Kapitola 1

Grafy, stromy

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 [24]

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 a 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 samostatně, 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 $W \times W \cup W$, tj. všech uspořádaných dvojic vrcholů z W a orientovaných smyček.

Stupněm vrcholu $v \in V$ rozumíme počet vrcholů spojených s vrcholem v a 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 a značíme jej $G(W)$.

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 $G = (V, E)$ a 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**.

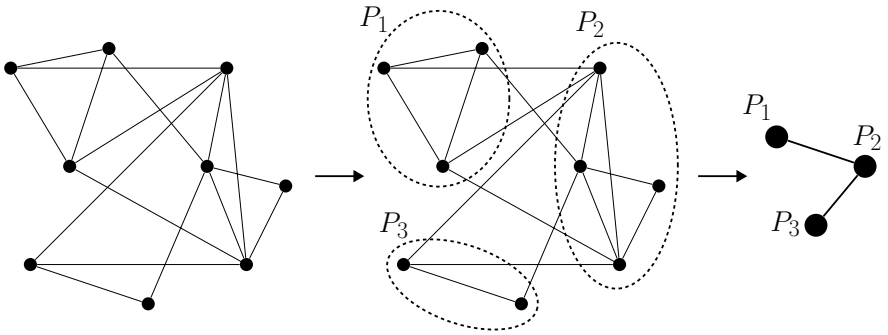
Existuje-li mezi libovolnými dvěma vrcholy grafu cesta, řekneme, že graf je **souvislý**. **Vzdáleností** dvou vrcholů v souvislém grafu $G = (V, E)$ nazveme délku nejkratší cesty mezi těmito dvěma vrcholy. Vzdáleností vrcholu $v \in V$ od množiny vrcholů $W \subset V$ nazveme minimální vzdálenost mezi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

Definice 5. Nechť $n \in \mathbb{N}$ a $\mathcal{P} = \{P_1, \ldots, P_n\}$ 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

1.2 Strom

Nyní zaveďme základní pojmy týkající speciální třídy grafů nazývané stromy [24].

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 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:

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

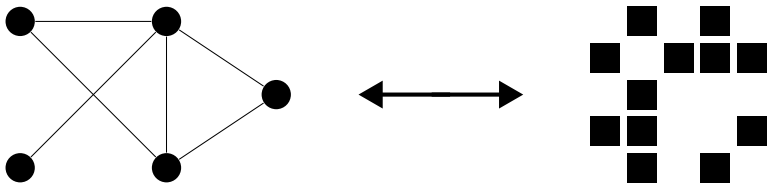
V kapitole 2.3 budeme potřebovat **Laplaceovu matici** Q grafu G. Jedná se o matici o rozměrech $n \times n$ definovanou následovně:

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

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.

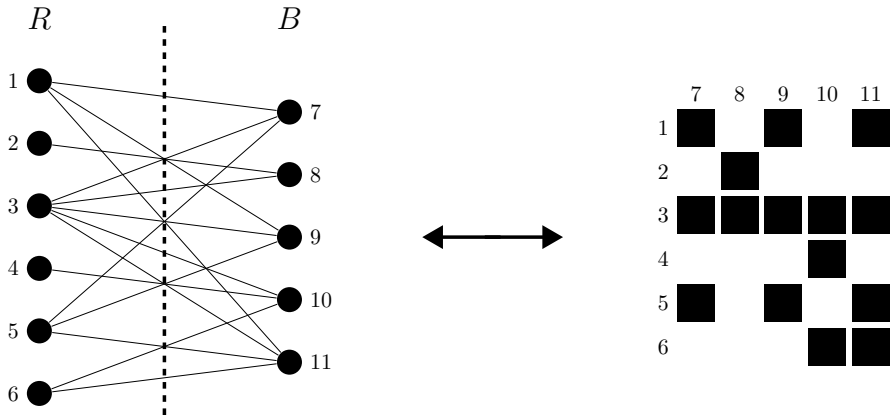


Obrázek 1.2: Příklad grafu a jemu odpovídající struktury matice

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 chceme s jeho pomocí reprezentovat matici A, pokládáme $|R| = m$, $|B| = n$ a $E = \{(v_i, v_j) \mid v_i \in R, v_j \in B, a_{ij} \neq 0\}$. Příklad reprezentace struktury matice pomocí bipartitního grafu je znázorněn na obrázku 1.3.



Obrázek 1.3: Reprezentace struktury matice pomocí bipartitního grafu

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 [20], 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 a vylepšovacímu algoritmu podle Kernighana a Lina [23]. Na závěr kapitoly zmíníme algoritmus metody vnořených řezů (Nested Dissection) jakožto příklad víceúrovňového algoritmu pro dělení grafů.

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 [37] a nezanedbatelnou roli hraje také při výrobě mikroprocesorů metodou VLSI nebo při řešení velkých systémů lineárních rovnic [23, 35]. Často dochází k omezení se na dělení grafu na dva podgrafy, v této práce se však pokusíme prozkoumat i dělení grafu na více částí.

Podmínky, které 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 s použitím dodatečných kritérií s důrazem na rozklady matic.

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.

Definice 9. Mějme graf $G = (V, E)$, $|V| = n$. **Rozdělením grafu** G na k podgrafů $G_1, \ldots, G_k$ nazveme rozklad množiny vrcholů V na vzájemně disjunktní podmnožiny $V_1, V_2, \ldots, V_k$, které indukují podgrafy $G_1, \ldots, G_k$.

