TODO: nekonzistence v n a N TODO: udelat poradek v citacich

Kapitola 1

Grafy, stromy

TODO: nejaky hezky obrazky? V této kapitole definujeme graf jakožto matematickou strukturu, popíšeme základní pojmy týkající se grafů a nastíníme možné vztahy mezi grafem a maticí. Dále definujeme strom, jakožto speciální případ grafu. Terminologie je převzata z [18]

1.1 Základní grafová terminologie

Definice 1. Mějme množinu V a množinu $E = \{\{u, v\} \,|u, v \in V\}$. Uspořádanou dvojici $G := (V, E)$, nazveme neorientovaný graf. Množinu V nazýváme množinou vrcholů grafu G, jejím prvkům říkáme vrcholy, množinu E nazýváme množinou hran grafu G, jejím prvkům říkáme hrany. Prvky hrany e označujeme jako vrcholy incidentní hraně e nebo koncové body hrany e. Říkáme, že hrana $e = \{v, w\}$ spojuje vrcholy v a w.

Pokud neuvažujeme hrany jako nejvýše dvouprvkové množiny vrcholů, ale jako uspořádané dvojice (u, v), nazýváme odpovídající graf **orientovaný**. Obvykle uvažujeme orientované a neorientované grafy zvlášť, ale je možné uvažovat i jejich kombinaci. Graf, v němž se vyskytují jak orientované tak neorientované hrany, nazýváme **smíšený**. Řekneme, že neorientovaný graf G je **úplný**, pokud $\forall u, v \in V$ $(\{u, v\} \in E)$.

Povšimněme si, že v definici grafu není vyloučen případ, kdy jsou oba koncové body hrany shodné. Hrana je pak jednoprvkovou množinou a nazýváme ji **smyčkou** v grafu.

Poznámka 1. V neorientovaném grafu $G = (V, E)$ platí, že jeho množina hran E je podmnožinou $\binom{V}{2} \cup V$, kde $\binom{V}{2}$ značí množinu všech dvouprvkových podmnožin množiny V. V orientovaném grafu $H = (W, F)$ je F podmnožinou množiny $W \times W$, tj. všech uspořádaných dvojic vrcholů z W.

Stupněm vrcholu $v \in V$ rozumíme počet vrcholů spojených s vrcholem v, značíme $d(v)$. Množinu všech vrcholů, které jsou v grafu G spojeny s vrcholem v značíme $\mathrm{adj}_G(v)$. Pokud je z kontextu jasné o jaký graf se jedná, dolní index vynecháváme.

Definice 2. Podgrafem grafu G nazveme libovolný graf $H = (W, F)$ který splňuje: $W \subseteq V$, $F \subseteq E$ a všechny vrcholy incidentní hranám z F náleží do W. Úplný podgraf grafu

G nazýváme **klikou** v grafu G. Podgrafem grafu G **indukovaným** množinou vrcholů W nazveme takový podgraf G, který obsahuje všechny hrany grafu G, jejichž oba koncové body náleží do W, značíme $G(W)$.

Definice 3. Mějme graf $G = (V, E)$ a zobrazení $\omega : V \to \mathbb{R}$, resp. $c : E \to \mathbb{R}$. Přidáním zobrazení ω, resp. c ke grafu G dostaneme graf, který nazýváme **ohodnocený**, resp. **vážený** reálným ohodnocením.

Definice 4. Mějme neorientovaný graf $G = (V, E)$, nechť k je přirozené číslo. Posloupnost vrcholů $(v_i)_{i=1}^{k}$ nazveme **sledem** v grafu G, pokud $\forall i \in \{1, ..., k\}(\{v_{i-1}, v_i\} \in E)$. Pokud navíc $\forall i, j \in \{1, ..., k\}(i \neq j \Rightarrow v_i \neq v_j)$, nazveme posloupnost $(v_i)_{i=1}^{k}$ **cestou**.

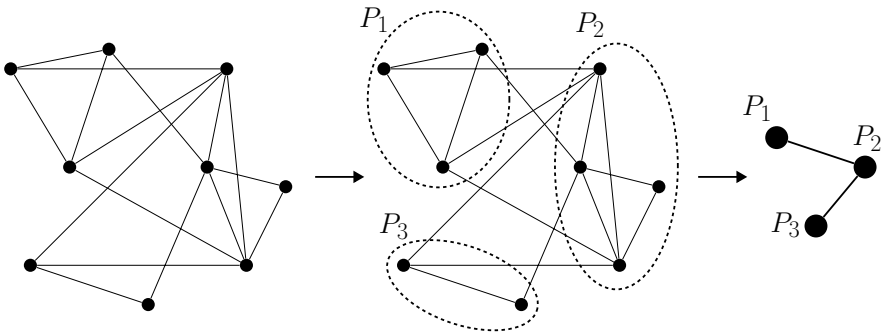
Existuje-li mezi libovolnými dvěma vrcholy grafu cesta, řekneme, že graf je **souvislý**. **Vzdáleností** dvou vrcholů v souvislém grafu $G = (V, E)$ nazveme délku nejkratší cesty mezi těmito dvěma vrcholy. Vzdáleností vrcholu $v \in V$ od množiny vrcholů $W \subset V$ nazveme minimální vzdálenost mezi vrcholem v a libovolným vrcholem náležícím do W.

Při používání víceúrovňových metod dělení grafů (viz kapitola 2.2) budeme často odvozovat z grafu jiný graf shluknutím množin jeho vrcholů Takový graf budeme nazývat faktorgraf a formálně jej definujeme následovně. TODO: n

Definice 5. Nechť $\mathcal{P} = \{P_1, \ldots, P_n\}$ je rozkladem množiny vrcholů V grafu $G = (V, E)$ a $|V| = n$. Faktorgrafem grafu G nazveme graf $G/\mathcal{P} = (\mathcal{P}, \mathcal{E}_\mathcal{P})$, kde

$$((P_i, P_j) \in \mathcal{E}_\mathcal{P} \text{ pro } i, j \in \{1, \ldots, n\}, i \neq j) \Leftrightarrow P_i \cap \mathrm{adj}_G(P_j) \neq \emptyset.$$

Proces vytváření faktorgrafu $G/\mathcal{P}$ z grafu G je znázorněn na obrázku 1.1



Obrázek 1.1: Proces vytváření faktorgrafu

TODO: potrebuju ctvercovou sit? Kdyz budu delat ND, tak jo

1.2 Strom

Nyní zaveďme základní pojmy týkající speciální třídy grafů nazývané stromy [18]

Definice 6. Stromem $T = (V, E)$ nazveme konečný souvislý neorientovaný graf bez cyklů. Pokud navíc v grafu T vyznačíme bod $r \in V$, nazýváme uspořádanou dvojici (T, r) **kořenovým stromem** a bod r nazveme kořenem tohoto stromu.

Z definice stromu je patrné, že každý vrchol v kořenového stromu (T, r) spojuje s kořenem tohoto stromu právě jedna cesta. Vrcholy ležící na této cestě nazveme **předchůdci** vrcholu v. Předchůdce vrcholu v různé od v nazýváme **vlastními předchůdci** vrcholu v. Vrcholy, jejichž předchůdcem je vrchol v, nazýváme **následníky** vrcholu v. Vrcholy bez následníků nazýváme **listy stromu** T, vrcholy alespoň s jedním následníkem nazýváme **vnitřní vrcholy** stromu T.

Definice 7. Podstromem stromu T určeným vrcholem v nazveme podgraf stromu T indukovaný vrcholem v a a všemi jeho následníky.

1.3 Vztah grafu a matice

Grafy a matice spolu úzce souvisí, což nám umožňuje převádět problémy na maticích na problémy na grafech a naopak. Nezanedbatelným praktickým důsledkem jejich vzájemného vztahu je i možnost používat grafové algoritmy při řešení některých maticových úloh, především může být tento přístup výhodný pro řídké matice. Dělení grafů může posloužit například při snaze o paralelizaci rozkladu matice.

Uvažujme neorientovaný graf $G = (V, E)$ s vrcholy $V = \{v_1, \ldots, v_n\}$ a hranami $E = \{e_1, \ldots, e_m\}$. Tento graf lze reprezentovat pomocí matice dvěma základními způsoby. **Maticí sousednosti** grafu G, neboli adjacenční maticí, nazveme matici A_G o rozměrech $n \times n$, jejíž prvek na pozici (i, j) je definován jako:

$$(A_G)_{i,j} := \begin{cases} 1 & \text{existuje-li hrana spojující vrcholy } v_i, v_j \\ 0 & \text{jinak} \end{cases}$$

Maticí incidence grafu G nazveme matici o rozměrech $n \times m$ definovanou následovně:

$$(\bar{A}_G)_{i,j} := \begin{cases} 1 & \text{je-li } v_i \text{ koncovým vrcholem hrany } e_j \\ 0 & \text{jinak} \end{cases}$$

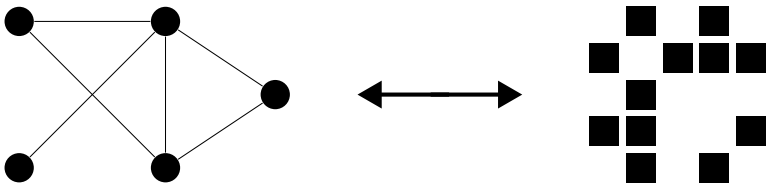
V kapitole 2.3 budeme potřebovat **Laplaceovu matici** Q grafu G. Jedná se o matici o rozměrech $n \times n$ definovanou následovně:

$$Q_{ij} := \begin{cases} -1 & \text{pro } i \neq j, (v_i, v_j) \in E \\ 0 & \text{pro } i \neq j, (v_i, v_j) \notin E \\ d(i) & \text{pro } i = j \end{cases}$$

Laplaceovu matici Q lze tedy vyjádřit jako $Q = D - A_G$, kde D značí diagonální matici se stupni jednotlivých vrcholů na diagonále.

