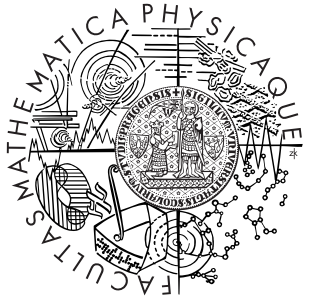
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

SVOČ 2009



Jakub Kratochvíl

Vizualizace grafů

Katedra Aplikované Matematiky

Vedoucí práce: RNDr. Martin Pergel

Studijní program: Informatika, programování

2008

Děkuji svoji rodině, že mě podporovala ve studiu. Děkuji také svému vedoucímu Martinu Perglovi za podněty při vypracování práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

v Praze dne Jakub Kratochvíl

Obsah

Kapitola 1

Úvod

Kreslení grafů je prostředkem pro vizualizaci komplexních struktur a jejich následnou prezentaci uživateli. První metody kreslení grafů vycházely z fyzikálních metod viz např. [23]. Centrem zájmu dlouhou dobu bylo zejména kreslení rovinných grafů a stromů. V 80. letech se objevily první algoritmy založené na hledání vlastních čísel spektra grafu. Fyzikální metody však postačovaly pro většinu aplikací, dokud se neobjevila potřeba kreslit rozsáhlé grafy s mnoha tisíci vrcholy.

Prvním krokem, který umožnil kreslení takto velkých grafů, byla implementace shlukování do fyzikálních algoritmů, která značně redukovala jejich asymptotickou složitost viz [21]. Zhruba ve stejné době se do středu zájmu dostaly metody založené na výpočtu vlastních čísel, které využívají shlukování. Ukazálo se totiž, že vlastní čísla grafu lze aproximovat pomocí vhodně zvoleného malého grafu s konstantní velikostí.

Stranou zájmu dlouhou dobu zůstavalo kreslení grafů ve 3D, přestože nabízí řadu výhod z uživatelského pohledu – jako zásadní se jeví možnost libovolně rotovat a přibližovat kameru. Takto lze získat mnohem detailnější přehled o výsledném nakreslení, a navíc odpadá potřeba minimalizovat počet křížení hran, jak bude ukázáno ve druhé kapitole. Dodejme, že minimalizace křížení ve 2D je $\mathcal{NP}$-úplná úloha.

V textu ukážeme, jak kreslit grafy za použití high dimensional embedding (HDE), a předložíme nový algoritmus založený na vzorkování HDE s lineární složitostí. Podrobně bude dokázána složitost našeho algoritmu a výsledky budou porovnány s algoritmem založeným na PCA-projekci [10].

Dále jsme vytvořili aplikaci pro Windows, která umožňuje využít výhody třídimenzionálního nakreslení grafu.

V následující části definujeme základní pojmy lineární algebry a teorie grafů.

Graf G je uspořádaná dvojice (V, E), kde V je neprázdná množina vrcholů a E množina dvojic prvků V. Prvky množiny E nazýváme hranami grafu G. Je-li E množina uspořádaných dvojic, pak hovoříme o *orientovaném grafu*. V dalším textu se omezíme na grafy neorientované tedy takové, kde E je množina neuspořádaných dvojic. Vrcholy u, v grafu G označíme jako sousedy, pokud mezi nimi existuje v G hrana. Cestou mezi v_1 a v_n rozumíme posloupnost sousedních vrcholů, kde se neopakují hrany ani vrcholy. Souvislý graf je takový graf, ve kterém mezi každými dvěma vrcholy existuje cesta.

Definujme nyní nakreslení grafu G.

Definice 1.1 Nakreslením grafu $G = (V, E)$ *do prostoru* R^n rovnými čarami *rozumíme zobrazení* $\phi : v \rightarrow R^n$, *kde hrany kreslíme úsečkami mezi jednotlivými vrcholy.*

V našem případě se budeme zabývat kreslením ve 2 a 3-rozměrném prostoru a tedy n bude v našem případě rovno 2 nebo 3.

Dále uvedeme základní poznatky o vlastních číslech. λ je vlastní číslo reálné (komplexní) čtvercové matice A, pokud existuje nenulový vektor x akový, že $Ax = \lambda x$. Vektor x je pak *vlastním vektorem* matice A příslušným λ. Zdůrazněme, že symetrická matice má všechna vlastní čísla reálná, jak je dokázáno v [13]. Symetrická čtvercová matice je *pozitivně definitní*, pokud $\forall x \neq \vec{0} : xAx^T > 0$. Na závěr uvedeme vztah mezi vlastními čísli a pozitivně definitními maticemi.

Věta 1.1 *Matice je pozitivně definitní tehdy a jen tehdy, když má všechna vlastní čísla kladná.*

Důkaz: viz [13].

Kapitola 2

Algebraické metody

V této části popíšeme algoritmus PCA projekce pocházející z [10] a porovnáme ho s námi navrhovaným novým algoritmem.

2.1 Problém k-center

Problém $\mathcal{Q}$ patří do *třídy* NP, pokud lze jeho řešení ověřit v polynomiálním čase. Pokud navíc lze každý problém třídy NP, převést v polynomiálním čase na problém $\mathcal{Q}$, pak takový problém nazýváme NP*-úplným.*

Definice 2.1 (Problém k-center) *Problém k-center definujeme následovně: Buď $\mathcal{S}$ množina n bodů v prostoru s mírou* dist . *Najdi množínu $\mathcal{C}$ velikosti k takovou, že*

$$\max_{s\in\mathcal{S}} \min_{c\in\mathcal{C}} \mathrm{dist}(c,s) \text{ je minimální.}$$

Věta 2.1 *Problém k-center je NP-těžký.*

Důkaz: viz [12].

Definice 2.2 *Buď I množina všech instancí NP-úplného problému $\mathcal{Q}$. Polynomiální algoritmus $\mathcal{A}$ nazveme ϵ-aproximací NP-úplné úlohy $\mathcal{Q}$, jestliže*

$$\max_I \left\{ \frac{C}{C^*}, \frac{C^*}{C} \right\} \leq \epsilon,$$

kde C je cena řešení nalezená $\mathcal{A}$ a C^* je optimální řešení $\mathcal{Q}$.

2-Aproximace problému k-center hladovým algoritmem

Vstup: Množina S

1. Vyber $s \in \mathcal{S}$ libovolně, polož $\mathcal{S} = \mathcal{S} \backslash \{x\}$ a $\mathcal{C} = \{s\}$.
2. Najdi bod $x \in \mathcal{S}$ takový, že $\min_{c \in \mathcal{C}} \text{dist}(x, c)$ je maximální.
3. Polož $\mathcal{S} = \mathcal{S} \backslash \{x\}$ a $\mathcal{C} = \mathcal{C} \cup \{x\}$.
4. Pokud $|\mathcal{C}| < \|$, vrať se na 2.

Definice 2.3 *Alternativně lze problém k-center definovat takto: Buď $\mathcal{S}$ množina. Rozhodni, zda existuje k hyperkoulí s poloměrem r,které pokrývají množinu $\mathcal{S}$.*

Věta 2.2 *Výše uvedený algoritmus je 2-aproximací problému k-center.*

Důkaz: viz [1]. Buď r^* optimální řešení problému k-center. Označme r řešení nalezené aproximačním algoritmem. Pro spor buď $r > 2r^*$. Existuje tedy bod x, který je vzdálen alespoň r od k center nalezených aproximačním algortimem. Pak ovšem body nalezené aproximačním algoritmem a tento bod mají vzájemnou vzdálenost alespoň r. Ovšem v žádné hyperkouli o poloměru r^* nemohou být dva body se vzdáleností větší než $2r^*$. Tudíž k pokrytí množiny by bylo třeba alespoň k+1 hyperkoulí, což je spor s faktem, že existuje pokrytí množiny k hyperkoulemi.

Hladový algoritmus není pouze 2-aproximací problému k-center, ale dokonce platí i následující tvrzení.