Poznámka 3. Rozdělení grafu běžně zapisujeme pomocí vektoru P o délce n takového,

že pro každý vrchol $v_i \in V$ je index podgrafu, v němž se vrchol v nachází, určen i-tou složkou vektoru P.

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.

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 $\delta(V_1, \ldots, V_k)$. Jinou variantou je hledat rozdělení grafu pomocí vrcholového separátoru. **Vrcholovým separátorem** S_G grafu G nazveme podmnožinu množiny vrcholů V takovou, že odstraněním všech vrcholů náležících do S_G 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 [36]. To nám ale nic neříká o řešení problému nalezení optimálního hranového nebo vrcholového separátoru [30].

Ukazuje se, že rozhodovací problém pro optimální rozdělení grafu, pokud je možné, je NP-úplný [12]. Existují však algoritmy, které rozdělí graf v rozumném čase, přičemž kvalita jimi nalezeného rozdělení bude poměrně dobrá [28] vzhledem k povaze úlohy.

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 celkovou dobu běhu algoritmu [22], proto je tento postup využíván v profesionálních softwarových nástrojích pro dělení grafů [16, 18]. 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, 17]. 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 $(G_i)_0^m$, který je formálně faktorgrafem. 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 10. Mějme graf $G = (V, E)$. **Párováním** grafu G nazveme podmnožinu množiny hran E, pro kterou platí, že žádné dvě hrany z této množiny nemají společný koncový bod. **Maximálním párováním** grafu G nazveme párování grafu G obsahující nejvyšší možný počet hran. Pokud je graf G vážený reálným ohodnocením c, nazveme maximálním párováním grafu G párování grafu G takové, že součet vah hran, které jsou v tomto párování grafu obsaženy, je nejvyšší možný.

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 [8, 26]. Aby došlo k rychlému zmenšení počtu vrcholů grafu, je vhodné volit maximální párování [18]. Druhý postup hledání vhodných množin vrcholů pro sloučení je založen na spojování skupin vrcholů, v nichž je vysoká hustota hran [17].

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.

Tato fáze víceúrovňového dělení grafu může být vynechána [21]. V takovém případě ve zhrubovací fázi víceúrovňového dělení pokračujeme až do té doby než má výsledný graf G_m počet vrcholů roven počtu částí, na které chceme graf G rozdělit. Každý z vrcholů grafu G_m pak přiřadíme do jiné části rozdělení P_m a můžeme přímo přistoupit k projekci tohoto rozdělení na původní graf.

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 [23], 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 [22].

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:

$$\vec{x}^T Q \vec{x} = \vec{x}^T D \vec{x} - \vec{x}^T A_G \vec{x} = \sum_{i=1}^{n} d_i x_i^2 - 2 \sum_{(i,j)\in E} x_i x_j =$$
$$= \sum_{(i,j)\in E} (x_i - x_j)^2 = \sum_{\substack{i\in V_1, j\in V_2 \\ (i,j)\in E}} (x_i - x_j)^2 = 4|\delta(V_1, V_2)|,$$

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.

$$4|\delta_{min}(V_1, V_2)| = \min_{\substack{x_i=\pm 1 \\ \sum_{i=1}^{n} x_i = 0}} \vec{x}^T Q \vec{x}.$$

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 [19] 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 [17]. Proto se v takovém případě často 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 plošných spojích [23]. 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 KL algoritmus přímo jako dělicí algoritmus, lze vstupní rozdělení 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 KL algoritmus obvykle používá v kombinaci s jiným algoritmem pro dělení grafů, případně jako součást většího celku - například víceúrovňového dělení grafu ve fázi projekce hrubého rozdělení na vstupní graf (viz kapitola 2.2). V tomto případě je vzhledem ke specifickému charakteru úlohy běžně využívána modifikace tohoto algoritmu, která bere v úvahu pouze vrcholy blízko separátoru.

Popišme KL algoritmus pro graf rozdělený na dva podgrafy. Kdyby byl graf obecně rozdělen na k částí, mohli bychom použít tento algoritmus na jednotlivé dvojice podgrafů. Existují i algoritmy založené na KL algoritmu, které přerozdělují graf rozdělený na více než dvě části. Například v [43] najdeme algoritmus pro přerozdělování grafu rozděleného na 4 části.

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.

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 jde 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 velikosti hranového separátoru.Abychom mohli popsat reálný zisk při výměně vrcholů, je vhodné ještě zavést pro $u, v \in V$ veličinu C_{uv}.

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

Reálný zisk při výměně vrcholů u a v ležících ve vzájemně různých částech rozdělení grafu G je pak 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 $w_j \in V_2$ 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 [23]. Nejznámějším praktickým vylepšením algoritmu podle Kernighana a Lina je algoritmus publikovaný v [10], který optimalizuje rychlost výpočtu a aktualizaci zisku pro jednotlivé vrcholy.