Pokud chceme reprezentovat matici pomocí grafu, většinou nám stačí zachytit její strukturu. V takovém případě můžeme pro popis obecně nesymetrické matice A o rozměrech $n \times n$ použít orientovaný graf s množinou vrcholů $V = v_1, \ldots, v_n$ a množinou hran $E = \{(v_i, v_j) \mid a_{ij} \neq 0\}$. V případě, že je matice A symetrická, můžeme ji analogickým způsobem reprezentovat pomocí neorientovaného grafu. Pokud bychom chtěli do grafu zanést i numerické hodnoty jednotlivých prvků matice, museli bychom použít ohodnocený graf.

Poznámka 2. Pro jednoduchost zaveďme následující terminologii. Říkáme, že graf G **odpovídá** matici A právě tehdy, když matice A má nenulové prvky na stejných pozicích jako matice sousednosti grafu G.



Obrázek 1.2: Příklad grafu a jemu odpovídající struktury matice

Pro reprezentaci ne nutně čtvercové matice A o rozměrech $m \times n$ můžeme také použít bipartitní graf, který je definován následovně:

Definice 8. Graf $G = (V, E)$ nazveme bipartitním, pokud existuje rozklad množiny V na podmnožiny R, B takové, že $\binom{R}{2} \cap E = \binom{B}{2} \cap E = \emptyset$.

Jinými slovy se jedná o graf, který lze rozdělit na dvě části tak, že v žádné z částí není ani jedna hrana. Takovýto bipartitní graf se standardně značí $G = (R, B, E)$ a pokud za jeho pomoci chceme reprezentovat matici A, pokládáme $|R| = m$, $|B| = n$ a $E = \{(v_i, v_j) \mid v_i \in R, v_j \in B, a_{ij} \neq 0\}$.

TODO: Příklad = obrázek

Kapitola 2

Dělení grafů

V této kapitole formálně popíšeme problém dělení grafu na k podgrafů a definujeme pojmy s ním spojené. Vzhledem k tomu, že v naší implementaci používáme pro dělení grafů profesionální softwarovou knihovnu METIS [15], zaměříme se nejprve na schéma víceúrovňového dělení grafu jakožto algoritmus používaný při profesionálních implementacích dělení grafů. Dále se budeme věnovat některým technikám, které tvoří součást víceúrovňového dělení grafů, konkrétně spektrálnímu dělení grafu TODO: zdroj a vylepšovacímu algoritmu podle Kernighana a Lina [17]. TODO: Doresit NDNa závěr kapitoly zmíníme algoritmus metody vnořených řezů (Nested Dissection) jakožto konkrétní příklad víceúrovňového algoritmu pro dělení grafů TODO: zdroj.

Dělení grafů na k podgrafů je praktický problém s bohatým teoretickým zázemím a mnoha aplikacemi. Může nám pomoci při řešení parciálních diferenciálních rovnic na moderních počítačových architekturách [29] a nezanedbatelnou roli hraje také při výrobě mikroprocesorů metodou VLSI nebo při řešení velkých systémů lineárních rovnic [17, 27]. Často dochází k omezení se na dělení grafu na dva podgrafy, v této práce se však pokusíme prozkoumat i vliv dělení grafu na více částí.

Podmínky, která jsou na výsledné rozdělení grafu kladeny, se mohou lišit v závislosti na daném použití. V této kapitole popíšeme klasická kritéria, která se používají pro určování kvality rozdělení grafu, v kapitole 4 poté bude popsán problém dělení grafů vzhledem k dodatečným kritériím (například vzhledem k maticovým operacím).

2.1 Formální definice dělení grafu

Jako dělení grafu na k částí označujeme hledání rozkladu množiny vrcholů tohoto grafu na k podmnožin. V nejklasičtějším případě je problém dělení grafů na k podgrafů definován následovně.

Mějme graf $G = (V, E)$, $|V| = n$. **Rozdělením grafu** G na k podgrafů nazveme rozklad množiny vrcholů V na vzájemně disjunktní podmnožiny $V_1, V_2, \ldots, V_k$. Řekneme, že rozdělení je **optimální** vzhledem k základním kritériím, pokud splňuje:

1. $\forall i \in \{1, \ldots, k\}\ |V_i| = n/k$

2. Počet hran spojujících vrcholy ležící v různých podmnožinách je minimální možný.

Rozdělení grafu běžně zapisujeme jako vektor P o délce n takový, že pro každý vrchol $v \in V$ je podgraf, v němž se vrchol v nachází, určen v-tou složkou vektoru P.

Pokud má graf G sudý počet vrcholů a rozdělíme ho na dvě části s množinami vrcholů V_1, V_2, které mají stejný počet prvků, nazveme toto rozdělení **bisekcí** a velikost hranového separátoru nazýváme **šířkou bisekce**.

Mějme graf $G = (V, E)$ a jeho rozdělení na k podgrafů s množinami vrcholů $V_1, V_2, \ldots, V_k$. Množinu hran, jejichž jeden koncový bod náleží do V_i a druhý do V_j pro $i \neq j$ nazveme **hranovým separátorem**, značíme $\delta(A, B)$. Jinou variantou nalézt rozdělení grafu definované pomocí vrcholového separátoru. **Vrcholovým separátorem** V_S grafu G nazveme podmnožinu množiny vrcholů V takovou, že odstraněním všech vrcholů náležících do V_S z grafu G dojde k jeho rozpadu na nejméně k komponent odpovídajících jednotlivým částem rozdělení. Řešení problému optimální transformace mezi hranovým a vrcholovým separátorem, známe-li jeden z nich, můžeme naleznout v [28]. To nám ale nic neříká o řešení problému samotného nalezení optimálního hranového nebo vrcholového separátoru [22].

Ukazuje se, že rozhodovací problém pro optimální rozdělení grafu je NP-úplný [8]. Existují však algoritmy, které rozdělí graf v rozumném čase, přičemž kvalita jimi nalezeného rozdělení bude poměrně dobrá [21].

2.2 Obecné schéma víceúrovňového dělení grafů

Víceúrovňové dělení grafů slouží k převedení problému dělení grafu s velkým počtem vrcholů na problém dělení grafu s počtem vrcholů výrazně menším. Tím může dojít k výraznému zkrácení času potřebného pro běh algoritmu TODO: zdroj, proto je tento postup využíván v profesionálních softwarových nástrojích pro dělení grafů [11, 15]. Nejobecnější popis tohoto schématu sestává ze tří základních fází: zhrubovací fáze (coarsening phase), rozdělení vzniklého grafu a projekce tohoto rozdělení zpět na původní graf (refinement phase).

Zhrubovací fáze Základní myšlenkou zhrubovací fáze víceúrovňového schématu dělení grafů je vytvořit z původního grafu G graf G_m s menším počtem vrcholů. Jinými slovy jde o konstrukci posloupnosti grafů $(G_i)_0^m$ takové, že $G_0 := G$ a pro každé dva po sobě jdoucí členy této posloupnosti platí, že G_{i+1} je faktorgrafem G_i s menším počtem vrcholů.

Pro konstrukci posloupnosti $(G_i)_0^m$ existuje několik možných postupů [16] TODO: zdroje? kaku je souhrn. Ve většině schémat pro tvorbu grafu s méně vrcholy jsou podmnožiny množiny vrcholů grafu G_i spojovány v jeden vrchol, čímž vznikne hrubší graf G_{i+1} splňující výše uvedené podmínky kladené na posloupnost $(G_i)_0^m$. Při konstrukci členu G_{i+1} z členu G_i je pro udržení strukturálních informací o původním grafu nutné, abychom použili ohodnocený graf, kde ohodnocení vrcholů a hran při tvorbě faktorgrafu pokládáme pro každý vrchol (resp. hranu) roven součtu vah všech vrcholů (resp. hran) spojením kterých daný vrchol (resp. hrana) vznikl.

Definice 9. Mějme graf $G = (V, E)$. Párováním grafu G nazveme podmnožinu množiny E, pro kterou platí, že žádné dvě hrany z této množiny nemají společný koncový bod. Maximálním párováním grafu G nazveme párování grafu G s nejvyšším možným počtem hran.

Pro získání hrubšího grafu byly popsány dva hlavní postupy. První z nich je založen na nalezení vhodného párování a následném spojení každé dvojice vrcholů spojených hranou náležící do párování do jednoho [4, 20]. Aby došlo k rychlému zmenšení počtu vrcholů grafu, je vhodné volit maximální párování [13]. Druhý postup hledání vhodných množin vrcholů pro sloučení je založen na spojování skupin vrcholů, které jsou spojeny mnoha hranami [16] TODO: konkretni? kaku je souhrn.

