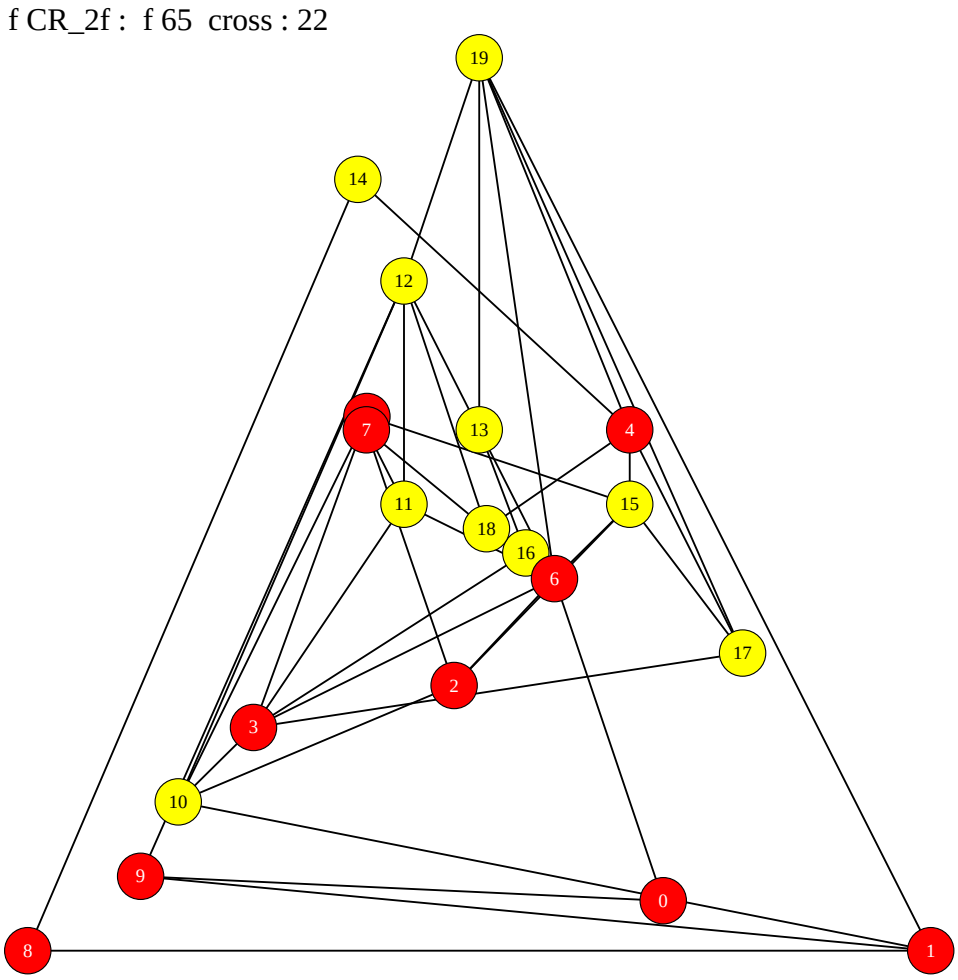
Dodatok D

Ukážky nakreslenia niektorých grafov

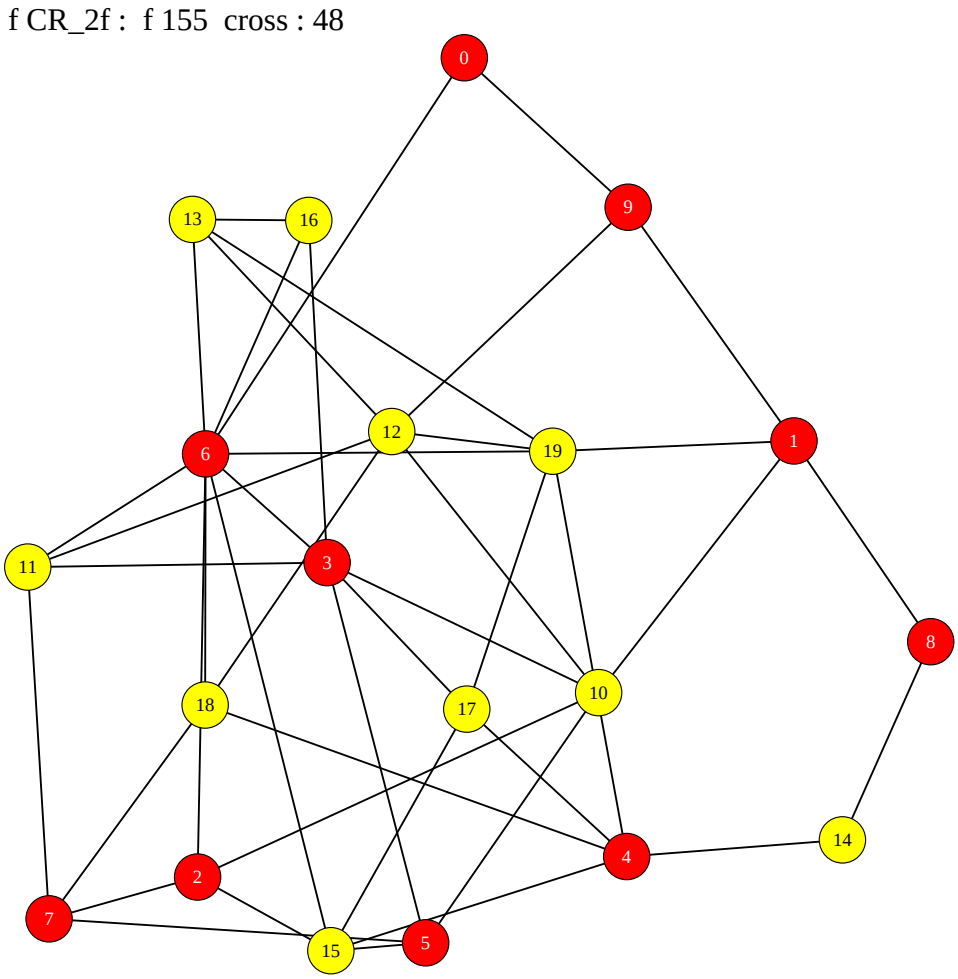
V tejto časti prílohy ukážeme výsledné nakreslenie dvoch grafov v implementovanom programe na kreslenie dvojfarebných grafov. Každý