Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Metody a algoritmy rozkladů Booleovských matic

2015

Radek Janoštík

Vedoucí práce:
Mgr. Martin Trnečka

Studijní obor:
Informatika, prezenční forma

Bibliografické údaje

Autor:	Radek Janoštík
Název práce:	Metody a algoritmy rozkladů Booleovských matic
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2015
Studijní obor:	Informatika, prezenční forma
Vedoucí práce:	Mgr. Martin Trnečka
Počet stran:	67
Přílohy:	1 CD/DVD
Jazyk práce:	český

Bibliograhic info

Author:	Radek Janoštík
Title:	Methods and algorithms for Boolean matrix decomposition
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2015
Study field:	Computer Science, full-time form
Supervisor:	Mgr. Martin Trnečka
Page count:	67
Supplements:	1 CD/DVD
Thesis language:	Czech

Anotace

Rozklad matice je vyjádření matice za pomocí dvou dílčích matic takových, že jejich roznásobením dostaneme původní matici. Rozklad slouží k odhalení skrytých faktorů v datech. Existuje mnoho algoritmů pro rozklad obecných matic, v této práci se zaměřuji na základní algoritmy pro rozklad Booleovských matic, tedy matic, které obsahují pouze hodnoty 0 a 1. Naleznete zde úvod do problematiky rozkladu matic, generování náhodných matic obsahujících faktory, podrobný popis základních algoritmů, vliv jejich parametrů na rozklad a jejich vlastnosti. V poslední části naleznete srovnání všech popsaných algoritmů jak na reálných, tak i na náhodných datech.

Synopsis

The Boolean matrix decomposition is a method for the decomposition of Boolean matrix into a product of two (smaller) matrices. Decomposition is used to reveal hidden relationships (factors) in data. There's lots of algorithms for decomposition of real-valued matrices, in this thesis are described basic algorithms for decomposition of Boolean matrices. This thesis contains description of the problem, detailed description of selected algorithms with their parameters and properties. In the last part is a comparsion of algorithms on real and randomly generated data.

Klíčová slova: Rozklad binárních matic; Booleovská faktorová analýza; formální konceptuální analýza

Keywords: Binary matrix decomposition; Boolean factor analysis; Formal concept analysis

Děkuji mému vedoucímu Mgr. Martinu Trnečkovi za trpělivost při konzultacích a jeho čas. Rodině za podporu.

Místopřísežně prohlašuji, že jsem celou práci včetně příloh vypracoval/a samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

datum odevzdání práce podpis autora

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Pomocí Booleovských (binárních) matic můžeme reprezentovat vztahy mezi objekty a atributy, kde řádky matice reprezentují objekty a sloupce atributy. Objekt i má atribut j, jestliže se v matici vyskytuje 1 na i-tém řádku v j-tém sloupci.

Tato reprezentace se může použít například v internetovém obchodě, kde by řádky reprezentovaly zákazníky a sloupce zboží, které si zakoupili. Případně může binární matice simulovat transakční databázi nebo výsledky měření fyzikálních jevů, kde řádky symbolizují jednotlivá měření.

Třetím, a ne posledním, využitím reprezentace vztahu mezi objekty a atributy pomocí binární matice je z medicínské praxe, kde řádky matice reprezentují pacienty a sloupce možné příznaky chorob.

Binární data lze snadno získat, ale při jejich velkém množství se stávají pro člověka těžce čitelná a je téměř nemožné z nich vyčíst závislost či vzor. Proto je vhodné provést nad daty analýzu, která odhalí skryté souvislosti v datech. Jednou z takových analýz je *faktorová analýza*, která dokáže odhalit faktory, které vysvětlují vznik dat.

Výsledkem faktorové analýzy prvního příkladu jsou faktory, které se snaží vysvětlit koupi daného zboží. Poté může faktory pojmenovat odborník a výsledkem bude například faktor „žena v domácnosti“ nebo „nákupčí knihkupectví“. Tento výsledek může poté posloužit ke klasifikaci zákazníků a k cílené reklamě. Ve třetím příkladě mohou faktory splývat s nemocemi.

V podkapitole 12.4 je proveden rozbor faktorů faktorové analýzy binární matice, která reprezentuje zvířata. Některé faktory splývají s taxonomickými kategoriemi, které zavedli zoologové.

Binární faktorová analýza přímo vede na rozklad binárních matic. Pokud známe jednotlivé faktory, můžeme sestrojit dvě matice. První bude reprezentovat vztah objekt-faktor, tedy jaké faktory ovlivňují vlastnosti objektu, druhá faktor-atribut, tedy jaké atributy zahrnuje daný faktor. Byla-li analýza přesná, (binárním) vynásobením takto vzniklých matic dostaneme původní objektově-atributovou matici.

1.1 Základní pojmy

Definice 1

Binární matice I dimenze $n \times m$ je matice, která má n řádků a m sloupců a kde $I_{ij} \in \{0, 1\}$ pro všechna $i \in \{1, \ldots, n\}, j \in \{1, \ldots m\}$. i-tý řádkový vektor matice značíme $I_{i\_}$ a j-tý sloupcový vektor $I_{\_j}$.

Definice 2

Objektově-atributová matice je jakákoliv binární matice I.

Každá binární matice může reprezentovat vztah objektů a atributů. Dané matici je pouze přiřazen určitý význam.

Příklad 1

Mějme 5 objektů, u kterých sledujeme 4 atributy. Přítomnost těchto atributů u objektů můžeme vyjádřit pomocí binární matice dimenze 5×4 následovně:

	a_1	a_2	a_3	a_4
o_1	×	×		×
o_2	×	×	×	
o_3	×			×
o_4	×		×	×
o_5	×		×	

$$\approx \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Definice 3

Hustota binární matice I dimenze $n \times m$ je poměr 1 obsažených v matici a násobku rozměrů matice.

$$dens(I) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} I_{ij}}{nm}$$

Je zřejmé, že $dens(I) \in [0, 1]$, přičemž $dens(I) = 0$ značí, že matice I je nulová a $dens(I) = 1$, že matice I obsahuje pouze 1. Někdy se hustota udává v procentech.

Příklad 2

Matice z příkladu 1 obsahuje 13 jedniček, tedy její hustota je $\frac{13}{20}$, tedy 65 %.

Definice 4

Pro binární matice I_1 a I_2 definujeme:

$$I_1 \leq I_2 \ (I_1 \text{ je obsažena v } I_2) \text{ p.k. } (I_1)_{ij} \leq (I_2)_{ij} \text{ pro všechna } i, j$$

Definice 5

Binární matice je klarifikovaná, jestliže neobsahuje duplicitní řádky ani duplicitní sloupce. Klarifikace, tedy odstranění duplicitních řádků a sloupců, neovlivňuje vlastnosti dat, pouze odstraňuje redundantní informace.

Definice 6

Rozkladem binární matice I dimenze $n \times m$ jsou dílčí matice A a B s dimenzemi $n \times k$ a $k \times m$ takové, že $A \circ B = I$. Násobení binárních matic je definováno:

$$(A \circ B)_{ij} = \bigvee_{l=1}^{k} A_{il} \cdot B_{lj}$$

kde $\bigvee$ značí logickou disjunkci a $\cdot$ logickou konjunkci. Číslo k je nazýváno prostředním rozměrem rozkladu nebo společným rozměrem matic A a B.

Příklad 3

Matici z příkladu 1 lze rozložit na dvě dílčí matice s dimenzemi 5×3 a 3×4 následovně:

$$\begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \circ \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Pro danou matici nemusí existovat jedinečný rozklad, jednotlivé rozklady se mohou lišit pořadím řádků a sloupců, dokonce i společným rozměrem k. Příkladem může být triviální rozklad za pomocí jednotkové matice, ten je možný vždy, ale nemá žádnou analytickou hodnotu.

Definice 7

Nejmenší k takové, že existuje rozklad matice I dimenze $n \times m$ na matice A, B s dimenzemi $n \times k$ a $k \times m$ se nazývá Booleovská hodnost (rank) matice I. Značí se $rank_B(I)$.

Problém 1 (Rozklad binární matice)**.** *Pro danou binární matici I dimenze $n \times m$ naleznout rozklad na matice A s dimenzí $n \times k$ a B s dimenzí $k \times m$ s co nejmenším společným rozměrem k.*

Věta 1

Problém nalezení rozkladu binární matice I dimenze $n \times m$ na matice A dimenze $n \times k$ a B dimenze $k \times m$ tak, aby společný rozměr k byl co nejmenší, je NP-těžký a odpovídající rozhodovací problém, zda existuje takové k, pro které existuje rozklad $I = A \circ B$ se společným rozměrem k je NP-úplný problém. [3]

Důkaz

Pomocí redukce *Set basis problému* (SBP) na problém rozkladu binární matice. Set basis problém: Rozhodnout, zda pro množinu $S = \{S_1, \dots S_n\}$ množin $S_i \subseteq \{1, \dots, m\}$ a kladné celé číslo k existuje množina $C = \{C_1, \dots, C_k\}$ podmnožin $C_l \subseteq \{1, \dots, m\}$ taková, že pro každou S_i existuje $D_i \subseteq \{C_1, \dots, C_k\}$, pro které $\bigcup D_i = S_i$. Příslušný optimalizační problém: Pro dané S najít nejmenší C takové, že splňuje předchozí podmínky. Rozhodovací problém je NP-úplný, příslušný optimalizační problém je NP-těžký.[25]

Redukce SBP na rozklad binární matice: Pro dané S sestrojíme $n \times m$ matici I, kde $I_{ij} = 1$ p.k. $j \in S_i$. Lze ověřit, že $I = A \circ B$ platí pro binární matice A, B s rozměry $n \times k$ a $k \times m$ p.k. C_l $(l = 1, \dots k)$ a D_i definované jako $j \in C_l$ p.k. B_{lj} a $C_l \in D_i$ p.k. $A_{il} = 1$ jsou řešením SBP pro dané S. Tedy problém nalezení rozkladu $I = A \circ B$ s co nejmenším společným rozměrem k je NP-těžký a příslušný rozhodovací problém je NP-úplný. □

Při reálných aplikacích se může stát, že data nejsou přesná. Jedná se buď o chyby v měření, nepřesně zaznamenaná data či chyby v datových přenosech. Tyto odchylky jsou někdy označovány jako *binární šum*. Algoritmy jako například Asso [18] či PaNDa [17] se snaží šum potlačit a částečně jej nějakým způsobem „odfiltrovat“. Touto filtrací nemusí docházet k přesnému rozkladu, ale zato k analyticky hodnotnějšímu výsledku. Naopak u přesných algoritmů může záměna několika 1 a 0 vést k naprosto odlišným výsledkům.

Při nepřesném rozkladu mohou nastat dva druhy chyb:

- Nepokrytí – V původní matici se na dané pozici vyskytuje 1, ve výsledném násobku je 0
- Překrytí – V původní matici se na dané pozici vyskytuje 0, ve výsledném násobku je 1

Celkovou chybu E vyjádříme jako:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{m} |I_{ij} - (A \circ B)_{ij}|$$

Tedy jako počet pozic, na kterých se liší původní matice s násobkem dílčích matic.

V případě binárního šumu je důležité si uvědomit, na kolik ovlivňuje data a nalézt hranici, jakou míru šumu vůbec uvažovat. Například v článku [18] autoři uvažují míru šumu až 40 %, což vyvolává diskuzi, zda lze vůbec s takovou mírou šumu pracovat a zda budou výsledky relevantní.

Definice 8
Algoritmus vytváří rozklad zdola, jestliže pro jakýkoliv vstup nedojde k překrytí.

1.2 Použití metod pro obecné matice

V následující podsekci popíši dvě známé metody pro rozklad matic s reálnými hodnotami a jejich použití pro binární matice. Metod pro matice s reálnými hodnotami existuje mnoho a jejich význam a interpretace rozkladu se odlišuje od binárního případu. Například LU rozklad [27], který se může použít pro řešení lineárních rovnic či výpočtu determinantu matice a další metody založené na rozkladu pomocí vlastních hodnot.

První metodou je *Singular value decomposition* (SVD) [10], která rozkládá matici A dimenze $n \times m, n \geq m$ na dvě ortogonální matice U a V s dimenzemi $n \times n$ a $m \times m$ a diagonální matici Σ dimenze $m \times m$, ve které jsou hodnoty na diagonále nerostoucí tak, že:

$$A = U \begin{pmatrix} \Sigma \\ 0 \end{pmatrix} V^T$$

Hodnoty na diagonále matice Σ se nazývají *singulární hodnoty* a interpretují se jako „důležitost“ faktoru. Je dokázáno, že rozklad pomocí SVD je optimální vzhledem k *Frobeniově normě*. Tato metoda může sloužit k výpočtům vlastností matic (hodnost, báze oboru hodnot a nulového prostoru, norma matice $|| \cdot ||_2$), používá se ve zpracování signálu a obrazu, doporučovacích systémech či získávání informací z dat. [4].

Při použití této metody na binární data dílčí matice obsahují reálné hodnoty, jež mohou být dokonce i záporné, což způsobuje nepřímočarou interpretaci rozkladu.

Příklad 4

Pro matici z příkladu 1 je výsledný rozklad (pomocí funkce *svd* [24] v Matlabu, dílčí matice zaokrouhleny na tisíciny) následující:

$$U = \begin{pmatrix} -0.474 & -0.500 & -0.522 & -0.047 & 0.500 \\ -0.474 & 0.500 & -0.522 & -0.047 & -0.500 \\ -0.375 & -0.500 & 0.294 & 0.522 & -0.500 \\ -0.518 & 0.000 & 0.530 & -0.671 & 0.000 \\ -0.375 & 0.500 & 0.294 & 0.522 & 0.500 \end{pmatrix}, \Sigma = \begin{pmatrix} 3.091 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.414 & 0.000 & 0.000 \\ 0.000 & 0.000 & 1.131 & 0.000 \\ 0.000 & 0.000 & 0.000 & 0.404 \\ 0.000 & 0.000 & 0.000 & 0.000 \end{pmatrix}$$

$$V = \begin{pmatrix} -0.717 & 0.000 & 0.066 & 0.694 \\ -0.307 & -0.000 & -0.924 & -0.230 \\ -0.442 & 0.707 & 0.267 & -0.483 \\ -0.442 & -0.707 & 0.267 & -0.483 \end{pmatrix}$$

Po roznásobení dostaneme výsledek:

$$U\Sigma V^T = \begin{pmatrix} 1.000 & 1.000 & -0.000 & 1.000 \\ 1.000 & 1.000 & 1.000 & 0.000 \\ 1.000 & -0.000 & -0.000 & 1.000 \\ 1.000 & -0.000 & 1.000 & 1.000 \\ 1.000 & -0.000 & 1.000 & -0.000 \end{pmatrix}$$

Což je přesně původní matice, avšak metoda nemusí vždy vést k přesnému výsledku, viz 4. kapitola [18], kde autoři uvažují verzi SVD se zaokrouhlením výsledných hodnot.

Problém s interpretací záporných hodnot řeší metoda Non-negative Matrix Factorization (NMF) [15], která sice není optimální (může konvergovat k lokálnímu optimu), ale rozkládá zdrojovou matici na dvě dílčí matice pomocí nezáporných reálných hodnot. Po zaokrouhlení výsledných matic může dávat dobré výsledky.

Příklad 5

Následující dvě matice tvoří nezáporný rozklad matice z příkladu 1, vypočteny byly v Matlabu pomocí funkce *nnmf* [20] s výchozím algoritmem.

$$A = \begin{pmatrix} 0.000 & 1.216 & 1.046 \\ 1.219 & 0.000 & 1.046 \\ 0.127 & 1.341 & 0.048 \\ 1.061 & 1.051 & 0.000 \\ 1.346 & 0.117 & 0.048 \end{pmatrix}, B = \begin{pmatrix} 0.573 & 0.000 & 0.820 & 0.005 \\ 0.569 & 0.000 & 0.000 & 0.823 \\ 0.294 & 0.956 & 0.000 & 0.000 \end{pmatrix}$$

$$(A \cdot B) = \begin{pmatrix} 0.997 & 1.001 & 0.002 & 1.002 \\ 1.005 & 1.007 & 1.002 & 0.011 \\ 0.851 & 0.045 & 0.104 & 1.104 \\ 1.206 & 0.000 & 0.869 & 0.869 \\ 0.852 & 0.045 & 1.103 & 0.029 \end{pmatrix} \approx \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

Rozklad je sice přesný, ale interpretace dílčích matic je náročnější. Nabízí se možnost zaokrouhlit i dílčí matice. U tohoto příkladu by se výsledek nezměnil, ale obecně se v dílčích maticích mohou vyskytovat jakákoliv nezáporná čísla a obyčejné zaokrouhlení by nemuselo přinést čistě binární matici.

Pokud nezaokrouhlíme dílčí matice, tak i násobení mezi maticemi je klasické násobení matic a výsledné hodnoty nemusí pouhým zaokrouhlením padnout do binárních hodnot. Nastává tedy problém, jak určit práh hodnot, který rozdělí, zda ve výsledné matici bude 0 či 1.

Tento problém a nepřímá interpretace rozkladů dělá většinu obecných metod nevhodnými pro rozklady binárních matic, a proto je vhodné pro binární data použít metody využívající čistě binární rozklad.

1.3 Základní pojmy z formální konceptuální analýzy

Formální konceptuální analýza (FCA) [7] je metoda pro analýzu (binárních) dat, na které jsou založeny první tři algoritmy, jež byly vybrány do této práce. Z těchto důvodů zde uvedu základní pojmy a vztahy.

Definice 9

Mějme množinu objektů $X = \{1, \ldots, n\}$ a množinu atributů $Y = \{1, \ldots, m\}$ a binární relaci I mezi X a Y. Trojice $\langle X, Y, I \rangle$ se nazývá formální kontext.

Definice 10

Formální koncept formálního kontextu $\langle X, Y, I \rangle$ je jakýkoliv pár $\langle C, D \rangle$ množin $C \subseteq X$, $D \subseteq Y$ takový, že $C^{\uparrow} = D$ a $D^{\downarrow} = C$, kde $C^{\uparrow}$ je množina všech atributů, které mají všechny objekty z C a $D^{\downarrow}$ je množina všech objektů, které mají všechny atributy z D, tedy:

$$\begin{aligned} C^{\uparrow} &= \{y \in Y \mid \forall x \in C : \langle x, y \rangle \in I\} \\ D^{\downarrow} &= \{x \in X \mid \forall y \in D : \langle x, y \rangle \in I\} \end{aligned}$$

Je-li $\langle C, D \rangle$ formální koncept, pak se C nazývá *extent* a D *intent*.

Definice 11

Množinu všech formální konceptů $\langle X, Y, I\rangle$ značíme $\mathcal{B}(X, Y, I)$ a je definována následovně:

$$\mathcal{B}(X, Y, I) = \{\langle C, D\rangle | C^{\uparrow} = D, D^{\downarrow} = C\}$$

Spolu s částečným uspořádáním formálních konceptů $\leq$ definovaným pro $\langle C_1, D_1\rangle$, $\langle C_2, D_2\rangle \in \mathcal{B}(X, Y, I)$ následovně:

$$\langle C_1, D_1\rangle \leq \langle C_2, D_2\rangle \text{ p.k. } C_1 \subseteq C_2 \text{ p.k. } D_2 \subseteq D_1$$

je $\mathcal{B}(X, Y, I)$ úplný svaz, nazývaný *konceptuální svaz*.

Formální koncepty lze použít pro konstrukci rozkladu binárních matic [12]. Z konceptuálního svazu lze vybrat množina konceptů $\mathcal{F} = \{\langle A_1, B_1\rangle, \ldots, \langle A_k, B_k\rangle\}$, $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$, ze kterých lze zkonstruovat rozklad $A_{\mathcal{F}} \circ B_{\mathcal{F}}$ matice I takto: [3]

$$(A_{\mathcal{F}})_{il} = \begin{cases} 0 & i \in A_l \\ 1 & i \notin A_l \end{cases}$$

$$(B_{\mathcal{F}})_{lj} = \begin{cases} 0 & j \in B_l \\ 1 & j \notin B_l \end{cases}$$

Pro $l = 1, \ldots, k$. Tedy l-tý sloupec $(A_{\mathcal{F}})_{\_l}$ matice $A_{\mathcal{F}}$ obsahuje charakteristický vektor extentu A_l a l-tý řádek $(B_{\mathcal{F}})_{l\_}$ obsahuje charakteristický vektor intentu B_l.

Příklad 6

Pro formální kontext $\langle X, Y, I\rangle$ reprezentovaný maticí z příkladu 1 je konceptuální svaz $\mathcal{B}(X, Y, I) = \Big\{\langle\{1,2,3,4,5\},\{1\}\rangle, \langle\{1,3,4\},\{1,4\}\rangle, \langle\{2,4,5\},\{1,3\}\rangle, \langle\{4\},\{1,3,4\}\rangle, \langle\{1,2\},\{1,2\}\rangle, \langle\{1\},\{1,2,4\}\rangle, \langle\{2\},\{1,2,3\}\rangle, \langle\emptyset,\{1,2,3,4\}\rangle\Big\}$
Označme koncepty postupně $\langle A_1, B_1\rangle, \ldots, \langle A_8, B_8\rangle$. Při výběru $\langle A_1, B_1\rangle, \langle A_2, B_2\rangle$, $\langle A_3, B_3\rangle$ a $\langle A_5, B_5\rangle$ zkonstruujeme rozklad:

$$A_{\mathcal{F}} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}, B_{\mathcal{F}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

Tento rozklad je přesný, ale není optimální. Lepším rozkladem je rozklad z příkladu 3, kde byly použity koncepty $\langle A_2, B_2\rangle, \langle A_3, B_3\rangle$ a $\langle A_5, B_5\rangle$.

