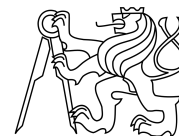
ESKÉ VYSOKÉ U ENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMA NÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁ SKÉ PRÁCE

Název: Efektivní LU rozklad pro ídké matice
Student: Gabriela Turcajová
Vedoucí: Ing. Ivan Šime ek, Ph.D.
Studijní program: Informatika
Studijní obor: Teoretická informatika
Katedra: Katedra teoretické informatiky
Platnost zadání: Do konce letního semestru 2016/17

Pokyny pro vypracování

1) Seznamte se s r znými formáty uložení ídkých matic, zejména se sou adnicovým, CSR a CSC formátem.

2) Naimplementujte LU rozklad pro ídké matice pomocí Doolittova algoritmu pro výše uvedené formáty.

3) Seznamte se s heuristikami, které snižují po et nov vzniklých prvk v matici. Naimplementujte pro výše uvedené formáty minimáln tyto: Multiple Minimum Degres Ordering, Reverse Cuthill-Mckee, Markowitz strategie.

4) Výsledné implementace otestujte na ídkých maticích získaných z ve ejn dostupných zdroj (MatrixMarket, ...) a porovnejte asovou a pam ovou náro nost pro testované matice.

Seznam odborné literatury

Dodá vedoucí práce.

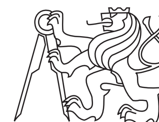
L.S.

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
d kan

V Praze dne 20. íjna 2015

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretické informatiky



Bakalářská práce

Efektivní LU rozklad pro řídké matice

Gabriela Turcajová

Vedoucí práce: Ing. Ivan Šimeček, Ph.D.

11. května 2016

Poděkování

Ráda bych touto cestou poděkovala vedoucímu bakalářské práce panu Ing. Ivanu Šimečkovi, Ph.D. za přínosné informace a odbornou pomoc, kterou mi poskytl při zpracování mé bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 11. května 2016 . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Gabriela Turcajová. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Turcajová, Gabriela. *Efektivní LU rozklad pro řídké matice*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Cílem práce je implementace Doolitlova algoritmu pro LU rozklad řídkých matic a seznámení s různými heuristikami pro snížení tvorby nenulových prvků, konkrétně s algoritmy Multiple Minimum Degree, Reverse Cuthill-McKee a Markowitzova strategie. Dále se práce zabývá implementací těchto algoritmů a jejich porovnáním z hlediska časové a paměťové složitosti.

Klíčová slova LU rozklad, řídké matice, Doolittlův algoritmus, Multiple Minimum Degree, Reverse Cuthill-McKee, Markowitzova strategie, C++

Abstract

The goal of this thesis is to implement Doolittle algorithm for LU decomposition of sparse matrices and to become acquainted with heuristics for reducing fill-in, concretely with algorithms Multiple Minimum Degree, Reverse Cuthill-McKee and Markowitz strategy. The thesis also goes in their implementation and comparasion of their time and space complexity.

Keywords LU decomposition, sparse matrices, Doolittle algorithm, Multiple Minimum Degree, Reverse Cuthill-McKee, Markowitz strategy, C++

Obsah

Úvod **1**

1 Popis problému **3**
1.1 Základní pojmy . . . 3
1.2 Formáty uložení řídkých matic . . . 4
1.3 Doolittlův algoritmus pro nalezení LU rozkladu matice . . . 8

2 Heuristiky pro snížení počtu nově vzniklých nenulových prvků **11**
2.1 Základy z teorie grafů . . . 11
2.2 Heuristiky pro symetrické matice . . . 13
2.3 Heuristiky pro nesymetrické matice . . . 18
2.4 Přínos heuristik . . . 21

3 Existující řešení **23**
3.1 Akademické práce . . . 23
3.2 Software . . . 24

4 Implementace **25**
4.1 Použité prostředky . . . 25
4.2 Uložení dat v paměti . . . 25
4.3 Třídní rozdělení . . . 26
4.4 Doolittlův algoritmus . . . 26
4.5 Implementace heuristik . . . 27

5 Testování **29**
5.1 Testovací data . . . 29
5.2 Výpočetní hardware . . . 32
5.3 Nastavení kompilátoru . . . 33
5.4 Měření na vygenerovaných datech . . . 33
5.5 Měření na datech z repozitáře „Matrix Market“ . . . 34

5.6 Porovnání vzniku nenulových prvků s existujícím řešením . . . 36

Závěr **41**

Literatura **43**

A Seznam použitých zkratek **47**

B Obsah přiloženého CD **49**

Seznam obrázků

2.1 Matice sousednosti neorientovaného grafu 12
2.2 Průběh heuristiky Minimum Degree Ordering 14
2.3 Graf vstupní matice $\mathbb{A}$ 17
2.4 Doolittle bez využití heuristik 21
2.5 Doolittle s využitím heuristik 21

5.1 Vizualizace 1138 BUS 29
5.2 Vizualizace 662 BUS 30
5.3 Vizualizace BCSSTK08 30
5.4 Vizualizace BCSSTK11 30
5.5 Vizualizace BCSSTK13 31
5.6 Vizualizace BCSSTK14 31
5.7 Vizualizace BCSSTK19 32
5.8 Vizualizace BFW782B 32
5.9 Rychlosti heuristik v závislosti na hustotě vstupní matice 35
5.10 Rychlosti heuristik v závislosti na rozměru vstupní matice 35
5.11 Počet nenulových prvků ve výsledné matici $\mathbb{L}$ v závislosti na počtu nenulových prvků vstupní matice 36
5.12 Rychlost heuristik na datech z repozitáře „Matrix Market“ 37
5.13 Počet nenulových prvků ve výsledné matici $\mathbb{L}$ na datech z repozitáře „Matrix Market“ 37
5.14 Porovnání počtu nenulových prvků ve výsledných maticích $\mathbb{L}$ a $\mathbb{U}$ s metodou QR a Choleského 38

Seznam tabulek

5.1 Výsledky měření času a počtu nenulových prvků 38

Seznam algoritmů

1	Doolittlova metoda pro LU rozklad	8
2	Minimum Degree	15
3	Cuthill-McKee	16
4	Markowitzova strategie	19

Úvod

Matice jsou v dnešní době nedílnou součástí mnohých oborů. Jsou využívány například v informatice, elektrotechnice nebo jaderném inženýrství. Jejich řešení, zejména pak matic řídkých, může být však často časově i paměťově velmi náročnou operací. LU rozklad matic může značně zjednodušit vyřešení matice, spočítání jejího determinantu nebo určení její inverze. Práce se proto věnuje efektivní implementaci LU rozkladu pro řídké matice.

Na implementaci LU rozkladu byl použit Doolittlův algoritmus. Jelikož se jedná o řídké matice, jejichž většina prvků je nulová, jsou v práci popsány vybrané formáty uložení těchto typů matic. Dále se práce zabývá heuristikami, které mají omezit vytváření nových nenulových prvků. Na implementaci byly vybrány následující heuristiky: Multiple Minimum Degree Ordering, Reverse Cuthill-McKee a Markowitz strategie. Hlavním cílem této práce je právě porovnání těchto heuristik z hlediska časové a paměťové složitosti.

V první části práce je popsán teoretický základ z lineární algebry, který je nezbytný pro popsání použitých metod. Následuje popis Doolittlova algoritmu a vybraných heuristik. Nakonec se práce zabývá existujícími řešeními a popisuje implementaci efektivního LU rozkladu matic.

Na závěr jsou vybrané heuristiky testovány na maticích získaných z veřejně dostupných zdrojů a výsledky jsou porovnávány z hlediska časové a paměťové náročnosti pro testované matice.

Kapitola 1

Popis problému

Pro popsání problému je potřeba znát některé z definic z lineární algebry, které jsou v této kapitole popsány. Dále jsou zde uvedeny různé formáty pro uložení řídkých matic a nakonec je rozebrán Doolittlův algoritmus.

1.1 Základní pojmy

Matice [1] (Reálná) matice typu (m, n) je schéma

$$\mathbb{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}.$$

Stručněji ji budeme zapisovat $\mathbb{A} = (a_{i,j})$, kde $i = 1, 2, ..., m$ a $j = 1, 2, ..., n$.

Násobení matic [2] Nechť $\mathbb{A} = (a_{i,j})$ je matice typu (m, n) a $\mathbb{B} = (b_{j,k})$ je matice typu (n, p). Pak je definován *součin matic* $\mathbb{A} \cdot \mathbb{B}$ (v tomto pořadí) jako matice typu (m, p) takto: každý prvek $c_{i,k}$ matice $\mathbb{A} \cdot \mathbb{B}$ je dán vzorcem

$$c_{i,k} = a_{i,1}b_{1,k} + a_{i,2}b_{2,k} + \cdots + a_{i,n}b_{n,k} = \sum_{j=1}^{n} a_{i,j}b_{j,k}, \qquad i \in 1, ..., m, \qquad k \in 1, ..., p.$$

Čtvercová matice [1] Pokud má matice stejný počet řádků i sloupců, znamená to, že je typu (n, n), nazýváme ji *čtvercovou maticí řádu n.* V opačném případě hovoříme o *obdélníkové matici.*

Jednotková matice [2] Čtvercovou matici $\mathbb{E}$ typu (n, n) nazýváme *jednotkovou maticí*, pokud pro její prvky $e_{i,j}$ platí: $e_{i,j} = 0$ pro $i \neq j$ a $e_{i,j} = 1$ pro $i = j$.

Regulární a singulární matice [3] Buď $\mathbb{A}$ čtvercová matice. Existuje-li čtvercová matice $\mathbb{B}$ taková, že platí

$$\mathbb{A}\mathbb{B} = \mathbb{B}\mathbb{A} = \mathbb{E},$$

nazýváme matici $\mathbb{A}$ *regulární* a $\mathbb{B}$ *inverzní maticí* k matici $\mathbb{A}$. Pokud $\mathbb{A}$ není regulární, nazýváme matici $\mathbb{A}$ *singulární.*

Transponovaná matice [2] Nechť $\mathbb{A} = (a_{i,j})$ je matice typu (m, n). Matici $\mathbb{A}^T = (a_{j,i})$, která je typu (n, m), nazýváme *transponovanou maticí* k matici $\mathbb{A}$. Matice $\mathbb{A}^T$ tedy vznikne z matice $\mathbb{A}$ přepsáním řádků matice $\mathbb{A}$ do sloupců matice $\mathbb{A}^\mathbb{T}$, respektive přepsáním sloupců matice $\mathbb{A}$ do řádků matice $\mathbb{A}^T$.

Symetrická matice [2] Čtvercová matice $\mathbb{A}$ se nazývá *symetrická*, pokud $\mathbb{A} = \mathbb{A}^T$.

Diagonála matice [2] Nechť $\mathbb{A} = (a_{i,j}$ je matice typu (n, n). *Diagonála matice* $\mathbb{A}$ je skupina jejích prvků $a_{1,1}, a_{2,2}, \ldots, a_{n,n}$. Prvek pod diagonálou je každý prvek $a_{i,j}$, pro který platí $i > j$. Prvek nad diagonálou je každý prvek $a_{i,j}$, pro který platí $i < j$.

Horní a dolní trojúhelníková matice [1] Řekneme, že matice $\mathbb{A}$ typu (m, n) je *horní trojúhelníková*, jestliže $a_{i,j} = 0$ pro všechna $i > j$. Řekneme, že matice $\mathbb{A}$ typu (m, n) je *dolní trojúhelníková*, jestliže $a_{i,j} = 0$ pro všechna $i < j$.

LU rozklad [4] *LU rozklad* je rozklad regulární čtvercové matice na součin dvou matic $\mathbb{A} = \mathbb{L} \cdot \mathbb{U}$, kde matice $\mathbb{L}$ je dolní trojúhelníková a $\mathbb{U}$ je horní trojúhelníková.

1.2 Formáty uložení řídkých matic

Tato část popisuje různé formáty vnitřního i vnějšího uložení určené pro řídké matice. V této sekci bylo čerpáno z [5], [6] a [7].

1.2.1 Definice řídké matice

Řídké matice jsou ty matice, které mají většinu prvků nulových, nicméně tato hranice není přesně definována. Orientačně volíme takové matice, které mají maximálně 10 % prvků nenulových.

Taková definice ovšem není přesná. Záleží totiž na tom, co máme v úmyslu s takovou maticí dělat. Komprimovaný formát uložení řídkých matic v počítači způsobuje pomalejší přístup k prvkům matice. V některých případech se nám tedy může vyplatit ukládání řídké matice v hustém formátu, pokud využíváme

rychlý algoritmus s častým přístupem k matici, jehož vlastnosti bychom zhoršili použitím některého formátu uložení pro řídké matice. Musíme se tedy před výběrem formátu uložení rozmyslet, zda-li upřednostníme rychlost algoritmu na úkor využité paměti či naopak.