Dělení získaného grafu V této fázi dochází k rozdělení grafu H libovolným algoritmem pro dělení grafů, například lze použít spektrální dělení popsané níže v kapitole 2.3. Pomocí tohoto algoritmu získáme rozdělení P_m grafu G_m. TODO: tahle faze se muze preskocit

Projekce rozdělení na původní graf Cílem této fáze je převést získané rozdělení P_m grafu G_m na rozdělení P_0 vstupního grafu G. Toho dosáhneme postupnou tvorbou posloupnosti rozdělení $P_{m-1}, \ldots, P_1$ grafů $G_{m-1}, \ldots, G_1$. Nejjednodušším způsobem projekce rozdělení grafu G_{i+1} na rozdělení grafu G_i je umístit všechny vrcholy grafu G_i, jejichž spojením vznikl vrchol v grafu G_{i+1} do části rozdělení, v níž leží vrchol v.

Tento postup však není optimální, protože graf G_i má větší počet vrcholů než G_{i+1} a má tedy více stupňů volnosti vzhledem k optimalitě rozdělení. Díky tomu může nastat situace, kdy rozdělení P_i získané z rozdělení P_{i+1} lze dále vylepšit. Nejběžněji využívaným algoritmem pro vylepšování rozdělení grafu je algoritmus podle Kernighana a Lina [17], který bude popsán v kapitole 2.4. Kvůli snížení časové náročnosti aplikace tohoto algoritmu se častěji setkáme s jeho aplikací pouze na části jednotlivých podgrafů ležící v blízkosti hranového či vrcholového separátoru než s jeho aplikací na celý graf [16].

2.3 Spektrální dělení

Spektrální algoritmus je jedním ze základních algoritmů řešících bisekci grafu. Rozdělení grafu nalezená spektrálním algoritmem bývají dobrá, ale jeho cena je poměrně vysoká. Z tohoto důvodu je pro velké grafy používán jako součást víceúrovňového dělení grafu, kde slouží k nalezení rozdělení hrubého grafu (viz kapitola 2.2).

Mějme graf $G = (V, E)$ se sudým počtem vrcholů a řešme problém bisekce tohoto grafu na dva podgrafy $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$. Základní myšlenkou spektrálního dělení je převést tento problém na problém minimalizace kvadratické formy. Definujme vektor $\vec{x}$ délky $n := |V_1| = |V_2|$ po složkách následovně:

$$x_i = \begin{cases} 1 & \text{pro } i \in V_1 \\ -1 & \text{pro } i \in V_2 \end{cases}$$

Pak pro Laplaceovu matici Q grafu G a diagonální matici D se stupni vrcholů grafu G na diagonále platí:

$$\vec{x}^T Q\vec{x} = \vec{x}^T D\vec{x} - \vec{x}^T A_G\vec{x} = \sum_{i=1}^{n} d_i x_i^2 - 2\sum_{(i,j)\in E} x_i x_j =$$
$$= \sum_{(i,j)\in E} (x_i - x_j)^2 = \sum_{\substack{i\in X, j\in Y \\ (i,j)\in E}} (x_i - x_j)^2 = 4|\delta(X,Y)|,$$

čímž jsme převedli problém dělení grafu na problém minimalizace kvadratické formy $\vec{x}^T Q\vec{x}$ přes vektory $\vec{x}$ délky n se složkami $x_i = \pm 1$ takové, že $\sum_{i=1}^{n} x_i = 0$, tj.

$$4|\delta_{min}(X,Y)| = \min_{\substack{x_i=\pm 1 \\ \sum_{i=1}^{n} x_i = 0}} \vec{x}^T Q\vec{x}.$$

Tento problém dále relaxujeme následovně:

$$\min_{\substack{x_i=\pm 1 \\ \sum_{i=1}^{n} x_i = 0}} \vec{x}^T Q\vec{x} \geq \min_{\substack{\sum_{i=1}^{n} x_i^2 = n \\ \sum_{i=1}^{n} x_i = 0}} \vec{x}^T Q\vec{x} \tag{2.1}$$

Označme $\vec{v_2}$ vlastní vektor matice Q příslušný druhému nejmenšímu vlastnímu číslu matice Q. Pak můžeme pomocí Courantovy-Fischerovy věty [14] výraz na pravé stran nerovnosti 2.1 přepsat jako

$$\min_{\substack{\sum_{i=1}^{n} x_i^2 = n \\ \sum_{i=1}^{n} x_i = 0}} \vec{x}^T Q\vec{x} = \vec{v_2}^T Q\vec{v_2} = \lambda_2 \vec{v_2}^T \vec{v_2} = n\lambda_2.$$

V případě relaxovaného problému nabývá tedy kvadratická forma svého minima pro vektor v_2. Nalezneme medián ze složek tohoto vektoru a podle něj provedeme bisekci grafu G.

Hlavní myšlenkou spektrálního algoritmu je tedy převést problém dělení grafu na problém hledání vlastních čísel matice, který umíme řešit. Pokud bychom chtěli graf dělit na více než dvě části, potřebovali bychom více vlastních vektorů matice Q. Pro dělení na čtyři, resp. osm částí by nám stačily dva, resp. tři vlastní vektory, pro dělení na více částí už nelze postupovat obdobně [12]. Proto se v takovém případě přistupuje k rekurzivnímu dělení jednotlivých částí.

2.4 Algoritmus podle Kernighana a Lina

Algoritmus podle Kernighana a Lina (KL algoritmus) vznikl v roce 1970 s cílem dělit elektrické obvody na kartách [17]. V jeho základní podobě se jedná o algoritmus pro vylepšování již získaného rozdělení grafu, pro jeho funkci je tedy nutné poskytnout mu vstupní rozdělení grafu. Pokud používáme algoritmus podle Kernighana a Lina přímo jako dělicí algoritmus, můžeme vstupní rozdělení lze zvolit libovolně. Jeho výsledky pro různá

počáteční rozdělení se však mohou lišit. Proto je v praxi je výhodné, aby představovalo vstupní rozdělení rozumnou aproximaci optimálního rozdělení. Kvůli tomu se algoritmus podle Kernighana a Lina obvykle používá v kombinaci s jiným algoritmem pro dělení grafů, případně jako součást většího celku - například při víceúrovňovém dělení grafu ve fázi projekce hrubého rozdělení na vstupní graf (viz kapitola 2.2). V tomto případě je většinou využita modifikace tohoto algoritmu, která bere v úvahu pouze vrcholy blízko separátoru.

TODO: KL algoritmus pro k částí - je to tak, že se vždycky přehazuje mezi dvojicí částí? Popišme algoritmus podle Kernighana a Lina pro graf rozdělený na dva podgrafy. Kdyby byl graf obecně rozdělen na k částí, mohli bychom použít tento algoritmus na jednotlivé dvojice podgrafů. TODO: cite{suaris, kedem(viz pothen)}, kteri deli rovnou na 4 casti TODO: end

Mějme graf $G = (V, E)$ rozdělený na dva podgrafy $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$. Základním principem KL algoritmu je, že v každé jeho iteraci dojde k výměně určitého počtu vrcholů z množiny V_1 za stejný počet vrcholů z množiny V_2 tak, aby byla snížena velikost hranového separátoru. TODO: a co vrcholový?

Vrcholy vhodné pro výměnu vybíráme tak, aby při jejich přesunu z části rozdělení, v níž se daný vrchol nachází, do části druhé došlo k maximálnímu zmenšení velikosti hranového separátoru. Tento rozdíl ve velikosti hranového separátoru při přesunu vrcholu v nazveme **ziskem** $D(v)$ a zřejmě jdo o rozdíl počtu hran (případně vah hran) spojujících vrchol v s vrcholy v podgrafu, v němž se vrchol v nachází, a počtu hran spojujících vrchol v s vrcholy v druhém podgrafu. Formálně můžeme zisk pro graf $G = (V, E)$ s rozdělením P a hranovým ohodnocením c zapsat následovně:

$$D(v) := \sum_{\substack{(u,v)\in E \\ P[u]\neq P[v]}} c(u,v) - \sum_{\substack{(u,v)\in E \\ P[u]=P[v]}} c(u,v).$$

Pokud přesuneme vrchol s kladným ziskem, dojde ke zmenšení hranového separátoru.Abychom mohli popsat reálný zisk při výměně vrcholů, je vhodné zavést si ještě pro $u, v \in V$ hodnotu C_{uv}.

$$C_{uv} = \begin{cases} 1 & u, v \in E \\ 0 & \text{jinak} \end{cases}$$

Pak **reálný zisk** při výměně vrcholů u a v ležících ve vzájemně různých částech rozdělení grafu G je zřejmě

$$g_{uv} = D(u) + D(v) - 2C_{uv}.$$

Základní varianta KL algoritmu se skládá ze dvou do sebe vnořených cyklů, z nichž vnější běží do té doby, dokud se velikost hranového separátoru zmenšuje a obsahuje tyto kroky

1. Vypočítáme zisk jednotlivých vrcholů.
2. Postupně spárujeme všechny vrcholy $v_i \in V_1$ s vrcholy s $w_j \in V_2$ tak, aby reálný zisk g_i při výměně dvojice vrcholů (v_k, w_k) pro $k \in \{1, \ldots, \min(|V_1|, |V_2|)\}$ za předpokladu, že by všechny dvojice vrcholů $(v_1, w_1), \ldots, (v_{k-1}, w_{k-1})$ byly vyměněné, byl maximální.