Věta 2 (Existence rozkladu při použití formálních konceptů)
Pro každou binární matici I existuje $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ taková, že $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ [3].

Důkaz
Vyplývá z toho, že $I_{ij} = 1$ p.k. existuje formální koncept $\langle C, D\rangle \in \mathcal{B}(X, Y, I)$ t.ž. $i \in C$ a $j \in D$. (Každá 1 v matici I je obsažena alespoň v jednom formálním konceptu.) Stačí vzít $\mathcal{F} = \mathcal{B}(X, Y, I)$ a dostaneme $(A_{\mathcal{F}} \circ B_{\mathcal{F}})_{ij}$ p.k. existuje l t.ž. $(A_{\mathcal{F}})_{il} = 1$ a $(B_{\mathcal{F}})_{lj}$ p.k. existuje $\langle C_l, D_l\rangle \in \mathcal{B}(X, Y, I), i \in C_l$ a $j \in D_l$ p.k. $I_{ij} = 1$. □

Věta 3 (Optimalita rozkladu při použití formálních konceptů)
Mějme rozklad matice $I = A \circ B$ na dílčí matice A a B s dimenzemi $n \times k$ a $k \times m$. Pak existuje množina formálních konceptů $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ o velikosti $|\mathcal{F}| \leq k$ taková, že matice $A_{\mathcal{F}}$ a $B_{\mathcal{F}}$ s dimenzemi $n \times |\mathcal{F}|$ a $|\mathcal{F}| \times m$ tvoří rozklad matice $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Důkaz
Kvůli rozsahu vynechán [3]. □

Část I
Algoritmy pro rozklad binárních matic

2 GreCon

Algoritmus GreCon [3] je založený na FCA a vytváří rozklad zdola. Zkratkové slovo GreCon zastupuje výraz *Greedy Concept*. Jak již název napovídá, algoritmus greedy způsobem vybírá formální koncept, který přidává do rozkladu.

Algoritmus 1 GreCon

Input: Binární matice I
Output: Množina faktorů $\mathcal{F}$ tvořící rozklad

1: $\mathcal{S} \leftarrow \mathcal{B}(X, Y, I)$
2: $\mathcal{U} \leftarrow \{\langle i, j\rangle | I_{ij} = 1\}$
3: $\mathcal{F} \leftarrow \emptyset$
4: **for each** $\langle C, D\rangle \in \mathcal{S}$ **do**
5: **if** $\langle C, D\rangle \in \mathcal{O}(X, Y, I) \cap \mathcal{A}(X, Y, I)$ **then**
6: $\mathcal{F} \leftarrow \mathcal{F} \cup \langle C, D\rangle$
7: $\mathcal{S} \leftarrow \mathcal{S} \setminus \langle C, D\rangle$
8: **for each** $\langle i, j\rangle \in C \times D$ **do**
9: $\mathcal{U} \leftarrow \mathcal{U} \setminus \langle i, j\rangle$
10: **while** $\mathcal{U} \neq \emptyset$ **do**
11: **vyber** $\langle C, D\rangle \in \mathcal{S}$ **které maximalizuje** $(C \times D) \cap \mathcal{U}$:
12: $\mathcal{F} \leftarrow \mathcal{F} \cup \langle C, D\rangle$
13: $\mathcal{S} \leftarrow \mathcal{S} \setminus \langle C, D\rangle$
14: **for each** $\langle i, j\rangle \in C \times D$ **do**
15: $\mathcal{U} \leftarrow \mathcal{U} \setminus \langle i, j\rangle$
16: **return** $\mathcal{F}$

Na řádku 1 dochází k výpočtu celého konceptuálního svazu. Implementace této části přináší několik dalších problémů, proto výpočtu konceptuálního svazu věnuji podkapitolu 2.1.

V cyklu na řádcích 4–9 dochází k výběru *povinných konceptů* do rozkladu. Tyto koncepty budou vždy přítomné v rozkladu.

Definice 12
Mějme množinu $\mathcal{O}(X, Y, I)$ všech objektových konceptů a množinu $\mathcal{A}(X, Y, I)$

všech atributových konceptů definovaných:

$$\begin{aligned}\mathcal{O}(X,Y,I) &= \left\{\langle\{x\}^{\uparrow\downarrow},\{x\}^{\uparrow}\rangle \,|\, x \in X\right\}\\ \mathcal{A}(X,Y,I) &= \left\{\langle\{y\}^{\downarrow},\{y\}^{\downarrow\uparrow}\rangle \,|\, y \in Y\right\}\end{aligned}$$

Koncepty z množiny $\mathcal{M}(X,Y,I) = \mathcal{O}(X,Y,I) \cap \mathcal{A}(X,Y,I)$ nazýváme povinnými.

Věta 4
Jestliže $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$ pro nějaké $\mathcal{F} \subseteq \mathcal{B}(X,Y,I)$ pak $\mathcal{M}(X,Y,I) \subseteq \mathcal{F}$.

Důkaz
Jestliže $\langle C,D\rangle \in \mathcal{M}(X,Y,I)$, tedy $\langle C,D\rangle = \langle\{x\}^{\uparrow\downarrow},\{x\}^{\uparrow}\rangle$ pro nějaké $x \in X$ a $\langle C,D\rangle = \langle\{y\}^{\downarrow},\{y\}^{\downarrow\uparrow}\rangle$ pro nějaké $y \in Y$, pak $\langle C,D\rangle$ je jediným formálním konceptem z $\mathcal{B}(X,Y,I)$ takovým, že $x \in C$ a $y \in D$.

Pokud by existoval jiný koncept $\langle C_1,D_1\rangle$ takový, že $\{x\} \in C_1$, pak $C = \{x\}^{\uparrow\downarrow} \subseteq C_1^{\uparrow\downarrow} = C_1$. Z antitonie $^{\uparrow}$ dostáváme: $D = C^{\uparrow} \supseteq C_1^{\uparrow} = D_1$. Dále z $\{y\} \subseteq D_1$ dostáváme $D = \{y\}^{\downarrow\uparrow} \subseteq D_1^{\downarrow\uparrow} = D$. Tedy $D = D_1$, a tedy $\langle C,D\rangle = \langle C_1,D_1\rangle$. Tedy $\langle C,D\rangle$ je jediným konceptem, který pokrývá 1 na průseku řádku a sloupce odpovídajícím x a y. □

Předchozí věta říká, že povinné koncepty $\mathcal{M}(X,Y,I)$ budou v každém rozkladu matice I, protože pro každý $\langle C,D\rangle \in \mathcal{M}(X,Y,I)$ platí, že jedničky pokryté formálním konceptem $\langle C,D\rangle$ nepokrývá žádný jiný formální koncept.

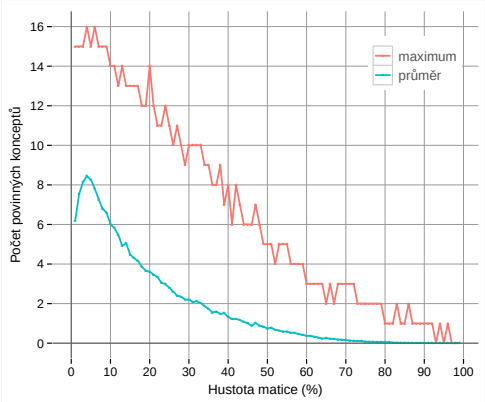
Cyklus na řádcích 4–9 je možné vynechat, protože povinné koncepty se stejně během výpočtu musí vybrat. Pokud je vybereme předem, dochází ke zrychlení, které je dobře patrné u vstupů, které generují velký konceptuální svaz. Průchod velkým svazem zabírá většinu času algoritmu a výběrem povinných konceptů zredukujeme počet průchodů, aniž bychom ovlivnili výsledek.

Při experimentech jsem si všiml, že počet povinných konceptů s hustotou matice klesá. Tedy přidání povinných konceptů sníží počet průchodů konceptuálního svazu v případech, kdy je svaz ještě relativně malý, tedy zrychlení není tak markantní. Na obrázcích 1 a 2 je zobrazen průměrný počet povinných faktorů v závislosti na hustotě matice. Minimální hodnota je vynechána, protože velmi rychle byla nulová. Průměr je z 1000 vzorků.

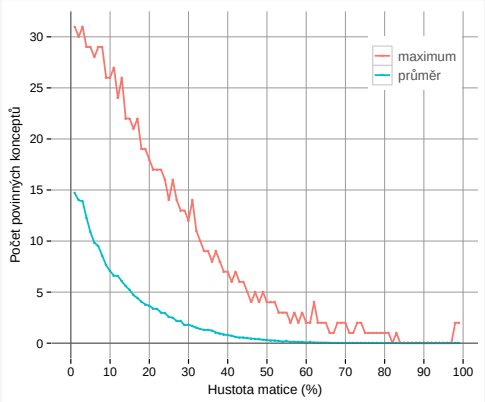
V cyklu na řádcích 10–15 probíhá výběr formálního konceptu, který obsahuje co nejvíce nepokrytých 1, proměnná $\mathcal{U}$ slouží k ukládání dosud nepokryté části, na řádcích 14–15 jsou z ní odebrány ty dvojice, které pokrývá nově vybraný formální koncept.

2.1 Algoritmy pro výpočet konceptuálního svazu

Výpočet konceptuálního svazu je v algoritmu GreCon podstatnou částí, která ovlivňuje celkovou paměťovou náročnost a také rychlost algoritmu. Všechny zde uvedené algoritmy se neliší asymptotickou složitostí, pro všechny je rovna



Obrázek 1: Počet povinných faktorů matice dimenze 64×64 v závislosti na hustotě matice.



Obrázek 2: Počet povinných faktorů matice dimenze 256×128 v závislosti na hustotě matice.

$O(|\mathcal{B}(X,Y,I)| \cdot |Y|^2 \cdot |X|)$, ale při reálných implementacích se podstatně liší rychlostí.

Jednotlivé algoritmy se výkonnostně liší v závislosti na velikosti a hustotě zdrojových dat. Podrobné porovnání některých algoritmů bylo provedeno v [13].

2.1.1 NextClosure

NextClosure [8] je obecný algoritmus pro výpočet pevných bodů uzávěrových operátorů. Lze tedy použít pro výpočet konceptuálního svazu. Algoritmus vygeneruje všechny intenty $Int(X,Y,I)$ a konceptuální svaz se poté musí dopočítat následovně:

$$\mathcal{B}(X,Y,I) = \{\langle B^{\downarrow}, B\rangle \mid B \in Int(X,Y,I)\}$$

Definice 13

Pro $A, B \subseteq Y$ a $i \in \{1,\dots n\}$ definujeme lexikografické uspořádání $<$ na množině atributů následovně:

$$A <_i B \quad \text{p.k.} \quad i \in B \setminus A \text{ a } A \cap \{1,\dots,i-1\} = B \cap \{1,\dots,i-1\}$$
$$A < B \quad \text{p.k.} \quad A <_i B \text{ pro nějaké } i$$

Definice 14

Pro $A \subseteq Y, i \in \{1,\dots,n\}$ definujeme operaci $\oplus$:

$$A \oplus i = ((A \cap \{1,\dots,i-1\}) \cup \{i\})^{\downarrow\uparrow}$$

Věta 5 (Lexikografický následník)

Nejmenší intent B^+ větší (vzhledem k $<$) než $B \subseteq Y$ je dán

$$B^+ = B \oplus i$$

kde i je největší i, které $B <_i B \oplus i$

Algoritmus 2 NextClosure

Input: Formální kontext $\langle X, Y, I \rangle$
Output: Množina intentů $Int(X, Y, I)$

1: $Int(X, Y, I) \leftarrow \emptyset$
2: $A \leftarrow \emptyset^{\downarrow\uparrow}$
3: $Int(X, Y, I) = Int(X, Y, I) \cup A$
4: **while** A!=Y **do**
5: $\quad A \leftarrow A^{+}$
6: $\quad Int(X, Y, I) \leftarrow Int(X, Y, I) \cup A$
7: **return** $Int(X, Y, I)$

Algoritmus je velmi jednoduchý na implementaci, začíná s prázdnou množinou a dopočítává lexikografické následníky, dokud neskončí s celou množinou atributů Y.

2.1.2 UpperNeighbor

UpperNeighbor [16] konstruuje konceptuální svaz spolu s uspořádáním $\leq$ (viz definice 11), je tedy vhodný pro aplikaci tam, kde je potřeba znát uspořádání konceptů. Například při vykreslování konceptuálního svazu.

Věta 6
Pokud $\langle A, B \rangle \in \mathcal{B}(X, Y, I)$ není největší koncept, pak $(A \cup \{x\})^{\uparrow\downarrow}$, $x \in X \setminus A$ je extent horního souseda p.k. pro všechna $z \in (A \cup \{x\})^{\uparrow\downarrow} \setminus A$ platí $(A \cup \{x\})^{\uparrow\downarrow} = (A \cup \{z\})^{\uparrow\downarrow}$.

Algoritmus 3 UpperNeighbor

Input: Formální kontext $\langle X, Y, I \rangle, A \subseteq X$
Output: *neighbors* množina horních sousedů A

1: $min \leftarrow X \setminus A$
2: $neighbors \leftarrow \emptyset$
3: **for** $x \in X \setminus A$ **do**
4: $\quad B_1 \leftarrow (A \cup \{x\})^{\uparrow}$
5: $\quad A_1 \leftarrow B_1^{\downarrow}$
6: $\quad$ **if** $min \cap ((A_1 \setminus A) \setminus \{x\} = \emptyset)$ **then**
7: $\quad\quad neighbors \leftarrow neighbors \cup \{\langle A_1, B_1 \rangle\}$
8: $\quad$ **else**
9: $\quad\quad min \leftarrow min \setminus \{x\}$
10: **return** *neighbors*

Pro výpočet celého konceptuální svazu se začíná od konceptu $\langle \emptyset^{\uparrow\downarrow}, \emptyset^{\uparrow} \rangle$, výsledky se zařadí do fronty a dokud je fronta neprázdná se pro každý její prvek vypočítají všichni jeho horní sousedi. Během výpočtu UpperNeighbor může

dojít k vypočtení již známých konceptů, musí se tedy kontrolovat duplikáty v konceptuálním svazu.

2.1.3 Fast CbO

Fast CbO (FCbO) [22] je algoritmus založený na Close-by-One (CbO) [14] algoritmu, který byl vylepšen tak, že byl zredukován počet formálních konceptů, které byly vypočítány vícekrát.

FCbO je jedním z nejrychlejších doposud známých algoritmů pro výpočet konceptuálního svazu, jeho popis je poměrně rozsáhlý, proto jej zde neuvádím.

2.2 Implementace

Při implementaci algoritmu GreCon jsem nejprve volil algoritmus NextClosure, který pro malé testovací vstupy postačoval. UpperNeighbor se projevil jako nevhodný, protože byl pomalejší. Nejvíce času zabrala kontrola, zda formální koncept již nebyl vygenerován.

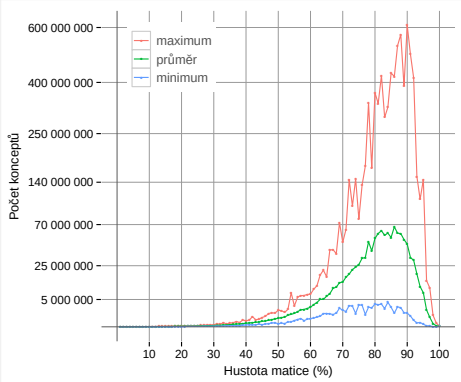
Pomocí NextClosure jsem byl schopen rozkládat i matice s více než 100 atributy s hustotou kolem 20 %. Avšak u náhodných experimentů s maticemi dimenze 256×128 s hustotou nad 20 % se začala projevovat neefektivnost tohoto algoritmu.

Proto jsem zvolil FCbO, který jsem neimplementoval, ale použil jsem volně dostupnou implementaci z [6]. Tato implementace byla řádově 200 krát rychlejší než má implementace NextClosure.

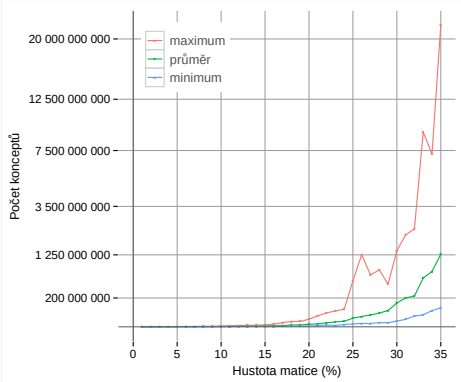
Nevýhodou algoritmu GreCon je to, že se musí vypočítat celý konceptuální svaz a ten se poté prochází. Testovací matice velikosti 256×128 s hustotou nad 25 % mohly mít konceptuální svaz, jehož velikost byla v rozmezí několika milionů konceptů až miliardy. Tak velký svaz se již nevejde do operační paměti běžných počítačů. Například se vyskytla matice, jejíž konceptuální svaz měl 926 547 510 prvků. Pro binární reprezentaci intentu potřebujeme 128 bitů, tedy 16 B, pro všechny intenty dostáváme $14\,824\,760\,160\,\mathrm{B} \approx 13,80\,\mathrm{GiB}$. Pro extenty potřebujeme dvakrát více místa. Dohromady tedy potřebujeme do paměti uložit $\approx 41,4$ GiB. Tento problém jsem řešil odkládáním na pevný disk, to se podstatně projevilo na rychlosti algoritmu a při souběhu více experimentů docházelo k přeplnění disku. Zkoušel jsem variantu s vynecháním ukládání extentů a jejich následným dopočítáním. Ta sice snížila náročnost na diskový prostor, ale nebyla výrazně rychlejší.

Nakonec se jako nejrychlejší, paměťově a diskově nenáročná, ukázala varianta, při které se konceptuální svaz vůbec neukládal a v každém průchodu cyklu z řádků 10–15 se volal algoritmus FCbO. Ukázalo se, že FCbO je při běhu v jiném vlákně schopen dodávat nové intenty rychleji, než trvá výpočet extentu a zjištění pokrytí a že je tato varianta rychlejší než načítání intentů z disku.

Na obrázcích 3 a 4 jsou znázorněny velikosti konceptuálních svazů v závislosti na hustotě. Graf pro matice dimenze 256×128 je pouze do hustoty 35 %, protože výpočet pro vyšší hustoty trval moc dlouho. Například algoritmus FCbO pro



Obrázek 3: Velikost konceptuálního svazu pro matice dimenze 64×64



Obrázek 4: Velikost konceptuálního svazu pro matice dimenze 256×128

matice s hustotou 36% trval průměrně přes dvě hodiny. Pro ilustraci na procesoru Intel Xeon E5-2680 v2 o taktu 2.80 GHz se za 24 hodin vypočítaly pouze dva konceptuální svazy matice hustoty 46 %, které měly přes 16 miliard konceptů.

3 GreConD

GreConD [3] je dalším algoritmem založeným na FCA, zkratkové slovo znamená *Greedy Concept on Demand*. Je modifikací algoritmu GreCon, upravuje strategii průchodu konceptuálního svazu tak, že jej nepotřebuje znát celý. Stejně jako GreCon vytváří rozklad zdola.

Algoritmus 4 GreConD

Input: Binární matice I
Output: Množina faktorů $\mathcal{F}$ tvořící rozklad

1: $\mathcal{U} \leftarrow \{\langle i, j\rangle | I_{ij} = 1\}$
2: $\mathcal{F} \leftarrow \emptyset$
3: **while** $\mathcal{U} \neq \emptyset$ **do**
4: $\quad D \leftarrow \emptyset$
5: $\quad V \leftarrow 0$
6: $\quad$ **while exists** $j \notin D$ **takové, že** $|D \oplus j| > V$ **do**
7: $\quad\quad$ **vyber** $j \notin D$ **které maximalizuje** $D \oplus j$:
8: $\quad\quad D \leftarrow (D \cup \{j\})^{\downarrow\uparrow}$
9: $\quad\quad V \leftarrow |(D^{\downarrow} \times D) \cap \mathcal{U}|$
10: $\quad C \leftarrow D^{\downarrow}$
11: $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \langle C, D\rangle$
12: $\quad$ **for each** $\langle i, j\rangle \in C \times D$ **do**
13: $\quad\quad \mathcal{U} \leftarrow \mathcal{U} \setminus \langle i, j\rangle$
14: **return** $\mathcal{F}$

Faktorové koncepty se budují postupným přidáváním „slibných sloupců“. Vy-

užívá se toho, že každý formální koncept $\langle C, D\rangle$ lze vyjádřit jako $D = \bigcup_{y\in D}\{y\}^{\downarrow\uparrow}$ a také toho, že pro $y \notin D$ platí, že $\langle (D \cup \{y\})^{\downarrow}, (D \cup \{y\})^{\downarrow\uparrow}\rangle$ je formální koncept a platí $D \subset (D \cup \{y\})^{\downarrow\uparrow}$.

Jakýkoliv formální koncept lze tedy vytvořit postupným přidáváním $\{y\}^{\downarrow\uparrow}$ do prázdné množiny atributů, to vede k úpravě greedy pravidla pro přidání dalšího konceptu do rozkladu. Zvolí se takové $y \in Y$, které maximalizuje

$$D \oplus y = ((D \cup \{y\})^{\downarrow} \times (D \cup \{y\})^{\downarrow\uparrow}) \cap \mathcal{U}$$

GreConD je rámcově 60 až 80 krát rychlejší než GreCon, podobně byla vylepšena i paměťová náročnost, protože se v paměti neuchovává celý konceptuální svaz, ale jen minimum nezbytných konceptů. Kvalita rozkladu úpravami nijak neutrpěla, dokonce byla vylepšena. Podrobnější srovnání naleznete v části II.