Věta 2.3 *Pokud $P \neq NP$, pak je uvedený hladový algoritmus nejlepší možnou polynomiální aproximací problému k-center.*

Důkaz: viz [12]

Problém k-center jsme zde definovali obecně, budeme ho však aplikovat na grafy. Množinou $\mathcal{S}$ bude tedy množina vrcholů. Metrikou dist v našem případě bude teoretická grafová vzdálenost - tedy cena nejkratší cesty mezi vrcholy i, j. V případě, že graf bude mít kladně ohodnocené hrany, využijeme k nalezení nejvzdálenějšího vrcholu v kroku 3 Dijkstrova algoritmu. Pokud budou mít hrany jednotkovou délku, aplikujeme průchod do šířky(BFS). Použitím hladového algoritmu pro řešení problému k-center získáme nakreslení grafu v k dimenzích. Toto nakreslení nazveme high-dimensional embedding(HDE). Jako optimální počet center se jeví $m = k = 50$ bez ohledu na velikost a strukturu grafu.

2.2 PCA (Principal Component Analysis)

V předcházející části jsme ukázali, jak získat HDE. Nyní je tedy třeba nalézt reprezentaci HDE v 2D nebo 3D, která odhalí strukturu HDE daného grafu. Jako vhodná metrika pro nalezení vhodné reprezentace HDE se jeví maximalizace vzdálenosti mezi jednotlivými vrcholy G tak, aby bylo možné vrcholy jednoznačně odlišit. Je však nutné nalézt vhodné omezení pro možná zobrazení tak, abychom zabránili divergenci vrcholů do nekonečna. [10]

V této části tedy ukážeme, jak věrně zobrazit více-dimenzionální data ve 2D a 3D. Tyto zobrazovací techniky souhrnně nazýváme metodami pro redukci dimenze(RD). V praxi je využívaná celá řada těchto technik viz [7]. V našem případě však bude hlavním kritériem pro volbu metody její časová složitost - Jak bude ukázáno v 2.5 lze graf nakreslit za využití Hallovy energie s kvadratickou složitostí. Toto dává rozumnou rychlost pro grafy se zhruba 5 000 vrcholy, ale je velmi pomalé pro velmi velké grafy.

Jednou ze základních metod pro redukci dimenze je právě PCA. Buď X tvaru $m \times n$ matice pozorování. X se skládá z n m-dimenzionálních *centralizovaných* pozorování. PCA hledá navzájem kolmé lineární kombinace $w_1 \dots w_p$ [1] vektorů $x_1 .... x_n$ takové, že $s_i = w_i x$ má maximální rozptyl. Tedy $w_1 = w_{1,1} \dots w_{1,m}$ jsou koeficienty lineární kombinace vektorů $x_1 \dots x_n$ takové, že vektor $s_1 = w_1 x$ má největší rozptyl. Zjevně by však pro $x \to \infty$ rozptyl divergoval, a tedy zavedeme omezení $|w_i| = 1$. Koeficienty $w_2 = w_{2,1} \dots w_{2,m}$ přísluší lineární kombinaci, která je kolmá na w_1 a navíc vektor $s_2 = w_2 x$ má největší rozptyl $\forall_w w \perp w_1$.

Algoritmus pro nalezení PCA

Vstup: matice X tvaru $m \times n$ - reprezentující HDE grafu $G = (V, E)$

1. Centralizuj jednotlivá pozorování - $x_{i,j} = x_{i,j} - \frac{\sum_{k=1}^{n} x_{i,k}}{n}$
2. Spočítej kovariační matici $Cov = XX^T$ - Kovariační matice je tvaru $m \times m$
3. Nalezni dominantní (největší) vlastní čísla $\lambda_1 \geq \dots \geq \lambda_p > 0$ a jím příslušné vlastní vektory $U = u_1 \dots u_p$ matice Cov

[1] p je dimenze finálního prostoru, do kterého budeme pozorování zobrazovat - typicky tedy 2 nebo 3

4. Vypočítej projekci $X^T U$ - toto je hledané zobrazení HDE v p dimenzích.

Věta 2.4 *Výše uvedený algoritmus nalezne projekci, která maximalizuje*

$$\sum_{i,j \in V} \text{dist}(i,j) = u^T X X^T u$$

na množině $|v| = 1$ *– tuto projekci nazýváme* PCA-projekcí. *Hledat tedy budeme především vlastní vektory příslušné největším vlastním číslům kovariační matice.*

Důkaz: viz [10] a [19]

Zbývá ukázat, jak nalézt vlastní čísla kovariační matice. Tato matice je symetrická, a má tedy všechna vlastní čísla reálná. K nalezení dominantních vlastních čísel použijeme mocninné metody. Mocninná metoda vychází z pozorování, že pokud je matice A pozitivně semidefinitní a u_1 vlastní vektor příslušný dominantnímu vlastnímu číslu A , pak pro libovolný vektor x, pro který $u_1 x \neq 0$, konverguje řada $Ax, A^2x \ldots A^n x$ k u_1.

Mocninná metoda viz[10]

Vstup: Pozitivně definitní matice A

1. $\hat{u_i}$:= náhodný normovaný vektor
2. $u_i := \hat{u_i}$
3. Ortogonalizuj u_i vůči předchozím vlastním vektorům:
   for $j := 1$ **to** $i - 1$
   $u_i := u_i - (u_i u_j)$
4. $\hat{u_i} := A\hat{u_i}$
5. normalizuj $\hat{u_i}$
6. pokud $u_i \hat{u_i} > 1 - \epsilon$: $i := i + 1$, jdi na 1
7. pokud $u_i \hat{u_i} \leq 1 - \epsilon$: jdi na 2

Shrnutí a složitost

Algoritmus PCA pro kreslení grafů

1. Získej matici HDE X
2. Spočítej kovariační matici XX^T
3. Nalezni dominantní vlastní vektory kovariační matice
4. Projekce HDE podle získaných vlastních vektorů

K získání HDE je třeba času $O(m * (|V| + |E|))$ pro graf s jednotkovým ohodnocením hran, kde m je počet pivotů. Pro většinu grafů se jeví volba $m = 50$ jako optimální pro kvalitu výsledného nakreslení. Výpočet kovariační matice lze provést v čase $O(m^2 * |V|)$. Tento algoritmus produkuje nejlepší výsledky pro relativně řídké grafy. Pro řídké grafy je tedy nejvýznamnější výpočet kovariační matice. Pro husté grafy naopak převládá výpočet HDE.

Projekce mezi 2D a 3D nakreslením

Dalším využitím algoritmu PCA je nalezení vhodné projekce třídimenzionálního nakreslení grafu do 2D. Jak bylo dokázáno v 2.4 maximalizuje PCA projekce vzájemné vzdálenosti mezi jednotlivými vrcholy. Pokud nás tedy bude zajímat nakreslení grafu ve 2D a máme již jeho nakreslení ve 3D, stačí matici tohoto nakreslení vynásobit její transpozicí. Tak získáme nakreslení ve 3D.

2.3 Laplacian Preserving Projection - LPP

Hallova energie

Definice 2.4 *Buďte A, B čtvercové matice.* Zobecněným vlastním číslem *matice A vůči matici B nazveme číslo λ, pokud existuje vektor $x \neq \vec{0}$: $Ax = \lambda Bx$. Vektor x pak nazýváme* zobecněným vlastním vektorem.

Definice 2.5 *Buď $G = (V, E)$ graf a $w_G : E \to \Re^+$ ohodnocení jeho hran.*

Pak Laplacián L *grafu* G *je matice tvaru* $|V| \times |V|$ *definovaná následovně:*

$$l_{i,j} = \begin{cases} -w_G(i,j), \textit{ pokud } \ i,j \in E \\ \sum\limits_{\{i,k\} \in E} w_G(i,k), \textit{ pokud } i = j \\ 0 \textit{ jinak} \end{cases}$$

Definice 2.6 Matice vah W *grafu* G *je definována následovně:*

$$w_{i,j} = \begin{cases} w_G(i,j), \textit{ pokud } \ i,j \in E \\ 0 \textit{ pokud i=j - smyčky jsou zakázány} \\ 0 \textit{ jinak} \end{cases}$$

Definice 2.7 *Diagonální maticí* D *grafu* G *rozumíme matici* $D = W - L$.