1.2.2 COO formát

Souřadnicový formát *(Coordinate format)* se řadí mezi nejintuitivnější komprimované formáty. K uložení matice potřebuje tři jednorozměrná pole *row, col, data* o velikosti celkového počtu nenulových prvků v matici. V poli *data* jsou uloženy všechny nenulové prvky v pořadí zleva doprava, shora dolů, v polích *row*, resp. *col*, pak indexy řádků, resp. sloupců, daných prvků, tedy *data[i]* je hodnota na pozici $(row[i], col[i])$. Mezi výhody tohoto formátu patří rychlá konstrukce matice a jednoduchý převod do jiného formátu uložení. Naopak ale zabírá více paměti oproti ostatním formátům pro řídké matice.

Příklad uložení matice v COO formátu
Předpokládejme následující matici $\mathbb{A}$ definovanou:

$$\mathbb{A} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 1 \end{pmatrix}.$$

Její uložení ve formátu COO při indexování od jedné by bylo:

data	2	10	5	4	1	7	3	1
row	1	2	2	3	3	4	5	5
col	1	2	4	1	3	4	3	5

1.2.3 CSR formát

CSR *(Compressed Sparse Row)* formát je upravený souřadnicový formát. K uložení matice potřebuje také tři jednorozměrná pole *row_ptr, col_ind, data*, na indexy matice však slouží pouze pole *col_ind*. Stejně jako v souřadnicovém formátu jsou prvky matice uloženy v pořadí zleva doprava, shora dolů a v poli *col_ind* je uložen index sloupce daného prvku. V poli *col_ptr* je oproti tomu uloženo, kolikáté v celkovém pořadí je první číslo na daném řádku. Pokud bychom tedy chtěli zjistit, kde v matici se nachází prvek *data[i]*, museli bychom najít, na jakém indexu pole *row_ptr* se nachází hodnota i (pořadí daného nenulového čísla v matici), a podle tohoto indexu bychom zjistili index řádku. Index sloupce jednoduše nalezneme v poli *col_ind*. Velikost polí *col_ind* a *data* je tedy stejná jako u COO formátu, velikost pole *row_ptr* závisí na počtu řádků matice.

Příklad uložení matice v CSR formátu
Uložení matice $\mathbb{A}$ ze sekce 1.2.2 ve formátu CSR při indexování od jedné by bylo následující:

$$\mathbb{A} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 1 \end{pmatrix}.$$

data	2	10	5	4	1	7	3	1
col_ind	1	2	4	1	3	4	3	5

row_ptr	1	2	4	6	7

1.2.4 CSC formát

CSC *(Compressed Sparse Column)* formát je analogický k CSR formátu. Pouze pole pro ukládání indexů sloupce a řádku jsou prohozené, tedy do pole *row_ind* ukládáme indexy řádků a do pole *col_ptr* ukládáme pořadí prvku v matici pro první prvek v daném sloupci. Aby toto mohlo fungovat, musí být prvky v poli *data* uloženy v pořadí shora dolů, zleva doprava. Při vybírání z CSR/CSC formátů hledíme na styl přístupu programu k prvkům matice. Pokud algoritmus přistupuje k matici po sloupcích, volíme formát CSC, pokud přistupuje po řádcích, volíme naopak formát CSR, neboť takovéto ukládání je efektivní z hlediska využití cache paměti.

Příklad uložení matice v CSC formátu
Uložení matice $\mathbb{A}$ ze sekce 1.2.2 ve formátu CSC při indexování od jedné by bylo následující:

$$\mathbb{A} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 1 \end{pmatrix}.$$

data	2	4	10	1	3	5	7	1
row_ind	1	3	2	3	5	2	4	5

col_ptr	1	3	4	6	8

1.2.5 Vnější formát uložení Matrix Market

MM *(Matrix Market)* je veřejně dostupný zdroj matic ukládaných v souborech ve stejnojmenném formátu. Existují dva typy tohoto formátu, a to *„array“*, který slouží pro uložení hustých matic, a *„coordinate“*, který slouží pro uložení matic řídkých. Zaměříme se na druhý typ tohoto formátu.

Soubor ve formátu MM má přesně danou strukturu. Na prvním řádku je vždy hlavička, která upřesňuje povahu celého souboru. Je zde obsaženo, o jaký objekt se jedná (většinou tedy matice - „matrix“, může jím být ale také vektor - „vector“), jestli je zvolen formát *„coordinate“* nebo *„array“*, jakého typu jsou prvky v objektu *(real, double, complex, integer)* a nakonec jaký je typ symetrie matice. Tento řádek vždy začíná řetězcem „%%MatrixMarket“.

Mohou, ale nemusí, následovat komentáře, které dále popisují matici. Tyto komentáře musí vždy začínat znakem „%“.

Následuje řádek, který specifikuje počet řádků, sloupců a nenulových prvků v matici. Vždy je ve tvaru m n *nonzeros*, kde m je počet řádků matice a n je počet sloupců v matici.

Nakonec soubor obsahuje samotná data. Při zvoleném formátu *„coordinate“* jsou data ve tvaru i j *value*, kde i je index řádku, j je index sloupce a *value* je samotná hodnota prvku matice. Pokud je navíc matice symetrická, je v datech obsažena pouze diagonála a prvky bezprostředně pod ní.
Pro názornost převedeme matici $\mathbb{A}$ ze sekce 1.2.3 do formátu MM:

$$\mathbb{A} = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 5 & 0 \\ 4 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 3 & 0 & 1 \end{pmatrix}$$

```
%%MatrixMarket matrix coordinate integer general
%Priklad matice 5x5 s 8 nenulovymi prvky v MM formatu
5 5 8
1 1 2
2 2 10
2 4 5
3 1 4
3 3 1
4 4 7
5 3 3
5 5 1
```

1.3 Doolittlův algoritmus pro nalezení LU rozkladu matice

Z definice LU rozkladu víme, že lze regulární čtvercovou matici rozložit na součin dolní trojúhelníkové a horní trojúhelníkové matice. Doolittlova metoda [8] tento LU rozklad řeší tak, že se navíc prvky na diagonále ve výsledné matici $\mathbb{L}$ rovnají jedné. Problém LU rozkladu pomocí Doolitlovy metody je nejlépe vidět na příkladu.

1.3.1 Průběh Doolittlova algoritmu

Algoritmus 1 Doolittlova metoda pro LU rozklad

```
 1: procedure Doolittle
 2:     for k ← 1 to n do
 3:         l[k][k] ← 1
 4:         for m ← k  to n do
 5:             u[k][m] ← a[k][m]
 6:             for j ← 1 to k − 1 do
 7:                 u[k][m] ← u[k][m] − (l[k][j] ∗ u[j][m])
 8:             end for
 9:         end for
10:         for i ← k + 1 to n do
11:             l[i][k] ← a[i][k]/u[k][k]
12:             for j ← 1 to k − 1 do
13:                 l[i][k] ← l[i][k] − (l[i][j] ∗ u[j][k]/u[k][k])
14:             end for
15:         end for
16:     end for
17: end procedure
```

Nechť matice $\mathbb{A}$ typu (3,3) je regulární, její LU rozklad bude vypadat následovně:

$$\mathbb{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{pmatrix} = \mathbb{L}\mathbb{U}$$

V tomto algoritmu [9] je velice podstatné pořadí vypočítávaných prvků, neboť na sobě samotné hodnoty závisí. Jak můžeme vidět v pseudokódu 1 na řádku 7, v prvním průchodu prvního vnitřního for cyklu bychom pro matici

$\mathbb{A}$ nejprve získali prvky $u_{1,1}$, $u_{1,2}$ a $u_{1,3}$:

$$u_{1,1} = a_{1,1} - \sum_{j=1}^{0}(l_{1,j} \cdot u_{j,1}) = a_{1,1},$$
$$u_{1,2} = a_{1,2} - \sum_{j=1}^{0}(l_{1,j} \cdot u_{j,2}) = a_{1,2},$$
$$u_{1,3} = a_{1,3} - \sum_{j=1}^{0}(l_{1,j} \cdot u_{j,3}) = a_{1,3}.$$

Druhý vnitřní for cyklus nám vypočítá hodnoty pro $l_{2,1}$ a $l_{3,1}$:

$$l_{2,1} = \frac{a_{2,1} - \sum_{j=1}^{0}(l_{2,j} \cdot u_{j,1})}{u_{1,1}} = \frac{a_{2,1}}{u_{1,1}},$$
$$l_{3,1} = \frac{a_{3,1} - \sum_{j=1}^{0}(l_{3,j} \cdot u_{j,1})}{u_{1,1}} = \frac{a_{3,1}}{u_{1,1}}.$$

V dalším průchodu bychom stejným způsobem získali hodnoty pro prvky $u_{2,2}$, $u_{2,3}$ a $l_{3,2}$:

$$u_{2,2} = a_{2,2} - \sum_{j=1}^{1}(l_{2,j} \cdot u_{j,2}) = a_{2,2} - (l_{2,1} \cdot u_{1,2}),$$
$$u_{2,3} = a_{2,3} - \sum_{j=1}^{1}(l_{2,j} \cdot u_{j,3}) = a_{2,3} - (l_{2,1} \cdot u_{1,3}),$$
$$l_{3,2} = \frac{a_{3,2} - \sum_{j=1}^{1}(l_{3,j} \cdot u_{j,2})}{u_{2,2}} = \frac{a_{3,2} - (l_{3,1} \cdot u_{1,2})}{u_{2,2}}.$$

A nakonec bychom dostali poslední chybějící prvek $u_{3,3}$:

$$u_{3,3} = a_{3,3} - \sum_{j=1}^{2}(l_{3,j} \cdot u_{j,3}) = a_{3,3} - (l_{3,1} \cdot u_{1,3} + l_{3,2} \cdot u_{2,3}).$$

V každém průchodu také samozřejmě nastavujeme každý prvek diagonály matice $\mathbb{L}$ na hodnotu 1.

1.3.2 Časová a paměťová složitost

Časová složitost Doolittlova algoritmu závisí na rozměrech vstupní matice. Z pseudokódu 1 na stránce 8 můžeme tuto složitost vyčíst jako

$$\mathcal{O}(\sum_{k=1}^{n}(\sum_{m=k}^{n}\sum_{j=1}^{k-1}1+\sum_{i=k+1}^{n}\sum_{j=1}^{k-1}1)).$$

Z toho vyvozujeme složitost $\mathcal{O}(n^3)$.

Co se týče paměťové složitosti, algoritmus pro výpočty prvků používá jak hodnoty vstupní matice $\mathbb{A}$, tak nově vypočítané hodnoty matic $\mathbb{L}$ a $\mathbb{U}$. Není tedy možné algoritmus naimplementovat in-place (s přidáním pouze konstantní paměti $\mathcal{O}(1)$). Pokud bychom zvolili hustý formát uložení, potřebovali bychom $\mathcal{O}(2 \cdot n^2)$ paměti navíc, kde n je rozměr matice. Toto bychom mohli optimalizovat na $\mathcal{O}(n^2)$ přidané paměti, kdybychom matice $\mathbb{L}$ i $\mathbb{U}$ ukládali do stejného pole, neboť se jedná o trojúhelníkové matice, kde navíc matice $\mathbb{L}$ má na diagonále pouze 1, které tedy nemusíme nikam ukládat. Nejlepší využití paměti (pokud pracujeme s řídkými maticemi) dosáhneme využitím některého z řídkých formátů uložení. Například při použití CRS/CCS formátu bychom přidali pouze $\mathcal{O}(2 \cdot (n + nnzL + nnzU))$ paměti, kde n je rozměr matice, $nnzL$ je počet nenulových prvků v matici $\mathbb{L}$ a $nnzU$ je počet nenulových prvků v matici $\mathbb{U}$. Musíme ale počítat s pomalejším vykonáním algoritmu kvůli složitějšímu přístupu k prvkům matice.

Kapitola **2**

Heuristiky pro snížení počtu nově vzniklých nenulových prvků

Protože je $\mathbb{LU}$ rozklad matice $\mathbb{A}$ založen na Gaussově eliminační metodě, během procesu mohou vzniknout nové nenulové prvky nazývané „fill-ins“. Vznik těchto nenulových prvků z původních nulových prvků matice $\mathbb{A}$ může zapříčinit i to, že nové matice $\mathbb{L}$ a $\mathbb{U}$ nebudou nadále řídké. Abychom předešli tomuto důsledku, můžeme využít široké škály heuristik, které se snaží minimalizovat tvorbu nových prvků v matici. Níže jsou popsány některé z nich.