Při implementaci algoritmu GreConD nebylo potřeba volit algoritmus pro výpočet konceptuálního svazu a ani se nevyskytly žádné jiné problémy. Přímý přepis pseudokódu vedl k dobrým výsledkům.

4 GreEss

GreEss [2] je posledním vybraným algoritmem, který je založený na FCA a stejně jako předchozí algoritmy vytváří rozklad zdola. Zkratkové slovo znamená *Greedy Essential*. GreEss má kromě vstupní matice I parametr ε, který udává, jaké největší chyby může rozklad dosáhnout. Podobně lze upravit předchozí algoritmy, stačí pouze pozměnit podmínku pro ukončení cyklu.

GreEss využívá vlastností *esenciální části*, díky kterým dojde k redukci prohledávaného prostoru formálních konceptů a to vede ke kvalitnějšímu rozkladu.

Definice 15

Pro formální koncepty $\langle C_1, D_1\rangle, \langle C_2, D_2\rangle \in \mathcal{B}(X, Y, I)$ definujeme interval v $\mathcal{B}(X, Y, I)$ ohraničený $\langle C_1, D_1\rangle$ a $\langle C_2, D_2\rangle$ jako podmnožinu $\mathcal{B}(X, Y, I)$ tvaru:

$$[\langle C_1, D_1\rangle, \langle C_2, D_2\rangle] = \{\langle E, F\rangle \in \mathcal{B}(X, Y, I) \mid \langle C_1, D_1\rangle \leq \langle E, F\rangle \leq \langle C_2, D_2\rangle\}$$

Dále pro $C \subseteq X$ a $D \subseteq Y$ mějme $\gamma(C) = \langle C^{\uparrow\downarrow}, C^{\uparrow}\rangle$ a $\mu(D) = \langle D^{\downarrow}, D^{\downarrow\uparrow}\rangle$. Tedy $\gamma(C)$ je nejmenší formální koncept, jehož extent obsahuje C a $\mu(D)$ je největší formální koncept, jehož intent obsahuje D.

Definice 16

Pro $C \subseteq X$ a $D \subseteq Y$ zavádíme interval:

$$\mathcal{I}_{C,D} = [\gamma(C), \mu(D)]$$

Dále pro $i \in X, j \in Y$:

$$\mathcal{I}_{ij} = [\gamma(i), \mu(i)]$$

kde $\gamma(i)$ je zkratkou za $\gamma(\{i\})$ a $\mu(j)$ zkratkou za $\mu(\{j\})$.

Věta 7 *(a) $\mathcal{I}_{C,D}$ je neprázdný p.k. $C \times D \subseteq I$, tedy $I_{ij} = 1$ pro všechna $i \in C$ a $j \in D$. $\mathcal{I}_{ij}$ je neprázdný p.k. $I_{ij} = 1$.*

(b) $\mathcal{I}_{C,D} = \{\langle E, F\rangle \in \mathcal{B}(X, Y, I) \,|\, C \subseteq E, D \subseteq F\} = \{\langle E, F\rangle \in \mathcal{B}(X, Y, I) \,|\, C^{\uparrow\downarrow} \subseteq E, D^{\downarrow\uparrow} \subseteq F\}$. $\mathcal{I}_{ij}$ je množina všech formálních konceptů, které pokrývají $\langle i, j\rangle$.

(c) Pokud $(A_\mathcal{F} \circ B_\mathcal{F})_{ij} = 1$ pak $\mathcal{F}$ obsahuje nejméně jeden formální koncept z $\mathcal{I}_{ij}$.

Definice 17 (Esenciální část matice I)
Binární matice $J \leq I$ splňující pro každou $\mathcal{F} \in \mathcal{B}(X, Y, I)$:

$$\text{jestliže } J \leq A_\mathcal{F} \circ B_\mathcal{F}, \text{ pak } E(I, A_\mathcal{F} \circ B_\mathcal{F}) \leq \varepsilon$$

která je minimální vzhledem k $\leq$ (viz definice 4) se nazývá ε-esenciální část matice I. V případě, že $\varepsilon = 0$ nazývá se pouze esenciální část.

Definice 18
Pro binární matici I dimenze $n \times m$ značíme $\mathcal{E}(I)$ binární matici dimenze $n \times m$ definovanou jako:

$(\mathcal{E}(I))_{ij} = 1$ p.k. $\mathcal{I}_{ij}$ je neprázdný a minimální vzhledem k množinové inkluzi $\subseteq$.

Věta 8
Pro každou klarifikovanou binární matici I je $\mathcal{E}(I)$ unikátní esenciální část matice I.

Věta 9
Mějme množinu $\mathcal{G} \subseteq \mathcal{B}(X, Y, \mathcal{E}(I))$ faktorových konceptů $\mathcal{E}(I)$, tedy $\mathcal{E}(I) = A_\mathcal{G} \circ B_\mathcal{G}$. Pak každá množina $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$ obsahující pro každý $\langle C, D\rangle \in \mathcal{G}$ alespoň jeden koncept z $\mathcal{I}_{C,D}$ je množinou faktorových konceptů matice I. Tedy $I = A_\mathcal{F} \circ B_\mathcal{F}$.

Důkaz
Pro $\langle C, D\rangle \in \mathcal{G}$ označme $\langle E, F\rangle_{\langle C,D\rangle}$ koncept z $\mathcal{F} \cap \mathcal{I}_{\langle C,D\rangle}$, který dle předpokladu existuje. Z věty 7(a): $C \subseteq E$ a $D \subseteq F$, to platí pro každý $\langle C, D\rangle \in \mathcal{G}$, tedy dostáváme $A_\mathcal{G} \circ B_\mathcal{G} \leq A_\mathcal{F} \circ B_\mathcal{F}$. Z předpokladu $\mathcal{E}(I) = A_\mathcal{G} \circ B_\mathcal{G}$ dostáváme $\mathcal{E}(I) \leq A_\mathcal{F} \circ B_\mathcal{F}$. Jelikož je $\mathcal{E}(I)$ esenciální část matice I, tak $I = A_\mathcal{F} \circ B_\mathcal{F}$. □

Důsledkem věty 9 je to, že pro nalezení rozkladu matice I stačí vypočítat rozklad matice $\mathcal{E}(I)$ a poté pro každý faktor $\langle C, D\rangle$ matice $\mathcal{E}(I)$ vypočítat $\mathcal{I}_{C,D}$ a z něj vybrat jen jeden formální koncept. Výhodou je to, že $\mathcal{E}(I)$ je podstatně řidší než I, a tedy její rozklad je jednodušší.

Věta 10
Pro každou binární matici I platí $rank_B(I) \leq rank_B(\mathcal{E}(I))$.

Důkaz

Nechť $k = rank_B(\mathcal{E}(I))$ a A, B jsou binární matice dimenze $n \times k$ a $k \times m$, pro které $\mathcal{E}(I) = A \circ B$. Uvažujme $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$, která pro každý $\langle C, D\rangle \in \mathcal{G}$ obsahuje právě jeden formální koncept z intervalu $\mathcal{I}_{C,D}$ v $\mathcal{B}(X, Y, I)$. Takové $\mathcal{F}$ existuje, protože pro každý formální koncept $\langle C, D\rangle \in \mathcal{G}$ platí $C \times D \subseteq \mathcal{E}(I)$, protože $\langle C, D\rangle$ je formálním konceptem $\mathcal{E}(I)$, a tedy z $\mathcal{E}(I) \leq I$ vyplývá $C \times D \subseteq I$. Z věty 7 (a) je $\mathcal{I}_{C,D}$ neprázdným intervalem v $\mathcal{B}(X, Y, I)$. Dle předchozí věty $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$, a tedy $rank_B(I) \leq |\mathcal{F}| \leq \mathcal{G} = rank_B(\mathcal{E}(I)$.

□

Věta 11

Pro každou binární matici I existuje množina $\mathcal{F} \subseteq \mathcal{B}(X, Y, \mathcal{E}(I))$ t.ž. $|\mathcal{F}| = rank_B(I)$, pro kterou $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Důkaz

Dle věty 3 existuje $\mathcal{F} \subseteq \mathcal{B}(X, Y, I)$, t.ž. $|\mathcal{F}| = rank_B(I)$, pro kterou platí $I = A_{\mathcal{F}} \circ B_{\mathcal{F}}$. Zbývá ukázat, že $\mathcal{F} \subseteq \mathcal{B}(X, Y, \mathcal{E}(I))$. Sporem předpokládejme, že existuje formální koncept $\langle C, D\rangle \in \mathcal{F} \setminus \mathcal{B}(X, Y, \mathcal{E}(I))$, tedy že $\langle C, D\rangle$ nepatří do žádného minimálního $\mathcal{I}_{ij}$. $\mathcal{F}$ pokrývá I a také $\mathcal{E}(I)$. Pro každé $\langle i', j'\rangle$, které $\mathcal{E}(I)_{i'j'} = 1$ existuje formální koncept $\langle C', D'\rangle \in \mathcal{F}$, který pokrývá $\langle i', j'\rangle$. Z věty 7 (b): $\langle C', D'\rangle \in \mathcal{I}_{i'j'}$, a tedy $\langle C', D'\rangle \neq \langle C, D\rangle$. Pro $\mathcal{F}' = \mathcal{F} \setminus \{\langle C, D\rangle\}$ tedy máme $A_{\mathcal{F}'} \circ B_{\mathcal{F}'} \geq \mathcal{E}(I)$ a tedy $A_{\mathcal{F}'} \circ B_{\mathcal{F}'} = I$ dle věty 8. Dostali jsme množinu faktorů $\mathcal{F}'$ matice I takovou, že $|\mathcal{F}'| = |\mathcal{F}| - 1 < rank_B$, což je spor s předpokladem. □

Výpočet $\mathcal{E}(I)$ je jednoduchý. $\mathcal{E}(I)_{ij} = 1$ p.k. jsou splněny všechny následující podmínky:

(a) $I_{ij} = 1$

(b) pro každé i' takové, že $\{i'\}^{\uparrow} \subset \{i\}^{\uparrow}$ platí: $I_{i'j} = 0$. Tedy žádné i', jehož řádek je obsažen v řádku i, neobsahuje j.

(c) pro každé j' takové, že $\{i'\}^{\downarrow} \subset \{i\}^{\downarrow}$ platí: $I_{ij'} = 0$. Tedy žádné j', jehož sloupce je obsažen ve sloupci j, neobsahuje i.

Algoritmus GreEss však nevybírá $\mathcal{G} \subseteq \mathcal{B}(X, Y, \mathcal{E}(I))$, která by rozkládala matici $\mathcal{E}(I)$, ale vybírá $\mathcal{G} \subseteq \mathcal{B}(X, Y, \mathcal{E}(I))$, která nemusí rozkládat $\mathcal{E}(I)$, může být menší, ale přesto splňuje důležitou vlastnost: pokud pro každý formální koncept $\langle C, D\rangle \in \mathcal{G}$ vybereme přesně jeden koncept $c_{\langle C,D\rangle}$ z intervalu $\mathcal{I}_{C,D}$, pak bez ohledu na to, který koncept vybereme, výsledná $\mathcal{F} = \{c_{\langle C,D\rangle} \mid \langle C, D\rangle \in \mathcal{G}\}$ rozkládá I, tedy $A_{\mathcal{F}} \circ B_{\mathcal{F}}$.

Množina $\mathcal{G}$ je vypočítána funkcí ComputeIntervals na prvním řádku algoritmu. V cyklu z řádků 4–21 je pro každé $\langle C, D\rangle \in \mathcal{G}$ vybrán právě jeden koncept $\langle E', F'\rangle$ z intervalu $\mathcal{I}_{C,D}$. Na řádcích 6–18 probíhá tento výběr následovně: pro každý dosud nepoužitý $\langle C, D\rangle \in \mathcal{F}$ se začíná s atributovými koncepty $\gamma(j) \in \mathcal{I}_{C,D}$ a greedy způsobem se zkouší jejich rozšíření. Vybrán je ten koncept $\langle E, F\rangle$, který

pokrývá nejvíce jedniček v U (řádek 10–11). Do $\mathcal{F}$ se přidá formální koncept $\langle E', F' \rangle$, jež pokrývá nejvíce jedniček v U ze všech kandidátů $\langle E, F \rangle$ (řádky 15-18). Koncept $\langle C', D' \rangle$, z jehož intervalu $\mathcal{I}_{C',D'}$ pochází $\langle E', F' \rangle$, je odebrán z $\mathcal{G}$ (řádek 20) a není již dále uvažován. Na řádku 21 probíhá aktualizace nepokryté části.

Algoritmus 5 GreEss

Input: Binární matice I, $\varepsilon \geq 0$
Output: Množina faktorů $\mathcal{F}$, pro které $||I - A_{\mathcal{F}} \circ B_{\mathcal{F}}|| \leq \varepsilon$

1: $\mathcal{G} \leftarrow$ ComputeIntervals(I)
2: $U \leftarrow \{\langle i, j \rangle | I_{ij} = 1\}$
3: $\mathcal{F} \leftarrow \emptyset$
4: **while** $|U| > \varepsilon$ **do**
5: $\quad s \leftarrow 0$
6: $\quad$ **for each** $\langle C, D \rangle \in \mathcal{G}$ **do**
7: $\quad\quad J \leftarrow I \cap (D^{\downarrow_I} \times C^{\uparrow_I})$
8: $\quad\quad F \leftarrow \emptyset$
9: $\quad\quad s_{\langle C,D \rangle} \leftarrow 0$
10: $\quad\quad$ **while** existuje $j \in C^{\uparrow_I} - F$ t.ž. $|(F \cup \{j\})^{\downarrow_J} \times (F \cup \{j\})^{\downarrow_J \uparrow_J} \cap U| > s_{\langle C,D \rangle}$ **do**
11: $\quad\quad\quad$ **vyber** j **které maximalizuje** $|(F \cup \{j\})^{\downarrow_J} \times (F \cup \{j\})^{\downarrow_J \uparrow_J} \cap U|$:
12: $\quad\quad\quad F \leftarrow (F \cup \{j\})^{\downarrow_J \uparrow_J}$
13: $\quad\quad\quad E \leftarrow (F \cup \{j\})^{\downarrow_J}$
14: $\quad\quad\quad s_{\langle C,D \rangle} \leftarrow |E \times F \cap U|$
15: $\quad\quad$ **if** $s_{\langle C,D \rangle} > s$ **then**
16: $\quad\quad\quad \langle E', F' \rangle \leftarrow \langle E, F \rangle$
17: $\quad\quad\quad \langle C', D' \rangle \leftarrow \langle C, D \rangle$
18: $\quad\quad\quad s \leftarrow s_{\langle C,D \rangle}$
19: $\quad \mathcal{F} \leftarrow \mathcal{F} \cup \langle E', F' \rangle$
20: $\quad \mathcal{G} \leftarrow \mathcal{G} \setminus \langle C', D' \rangle$
21: $\quad U \leftarrow U \setminus E' \times F'$
return $\mathcal{F}$

Funkce ComputeIntervals nejprve vypočítá esenciální část $\mathcal{E}(I)$, poté podobně jako v algoritmu GreEss vypočítává koncepty $\mathcal{G} \in \mathcal{B}(X, Y, \mathcal{E}(I))$ s tím rozdílem, že jedničky v U jsou pokryty kandidátem $\langle C, D \rangle$ pokud náleží do $C^{\uparrow_I \downarrow_I} \times D^{\downarrow_I \uparrow_I}$ a ne pouze ty z $C \times D$. To je možné, protože faktory I jsou v GreEss vybírány z intervalů $\mathcal{I}_{C,D}$ a kvůli větě 7 (b) každý takový faktor pokrývá $C^{\uparrow_I \downarrow_I} \times D^{\downarrow_I \uparrow_I}$. Z těchto důvodů nemusí $\mathcal{G}$ rozkládat $\mathcal{E}(I)$, ale může být menší.

Implementačně je GreEss nenáročný, podobně jako GreConD není příliš náročný na operační paměť (opět se nepočítá celý konceptuální svaz). Za zmínku stojí jen to, že funkce ComputeIntervals trvá téměř polovinu běhu celého algoritmu a také to, že stejně jako předchozí algoritmy, je GreEss velmi závislý na rychlosti implementace šipkových operátorů $^{\downarrow\uparrow}$, které se v algoritmu hojně používají.

Algoritmus 6 ComputeIntervals

Input: Binární matice I
Output: Množina $\mathcal{G} \subseteq \mathcal{B}(\mathcal{E}(I))$

```
1: E ← E(I)
2: U ← {⟨i, j⟩|E_ij = 1}
3: G ← ∅
4: while U ≠ ∅ do
5:     D ← ∅
6:     s ← 0
7:     while exists j ∉ D with |((D ∪ {j})^↓ε)^↑I↓I × ((D ∪ {j})^↓ε↑ε)^↓I↑I ∩ U| > s do
8:         vyber j které maximalizuje |((D ∪ {j})^↓ε)^↑I↓I × ((D ∪ {j})^↓ε↑ε)^↓I↑I ∩ U|:
9:         D ← (D ∪ {j})^↓ε↑ε
10:        C ← (D ∪ {j})^↓ε
11:    G ← G ∪ ⟨C, D⟩
12:    U ← U \ C^↑I↓I × D^↓I↑I
   return G
```

5 Asso

Algoritmus Asso [18] je prvním vybraným algoritmem, který nemusí produkovat přesný rozklad matice. Avšak výsledek lze výrazně ovlivnit nastavením vstupních parametrů. Je navržen pro *Discrete basis problém* (DBP), snaží se minimalizovat chybu rozkladu při pevně daném prostředním rozměru rozkladu k.

Problém 2 (Discrete basis problém)**.** *Pro danou binární matici I dimenze $n \times m$ a kladné celé číslo $k < \min(n, m)$ nalézt binární matice S dimenze $n \times k$ a B dimenze $k \times m$, které minimalizují:*

$$|I - S \circ B| = \sum_{i=1}^{n} \sum_{j=1}^{m} |I_{ij} - (S \circ B)_{ij}|$$

Věta 12
DBP je NP-těžký problém a rozhodovací verze DBP je NP-úplný problém.

Důkaz
Redukcí SBP. □

Asso je založeno na asociacích mezi sloupci matice. Míra jistoty asociace mezi sloupci i a j v matici I je definována jako při získávání asociačních pravidel z dat [1] následovně:

$$c(i \Rightarrow j) = \frac{\langle I_{\_i}, I_{\_j} \rangle}{\langle I_{\_i}, I_{\_i} \rangle}$$

kde $\langle \cdot, \cdot \rangle$ značí skalární součin vektorů. Asociace mezi sloupci i a j je τ-silná, jestliže $c(i \Rightarrow j) \geq \tau$.

V algoritmu se nejprve buduje *asociační matice* A, ve které řádek $A_{i\_}$ obsahuje jedničky ve sloupcích j, pro které platí: $c(i \Rightarrow j) \geq \tau$. Řádky asociační matice A jsou kandidáti na řádky matice B. τ je vstupní parametr algoritmu,

Algoritmus 7 Asso

Input: Binární matice I, $k \in \mathbb{N}, k < min\{n, m\}$ práh $\tau \in [0, 1]$ váhy w^+, w^-
Output: Matice $B \in \{0, 1\}^{k \times m}$ a $S \in \{0, 1\}^{n \times k}$

1: $A \leftarrow 0^{m \times m}$
2: **for** $i = 1, \ldots, m$ **do**
3: **for each** $j : c(i \Rightarrow j) \geq \tau$ **do**
4: $A_{ij} = 1$
5: $B \leftarrow 0^{k \times m}$
6: $S \leftarrow 0^{n \times k}$
7: **for** $l = 1, \ldots, k$ **do**
8: $(A_{i\_}, s) \leftarrow \max_{A_{i\_}, s \in \{0,1\}^{n \times 1}} cover(\begin{pmatrix} B \\ A_{i\_} \end{pmatrix}, \begin{pmatrix} S & s \end{pmatrix}, I, w^+, w^-)$
9: $B \leftarrow \begin{pmatrix} B \\ A_{i\_} \end{pmatrix}$
10: $S \leftarrow \begin{pmatrix} S & s \end{pmatrix}$
11: **return** B, S

určuje minimální míru jistoty mezi sloupci. Vybírá se ten řádek asociační matice A, který maximalizuje funkci *cover*:

$$cover(B, S, I, w^+, w^-) = w^+|\{(i, j) : I_{ij} = 1, (S \circ B)_{ij} = 1\}| \\ -w^-|\{(i, j) : I_{ij} = 0, (S \circ B)_{ij} = 1\}|$$

V [18] je tento výběr popsán jen vágně, přesný způsob výběru musel být zjištěn až z implementace autora. Při výpočtu maxima funkce *cover* (řádek 8) dochází zároveň k výpočtu sloupcového vektoru s. Pro každý řádek vstupní matice I je zvlášť vypočtena funkce *cover* a $s_j = 1$ p.k. $cover > 0$ pro řádek $I_{j\_}$.

5.1 Vliv parametrů

Na rozdíl od předchozích algoritmů vyžaduje algoritmus Asso vstupní parametry, kterými lze ovlivnit vlastnosti výsledného rozkladu. V následujících podkapitolách je popsáno, jak parametry ovlivňují výsledek.

5.1.1 Prostřední rozměr k

Parametr k udává maximální prostřední rozměr výsledného rozkladu. Může se stát, že Asso vypočítá rozklad s nižším prostředním rozměrem, než je k. Při testování se například stalo, že algoritmus vypočítal rozklad s polovičním prostředním rozměrem než k. Formálně však algoritmus vždy dané k dodrží, do rozkladu přidává prázdné faktory. Ty můžeme z výsledku vyřadit.