Definice 2.8 (Hallova energie) *Hallovou energií* $E_H(G, \Phi)$ *nakreslení* Φ *grafu* G *rozumíme*

$$E_H(G, \Phi) = \frac{1}{2} \sum_{\{i,j\} \in E} w_{i,j} (\Phi(i) - \Phi(j))^2.$$

Předchozí definice odpovídá intuitivní představě, že čím je váha hrany e významnější, tím blíže by měli ve výsledném nakreslení být její vrcholy. Označme nyní $\Phi(i) = x_i$ a $X = \{x_1 \dots x_n\}$. V optimálním nakreslení by tedy měla být minimalizována délka hran. Je zjevné, že umístění všech vrcholů v počátku však není vhodným minimem, a proto je třeba definovat množinu, na které budeme Hallovu energii optimalizovat. Logickou volbou se jeví $xx^t = 1$ - tedy množina vektorů s normou 1.

Věta 2.5 *Platí, že* $E_H(G, \Phi) = XLX^T$ *a* $\min XLX^T$.

Důkaz: viz [15]. Roznásobením dostáváme:

$$E_H(G, \Phi) = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j} (x_i - x_j)^2 = \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}\, x_i^2 + \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}\, x_j^2 - \sum_{i,j=1}^{n} w_{i,j}\, x_i\, x_j$$

Protože je W symetrická, a tedy $\sum\limits_{i,j=1}^{n} w_{i,j}\, x_i^2 = \sum\limits_{i,j=1}^{n} w_{i,j}\, x_j^2$, platí:

$$\frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}\, x_i^2 + \frac{1}{2} \sum_{i,j=1}^{n} w_{i,j}\, x_j^2 - \sum_{i,j=1}^{n} w_{i,j}\, x_i\, x_j = \sum_{i,j=1}^{n} w_{i,j}\, x_i^2 - \sum_{i,j=1}^{n} w_{i,j}\, x_i\, x_j$$

Analýzou první sumy dostáváme:

$$\sum_{i,j=1}^{n} w_{i,j}\, x_i^2 = \sum_{i=1}^{n} \left(\sum_{j=1}^{n} w_{i,j} \right) x_i^2 = \sum_{i=1}^{n} l_{i,i}\, x_i^2$$

Z druhé sumy dostáváme,neboť $\forall i : w_{i,i} = 0$:

$$\sum_{i,j=1}^{n} w_{i,j}\, x_i\, x_j = \sum_{i,j=1;\, i \neq j}^{n} w_{i,j}\, x_i\, x_j = - \sum_{i,j=1;\, i \neq j}^{n} l_{i,j}\, x_i\, x_j$$

Dohromady tedy dostávame:

$$E_H(G, \Phi) = \sum_{i,j=1}^{n} w_{i,j}\, x_i^2 - \sum_{i,j=1}^{n} w_{i,j}\, x_i\, x_j = \sum_{i,j=1;\, i \neq j}^{n} l_{i,j}\, x_i\, x_j + \sum_{i=1}^{n} l_{i,i}\, x_i^2 = XLX^T$$

Věta 2.6 *Extrémy Hallovy energie získáme jako vlastní vektory Laplaciánu, příslušná vlastní čísla pak odpovídají hodnotě Hallovy energie v daném extrému.*

Důkaz: viz [15].

Odvození algoritmu

Naše metoda vychází z myšlenky, že zejména u symetrických grafů je graf indukovaný pivoty G^p dobrou aproximací původního grafu, a stačí tedy minimalizovat jeho Hallovu energii. Takto vytvořené nakreslení dává pro řadu grafů velmi přesnou aproximaci optimálního nakreslení původního grafu. Ohodnocení hran W^p grafu G^p definujeme následovně:

$w_{i,j}^p = d_{i,j}$, kde $d_{i,j}$ je délka nejkratší cesty mezi pivoty i, j v grafu G.

Laplacián a diagonální matici G^p označíme L^p a D^p. Je třeba nalézt obdobné omezení jako u původního grafu. Jako vhodná norma pro graf G^p se jeví $xDx^T = 1$. Tato norma rozmisťuje vrcholy do n-dimenzionálního elipsoidu, kde vrcholy s větší vahou leží blíže počátku. Naopak vrcholy s nižší vahou budou umístěny dále od počátku. Názornou analogií je zde model atomu – hmotné protony a neutrony se nachází v jádru, naopak lehké elektrony obíhají po eliptických drahách v elektronovém obalu. Nahradíme tedy problém minimalizace Hallovy energie původního grafu následujícím minimalizačním problémem pro graf G^p :

$$\min_{xD^p x^T = 1} xL^p x^T.$$

Definice 2.9 *Prostor spojitě diferencovatelných funkcí značíme* $\mathcal{C}$.

Definice 2.10 *Buď* $f(x)$ *z* $\mathcal{C}$ *skalární funkcí vektoru* x. Gradient f *definujeme jako n-dimenzionální vektor parciálních derivací, značíme* $fracdfdx$.

Věta 2.7 *Buď* S *symetrická matice a* $f(x) = x^T Sx$

Důkaz: viz [15]

Ukážeme nyní, že námi definovaný problém minimalizuje Hallovu energii grafu G^p.

Věta 2.8 *Extrémy předchozího minimalizačního problému získáme jako vlastní vektory následujícího zobecněného problému vlastních čísel, příslušná vlastní čísla pak odpovídají hodnotě Hallovy energie v daném bodě.*

Důkaz: Vyjdeme z věty o Lagrangeových multiplikátorech pro nalezení vázaného extrému. Minimalizujeme tedy $xL^p x^T$ vůči vazbě $xD^p x^T - 1 = 0$. Dostáváme tedy následující minimalizační problém

$$xL^p x^T - \lambda(xD^p x^T - 1),$$

kde λ je hledaný Lagrangeův multiplikátor. Derivací podle x viz 2.7 obržíme:

$$L^p x - \lambda D^p x = 0$$

$$L^p x = \lambda D^p x,$$

a navíc pro Hallovu energii G^p platí:

$$E_H(G^p, \Phi) = xL^p x^T = xD^p x^T.$$

Z výše uvedeného tedy vyplývá, že na rozdíl od PCA-projekce, zde budeme hledat nejmenší vlastní čísla. Nejmenší vlastní číslo λ_1 je ovšem rovno 0 a výsledný vlastní vektor odpovídá degenerovanému řešení $x = (\frac{1}{d_{1,1}} \dots \frac{1}{d_{n,n}})$, proto budeme hledat vlastní vektory $\lambda_2 \dots \lambda_4$ viz[9]. Je ovšem vhodné spočítat i další vlastní vektory, neboť nakreslení získané na základě těchto vektorů má vyšší estetickou kvalitu než to získané pomocí $\lambda_2 \dots \lambda_4$.

Výpočet zobecněných vlastních čísel

V této části ukážeme, jak převést problém nalezení zobecněných vlastních vektorů na problém nalezení klasických vlastních vektorů. Proces je velmi zjednodušen faktem, že D je diagonální, a vyhneme se tak definici pseudo-inverzní matice. V případě, že D není diagonální, lze postupovat obdobně a transformační algoritmus pro tento případ je popsán a vysvětlen v [11]. K vyřešení výsledného problému využíváme algoritmus QR implementovaný v knihovně ALGLIB viz[28]. Detailní přehled algoritmů pro nalezení vlastních lze nalézt v EIG001.

Definice 2.11 *Buď A pozitivně definitní čtvercová matice. Pak existuje $A^{\frac{1}{2}}$ horní trojúhelníková taková, že $A = A^{\frac{1}{2}}A^{\frac{1}{2}T}$. Tento rozklad matice A nazýváme* Choleského dekompozicí.

Mějme tedy $Lx = \lambda Dx$, kde D je diagonální matice a L je symetrická. Pak Choleského dekompozicí D dostáváme:

$$Lx = \lambda D^{\frac{1}{2}} D^{\frac{1}{2}} x.$$

Inverzní matici $D^{\frac{-1}{2}}$ k $D^{\frac{1}{2}}$ získáme invertováním prvků na diagonále v $D^{\frac{1}{2}}$. A tak dostáváme:

$$Lx = \lambda Dx$$
$$Lx = \lambda D^{\frac{1}{2}} D^{\frac{1}{2}} x$$
$$LD^{\frac{-1}{2}} D^{\frac{1}{2}} x = \lambda D^{\frac{1}{2}} D^{\frac{1}{2}} x$$
$$D^{\frac{-1}{2}} LD^{\frac{-1}{2}} D^{\frac{1}{2}} x = \lambda D^{\frac{1}{2}} x$$

Položme nyní $y = D^{\frac{1}{2}} x$ a získáme standardní problém zobecněných vlastních vektorů tvaru:

$$D^{\frac{-1}{2}} LD^{\frac{-1}{2}} y = \lambda y$$

Vlastní vektory původní úlohy získáme jako $x = D^{\frac{-1}{2}} y$.