2.1 Základy z teorie grafů

Protože některé heuristiky na redukci tvorby „fill-ins“ využívají grafy, v této části se seznámíme se základy z teorie grafů. Definice jsou čerpány z [10].

Neorientovaný graf Nechť H, U jsou libovolné disjunktní množiny a $\varrho : H \mapsto U \otimes U$ zobrazení. *Neorientovaným grafem* nazýváme uspořádanou trojici $G = \langle H, U, \varrho \rangle$, prvky množiny H jsou *hranami grafu* G, prvky množiny U *uzly grafu* G a zobrazení ϱ *incidencí grafu* G.

Incidence ϱ grafu je zobrazení z množiny všech hran do množiny všech neuspořádaných dvojic $[u, v]$, $u, v \in U$. Každé hraně grafu přiřazuje neuspořádanou dvojici uzlů, které tato hrana propojuje. Díky incidenci ϱ tedy můžeme zaznamenat i více různých hran spojujících stejné dva uzly.

Smyčka *Smyčka* v grafu je zvláštním případem mezi hranami, která vede do stejného uzlu, ze kterého vychází, tedy $\varrho(h) = [u, u]$.

Prostý graf *Prostý graf* je takový graf, který neobsahuje žádné *rovnoběžné hrany*, tj. žádné dva uzly nejsou spojeny více než jednou hranou. V opačném

případě se jedná o *multigraf.*

Obyčejný graf *Obyčejný graf* je takový prostý graf, který neobsahuje žádné smyčky.

Úplný graf *Úplným grafem* nazýváme obyčejný graf $K_u = \langle \binom{U}{2}, U \rangle$; pro n přirozené bude K_n označovat úplný graf o n uzlech.

Podgraf grafu Graf $G' = \langle H', U', \varrho' \rangle$ nazýváme *podgrafem* grafu $G = \langle H, U, \varrho \rangle$ (zapisujeme $G' \subseteq G$), jestliže platí

$$(H' \subseteq H) \quad \& \quad (U' \subseteq U) \quad \& \quad \forall h \in H'(\varrho'(h) = \varrho(h)).$$

Množina sousedů Nechť $G = \langle H, U, \varrho \rangle$ je graf, $u \in U$ libovolný uzel a $A \subseteq U$ libovolná podmnožina uzlů. *Množinou sousedů* $\Gamma(u)$ *uzlu* u nazýváme podmnožinu uzlů definovanou vztahem

$$\Gamma(u) = v \ \in U \ : \exists h \in H(\varrho(h)) = [u, v].$$

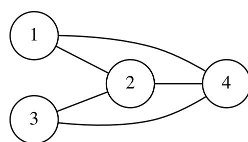
Množinu sousedů $\Gamma(A)$ *podmnožiny* A definujeme vztahem $\Gamma(A) = \cup_{v \in A} \Gamma(v)$.

Stupeň uzlu *Stupněm* $\delta_G(u)$ uzlu u v grafu G nazýváme počet hran s ním incidujících. Symboly $\delta(G)$ a $\Delta(G)$ označujeme minimální, resp. maximální stupeň uzlu v grafu G.

Klika grafu *Klikou grafu* $G = \langle H, U, \varrho \rangle$ nazýváme každý maximální úplný podgraf grafu G, tzn. $K \subseteq G, K \ \cong K_n$, pro který neexistuje podgraf K' takový, že $K \subset K' \subseteq G, K \ \cong K_m$ pro nějaké $m > n$.

Matice sousednosti Nechť $G = \langle H, U, \varrho \rangle$ je neorientovaný graf s množinou uzlů $U = \{u_1, u_2, \ldots, u_n\}$. *Matici sousednosti* grafu G nazýváme čtvercovou matici $V = [v_{ik}]$ řádu n, jejíž prvky jsou celá čísla daná vztahem

$$v_{ik} = |\varrho^{-1}([u_i, u_k])|$$



$$\mathbb{V} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Obrázek 2.1: Matice sousednosti neorientovaného grafu

2.2 Heuristiky pro symetrické matice

Níže popsané heuristiky využívají k přeskupení řádků a sloupců symetrii matice, jsou tedy určené pro symetrické matice. Pokud bychom ale chtěli tyto heuristiky použít i na nesymetrické matice, stačí vstupní matici upravit tak, že k ní přičteme její matici transponovanou, tedy $\mathbb{A}^T + \mathbb{A}$.

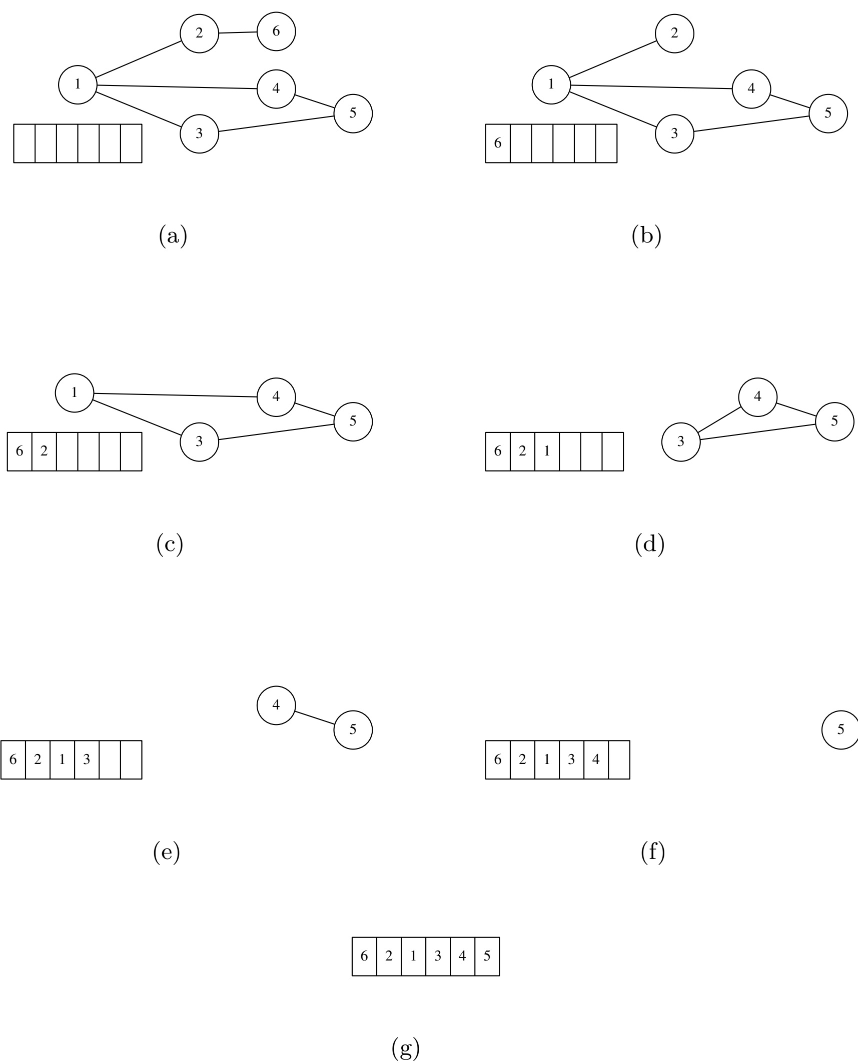
2.2.1 Minimum Degree Ordering

Heuristika Minimum Degree Ordering [11] minimalizuje tvorbu nenulových prvků symetrické matice. Vstupem MD algoritmu je graf matice sousednosti, z něhož se postupně odebírají uzly. Výstupem je permutační vektor, který obsahuje uzly v pořadí, ve kterém byly odebrány z grafu. Podle tohoto výsledného vektoru se změní pořadí nejprve řádků a potom sloupců upravované matice.

Myšlenkou MD algoritmu je odstraňovat uzly z grafu v pořadí podle velikosti stupně. Algoritmus 2 na stránce 15 pracuje s grafem matice sousednosti vstupní matice. Nejprve z grafu vybereme uzel s nejmenším stupněm, pojmenujme ho uzel X, a uložíme jej do výsledného pole (řádky 4 a 5). Projdeme všechny sousedy uzlu X a vytvoříme z nich kliku grafu (řádky 6 až 9), tedy každý soused uzlu X se stane sousedem se všemi jeho dalšími sousedy. Vytvoření kliky je v podstatě sjednocení množiny sousedů uzlu X s množinou sousedů každého uzlu, který s tímto uzlem X sousedí. Je ale potřeba při sjednocení vynechat uzel, který by pak odkazoval sám na sebe, a zároveň smazat z množiny sousedů uzel X. S každým nově přidaným nebo odstraněným sousedem se změní velikost stupně. Nakonec uzel vymažeme z grafu (řádek 15). Na obrázku 2.2 je znázorněn postup MD algoritmu pro matici

$$\mathbb{A} = \begin{pmatrix} 1 & 7 & 8 & 9 & 0 & 0 \\ 10 & 2 & 0 & 0 & 0 & 11 \\ 12 & 0 & 3 & 0 & 13 & 0 \\ 14 & 0 & 0 & 4 & 15 & 0 \\ 0 & 0 & 16 & 17 & 5 & 0 \\ 0 & 18 & 0 & 0 & 0 & 6 \end{pmatrix}.$$

(Smyčky jsou zde i dále v textu zanedbávány).



Obrázek 2.2: Průběh heuristiky Minimum Degree Ordering

Po změně pořadí podle výsledného permutačního vektoru $P = (6, 2, 1, 3, 4, 5)$ z obrázku 2.2 získáváme matici

$$\bar{\mathbb{A}} = \begin{pmatrix} 6 & 18 & 0 & 0 & 0 & 0 \\ 11 & 2 & 10 & 0 & 0 & 0 \\ 0 & 7 & 1 & 8 & 9 & 0 \\ 0 & 0 & 12 & 3 & 0 & 13 \\ 0 & 0 & 14 & 0 & 4 & 15 \\ 0 & 0 & 0 & 16 & 17 & 5 \end{pmatrix}.$$

Algoritmus 2 Minimum Degree

Vstup: Graf matice sousednosti G
Výstup: Permutační vektor P

```
 1: procedure MINIMUM DEGREE
 2:     index ← 1
 3:     while P.Size < G.Size do
 4:         node ← min(G)
 5:         P[index + +] ← node
 6:         for i ← 1 to node.Neighbors.Size do
 7:             union(node.Neighbors, node.Neighbors[i].Neighbors)
 8:             delete(node.Value, node.Neighbors[i].Neighbors)
 9:         end for
10:     end while
11:     return P
12: end procedure
```

2.2.2 Multiple Minimum Degree Ordering

Protože je v heuristice Minimum Degree 2.2.1 přepočítávání stupně uzlů (tedy i vytváření kliky grafu) velmi náročnou operací, Liu [12] navrhl vylepšení, které všechny uzly s nejnižším uzlem odstraní najednou. Pokud mají totiž dva uzly X a Y stejný minimální stupeň a uzel X je odstraněn, je jasné, že v další iteraci bude odstraňován uzel Y. Mohou nastat dva případy. Buď uzel Y sousedil s uzlem X, a jeho stupeň se o jedna sníží, v této iteraci bude mít jistě nejnižší stupeň. Nebo s uzlem X nesousedil, ale i tak bude mít minimální stupeň, protože i když se o jedna snížily stupně uzlů, které sousedily s uzlem X, v této iteraci jejich stupeň bude větší nebo roven stupni uzlu Y. Nikdy nemůže být menší, neboť před vykonáním předchozí iterace měly stupeň vyšší než uzel Y.

Algoritmus si nejprve uloží všechny uzly, které mají minimální stupeň. Následně označí všechny sousedy těchto uzlů, které poté prochází a hledá v jejich polích sousedů některý z množiny mazaných uzlů. Odkaz na tyto uzly vymaže, a vytvoří kliky ze všech jejich sousedů. Algoritmus končí až tehdy, když jsou z grafu všechny uzly smazané.

V případě matice $\mathbb{A}$ ze sekce 2.2.1 by byl výsledný permutační vektor stejný jako pro heuristiku Minimum Degree, protože v bodě (c) na obrázku 2.2 bychom odstranili všechny uzly najednou, neboť mají všechny stejně velký stupeň. Protože ale není definována jiná priorita výběru uzlu, seřadíme je vzestupně podle hodnoty uzlu.