5.1.2 Práh τ

Funkce parametru τ je zřejmá, určuje, jak bude vypadat asociační matice A. Avšak vliv parametru τ na výsledný rozklad mi zřejmý nepřipadal. Provedl jsem proto testy nad náhodnými maticemi s dimenzemi 256×128 s různými hustotami. Výsledky testů jsou znázorněny v grafech na obrázcích 5,6,7 a 8. Z grafů můžeme pozorovat například to, že $\tau = 0.1$ produkuje velmi malé rozklady, ale s poměrně velkou chybou. Ale $\tau = 0.9$ dává velké rozklady a přitom chyba není nejmenší. Hodnoty τ okolo 0.8 produkují „vyvážené“ rozklady s ohledem na velikost a chybu. K obrázku 6 je třeba zdůraznit to, že osa y je v procentech překrytých nul, a že nul s rostoucí hustotou ubývá. Z obrázku 8 soudím, že velikost výsledného rozkladu přímo závisí na parametru τ. Všechny algoritmy byly pouštěny s parametrem $k = 32$, což je i prostřední rozměr matic při generování. Viz kapitola 11.1.

Všechny hodnoty jsou získány průměrem z 1000 matic pro každé procento hustoty a s parametry $w^+ = 1$ a $w^- = 1$ tak, že nejprve byly vygenerovány matice s danou hustotou a poté na nich byly spuštěny algoritmy s rozdílným parametrem τ. Procento chyby značí procento odlišných pozic ve vynásobeném rozkladu oproti původní matici.

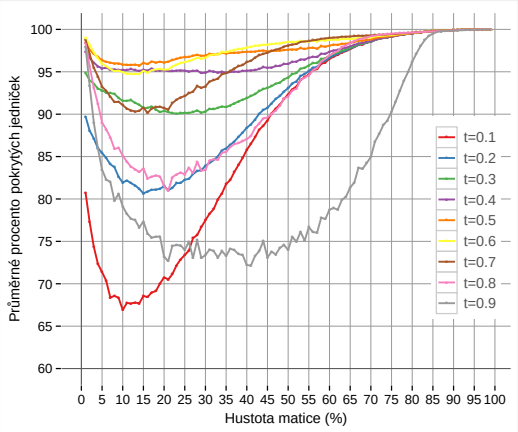
5.1.3 Váhy w^+, w^-

Parametry w^+ a w^- slouží k nastavení vah funkce *cover*. w^+ určuje bonifikaci pokrytí a w^- penalizuje překrytí. Před implementací jsem se domníval, že například desetinásobné zvýšení w^- oproti w^+ desetkrát sníží počet překrytí, ale není tomu tak. Počet překrytí je sice nižší, ale ne desetinásobně.

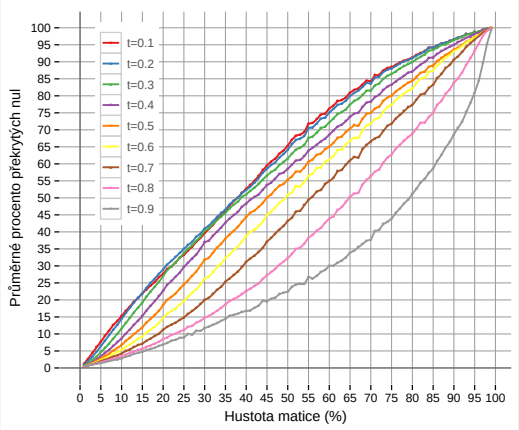
Hlavní roli hraje poměr mezi w^+ a w^-, Asso vypočítá shodný rozklad s parametry $w^+ = 1, w^- = 10$ jako s parametry $w^+ = 10, w^- = 100$. Je tedy možné jeden z parametrů zafixovat na 1 a udávat pouze druhý parametr větší než 0 a tím určovat poměr.

5.2 Implementace

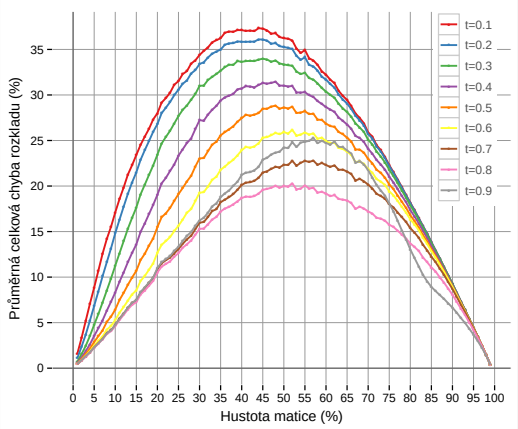
Nejkritičtější částí implementace bylo zjištění, jak vlastně funguje funkce *cover* a jak se získává vektor s na 8. řádku algoritmu. Ve článku není tento výpočet téměř vůbec popsán a než jsem jej naimplementoval správně, prošel jsem několika nesprávnými verzemi, které také „nějak fungovaly“. Velkou výhodou algoritmu Asso je jeho rychlost a paměťová nenáročnost, která značně ulehčila odladění implementace.



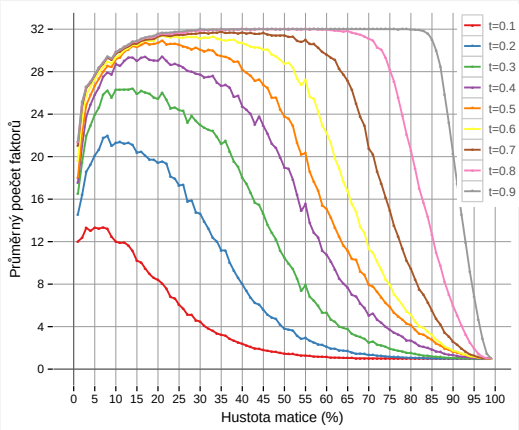
Obrázek 5: Průměrné procento pokrytých jedniček v závislosti na hustotě matic a parametru τ.



Obrázek 6: Průměrné procento překrytých nul v závislosti na hustotě matic a parametru τ.



Obrázek 7: Průměrná celková chyba rozkladu v procentech v závislosti na hustotě matic a parametru τ.



Obrázek 8: Průměrný počet faktorů v závislosti na hustotě matic a parametru τ.

6 PaNDa

Algoritmus PaNDa [17] je dalším algoritmem, který nemusí produkovat přesný rozklad matice. Název je zkratkou za *Patterns in Noisy Datasets*. Z názvu je zřejmé, že se algoritmus snaží potlačit určitou míru binárního šumu v matici. Primárně je navrhnut pro řešení *Top-k pattern discovery problému* a pro jeho vysvětlení je potřeba nadefinovat několik pojmů.

Definice 19

Vzorem (pattern) P v matici D je dvojice $P = \langle P_I, P_T \rangle$, kde P_I je množina atributů a P_T je množina objektů.

Definice 20

Množina Π vzorů matice D dimenze $n \times m$ je vzorovým modelem matice D jestliže

$$D = \bigvee_{P \in \Pi} (char(P_T) \cdot char(P_I)^T) \oplus \mathcal{N}$$

kde funkce $char(\cdot)$ značí charakteristický vektor množin a $\oplus$ značí operaci XOR po složkách a $\mathcal{N}$ je matice dimenze $n \times m$, která modeluje binární šum, tedy náhodné prohození jedniček a nul.

Problém 3 (Top-k pattern discovery problém)**.** *Pro binární matici D dimenze $n \times m$ a celé číslo k naleznout množinu vzorů Π takovou, aby $|\Pi| \leq k$ a aby Π byla vzorovým modelem D a minimalizovala cenu:*

$$\gamma(\Pi, D) = \sum_{P \in \Pi} \gamma_P(P_T, P_I) + \gamma_{\mathcal{N}}(\mathcal{N})$$

kde $\gamma_P(x, y) = |x| + |y|$ a $\gamma_{\mathcal{N}}(z) = ||z||_F$. Kde $|| \cdot ||_F$ značí Frobeniovu normu, tedy v binárním případě počet jedniček v matici.

Algoritmus 8 PaNDa

Input: Binární matice C, $k \in \mathbb{N}, k < min\{n, m\}$
Output: Množina vzorů Π

1: $\Pi \leftarrow \emptyset$
2: $\mathcal{D}_R \leftarrow \mathcal{D}$
3: **for** $iter \leftarrow 1, \ldots, k$ **do**
4: $\quad C, E \leftarrow$Find-Core$(\mathcal{D}_R, \Pi, \mathcal{D})$
5: $\quad C^+ \leftarrow$ Extend-Core(C, E, Π, D)
6: $\quad$ **if** $\gamma(\Pi, \mathcal{D}) < \gamma(\Pi \cup C^+, \mathcal{D})$ **then**
7: $\quad\quad$ **break**
8: $\quad \Pi \leftarrow \Pi \cup C^+$
9: $\quad \mathcal{D}_R(i, j) \leftarrow 0 \;\; \forall i, j$ t.ž. $C_T^+(i) = 1 \wedge C_I^+(j) = 1$
10: **return** Π

Prostor řešení problému 3 je obrovský – pro matici dimenze $n \times m$ existuje $2^n \cdot 2^m$ kandidátů na vzory. PANDA tento prostor redukuje pomocí dvou heuristik. První rozkládá problém nalezení vzoru na dva jednodušší problémy: na nalezení *jádra vzoru* [11] a jeho rozšíření. Druhá seřadí atributy a prochází je v daném pořadí bez backtrackingu namísto toho, aby se testovalo všech 2^m kombinací.

Na druhém řádku algoritmu se zkopíruje původní matice do matice $\mathcal{D}_\mathcal{R}$, která slouží k evidenci, jaká část matice již byla pokryta. Na řádcích 3–9 probíhá hlavní cyklus, který končí po nalezení k vzorů nebo jestliže přidáním C^+ do výsledku dojde ke zhoršení (řádek 6 a 7). Na řádcích 4 a 5 je prováděna první heuristika. Nejprve je nalezeno jádro vzoru pomocí procedury FIND-CORE a to je poté rozšířeno procedurou EXTEND-CORE. Vybraný vzor je přidán do výsledku (řádek 8) a poté jsou v matici $\mathcal{D}_\mathcal{R}$ vynulovány pokryté jedničky (řádek 9). Procedura

Algoritmus 9 FIND-CORE

Input: Binární matice $\mathcal{D}_R, \mathcal{D}$, množina vzorů Π
Output: E, C

1: $E \leftarrow \emptyset$
2: $S = \{s_1, \ldots, s_M\} \leftarrow$ SORT-ITEMS-IN-DB($\mathcal{D}_R$)
3: $C \leftarrow \langle C_T = 0^N, C_I = 0^M \rangle$
4: $C_I(s_1) \leftarrow 1$
5: $C_T(i) \leftarrow 1 \ \forall i$ t.ž. $\mathcal{D}_R(i, s_1) = 1$
6: **for each** $h \leftarrow 2, \ldots, M$ **do**
7: $\quad C^* \leftarrow C$
8: $\quad C^*_I(s_h) \leftarrow 1$
9: $\quad C^*_T \leftarrow 0 \ \forall i$ t.ž. $\mathcal{D}_R(i, s_h) = 0$
10: $\quad$ **if** $\gamma(\Pi \cup C^*, \mathcal{D}) \leq \gamma(\Pi \cup C, \mathcal{D})$ **then**
11: $\quad\quad C \leftarrow C^*$
12: $\quad$ **else**
13: $\quad\quad E$.append(s_h)
14: **return** C, E

FIND-CORE využívá druhou heuristiku. Na řádku 2 dochází k seřazení atributů. V pseudokódu není uvedeno, jak atributy seřadit. Podrobnosti o řazení a jeho vliv na výsledek naleznete v [29]. V implementaci jsem volil řazení podle frekvence výskytu, která se mi jevila jako nejjasnější a dle článku podává dobré výsledky a je rychlá.

Na 3. řádku se vytváří prázdný kandidát C, do kterého se přidá atribut s_1, který je po seřazení první. A přidají se ty objekty mající atribut s_1. V cyklu na řádcích 6–13 jsou postupně testovány zbylé atributy. Pokud přidáním atributu s_h do kandidáta a odebráním objektů, které atribut s_h nemají dojde ke snížení funkce γ, je daný atribut vybrán. Pokud ke zlepšení nedojde, přidá se atribut s_h do fronty E, která se využívá v proceduře EXTEND-CORE.

Hlavní cyklus procedury EXTEND-CORE probíhá přes všechny atributy e z fronty E, která vznikla na 14. řádku procedury FIND-CORE. Na řádcích 5–8 se

Algoritmus 10 EXTEND-CORE

Input: Binární matice $\mathcal{D}$, množina vzorů Π, C, E
Output: C

```
1: while E ≠ ∅ do
2:     e ← E.pop()
3:     C* ← C
4:     C*_I(e) ← 1
5:     if γ(Π ∪ C*, D) ≤ γ(Π ∪ C, D) then
6:         C ← C*
7:     for i ∈ {1, ..., N} t.ž. C*_T(i) = 0 do
8:         C* ← C
9:         C*_T(i) ← 1
10:        if γ(Π ∪ C*, D) ≤ γ(Π ∪ C, D) then
11:            C ← C*
12: return C
```

zkouší, zda přidáním atributu e do C nedojde ke snížení γ. Pokud ano, přidá se atribut e do C. Poté následuje podobné testování pro objekty, které ještě nejsou zahrnuty v C. Dojde-li ke snížení γ, přidá se daný objekt do C.

Výsledný rozklad matice je z Π zkonstruován obdobně jako v případě FCA algoritmů z množiny formálních konceptů.

6.1 Implementace

Implementace algoritmu PANDA byla přímočará. Pseudokódy neobsahují žádné nejasné části. Algoritmus je poměrně rychlý a nemá vysokou paměťovou náročnost. V algoritmu je často volána funkce γ, převážně v proceduře EXTEND-CORE v případech, kdy má matice mnoho řádků. Efektivnost funkce γ značně ovlivňuje celkovou rychlost algoritmu. Pro řazení atributů dle frekvence jsem použil algoritmus *Quicksort*, konkrétně *qsort* [26] z *GNU libc*.

7 Hyper

Algoritmus Hyper [29] produkuje přesný rozklad matice, nesnaží se však minimalizovat prostřední rozměr k jako například FCA algoritmy. Obdélník v matici plný jedniček se nazývá *hyperobdélník* a je definován podobně jako vzor u algoritmu PaNDa nebo dlaždice u TilinGu.

Definice 21

Hyperobdélník v matici DB je kartézský součin množiny objektů T a množiny atributů I matice DB. Tedy $H = T \times I = \{(i, j) \mid i \in T \text{ a } j \in I\}$

Mějme množinu hyperobdélníků $CDB = \{H_1, \ldots, H_p\}$, množina buněk matice DB, které pokrývá CDB se značí CDB^c a je rovna $\bigcup_i^p H_i$

Cena hyperobdélníku $H = T \times I$ se značí $cost(H)$ a je definována jako součet velikostí množin T a I, tedy $cost(H) = |T| + |I|$.

Definice 22

Je-li matice DB podmnožinou CDB^c, pak se CDB nazývá pokrytím či sumarizací matice DB. Cenu pokrytí získáme součtem cen dílčích hyperobdélníků. $cost(CDB) = \sum_{i=1}^{p} cost(H_i) = \sum_{i=1}^{p} |T_i| + |I_i|$

Algoritmus Hyper se snaží nalézt sumarizaci CDB matice DB bez překrytí nul, která má nejmenší cenu $cost(CDB)$. Na vstup dostává matici DB a množinu $\mathcal{C}_\alpha^- = \{T(I_i) \times I_i : I_i \in \mathcal{F}_\alpha \cup \mathcal{I}\}$ kde $T(\cdot)$ značí všechny objekty mající všechny atributy z I_I (tedy totéž co FCA šipkový operátor $^\downarrow$), $\mathcal{I}$ je množina všech atributů a $\mathcal{F}_\alpha$ je takzvaná *frequent itemset*, což je množina všech $I \subseteq \mathcal{I}$, které má alespoň $\alpha \cdot n$ objektů, kde n je počet řádků matice DB.

Algoritmus 11 Hyper

Input: Binární matice DB, $\mathcal{C}_\alpha^-$
Output: Pokrytí matice CDB

1: $R \leftarrow \emptyset$
2: $CDB \leftarrow \emptyset$
3: **while** $R \neq DB$ **do**
4: **for each** $H_i \in \mathcal{C}_\alpha^-$ **do**
5: $X_i \leftarrow$FindHyper(H_i, R)
6: $H' \leftarrow arg\min_{X_i} \gamma(X_i)$
7: $CDB \leftarrow CDB \cup \{H'\}$
8: $R \leftarrow R \cup H'$
9: **return** CDB

Kostra algoritmu je jednoduchá. Dokud se nepokryje celá matice DB, zavolá se pro všechny H_i z $\mathcal{C}_\alpha^-$ procedura FindHyper (řádky 4–6) a vybere se to H'

(řádek 6), které má nejmenší cenu dle funkce γ, jež je definována:

$$\gamma(H) = \frac{|T| + |I|}{|T \times I \setminus R|}$$

tedy poměr ceny hyperobdélníku a počtu nově pokrytých jedniček. Na řádku 7 je vybrané H' přidáno k budoucímu pokrytí CDB a na řádku 8 se aktualizuje pomocná matice R, do které se ukládá již pokrytá část.

Hlavní část algoritmu se odehrává v proceduře FindHyper, kde se hyperobdélník H rozdělí na jednoobjektové podobdélníky a pro ty je zvlášť spočítáno, kolik pokrývají dosud nepokrytých jedniček v matici DB (řádky 1–3). Poté jsou podle výsledku seřazeny (řádek 3) a první je přidán výsledku H'. V cyklu na řádcích 5–10 jsou postupně podobdélníky procházeny a zkoumá se, zda přidáním nového podobdélníku S vzroste hodnota funkce γ či nikoliv. Pokud vzroste, cyklus se přeruší a procedura vrátí H'. Pokud ne, přidá se S do H'. Procedura díky přidání prvního podobdélníku (řádek 4) vždy vrátí nějakou část původního H.

Algoritmus 12 FindHyper

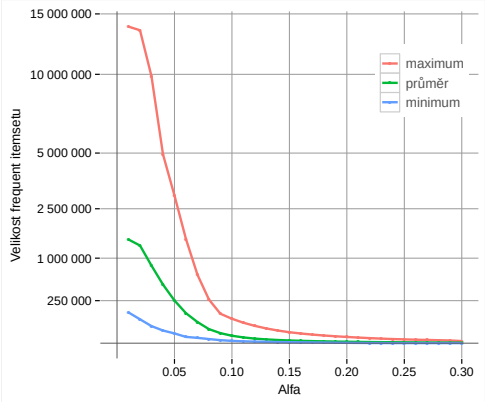
Input: $H = T(I_i) \times I_i, R$
Output: $H' = T_i \times I_i, T_i \subseteq T(I_i)$
1: **for each** $\mathcal{S} = \{t_j\} \times I_i \subseteq H$ **do**
2: spočítej počet nepokrytých prvků DB v $\mathcal{S}, |\mathcal{S} \setminus R|$
3: seřaď $\mathcal{S}$ podle $|\mathcal{S} \setminus R|$ ulož jej do U
4: $H' \leftarrow U$.pop()
5: **while** $U \neq \emptyset$ **do**
6: $\mathcal{S} \leftarrow U$.pop()
7: **if** přidáním $\mathcal{S}$ do H' zvýšíme $\gamma(H')$ **then**
8: **break**
9: **else**
10: $H' \leftarrow H' \cup \mathcal{S}$
11: **return** H'

7.1 Parametr α

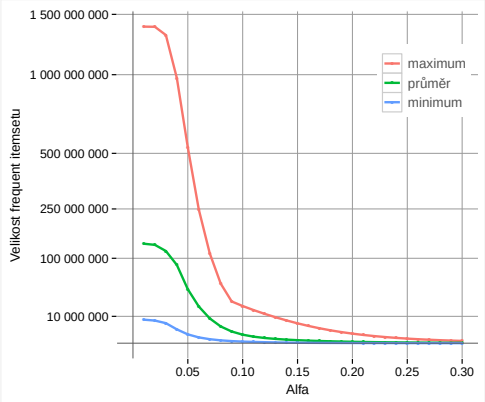
Parametr α výrazně omezuje velikost množiny $\mathcal{F}_\alpha$ a tedy i množiny $\mathcal{C}_\alpha^-$ a tím i ovlivňuje paměťovou náročnost algoritmu a rychlost. Má však velký vliv na kvalitu rozkladu.

Na obrázcích 9 a 10 je znázorněna velikost $\mathcal{F}_\alpha$ v závislosti na α pro matice dimenze 256×128 s hustotou 20 a 30 %. Je vidět, že velikost $\mathcal{F}_\alpha$ s rostoucím α značně klesá.

Z výsledků vyplývá, že algoritmus s nižším parametrem α produkuje rozklady, které mají většinou větší první faktory než rozklady s vyšším parametrem α. Avšak nižší α kupodivu průměrně produkuje rozklad na více faktorů než vyšší α.



Obrázek 9: Velikost frequent itemsetu pro matice dimenze 256 × 128 s hustotou 20 % v závislosti na parametru α



Obrázek 10: Velikost frequent itemsetu pro matice dimenze 256 × 128 s hustotou 30 % v závislosti na parametru α

To je dáno tím, že jedničky u objektů, které nemají všechny atributy z prvních faktorů, po přidání velkých faktorů pokrývají pouze jednoatributové faktory, které jsou přidávány převážně v závěru dekompozice. Celkově algoritmus Hyper má tendenci rozkládat matici na faktory, které mají málo atributů a často dochází k velkým rozkladům, kde jsou faktory jednoatributové.