Shrnutí algoritmu LPP

Algoritmus LPP pro kreslení grafů **Vstup:** graf G

1. Nalezni matici X příslušenou k HDE grafu G
2. Vytvoř Laplacián a Diagonální matici pivotového grafu G^p

3. Nalezni vlastní čísla $\lambda_2 \dots \lambda_4$ a vlastní vektory $u_2 \dots u_4$ zobecněného problému vlastních čísel $L^p x = \lambda D^p x$

4. Spočítej projekci $x_i = X u_i$

Složitost a srovnání s jinými metodami

V [14] je uvedena podobná metoda SSO (Subspace optimalization), avšak k získání Laplaciánu grafu G^p je použito relativně zdlouhavé násobení matic, které využívá Laplaciánu původního nekontrahovaného grafu. Složitost se tedy zvyšuje velmi rychle s růstem počtu hran grafu G. Naše metoda k získání Laplaciánu využívá pouze pivotových vrcholů, a je tedy zřetelně rychlejší než SSO, a dokonce rychlejší než PCA. Nejnáročnějším krokem PCA-projekce je u řídkých grafů právě výpočet XX^T.

Na rozdíl od SSO není potřeba násobit matice XLX^T, což ušetří $O(m|E|+ m^2 n)$ času viz [14]. Oproti PCA odpadá násobení XX^T, které je u řídkých grafů výpočetně náročnější než BFS. Zbývá tedy jen zpětná projekce, kterou lze ve 3D provést v čase $O(3 * m * n)$. Označme dobu potřebnout pro výpočet vlastních čísel matic řádu m QR(m). Pak asymptotická složitost našeho algoritmu je tedy $O(m * \mathrm{BFS} + 3 * m * n + \mathrm{QR}(m)) = O(m * \mathrm{BFS} + 3 * m * n) = O(m * (n + E))$ pro grafy s jednotkovým ohodnocením hran a $O(m * \mathrm{Dijkstra} + 3 * m * n + \mathrm{QR}(m)) = O(m * \mathrm{Dijkstra}) = O(m * (|V| \log |V| + |E|))$ pro grafy s nezáporně ohodnocenými hranami za použití Dijkstrova algoritmu a Fibonacciho haldy viz [17], implementované v Boost::Graph.

Metoda ACE [15] provádí shlukování grafu ve více krocích. Při každé iteraci se počet vrcholů grafu zmenší zhruba na polovinu. Hlavním rozdílem oproti našemu algoritmu však je fakt, že velikost finálního Laplaciánu u ACE závisí na velikosti původního grafu, což vede k relativně komplikovanému a pomalému výpočtu vlastních čísel potenciálně velké matice. Zejména při zobrazení grafu ve 3D je toto obzvláště citelné - je totiž třeba spočítat 3 vlastní vektory pomocí mocninné metody oproti 2 ve 2D. Na druhou stranu ACE nevyužívá HDE a může se tak vyhnout některým nedostatkům této metody – např. problémy u grafů s nízkým poloměrem.

Naše metoda je tedy rychlejší než ACE, PCA i HDL. U těchto metod v praxi nejvíce času trvá právě násobení matic viz [14] a [10], které u naší metody odpadá.

Graf	$\|V\|$	LPP v sekundách	HDE	Zpětná projekce
Grid 100×100	10 000	0.844	0.687	0.125
Grid 200×200	40 000	4.282	3.832	0.657
Grid 400×400	160 000	17.28	15.22	1.86

Tabulka 2.1: Doba běhu algoritmu LPP pro čtvercové mřížky.

Poté, co jsme algoritmus implementovali, objevili jsme podobnou myšlenku v [2] se shodnou složitostí. V tomto algoritmu je k definici Laplaciánu navíc využito vzorkování řádek a sloupců matice X. Výsledný minimalizační problém využívá jinou metriku kontrahovaného prostoru - xWx^T. Kvůli této definici je zde zapotřebí výpočet pseudoinverzní matice, což vede k určité numerické nestabilitě, která je sice odstranitelná, ale jak ukazuje naše metoda, lze se jí zcela vyhnout. V naší metodě je totiž D diagonální a Choleskéhо rozklad je tedy triviální. Problém zobecněných vlastních čísel lze tedy přímo redukovat na klasický problém vlastních čísel, jak bylo ukázáno výše. Rozhodně by bylo zajímavé provést detailní srovnání obou metod. Příklady uvedené v [2] se zdají být velmi podobné našim výsledkům.

Porovnání PCA a LPP

Algoritmy jsme porovnávali na počítači s procesorem AMD64 3500 běžícím na taktovací frekvenci 2.21 GHz s 1GB RAM. Počet pivotů pro vytvoření HDE matice byl ponechán na 50.

Nejprve prakticky demonstrujeme, že algoritmus LPP skutečně pracuje v lineárním čase vůči počtu vrcholů. Pro demonstraci použijeme čtvercové mřížky o velikosti $100 \times 100, 200 \times 200$ a 400×400. Vidíme, že pro čtyřnásobný vstup je doba běhu programu zhruba čtyřikrát delší. Bohužel se zdá, že námi využívaná knihovna Boost::Graph je málo výkonná ve srovnání s optimalizovanými ad hoc řešeními, a proto je i naše implementace relativně pomalá. Na druhou stranu je aplikace i tak dostatečně rychlá pro vykreslení grafů s velikostí řádově 100 000 vrcholů v rozumném čase, a to pro praktickou demonstraci vlastností LPP zcela postačuje.

Pro srovnání uveďme dobu běhu algoritmu PCA pro výše zmíněné grafy. Vidíme, že doba běhu algoritmu PCA je zhruba dvojnásobná oproti LPP. Většina času je dle očekávaní využita pro výpočet kovariační matice. Dodejme, že obě metody naleznou správné nakreslení mřížky.

Po lehké rozcvičce je na čase demonstrovat jednu velmi příjemnou vlast-

Graf	$\|V\|$	PCA v sekundách	HDE	Kovariční matice
Grid 100×100	10 000	2.593	0.843	1.719
Grid 200×200	40 000	13.13	4.745	8.464
Grid 400×400	160 000	36.12	13.11	22.59

Tabulka 2.2: Doba běhu algoritmu PCA pro čtvercové mřížky.

nost algoritmů založených na HDE. Z grafu lze náhodně odebírat hrany a výsledné nakreslení zůstane nadále kvalitní. Na rozdíl od fyzikálních algoritmů, kde se odebíráním hran narušuje konvergence, zde zůstavají nejkratší cesty v podstatě nezměněny. Tím pádem je i výsledné nakreslení kvalitní. Na obrázku 2.14 vidíte krychlovou mřížku o hraně velikosti 11, ze které bylo náhodně odebráno 20 % hran. Tato mřížka byla vykreslena algoritmem LPP vlastními čísli 2, 3, 4. Projekce PCA nedokáže tento graf uspokjivě vykreslit.[2] Tento graf také demonstruje, že ne vždy jsou vlastní čísla 1, 2, 3 ta nejlepší. Další úvaha nás vede k tomu, že ne vždy je hladový algoritmus pro výběr pivotů zcela vhodný. Představme si například mřížku, ke které bylo přidáno 50 izolovaných vrcholů. Hladový výběr bude brát izolované vrcholy jeden po druhém a výsledek bude pochopitelně zhoubný. Naopak náhodným výběrem můžeme v takovém případě získat rozumné nakreslení.