3. Nalezneme takové $N \in \{1, \ldots, \min|V_1|, |V_2|\}$, aby $\sum_{j=1}^{N} g_i$ byla maximální.

4. Vyměníme vrcholy $v_1, ..., v_N$ s vrcholy $w_1, ..., w_N$.

Nejjednodušším vylepšením tohoto algoritmu je nechat vnější cyklus běžet po pevně stanovený počet iterací i poté, co již nedochází ke zmenšování velikosti hranového separátoru. Můžeme se díky tomu dostat z lokálního minima TODO: cite. Nejznámějším praktickým vylepšením algoritmu podle Kernighana a Lina je algoritmus publikovaný v [6], který optimalizuje rychlost výpočtu a aktualizaci zisku pro jednotlivé vrcholy.

2.5 Metoda vnořených řezů

TODO: Je vubec tahle kapitola vhodna? Metoda vnořených řezů byla navržena a publikována v roce 1973 Alanem Georgem TODO: cite. Jedná se o analyticky dobře popsaný algoritmus využívající metodu rozděl a panuj pro řešení problémů na řídkých maticích pomocí jejich převedení na grafy a dělení těchto grafů. Tento algoritmus je teoreticky velmi dobře popsán pro některé speciální typy grafů, například pro čtvercové sítě. V této práci jej uvádíme jako konkrétní příklad přístupu k víceúrovňovému dělení grafů a proto se TODO: ?budeme zabývat jeho obecnou formou pomocí níž lze dělit libovolné grafy. TODO: ND je prikladem multilevel - jak?

Kapitola 3

Rozklady matic

Při řešení maticových úloh velkých rozměrů je často výhodné si danou matici rozložit na součin dvou nebo více matic a díky tomu původní problém převést na sérii výpočetně jednodušších problémů. Jako klasické příklady úloh z numerické lineární algebry, při jejichž řešení nám mohou rozklady matic pomoci, uveďme hledání řešení soustav lineárních algebraických rovnic či problém nalezení vlastních či singulárních čísel a příslušných vektorů. Rozklady matic navíc hrají nezanedbatelnou roli například v QR algoritmu, LR algoritmu [10] nebo Lanczosově algoritmu [19, 26]. Rozklady matic nám mohou pomoci odhalit i teoretické vlastnosti a strukturu matic a odpovídajících maticových problémů. Příkladem takového rozkladu je singulární rozklad (SVD decomposition) [10].

Rozklady matic můžeme rozdělit na dvě základní skupiny - úplné rozklady a neúplné rozklady. V této práci nás budou zajímat především rozklady, které se týkají řídkých matic, tedy matic, které obsahují velké množství nulových prvků. U takových matic bereme pro zefektivnění výpočtů do úvahy strukturu jejich nenulových prvků. Při výpočtu úplného rozkladu řídké matice může v průběhu rozkladu docházet ke změnám ve struktuře jejích nenulových prvků. Proces výpočtu úplného rozkladu může být pak výpočetně velmi náročný. V takovém případě je častou praxí nepočítat úplný rozklad, ale vypočítat jen jeho aproximaci, o které hovoříme jako o neúplném rozkladu a pro získání řešení soustavy je třeba jej kombinovat s nějakou iterační metodou [3].

V námi popisovaném základním modelu rozkladů matic se omezíme na úplný rozklad symetrické a pozitivně definitní matice. Nejedná se o samoúčelné omezení, při řešení mnoha praktických úloh se setkáváme právě s maticemi splňujícími tyto podmínky. Navíc se ukazuje, že i pro matice, které nejsou symetrické nebo pozitivně definitní, lze při jejich rozkladech vyjít z tohoto modelu.

3.1 Úplné rozklady

Společným znakem algoritmů pro úplný rozklad matice je, že při práci v přesné aritmetice, dokážou tyto algoritmy najít přesný rozklad matice v konečně mnoha krocích. Nejklasičtějším příkladem takovéhoto algoritmu je symetrická eliminace, jejímž speciálním případem je Choleského rozklad, který pro nás bude obzvláště zajímavý. V této

kapitole nebudeme ve schématech rozlišovat řídké a husté matice.

V této práci pro nás bude zajímavý především Choleského rozklad jakožto konkrétní příklad algoritmu pro úplný rozklad matice. Jedná se o speciální případ symetrické eliminace, která bude v této kapitole taktéž popsána.

3.1.1 Symetrická eliminace

Cílem **symetrické eliminace** je rozložit danou čtvercovou matici A na tvar LDL^T, kde L je dolní trojúhelníková matice a D je diagonální matice. Obecně pro indefinitní matice je třeba pro zajištění numerické stability tohoto algoritmu vynásobit danou matici nějakou permutační maticí [7]. Pro pozitivně definitní matice je však stabilita zajištěna i bez toho [33].

Mějme matici A o rozměrech $N \times N$. Symetrickou eliminací matice A nazveme hledání konečných posloupností $A_0, \ldots, A_{N-1}, L_1, \ldots, L_{N-1}$ splňujících, že $(\forall i \in \{1, ..., N-1\})\left(A_{i-1} = L_i A_i L_i^T\right)$ a $A_{N-1} = D$. Předpis pro hledání těchto posloupností je následující:

$$\begin{aligned}
A_0 = A &= \begin{pmatrix} d_1 & v_1^T \\ v_1 & \bar{H}_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{v_1}{d_1} & I_{N-1} \end{pmatrix} \begin{pmatrix} d_1 & 0 \\ 0 & \bar{H}_1 - \frac{v_1 v_1^T}{d_1} \end{pmatrix} \begin{pmatrix} 1 & \frac{v_1^T}{d_1} \\ 0 & I_{N-1} \end{pmatrix} \\
&= L_1 \begin{pmatrix} d_1 & 0 \\ 0 & H_1 \end{pmatrix} L_1^T = L_1 A_1 L_1^T \\
A_1 &= \begin{pmatrix} d_1 & 0 & 0 \\ 0 & d_2 & v_2^T \\ 0 & v_2 & \bar{H}_2 \end{pmatrix} = L_2 A_2 L_2^T \\
&\vdots \\
A_{N-1} &= D,
\end{aligned} \tag{3.1}$$

kde $H_i = \bar{H}_i - \frac{v_i v_i^T}{d_i}$. Zřejmě pak $A = LDL^T$, kde $L := L_1 \cdots L_{N-1}$

3.1.2 Choleského rozklad

Choleského rozklad je speciálním případem symetrické eliminace. Jeho cílem je rozložit pozitivně definitní matici A na tvar $A = LL^T$, kde matici L označujeme jako faktor matice A. Jinými slovy se jedná o symetrickou eliminaci matice A, kde požadujeme, aby matice D byla rovna jednotkové matici. Jedná se o modifikaci Gaussovy eliminace pro symetrické pozitivně definitní matice. Základní informaci o existenci Choleského rozkladu pozitivně definitní matice nám dává následující věta, jejíž důkaz můžeme nalézt například v [10].

Věta 1. *Pro libovolnou pozitivně definitní matici A existuje jednoznačný rozklad $A = LL^T$, kde L je dolní trojúhelníková matice s kladnými prvky na diagonále.*

Pro výpočet Choleského rozkladu matice A stačí modifikovat algoritmus 3.1 tak, aby matice D vznikající při symetrické eliminaci matice A byla rovná jednotkové matici. Toho dosáhneme následovně:

$$\begin{aligned} A_0 = A &= \begin{pmatrix} d_1 & v_1^T \\ v_1 & \bar{H}_1 \end{pmatrix} = \begin{pmatrix} \sqrt{d_1} & 0 \\ \frac{v_1}{\sqrt{d_1}} & I_{N-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \bar{H}_1 - \frac{v_1 v_1^T}{d_1} \end{pmatrix} \begin{pmatrix} \sqrt{d_1} & \frac{v_1^T}{\sqrt{d_1}} \\ 0 & I_{N-1} \end{pmatrix} \\ &= L_1 \begin{pmatrix} 1 & 0 \\ 0 & H_1 \end{pmatrix} L_1^T = L_1 A_1 L_1^T \\ A_1 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & d_2 & v_2^T \\ 0 & v_2 & \bar{H}_2 \end{pmatrix} = L_2 A_2 L_2^T \\ &\vdots \\ A_{N-1} &= L_N I_N L_N^T, \end{aligned} \tag{3.2}$$

kde opět zřejmě platí, že $L = L_1 ... L_N$.

Matici L můžeme však z matic $L_1, \ldots, L_N$ vypočítat výrazně jednodušším způsobem.

Lemma 1. *Při konstrukci Choleského rozkladu LL^T matice A pomocí algoritmu (3.2) platí*

$$L = L_1 + L_2 + \ldots + L_N - (N-1)I_N, \tag{3.3}$$

tj. i-tý sloupec L je roven i-tému sloupci L_i.