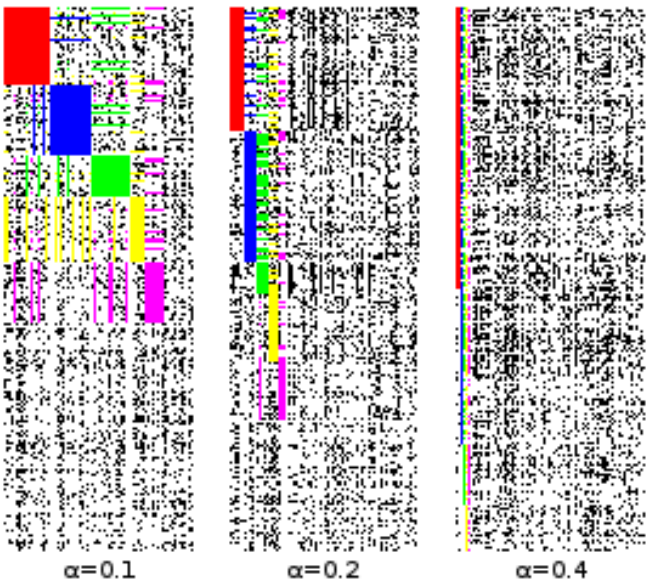
Na obrázku 11 je zobrazeno prvních 5 faktorů z rozkladu algoritmem Hyper, kde lze pozorovat vliv parametru α. Byla použita matice dimenze 256 × 128 s hustotou 26 %, zobrazeny jsou pouze výřezy. Vysvětlení zobrazení je v kapitole 9.

7.2 Implementace

Implementace algoritmu Hyper skýtala podobné problémy jako implementace algoritmu GreCon. Rychlost algoritmu značně ovlivňuje volba algoritmu pro výpočet $\mathcal{F}_\alpha$. Autoři v článku zmiňují, že předpokládají použití známého Apriori algoritmu, který v běhu mé implementace zabral podstatnou část času. Nahradil jsem jej algoritmem FCbO, jehož implementace [6] umožňuje použít parametr α. Náhrada je možná, protože i autoři algoritmu uvádí, že při použití uzavřených frequent itemsetů dosahuje algoritmus Hyper lepších výsledků.

Paměťová náročnost je ovlivněna parametrem α. Při použití $\alpha > 0$ bude vždy nižší než u algoritmu GreCon, tím je umožněno použít Hyper i pro matice s vyšší hustotou. I přesto praktická použitelnost pro $\alpha = 0.1$ končila u matic dimenze 256 × 128 s hustotou okolo 35 %.

Za zmínku ještě stojí to, že se velmi často volá řadící algoritmus. V každé iteraci hlavního cyklu se řadící algoritmus volá pro každý hyperobdélník H_i a musí seřadit všechny jeho jednoobjektové podobdélníky. Tedy i volba řadícího algoritmu má velký vliv na celkovou rychlost algoritmu.



Obrázek 11: Pokrytí prvních 5 faktorů algoritmu HYPER dle parametru α.

8 Tiling

Posledním vybraným algoritmem je TILING, což je upravená verze algoritmu LTM(*Large Tile Mining*) [9] pro rozklad binární matice. Název TILING je spíše pracovní název, který se však ujal, a proto jej budu používat.

8.1 Algoritmus LTM

Pro popis algoritmu LTM je potřeba nadefinovat základní pojmy, které jsou v článku [9] použity. Po jejich definicích následuje samotný popis algoritmu.

Definice 23

Mějme množinu atributů I. Řádek i binární matice D pokrývá I, jestliže $I \subseteq D_{i\_}$. Pokrytí množiny I značíme $cover(I, D)$ a je složeno ze všech indexů řádků, které pokrývají I. Dále definujeme $supp(I, D) = |cover(I, D)|$. Je-li D zřejmé z kontextu, je možné jej vynechat.

Definice 24

Pro množinu atributů I a binární matici D dlaždici příslušnou k I definujeme jako

$$\tau(I, D) = \{(i, j) \mid i \in cover(I, D), j \in I\}$$

Je-li D zřejmé z kontextu, lze psát pouze $\tau(I)$. Obsah dlaždice $\tau(I)$ je roven její velikosti, tedy

$$area(\tau(I, D)) = |\tau(I)| = |I| \cdot |cover(I, D)|$$

Definice 25

Dlaždice $\tau(I, D)$ se nazývá maximální, jestliže neexistuje I' takové, že $I \subset I'$ a $cover(I, D) = cover(I', D)$.

Problém 4 (Large Tile Mining)**.** *Pro binární matici D a práh σ nalézt všechny dlaždice v D, pro které platí $area(\tau(I, D)) \geq \sigma$.*

Large Tile Mining problém řeší algoritmus LTM, který se volá s $I = \emptyset$.

Algoritmus 13 LTM

Input: Matice D, práh σ, I
Output: $\mathcal{T}[I](D, \sigma)$

```
1: T[I] ← ∅
2: PRUNE(D, σ, I)
3: for each i vyskytující se v D do
4:     if |cover({i})|(|I| + 1) ≥ σ then
5:         T[I] ← T[I] ∪ τ(I ∪ {i})
6:     D^i ← ∅
7:     for each j vyskytující se v D t.ž. j > i do
8:         C ← cover({i}) ∩ cover({j})
9:         D^i ← D^i ∪ (j, C)
10:    T[I] ← T[I] ∪ LTM(D^i, σ)
11: return T[I](D, σ)
```

Na počátku se volá procedura PRUNE, která slouží k prořezání „slepých větví“ algoritmu. Zjišťuje, zda dlaždice může vůbec dosáhnout prahu σ a pokud ne, tak odebere atributy a objekty, které se nebudou podílet na rozšíření dlaždice. Tato procedura bude vysvětlena později.

Na řádcích 3–10 je hlavní cyklus algoritmu, který proběhne pro všechny atributy i, které má alespoň jeden objekt. Na řádcích 4–5 se kontroluje, zda přidáním i do množiny atributů I dostaneme dlaždici s větším obsahem než je σ. Případně se přidá do výsledku.

Na řádcích 6–9 se buduje nová matice D^i, pro kterou se poté rekurzivně zavolá algoritmus LTM. Zde se využívá faktu, že všechny velké dlaždice obsahující atribut i, ale neobsahující žádný atribut menší než i mohou být nalezeny v i-podmiňující matici D^i. Ta obsahuje ty řádky D, které obsahují i, ale byly z nich odstraněny všechny atributy menší než i.

Procedura PRUNE nejprve pro každý vyskytující se atribut spočítá horní odhad $UB_{I\cup\{i\}}$. Pro jeho výpočet se nejprve spočítá, kolik řádků současné I-podmiňující matice obsahujících i má velikost alespoň ℓ pro všechna možná ℓ. Tento počet se značí $supp_{\geq\ell}(i, D^I)$. Pak je horní odhad velikosti největší možné dlaždice obsahující $I \cup \{i\}$ dán $UB_{I\cup\{i\}} = max\{(|I| + \ell) \cdot supp_{\geq\ell}(i, D^I) \,|\, \ell \in \{1, 2, \dots\}\}$. Je-li horní odhad menší než dané σ, pak víme, že i nebude součástí množiny atributů, jejíž dlaždice je větší než σ.

Algoritmus 14 PRUNE

Input: Matice D, práh σ, I

```
1: repeat
2:     for each i vyskytující se v D do
3:         if UB_{I∪{i}} < σ then
4:             Odeber i z D
5:         for each tid ∈ cover({i}) do
6:             if size(tid) < ML_{I∪{i}} then
7:                 Odeber tid z cover({i})
8: until došlo ke změně
```

V cyklu na řádcích 5–7 procedura testuje, zda nemůže odebrat některé řádky z matice. Počítá se minimální velikost řádku obsahujícího i, který ještě může vytvořit dlaždici o velikosti alespoň σ. Tato velikost je $ML_{I\cup\{i\}} = min\{\ell \,|\, \ell \cdot supp_{\geq\ell}(i, D^I) \geq \sigma\}$. Obsahuje-li některý řádek méně jedniček než $ML_{I\cup\{i\}}$, může být odebrán.

8.2 Úprava LTM pro rozklad binárních matic

LTM algoritmus lze upravit tak, aby zohledňoval již pokrytou část a vracel dlaždice, které nově pokrývají počet jedniček větší než σ. Z výsledku upraveného algoritmu se vybere dlaždice, jež pokrývá nejvíce jedniček. To se opakuje, dokud nedojde k pokrytí všech jedniček podobně jako u algoritmu GRECON.

První úprava je na řádku 4 algoritmu LTM, kde se při počítání velikosti dlaždice musí odečíst již pokrytá část dané dlaždice. Druhou úpravou je změna počítání horního odhadu velikosti dlaždice v proceduře PRUNE. Musí se počítat další horní odhad, který bere v úvahu již pokrytou část matice. Nakonec se bere jejich minimum: $UB^*_{I\cup\{i\}} = \min\{UB_{I\cup\{i\}}, \sum_{j\in cover(\{i\})} size^*(j)\}$, kde $size^*(j)$ je počet nepokrytých jedniček na řádku $D_{j\_}$.

8.3 Výkon a kvalita rozkladu

Jelikož se do rozkladu přidá vždy jen jedna dlaždice, je vhodné LTM algoritmus ještě vylepšit tak, aby si pamatoval pouze dosud největší dlaždici a také upravoval práh σ na její velikost. Tím se zajistí i větší účinnost procedury PRUNE a také menší paměťová náročnost. I přes tuto úpravu je algoritmus TILING velmi pomalý a pro větší vstupy prakticky nepoužitelný.

Zajímavým faktem je to, že pojem maximální dlaždice splývá s pojmem formální koncept. Algoritmus TILING v každé iteraci přidá dlaždici, která pokrývá nejvíce jedniček. Je tedy principiálně shodný s algoritmem GRECON. Shodnost výsledků jsem experimentálně ověřil na maticích velikosti 32×32. Ve většině případů algoritmy vrátily stejný výsledek, ale neshodovaly se ve všech případech. Je to tím, že není zaručené stejné pořadí procházených konceptů (dlaždic), tudíž

může docházet k rozdílům tam, kde existují dva koncepty (dlaždice) se shodným pokrytím.

Vzhledem k tomu, že jsou výsledky algoritmu Tiling téměř shodné s algoritmem GreCon a že GreCon je výrazně rychlejší, není algoritmus Tiling zahrnut do porovnání v kapitolách 12 a 11.

Obrázek 12: Matice dimenze 256×128 s hustotou 10 % zobrazena do obrázku

Obrázek 13: Matice dimenze 256×128 s hustotou 25 % zobrazena do obrázku

Část II
Porovnání algoritmů

9 Vizualizace faktorů v matici

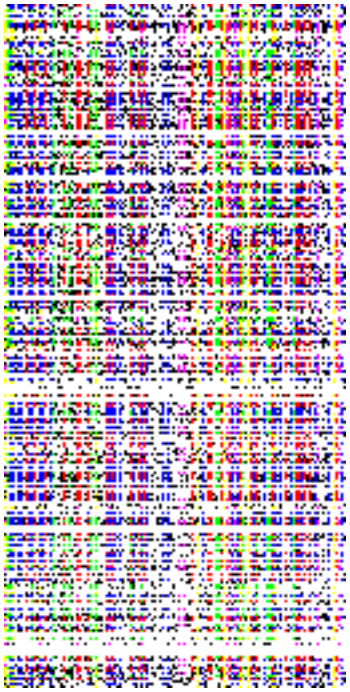
Prohlížení velkých binárních matic v textové podobně není pro člověka přehledné. Pro snazší orientaci v datech a vizualizaci faktorů je vhodné zobrazovat binární matici do obrázku tak, že každý pixel obrázku reprezentuje jednu pozici v matici. Tedy matice I dimenze $n \times m$ bude reprezentována obrázkem s rozměry $n \times m$ a jestliže $I_{ij} = 1$, pak obrázek na pozicích i, j bude obsahovat černý pixel. Toto zobrazení je na obrázcích 12 a 13, na kterých mají matice hustotu 10 % a 25 %.

Uvedené zobrazení slouží jen k rychlému náhledu na data, ale i přes to v něm můžeme pozorovat jisté vzory, které se opakují.

9.1 Zobrazení prvních n faktorů

Většina algoritmů rozkladu se snaží nejprve přidat faktory, které pokrývají nejvíce jedniček. Proto je vhodné zobrazit prvních n faktorů. Pro tento účel se mi jeví jako ideální použít rozdílné barvy faktorů. Na obrázcích 14 a 15 je zobrazeno prvních 5 faktorů, které byly získány algoritmem Asso s parametry $\tau = 0.7, w^+ = 1, w^- = 1$. Pro prvních pět faktorů jsem zvolil barvy červená,

Obrázek 14: Prvních 5 faktorů matice dimenze 256×128 s hustotou 10 %



Obrázek 15: Prvních 5 faktorů matice dimenze 256×128 s hustotou 25 %

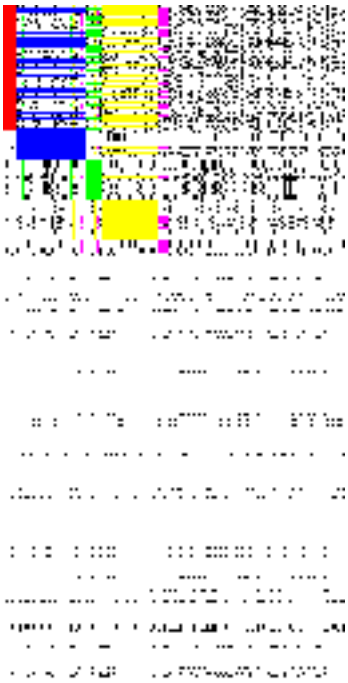
modrá, zelená, žlutá a fialová. Černou barvou jsou jedničky pokryté jinými faktory.

Barevné zobrazení pomáhá odlišit jednotlivé faktory od sebe. Pro lepší zobrazení prvních n faktorů je možné permutovat řádky a sloupce matice tak, aby faktory tvořily viditelný obdélník. Permutaci jsem udělal následovně: všechny řádky a sloupce prvního faktoru byly přerovnány do levého horního rohu. Pro další faktory byly jejich dosud nepermutované řádky a sloupce přerovnány ihned za předchozí. Tím docílíme toho, že dosud nepermutované řádky a sloupce faktorů utvoří obdélník a ten bude sousedit s ostatními faktory jedním vrcholem. Výsledek je na obrázcích 16 a 17, které vznikly permutací řádků a sloupců obrázků 14 a 15. Pro interpretaci faktorů je poté potřeba dodat i použitou permutaci.

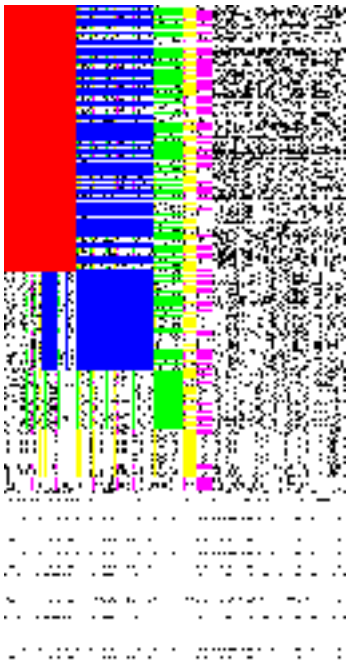
9.2 Zobrazení překryvů faktorů

Předchozí zobrazení dobře ukazuje velikost faktorů, ale jelikož má každý pixel právě jednu barvu(nedochází k míchání barev), nelze z tohoto zobrazení vyčíst, jak moc se faktory překrývají. Pro zobrazení, jak moc se faktory překrývají, jsem zvolil azurovou barvu. Tedy azurový pixel je pokryt alespoň dvěma faktory z prvních pěti. Velké překryvy faktorů jsou typickým rysem některých algoritmů. Na obrázcích 18 a 19 jsou azurovou barvou vyobrazeny překryvy faktorů z obrázků 16 a 17.

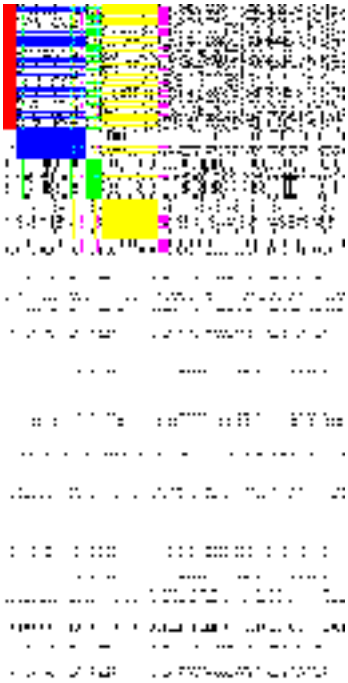
Pro celkový přehled překryvů faktorů jsem použil teplotní mapu ve stupních šedi. Tedy zobrazení, ve kterém je odstín pixelu závislý na počtu faktorů, jež



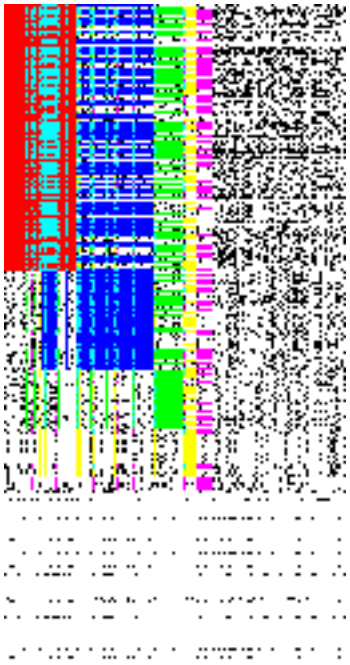
Obrázek 16: Prvních 5 faktorů matice dimenze 256×128 s hustotou 10 % s permutací řádků a sloupců



Obrázek 17: Prvních 5 faktorů matice dimenze 256×128 s hustotou 25 % s permutací řádků a sloupců



Obrázek 18: Prvních 5 faktorů matice dimenze 256×128 s hustotou 10 % s permutací řádků a sloupců a se zobrazením překryvů.



Obrázek 19: Prvních 5 faktorů matice dimenze 256×128 s hustotou 25 % s permutací řádků a sloupců a se zobrazením překryvů.

Obrázek 20: Teplotní mapa faktorů matice 256×128 s hustotou 25 % získaných algoritmem Asso s parametry: $\tau = 0.7, w^+ = 1, w^- = 1$.

Obrázek 21: Teplotní mapa faktorů matice 256×128 s hustotou 25 % získaných algoritmem Asso s parametry: $\tau = 0.9, w^+ = 1, w^- = 1$.

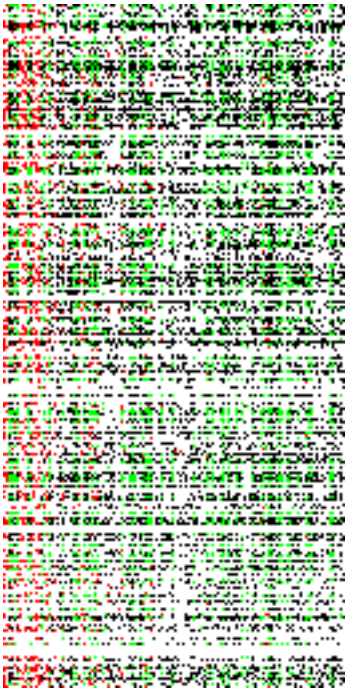
jej překrývají. Pokud má teplotní mapa sloužit k porovnání algoritmů, musí mít stejné měřítko. Na obrázcích 20 a 21 jsou teplotní mapy faktorů matice získaných pomocí algoritmu Asso s parametry $\tau = 0.7, w^+ = 1, w^- = 1$ a $\tau = 0.9, w^+ = 1, w^- = 1$. Lze pozorovat, že u obrázku 20 dochází častěji k překryvům.

9.3 Zobrazení nepokrytí a překrytí

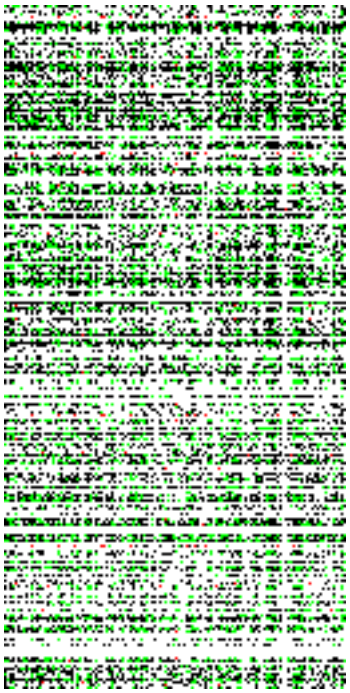
Algoritmy Asso a PaNDa nemusí pokrýt celou matici a také může dojít k překrytí. To lze spočítat porovnáním původní matice s násobkem dílčích matic. Samotný číselný údaj, ke kolika nepokrytím a překrytím došlo, k porovnání algoritmů stačí. Jinou metodou porovnání může být zobrazení nepokrytí a překrytí do obrázku tak, že nepokryté jedničky odlišíme červenou barvou a překryté nuly zelenou. Ukázka je na obrázcích 22 a 23, kde jsou porovnány algoritmy Asso s parametry $\tau = 0.9, w^+ = 1, w^- = 1$ a PaNDa.