Naopak u grafů, kde jsou náhodné hrany přidávány, vzniká problém. Hrany, které spojují jinak odlehlé části grafu, poruší i celé HDE. Je třeba si uvědomit, že v případě, že existují hrany, které spojují odlehlé části grafu, je metrika indukovaná nejkratšími cestami značně divoká a obtížně představiltená. Jako zajímavý problém se nabízí otázka charakterizace prostorů generovaných metrikou nejkratších cest. Dále platí, že metodami založenými na HDE nelze kreslit stromy. *Koren se domnívá, že je to způsobeno tím, že prostor generovaný HDE stromů má vysokou dimenzi.*[10] Dalšími grafy, které nelze pomocí HDE věrně nakreslit, jsou husté grafy. Například HDE úplného grafu je matice se samými jedničkami, jen pivotové vrcholy mají jednu nulu. Jakákoliv projekce HDE zjevně musí mapovat všechny nepivotové vrcholy do jednoho bodu.

Na obrázku 2.8 vidíte Sierpinského fraktál hloubky 9 nakreslený algoritmem LPP. Velmi zajímavé je perfektní 3D nakreslení grafu `finap512` 2.10 na více než 70 000 vrcholech a s 200 000 hranami ve 3D s využitím vlastních čísel 0, 1 a 2. Tento graf demonstruje, že LPP přesně zobrazí globální kruhovou

[2]Tento graf najdete po instalaci v adresáři graphs pod jménem cube_rand.graphml.

topologii, a zároveň věrně zobrazí i detailní věžovité struktury na obvodu. LPP je velmi přesná, pokud jde o nalezení věrného globálního nakreslení, jak demonstruje globální zobrazení grafu `4elt2` 2.1. Oproti stejnému nakreslení metodou PCA 2.2 jsou u LPP hrany nakreslení raketoplánu zaoblené. Na druhou stranu, Hallova energie žádným způsobem nebrání shlukování vrcholů, a proto jsou vrcholy ve 2D nakreslení metodou LPP rozmístěny nerovnoměrně. Naopak PCA rozmísťuje vrcholy lokálně symetricky. Lokální symetrie však zde není tak významná, podstatný je především globální tvar křídla.

Algebraické metody jsou velmi vhodné pro využití ke 3D zobrazení, jak ukazuje velmi pěkné nakreslení téhož grafu na obrázku 2.3. Na rozdíl od fyzikálních metod, které vznikly primárně pro 2D, vychází algebraické metody z prostorů vyšší dimenze. Zobrazení ve 3D odstraní nedostatky obou porovnávaných metod. Nedochází zde k žádnému shlukování vrcholů. Důvodem je, že třetí vlastní číslo zde podává dodatečnou informaci právě o okrajových částech grafu. Předchozí pozorování platí nejen pro graf `4elt2`, ale i řadu dalších.

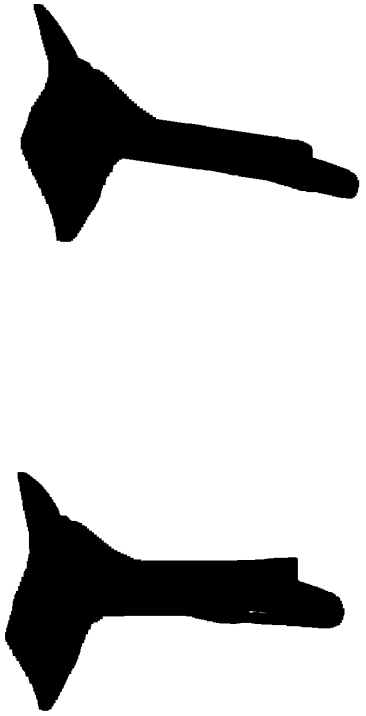
Oblíbeným grafem testerů byl graf s názvem `crack`, jehož nakreslení algoritmem LPP opět vykazuje vysokou míru symetrie (obrázek 2.11). Ilustrace 2.4 a 2.5 kontrastují nakreslení grafu `big` pomocí LPP a PCA ve 3D. Obě metody zde naleznou podobná nakreslení (pro PCA jsme použili vlastní čísla 1, 2 a 3). LPP je ovšem o 50 % rychlejší. Konečně ilustrace 2.6 a 2.7 porovnávají nakreslení grafu `3elt` oběma metodami. Zde je naopak zřejmým vítězem PCA projekce, která zobrazí lokální symetrii přesněji nežli LPP.

Příklady ukazují, že LPP produkuje globálně symetrická nakreslení. Při zobrazení ve 2D dimenzi vykazují však tendenci ke shlukování vrcholů, která vyplývá z definice Hallovy energie. Naopak PCA projekce částečně obětuje globální symetrii ve prospěch symetrie lokální a je na srovnatelných datech pomalejší než LPP. V tabulce 2.3 vidíte srovnání doby běhu obou algoritmů na některých zde uvedených grafech. Je vidět, že ve většině případů je LPP až o polovinu rychlejší než PCA projekce, neboť nehledá kovariační matici. Prezentované grafy byly získany z [26] a [24].

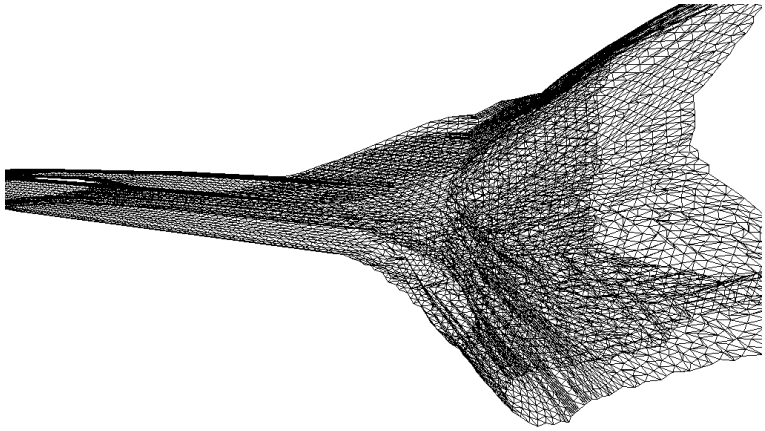
Tabulka 2.3: Porovnání běhu algoritmů

Graf	$\|V\|$	$\|E\|$	PCA v sekundách	LPP v sekundách
4elt2	11143	65636	3.047	2.000
big	15606	45878	5.751	2.062
finap512	74752	261120	30.67	11.96
3elt	4720	27444	1.204	0.656
crack	10240	30380	2.875	1.360

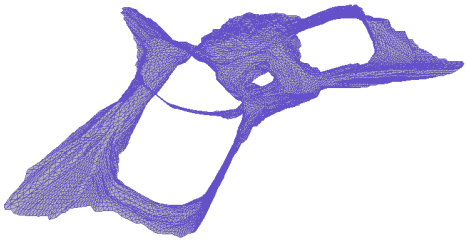
Obrázek 2.1: Globální nakreslení grafu *4elt2* pomocí LPP.



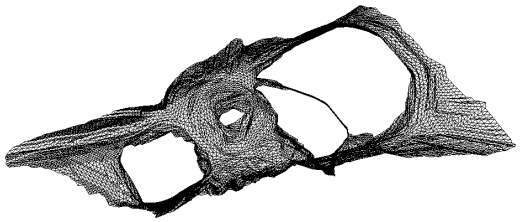
Obrázek 2.2: Globální nakreslení grafu *4elt2* pomocí PCA.



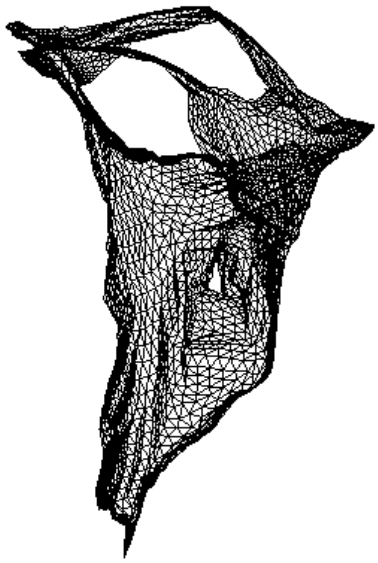
Obrázek 2.3: Nakreslení grafu *4elt2* pomocí LPP ve 3D.



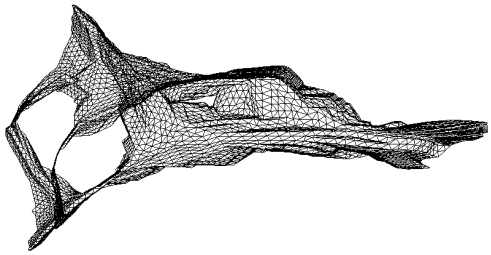
Obrázek 2.4: Nakreslení grafu *big* metodou PCA.