2.5 Metoda vnořených řezů

Metoda vnořených řezů byla navržena a publikována v roce 1973 Alanem Georgem [13]. 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. V této práci jej uvádíme pro úplnost jakožto příklad algoritmu, který je již ze své podstaty zaveden jako víceúrovňový i přestože se striktně nedrží víceúrovňového schématu. Toto schéma z něj však převzalo myšlenku jak vytvářet hierarchii ale samotná metoda dělení na jednotlivých úrovních a techniky jak hledat oblasti byly nahrazeny. Dalším specifikem metody vnořených řezů je, že se tento algoritmus omezil na dělení grafu na dvě části, zatímco výše popsané víceúrovňové schéma toto omezení nemá. Metodu vnořených řezů můžeme dále vnímat i jako algoritmus pro hledání vhodného očíslování vrcholů grafu.

Metoda vnořených řezů je teoreticky velmi dobře popsána pro některé speciální typy grafů, například pro čtvercové sítě. Pro obecné grafy bylo navrženo několik heuristických algoritmů, z nichž jeden popíšeme.

2.5.1 Jeden řez grafu

Mějme graf $G = (V, E)$ s maticí sousednosti A_G a nechť $|V| = n$. Nechť $S \subset V$ je vrcholovým separátorem grafu G, jehož odebráním dostaneme dva podgrafy grafu G s množinami vrcholů C_1, C_2. Mějme dále zobrazení $\varphi : V \to \{1, 2, \ldots, n\}$, které každému vrcholu $v_i \in V$ přiřadí index i. Pokud pro zobrazení φ platí

1. $\varphi^{-1}(\{1, \ldots, |C_1|\}) = C_1$
2. $\varphi^{-1}(\{|C_1| + 1, \ldots, |C_1| + |C_2|\}) = C_2$
3. $\varphi^{-1}(\{|C_1| + |C_2| + 1, \ldots, n\}) = S$,

pak matici sousednosti A_G grafu G lze blokově zapsat ve tvaru

$$A_G = \begin{pmatrix} A_1 & O & V_1 \\ O & A_2 & V_2 \\ V_1^T & V_2^T & A_S \end{pmatrix}, \tag{2.2}$$

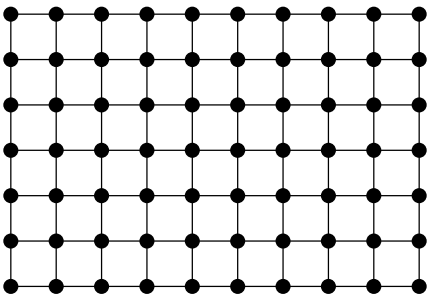
kde A_1, A_2 jsou matice sousednosti podgrafů $G(C_1)$, $G(C_2)$, dále V_1, resp. V_2 jsou matice charakterizující propojení vrcholů z množiny C_1, resp.C_2 s vrcholy ze separátoru S a matice O značí nulovou matici. Pro následnou manipulaci s maticí sousednosti A_G je výhodné, aby nulové matice O měly co nejvíce prvků.

Cílem jednoho řezu v metodě vnořených řezů je najít vrcholový separátor S a zobrazení φ splňující podmínky výše.

2.5.2 Metoda vnořených řezů pro čtvercové sítě

Popišme metodu vnořených řezů nejprve pro speciální třídu grafů nazývaných čtvercové sítě, se kterými se v praxi můžeme často setkat při diskretizaci složitých systémů.

Čtvercovou sítí o rozměrech $m \times l$, kde $m \leq l$ rozumíme graf o $N = ml$ vrcholech, jehož příklad můžeme vidět zobrazený na obrázku 2.1.



Obrázek 2.1: Čtvercová síť o rozměrech $m = 7$ a $l = 10$

Jeden řez čtvercové sítě: Mějme čtvercovou síť $N = (V, E)$ o rozměrech $n \times n$ a nechť $S_0 \subset V$. Nechť dále vrcholy z množiny S_0 leží v síti na jedné přímce a S_0 je vrcholovým separátorem sítě N, jehož odebráním dojde k rozpadnutí sítě N na dva podgrafy s množinami vrcholů V_1 a V_2 s pokud možno stejným počtem prvků. Z konstrukce S jsou tyto podgrafy zřejmě také čtvercovými sítěmi. Očíslujme nyní postupně po řádcích vrcholy z množin V_1 a V_2 a nakonec očíslujme vrcholy S_0. Toto očíslování nazýváme jednoúrovňovědělicím (one-level dissection ordering) a vidíme, že zřejmě splňuje podmínky kladené výše na zobrazení φ.

Vnořené řezy pro čtvercovou síť: Abychom získali očíslování pro metodu vnořených řezů, pokračujeme v dělení komponent s množinami vrcholů V_1, V_2. Obdobným způsobem jako v prvním kroku vybereme S_1, resp. S_2 tak, aby byla vhodným vrcholovým separátorem $G(V_1)$, resp. $G(V_2)$. Nyní opět každou část přečíslujeme tak, aby vrcholy v separátoru měly nejvyšší čísla. Tím se nám podaří docílit toho, že nejen matice sousednosti, ale i její libovolný diagonální blok má rekurzivně tvar (2.2).

2.5.3 Dělení obecných grafů metodou vnořených řezů

Pro použití metody vnořených řezů pro obecné grafy existují různé přístupy [29]. V této kapitole popíšeme jeden z nich. Jedná se o heuristický algoritmus fungující na principu hledání struktury úrovní daného grafu a následného rozdělení grafu podle úrovně "uprostřed".