10 Sledované veličiny rozkladů

Při srovnání výsledků algoritmů jsem sledoval několik základních veličin rozkladu. Jejich definice a význam je uveden v této kapitole.



Obrázek 22: Nepokrytí a překrytí matice 256×128 s hustotou 25 % algoritmem Asso s parametry: $\tau = 0.9, w^+ = 1, w^- = 1$.



Obrázek 23: Nepokrytí a překrytí matice 256×128 s hustotou 25 % algoritmem PaNDa.

10.1 Počet faktorů

Počet faktorů je základním měřítkem kvality rozkladu. Dle definice problému 1 se má nalézt co nejmenší rozklad. Při průměrování výsledků více rozkladů je vhodné doplnit i směrodatnou odchylku, abychom věděli, jak jsou výsledky stabilní. U nepřesných algoritmů se musí musí brát v potaz i jaké chyby se algoritmy při daném počtu faktorů dopustily.

10.2 Chyba rozkladu

Celkovou chybou rozkladu rozumíme počet pozic, ve kterých se liší původní matice a násobek matic vrácených algoritmy. Celková chyba se zpravidla uvádí v procentech.

Celková chyba je součet chyb překrytí a nepokrytí. V případech, kdy klademe větší důraz na překrytí nebo nepokrytí, je vhodné chyby uvádět zvlášť, jako například v tabulce 5.

10.3 Průběh pokrytí

Další charakteristikou algoritmů je to, jak velké faktory jsou během výpočtu přidávány do rozkladu. To lze zobrazit grafem tak, že se vyobrazí, kolik procent

jedniček bylo v dané iteraci algoritmu celkem pokryto. U algoritmů, které se mohou dopustit nepokrytí, lze z těchto grafů vyčíst, jaké procento jedniček zůstalo nepokryto.

Příkladem jsou průběhy uvedené na obrázcích 31 a 45. U výsledků s náhodně generovanými maticemi bylo procento pokrytých jedniček zprůměrováno.

11 Náhodná data

Pro porovnání algoritmů nad náhodnými daty bylo potřeba vygenerovat matice s různými hustotami. Dimenzi matic jsem zvolil 256×128, při generování byl použit prostřední rozměr dílčích matic 32. Pro každé procento hustoty bylo vygenerováno 100 matic, nad kterými byly spuštěny jednotlivé algoritmy. Dimenze i počet matic byly voleny kompromisně, aby byly testy časově zvládnutelné i pomalejšími algoritmy. Pro algoritmy GreConD, GreEss, Asso a PaNDa by nebyl problém provádět experimenty na řádově větších maticích a ve větším počtu, ale aby mohlo být provedeno srovnání s algoritmy GreCon a Hyper byla velikost i počet zvolena takto.

Náhodné matice byly generovány způsobem, který je uveden v následující podkapitole.

11.1 Generování matic obsahující faktory

Obyčejné generování náhodných pozic jedniček v matici by sice umožňovalo absolutní kontrolu nad hustotou generované matice, ale výsledná matice by prakticky neobsahovala žádné faktory. Naopak, pokud chceme generovat matice s předem určeným počtem faktorů, není jednoduché dosáhnout předem stanovené hustoty. Pro porovnání kvality rozkladu algoritmů je znalost počtu faktorů v matici výhodou, proto jsem volil tento způsob generování.

Jednou z možností, jak matice s faktory generovat, je využití faktu, že faktory tvoří obdélníky v matici. Obdélníky můžeme generovat tak, že náhodně určíme pozici jednoho z vrcholů a také náhodně určíme výšku a šířku obdélníku a na všechny souřadnice obsažené v obdélníku umístíme jedničky. Může se stát, že některý obdélník bude podobdélníkem jiného a tím se zmenší chtěný počet faktorů. Tomu je možné se vyhnout jednoduchou kontrolou.

Příklad 7

Chceme-li, aby matice obsahovala 3 faktory, umístíme do ní 3 náhodné obdélníky plné jedniček.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \vee \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \vee \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

Metoda umísťování celistvých obdélníků do matice sice generuje matici se známým počtem faktorů, ale nejeví se jako obecná. Jelikož generuje pouze celistvé obdélníky, může se stát, že se některé algoritmy budou chovat odlišně.

Druhou možností je generování dvou náhodných matic A a B s dimenzemi $n \times k$ a $k \times m$ a ty následně vynásobit. Výsledná matice $I = A \circ B$ obsahuje předem daný počet faktorů, který je roven společnému rozměru dílčích matic k. Výhodou je, že je předem známý jeden z rozkladů a také přímočařejší implementace.

Problémem je, jak volit hustoty dílčích matic tak, aby výsledkem byla matice s předem danou hustotou. Tento problém není triviální a je mu věnována následující podkapitola.

11.2 Závislost hustoty výsledné matice na hustotách dílčích matic

Násobením různých dvojic matic se stejnou hustotou můžeme dosáhnout zcela odlišných hustot výsledné matice. Násobek matic A, B s hustotami α, β může mít nulovou hustotu, ale také stoprocentní.

Příklad 8

Mějme matice A, B s hustotami $dens(A) = \alpha = 25\%$ a $dens(B) = \beta = 75\%$:

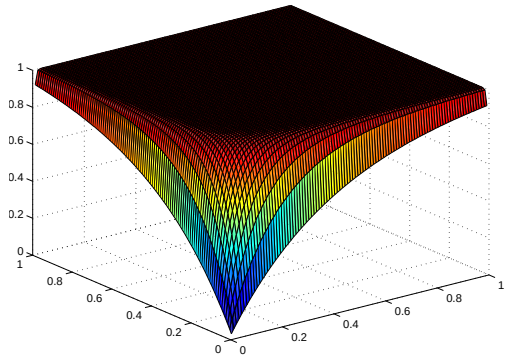
$$A = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Výsledná matice $I = A \circ B$ má hustotu $dens(I) = 0\%$. Naopak matice se stejnými hustotami mohou vést k odlišným výsledkům:

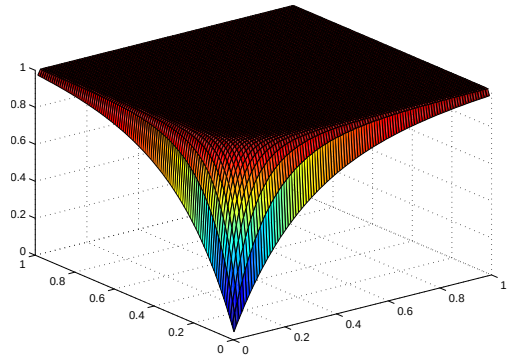
$$C = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, D = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}, C \circ D = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

$dens(C \circ D) = 44\,\%$. Pro první část příkladu byl zvolen záměrně mezní případ, aby vynikla nepředvídatelnost hustoty výsledku.

Abych zjistil, jak je závislá hustota výsledné matice na hustotách dílčích matic, provedl jsem experiment, ve kterém se generovaly dvě náhodné matice s danou hustotou a vynásobily se. Průměrnou hodnotu hustoty výsledné matice z 1000 iterací jsem zanesl do grafů. Na obrázcích 24 a 25 má výsledná matice dimenzi 1000×1000 a společný rozměr matic je 250 a 350. Na obrázcích 26 a 27 má výsledná matice dimenzi 500×500 a společný rozměr matic je 75 a 150.



Obrázek 24: Průměrná hustota výsledné matice dimenze 1000×1000 v závislosti na hustotě dílčích matic s prostředním rozměrem 250.



Obrázek 25: Průměrná hustota výsledné matice dimenze 1000×1000 v závislosti na hustotě dílčích matic s prostředním rozměrem 350.

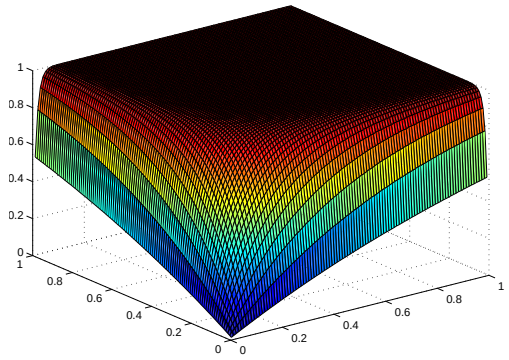
Z grafů lze vyčíst, že čím je společný rozměr větší, tím rychleji roste průměrná hodnota hustoty výsledné matice. Na první pohled se graf jeví jako symetrický, avšak symetrický není. Tedy násobení binárních matic není vzhledem k hustotě dílčích matic a hustotě výsledné matice komutativní. Při podrobném zkoumání symetrie se ukázalo, že ani při více iteracích graf symetričtější není. Lze pozorovat, že s rostoucí hustotou je graf symetričtější. Průměrný rozdíl hustot matic $I_1 = A \circ B$, $dens(A) = \alpha, dens(B) = \beta$ a $I_2 = C \circ D$, $dens(C) = \beta, dens(D) = \alpha$ je řádově v tisících procenta.

11.3 Počet faktorů

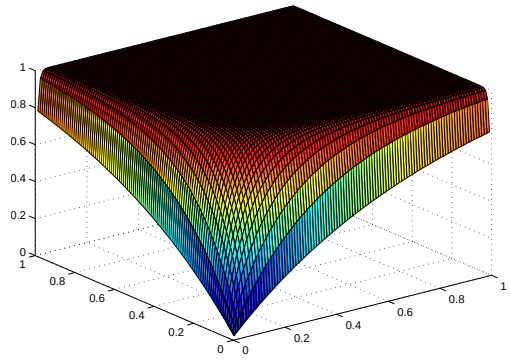
Základním měřítkem kvality rozkladu je počet faktorů, které algoritmy nalezly. V tabulce 1 jsou uvedeny průměrné počty faktorů a směrodatné odchylky pro matice do hustoty matic 30 %. Do tohoto procenta byly všechny algoritmy schopné vypočítat výsledky v „únosném čase“, hodnota byla hraniční pro algoritmus GreCon. Údaje pro zbylé algoritmy a vyšší hustoty matic jsou v tabulce 2.

Z přesných algoritmů je na tom pro řídké matice nejlépe algoritmus GreEss, který produkuje rozklady s nejmenším počtem faktorů a také nízkou odchylkou, avšak od hustoty 63 % podává lepší výsledky GreConD. Naopak nejhůře je na tom algoritmus Hyper, jež produkuje velmi velké rozklady s velkou odchylkou. Vzhledem k jeho nízké rychlosti je pro rozklady matic prakticky nepoužitelný.

Z algoritmů Asso a PaNDa je na tom v průměru lépe PaNDa, ale jeho odchylka je značně větší než odchylka algoritmu Asso. Důležitá je i chyba rozkladu, výsledky jsou v další podkapitole.



Obrázek 26: Průměrná hustota výsledné matice dimenze 500 × 500 v závislosti na hustotě dílčích matic s prostředním rozměrem 75.



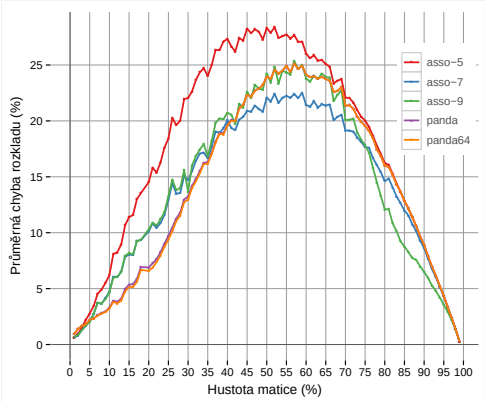
Obrázek 27: Průměrná hustota výsledné matice dimenze 500 × 500 v závislosti na hustotě dílčích matic s prostředním rozměrem 150.

	5 %	10 %	15 %	20 %	25 %	30 %
GreCon	25.43 ± 6.38	30.67 ± 4.13	32.77 ± 4.43	34.17 ± 6.59	33.60 ± 8.15	34.72 ± 9.66
GreConD	25.34 ± 6.37	30.52 ± 3.24	33.14 ± 3.81	33.59 ± 3.95	34.60 ± 5.07	37.85 ± 9.06
GreEss	24.63 ± 6.85	29.83 ± 3.30	31.35 ± 1.75	32.46 ± 2.76	32.80 ± 2.19	34.19 ± 5.41
Asso-5	25.5 ± 7.68	29.56 ± 4.17	30.40 ± 4.02	30.72 ± 3.65	30.42 ± 4.15	29.21 ± 5.30
Asso-7	26.13 ± 7.65	29.61 ± 4.03	30.76 ± 2.70	31.55 ± 1.71	31.65 ± 1.47	32.00 ± 1.48
Asso-9	26.23 ± 7.67	29.77 ± 3.98	30.98 ± 2.35	31.66 ± 1.27	31.79 ± 1.00	32.00 ± 0.97
PaNDa	17.32 ± 6.57	25.23 ± 6.19	28.15 ± 5.20	28.77 ± 5.04	28.99 ± 5.59	28.29 ± 5.22
PaNDa64	17.33 ± 6.60	26.13 ± 7.34	30.49 ± 7.97	31.51 ± 7.56	32.11 ± 7.96	30.95 ± 7.82
Hyper-1	87.04 ± 36.62	112.13 ± 40.96	149.42 ± 53.69	134.94 ± 53.76	111.42 ± 45.31	143.59 ± 87.61
Hyper-2	80.39 ± 42.96	92.48 ± 33.67	107.89 ± 28.09	128.92 ± 31.76	150.27 ± 48.04	166.29 ± 56.40
Hyper-4	78.20 ± 45.51	90.75 ± 35.03	102.87 ± 30.34	111.46 ± 21.42	116.15 ± 17.69	120.01 ± 21.31

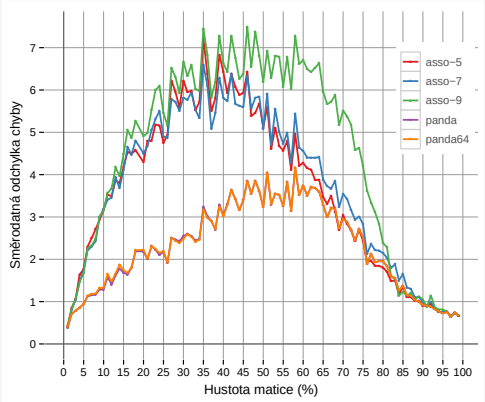
Tabulka 1: Průměrný počet faktorů a směrodatné odchylky dle hustoty matic.

	35 %	40 %	60 %	70 %	80 %	90 %
GreConD	38.90 ± 8.40	39.19 ± 6.81	53.43 ± 16.72	61.37 ± 17.88	74.36 ± 14.17	82.39 ± 6.14
GreEss	34.30 ± 4.49	33.88 ± 3.57	50.88 ± 13.78	81.76 ± 16.32	112.03 ± 11.36	97.52 ± 6.17
Asso-5	28.95 ± 4.69	29.23 ± 5.21	15.44 ± 7.25	8.76 ± 5.14	4.76 ± 3.63	1.52 ± 1.00
Asso-7	32.00 ± 1.46	31.70 ± 1.09	29.79 ± 3.79	20.8 ± 6.20	10.16 ± 4.82	2.33 ± 1.51
Asso-9	32.00 ± 0.95	32.00 ± 0.00	32.00 ± 0.00	32.00 ± 0.00	31.98 ± 0.20	19.90 ± 4.42
PaNDa	27.12 ± 5.64	26.98 ± 4.80	17.03 ± 6.91	12.63 ± 6.43	7.02 ± 5.44	1.72 ± 1.82
PaNDa64	28.42 ± 7.09	27.81 ± 5.92	17.04 ± 6.94	12.63 ± 6.43	7.02 ± 5.44	1.72 ± 1.82
Hyper-2	158.82 ± 52.03	141.81 ± 48.57	–	–	–	–
Hyper-4	125.10 ± 27.89	152.08 ± 23.22	–	–	–	–

Tabulka 2: Průměrný počet faktorů a směrodatné odchylky dle hustoty matic.



Obrázek 28: Průměrná chyba rozkladu v závislosti na hustotě matice.



Obrázek 29: Směrodatná odchylka průměrné chyby v závislosti na hustotě.

11.4 Chyba rozkladu

V této podkapitole jsou uvažovány pouze algoritmy, které se dopouští chyby. Z výsledků vyplývá, že do hustoty 40 % se nejmenší chyby dopouští algoritmus PANDa, dle směrodatné odchylky je chyba poměrně stabilní, a to i přesto, že dává menší rozklady. Hustoty mezi 40 % a 75 % nejlépe zvládá algoritmus ASSO s parametrem $\tau = 0.7$, pro hustší matice podává lepší výsledky jen ASSO s parametrem $\tau = 0.9$.

Zaujala mě podobnost výsledků algoritmu PANDa při použití $k = 32$ a $k = 64$. Liší se jen velmi málo, algoritmus s parametrem $k = 64$ zřídkakdy překročil 32 faktorů. Při použití $k = 16$ jsou výsledky odlišné.

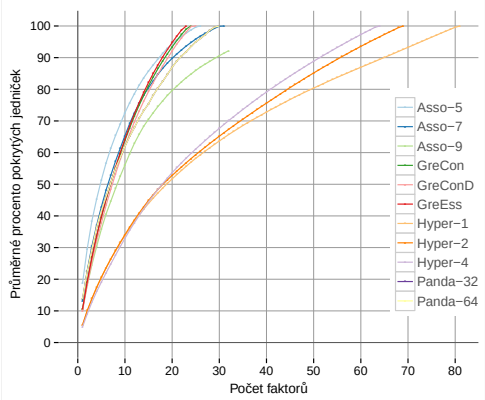
Podrobné zachycení chyby a směrodatné odchylky je na obrázcích 28 a 29. Výsledky jsou opět ze 100 testů pro každé procento hustoty.

11.5 Průběh pokrytí

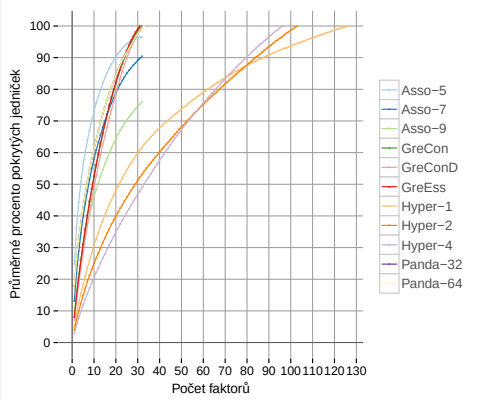
Algoritmy založené na FCA mají prakticky totožné průběhy. Při podrobnějším zkoumání lze pozorovat, že u nižších hustot GREESS vždy pokrývá více než ostatní. Na posledním grafu na obrázku 35 již lze pozorovat propad algoritmu GREESS, který se okolo 40. faktoru dostává pod GRECOND.

V grafech lze dobře vidět vliv parametru τ u algoritmu ASSO. Algoritmus PANDa se stabilně vyskytuje mezi algoritmem ASSO s $\tau = 0.7$ a $\tau = 0.5$.

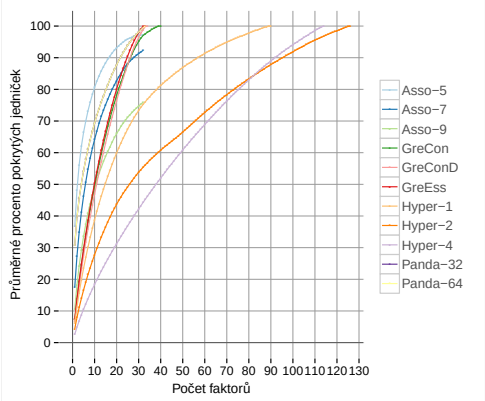
Zajímavé průběhy má algoritmus HYPER, kde například na obrázku 31 můžeme pozorovat, že verze s $\alpha = 0.1$ nejprve přidává větší faktory a je na tom lépe než verze s vyšším α, avšak od 70 % je průběh přidávání prakticky lineární a nakonec verze s $\alpha = 0.2$ a $\alpha = 0.4$ dosáhnou menšího rozkladu.



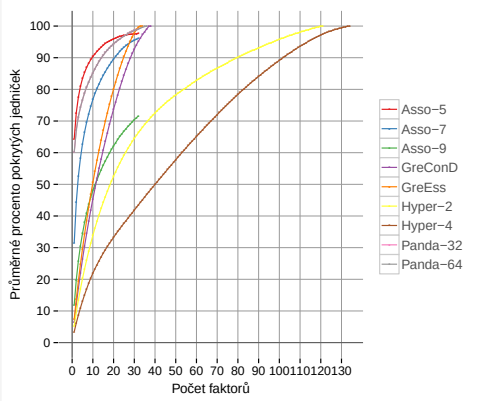
Obrázek 30: Průměrný průběh pokrytí matic s hustotou 5 %.



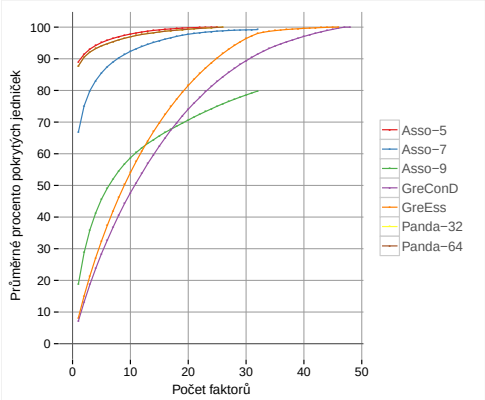
Obrázek 31: Průměrný průběh pokrytí matic s hustotou 15 %.