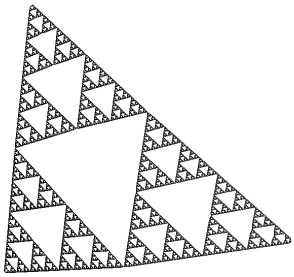
Obrázek 2.5: Nakreslení grafu *big* metodou LPP.



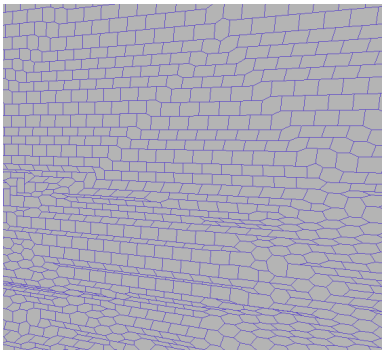
Obrázek 2.6: Nakreslení grafu *3elt* metodou PCA.



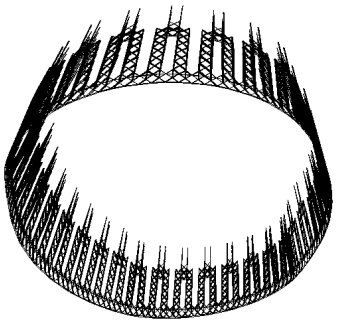
Obrázek 2.7: Nakreslení grafu *3elt* metodou LPP.



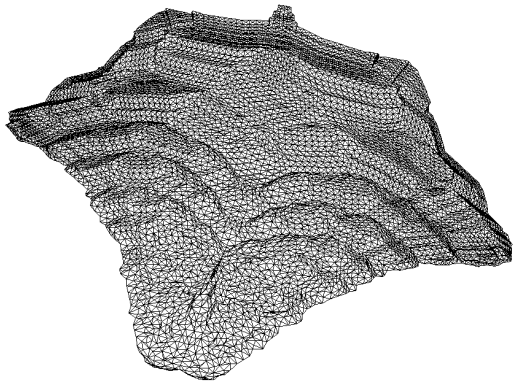
Obrázek 2.8: Sierpinského hvězda hloubky 9 metodou LPP.



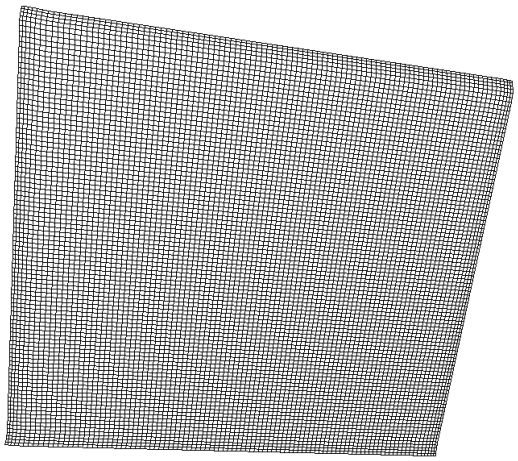
Obrázek 2.9: Graf *whitakerdual* nakreslený metodou LPP.



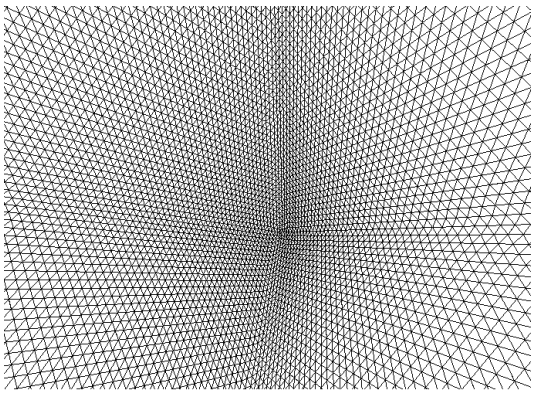
Obrázek 2.10: Graf Finap512 metodou LPP – pomocí vlastních čísel 0, 1 a 2.



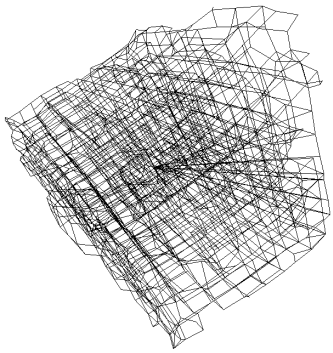
Obrázek 2.11: Graf crack nakreslený metodou LPP ve 3D.



Obrázek 2.12: Mřížka 100×100 3D LPP-projekce. na obrázku patrné určité interference.



Obrázek 2.13: Pohled do centra grafu sphere, LPP projekce ve 3D.



Obrázek 2.14: Krychle $11 \times 11 \times 11$ pomocí vlastních čísel 2, 3 a 4 metodou LPP.

Kapitola 3

Program GraphDraw

Součástí práce je program GraphDraw, který slouží k editaci a vizualizaci grafů jak ve 2D, tak ve 3D. Pro vizualizaci ve 3D je využit rychlý engine Irrlicht. Umožňuje dále ukládat grafy včetně jejich nakreslení ve formátu GRAPHML. Aplikace je psána pro operační systém Windows.

3.1 Instalace

Vložte CD do mechaniky. Otevřte mechaniku a dvakrát klikněte na ikonu setup.exe. Postupujte podle instrukcí instalátoru a vyberte adresář, kam chcete GraphDraw uložit. Program se automaticky nainstaluje. Pokud máte potíže s instalací, extrahujte soubor graph_draw.zip do libovolné složky, kde máte právo zápisu.

3.2 Spuštění programu

Program můžete po instalaci spustit dvojitým kliknutím na ikonu graph_draw přímo z Plochy. Alternativně dvakrát klikněte na soubor graph_draw.exe v adresáři, kam jste program nainstalovali. V následující části popíšeme jaké možnosti pro vizualizaci GraphDraw nabízí. Všechny možnosti jsou dostupné z hlavního menu aplikace. Kromě dále uvedených metod umožňuje GraphDraw ukládání grafů v několika formátech a základní editační funkce.

3.3 Draw2D – Kreslení grafů ve 2D

Random Layout – náhodné nakreslení

Rozmístí vrcholy náhodně na plochu

Radial Layout – umístění vrcholů na kružnici

Umístí vrcholy na kružnici. Parametr *Radius in Pixels* udává poloměr kružnice v pixelech.

Spring Layout – Nakreslení pomocí algoritmu Kawada-Kamai

Nalezne nakreslení grafu pomocí algoritmu Kawada-Kamai, který je relativně pomalý a není vhodné ho použít pro grafy s více než 50 uzly. Prvním parametrem algoritmu je *spring power* - síla pružiny. Zvýšení spring power dosáhnete rychlejší konvergence algoritmu za cenu kvality výsledného nakreslení. *Layout tolerance* udává maximální energii potřebnou pro ukončení programu.

LPP Drawing – 2D nakreslení pomocí LPP

Parametr *Center Count* udává počet pivotů využitých k nalezení HDE. Vyšší hodnota může částečně vylepšit výsledné nakreslení, na druhou stranu vede k pomalejšímu běhu algoritmu. Pro většinu grafů je standardní hodnota 50 optimální.

Parametry *Eigenvalue for x-coordinate* a *Eigenvalue for y-coordinate* udávají, která vlastní čísla mají být použita pro x-ovou respektive y-ovou souřadnici výsledného nakreslení. V některých případech je vhodné zaexperimentovat s nalezením správné volby těchto parametrů, neboť další vlastní čísla mohou odkrýt významné detaily grafu, ale pro většinu grafů je standardní nastavení vyhovující. Pokud zatrhnete volbu *Weighted*, bude uvažováno Vámi definované ohodnocení hran – viz část o editování hran. V opačném případě budou všechny hrany uvažovány jako jednotkové. Po kliknutí na tlačítko *OK*, nalezne program nakreslení současného grafu pomocí algortimu LPP ve 2D.