2.2.3 Cuthill-McKee

Algoritmus Cuthill-McKee [13] byl vytvořen E. Cuthillem a J. McKeem v roce 1969. Stejně jako v 2.2.1 a 2.2.2, výstupem heuristiky je permutační vektor,

podle kterého se nejprve změní pořadí řádků a následně pořadí sloupců vstupní matice.

Nejprve z grafu vybereme uzel s nejmenším stupněm a uložíme jej do výsledného pole (řádky 5 a 6 v pseudokódu 3 na stránce 16). Jeho množinu sousedů vložíme do fronty v pořadí vzestupně podle velikosti jejich stupně (řádek 7). Poté vybereme první uzel z fronty, vložíme ho na první volnou pozici ve výsledném poli a všechny jeho sousední uzly opět vložíme do fronty (řádky 9 - 11). Tento krok opakujeme, dokud není fronta prázdná. Pokud je fronta prázdná, ale nemáme ještě všechny uzly uložené ve výsledném poli, opakujeme celý postup od začátku.

Algoritmus 3 Cuthill-McKee

Vstup: Graf matice sousednosti G
Výstup: Permutační vektor P

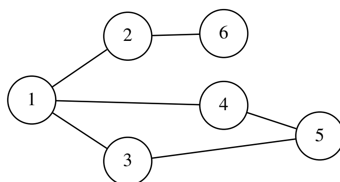
```
 1: procedure Cuthill-McKee
 2:     i ← 0
 3:     init(queue)
 4:     while P.Size < G.Size do
 5:         node ← min(G)
 6:         P[i++] ← node
 7:         queue.push(node.Children)
 8:         while not queue.isEmpty() do
 9:             node ← queue.pop()
10:             P[i++] ← node
11:             queue.push(node.Children)
12:         end while
13:     end while
14:     return P
15: end procedure
```

Dále si popíšeme, jaký by byl postup algoritmu pro matici $\mathbb{A}$ ze sekce 2.2.1, jejíž graf je znázorněn na obrázku 2.3.

$$\mathbb{A} = \begin{pmatrix} 1 & 7 & 8 & 9 & 0 & 0 \\ 10 & 2 & 0 & 0 & 0 & 11 \\ 12 & 0 & 3 & 0 & 13 & 0 \\ 14 & 0 & 0 & 4 & 15 & 0 \\ 0 & 0 & 16 & 17 & 5 & 0 \\ 0 & 18 & 0 & 0 & 0 & 6 \end{pmatrix}$$

Nejprve musíme nalézt uzel s nejnižším stupněm, kterým je v našem grafu uzel s číslem 6. Do pole P tedy uložíme uzel 6 a do fronty *queue* jeho množinu sousedů, tedy: $P = (6)$ a $queue = (2)$.

Protože máme ve frotně *queue* pouze jeden prvek, automaticky ho vybereme a vložíme do P. Tímto nám vzniká $P = (6, 2)$ a $queue = (1)$.



Obrázek 2.3: Graf vstupní matice $\mathbb{A}$

Stejně tak pokračujeme s uzlem 1, získáváme $P = (6, 2, 1)$ a *queue* = (3, 4). Protože má uzel 1 oba sousední uzly se stejným stupněm, v tomto kroku se může algoritmus rozcházet a dávat různé výsledky, protože není definováno, který z těchto dvou uzlů má přednost.

Nakonec postupně přidáme uzly 3, 4 a 5 v tomto pořadí. Výsledný permutační vektor tedy bude $P = (6, 2, 1, 3, 4, 5)$ a po prohození řádků a sloupců získáváme matici:

$$\bar{\mathbb{A}} = \begin{pmatrix} 6 & 18 & 0 & 0 & 0 & 0 \\ 11 & 2 & 10 & 0 & 0 & 0 \\ 0 & 7 & 1 & 8 & 9 & 0 \\ 0 & 0 & 12 & 3 & 0 & 13 \\ 0 & 0 & 14 & 0 & 4 & 15 \\ 0 & 0 & 0 & 16 & 17 & 5 \end{pmatrix}.$$

2.2.4 Reverse Cuthill-McKee

Heuristika Reverse Cuthill-McKee je upravená verze algoritmu Cuthill-McKee, která byla uvedena Alanem Georgem. Jediným rozdílem oproti Cuthill-McKee je, že pořadí prvků v permutačním vektoru se na konci algoritmu obrátí. Výsledný permutační vektor matice $\mathbb{A}$ ze sekce 2.2.3 a její výstupní matice $\bar{\mathbb{A}}$ vypadají následovně: $P = (5, 4, 3, 1, 2, 6)$ a

$$\bar{\mathbb{A}} = \begin{pmatrix} 5 & 17 & 16 & 0 & 0 & 0 \\ 15 & 4 & 0 & 14 & 0 & 0 \\ 13 & 0 & 3 & 12 & 0 & 0 \\ 0 & 9 & 8 & 1 & 7 & 0 \\ 0 & 0 & 0 & 10 & 2 & 11 \\ 0 & 0 & 0 & 0 & 18 & 6 \end{pmatrix}.$$

2.3 Heuristiky pro nesymetrické matice

Níže popsané heuristiky jsou primárně určené pro nesymetrické matice, ale dají se samozřejmě použít i na matice symetrické.

2.3.1 Markowitzova strategie

Markowitzova strategie [14] řeší záměnu řádků v matici pro snížení vzniku „fill-ins“ pomocí pivotizace, je tedy vhodná i pro nesymetrické matice. Algoritmus funguje tak, že v každé iteraci vybere pravou dolní část matice typu $(n-i, n-i)$, kde $i \in \langle 0, n-2 \rangle$ je počet dosud proběhlých iterací, a pro všechna nenulová čísla vypočítá Markowitzovu cenu, která je definována následovně:

$$M_{ijk} = (r(j,i) - 1) \cdot (c(k,i) - 1),$$

kde $r(j,i)$ je celkový počet nenulových čísel v j-tém řádku ve výřezu matice typu $(n-i, n-i)$ a $c(k,i)$ je celkový počet nenulových čísel v k-tém sloupci ve výřezu matice typu $(n-i, n-i)$. Řádek, ve kterém se nachází číslo s nejnižší Markowitzovou cenou, se vymění s prvním řádkem, který je ještě obsažen v aktuálním výřezu matice. Stejná operace se provede se sloupci, tedy zjistíme, ve kterém sloupci se toto číslo s nejnižší Markowitzovou cenou nachází, a celý sloupec prohodíme s prvním sloupcem aktuálního výřezu.

V pseudokódu 4 na stránce 19 je Markowitzova strategie popsána. Funkce `countNNZ` na řádku 4 vypočítá pro každý řádek, resp. sloupec, počet nenulových prvků v daném řádku, resp. sloupci, a zaznamená výsledek do polí *rows*, resp. *cols*. Na řádku 5 je nastavena proměnná *end* na rozměr vstupní matice - 2, tedy počet iterací, které mají proběhnout. Na řádku 9 začíná hlavní cyklus, který postupně ořezává matici, a pro daný výřez vypočítává pro každý nenulový prvek Markowitzovu cenu. Na řádcích 25 až 30 se vymění řádky a sloupce, pokud byla minimální Markowitzova cena nalezena v jiném než aktuálním prvním řádku nebo sloupci. Na řádcích 31 - 32 proběhne aktualizace proměnných a na řádku 34 se zavolá funkce `updateNNZ`, která zaktualizuje pole *cols* a *rows* podle nového výřezu a nové podoby matice.

Stejně jako v předchozích sekcích 2.2.1 a 2.2.3 si ukážeme průběh Markowitzovy strategie na matici

$$\mathbb{A} = \begin{pmatrix} 1 & 7 & 8 & 9 & 0 & 0 \\ 10 & 2 & 0 & 0 & 0 & 11 \\ 12 & 0 & 3 & 0 & 13 & 0 \\ 14 & 0 & 0 & 4 & 15 & 0 \\ 0 & 0 & 16 & 17 & 5 & 0 \\ 0 & 18 & 0 & 0 & 0 & 6 \end{pmatrix}.$$

Nejprve vypočítáme Markowitzovu cenu pro všechna nenulová čísla v celé matici. Například pro číslo $a_{11} = 1$ to tedy bude $M_{011} = (4-1) \cdot (4-1) = 9$.

Algoritmus 4 Markowitzova strategie

Vstup: Matice A

1: **procedure** MARKOWITZ STRATEGY
2: array $cols$
3: array $rows$
4: `countNNZ`$(cols, rows)$
5: $end \leftarrow A$.Size -2
6: $actual \leftarrow 1$
7: $minRow \leftarrow 1$
8: $minCol \leftarrow 1$
9: **for** $i \leftarrow 1$ to end **do**
10: $index \leftarrow 1$
11: $minCost \leftarrow (A.\text{Size}*A.\text{Size})$
12: **for** $j \leftarrow i$to$A$.Size **do**
13: **for** $k \leftarrow i$to$A$.Size **do**
14: **if** $A[j][k] \neq 0$ **then**
15: $cost \leftarrow (cols[index] - 1) * (row[index] - 1)$
16: $index \leftarrow index + 1$
17: **end if**
18: **if** $minCost > cost$ **then**
19: $minCost \leftarrow cost$
20: $minRow \leftarrow j$
21: $minCol \leftarrow k$
22: **end if**
23: **end for**
24: **end for**
25: **if** $actual \neq minRow$ **then**
26: `swapRows`$(minRow, actual)$
27: **end if**
28: **if** $actual \neq minCol$ **then**
29: `swapCols`$(minCol, actual)$
30: **end if**
31: $actual \leftarrow acutal + 1$
32: $minRow \leftarrow actual$
33: $minCol \leftarrow actual$
34: `updateNNZ`$(cols, rows)$
35: **end for**
36: **end procedure**

Celá matice s vypočítanou cenou bude vypadat:

$$\mathbb{M}_0 = \begin{pmatrix} 9 & 6 & 6 & 6 & & \\ 6 & 4 & & & & 2 \\ 6 & & 4 & & 4 & \\ 6 & & & 4 & 4 & \\ & & 4 & 4 & 4 & \\ & 2 & & & & 1 \end{pmatrix}.$$

Z toho vyplývá, že se poslední řádek vymění s prvním, protože v posledním řádku je číslo, které má nejnižší Markowitzovu cenu. Po prohození řádků se ze stejného důvodu také vymění poslední s prvním sloupcem. Po těchto krocích bude matice vypadat následovně:

$$\mathbb{A} = \begin{array}{c} \\ (6) \\ (2) \\ (3) \\ (4) \\ (5) \\ (1) \end{array} \begin{array}{c} \begin{array}{cccccc} (6) & (2) & (3) & (4) & (5) & (1) \end{array} \\ \begin{pmatrix} 6 & 18 & 0 & 0 & 0 & 0 \\ 11 & 2 & 0 & 0 & 0 & 10 \\ 0 & 0 & 3 & 0 & 13 & 12 \\ 0 & 0 & 0 & 4 & 15 & 14 \\ 0 & 0 & 16 & 17 & 5 & 0 \\ 0 & 7 & 8 & 9 & 0 & 1 \end{pmatrix} \end{array}.$$

Dále vypočítáme Markowitzovu cenu pro všechna nenulová čísla, která se nacházejí v dolním čtverci o velikosti 5×5 nově vytvořené matice $\mathbb{A}$:

$$\mathbb{M}_1 = \begin{pmatrix} 1 & & & & 3 \\ & 4 & & 4 & 6 \\ & & 4 & 4 & 6 \\ & 4 & 4 & 4 & \\ 3 & 6 & 6 & & 9 \end{pmatrix}.$$

Protože je nyní na prvním řádku a v prvním sloupci matice $\mathbb{M}_1$ nejnižší Markowitzova cena, druhý řádek a druhý sloupec v matici $\mathbb{A}$ zůstane na svém místě. Nyní bychom pomocí matic s vypočtenými cenami zjistili, který řádek, resp. sloupec, bude na třetím až šestém řádku, resp. sloupci, a vyšla by nám výsledná matice:

$$\mathbb{A} = \begin{array}{c} \\ (6) \\ (2) \\ (3) \\ (5) \\ (4) \\ (1) \end{array} \begin{array}{c} \begin{array}{cccccc} (6) & (2) & (3) & (5) & (4) & (1) \end{array} \\ \begin{pmatrix} 6 & 18 & 0 & 0 & 0 & 0 \\ 11 & 2 & 0 & 0 & 0 & 10 \\ 0 & 0 & 3 & 13 & 0 & 12 \\ 0 & 0 & 16 & 5 & 17 & 0 \\ 0 & 0 & 0 & 15 & 4 & 14 \\ 0 & 7 & 8 & 0 & 9 & 1 \end{pmatrix} \end{array}.$$

$$\mathbb{A} = \begin{pmatrix} \bullet & & & & & \\ \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & & & \\ \bullet & \bullet & \bullet & \bullet & & \\ & & \bullet & \bullet & \bullet & \\ & \bullet & \bullet & \bullet & \bullet & \bullet \end{pmatrix} \cdot \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & & \\ & \bullet & \bullet & \bullet & & \bullet \\ & & \bullet & \bullet & \bullet & \bullet \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{pmatrix}$$