Důkaz. Matice L_i má tvar

$$\begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & 0 & \\ & & 1 & & & & \\ & & & \sqrt{d_i} & & & \\ & & & \frac{\vec{v_i}}{\sqrt{d_i}} & 1 & & \\ & 0 & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}.$$

Vynásobením matice L_i a L_j, kde $i < j$, zjevně dostanu

$$\begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & 0 & \\ & & \sqrt{d_i} & & & & \\ & & \frac{\vec{v_i}}{\sqrt{d_i}} & \ddots & & & \\ & & & & \sqrt{d_j} & & \\ & 0 & & & \frac{\vec{v_j}}{\sqrt{d_j}} & \ddots & \\ & & & & & & 1 \end{pmatrix},$$

což je přesně matice $L_i + L_j - I$. Z toho již plyne tvrzení lemmatu. □

Pro výpočet prvků matice L existují tři základní postupy: submaticové schéma vyplývající přímo z lemmatu 1, řádkové schéma s použitím metody ohraničování a sloupcové schéma. V přesné aritmetice jsou všechna tři tato schémata ekvivalentní, ale v případě aritmetiky v konečné přesnosti, se kterou se standardně při počítačových výpočtech setkáváme, můžou schémata dávat různé výsledky. Tato schémata napočítávají prvky matice L v různém pořadí. Navíc, efektivní modifikace rozdělení je velmi pevně propojena s očíslováním vrcholů částí rozděleného grafu (popsáno v 4.1).

Submaticové schéma S pomocí lemmatu 1 můžeme navrhnout submaticové schéma pro výpočet faktoru L následovně. Matici L získáváme po sloupcích, ale při výpočtu i-tého sloupce zároveň počítáme submatici $H_i = \bar{H}_i - \frac{v_i v_i^T}{d_i}$ matice A, kterou potřebujeme pro výpočet zbylých částí faktoru matice A (viz algoritmus (3.2)).

Řádkové schéma - metoda ohraničování Jiným přístupem, který můžeme zvolit pro získání Choleského rozkladu LL^T matice A je řádkové schéma využívající metodu ohraničování (bordering method) [5, 25]. Nejprve přepišme matici A do tvaru

$$A = \begin{pmatrix} M & \vec{u} \\ \vec{u^T} & s \end{pmatrix},$$

přičemž pro odvození vztahu pro výpočet prvků faktoru L předpokládejme, že Choleského rozklad $L_M L_M^T$ matice M již známe. Pak pro Choleského rozklad matice A platí:

$$A = \begin{pmatrix} L_M & 0 \\ \vec{w}^T & t \end{pmatrix} \begin{pmatrix} L_M^T & \vec{w} \\ 0 & t \end{pmatrix},$$

kde je zřejmě $\vec{w} = L_M^{-1}\vec{u}$ a $t = (s - \vec{w}^T\vec{w})^{1/2}$.

Pro získání Choleského rozkladu matice M, jehož znalost jsme výše předpoládali, můžeme použít stejný postup. Rekurzivním použitím této metody dostáváme soustavu lineárních algebraických rovnic pro výpočet i-tého řádku matice L.

$$\begin{pmatrix} l_{1,1} & & 0 \\ \vdots & \ddots & \\ l_{i-1,1} & \cdots & l_{i-1,i-1} \end{pmatrix} \begin{pmatrix} l_{i,1} \\ \vdots \\ l_{i,i-1} \end{pmatrix} = \begin{pmatrix} a_{i,1} \\ \vdots \\ a_{i,i-1} \end{pmatrix}$$

$$l_{i,i} = \left(a_{i,i} - \sum_{j=1}^{i-1} l_{i,j}^2 \right)^{\frac{1}{2}}.$$

Sloupcové schéma Jedná se o obdobný postup jako v řádkovém schématu, pouze napočítáváme prvky matice L po sloupcích. Vzorce, které nám vznikou jsou pro $j =$

$1, 2, \ldots, N$ a pro $i = j+1, j+2, \ldots, N$ následující:

$$l_{j,j} = \left(a_{j,j} - \sum_{k=1}^{j-1} l_{j,k}^2 \right)^{\frac{1}{2}}$$
$$l_{i,j} = \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} l_{j,k} \right) / l_{j,j}.$$

3.1.2.1 Choleského rozklad pro řídké matice

V praktických aplikacích Choleského rozkladu se často setkáváme s řídkými maticemi, navíc jsou pro nás řídké matice zajímavé i z hlediska dělení příslušných grafů. Pokud máme řídkou matici, v optimálním případě bychom chtěli, aby i její Choleského faktor byl řídká matice, jejíž dolní trojúhelník má stejnou strukturu jako dolní trojúhelník původní matice.

Mějme čtvercovou matici A o rozměrech $N \times N$ a její Choleského rozklad LL^T. Základním problémem výše zmíněných algoritmů je, že matice L běžně obsahuje nenulové prvky na místech, na nichž měla matice A nuly. Toto zaplňování je způsobeno při vytváření posloupnosti matic H_i (viz algoritmus 3.2), konkrétně při přechodu od matice $\bar{H}_i$ k matici H_i. Připomeňme, že mezi těmito dvěma maticemi platí vztah

$$H_i = \bar{H}_i - \frac{\vec{v_i}\vec{v_i}^T}{d_i}.$$

Označme $\eta(A)$, resp. $\eta(\vec{v})$ počet nenulových prvků matice A, resp. vektoru v. Pak z lemmatu 1 a ze vztahu pro výpočet L_i v Choleského algoritmu 3.2 zřejmě platí

$$\eta(L) = N + \sum_{i=1}^{N-1} \eta(\vec{v_i}).$$

Vyslovme větu, ze které nám vyplyne počet informaci o počtu multiplikativních operací (násobení a dělení) potřebných pro pro výpočet Choleského rozkladu matice A.

Věta 2. *Počet multiplikativních operací potřebných pro výpočet Choleského rozkladu LL^T matice A o rozměrech $N \times N$ je roven*

$$\frac{1}{2} \sum_{i=1}^{N-1} \eta(v_i) \left(\eta(v_i) + 3\right) = \frac{1}{2} \sum_{i=1}^{N-1} \left(\eta(L_{*i}) - 1\right) \left(\eta(L_{*i}) + 2\right),$$

kde L_{*i} označuje i-tý sloupec matice L.

Důkaz. Jednotlivá schémata pro výpočet matice L popsaná v předchozí kapitole se liší pouze v pořadí, v němž výpočet prvků matice L provádějí.

V i-tém kroku algoritmu potřebujeme $\eta(\vec{v_i})$ operací pro výpočet $v_i/\sqrt{d_i}$ a $\frac{1}{2}\eta(\vec{v_i})\left(\eta(\vec{v_i}) + 1\right)$ operací pro vytvoření matice $\frac{\vec{v_i}\vec{v_i}^T}{d_i}$. Tvrzení věty dostaneme sečtením přes všechna i. □

Poznámka 3. Pro hustou matici B platí, že počet nenulových prvků v jejím Choleského faktoru L_B je roven $\frac{1}{2}N(N+1)$. Z věty 2 je tedy počet operací potřebný pro výpočet Choleského rozkladu matice B roven

$$\frac{1}{2}\sum_{1}^{N-1} i(i+3) = \frac{1}{6}N^3 + \frac{1}{2}N^2 - \frac{2}{3}N. \tag{3.4}$$

Z toho plyne, že počet operací pro výpočet Choleského rozkladu libovolné řídké matice o rozměrech $N \times N$ lze seshora odhadnout číslem daným vztahem (3.4).

Problém zaplňování se dá alespoň částečně vyřešit tím, že místo rozkladu matice A provádíme rozklad matice PAP^T, kde P je nějaká permutační matice (tj. matice, kterou když vynásobíme matici A, dojde pouze k permutaci řádků, resp. sloupců matice A). Vzhledem k tomu, že jsme od začátku předpokládali, že je matice A symetrická a pozitivně definitní, je i matice PAP^T symetrická a pozitivně definitní. Pokud máme zadán problém řešení soustavy $Ax = b$, převedeme jej na problém řešení soustavy $(PAP^T)(Px) = Pb$. Tento problém je úzce spjat s problémem očíslování grafů při jejich dělení a budeme se jím zabývat v dalších kapitolách.

Kapitola 4

Další parametry pro dělení grafu

Dělení grafu lze využít jako nástroj pro paralelizaci problémů, které lze převést na úlohy na grafech. Vzhledem ke vztahu matic a grafů popsanému výše v kapitole 1.3 lze problém minimalizace zaplnění v Choleského faktoru dané matice pomocí permutace řádků a slouců popsaný v kapitole 3.1.2.1 převést na problém na odpovídajícím grafu.

Jak bylo zmíněno, při dělení grafu podle klasické definice bereme ohled pouze na dvě kritéria: na velikost separátoru a vyváženost jednotlivých částí rozdělení. V případě, že dělení grafu využíváme jako součást komplexnějšího algoritmu, například pro paralelizaci nějaké maticové úlohy, nemusí být rozdělení vytvořené s ohledem na tato kritéria optimální. V takovém případě je nutné brát ohled na to, jaké operace budeme na výsledných podgrafech provádět a v závislosti na tom specifikovat nová kritéria.