z týchto grafov bol nakreslený pomocou troch algoritmov : Heuristika 1, Heuristika 2 a spring metóda.



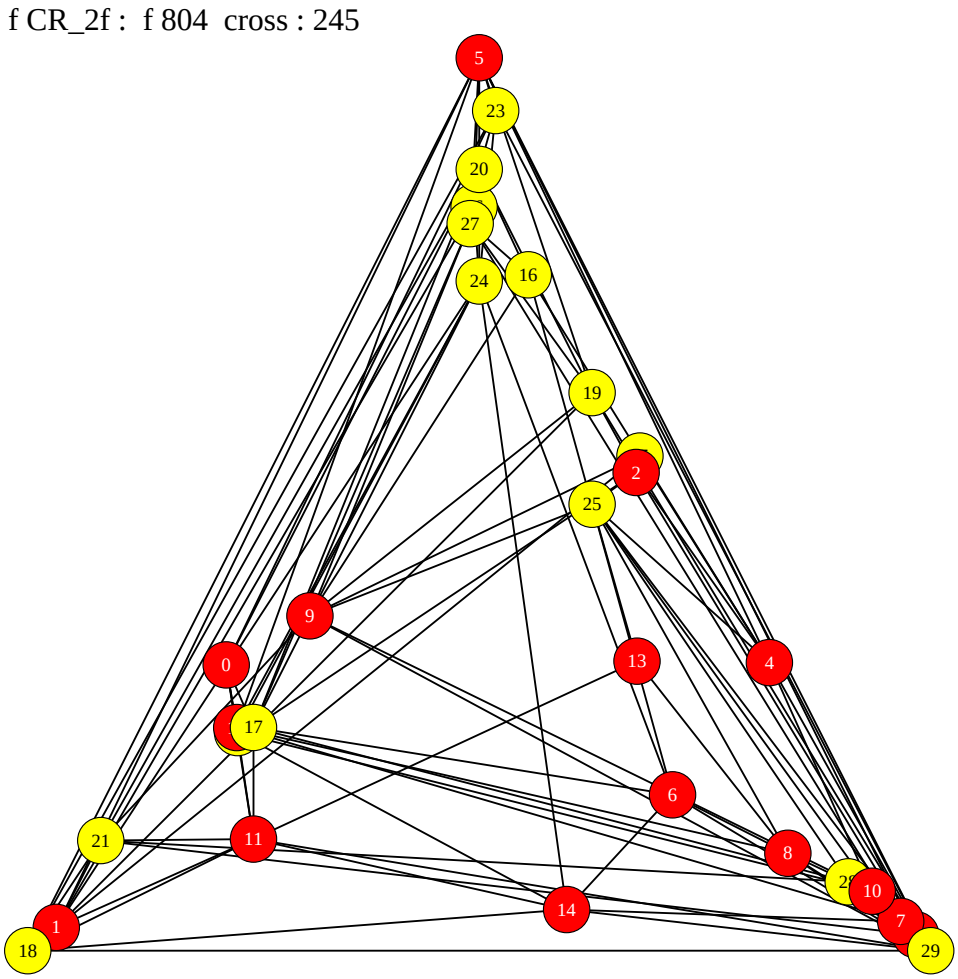
Obrázok D.1: Nakreslenie grafu, ktorý má 20 vrcholov a 43 hrán Heuristikou 1, `Cr2f : 64 cross : 21`.