Obrázek 2.4: Doolittle bez využití heuristik

2.4 Přínos heuristik

Heuristiky mají snižovat tvorbu nových nenulových prvků, podíváme se tedy, jaké výsledky budeme mít pro matici $\mathbb{A}$.

$$\mathbb{A} = \begin{pmatrix} \bullet & & & & & \\ \bullet & \bullet & & & & \\ & \bullet & \bullet & & & \\ & & \bullet & \bullet & & \\ & & \bullet & \bullet & \bullet & \\ & & & \bullet & \bullet & \bullet \end{pmatrix} \cdot \begin{pmatrix} \bullet & \bullet & & & & \\ & \bullet & \bullet & & & \\ & & \bullet & \bullet & \bullet & \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{pmatrix}$$

(a) Multiple Minimum Degree

$$\mathbb{A} = \begin{pmatrix} \bullet & & & & & \\ \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & & & \\ & \bullet & \bullet & \bullet & & \\ & & & \bullet & \bullet & \\ & & & & \bullet & \bullet \end{pmatrix} \cdot \begin{pmatrix} \bullet & \bullet & \bullet & & & \\ & \bullet & \bullet & \bullet & & \\ & & \bullet & \bullet & & \\ & & & \bullet & \bullet & \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{pmatrix}$$

(b) Reverse Cuthill-McKee

$$\mathbb{A} = \begin{pmatrix} \bullet & & & & & \\ \bullet & \bullet & & & & \\ & & \bullet & & & \\ & & \bullet & \bullet & & \\ & & & \bullet & \bullet & \\ & \bullet & \bullet & \bullet & \bullet & \bullet \end{pmatrix} \cdot \begin{pmatrix} \bullet & \bullet & & & & \\ & \bullet & & & & \bullet \\ & & \bullet & \bullet & & \bullet \\ & & & \bullet & \bullet & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \bullet \end{pmatrix}$$

(c) Markowitzova strategie

Obrázek 2.5: Doolittle s využitím heuristik

Pokud bychom žádnou heuristiku nevybrali, výsledné matice $\mathbb{L}$ i $\mathbb{U}$ mají obě 18 nenulových prvků, jak můžeme vidět na obrázku 2.4. Nový prvek oproti

vstupní matici vznikne například na pozici $(3, 2)$ v matici $\mathbb{L}$, neboť

$$l_{3,2} = \frac{a_{3,2} - l_{2,1} \cdot u_{1,2}}{u_{2,2}} \neq 0.$$

Pokud využijeme k výpočtu heuristiky, na obrázku 2.5 je vidět, že heuristiky MMD, RCM i Markowitzova strategie mají všechny ve výsledných maticích $\mathbb{L}$ a $\mathbb{U}$ 13 prvků namísto 18. I na tak malé matici o velikosti 6×6 je rozdíl 5 čísel. V kapitole 5 porovnáme heuristiky mezi sebou na větších maticích.

Kapitola 3

Existující řešení

V této kapitole jsou popsána různá řešení podobná tématu této práce. Jsou zde uvedeny jiné bakalářské práce zabývající se podobným problémem a existující nástroje pro LU rozklad.

3.1 Akademické práce

Charakteristikou nejpodobnější z prací je stejnojmenná práce kolegy Kusého [15], který porovnává mezi sebou jednotlivé metody LU rozkladu. Naimplementoval Croutovu, Choleskyho a QR metodu, které následně mezi sebou porovnal z hlediska časové a paměťové složitosti. Dále také v práci měřil paměťovou náročnost v závislosti na využitém formátu uložení matic. Ve všech případech bylo nejvýhodnější použít CRS/CCS formát, proto i v této práci bylo v implementaci využito toto uložení pro matice. Na druhou stranu je ale z měření vidět, že nejrychlejší je použití hustého formátu uložení, CRS/CCS formát pro matici o velikosti 2500 potřebuje pro provedení víc než dvojnásobek času, kolik potřebuje hustý formát.

Další bakalářská práce [16] měla za cíl naimplementovat řešič řídkých soustav lineárních rovnic pomocí LU rozkladu. Výsledný program obsahoval grafické uživatelské rozhraní, kde si uživatel mohl vybrat, která z metod (LU - Doolittlův algoritmus, Cholesky a GEM) bude na vyřešení matice použita a která z heuristik (AMD, COLAMD, RCM nebo žádná) rozklad ulehčí. Výsledná aplikace je následně porovnávána s existujícími řešeními.

Práce s názvem Řešiče rozsáhlých soustav lineárních rovnic [17] porovnává mezi sebou metody přímé - LU rozklad a GEM, a metody iterační - metoda konjugovaných gradientů a metoda bikonjugovaných gradientů. Výsledkem měření bylo, že pro menší matice jsou vhodnější iterační metody a pro rozsáhlejší matice naopak metody přímé.

3.2 Software

Existuje mnoho aplikací a knihoven, které pro různé operace využívají LU rozklad. Některé z nich jsou popsány níže.

3.2.1 R

„R“ [18] je volně dostupný software pro provádění statistických výpočtů a z nich vytváření grafů. LU rozklad je zde použit pro funkci `det`, která vypočítá determinant matice a funkci `solve`, která vyřeší danou lineární soustavu rovnic. Funkce `solve` vedle LU rozkladu využívá Choleskyho metodu a QR metodu podle předchozího nastavení. Dále je v R k dispozici funkce `sparseLU`, která vypočítá LU rozklad pro řídkou čtvercovou matici.

3.2.2 SciPy

„SciPy“ [19] je kolekce matematických algoritmů v rozšíření „NumPy“ pro jazyk Python. Tato kolekce obsahuje mnoho rozkladů matice, mezi nimi například Cholesky, QR nebo Schur rozklad. LU rozklad získáme zadáním příkazu `linalg.lu`.

3.2.3 MATLAB

„MATLAB“ [20] je nástroj pro práci s maticemi a jejich vizualizaci. „MATLAB“ využívá LU rozklad pro výpočet determinantu (funkce `det`) a inverze (funkce `inv`). Dále je možné přímo získat LU rozklad pomocí funkce `lu`.

3.2.4 SuperLU

„SuperLU“ [21] je software pro vyřešení lineární soustavy rovnic, který pro výpočet využívá LU rozklad. Pro snížení tvorby nových nenulových prvků využívá heuristiky MMD a COLAMD, umožňuje ale uživateli vložit vlastní permutační matici.

Kapitola **4**

Implementace

Tato kapitola popisuje samotnou realizaci Doolittlova algoritmu a vybraných heuristik (Multiple Minimum Degree, Reverse Cuthill-McKee a Markowitzovy strategie) v jazyce C++. Nejprve jsou uvedeny použité prostředky, následuje zvolené uložení dat v paměti a popis tříd a jejich funkcionalit. Nakonec jsou uvedené jednotlivé implementace algoritmů.

4.1 Použité prostředky

Jak už bylo výše uvedeno, program je napsán v jazyce C++. Kromě standardních knihoven byly využity datové struktury z STL, přesněji kontejner *queue* typu integer pro vkládání sousedů vybraného uzlu do fronty v heuristice RCM a kontejner *set* pro reprezentaci sousedů v grafu u heuristiky MMD.

Z fronty jsou využity operace `empty, front, pop` a `push`. První tři mají časovou složitost konstantní $\mathcal{O}(1)$, operace push má amortizovanou časovou složitost také $\mathcal{O}(1)$, ale pokud je nutná realokace, je časová složitost $\mathcal{O}(n)$.

Program u kontejneru *set* využívá operace `insert, erase, find, begin, end` a `size`. První tři operace mají časovou složitost logaritmickou $\mathcal{O}(\log n)$, ty ostatní mají časovou složitost konstantní $\mathcal{O}(1)$.

Další zajímavou knihovnou použitou v programu je knihovna *limits*, díky které se dá přesně určit, jestli daná hodnota typu *double* je nulová nebo ne.

Nakonec pro měření času potřebného k vykonání heuristik a Doolitlova algoritmu byla využita knihovna *OpenMP*.

4.2 Uložení dat v paměti

Software přijímá data ve formátu *MatrixMarket*, který byl popsán v kapitole 1.2.5. Podporuje dva typy tohoto formátu, a to pro matice symetrické, kde jsou v souboru zapsány pouze nenulové prvky na a pod diagonálou, a pro ostatní matice, kde jsou zapsány všechny nenulové prvky.

Protože program pracuje primárně s řídkými maticemi, byl zvolen vnitřní formát uložení pro řídké matice - CRS/CCS. Vstupní matice $\mathbb{A}$ je uložena ve formátu CRS, ale výběr formátu pro tuto matici není tak podstatný. Doolittlova metoda k ní přistupuje jak po řádcích (při vypočítávání hodnot pro matici $\mathbb{U}$), tak po sloupcích (při vypočítávání hodnot pro matici $\mathbb{L}$) a v heuristikách na pořadí procházení prvků nezáleží. Mnohem důležitější je výběr formátu pro výsledné matice $\mathbb{L}$ a $\mathbb{U}$. Doolittlova metoda při výpočtu prvků přistupuje do paměti matice $\mathbb{L}$ po řádcích, vyplatí se tedy využít formát CRS. U matice $\mathbb{U}$ je tomu přesně naopak, použijeme tedy formát CCS.

Vystupní matice $\mathbb{L}$ a $\mathbb{U}$ jsou uloženy do souboru opět ve formátu MatrixMarket.

4.3 Třídní rozdělení

Program je rozdělen do čtyř tříd, které jsou v této části popsány.

CInputMatrix Ve třídě *CInputMatrix* jsou naimplementovány veškeré operace, které se provádějí se vstupní maticí. V konstruktoru se načtou data ze souboru podle toho, jestli je matice symetrická nebo ne. Třída obsahuje implementace všech heuristik a Doolittlovy metody.

COutputMatrix Ve třídě *COutputMatrix* jsou naimplementovány veškeré operace, které se provádějí s výstupní maticí. Jedná se především o hledání daného prvku v matici a ukládání hodnot získaných z Doolittlovy metody. Nakonec je napsána metoda pro uložení výsledných matic do souboru ve formátu *MatrixMarket*.

CMatrix Protože vstupní i výstupní matice mají společné jak členské proměnné, tak některé metody, jsou tyto členské proměnné a metody definovány ve třídě *CMatrix*. *CInputMatrix* a *COutputMatrix* z této třídy dědí.

CGraph Objekt typu *CGraph* se vytváří pouze v heuristikách MMD a RCM. V konstruktoru je vytvořen graf matice sousednosti, dále jsou zde definovány metody pro úpravu grafu, jako je například vymazání uzlu z grafu v heuristice MMD. V této třídě je navíc definována struktura *TNode*, která představuje jednotlivé uzly grafu a obsahuje hodnotu a stupeň uzlu, příznaky typu *bool* a pole sousedů, které obsahuje hodnoty sousedících uzlů.

4.4 Doolittlův algoritmus

Doolittlův algoritmus byl naimplementován podle pseudokódu 1 na stránce 8. Jediný rozdíl je ve formátu uložení dat, neboť v programu jsou matice $\mathbb{A}$, $\mathbb{L}$ a $\mathbb{U}$ uloženy v CRS/CCS formátu a ne ve dvourozměrném poli. Kvůli tomu musely

být přidány metody, které dokáží najít nebo uložit správný prvek v těchto maticích. Jsou to například metody *getElem*, která vrátí prvek na pozici (i, j), nebo *saveValue*, která hodnotu uloží na správnou pozici.

Při ukládání hodnot do CRS/CCS formátu nastává problém v tom, když první sloupec (v případě CRS formátu), resp. první řádek (v případě CCS formátu), obsahuje nuly. Protože pracujeme s řídkými maticemi, tato situace nastane téměř vždy. Při postupném vkládání prvků do polí totiž nulové hodnoty přeskočíme a CRS/CCS formát takové řádky, resp. sloupce, nezaznamená a další hodnoty může uložit na špatné místo. Pokud je například ve výsledné matici $\mathbb{L}$ nenulový prvek na prvním řádku a další až na třetím řádku, po uložení do polí v CRS formátu bude mít matice druhý z prvků uložený na druhém řádku místo na třetím. Podle toho budou i další prvky v matici špatně uložené. V implementaci jsou proto do prvního řádku, resp. sloupce, ukládány i nulové hodnoty, abychom této chybě předešli. Částečně se tím zvýší paměťová i časová složitost, ale je to nejjednodušší způsob, jak se této chybě vyhnout.