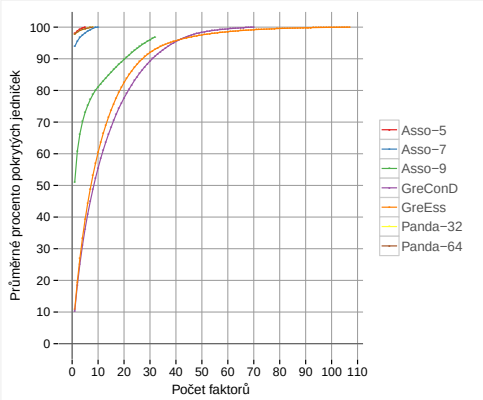
Obrázek 32: Průměrný průběh pokrytí matic s hustotou 25 %.



Obrázek 33: Průměrný průběh pokrytí matic s hustotou 40 %.



Obrázek 34: Průměrný průběh pokrytí matic s hustotou 60 %.



Obrázek 35: Průměrný průběh pokrytí matic s hustotou 80 %.

Dataset	Dimenze	$dens(I)$	$\lvert\mathcal{B}(X,Y,I)\rvert$	$dens(\mathcal{E}(I))$	$\lvert\mathcal{F}_{0.1}\rvert$	$\lvert\mathcal{F}_{0.2}\rvert$
adult	32561×149	10.0 %	3 784 601	9.2 %	6 467	1392
dblp	6980×19	12.9 %	2 495	1.2 %	16	4
DNA	4590×392	1.4 %	4 483	0.09 %	2	1
flag	194×107	19.7 %	35 717	9.1 %	10 355	4 392
hayes	132×18	27.8 %	382	26.5 %	64	22
chess	3196×76	48.7 %	930 851 337	29.3 %	123 243 073	22 808 625
lenses	24×12	41.7 %	130	31.3 %	105	31
mushroom	8124×119	19.3 %	238 710	8.2 %	4 897	1203
nursery	12960×32	28.1 %	183 079	28.0 %	156	30
paleo	501×139	5.0 %	10 225	2.7 %	13	1
plants	34781×70	12.4 %	5 052 567 629	0.5 %	360	4
post	90×25	32.0 %	1 503	30.4 %	441	137
servo	167×19	21.0 %	434	19.7 %	25	11
shuttle	15×23	30.4 %	54	8.4 %	53	28
zoo	101×28	30.5 %	379	6.9 %	192	102

Tabulka 3: Vybrané reálné datasety.

12 Reálná data

Experimenty s reálnými daty byly prováděny s vybranými datasety, které byly převážně čerpány z [28]. Dataset dblp byl čerpán z [5], DNA z [19], paleo z [21]. Původní data nebyla v binární podobě, do binárních jsem je transformoval klasickým způsobem. Vícehodnotový atribut byl převeden na tolik binárních atributů, kolika hodnot mohl nabývat. Jednička je pak přítomna právě v jednom novém atributu, který reprezentuje přítomnou hodnotu. Pro každý číselný atribut byl dle charakteristiky hodnot zvolen počet intervalů, pro které se vytvořil nový binární atribut a jednička byla přidělena dle příslušnosti původní hodnoty v intervalu.

Zvolené datasety jsou uvedeny v tabulce 3, kde jsou uvedeny i jejich základní vlastnosti. Zajímavé je, že u zdánlivě podobných datasetů se mohou rapidně lišit odvozené údaje jako je hustota esenciální matice či velikost $\mathcal{F}_\alpha$. Například datasety adult a plants mají podobnou hustotu i počet objektů, ale velikost konceptuálního svazu má plants mnohem větší. Dále esenciální matice datasetu adult je jen o trochu řidší než původní matice, ale esenciální matice pro dataset plants je téměř prázdná.

12.1 Počet faktorů

V tabulce 4 jsou znázorněny počty faktorů, které nalezly algoritmy pro vybrané datasety. Ve většině případů nalezl nejméně faktorů algoritmus PaNDa, zajímavé však je to, že u datasetů nursery a servo nalezl faktorů nejvíce. Malý počet

Dateset	GreCon	GreConD	GreEss	Asso-5	Asso-7	Asso-9	PaNDa	Hyper-1	Hyper-2	Hyper-4
adult	184	191	171	159	142	**136**	142	**318**	218	178
dblp	22	22	19	15	15	15	**11**	**24**	19	19
DNA	508	**511**	372	89	128	217	**11**	391	391	391
flag	108	107	97	57	72	76	**16**	**145**	117	98
hayes	24	25	24	**14**	**14**	**14**	**14**	**39**	25	18
chess	–	**124**	123	**36**	43	56	56	–	–	–
lenses	11	11	11	**6**	9	9	10	**15**	**15**	12
mushroom	116	120	104	**78**	81	85	150	**182**	150	130
nursery	31	31	31	27	**25**	**25**	**107**	46	31	31
paleo	**154**	151	145	125	111	111	**19**	139	139	139
plants	–	112	70	44	63	63	**24**	**113**	70	70
post	28	29	30	14	20	20	**9**	**48**	37	26
servo	19	19	19	16	**15**	**15**	**25**	22	19	19
shuttle	16	17	13	13	15	14	**2**	22	21	**23**
zoo	29	30	25	13	15	21	**6**	**38**	35	30

Tabulka 4: Počet nalezených faktorů s vyznačenými maximy a minimy.

faktorů však způsobuje větší chybu rozkladu, která je podrobněji popsána v další podkapitole.

Algoritmy GreCon, GreConD a GreEss podávají podobné výsledky. Nejlépe je na tom GreEss, který až na dataset post vypočítal nejmenší rozklady z těchto tří algoritmů.

Nejhorší výsledky podává algoritmus Hyper s parametrem $\alpha = 0.1$, velmi podobně jsou na tom i varianty s $\alpha = 0.2$ a $\alpha = 0.4$, přitom výpočty těchto algoritmů trvaly většinou nejvíce času.

Chybějící hodnoty u datasetů chess a plants jsou způsobeny obrovským konceptuálním svazem datasetů. Rozklad byl z časových důvodů neproveditelný.

12.2 Chyba rozkladu

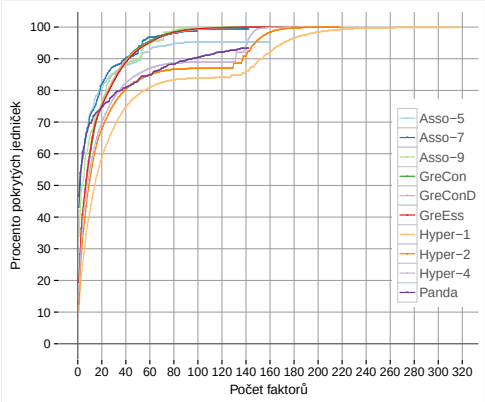
Algoritmy, které mohou produkovat nepřesný rozklad, produkují většinou výrazně menší rozklady. Rozhodující je, jaké chyby se při tom dopustí. V tabulce 5 je v tomto pořadí uveden počet nepokrytých jedniček, překrytých nul a celková chyba rozkladu.

Největší chyby nepokrytí se dopouští algoritmus PaNDa, nejvíce nul překrývá algoritmus Asso s parametrem $\tau = 0.5$, ten se také dopouští největší celkové chyby.

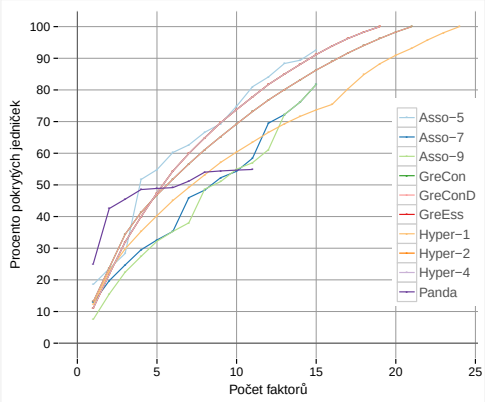
Při rozkladu binárních matic pomocí algoritmů, které mohou produkovat nepřesné rozklady, je důležité rozhodnout, zda chceme rozklad na méně faktorů s potenciálně větší chybou nebo nám nevadí větší rozklady a požadujeme malou chybu. Toto rozhodnutí závisí na kvalitě zdrojových dat a na zamýšlené aplikaci.

Dateset	Asso-5			Asso-7			Asso-9			PaNDa		
adult	22 089	106 466	2.65 %	2 173	80 526	1.70 %	288	105 882	2.19 %	31 948	99 787	2.72 %
dblp	1 264	5 421	5.04 %	3 143	378	2.65 %	3 143	0	2.37 %	7 751	2 287	7.57 %
DNA	2 393	12 120	0.81 %	14 04	10 826	0.68 %	308	9 450	0.54 %	17 626	2 564	1.12 %
flag	109	1 752	8.97 %	41	893	4.50 %	2	857	4.14 %	801	694	7.20 %
hayes	37	224	10.98 %	120	0	5.05 %	120	0	5.05 %	171	79	10.52 %
chess	1 265	29 865	12.82 %	994	23 614	10.13 %	116	30 693	12.68 %	1 940	27 616	12.17 %
lenses	27	28	19.10 %	10	9	6.60 %	8	6	4.86 %	15	8	7.99 %
mushroom	9 840	92 141	10.55 %	3 634	66 080	7.21 %	204	58 652	6.09 %	4 467	46 757	5.30 %
nursery	5 224	21 552	6.46 %	14 154	0	3.41 %	14 154	0	3.41 %	1 878	9 060	2.64 %
paleo	386	2141	3.63 %	359	406	1.10 %	408	3	0.59 %	2 444	384	4.06 %
plants	25 518	185 430	8.66 %	13 855	124 542	5.68 %	1 570	24 858	1.09 %	55 348	117 820	7.11 %
post	18	346	16.18 %	31	79	4.89 %	13	9	0.98 %	95	241	14.93 %
servo	7	49	1.76 %	79	3	2.58 %	79	0	2.49 %	134	73	6.52 %
shuttle	1	48	14.20 %	0	25	7.24 %	0	41	11.88 %	48	10	16.81 %
zoo	24	196	7.78 %	22	175	6.97 %	7	132	4.91 %	88	203	10.29 %

Tabulka 5: Počet nepokrytých jedniček, překrytých nul a celková chyba rozkladu algoritmů.



Obrázek 36: Průběh pokrytí algoritmů na datasetu adult.



Obrázek 37: Průběh pokrytí algoritmů na datasetu dblp.

12.3 Průběh pokrytí

Na obrázcích 36–50 jsou vyobrazeny průběhy algoritmů na všech vybraných datasetech. Číslo u názvu algoritmu značí desetinnou hodnotu jeho atributu. Hyper-1 značí algoritmus Hyper s parametrem $\alpha = 0.1$, Asso-7 značí algoritmus Asso s parametrem $\tau = 0.7$.

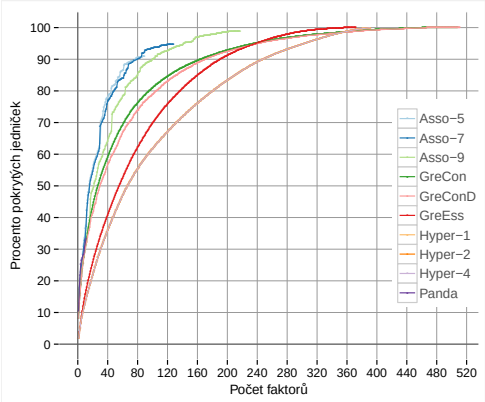
Nejvíce mě zaujal průběh algoritmu Hyper na datasetech adult (obr. 36), flag (obr. 39) a mushroom (obr. 43), kde je pozorovatelné plynulé přidávání faktorů, které se v určitém okamžiku výrazně změní. Konkrétně u datasetu adult se velikost přidávaného faktoru plynule zmenšuje, avšak okolo 130. faktoru nastane zvrat a začnou se přidávat výrazně větší faktory. Při pohledu do výsledných dat algoritmus Hyper s parametrem $\alpha = 0.2$ do 17. faktoru přidával víceatributové faktory. Od 18. faktoru do 130. byly přidávány pouze jednoatributové faktory. Poté byl přidán víceatributový faktor, který byl následován několika dalšími jednoatributovými faktory a poté, až na drobné výjimky, byly přidány další víceatributové faktory.

Významným jevem je i to, že algoritmy založené na FCA mají plynulé a hladké křivky. To je způsobeno vlastnosti jejich greedy pravidla výběru dalšího faktoru, tedy to, že nikdy nemůže být vybrán větší faktor, než v předchozí iteraci. Tuto vlastnost zbylé algoritmy nemají.

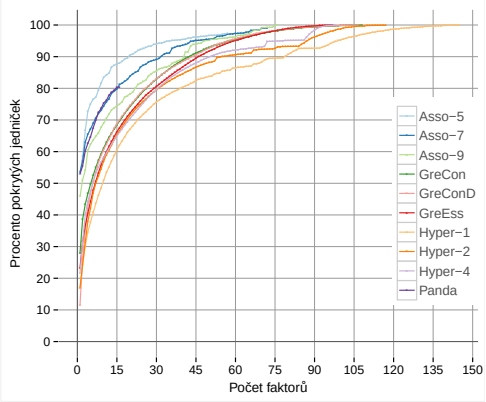
Zaujaly mě i průběhy rozkladů datasetu plants, u kterého algoritmy GreEss, Hyper s parametrem $\alpha = 0.2$ a Hyper s parametrem $\alpha = 0.4$ vypočítaly prakticky totožný rozklad. Rozdíl byl pouze v posledním faktoru, který měl GreEss o jeden atribut větší. V grafu jsou křivky těchto algoritmů nerozlišitelné.

12.4 Rozbor faktorů

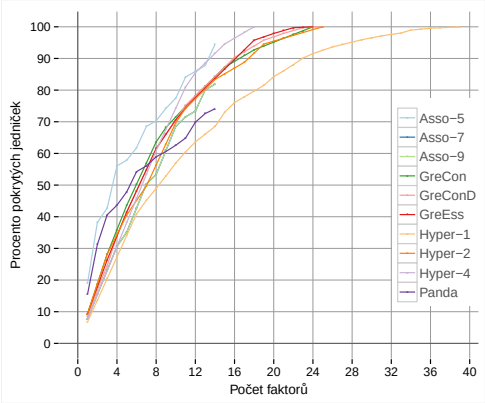
Výhodou reálných dat je to, že na nich lze lépe ukázat smysl rozkladu binárních matic a také to, že faktory jsou lépe představitelné a pojmenovatelné. Pro roz-



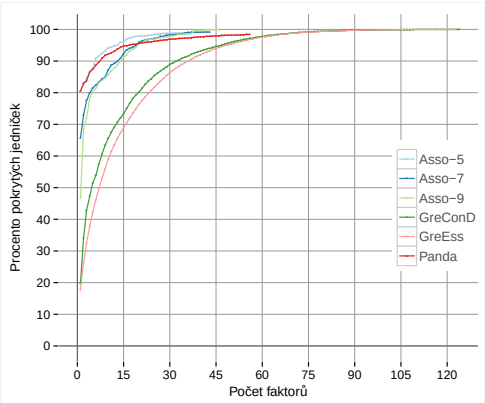
Obrázek 38: Průběh pokrytí algoritmů na datasetu DNA.



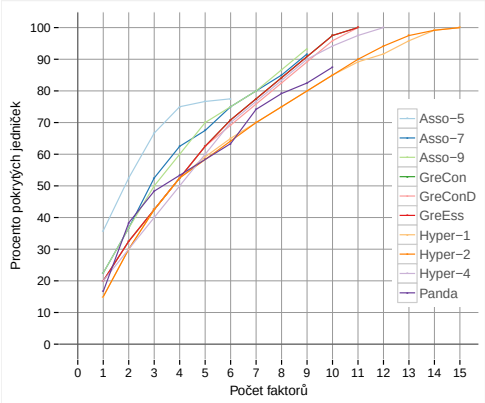
Obrázek 39: Průběh pokrytí algoritmů na datasetu flag.



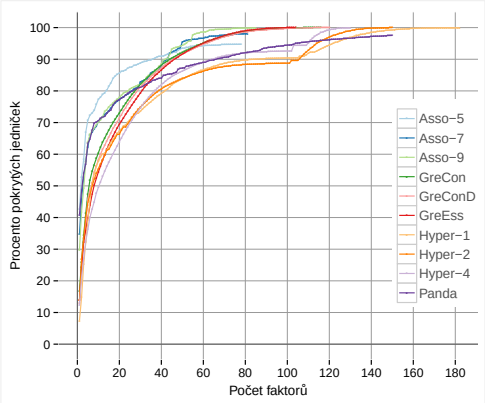
Obrázek 40: Průběh pokrytí algoritmů na datasetu hayes.



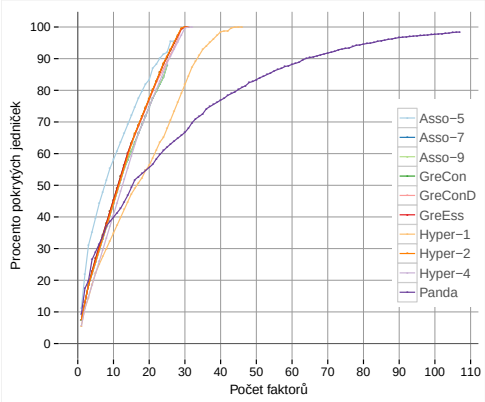
Obrázek 41: Průběh pokrytí algoritmů na datasetu chess.



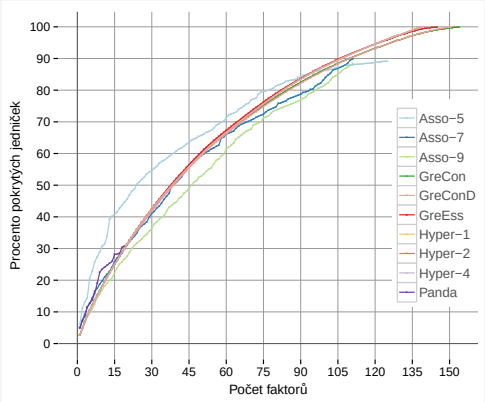
Obrázek 42: Průběh pokrytí algoritmů na datasetu lenses.



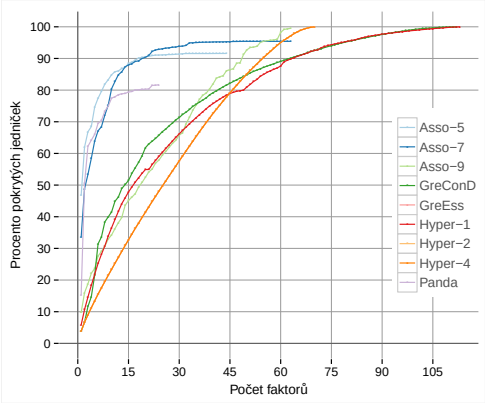
Obrázek 43: Průběh pokrytí algoritmů na datasetu mushroom.



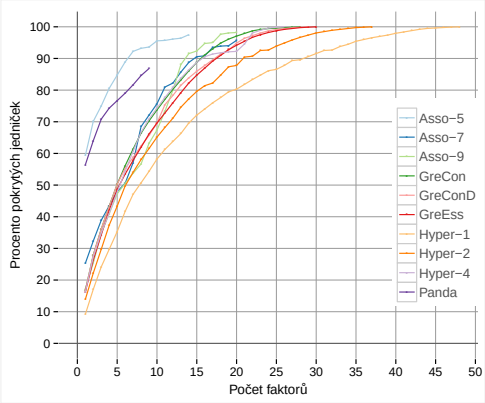
Obrázek 44: Průběh pokrytí algoritmů na datasetu nursery.



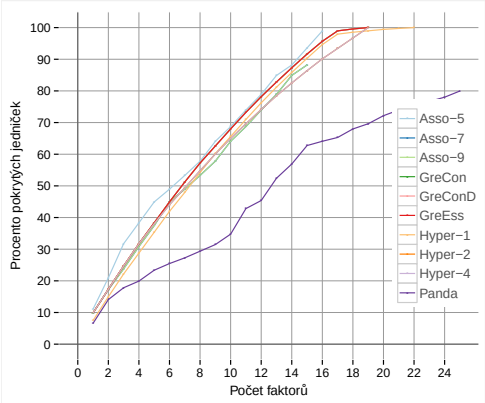
Obrázek 45: Průběh pokrytí algoritmů na datasetu paleo.



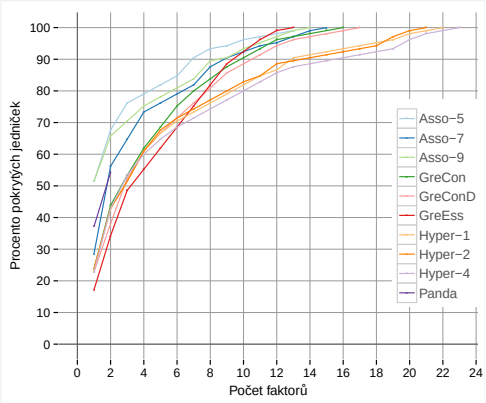
Obrázek 46: Průběh pokrytí algoritmů na datasetu plants.



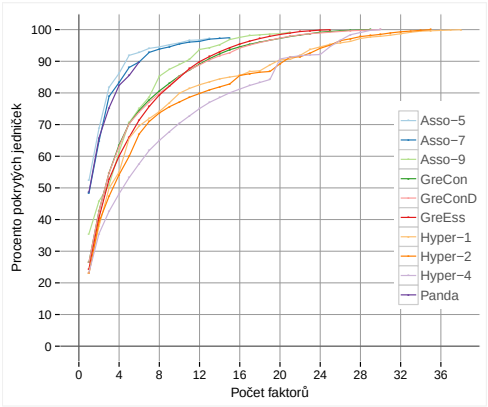
Obrázek 47: Průběh pokrytí algoritmů na datasetu post.