Pro dělení grafu s ohledem na více než dvě základní kritéria popsaná výše existují dva základní přístupy. Můžeme hledat rozdělení grafu optimalizující všechna kritéria zároveň (apriorní přístup), nebo můžeme graf rozdělit pomocí některého ze základních algoritmů pro dělení grafu a poté vzniklé rozdělení vylepšit tak, abychom optimalizovali vyváženost operací (aposteriorní přístup).

V této práci se zaměříme na situaci, kdy je dělení grafů použito jako nástroj pro vyvažování počtu operací na jednotlivých oblastech při paralelizaci Choleského rozkladu. Požadujeme tedy, aby počet operací potřebný pro Choleského rozklad na jednotlivých oblastech byl v optimálním případě stejný. Zřejmě nám takto vzniká kritérium, které není zahrnuto v kritériích podle klasické definice - ta totiž neberou v potaz počet hran v grafech odpovídajících jednotlivým oblastech, neboli v řeči matic počet nenulových prvků matic sousednosti jednotlivých podgrafů. Čas pro vypočítání rozkladu na jednotlivých oblastech může být kvůli tomu výrazně odlišný. Námi navržený algoritmus bude využívat aposteriorního přístupu, nejprve tedy nalezneme suboptimální řešení vzhledem k základním kritériím a poté budeme přesouvat vrcholy mezi jednotlivými částmi rozdělení s cílem vylepšit jej vzhledem k Choleského rozkladu.

Mějme graf G a jeho rozdělení na podgrafy $G_1, \ldots, G_k$ s maticemi sousednosti $A_{G_1}, \ldots, A_{G_k}$. Provedeme Choleského rozklad matic A_{G_1}, A_{G_2}, označme

$$A_{G_i} = L_{G_i} L_{G_i}^T \text{ pro } i = 1, 2.$$

Pokud existují $i, j \in \{1, \ldots, k\}$ tak, že matice $L_{G_i}^T$ a $L_{G_j}^T$ budou mít výrazně různý počet nenulových prvků, je zřejmé, že toto rozdělení grafu G je nevhodné pro napočítávání Choleského rozkladu na oblastech vzhledem k vyváženosti na počet operací. Někdy pro vyvážení rozdělení za účelem Choleského rozkladu na oblastech stačí vhodně přečíslovat vrcholy grafu G a tím změnit matice sousednosti odpovídající jednotlivým podgrafům.TODO: Budeme to dělat takhle, nebo budeme číslovat vrcholy až po rozdělení?

Poznámka 4. Mějme soustavu lineárních algebraických rovnic $Ax = b$ a hledejme její řešení pomocí Choleského rozkladu této matice $A = LL^T$ za pomoci dělení grafů. Pro vyřešení soustavy potřebujeme vyřešit rovnice

$$\begin{aligned} Ly &= b \\ L^T x &= y \end{aligned}$$

Pokud rozdělíme graf příslušný matici A na k částí a provedeme rozklad matic $A_1 = L_1 L_1^T, \ldots A_k = L_k L_k^T$ odpovídajících vzniklým podgrafům a matice $A_S = L_S L_S^T$ odpovídající vrcholovému separátoru, pak lze soustavu $Ly = b$ napsat ve tvaru

$$\begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ddots & & \\ & & & L_k & \\ S_1 & S_2 & & S_k & L_S \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \\ y_S \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \\ b_S \end{pmatrix}$$

a tedy

$$\begin{aligned} y_1 &= L_1^{-1} b_1 \\ y_2 &= L_2^{-1} b_2 \\ &\vdots \\ y_k &= L_k^{-1} b_k \\ y_S &= L_1^{-1}(b_S - S_1 y_1 - S_2 y_2 - \ldots - S_k y_k) \end{aligned}$$

Je vidět, že zatímco výpočet $y_1, \ldots, y_k$ může probíhat paralelně, výpočet y_S je závislý na $y_1, \ldots, y_k$. Je tedy vhodné, aby separátor byl malý. Soustavu $L^T x = y$ poté vyřešíme obdobně.

Jak vidíme, vyvstávají nám při dělení grafu s cílem vypočítávat Choleského rozklad na oblastech dvě základní otázky:

1. Jak najít očíslování grafu takové, aby Choleského rozklad na jednotlivých podgrafech byl vyvážený? TODO: Přikláněl bych se k tomu rovnat až výsledné podgrafy
2. Jak určit počet nenulových prvků v Choleského rozkladu matice se strukturou matice sousednosti jednotlivých podgrafů?

Způsob hledání odpovědí na tyto dvě otázky popíšeme v následujících dvou oddílech.

4.1 Algoritmy pro očíslování vrcholů grafu

V tomto oddíle popíšeme základní algoritmy pro očíslování vrcholů grafu. Zřejmě platí, že přečíslováním vrcholů grafu dojde k permutaci řádků a sloupů v jeho matici sousednosti. Hlavní motivací, proč je pro nás číslování vrcholů grafu důležité, tedy je, že permutací řádků matice můžeme výrazně redukovat počet operací potřebných pro její Choleského rozklad.

Na závěr kapitoly se zmíníme o topologickém číslování vrcholů stromu, které pro nás bude potřebné při popisu eliminačních stromů.

4.1.1 Číslování vrcholů grafu v závislosti na vzdálenosti od separátoru

V této podkapitole popíšeme nejjednodušší metodu číslování vrcholů podgrafu, který vznikl rozdělením původního grafu na n částí. Tuto metodu lze používat samostatně, ale vzhledem k její povaze ji lze využít i pro vylepšení ostatních metod očíslování grafu, například ji lze kombinovat s metodou minimálního stupně.

Mějme graf $G = (V, E)$ a jeho vrcholový separátor G_S, jehož odebráním se graf rozpadne na k podgrafů $G_1, \ldots, G_k$. Popišme číslování vrcholů podgrafu G_i:

1. Položme `j := 1`.
2. Nalezneme neočíslovaný vrchol v grafu G_i takový, že jeho vzdálenost od vrcholového separátoru v grafu G je maximální.
3. Tomuto vrcholu dáme číslo `j`, položíme `j := j + 1`.
4. Pokud jsou všechny vrcholy očíslovány, skončíme, jinak se vrátíme na krok 2

Z algoritmu je vidět, že výsledné očíslování vrcholů grafu nemusí být jednoznačné, protože pokud nalezneme dva nebo více vrcholů, jejichž vzdálenost od separátoru je shodná, můžeme je očíslovat v libovolném pořadí.

4.1.2 Číslování vrcholů pomocí metody minimálního stupně

Metoda minimálního stupně je jednoduchým algoritmem pro nalezení očíslování grafu. Algoritmus pro hledání očíslování grafu pomocí této metody je následující:

1. Mějme graf $G = (V, E)$ a položme $j := 1$.
2. Nalezneme neočíslovaný vrchol v grafu G s nejmenším stupněm a přiřadíme mu číslo j.
3. Přidáme hrany mezi vrcholy z $\mathrm{adj}_G(v)$ tak, aby $\mathrm{adj}_G(v)$ byla klika v grafu G.
4. Pokud nejsou všechny vrcholy očíslované, zvětšíme j o 1 a vrátíme se na krok 2.

Očíslování vrcholů grafu G pomocí tohoto algoritmu není jednoznačné, protože vrcholů s minimálním stupněm může být více.

4.1.3 Smíšené číslování

Vzhledem k povaze číslování popsaných v oddílech 4.1.2 a 4.1.1 můžeme tyto algoritmy zkombinovat. V tomto oddíle popisujeme dvě možnosti přístupu k tomuto problému.

Pokud máme rozdělení $G_1, \ldots, G_k$ grafu G s vrcholovým separátorem G_S, můžeme pro očíslování části G_i použít číslování vrcholů pomocí metody minimálního stupně, kde při výběru vrcholu ve 2. kroku přidáme kritérium vzdálenosti od separátoru popsané v 4.1.1. Nejprve tedy nalezneme množinu všech vrcholů grafu G, které mají minimální stupeň a poté mezi nimi zvolíme ten, který má nejmenší stupeň.

TODO: smíšené s různými koef.

4.1.4 Topologické číslování vrcholů stromu

Topologické číslování vrcholů stromu je intuitivní způsob pro očíslování vrcholů grafu, který je stromem.

Definice 10. Mějme graf $G = (V, E)$, který je stromem. Očíslování jeho vrcholů nazveme topologickým právě tehdy, když pro každý vrchol $v \in V$ platí, že libovolný následník vrcholu v ve stromu G má nižší číslo než vrchol v.

4.2 Eliminační stromy

TODO: Udelat poradek s $[A], [L], a, l$

V této kapitole se budeme zabývat eliminačními stromy a jejich významem pro rozklady řídkých matic [23, 24]. Eliminační stromy při rozkladu matic hrají důležitou roli, protože nám dávají informaci o zaplnění v Choleského faktoru matice bez toho, abychom museli počítat jednotlivé numerické hodnoty. Lze tedy díky nim jednoduše porovnávat vhodnost zvoleného uspořádání řádků a sloupců matice pro Choleského rozklad.

V této kapitole bez újmy na obecnosti předpokládáme, že matice, jejíž Choleského rozklad chceme napočítávat, je ireducibilní, a tedy graf odpovídající této matici je souvislý.