4.5 Implementace heuristik

4.5.1 Multiple Minimum Degree Ordering

Heuristika MMD byla naimplementována podle kapitol 2.2.1 a 2.2.2. Nejprve je hledán minimální stupeň uzlu, potom jsou všechny uzly s tímto stupněm uloženy do pomocného pole a jejich sousedi jsou označeny příznakem. Nakonec jsou všechny uzly obsažené v pomocném poli odstraněny z grafu i z polí sousedů a pomocí vkládání do množiny *set* je vytvořena klika grafu všech sousedů. Příznak *m_Inserted* zajišťuje, že není žádný uzel do výsledného permutačního vektoru uložen dvakrát.

4.5.2 Reverse Cuthill-McKee

Heuristika RCM je naimplementována podle pseudokódu 3 na stránce 16. Vkládání sousedů aktuálně zpracovávaného uzlu do fronty podle velikosti stupně je řešené pomocí binární haldy. Po vytvoření haldy se jednoduše pomocí metody `extractMin` získá sousední uzel s nejnižším stupněm a vloží se do fronty. To probíhá, dokud není halda prázdná.

4.5.3 Markowitzova strategie

Markowitzova strategie byla naimplementována podle pseudokódu 4 na stránce 19. Nejsložitější operací v tomto algoritmu je vyměňování řádků a sloupců. Sloupce se prohazují ještě relativně jednoduše, stačí v poli indexů sloupců vyměnit čísla a následně pro každý řádek seřadit vzestupně tyto hodnoty. Komplikace nastávají s vyměňováním řádků. Nikdy nemáme zajištěno, že bude počet nenulových prvků v prohazovaných řádcích stejný. Proto byla defino-

vána metoda `swapToSize`, která nejprve vymění pouze tolik prvků, které má řádek s menší počtem nenulových prvků. Podle toho, který z řádků má menší počet prvků, se potom pomocí posouvání polí vkládají zbylé prvky na správné místo.

Kapitola **5**

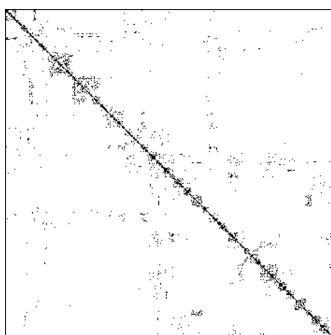
Testování

Kapitola testování popisuje průběh měření heuristik z hlediska časové a paměťové složitosti. Nejprve jsou popsána testovací data a výpočetní hardware. Poté jsou uvedeny výsledky testování nejprve na vygenerovaných datech a následně na datech získaných z repozitáře „Matrix Market“.

5.1 Testovací data

Program byl testován na datech z repozitáře „Matrix Market“. Byly zde vybrány matice ve velikosti v rozmezí od $(600, 600)$ do $(2000, 2000)$ z kolekcí „Harwell-Boeing Collection“ a „NEP Collection“. Protože se ale rychlost algoritmů i počet nenulových prvků výsledných matic odvíjí od více faktorů, byla implementace pro názornost otestována i na náhodně vygenerovaných datech, nejprve s narůstající hustotou se stejnými rozměry a následně se stálou hustotou a navyšujícími rozměry vstupní matice.
Na testování byly použity tyto symetrické matice z repozitáře MM:

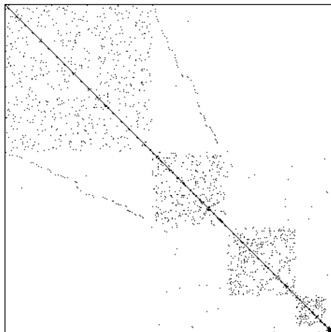
1138 BUS: Power system networks



Obrázek 5.1: Vizualizace 1138 BUS

- **disciplína**: Systémová síť
- **velikost**: 1138×1138
- **NNZ**: 4054
- **hustota zaplnění**: 0,3130 %

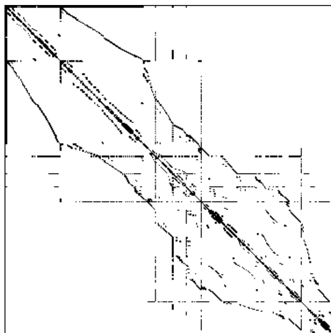
662 BUS: Power system networks



Obrázek 5.2: Vizualizace 662 BUS

- **disciplína**: Systémová síť
- **velikost**: 662 × 662
- **NNZ**: 2474
- **hustota zaplnění**: 0,5645 %

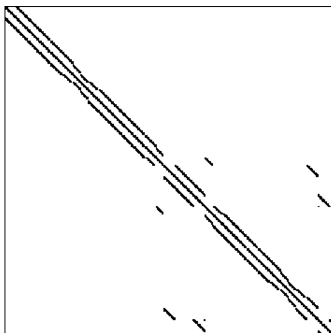
BCSSTK08: TV studio



Obrázek 5.3: Vizualizace BCSSTK08

- **disciplína**: Stavební inženýrství
- **velikost**: 1074 × 1074
- **NNZ**: 12960
- **hustota zaplnění**: 1,1236 %

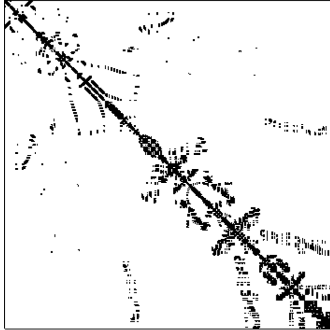
BCSSTK11: Ore car – lumped mass



Obrázek 5.4: Vizualizace BCSSTK11

- **disciplína**: Stavební inženýrství
- **velikost**: 1473 × 1473
- **NNZ**: 34241
- **hustota zaplnění**: 1,5781 %

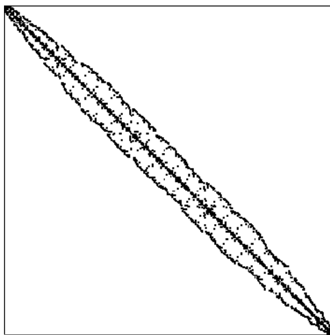
BCSSTK13: Fluid flow generalized eigenvalues



- **disciplína**: Stavební inženýrství
- **velikost**: 2003 × 2003
- **NNZ**: 83883
- **hustota zaplnění**: 2,0908 %

Obrázek 5.5: Vizualizace BCSSTK13

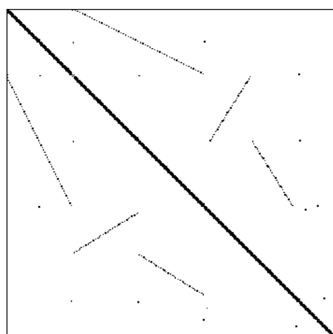
BCSSTK14: Roof of the Omni Coliseum, Atlanta



- **disciplína**: Stavební inženýrství
- **velikost**: 1806 × 1806
- **NNZ**: 63454
- **hustota zaplnění**: 1,9455 %

Obrázek 5.6: Vizualizace BCSSTK14

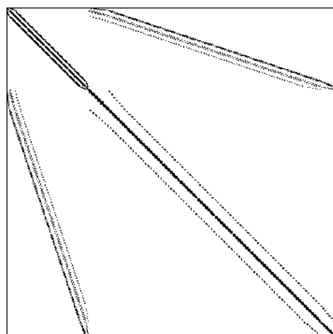
BCSSTK19: Part of a suspension bridge



- **disciplína**: Stavební inženýrství
- **velikost**: 817 × 817
- **NNZ**: 6853
- **hustota zaplnění**: 1,0267 %

Obrázek 5.7: Vizualizace BCSSTK19

BFW782B: Bounded Finline Dielectric Waveguide



- **disciplína**: Elektroinženýrství
- **velikost**: 782 × 782
- **NNZ**: 5982
- **hustota zaplnění**: 0,9782 %

Obrázek 5.8: Vizualizace BFW782B

5.2 Výpočetní hardware

Program byl otestován na serveru star.fit.cvut.cz, kde je pomocí plánovače zajištěno, že je procesor využit pouze na danou úlohu a výsledný čas tak není ovlivněn žádnými dalšími procesy. Specifikace serveru jsou následující:

- **CPU:** 2x Intel Xeon E5-2620 v2 (2.1 GHz, 15 MB L3 cache, 6 jader, 12 vláken),
- **RAM:** 32 GB.

5.3 Nastavení kompilátoru

Pro kompilaci programu bylo použito následující nastavení kompilátoru:

```
g++ -Ofast -fopenmp -mavx
```

`Ofast` Přepínač *Ofast* patří mezi skupinu přepínačů, které spouští při kompilaci optimalizace zlepšující výkonnost programu. Je to společné zapnutí přepínačů *O3*, *ffast-math*, *fno-protect-parens* a *fstack-arrays*. Úplná definice přepínače a jeho optimalizací je dostupná v [22].

`fopenmp` Přepínač *fopenmp* aktivuje rozšíření OpenMP, které se nejčastěji používá pro paralelizaci programu. V této práci bylo rozšíření použito na měření doby běhu programu pomocí funkce `omp_get_wtime`.

`mavx` Přepínač *mavx* využije místo SSEx instrukcí AVX instrukce. Toto nastavení je vhodné pro server star.fit.cvut.cz, jelikož využívá vektorovou instrukční sadu AVX.

5.4 Měření na vygenerovaných datech

Rychlost algoritmu závisí na rozměrech vstupní matice, počtu jejích nenulových prvků a nakonec i počtu nenulových prvků výsledných matic. Pro porovnání heuristik tedy byla vygenerována data, která mají proměnnou pouze jednu z prvních dvou hodnot.

První graf 5.9 ukazuje rychlost heuristik na datech, kde je rozměr matice nastaven na 1000 a hustota zaplnění nenulovými prvky se pohybuje mezi 1 až 10 %. Druhý graf 5.10 porovnává rychlost heuristik v závislosti na rozměru vstupní matice a hustota se pohybuje okolo 2 % (v intervalu $\langle 2,077, 2,196 \rangle$.

Z obou grafů je vidět, že nejrychlejší je heuristika MMD a nejpomalejší Markowitzova strategie. V grafu 5.9 s neměnným rozměrem matice s počtem nenulových prvků 50 852 (tedy 10 % zaplnění) je rozdíl mezi MMD a MS o víc než 30 sekund, heuristika MMD trvá 263,339 sekund, RCM 272,32 sekund a MS 293,962 sekund. V druhém grafu 5.10 je rozdíl skoro 300 sekund u rozměru matice 1400, heuristika MMD běží 674,546 sekund, RCM 858.335 a MS 933,158 sekund.

Stejně tak jako časová složitost, i paměťová složitost závisí na více parametrech. Pro každou heuristiku jsou to nejprve pomocná pole pro získání přeskupení řádků a sloupců matice. MMD a RCM potřebují pro výpočet uložit graf matice sousednosti. Počet uzlů je neměnný a vždy odpovídá rozměru matice, máme tedy $\mathcal{O}(n)$ paměti navíc. Pro každý uzel je potřeba zaznamenat jeho sousedící uzly. V případě hustých matic bychom mohli potřebovat až dalších $\mathcal{O}(n)$ paměti navíc pro každý uzel. I přestože u řídkých matic k takto

vysokému počtu sousedů u každého uzlu nikdy nedojdeme, musíme nyní počítat s $\mathcal{O}(n^2)$. Dále je pak potřebné uložit výsledný permutační vektor, podle kterého se řádky a sloupce matice přeskupí. Ten vždy obsahuje počet prvků rovný počtu uzlů v grafu, výsledkem je tedy $\mathcal{O}(2 \cdot n^2)$ paměti navíc.

Co se týče Markowitzovy strategie, ta pro svůj výpočet potřebuje dvě pole o velikosti rozměru matice, ve kterých se uchovává počet nenulových prvků v jednotlicých řádcích a sloupcích. Z toho vyplývá paměťová složitost $\mathcal{O}(2 \cdot n)$.