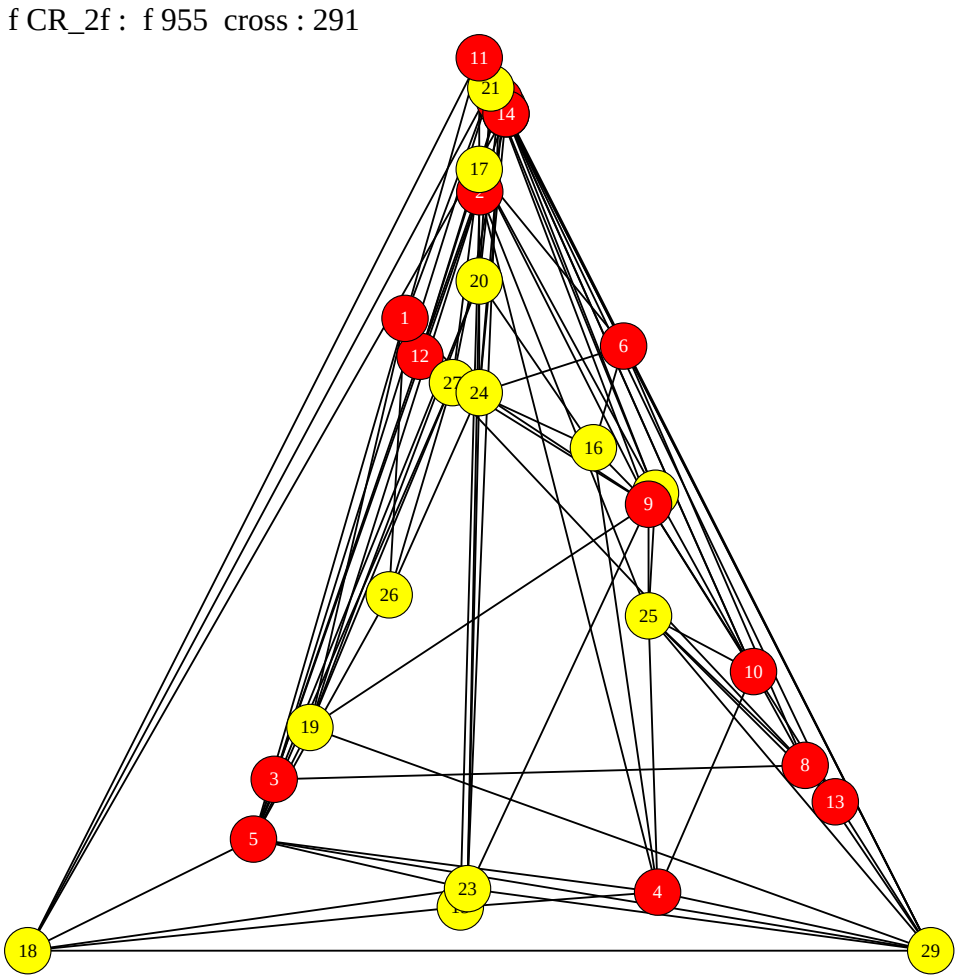
Obrázok D.2: Nakreslenie grafu, ktorý má 20 vrcholov a 43 hrán Heuristikou 2, `Cr2f : 65 cross : 22`.