Mějme souvislý graf $G = (V, E)$. Algoritmus pro jeho rozdělení využívající metodu vnořených řezů pak vypadá následovně.

1. Položíme $W = V$ a $N = |V|$.
2. Nalezneme souvislou komponentu $G(C)$ grafu $G(W)$.
3. Vezmeme libovolný vrchol $v \in C$. Vrchol v je vhodné jej volit tak, aby byl daleko od centra grafu [27] $G(C)$. Takový vrchol se nazývá periferní.
4. Vytvoříme strukturu úrovní $\mathcal{L} = \{L_0, \ldots, L_l\}$ grafu $G(C)$ vycházející z vrcholu v, tj. rozdělíme množinu vrcholů C grafu $G(C)$ na vzájemně disjunktní množiny $L_1, \ldots, L_l$ definované následovně:

$$L_0 := v$$
$$L_i := \mathrm{adj}_{G(C)}(L_{i-1}) \setminus \bigcup_{k=0}^{i-1} L_k \quad \text{pro } i \in \{2, \ldots, l\}$$

5. Separátor S grafu $G(C)$ volíme následovně. Pokud $l \leq 2$ položíme separátor roven C. Jinak definujeme $j := \left\lfloor \frac{l+1}{2} \right\rfloor$ a separátor položíme roven

$$S = \left\{ y \in L_j \mid \mathrm{adj}_{G(C)}(y) \cap L_{j+1} \neq \emptyset \right\}$$

6. Očíslujeme vrcholy separátoru čísly $N - |S| + 1, \ldots, N$
7. Položíme $W := W \setminus S$ a $N := N - |S|$.
8. Pokud $W \neq \emptyset$ vrátíme se na krok 2, jinak skončíme.

Kapitola 3

Rozklady matic

Při řešení maticových úloh velkých rozměrů je často výhodné 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 [15] nebo Lanczosově algoritmu [25, 34]. Rozklady matic nám mohou pomoci odhalit i teoretické vlastnosti a strukturu matic a odpovídajících maticových problémů. Příkladem takového rozkladu je singulární rozklad (SVD decomposition) [15].

Rozklady matic můžeme rozdělit na dvě základní skupiny - úplné rozklady a neúplné rozklady. Úplné rozklady dokážou nalézt z teoretického hlediska přesný výsledek v konečně mnoha krocích (v praxi při výpočtech v konečné počítačové aritmetice tento výsledek přesný být nemusí). Naopak neúplné rozklady jsou iterativními metodami, tj. algoritmy, které získávají v každém svém kroku přesnější aproximaci řešení s použitím výsledku předchozí aproximace a k přesnému výsledku konvergují. Pro získání celkového řešení soustavy je pak třeba je kombinovat s nějakou iterační metodou [2]. Vzhledem k našemu cíli zkoumat Choleského rozklad, jakožto zástupce úplných rozkladů, se v dalším textu nebudeme neúplnými rozklady hlouběji zabývat.

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 [6].

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 [5].

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.

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 [11]. Pro pozitivně definitní matice je však stabilita zajištěna i bez toho [46].

Mějme matici A o rozměrech $N \times N$. Symetrickou eliminací matice A nazveme hledání konečných posloupností $(A_j)_{j=0}^{N-1}$, $(L_j)_{j=1}^{N-1}$ 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 Choleského faktor matice A. Jinými slovy se jedná o symetrickou eliminaci matice A, kde požadujeme, aby matice D byla rovna jednotkové matici. Z praktického hlediska jde o modifikaci Gaussovy eliminační metody pro symetrické pozitivně definitní matice. Základní informaci o existenci Choleského rozkladu pozitivně definitní matice nám dává následující věta, jejíž důkaz můžeme nalézt například v [15].

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 matic 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.

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 ovroubení Jiným přístupem, který můžeme zvolit pro získání Choleského rozkladu LL^T matice A je řádkové schéma využívající metodu ovroubení (bordering method) [9, 33]. 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 předpokládali, můžeme použít stejný postup. Rekurzivním použitím této metody dostáváme soustavu lineárních algebraických rovnic pro výpočet i-tého řádku matice L, kde $l_{i,j}$ značí (i,j)-tý prvek 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 pro výpočet prvků matice

L vzniknou 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 řídkých matic

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 trojúhelníková část má stejnou strukturu jako dolní trojúhelníková část 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 4. 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. Jedním z takových problémů je problém minimalizace zaplnění v Choleského faktoru dané matice pomocí permutace řádků a sloupců popsaný v kapitole 3.1.2.1, který lze vzhledem ke vztahu matic a grafů popsanému výše v kapitole 1.3 převést na problém dělení odpovídajícího 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 a zároveň co nejmenší. Zřejmě nám takto vzniká kritérium, které není zahrnuto v kritériích podle klasické definice - ta totiž neberou v potaz hustotu hran v podgrafech vzniklých dělením původního grafu, neboli v řeči matic počet nenulových prvků matic sousednosti jednotlivých podgrafů. Počet operací potřebných 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 hledat vhodné očíslování vrcholů daných podgrafů a 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 $A_{G_1}, \ldots, A_{G_k}$, označme

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

Pokud existují $i, j \in \{1, \ldots, k\}$ takové, že matice $L_{G_i}^T$ a $L_{G_j}^T$ mají 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 podgrafů $G_1, \ldots, G_k$ a tím změnit matice sousednosti odpovídající jednotlivým podgrafům.