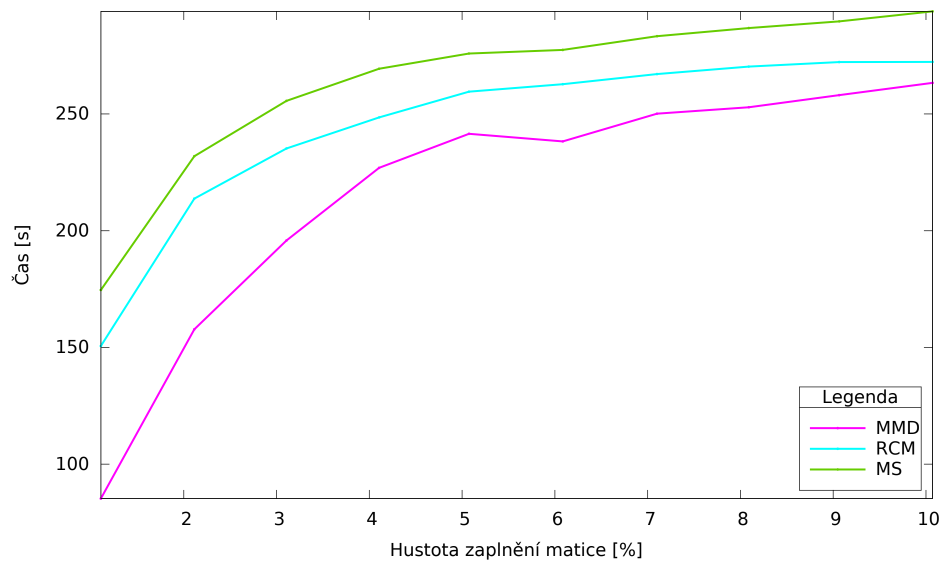
Všechny tři heuristiky navíc potřebují uložit výsledné matice při provádění Doolitlovy metody. Jak již bylo zmíněno v sekci 1.3, s využitím CRS/CCS formátu potřebujeme $\mathcal{O}(2 \cdot (n + nnzL + nnzU))$, kde n je rozměr matice a $nnzL$ a $nnzU$ počet nenulových prvků prvků ve výsledných maticích. Dvojice $nnzL$ a $nnzU$ ale zavisí na tom, jak moc byla heuristika v přeskupování řádků a sloupců úspěšná. Poslední graf 5.11 tedy znázorňuje počet nenulových prvků ve výsledných maticích (jelikož bylo testování prováděno na symetrických maticích, počet nenulových prvků v matici $\mathbb{L}$ je stejný jako počet nenulových prvků v matici $\mathbb{U}$) v závislosti na počtu nenulových prvků ve vstupní matici. Stejně jako předtím, i zde vyšla heuristika MMD nejlépe. Pro vstupní matici s 50 852 nenulovými prvky je ve výsledných maticích s použitím heuristiky MMD 436 387 nenulových prvků, s RCM 470 408 nenulových prvků, s MS 485 877 nenulových prvků a bez použití heuristiky 486 576 nenulových prvků. Je tedy i vidět, že výsledky jsou lepší s heuristikami než bez nich.

I přestože se výsledky na těchto datech tváří optimisticky, nemusí tomu tak být vždy. Heuristika MMD se tady zdá být časově i paměťově nejméně náročná, ale v následující části si ukážeme grafy z měření na datech z repozitáře „Matrix Market“, jejichž výsledky už nejsou takto jednoznačné. Je to právě i kvůli mnoha faktorům, od kterých se doba běhu algoritmů odvíjí, ale hlavně kvůli struktuře vstupní matice.

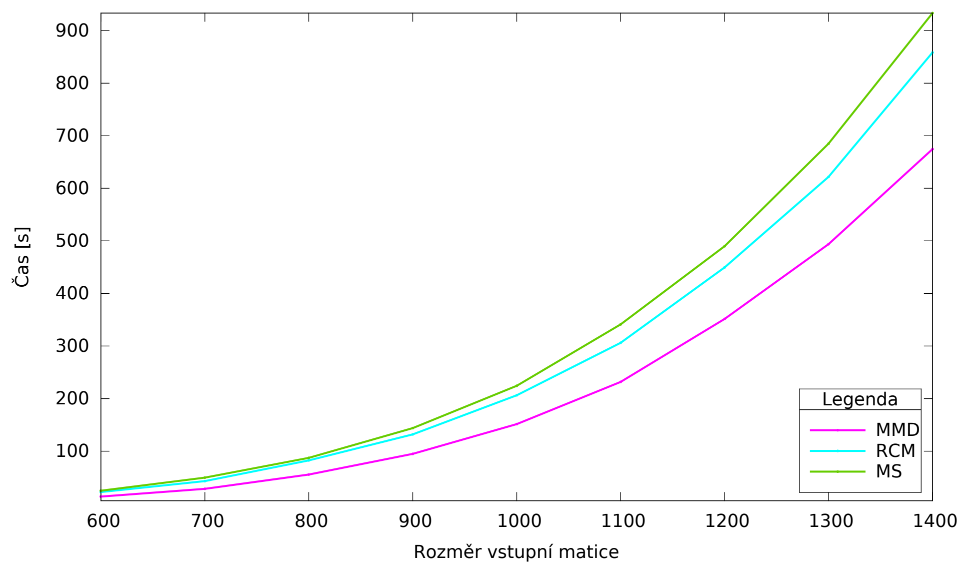
5.5 Měření na datech z repozitáře „Matrix Market“

Testování proběhlo na maticích představených v sekci 5.1. Nejprve se zaměříme na rychlost heuristik. Hned na první pohled je z grafu 5.12 vidět, že výsledky nejsou už tak jednoznačné, jako je tomu v předchozí sekci. Protože graf závisí na více parametrech, velikost bodů určuje počet nenulových prvků ve výsledných maticích, tedy čím víc má matice $\mathbb{L}$ nebo $\mathbb{U}$ NNZ, tím větší je bod.

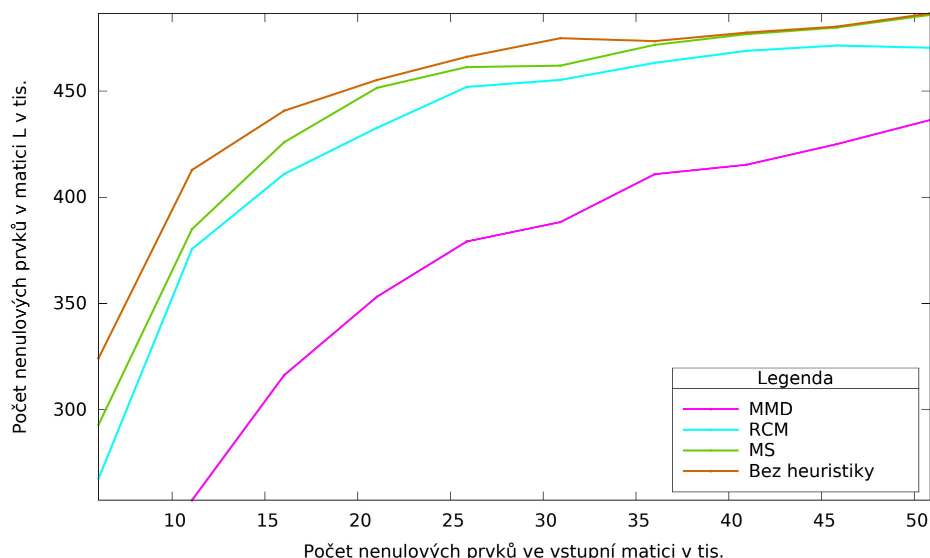
Na vygenerovaných datech jasně vítězila heuristika MMD, zde je to více proměnlivé právě kvůli konečnému počtu nenulových prvků. Na grafu 5.13 jsou porovnávány právě tyto hodnoty. Překvapivě bylo v některých případech lepší spustit Doolittlův algoritmus bez heuristiky. Například pro matici BCSSTK11, která je na grafu 5.13 znázorněna šestým bodem, vytvoří samostatná Doolittlova metoda 77 270 nenulových prvků, se spuštěním heuristiky MS 89 125



Obrázek 5.9: Rychlosti heuristik v závislosti na hustotě vstupní matice



Obrázek 5.10: Rychlosti heuristik v závislosti na rozměru vstupní matice



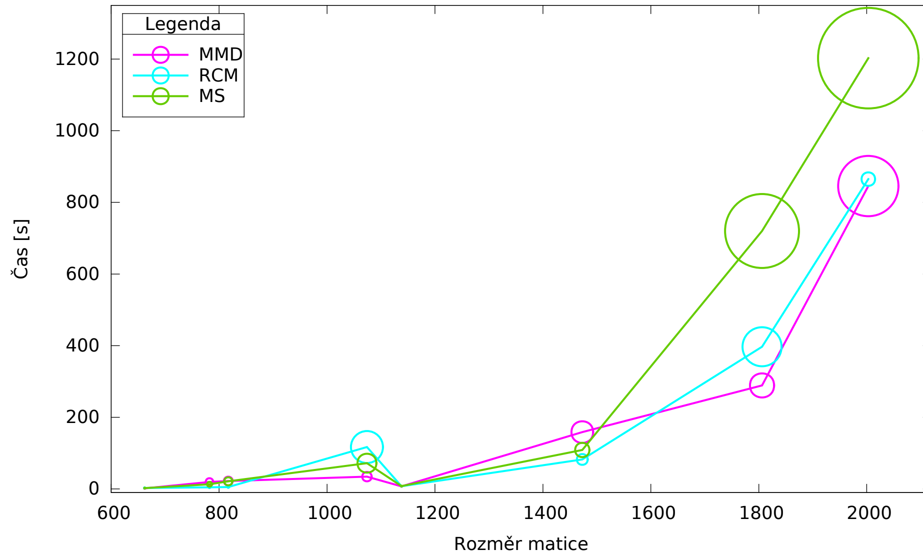
Obrázek 5.11: Počet nenulových prvků ve výsledné matici $\mathbb{L}$ v závislosti na počtu nenulových prvků vstupní matice

prvků, s heuristikou MMD dokonce 135 348 prvků, což je skoro dvojnásobek toho, co provedení Doolittlovy metody bez heuristiky. Na vizualizaci 5.4 této matice je vidět, že jsou nenulové prvky matice převážně soustředěny k diagonále. To je přesně to, o co heuristiky usilují, tedy pokud je vstupem matice, která už sama o sobě bude tvořit málo nových NNZ, heuristika ji už nemůže vylepšit, a není ani vyloučeno, že ji zhorší.

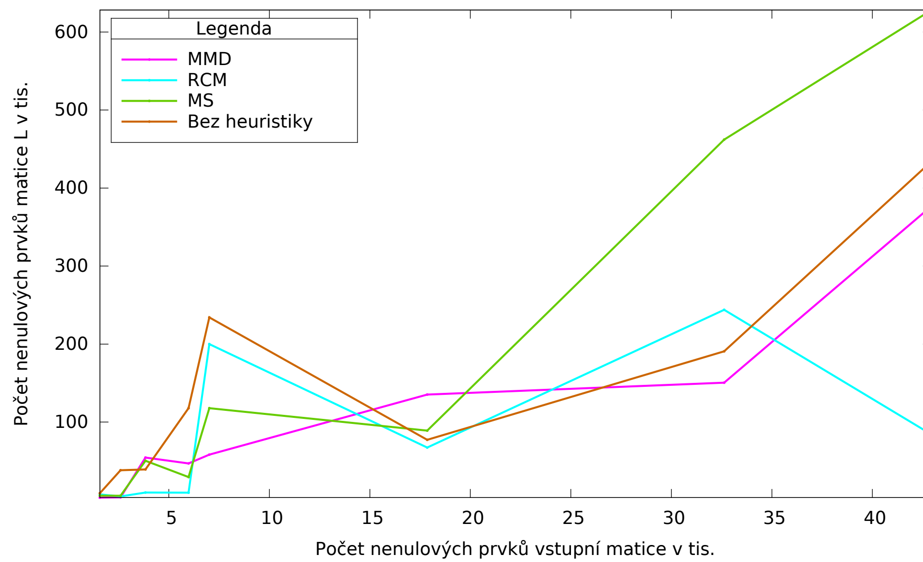
Pro tyto matice vycházela nejlépe heuristika Reverse Cuthill-McKee, jejíž křivka se převážně nacházela pod hodnotami spuštění algoritmu bez heuristiky. Naopak Markowitzova strategie vycházela lépe pouze na matice, které mají více rozprostřené nenulové prvky, tedy matice BFW782B, 622 BUS a 1138 BUS. Pro matice s nenulovými prvky okolo diagonály vychází Markowitzova strategie velmi špatně (například pro matici BCSSTK14 mají výsledné matice po použití Markowitzova strategie skoro o 2,5× víc nenulových prvků, jak můžeme vidět v tabulce 5.1).