PCA Drawing – 2D nakreslení pomocí PCA-projekce

Combobox *Covariance Matrix Type* udává, jakým způsobem má být získána kovariační matice. Pokud zvolíte *High Dimensional*, bude k nalezení kovariační matice použito vysoko-dimenzionální nakreslení grafu. Pokud zvolíte *3D Projection*, je jako základ kovariační matici bráno současné 3-dimenzionální nakreslení. Parametr *Center Count* udává počet pivotů využitých k nalezení HDE. Vyšší hodnota může částečně vylepšit výsledné nakreslení, na druhou stranu vede k pomalejšímu běhu algoritmu. Pro většinu grafů je standardní hodnota 50 optimální.

Parametry *PC for x-coordinate* a *PC for y-coordinate* udávají, které hlavní komponenty mají být použity pro x-ovou respektive y-ovou souřadnici výsledného nakreslení. V některých případech je vhodné zaexperimentovat s nalezením správné volby těchto parametrů, neboť další vlastní čísla mohou odhalit zajímavé struktury grafu, ale pro většinu aplikací je standardní nastavení vyhovující. Pokud zatrhnete volbu *Weighted*, bude uvažováno Vámi definované ohodnocení hran – viz část o editování hran. V opačném případě budou všechny hrany uvažovány jako jednotkové. Po kliknutí na tlačítko *OK*, nalezne program nakreslení současného grafu pomocí PCA-projekce ve 2D.

3.4 Draw3D – Kreslení grafů ve 3D

Orthogonal Grid Layout - 7 bends

Nalezne ortogonální nakreslení grafu se 7 zlomy kompaktním algoritmem. Graf musí být souvislý a jeho maximální stupeň nesmí přesáhnout 6.

LPP Projection – 3D nakreslení pomocí LPP

Parametr *Center Count* udává počet pivotů využitých k nalezení HDE. Vyšší hodnota může částečně vylepšit výsledné nakreslení, na druhou stranu vede k pomalejšímu běhu algoritmu. Pro většinu grafů je standardní hodnota 50 optimální.

Parametry *Eigenvalue for x-coordinate, Eigenvalue for y-coordinate* a *Eigenvalue for z-coordinate* udávají, která vlastní čísla mají být použita pro x-ovou,y-ovou a z-ovou souřadnici výsledného nakreslení. V některých případech je vhodné zaexperimentovat s nalezením správné volby těchto parametrů, neboť další vlastní čísla mohou odkrýt významné detaily grafu, ale

pro většinu grafů je standardní nastavení vyhovující. Pokud zatrhnete volbu *Weighted*, bude uvažováno Vámi definované ohodnocení hran – viz část o editování hran. V opačném případě budou všechny hrany uvažovány jako jednotkové. Po kliknutí na tlačítko *OK*, nalezne program nakreslení současného grafu pomocí algortimu LPP ve 3D.

PCA Projection – 3D nakreslení pomocí PCA-projekce

Parametr *Center Count* udává počet pivotů využitých k nalezení HDE. Vyšší hodnota může částečně vylepšit výsledné nakreslení, na druhou stranu vede k pomalejšímu běhu algoritmu. Pro většinu grafů je však standardní hodnota 50 optimální.

Parametry *PC for x-coordinate, PC for y-coordinate* a *PC for z-coordinate* udávají, které hlavní komponenty mají být použity pro x-ovou,y-ovou respektive z-ovou souřadnici výsledného nakreslení. V některých případech je vhodné zaexperimentovat s nalezením správné volby těchto parametrů, neboť další komponenty mohou odhalit zajímavé struktury grafu, ale pro většinu aplikací je standardní nastavení vyhovující. Pokud zatrhnete volbu *Weighted*, bude uvažováno Vámi definované ohodnocení hran – viz část o editování hran. V opačném případě budou všechny hrany uvažovány jako jednotkové. Po kliknutí na tlačítko *OK*, nalezne program nakreslení současného grafu pomocí PCA-projekce ve 3D.

Force Embedding – 3D nakreslení pomocí fyzikálního modelu

Nalezne nakreslení aktuálního grafu ve 3D pomocí Fruchterman-Reingoldova algoritmu, algoritmus se nehodí pro velké grafy s více než 200 vrcholy. Prvním parametrem je multiplikátor přitažlivé síly hran *Hook Force Multiplier.* Druhým parametrem je multiplikátor odpudivé síly mezi vrcholy *Electrical Force Multiplier.* Do pole *number of iterations* zadejte požadovaný počet iterací. Zatrhnete-li hodnotu *Use current 3D position* bude použito současné umístění vrcholů ve 3D jako iniciální konfigurace algoritmu. Toto nastavení lze využít pro vylepšení výsledků dosažených jinými algoritmy – například LPP projekcí. V opačném případě budou vrcholy nejprve rozmístěny náhodně v prostoru. Pokud zatrhnete volbu *Generate 2D Projection* bude výsledné nakreslení ve 3D převedeno do 2D pomocí algoritmu PCA.

3.5 Ovládání kamery a volba vykreslovacího režimu

Uprostřed na liště vidíte combobox označený *Drawing style*. Pomocí tohoto ovládacího prvku přepínáte mezi jednotlivými režimy kreslení. Pro editování grafu slouží 2D režim *2D Straight Line*. Dále je k dispozici režim kreslení ve 3D rovnými čarami*3D Straight Line* a ortogonální kreslení*Orto 3D*, které Vám dávají plnou kontrolu nad kamerou.

V režimu 3D a při ortogonálním kreslení máte plnou kontrolu nad úhlem pohledu na vykreslovaný graf. Pokud držíte levé tlačítko myši, můžete kamerou libovolně posunovat. Pro rotaci kamery přidržte pravé tlačítko myši. Pro přiblížení kamery držte obě tlačítka myši současně.

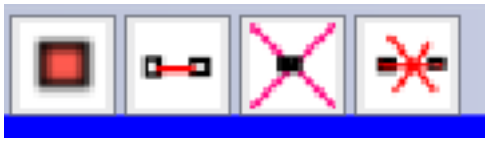
V režimu 2D můžete posunovat graf pomocí šipek a kláves WASD. Stisknutím klávesy 'O' oddálíte graf. Naopak stisknutím klávesy 'I' přiblížíte graf.

3.6 Editace grafu

Pro editaci grafu zvolte režim vykreslování ve 2D. Pokud jste dosud graf nevykreslili v režimu 2D, zvolte libovolné nakreslení grafu ve 2D. Na liště vlevo vidíte čtyři tlačítka jako na obrázku 3.1. Pokud podržíte nad těmito tlačítky kurzor objeví se text nápovědy.

Po stisknutí první ikony *(Add Vertex)* můžete přidávat vrcholy kliknutí levým tlačítkem na plochu hlavního okna. Pokud kliknete na existující vrchol objeví se menu, ve kterém můžete upravit vlastnosti daného vrcholu.

Editace vlastností vrcholu



Obrázek 3.1: Tlačítka pro editaci grafu.

U vrcholů můžete v záložce *Color* měnit jejich barvu. Barva je zadávána ve formátu RGB. Zadáte-li $(0, 0, 0)$, bude vrchol černý. Pokud zadáte

$(255, 255, 255)$, bude vrchol bílý. V záložce *Position in 2D and Weight* můžete upravit pozici vrcholu ve 2D a nastavit jeho ohodnocení, které je využito pro kreslení pomocí fyzikálních algoritmů ve 3D.

Po stisknutí druhé ikony *(Add Edge)* můžete přidávat hrany mezi existujícími vrcholy. Klikněte na první vrchol levým tlačítkem a následně klikněte levým tlačítkem na koncový vrchol Vámi požadované hrany. Pokud mezi Vámi vybranými vrcholy hrana již existuje, objeví se okno pro editaci vlastností zvolené hrany.

Editace vlastností hran

U hrany můžete měnit její barvu. Podobně jako u vrcholů je barva zadávána ve formátu RGB. Zadáte-li $(0, 0, 0)$, bude hrana černá. Pokud zadáte $(255, 255, 255)$, bude hrana bílá. Položka *Weight* slouží k nastavení ohodnocení hrany, které je využito pro kreslení pomocí fyzikálních algoritmů ve 3D i 2D, vážené PCA a LPP projekce.

Po stisknutí třetí ikony *(Remove Vertex)* můžete odebírat vrcholy z existujího grafu kliknutím levým tlačítkem na vrchol, který si přejete odebrat.