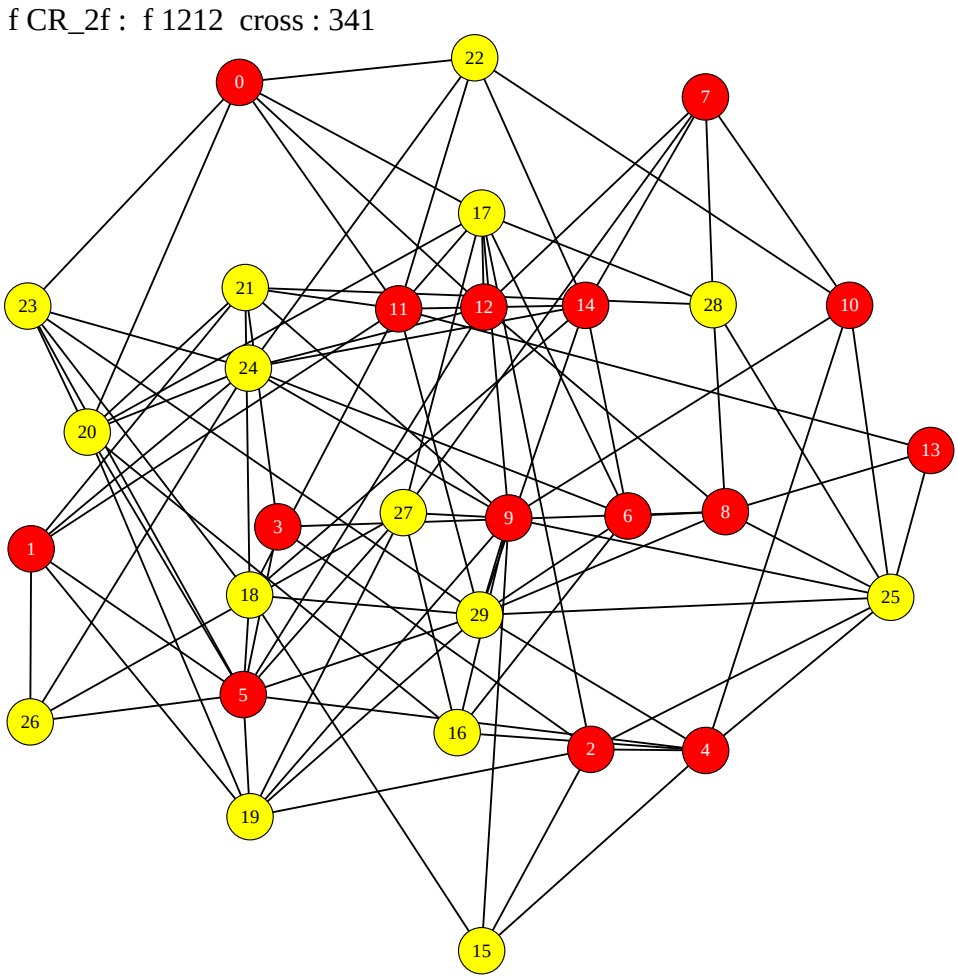
Obrázok D.3: Nakreslenie grafu, ktorý má 20 vrcholov a 43 hrán spring metódou, `Cr2f : 155 cross : 48`.



Obrázok D.4: Nakreslenie grafu, ktorý má 30 vrcholov a 100 hrán Heuristikou 1, `Cr2f : 804 cross : 245`.



Obrázok D.5: Nakreslenie grafu, ktorý má 30 vrcholov a 100 hrán Heuristikou 2, `Cr2f : 955 cross : 291`.



Obrázok D.6: Nakreslenie grafu, ktorý má 30 vrcholov a 100 hrán spring metódou, `Cr2f : 1212 cross : 341`.