5.6 Porovnání vzniku nenulových prvků s existujícím řešením

Na grafu 5.14 můžeme vidět porovnání vzniku nenulových prvků s QR a Choleského rozklady, které byly naimplementovány kolegou Stanislavem Kusým v rámci bakalářské práce *Efektivní LU rozklad pro řídké matice*. Nyní je na ose y součet nenulových prvků v maticích $\mathbb{L}$ a $\mathbb{U}$, neboť QR a Choleského metody



Obrázek 5.12: Rychlost heuristik na datech z repozitáře „Matrix Market“

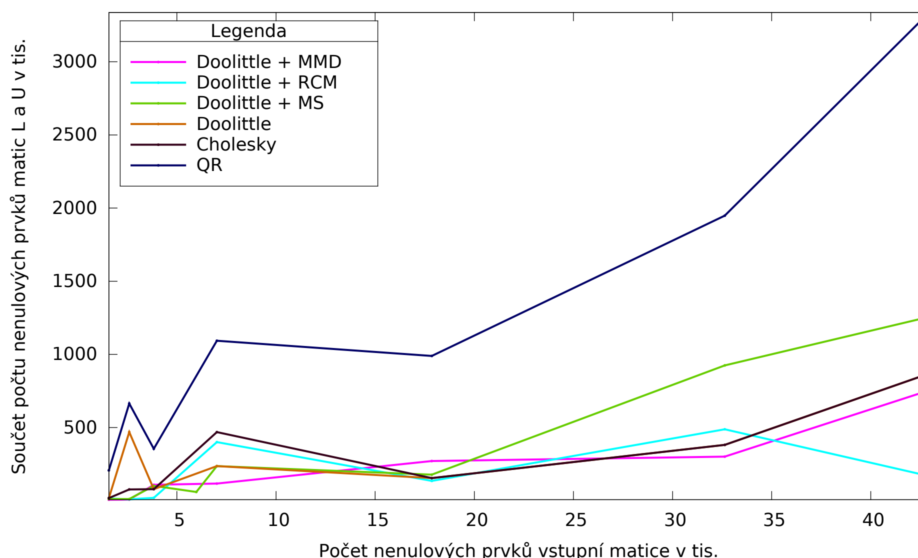


Obrázek 5.13: Počet nenulových prvků ve výsledné matici $\mathbb{L}$ na datech z repozitáře „Matrix Market“

			MMD		RCM		MS		Bez heuristiky
Matice	Rozměr	NNZ	Čas	NNZ	Čas	NNZ	Čas	NNZ	NNZ
662 BUS	662	1568	1,599	3252	2,795	7349	2,264	5574	8830
BFW782B	782	5982	19,438	47026	4,348	9572	13,081	29604	38312
BCSSTK19	817	3835	22,043	54499	5,105	9738	21,277	50435	39433
BCSSTK08	1074	7017	34,265	58372	116,861	199964	72,066	117922	117926
1138 BUS	1138	2596	6,973	3661	7,627	4842	7,816	5597	234160
BCSSTK11	1473	17857	158,696	135348	82,276	67367	108,932	89125	77270
BCSSTK14	1806	32630	288,839	150495	397,035	243956	720,068	462009	190791
BCSSTK13	2003	42943	845,55	376855	864,804	84148	1202,57	628214	434214

Tabulka 5.1: Výsledky měření času a počtu nenulových prvků

nemají při rozkladu symetrických matic stejný počet nenulových prvků ve výsledných maticích, jako je tomu u Doolittlovy metody. Můžeme si všimnout srovnatelných výsledků Doolittlovy metody bez použití heuristiky a Choleského metody. Pro poslední matice vytváří dokonce stejný počet nenulových prvků. Výpočet jednotlivých prvků výsledných matic je velmi podobný. Výsledkem Choleského rozkladu je matice $\mathbb{L}$ a její matice transponovaná $\mathbb{L}^T$, tedy $\mathbb{A} = \mathbb{L}\mathbb{L}^T$.



Obrázek 5.14: Porovnání počtu nenulových prvků ve výsledných maticích $\mathbb{L}$ a $\mathbb{U}$ s metodou QR a Choleského

Pro srovnání jsou níže uvedeny vzorce pro výpočet prvků pro vstupní matice typu (n, n), vlevo pro Doolittlovu metodu a vpravo pro Choleského metodu.

Pro každé $k = 1, 2, 3, ..., n$:

$$u_{k,m} = a_{k,m} - \sum_{j=1}^{k-1}(l_{k,j} \cdot u_{j,m})$$

pro $m = k, k+1, ..., n$,

$l_{k,k} = 1$,

$$l_{i,k} = \frac{1}{u_{k,k}}(a_{i,k} - \sum_{j=1}^{k-1}(l_{i,j} \cdot u_{j,k}))$$

pro $i = k+1, k+2, ..., n$.

Pro každé $k = 1, 2, 3, ..., n$:

$$l_{k,k} = \sqrt{a_{k,k} - \sum_{j=1}^{k-1} l_{j,i}^2},$$

$$l_{k,i} = l_{i,k} = \frac{1}{u_{k,k}}(a_{k,i} - \sum_{j=1}^{k-1}(l_{j,k} \cdot l_{j,i}))$$

pro $i = k+1, k+2, ..., n$.

Výsledky QR metody jsou naopak o dost horší. Je to způsobeno tím, že matice Q není trojúhelníková, počet prvků zde tedy narůstá.

Závěr

V této práci jsme se seznámili s Doolittlovým algoritmem LU rozkladu pro řídké matice a s heuristikami pro snížení počtu nově vzniklých nenulových prvků. Pro symetrické matice to byly heuristiky Minimum Degree a její vylepšená verze Multiple Minimum Degree, Cuthill-McKee a Reverse Cuthill-McKee. Pro nesymetrické matice byla popsána heuristika Markowitzova strategie.

V první kapitole byly zavedeny základní definice z lineární algebry, které jsou nezbytné pro tuto problematiku. Protože pracujeme s řídkými maticemi, byly v této kapitole ukázány různé formáty pro efektivní uložení řídkých matic v paměti. Dále byl popsán Doolittlův algoritmus a byla prozkoumána jeho časová a paměťová složitost.

Poté byly popsány heuristiky pro snížení počtu nově vzniklých nenulových prvků v matici. Protože některé z nich využívají grafy, byly zde uvedeny také základní pojmy z teorie grafů.

Cílem práce bylo naimplementovat Doolittlův algoritmus a tři heuristiky, MMD, RCM a MS, na kterých bylo následně provedeno měření a porovnání časové a paměťové náročnosti.

Na vygenerovaných datech měla nejlepší výsledky heuristika Multiple Minimum Degree. Naopak nejhorších výsledků dosahovala heuristika Markowitzova strategie. Při měření na datech z repozitáře „Matrix Market“ jsme ale získali úplně jiné výstupy. Zjistili jsme, že efektivita heuristik se odvíjí od struktury vstupní matice, a že je pro některé matice dokonce lepší heuristiky vůbec nepoužívat. Obecně ale platí, že je použití heuristik výhodné, a že můžeme dosáhnout rozdílů v řádu až stovek tisíc nevytvořených nenulových prvků oproti výpočtu LU rozkladu bez heuristik.

Literatura

[1] KALOUSOVÁ, Anna: Úvod do algebry. *ČVUT Fakulta elektrotechnická* [online]. 2007, [cit. 2016-03-17]. Dostupné z: https://math.feld.cvut.cz/ftp/kalous/Skripta/skripta.pdf

[2] OLŠÁK, Petr: *Úvod do algebry, zejména lineární.* Vyd 1., Praha: FEL ČVUT v Praze, 2007, ISBN 978-80-01-03775-1.

[3] ŠTAMPACH, František a Karel KLOUDA: BI-LIN : Lineární algebra. *FIT ČVUT* [online]. 2015, [cit. 2016-03-17]. Dostupné z: https://edux.fit.cvut.cz/courses/BI-LIN/_media/lectures/lin-prednaska-all-handouts.pdf

[4] KALOUSOVÁ, Anna: LU rozklad. *ČVUT Fakulta elektrotechnická* [online]. 2011, [cit. 2016-03-17]. Dostupné z: https://math.feld.cvut.cz/ftp/kalous/bilin/prednasky/LUrozklad.pdf

[5] BARTOVSKÝ, Vojtěch: *Násobení řídkých matic.* Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, 2007.

[6] TEMEL, Aleš: *Knihovna pro matematické výpočty v jazyce C++*. Bakalářská práce, Vysoké učení technické v Brně, Brno, 2012.

[7] BURKARDT, John: The Matrix Market File Format [online]. 2006, [cit. 2016-03-17]. Dostupné z: http://people.sc.fsu.edu/~jburkardt/data/mm/mm.html

[8] THOMPSON, Erik G.: Doolittle Decomposition of a Matrix [online]. 2005, [cit. 2016-03-17]. Dostupné z: http://www.engr.colostate.edu/~thompson/hPage/CourseMat/Tutorials/CompMethods/doolittle.pdf

[9] XU, Zhiliang: Doolittle's method of LU factorization [online]. 2011, [cit. 2016-03-17]. Dostupné z: http://www3.nd.edu/~zxu2/acms40390F11/Alg-LU-Crout.pdf

[10] KOLÁŘ, Josef: *Teoretická informatika*. 2. vyd., Praha: Česká informatická společnost, 2000, ISBN 80-900853-8-5.

[11] HEGGERNES, P., S.C. EISENSTAT, G. KUMFERT, a A. POTHEN: The Computational Complexity of the Minimum Degree Algorithm. 2001, [cit. 2016-03-19]. Dostupné z: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA398632

[12] LIU, Joseph W. H.: Modification of the Minimum-degree Algorithm by Multiple Elimination. *ACM Trans. Math. Softw.*, ročník 11, č. 2, Červen 1985: s. 141–153, ISSN 0098-3500, doi:10.1145/214392.214398. Dostupné z: http://doi.acm.org/10.1145/214392.214398

[13] ZĂVOIANU, Ciprian: Tutorial: Bandwidth reduction - The CutHill-McKee Algorithm [online]. 2009, [cit. 2016-03-18]. Dostupné z: http://ciprian-zavoianu.blogspot.cz/2009/01/project-bandwidth-reduction.html

[14] Sparsity and the Optimal Ordering of Circuit Equations [online]. 1996, [cit. 2016-04-02]. Dostupné z: http://users.ecs.soton.ac.uk/mz/CctSim/chap1_6.htm

[15] KUSÝ, Stanislav: *Efektivní LU rozklad pro řídké matice*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

[16] JIRÁSEK, Milan: *Řešiče rozsáhlých soustav lineárních rovnic*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, 2010.

[17] TURČAN, Lukáš: *Řešiče rozsáhlých soustav linéárních rovnic*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

[18] The R Core Team: R: A Language and Environment for Statistical Computing. 2016, [cit. 2016-05-11]. Dostupné z: https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf

[19] Linear algebra (scipy.linalg). 2014, [cit. 2016-05-11]. Dostupné z: http://docs.scipy.org/doc/scipy-0.14.0/reference/tutorial/linalg.html

[20] LU matrix factorization (lu). 2016, [cit. 2016-05-11]. Dostupné z: http://www.mathworks.com/help/matlab/ref/lu.html

[21] LI, Xiaoye S., James W. DEMMEL, John R. GILBERT, Laura GRIGORI, Meiyue SHAO a Ichitaro YAMAZAKI: SuperLU Users' Guide. 2011, [cit. 2016-05-11]. Dostupné z: http://crd-legacy.lbl.gov/~xiaoye/SuperLU/superlu_ug.pdf

[22] GCC Team: GCC online documentation. 2016, [cit. 2016-05-10]. Dostupné z: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Příloha A

Seznam použitých zkratek

AMD Approximate Minimum Degree

COLAMD Column Approximate Minimum Degree

CM Cuthill-McKee

COO Coordinate format

CSC Compressed Sparse Column format

CSR Compressed Sparse Row format

GEM Gaussova eliminační metoda

LU Lower-Upper

MD Minimum Degree

MMD Multiple Minimum Degree

MM Matrix Market

MS Markowitzova strategie

NNZ Nonzeros (počet nenulových prvků)

RCM Reverse Cuthill-McKee

STL Standard Template Library

Příloha **B**

Obsah přiloženého CD

```
readme.txt.................................stručný popis obsahu CD
src
  impl.................................zdrojové kódy implementace
    data.........................................testovací data
  thesis.................... zdrojová forma práce ve formátu LaTeX
    pictures...................................obrázky k práci
text ..................................................text práce
  BP_Turcajová_Gabriela_2016.pdf ...... text práce ve formátu PDF
  ZZP.pdf ........................... zadání práce ve formátu PDF
```