4.2.1 Definice eliminačního stromu matice

Nejprve se omezme na ireducibilní, pozitivně definitní, symetrickou matici A_T o rozměrech $n \times n$, jejíž přidružený graf $G(A_T)$ je kořenový strom. Aby při Choleského rozkladu matice A_T nedošlo k žádnému zaplnění, stačí když pomocí topologického číslování očíslujeme vrcholy jí přidruženého grafu (z předpokladu se jedná o strom) a řádky a sloupce matice A_T seřadíme odpovídajícím způsobem. Pak zjevně platí, že matice A_T má, s výjimkou posledního řádku, pod diagonálou vždy právě jeden nenulový prvek. Díky této vlastnosti označujeme A_T jako perfektní eliminační matici, tj. existuje permutační matice P taková, že Choleského rozklad matice PA_TP^T nebude obsahovat žádné zaplnění [30] (Matici PA_TP^T můžeme vnímat pouze jako přečíslování řádků a sloupců matice A_T).

Proto pro matici A_T můžeme definovat funkci $\texttt{PARENT} : \{1,\dots,n\} \to \{1,\dots,n\}$ následovně:

$$\begin{aligned} \forall j \in \{1,\dots,n-1\} \quad \texttt{PARENT}[j] := p \quad &\Leftrightarrow \quad a_{p,j} \neq 0 \ \wedge \ p > j \\ \text{a speciálně:} \quad \texttt{PARENT}[n] &:= 0. \end{aligned}$$

Zřejmě ve stromu přidruženém k matici A_T platí, že předchůdcem vrcholu x_j je vrchol $x_{\texttt{PARENT}[j]}$.

Většinou však nepracujeme s maticemi, jejichž přidružený graf by byl stromem. Zobecněme tedy výše popsanou konstrukci pro libovolnou řídkou, ireducibilní, pozitivně definitní, symetrickou matici A o rozměrech $n \times n$. Předpokládejme, že známe Choleského rozklad této matice, tj. $A = LL^T$. **Maticí se zaplněním** nazveme matici F definovanou jako $F = L + L^T$. Dále zavedeme matice L_t a F_t následovně: L_t je matice vzniklá z L tím, že v každém sloupci vynulujeme všechny prvky pod diagonálou kromě prvku s nejnižším řádkovým indexem a $F_t := L_t + L_t^T$.

Z definice F_t vidíme, že se jedná o matici, jejíž přidružený graf $G(F_t)$ je strom. Díky tomu můžeme vyslovit následující definici.

Definice 11. Eliminačním stromem matice A nazveme graf $G(F_t)$ popsaný výše, značíme $T(A)$. Podstrom eliminačního stromu $T(A)$ s kořenem x_j značíme $T[x_j]$. Množinu vrcholů tohoto stromu značíme taktéž $T[x_j]$.

Díky této definici můžeme definici funkce PARENT přirozeně rozšířit na matici A následovně:

$$\texttt{PARENT}[j] := \min\{i > j \mid l_{i,j} \neq 0\},$$

kde $l_{i,j}$ označuje i,j-tý prvek matice L.

Pozorování 1. Přímo z definice plyne, že $T(A)$ a $T(F)$ jsou identické.

Pozorování 2. Pokud x_i je vlastním předchůdcem x_j v eliminačním stromu, pak $i > j$.

Tvrzení 1. *Nechť A je řídká, ireducibilní, pozitivně definitní, symetrická matice o rozměrech $n \times n$ s Choleského rozkladem LL^T a nechť $i, j \in \{1,\dots,n\}$, $i > j$. Pak numerické hodnoty sloupce $L_{\bullet i}$ závisí na sloupci $L_{\bullet j}$ právě tehdy, když $l_{i,j} \neq 0$.*

Důkaz. Tvrzení plyne přímo ze vzorců pro výpočet Choleského rozkladu pomocí sloupcového algoritmu popsaného v kapitole 3.1.2 □

Přímým důsledkem tvrzení 1 je, že graf $G(F)$ zachycuje informace o závislostech mezi jednotlivými sloupci při výpočtu Choleského rozkladu.

4.2.2 Eliminační stromy a Choleského rozklad

Z definice eliminačních stromů je zřejmé, že budou obsahovat velké množství informací o rozkladu matice a procesu jeho vytváření. Například v [32] byly popsány vlastnosti eliminačních stromů vzhledem ke Gaussově eliminaci. My však budeme eliminační stromy

využívat jako nástroj pro výpočet počtu nenulových prvků v Choleského faktoru matice [24]. Uveďme větu, která dává do souvislosti nenulové prvky v Choleského faktoru a strukturu eliminačního stromu [23].

Věta 3. *Nechť A je řídká, ireducibilní, pozitivně definitní, symetrická matice o rozměrech $n \times n$ s Choleského rozkladem LL^T a nechť $i, j \in \{1, \ldots, n\}$. Pak $[L]_{ij} \neq 0$ právě tehdy, když vrchol v_j je v eliminačním stromě následníkem vrcholu v_k takového, že $[A]_{ij} \neq 0$.*

Tato věta nám dává návod, jak vypočítat počet nenulových prvků v Choleského faktoru. Uveďme algoritmus popsaný v [24], který nám dává návod, jak napočítat počty nenulových prvků v jednotlivých řádcích a sloupcích Choleského faktoru. Obdobný algoritmus s důrazem na implementaci na multiprocesorech můžeme nalézt v [34]. Námi popsaný algoritmus je poměrně jednoduchý, efektivnější ale složitější algoritmus pro výpočet počtu nenulových prvků můžeme najít v [9]. Dále v [2] můžeme najít postup, jakým napočítávat počet nenulových prvků ve faktoru a strukturu eliminačního stromu současně.

V uvedeném algoritmu 1 je použit pracovní vektor marker, který složí k označování vrcholů eliminačního stromu, které byly v dané iteraci již uvažovány - indexování vrcholů eliminačního stromu uvažujeme tak, že odpovídá vstupní matici. Eliminační strom je popsán pomocí vektoru `PARENT`, který byl zmíněn výše.

TODO: Neni to plagiatorstvi?

```
for j := 1 to n do
    η(L_{*j}) := 1;
end
for i := 1 to n do
    η(L_{i*}) := 1;
    marker[i] := i;
    for k := 1 to i − 1 do
        if [A]_{i,k} ≠ 0 then
            j := k;
            while marker[j] ≠ i do
                η(L_{i*}) := η(L_{i*}) + 1;
                η(L_{*j}) := η(L_{*j}) + 1;
                marker[j] = i;
                j :=PARENT[j];
            end
        end
    end
end
```

Algoritmus 1: Výpočet počtu nenulových prvků z eliminačního stromu ([24], upraveno)

Algoritmus 1 nám dává návod jak vypočítat počet prvků v jednotlivých řádcích a sloupcích matice. Pokud nás zajímá pouze počet nenulových prvků v celé matici, můžeme počítat pouze počet prvků v řádcích, resp. sloupcích.

Tento algoritmus nám dává odpověď na to, jak zjistit počet nenulových prvků v matici L z eliminačního stromu matice A. Pokud bychom eliminační strom vytvářeli přímo z definice, museli bychom vypočítat Choleského rozklad. Proto potřebujeme algoritmus, který nám sestrojí Eliminační strom - tedy vektor `PARENT` - bez toho.

Uveďme základní algoritmus pro sestrojení eliminačního stromu matice bez výpočtu jejího Choleského rozkladu. Dále popišme jeho vylepšení, které budeme používat v implementaci. Oba algoritmy jsou převzaté z [23].

```
for i := 1 to n do
    PARENT[i] = 0;
    for j := 1 to i − 1 do
        if [A]_{i,j} ≠ 0 then
            r := j;
            while PARENT[r] ≠ 0 and PARENT[r] ≠ i do
                r :=PARENT[r];
            end
            if PARENT[r] == 0 then
                PARENT[r] := i;
            end
        end
    end
end
```

Algoritmus 2: Základní algoritmus pro hledání eliminačního stromu ([23], upraveno)

Pro zrychlení tohoto algoritmu použijeme pomocný vektor `ANCESTOR`, který nám zkrátí procházení stromu směrem ke kořeni díky tomu, že v něm bude pro každý vrchol zaznamenáván poslední zpracovaný předek daného vrcholu. Algoritmus pak bude vypadat následovně:

Algoritmus 3 má složitost $O(|E| \log_2 n)$ TODO: slozitost+citace viz liu:86. Existuje ještě efektivnější verze tohoto algoritmu, kterou můžeme nalézt v TODO: cite Tarjan Data structures and network algorithms.