Poznámka 5. 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

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

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 vrcholový separátor byl malý.

Soustavu $L^T x = y$ poté vyřešíme obdobně.

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

1. Jak najít očíslování podgrafů vzniklých dělením původního grafu takové, aby k vypočítání Choleského rozkladu na jednotlivých oblastech bylo potřeba co nejméně operací?

2. Jak určit počet nenulových prvků v Choleského faktoru matice bez toho, abychom Choleského rozklad vypočítali?

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

4.1 Algoritmy pro očíslování vrcholů grafu

V této podkapitole popíšeme základní algoritmy pro očíslování vrcholů grafu. Zřejmě platí, že přečíslování vrcholů grafu odpovídá 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, jak jsme zmínili výše.

Na závěr kapitoly se zmíníme o topologickém číslování vrcholů stromu, které pro nás bude potřebné při popisu eliminačních stromů v kapitole 4.2.

4.1.1 Číslování vrcholů grafu v závislosti na vzdálenosti od separátoru

V tomto oddíle 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 $\text{adj}_G(v)$ tak, aby $G(\text{adj}_G(v))$ byla klika v grafu G.
4. Pokud nejsou všechny vrcholy očíslované, položíme `j := j + 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 algoritmu popsaného v 4.1.2 přidáme kritérium vzdálenosti od separátoru popsané v 4.1.1. Nejprve tedy nalezneme množinu všech vrcholů grafu G, které mají minimální stupeň a poté z nich zvolíme ten, který je nejdál od vrcholového separátoru.

Druhou možností, jak je výše zmíněná očíslování možné kombinovat je použít jejich konvexní kombinaci. Tímto způsobem nemusíme kombinovat pouze očíslování popsaná v oddílech 4.1.2 a 4.1.1, ale můžeme jej použít na libovolná očíslování. Mějme množinu vrcholů V, $|V| = n$ a dvě různá očíslování $\varphi, \psi : V \to \{1, \ldots, n\}$. Nechť dále $\alpha \in [0, 1]$. Napočítáme pro každý vrchol v hodnotu výrazu $\varphi(v) + (1 - \alpha)\psi(v)$. Smíšené očíslování vytvoříme tak, že vrcholy seřadíme podle této hodnoty od nejmenší po největší. Ač by se mohlo zdát, že tento postup nám nemůže přinést nic nového, nemusí to být pravidlem[42, 38].

4.1.4 Topologické číslování vrcholů stromu

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

Definice 11. 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

V této kapitole se budeme zabývat eliminačními stromy a jejich významem pro rozklady řídkých matic [31, 32]. 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 sloupce, 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 [39] (Matici PA_TP^T můžeme vnímat pouze jako přečíslování řádků a sloupců matice A_T).

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

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

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 12. Eliminačním stromem matice A nazveme graf $G(F_t)$ popsaný výše a značíme jej $T(A)$. Podstrom eliminačního stromu $T(A)$ s kořenem x_j značíme $T[x_j]$. Nemůže-li dojít k záměně, značíme množinu vrcholů tohoto stromu taktéž $T[x_j]$.

Díky této definici můžeme funkci $\texttt{PARENT}$ přirozeně rozšířit na matici A následovně:

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

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 $i, j \in \{1, \ldots, n\}$, $i > j$. 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 Využití a konstrukce eliminačních stromů

Z definice eliminačních stromů je zřejmé, že budou obsahovat velké množství informací o rozkladu matice a procesu jeho vytváření. Například v [41] 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 [32]. Uveďme větu, která dává do souvislosti nenulové prvky v Choleského faktoru a strukturu eliminačního stromu [31].

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 $[L]_{i,j} \neq 0$ právě tehdy, když vrchol v_j je v eliminačním stromě následníkem vrcholu v_k takového, že $[A]_{i,j} \neq 0$.*

Tato věta nám dává návod, jak vypočítat počet nenulových prvků v Choleského faktoru. Uveďme algoritmus popsaný v [32], 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 [47]. Námi popsaný algoritmus je poměrně jednoduchý. Efektivnější, ale složitější algoritmus pro výpočet počtu nenulových prvků můžeme najít v [14]. Dále v [1] 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, aby odpovídalo vstupní matici. Eliminační strom je popsán pomocí vektoru `PARENT`, který byl zmíněn výše.

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.