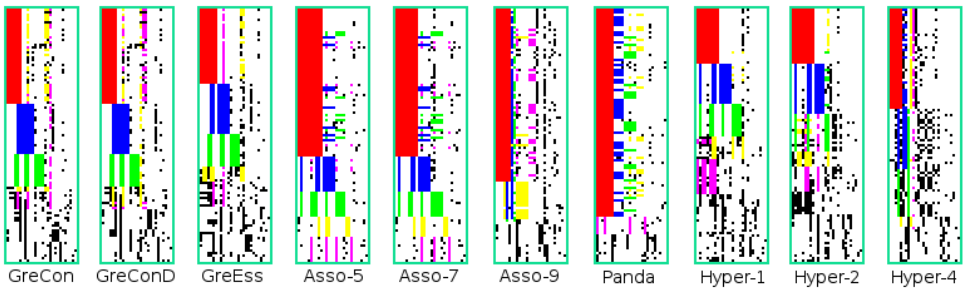
Obrázek 48: Průběh pokrytí algoritmů na datasetu servo.



Obrázek 49: Průběh pokrytí algoritmů na datasetu shuttle.



Obrázek 50: Průběh pokrytí algoritmů na datasetu zoo.



Obrázek 51: Vizualizace prvních 5 faktorů datasetu zoo s permutací řádků a sloupců.

bor faktorů jsem si vybral dataset zoo, který má přijatelnou velikost, pro jeho nastudování není zapotřebí speciálních znalostí a výsledné faktory jsou přímo představitelné.

Na obrázku 51 je vyobrazeno prvních 5 faktorů, které vypočítaly jednotlivé algoritmy. Pozorovatelné jsou rozdílné velikosti prvních faktorů a také to, že například algoritmus Asso s parametrem $\tau = 0.5$ přiřadí každému objektu alespoň jeden z pěti prvních faktorů. Dále je vidět, jak algoritmus Hyper se stoupajícím α dodává faktory s méně atributy.

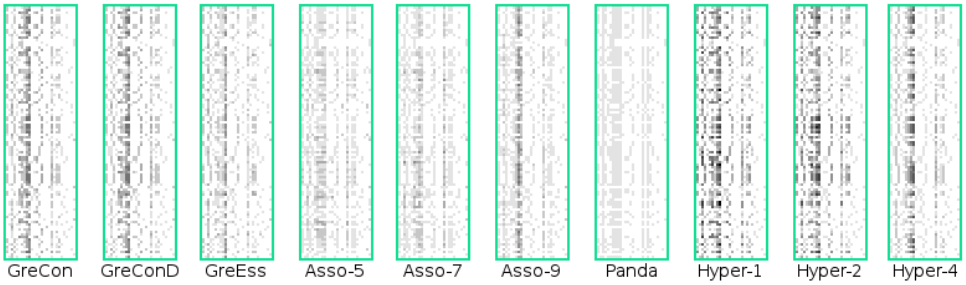
Tabulka 6 ukazuje podrobný rozbor prvních 5 faktorů datasetu zoo. V tabulce jsou uvedeny všechny atributy faktorů a vybraných pět objektů tak, aby byly dostatečně reprezentativní, případně, aby byla vyobrazena možná chyba algoritmů. Chyba je zvýrazněna tučně. Zajímavá chyba je například u prvního faktoru algoritmu Asso s parametrem $\tau = 0.5$, který mezi čtyřnohé savce, predátory se srstí, zařadil labuť.

Při podrobném pohledu do tabulky můžeme pozorovat, že některé faktory prakticky splývají s taxonomickými kategoriemi zvířat. Například druhý faktor algoritmu GreCon jsou pravděpodobně ptáci a třetí faktor ryby. Hmyz můžeme nalézt ve čtvrtém faktoru algoritmu Asso s parametrem $\tau = 0.5$ a $\tau = 0.7$.

V tabulce může být matoucí atribut *kočkovitý*, který jsem přeložil z anglického *catsize*. Patrně je to obecnější výraz pro stavbu těla, protože tento atribut má

třeba i medvěd. Přesný význam tohoto atributu se mi nepodařilo dohledat.

Na závěr porovnání zbývá ukázat teplotní mapu překryvů faktorů, která je zobrazena na obrázku 52. Největších překryvů se dopouští algoritmus HYPER, následován algoritmem GRECOND. Suverénně nejmenších překryvů se dopouští algoritmus PANDA, kde je každá pokrytá jednička pokryta právě jedním faktorem.



Obrázek 52: Teplotní mapa datasetu zoo.

		GreCon	GreConD	GreEss	Asso-5	Asso-7	Asso-9	PaNDa	Hyper-1	Hyper-2	Hyper-4
1.	A	srst, mléko, zuby, páteř, dýchá, 1. kat.	srst, mléko, zuby, páteř, dýchá, 1. kat.	srst, mléko, zuby, páteř, dýchá, 4 končetiny, 1. kat.	srst, mléko, predátor, zuby, páteř, dýchá, 4 končetiny, ocas, kočkovitý, 1. kat.	srst, mléko, zuby, páteř, dýchá, 4 končetiny, ocas, kočkovitý, 1. kat.	srst, mléko, zuby, páteř, dýchá, 1. kat.	**vajíčka**, predátor, zuby, páteř, dýchá, ocas, kočkovitý	srst, mléko, zuby, páteř, dýchá, 4 končetiny, ocas, kočkovitý, 1. kat.	srst, mléko, zuby, páteř, dýchá, 4 končetiny, ocas, kočkovitý, 1. kat.	mléko, zuby, páteř, dýchá, 1. kat.
	O	antilopa, gepard, žirafa, sob, kaloň	antilopa, gepard, žirafa, sob, kaloň	antilopa, gepard, žirafa, sob, vlk	medvěd, koza, gorila, **labuť**, vlk	medvěd, koza, gorila, **labuť**, vlk	antilopa, gepard, zajíc, **bažant**, krtek	antilopa, gepard, **jestřáb**, tchoř, veverka, vlk	koza, gepard, tchoř, jelen, vlk	koza, gepard, tchoř, jelen, vlk	antilopa, gepard, křeček, tchoř, veverka
2.	A	peří, vajíčka, páteř, dýchá, 2 končetiny, ocas, 2. kat.	peří, vajíčka, páteř, dýchá, 2 končetiny, ocas, 2. kat.	peří, vajíčka, páteř, dýchá, 2 končetiny, ocas, 2. kat.	peří, vajíčka, létá, páteř, dýchá, 2 končetiny, ocas, 2. kat.	peří, vajíčka, létá, páteř, dýchá, 2 končetiny, ocas, 2. kat.	páteř, ocas	srst, mléko, 4 končetiny, 1. kat.	peří, vajíčka, létá, páteř, dýchá, 2 končetiny, ocas, 2. kat.	peří, vajíčka, páteř, dýchá, 2 končetiny, ocas, 2. kat.	predátor, páteř, ocas
	O	kuře, racek, bažant, vrabec, sup	kuře, racek, bažant, vrabec, sup	kuře, racek, bažant, vrabec, sup	kuře, plameňák, **moucha**, chaluha, upír	kuře, plameňák, **moucha**, chaluha, upír	antilopa, gepard, kuře, tučňák, vrabec	antilopa, gepard, jelen, zajíc, **tuleň**	kuře, plameňák, chaluha, bažant, sup	kuře, plameňák, chaluha, bažant, sup	okoun, gepard, delfín, štika, vlk
3.	A	vajíčka, voda, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	vajíčka, voda, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	vajíčka, voda, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	vajíčka, voda, predátor, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	vajíčka, voda, predátor, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	páteř, kočkovitý	peří, létá, **voda**, 2 končetiny, 2. kat.	vajíčka, voda, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	voda, predátor, páteř, ocas	vajíčka, páteř
	O	okoun, ostroun, sleď, mořský koník, tuňák	okoun, ostroun, sleď, mořský koník, tuňák	okoun, ostroun, sleď, mořský koník, tuňák	okoun, sumec, delfín, piraňa, **tuleň**	okoun, sumec, delfín, piraňa, **tuleň**	gepard, slon, **štika**, poník, vlk	kuře, holub, jestřáb, **tučňák**, sup	okoun, kapr, ostroun, štika, tuňák	okoun, ostroun, štika, lachtan, tuňák	okoun, kapr, mořský koník, tuňák, ptakopysk
4.	A	páteř, dýchá, 4 končetiny, ocas, kočkovitý	predátor, páteř, ocas	mléko, predátor, páteř, dýchá, kočkovitý, 1. kat.	vajíčka, létá, dýchá, 6 končetin, 6. kat.	vajíčka, létá, dýchá, 6 končetin, 6. kat.	vajíčka, voda, zuby, páteř, ploutve, 0 končetin, ocas, 4. kat.	voda, ploutve, 0 končetin, 4. kat.	voda, predátor, páteř, dýchá	mléko, predátor, zuby, páteř, dýchá, 1. kat.	srst, dýchá
	O	antilopa, gepard, rys, tchoř, vlk	okoun, racek, mangusta, sviňucha, sup	medvěd, gepard, tchoř, kočka, vlk	blecha, komár, včela, beruška, termit	blecha, komár, včela, beruška, termit	okoun, ostroun, štika, mořský koník, tuňák	okoun, kapr, delfín, mořský koník, tuňák	delfín, žába, tučňák, tuleň, chaluha	medvěd, gepard, lev, tchoř, vlk	antilopa, gepard, lev, ptakopysk, vlk
5.	A	predátor, páteř, ocas	páteř, dýchá, 4 končetiny, ocas, kočkovitý	dýchá, ocas	vajíčka, voda, predátor, 7. kat.	vajíčka, predátor, 7. kat.	peří, vajíčka, páteř, dýchá, 2 končetiny, ocas, 2. kat.	vajíčka, voda, predátor, 7. kat.	srst, mléko, zuby, páteř, dýchá, 1. kat.	vajíčka, zuby, páteř	kočkovitý
	O	okoun, racek, mangusta, sviňucha, sup	antilopa, gepard, rys, tchoř, vlk	antilopa, plameňák, rys, puma, vrabec	škeble, rak, chobotnice, slimák, ropucha	škeble, chobotnice, mořský koník, hatérie, červ	kuře, holub, jestřáb, bažant, sup	škeble, krab, humr, rak, hvězdice	antilopa, gepard, gorila, sob, vlk	okoun, kapr, štika, trnucha, tuňák	antilopa, gepard, lev, sob, vlk

Tabulka 6: Prvních 5 faktorů datasetu zoo s vybranými objekty.

13 Diskuze

Tato kapitola obsahuje shrnutí výsledků a poznatků z této práce, doporučení pro nastavení algoritmů.

- Možnost použití algoritmů GreCon a Hyper je závislá na velikosti konceptuálního svazu, která s rostoucí hustotou prudce roste. Pro matice dimenze 256×128 praktická použitelnost končí okolo hustoty 30 %.
- GreCon potřebuje vypočítat celý konceptuální svaz, to může způsobit velké nároky na operační paměť počítače. Paměťovou náročnost lze snížit uložením svazu na disk, což výrazně zpomalí algoritmus nebo lze konceptuální svaz vypočítávat v každé iteraci a k tomu je zapotřebí mnoho procesorového času.
- Průměrný počet povinných konceptů klesá s rostoucí hustotou matice.
- Algoritmus GreConD podává velmi podobné výsledky jako algoritmus GreCon a přitom je řádově rychlejší a nenáročný na paměť.
- Algoritmus GreEss pro matice do hustoty kolem 60 % většinou vypočítá nejmenší rozklad z algoritmů, které produkují rozklad zdola. Pro hustoty nad 60 % je na tom v průměru lépe algoritmus GreConD.
- Algoritmus Asso je velmi rychlý. Chyba překrytí nul klesá s rostoucím atributem τ. Pro malé τ vrací algoritmus rozklady na malý počet faktorů, s rostoucím τ průměrný počet faktorů stoupá.
- Pro vyvážený poměr překrytí nul, pokrytí jedniček a celkovou chybu doporučuji nastavit parametr τ algoritmu Asso kolem 0.8. Díky rychlosti algoritmu je snadné odladit parametr τ pro konkrétní dataset.
- Pro nastavení parametrů w^+ a w^- algoritmu Asso je rozhodující jejich vzájemný poměr, nikoliv absolutní velikost.
- Algoritmus PaNDa je také rychlý, v průměrů dává nejmenší rozklady, avšak s velkou celkovou chybou.
- Algoritmus Hyper s parametrem $\alpha = 0.1$ dává v průměru největší rozklady, a to i přesto, že první faktory jsou výrazně větší než faktory vypočítané algoritmem Hyper s parametry $\alpha = 0.2$ a $\alpha = 0.4$.
- Rychlost algoritmu Hyper je závislá na volbě algoritmu pro výpočet $\mathcal{F}_\alpha$ a také na řadícím algoritmu. Pro matice s mnoha objekty se často řadí mnoho objektů.
- Pro výpočet množiny $\mathcal{F}_\alpha$ doporučuji použít algoritmus FCbO, který je výrazně rychlejší než Apriori algoritmus. S použitím algoritmu FCbO dosahuje Hyper lepších výsledků.

- Algoritmus Tiling podává prakticky totožné výsledky jako algoritmus GreCon. Mírné odlišnosti jsou způsobeny odlišným pořadím průchodu konceptů/dlaždic. Z důvodů jeho pomalosti je prakticky nevhodný pro rozklad binárních matic.

- Nepřesné algoritmy mohou vypočítat velmi malé rozklady, avšak s možnou chybou. Chyba však může prakticky znemožnit rozumnou interpretaci faktoru. Příkladem je labuť zařazená mezi čtyřnohé savce se srstí.

- Algoritmy Asso a PaNDa se možnou chybou snaží potlačit vliv binárního šumu v datech. Otázkou je, nakolik dokáží potlačit skutečný šum a nakolik mohou způsobit ještě větší nepřesnosti. Výsledkem mohou být faktory s nesmyslnou interpretací.

- Celkově bych pro matice do hustoty 60% doporučil algoritmus GreEss, pro hustší matice pak algoritmus GreConD. Oba algoritmy poskytují přesný rozklad, jsou rychlé a paměťově nenáročné.

Závěr

V této práci jsem popsal problém rozkladu binárních matic včetně možnosti jeho řešení za pomocí metod pro rozklad obecných matic. Dále jsem definoval základní pojmy, které jsou potřebné pro pochopení problematiky.

Zvolené algoritmy GreCon, GreConD, GreEss, Asso, PaNDa, Hyper a Tiling jsem podrobně popsal včetně jejich teoretických základů, vlivu jejich parametrů na rozklad a implementačních aspektů. Všechny zvolené algoritmy jsem implementoval v jazyce Java a pro rychlejší experimenty poté přepsal do jazyka C.

Ve druhé části práce jsem prezentoval výsledky pokusů s algoritmy jak na reálných datech, tak i na náhodně generovaných maticích. Podrobně jsem analyzoval výsledky algoritmů s datesetem zoo a popsal problematiku generování náhodných matic obsahujících faktory.

Vedlejším produktem této práce jsou podpůrné aplikace pro zpracování výsledků, generování matic a pro vizualizaci binárních matic s výsledky rozkladů.

Conclusions

In this thesis I have described a problem of the decomposition of binary matrices including the possibilities of its solution by using methods for decomposition of the general matrices. Further I defined basic concepts, which are required for the understanding of the issue.

I have described the chosen algorithms in detail, including their theoretical bases, influence of their parametres on the decomposition and implementation aspects. I have implemented all chosen algorithms in Java language and then I have rewritten it to the C language in order to do faster experiments.

In the second part of this thesis I have presented the results of the experiments with algorithms on the real data as well as on randomly generated matrices. I have analyzed the results of algorithms with dataset zoo in detail and I described issue of generating of random matrices containing factors.

The by-products of this thesis are the support aplications for processing results, generation of matrices and for visualisation of binary matrices with the results of decomposition.

A Obsah přiloženého CD/DVD

bin/
: Zkompilovaná verze implementace algoritmů.

data/
: Použité reálné datasety, podklady pro zpracování grafů.

doc/
: Diplomová práce ve formátu PDF a ZIP soubor se zdrojovými soubory práce a s obrázky.

src/
: Kompletní zdrojové kódy všech programů použitých při vypracování této práce.

readme.txt
: Soubor s adresářovou strukturou a podrobným popisem.

Literatura

[1] AGRAWAL, Rakesh; IMIELIŃSKI, Tomasz; SWAMI, Arun. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* 1993, roč. 22, č. 2, s. 207–216. ISSN 0163-5808.

[2] BĚLOHLÁVEK, R.; TRNEČKA, M. From-Below Approximations in Boolean Matrix Factorization: Geometry and New Algorithm. *CoRR*. 2013, roč. abs/1306.4905. Dostupný také z: ⟨http://arxiv.org/abs/1306.4905⟩.

[3] BĚLOHLÁVEK, R.; VYCHODIL, V. Discovery of optimal factors in binary data via a novel method of matrix decomposition. *J. Comput. Syst.* 2010, roč. 76, s. 3–20.

[4] BERRY, M. W.; DUMAIS, S.T.; O'BRIEN, G.W. Using linear algebra for intelligent information retrieval. *Siam Review*. 1995, roč. 37, č. 583–595.

[5] *dblp: computer science bibliography*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://dblp.uni-trier.de/db/⟩.

[6] *FCALGS: Algorithms for Formal Concept Analysis*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://fcalgs.sourceforge.net/⟩.

[7] GANTER, B.; WILLE, R. *Formal Concept Analysis: Mathematical Foundations*. First. Berlin: Springer-Verlag. x, 284 s. ISBN 3-540-62771-5.

[8] GANTER, Bernhard. Two Basic Algorithms in Concept Analysis. In. *Formal Concept Analysis*. 2010, s. 312–340. Dostupný také z: ⟨http://dx.doi.org/10.1007/978-3-642-11928-6_22⟩. ISBN 978-3-642-11927-9.

[9] GEERTS, F.; GOETHALS, B.; MIELIKAINEN, T. Tiling databases. *Proc. Discovery Science*. 2004, č. 278–289.

[10] GOLUB, G.; VAN LOAN, C. *Matrix Computations*. 1996.

[11] CHENG, Hong; YU, P.S.; HAN, Jiawei. AC-Close: Efficiently Mining Approximate Closed Itemsets by Core Pattern Recovery. In. *Data Mining, 2006. ICDM '06. Sixth International Conference on*. 2006, s. 839–844.

[12] KEPRT, A.; SNÁŠEL, V. Binary factor analysis with help of formal concepts. *Proc. CLA 2004, Ostrava*. 2004, č. 90–101.

[13] KUZNETSOV, Sergei O.; OBIEDKOV, Sergei. Comparing Performance of Algorithms for Generating Concept Lattices. *JOURNAL OF EXPERIMENTAL AND THEORETICAL ARTIFICIAL INTELLIGENCE*. 2002, roč. 14, s. 189–216.

[14] KUZNETSOV, S.O. A fast algorithm for computing all intersections of objects from an arbitrary semilattice. *Nauch.-Tekh. Inf. Ser.2*. 1993, č. 1, s. 17–20.

[15] LEE, D.; SEUNG, H. Learning the parts of objects by Non-negative Matrix Factorization. *Nature*. 1999, roč. 401, č. 788–791.

[16] LINDIG, Christian. Fast Concept Analysis. In. *Working with Conceptual Structures - Contributions to ICCS 2000*. Aachen, Germany: Shaker Verlag, 2000, s. 152–161.

[17] LUCCHESE, C.; ORLANDO, S.; PEREGO, R. Mining top-K patterns from binary datasets in presence of noise. *SIAM DM*. 2010, č. 165–175.

[18] MIETTINEN, P.; MIELIKAINEN, T.; GIONIS, A.; DAS, G.; MANNILA, H. The discrete basis problem. *IEEE Trans. Knowledge and Data Eng.* 2008, roč. 20, č. 1348–1362.

[19] MYLLYKANGAS, S. et al. DNA copy number amplification profiling of human neoplasms. *Oncogene*. 2006, roč. 25, s. 7324–7332.

[20] *Nonnegative matrix factorization - MATLAB nnmf*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://www.mathworks.com/help/stats/nnmf.html⟩.

[21] *NOW - Fossil Mammal Database*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://www.helsinki.fi/science/now/⟩.

[22] OUTRATA, J.; VYCHODIL, V. Fast algorithm for computing fixpoints of Galois connections induced by object-attribute relational data. *Information Sciences*. 2012, roč. 185, č. 114–127.

[23] RISSANEN, Jorma. *Stochastic Complexity in Statistical Inquiry Theory*. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1989. ISBN 9971508591.

[24] *Singular value decomposition - MATLAB svd*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://www.mathworks.com/help/matlab/ref/svd.html⟩.

[25] STOCKMEYER, L. J. The set basis problem is NP-complete. *IBM Research Report RC5431*. 1975.

[26] *The GNU C Library: Array Sort Function*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://www.gnu.org/software/libc/manual/html_node/Array-Sort-Function.html⟩.

[27] TURING, A. M. Rounding-Off Errors in Matrix Processes. *he Quarterly Journal of Mechanics and Applied Mathematics*. 1948, roč. 1, č. 287–308.

[28] *UCI Machine Learning Repository: Data Sets*. [online]. 2015 [cit. 2015-03-17]. Dostupný z: ⟨http://archive.ics.uci.edu/ml/datasets.html⟩.

[29] XIANG, Y.; JIN, R.; FUHRY, D.; DRAGAN, F. F. Summarizing transactional databases with overlapped hyperrectangles. *Data Mining and Knowledge Discovery*. 2011, roč. 23, č. 215–251.