Algoritmy uvedené v této kapitole jsou popsány tak, aby byly dostatečně názorné. Ve skutečnosti pro řídké matice uložené v CSR formátu (viz 5.1.2) neprocházíme všechny prvky dané matice, ale pouze nenulové prvky uložené ve struktuře.

```
for i := 1 to n do
    PARENT[i] = 0;
    ANCESTOR[i] = 0;
    for j := 1 to i − 1 do
        if [A]_{i,j} ≠ 0 then
            r := j;
            while ANCESTOR[r] ≠ 0 and ANCESTOR[r] ≠ i do
                t :=ANCESTOR[r];
                ANCESTOR[r] := i;
                r := t;
            end
            if ANCESTOR[r] == 0 then
                ANCESTOR[r] := i;
                PARENT[r] := i;
            end
        end
    end
end
```

Algoritmus 3: Vylepšený algoritmus pro hledání eliminačního stromu ([23], upraveno)

Kapitola 5

Algoritmizace a implementace

Cílem programu, který jsme implementovali, je ukázat, že rozdělení grafu získané za pomoci profesionální numerické knihovny pro dělení grafů METIS, nemusí být vyvážené vzhledem k výpočtu Choleského rozkladu na vzniklých oblastech. Dále jsme se pokusili navrhnout a otestovat metodu přerozdělení grafu založenou na přečíslování vrcholů grafu. Cílem navržené metody je zlepšit rozdělení grafu tak, aby počet operací potřebných pro Choleského faktorizaci na jednotlivých oblastem byl pokud možno stejný a nebyl vyšší než maximum z počtu operací při rozdělení počátečním.

TODO: Fortran, Fortran90, c++ TODO: dělení na kolik částí?

Naši implementaci můžeme rozdělit na několik logických celků. TODO: doplnit

5.1 Formát a uložení vstupní matice

V této podkapitole bude popsán vstupní formát matice a způsob její reprezentace v našem programu. TODO: Zajímá nás jen struktura

5.1.1 Vstupní formát matice

Jako základní matice pro testování výsledků našeho programu nám posloužily matice uložené ve formátu RSA v souborech s příponou `.rsa` nebo `.rb`. Jako zdroj pro tyto matice jsme použili kolekci [1].

TODO: popis HB formátu

Pro testování jsme dále používali matice vygenerované při jednoduché pětibodové diskretizaci dvojrozměrné Poissonovy rovnice (podprogram (poisson)) a několik jednoduchých testovacích matic, které jsme si ručně zapsali do souboru `testing.f90`

5.1.2 Reprezentace matice v programu

Matice je v našem programu uložena v **CSR formátu** (compressed sparse row format) [27, 31], který je běžně využívaným formátem sloužícím pro reprezentaci řídkých matic a grafů.

Mějme řídkou matici A o rozměrech $n \times n$, která obsahuje n_e nenulových prvků. Tuto matici v programu reprezentujeme pomocí tří polí: pole `ia` o délce $n+1$ a polí `ja`,`aa` o délce n_e. V poli `ja` na pozicích `ia(`i`)`,...,`ia(`$i+1$`)-1` jsou uloženy sloupcové indexy nenulových prvků na i-tém řádku matice A, na odpovídajících pozicích v poli `aa` jsou uloženy numerické hodnoty těchto prvků.

Příklad 1. Mějme následující symetrickou matici

$$\begin{pmatrix} 0 & 5 & 0 & 3 & 0 \\ 5 & 0 & 9 & 4 & 2 \\ 0 & 9 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 4 \\ 0 & 2 & 0 & 4 & 0 \end{pmatrix}$$

Pak její reprezentace ve formátu CSR vypadá následovně:

```
ia = [ 1, 3, 7, 8, 11, 13 ]
ja = [ 2, 4, 1, 3, 4, 5, 2, 1, 2, 5, 2, 4 ]
aa = [ 5, 3, 5, 9, 4, 2, 9, 3, 4, 4, 2, 4 ]
```

Pokud budeme uvažovat pouze pole `ia`, `ja`, je podle teorie popsané v oddíle 1.3 výsledná reprezentace matice A zároveň reprezentací neorientovaného grafu, kterému matice A odpovídá. Pokud tedy nebudeme brát v úvahu hodnoty jednotlivých prvků matice, můžeme při popisu implementace bez újmy na obecnosti pojmy graf a matice zaměňovat.

Příklad 2. Vezměme matici z příkladu 1. Pak graf, který je reprezentován pomocí polí `ia`, `ja` je zobrazen na obrázku 1.3

5.2 Knihovna METIS pro dělení grafu

Pro samotné dělení grafu na k částí jsme využili knihovnu pro dělení grafů METIS [15] verze 5.1.0. Vzhledem k tomu, že jsme dělení grafů používali jako nástroj pro vyvažování počtu operací při Choleského rozkladu na jednotlivých oblastech, potřebovali jsme nalézt rozdělení grafu pomocí vrcholového separátoru. Takové rozdělení nám poskytne rutina `METIS_ComputeVertexSeparator`, kterou jsme ve Fortranu implementovali pomocí rozhraní `metis_interface.f95` a `metisinclude.c`.

Zmíněná rutina není zmíněna v oficiální dokumentaci knihovny METIS a není tedy ve své základní verzi kompatibilní s jazykem Fortran. Proto bylo nutné učinit několik změn ve formátu CSR tak, aby bylo možné rutinu použít. Nejprve bylo třeba odebrat z grafu smyčky a poté jej přečíslovat tak, aby bylo vrcholy indexovány od 0, jak je požadováno v jazycích C a C++.

Výstupem rutiny `METIS_ComputeVertexSeparator` je pole *part* o délce rovné počtu vrcholů grafu. Na i-té pozici tohoto pole je číslo od 1 do $k+1$ (po převedení do Fortranového zápisu), které určuje, ve které části rozdělení se daný vrchol nachází. $k+1$ značí, že daný vrchol je ve vrcholovém separátoru.

5.3 Tvorba podgrafů a přerozdělení

TODO: doplnit

5.3.1 Číslování

TODO: doplnit

5.3.2 Přerozdělení

TODO: doplnit

5.4 Výpočet Choleského rozkladu

TODO: doplnit

5.5 Softwarové požadavky pro běh

TODO: mám to tam vůbec dávat? Pro úspěšné slinkování a zkompilování TODO: nainstalovaný METIS

Závěr

TODO: doplnit TODO: motivace: nelineární problém viz Tumovy stranky

Literatura

[1] Harwell-boeing collection. `https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/`. Accessed: 5. 10. 2018.

[2] Randolph Bank and R. Kent Smith. General sparse elimination requires no permanent integer storage. *SIAM J. Sci. Stat. Comp*, 8:574–584, 1987.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[4] J. Edmonds. Maximum matching and a polyhedron with $0, 1$-vertices. *Journal of Research of the National Bureau of Standards*, 69B:125–130, 1965.

[5] V. N. Faddeeva. *Computational methods of linear algebra*. Dover books on advanced mathematics. Dover Publications, 1959.

[6] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. pages 175–181, 1982.

[7] George E. Forsythe and Cleve B. Moler. *Computer solution of linear algebraic systems*. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman & Co., 1979.

[9] John R. Gilbert, Esmond G. Ng, and Barry W. Peyton. An efficient algorithm to compute row and column counts for sparse cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15(4):1075–1091, October 1994.

[10] G. H. Golub and C. F. Van Loan. *Matrix computations*, volume 3 of *Johns Hopkins Series in the Mathematical Sciences*. Johns Hopkins University Press, Baltimore, MD, 1983.

[11] B. Hendrickson and R. Leland. The chaco user's guide version 2.0. 1995.

[12] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.*, 16(2):452–469, 1995.

[13] Bruce Hendrickson and Robert Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, Supercomputing '95, New York, NY, USA, 1995. ACM.

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

[15] G. Karypis. Metis - a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 5.1.0. 2013.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Matrix Anal. Appl.*, 20(1):359–392, 1998.

[17] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, 1970.

[18] A. Koubková and V. Koubek. *Datové struktury 1.* Praha: Matfyzpress, 1 edition, 2011.

[19] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Research Nat. Bur. Standards*, 45:255–282, 1950.

[20] E. L. Lawler. *Combinatorial optimization: networks and matroids.* Holt, Rinehart and Winston, New York-Montreal, Que.-London, 1976.

[21] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36(2):177–189, 1979.

[22] J. W. H. Liu. A graph partitioning algorithm by node separators. *ACM Trans. Math. Software*, 15(3):198–219, 1989.

[23] Joseph W. Liu. A compact row storage scheme for cholesky factors using elimination trees. *ACM Trans. Math. Softw.*, 12(2):127–148, June 1986.

[24] Joseph W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, January 1990.

[25] J. M. Ortega. *Introduction to Parallel &Amp; Vector Solution of Linear Systems.* Plenum Press, New York, NY, USA, 1988.

[26] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.*, 10, 1972.

[27] S. Pissanetzky. *Sparse matrix technology.* Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], London, 1984.

[28] A. Pothen and Ch.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Software*, 16(4):303–324, 1990.

[29] A. Pothen, H. D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[30] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. pages 183–217, 1972.

[31] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations, version 2. Technical report, 1994.

[32] Robert Schreiber. A new implementation of sparse gaussian elimination. *ACM Trans. Math. Softw.*, 8(3):256–276, September 1982.

[33] J. H. Wilkinson. A priori error analysis of algebraic processes. In I. G. Petrovsky, editor, *Proc. Int. Congr. Mthns*, pages 629–640, Moscow, USSR, 1968. Izdatel'stvo Mir.

[34] Earl Zmijewski and John R. Gilbert. A parallel algorithm for sparse symbolic Cholesky factorization on a multiprocessor. *Parallel Comput.*, 7(2):199–210, 1988.