Díky Algoritmu 1 dostáváme odpověď na otázku, 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.

```
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 ([32], upraveno)

Uveďme základní algoritmus pro sestrojení eliminačního stromu matice bez výpočtu jejího Choleského rozkladu (Algoritmus 2). Dále popišme jeho vylepšení (Algoritmus 3), v němž pro zrychlení Algoritmu 2 používáme pomocný vektor `ANCSTR`, který nám zkrátí procházení stromu směrem od listů ke kořeni. V tomto pomocném vektoru je pro každý vrchol zaznamenáván jeho poslední zpracovaný předek. Oba algoritmy jsou převzaté z [31].

Složitost Algoritmu 3 lze seshora odhadnout jako $O(|E| \log_2 n)$ [44]. Existuje ještě efektivnější verze tohoto algoritmu, kterou můžeme nalézt v [45].

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;
    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 ([31], upraveno)

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

Algoritmus 3: Vylepšený algoritmus pro hledání eliminačního stromu ([31], upraveno)

Kapitola 5

Algoritmizace a implementace

Cílem programu, který jsme implementovali, je analyzovat rozdělení grafu získané za pomoci profesionální numerické knihovny pro dělení grafů METIS a ukázat, že nemusí být vyvážené vzhledem k výpočtu Choleského rozkladu na vzniklých oblastech. Dále jsme testovali závislost kvality rozdělení vzhledem k Choleského rozkladu na očíslování grafu. Prozkoumali jsme, zda lze lehkou modifikací vrcholového separátoru dosáhnout zlepšení vyváženosti počtu operací potřebných pro Choleského rozklad na jednotlivých oblastech. Vzhledem k analytickému charakteru této práce jsme zvolili implementaci, která si neklade za cíl používat co nejefektivnější algoritmy, ale dává přednost dobré přehlednosti a názornosti programu.

V naší implementaci jsme se omezili na dělení grafu na dvě části pomocí vrcholového separátoru. Toto omezení nám přirozeně vyplynulo z implementace, jelikož knihovna METIS neposkytuje rutiny pro dělení grafu pomocí vrcholového separátoru na více než dvě části. Námi implementovaný kód je napsán obecně pro graf rozdělený na n částí. Implementací jiného grafového děliče nebo použitím algoritmů pro převod hranového separátoru na vrcholový by tedy bylo možné výsledky rozšířit i pro dělení grafu na $n > 2$ částí.

Námi implementovaný program se volá z příkazové řádky a jeho funkce závisí na argumentech, s nimiž je zavolán. Tyto argumenty budou popsány níže v odstavci TODO: ref. Základní struktura programu, která je nezávislá na tom, s jakými argumenty je zavolán, je následující. Nejprve dojde k načtení matice, která je poté rozdělena na TODO: několik částí pomocí profesionální knihovny pro dělení grafů METIS. Následně v hlavní smyčce programu probíhá samotné zpracování matice. Nejprve jsou za použití rozdělení získaného pomocí knihovny METIS vytvořeny CSR reprezentace podgrafů. Následně dojde k přečíslování vrcholů jednotlivých podgrafů, pokud se jedná o poslední iteraci hlavní smyčky a je to požadováno. Poté je zkonstruován eliminační strom a s jeho pomocí určen počet nenulových prvků v jednotlivých Choleského faktorech matic odpovídajících jednotlivým podgrafům. Pokud je to požadováno, je posunutím vrcholového separátoru vytvořeno nové rozdělení grafu odpovídajícího vstupní matici a hlavní smyčka spuštěna znovu. Dále náš program podporuje výstup v jazyce DOT, který může být použit pro zobrazení grafu a jeho výsledného rozdělení pomocí knihovny GraphViz.

Náš program byl implementován v programovacím jazyce Fortran, pro použití knihovny METIS napsané v C++ jsme vytvořili rozhraní pro volání rutin napsaných v C++ z Fortranu. Pro opakované spouštění programu za účelem testování jsme používali skript napsaný v jazyce Python.

5.1 Formát a uložení vstupní matice

V této podkapitole bude specifikován formát vstupní matice a způsob její reprezentace v programu.

5.1.1 Podporované formáty vstupních matic

Náš program pracuje s maticemi uloženými v souborech v Harwell-Boeingově formátu [3]. Konkrétně je schopen zpracovat reálné symetrické matice ve formátech RSA a PSA. Tyto dva formáty mají navzájem si podobnou strukturu, ve formátu RSA jsou navíc uloženy i numerické hodnoty matice. Ty však v našem programu nebereme v úvahu, protože se nezabýváme výpočtem numerických hodnot v Choleského faktoru zpracovávané matice, ale pouze jeho strukturou. Popis kolekcí matic, z nichž jsme čerpali při testování našeho programu, lze nalézt například v [4] a [7]. Tyto matice bývají standardně uloženy v souborech s příponou `.rsa` nebo `.rb`.

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 vytvořili v souboru `testing.f90`. Všechny tyto matice jsou uloženy nebo generovány v programu a jejich zpracování zve vyvolat pomocí argumentu `-mt` z příkazové řádky.

5.1.2 Reprezentace matice v programu

Matice je v našem programu uložena v **CSR formátu** (compressed sparse row format) [35, 40], který je běžně využívaným formátem sloužícím pro reprezentaci řídkých matic a grafů.

Popišme CSR formát pro řídkou matici A o rozměrech $n \times n$, která obsahuje n_e nenulových prvků. Tuto matici v programu reprezentujeme pomocí dvou polí: pole ukazatelů `ia` o délce $n+1$ a pole `ja` o délce n_e. Pro $i \in \{1, \ldots, n\}$ jsou v poli `ja` na pozicích `ia(`i`)`, $\ldots$, `ia(`$i+1$`)` -1 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 ]
```

Vzhledem k tomu, že v našem programu pracujeme pouze se strukturou matice, můžeme při reprezentaci matice vynechat pole `aa`. Zřejmě je pak výsledná reprezentace matice A zároveň reprezentací neorientovaného grafu, kterému matice A odpovídá a můžeme tedy 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 1.2

5.2 Knihovna METIS pro dělení grafu

Pro samotné dělení grafu na k částí jsme využili knihovnu pro dělení grafů METIS [20] 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`. TODO: cite url?