Poslední tlačítko *(Remove Edge)* slouží k odebírání hran z grafu. Postupujte stejně jako u přidávání hran. Nejprve zvolte první vrchol hrany levým tlačítkem myši, následně zvolte koncový vrchol stejným způsobem.

Kapitola 4

Závěr

V práci jsme demonstrovali využití algoritmů pro redukci dimenze pro kreslení grafů na příkladu PCA-projekce. Popsali jsme také, jak nalézt vhodnou projekci nakreslení grafu z 3D do 2D pomocí PCA-projekce.

Zejména jsme však prezentovali nový algoritmus pro kreslení rozsáhlých grafů s více než 10 000 vrcholy. Teoreticky i praktickou implementací jsme demonstrovali jeho lineární složitost a porovnali ji s ostatními současnými algoritmy. Naše výsledky ukazují, že nový algoritmus je ve srovnání s PCA projekcí o více než 50 procent rychlejší pro řídké grafy. Oproti algoritmu prezentovanému v [2] je náš algoritmus implementacně značně jednodušší a vyhýbá se numerickým problémům, které v původním algoritmu nastávají. Kvalita výsledných nakreslení je vysoká, zejména pokud se jedná o zobrazení globální symetrie. Pokud je zapotřebí nalézt precizní lokální nakreslení, je vhodné kombinovat naši techniku s některým z fyzikálních algoritmů.

Dále jsme vytvořili aplikaci, která implementuje kromě PCA a LPP projekcí i některé další metody pro kreslení grafů jako například fyzikální modely a orotogonální kreslení pomocí lomených čar ve 3D. Tato aplikace umožňuje uživateli využít veškeré výhody třídimenzionálního nakreslení jako zoomování a rotaci kamery.

Otevřená zůstává otázka, zda lze nalézt nejkratší cesty v grafu pro k zdrojových vrcholů rychleji než v čase O(k*Dijkstra). Nalezení takového algoritmu by vedlo ke zrychlení našeho algoritmu pro ohodnocené grafy.

Další výzvou je bližší charakterizace prostorů generovaných metrikou HDE pro různé třídy grafů. Z naší práce vyplývá, že algebraické metody jsou na rozdíl od fyzikálních metod robustní v případě, že z grafu jsou odebírány hrany. Na druhou stranu fyzikální metody se zlepšují, pokud do grafu ně-

které hrany doplníme. Pokud takto přidané hrany spojí jinak odlehlé části grafu, přestávají algebraické metod založené na grafové vzdálenosti fungovat.

Poznamenejme, že naši techniku lze využít k extrémně rychlému odhadu dominantních vlastních čísel libovolné matice. Nabízí se tedy podstatně širší aplikace než pouhé kreslení grafů, neboť vlastní čísla jsou využívána např. pro nalezení optimálního rozložení sil působících na mostní konstrukce. Další významnou aplikací algoritmů pro kreslení grafů ve 3D je rozklad meshí 3D modelů viz [30] a [24].

Literatura

[1] P. K. Agarwal, *Lecture notes on Geometric Optimization*, dostupné na `http://www.cs.duke.edu/courses/spring07-/cps296.2/scribe_notes/lecture14.pdf¨`

[2] A. Civril, M. Magdon-Ismail and E. Bocek-Rivele, *SSDE: Fast Graph Drawing Using Sampled Spectral Distance Embedding*, Proceedings of 14th Int. Symp. Graph Drawing (GD'06), 2006, str. 30-41

[3] R. Cohen, P. Eades, T. Lin, F. Ruskey et. al, *Three diminsional drawing*, Proceeding of 12th Annual ACM Symposium on Computational Geometry, Springer Verlag, str. 319-328, 1996

[4] J. Edmonds, *Paths, trees, and flowers*, Canadian Journal of Mathematics, č. 17, str. 449-467, 1965

[5] J. Felcman, Skripta z numerické matematiky, dostupné na `http://www.karlin.mff.cuni.cz/∼felcman/nm.pdf`

[6] R.W. Floyd, *Algorithm 97: Shortest Path*, Communications of the ACM, 1962

[7] I. K. Fodor, *A survey of Dimesionality Reduction Techniques*, dostupné na `https://computation.llnl.gov/casc/sapphire/pubs-/148494.pdf`

[8] P. Hall, *On representation of subsets*, Journal of London Math Society, č. 10, str. 26-30, 1930

[9] K. M. Hall, *An r-dimensional Quadratic Placement Algorithm*, Management Science, č. 17, str. 219-229, 1970

[10] D. Harel and Y. Koren, *Graph Drawing by High-Dimensional Embedding*, Proceedings of 10th Int. Symp. Graph Drawing (GD'02), Lecture Notes in Computer Science, Vol. 2528, Springer Verlag, str. 207-219, 2002

[11] X. He, P. Niyogi, *Locality Preserving Projections*, The University of Chicago, dostupné na `http://www.cs.uchicago.edu/files/-tr_authentic/TR-2002-09.pdf`

[12] D. S. Hochbaum, *Approximation Algorithms for NP-hard problems*, PWS Publishing, 1999

[13] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985

[14] Y. Koren, *Graph Drawing by Subspace Optimization*, Proceedings of 6th Joint Eurographics - IEEE TCVG Symp. Visualization (VisSym '04), Eurographics, str. 65-74, 2004

[15] Y. Koren, L. Carmel and D. Harel, *ACE: A Fast Multiscale Eigenvector Computation for Drawing Huge Graphs*, Proceedings of IEEE Information Visualization 2002 (InfoVis'02), IEEE, str. 137-144, 2002

[16] C. Lanius, *The Sierpinski Fractal*, dostupné na `http://math.rice.edu/~lanius/fractals/`

[17] J. Matoušek, J. Nešetřil, *Kapitoly z diskrétní matematiky*, Nakladatelství Karolinum, 2003

[18] A. Papakostas, Ioannis G. Tollis
Incremental Orthogonal Graph Drawing In Three Dimensions, dostupné na `http://www.utdallas.edu/~tollis-/papers_ortho.html`, 1997

[19] K. Pearson, *On Lines and Planes of Closest Fit to Systems of Points in Space*, Philosophical Magazine 2 (6) : str. 559–572, 1901

[20] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C++: Scientific Art of Computing*, Cambridge University Press, 2007

[21] A. J. Quigley and P. Eades, FADE: Graph Drawing, Clustering and Visual Abstraction, Proceedings of Graph Drawing, 2000

[22] A. Schrijver, Bipartite edge-coloring in $O(maxdeg*n)$, SIAM J. Comput., č. 28/3, str. 841-846, 1998

[23] W. T. Tutte, *How to draw a graph*, Proceedings of London Mathematical Society, č. 13, str. 743-768, 1963

[24] C. Walshaw, University of Greenwitch Partitioning Archive, *http://staffweb.cms.gre.ac.uk/∼wc06/partition/*

[25] D. Wood, *On higher-dimensional orthogonal graph drawing*, Proceedings of CATS'97, Computing: The Australasian Theory Symposium, str. 3-8, 1997

[26] University of Paderborn, Graph Collection, `http://wwwcs.uni-paderborn.de/fachbereich/AG-/monien/RESEARCH/PART/graphs.html`

[27] Boost Graph Library, Adjacency List Documentation, `http://www.boost.org/doc/libs/1_34_1/libs/-graph/doc/using_adjacency_list.html`

[28] V. Bystritsky, S. Bochkanov, The ALGLIB library – symmetric eigenvector problem, dostupné na `http://www.alglib.net/eigen-/symmetric/symmevd.php`

[29] V. Bystritsky, S. Bochkanov, The ALGLIB library – generalized symmetric eigenvector problem, dostupné na http://www.alglib.net/eigen/symmetric/generalizedsymmevd.php

[30] `http://www-unix.mcs.anl.gov/sumaa3d/PARS/partition.html`

[31] Boost libraries homepage `http://www.boost.org/`

[32] Boost Library License, `http://www.boost.org/LICENSE_1_0.txt`

[33] Documentation of the Boost::Graph Library `http://www.boost.org/doc/libs/1_34_1/libs/graph/doc/index.html`

[34] Documentation of the Boost::Thread Library
`http://www.boost.org/doc/libs/1_34_1/doc/html/thread.html`

[35] Boost Instalation Guide
`http://www.boost.org/doc/libs/1_34_1/more/getting_started/index.html`