Zmíněná rutina není popsána 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 byly 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 3 (po převedení do indexování od 1 používaného v jazyce Fortran). To určuje, ve které části rozdělení se daný vrchol nachází. Pro vrchol s indexem j ve vrcholovém separátoru je `part`$[j] = 3$.

5.3 Tvorba podgrafů

Pro tvorbu podgrafů při znalosti vstupního rozdělení jsme použili námi napsanou rutinu `createSubgraphs`. Hlavní princip této rutiny lze popsat následovně. Jejím vstupem je graf v CSR formátu a vektor `part` popisující rozdělení tohoto grafu. Výstupem této rutiny jsou jednotlivé podgrafy popsané pomocí vektoru `part` uložené v CSR formátech. Pro snadnou manipulaci s těmito podgrafy je ukládáme do dvou datových struktur `iap` a `jap`. Formát těchto datových struktur je definován v `raggedmultiarray.f90` a byl převzat z TODO: cite? odkaz. i-tý podgraf ja pak uložen v polích `iap%vectors(`i`)%elements`, `jap%vectors(`i`)%elements`.

5.4 Číslování

V našem programu jsme implementovali 4 způsoby číslování vrcholů,mezi nimiž lze přepínat pomocí argumentu `-ot` z příkazové řádky.

První metodou, kterou jsme implementovali, je číslování vrcholů grafu v závislosti na vzdálenosti od separátoru popsané v kapitole 4.1.1. Jedná se o rutinu `orderByDistance`. Druhou implementovanou metodou je číslování vrcholů pomocí metody minimálního stupně popsané v odstavci 4.1.2. Tuto metodu jsme implementovali jako rutinu `orderByMD`. Třetí a čtvrtá metoda obsažená v našem programu používá smíšené číslování popsané v kapitole 4.1.3. Rutina `orderMixed` používá pro výběr vrcholu kombinaci kritérií metody minimálního stupně a metody maximální vzdálenosti od separátoru. Naproti tomu v rutině `orderCoefMixed` je zvlášť napočítáno číslování v závislosti na vzdálenosti od separátoru a pomocí metody minimálního stupně. Poté dojde ke konvexní kombinaci těchto očíslování s koeficientem $C \in [0, 1]$, který je vstupním parametrem této rutiny.

Výstupem všech výše zmíněných rutin jsou vektory `ordperm` a `invordperm` popisující nově získanou permutaci vrcholů.

Číslování vždy aplikujeme na původní graf a poté ho pomocí rutin `partOrdering` a `applyOrdering` projektujeme na podgrafy získané dříve. Tím je zajištěno, že metody používající číslování pomocí metody minimálního stupně budou brát v potaz hrany, které spojují vrcholy v jednotlivých částech s vrcholy ve vrcholovém separátoru.

5.5 Tvorba eliminačního stromu a výpočet Choleského rozkladu

Tvorbu eliminačního stromu a následný výpočet Choleského rozkladu provádíme pro každý podgraf zvlášť. Abychom zjistili počet prvků v Choleského faktoru matice, potřebujeme nejprve pomocí Algoritmu 3 popsaného v oddíle 4.2.2 sestrojit eliminační strom odpovídající matice. Pro tvorbu eliminačního jsme použili rutinu `eltree2` TODO: cite z balíku?. Poté jsme pomocí rutiny `colcnts` TODO: cite z balíku? využívající Algoritmus 1 vypočítali počet nenulových prvků v jednotlivých sloupích Choleského faktoru matice a sečtením těchto hodnot jsme dostali výsledný počet nenulových prvků.

Rutina `colcnts` nezahrnuje do počtu prvků v Choleského faktoru diagonální prvky matice. Ty však pro celkový výsledek nejsou důležité, protože na diagonále nemůže dojít ke vzniku zaplnění.

5.6 Přesun vrcholového separátoru

K přesunu vrcholového separátoru dojde pouze v případě, že si jej uživatel našeho programu explicitně vyžádá při volání programu z příkazové řádky pomocí argumentu `-mvs`. Přesun vrcholového separátoru je používán jako nástroj pro vyvážení počtu nenulových prvků v Choleského faktorech jednotlivých částí rozdělení. Proto je vrcholový separátor vždy přesouván směrem do části, v níž je zaplnění nejvyšší.

Označme G_{max} část rozdělení s největším zaplněním v Choleského faktoru. Pro přesun vrcholového separátoru S jsme použili jednoduchý způsob, kdy přidáme k vrcholovému separátoru $(G_{max} \cap \text{adj}(S))$. Následně z něj ubereme vrcholy, které ve vrcholovém separátoru původně byly, ale přidáním $(G_{max} \cap \text{adj}(S))$ v něm ztratily svůj význam.

Zřejmě tímto postupem může dojít u obecného grafu k výraznému zvětšení velikosti vrcholového separátoru. V běžných aplikacích by však toto zvětšení nemělo být natolik výrazné, aby nedávalo smysl přesun vrcholového separátoru prozkoumat.

5.7 Volání programu z příkazové řádky

Při volání našeho programu z příkazové řádky je třeba mu poskytnout informace o tom jakou matici a jakým způsobem má program zpracovat. K tomu slouží argumenty, s nimiž může být program volán, které jsou blíže popsány v Tabulce 5.1.

Argument	Priorita	Funkce
`-mt [matrixtype]`	2	Používá se pro nastavení typu vstupní matice. Parametr `matrixtype` je povinný a jeho možné hodnoty jsou `RSA` pro matici načítanou ze souboru, `P[number]` pro matici o rozměrech `number`×`number` vzniklou při jednoduché pětibodové diskretizaci dvojrozměrné Poissonovy rovnice, `T[number]` pro testovací matici s indexem `number` popsanou v souboru `testing.f90`. Tento argument je povinný, pokud není použit argument `-f`.
`-f [path]`	3	Slouží k nastavení cesty (`path`) ke vstupnímu souboru matice. Tento argument je ignorován, pokud je použit argument `-mt` s parametrem `P[number]` nebo `T[number]`, v opačném případě je povinný.
`-ot [orderingtype]`	2	Využívá se pro nastavení typu uspořádání vrcholů v poslední iteraci hlavního cyklu programu. Možné hodnoty povinného parametru `orderingtype` jsou `DIST` pro číslování vrcholů grafu v závislosti na vzdálenosti od separátoru, `MD` pro číslování vrcholů pomocí metody minimálního stupně, `MIX` pro číslování pomocí smíšeného číslování používajícího současně metodu minimálního stupně a vzdálenost od separátoru a `MIX[number]` pro konvexní kombinaci těchto očíslování s koeficientem `number` $\in [0, 1]$. Pokud není argument `-ot` uveden, nedojde k žádnému přeuspořádání vrcholů v částech.
`-mvs [number]`	2	Slouží k nastavení počtu přesunů vrcholového separátoru směrem do části rozdělení s větším zaplněním. Parametr `number` je povinný a může nabývat hodnot $0 - 20$. Pokud není argument použit, k přesunu vrcholového separátoru nedojde.
`-o [path]`	2	Užívá se pro vytvoření výstupního souboru ve formátu, který lze zobrazit pomocí softwarové knihovny GraphViz. Parametr `path` slouží k nastavení cesty k výstupnímu souboru. Pokud soubor již existuje, bude přepsán. Pokud argument není použit, k vypsání výstupu nedojde.
`-w`	2	Pokud je tento argument použit, program v průběhu svého běhu vypisuje informaci o tom, která jeho část právě probíhá.
`-h`	1	Pokud je tento argument použit, jsou všechny ostatní argumenty ignorovány a je pouze vypsána nápověda k programu.

Tabulka 5.1: Popis argumentů programu při volání z příkazové řádky

Kapitola 6

Výsledky

6.1 Informace o systému

Náš program byl testován na MacBook Pro (Retina, 15-inch, Mid 2015) s procesorem 2,5 GHz Intel Core i7, operační pamětí 16GB 1600 MHz DDR3 a grafickou kartou Intel Iris Pro 1536 MB.

Pro úspěšné zkompilování a slinkování projektu je třeba mít nainstalovaný GNU Make (testováno na verzi 3.81), gcc a gfortran (testováno pro verzi 6.3.0) a dělič grafů METIS (testováno pro verzi 5.1.0). TODO: dot-Grapvhiz TODO: citovat balik na razeni vrcholu

6.2 Dělení grafu na dvě části

6.3 Dělení grafu na víc částí

TODO: bude to?

Závěr

TODO: doplnit TODO: motivace: nelineární problém viz Tumovy stranky TODO: radkovy vs. sloupcovy algoritmus

Literatura

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

[2] 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.

[3] Ronald Boisvert, Roldan Pozo, and Karin A. Remington. The matrix market exchange formats: Initial design. *NISTIR*, 5935, 1997.

[4] Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

[5] Timothy A. Davis, Sivasankaran Rajamanickam, and Wissam M. Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numer.*, 25:383–566, 2016.

[6] Iain S Duff, Albert M Erisman, and John K Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Inc., New York, NY, USA, 1986.

[7] Iain S. Duff, Roger G. Grimes, and John G. Lewis. Users' guide for the harwell-boeing sparse matrix collection (release i), 1992.

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

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

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

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

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

[13] J. Alan George. Nested dissection of a regular finite element mesh. *SIAM J. on Numerical Analysis*, 10(2):345—-363, 1973.

[14] 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.

[15] 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.

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

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

[18] 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.

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

[20] 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.

[21] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, pages 113–122. CRC Press, 1995.

[22] 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.

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

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

[25] 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.

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

[27] John G. Lewis. Implementation of the gibbs-poole-stockmeyer and gibbs-king algorithms. *ACM Trans. Math. Softw.*, 8(2):180–189, June 1982.

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

[29] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. Generalized nested dissection. 16(2):346–358, 1979. Exported from https://app.dimensions.ai on 2018/11/19.

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

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

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

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

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

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

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

[37] 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.

[38] John Ker Reid and Jennifer A Scott. Ordering symmetric sparse matrices for small profile and wavefront. *International Journal for Numerical Methods in Engineering*, 45(12):1737–1755, 1999.

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

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

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

[42] Scott Sloan. A fortran program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering*, 28:2651 – 2679, 11 1989.

[43] P. R. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *IEEE Trans. on Circuits and Systems*, 35(3):294–303, 1988.

[44] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, April 1975.

[45] Robert Endre Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.

[46] 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.

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

TODO: udelat poradek v citacich
TODO: nemam citovane dva zdroje ze zadani