VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

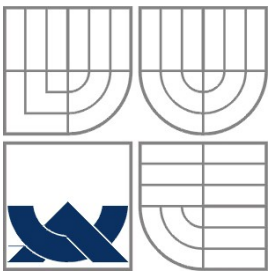
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ ALGORITMY PRO DOLOVÁNÍ ASOCIAČNÍCH PRAVIDEL

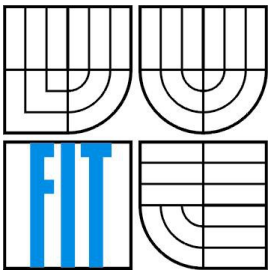
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE LUKÁŠ MATULA
AUTHOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

NOVÉ ALGORITMY PRO DOLOVÁNÍ ASOCIAČNÍCH PRAVIDEL

NEW ASSOCIATION RULE MINING ALGORITHMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ MATULA
AUTHOR

VEDOUCÍ PRÁCE — ING. VLADIMÍR BARTÍK, Ph.D.
SUPERVISOR

BRNO 2014

Abstrakt

Cílem této práce je přiblížení principu algoritmu Closure a ověření jeho vlastností. Tento algoritmus není moc známý a v problematice dolování asociačních pravidel je považován za nový. Algoritmus Closure by měl být krokem vpřed v rychlosti zpracování dat a získávání asociačních pravidel. Pro porovnání bude sloužit algoritmus Apriori, který je naopak v této problematice již dlouho využíván.

Abstract

The goal of this barchelor´s thesis is to study and implement the Closure algorithm and verification of its characteristics. The algorithm is less known than other approaches and is relatively new in the field of data mining. The Closure algorithm could be a step forward to fast data processing in data mining and knowledge discovery. The thesis also compares the Closure algorithm with the Apriori algorithm. The Apriori algorithm is well known in the field of data mining.

Klíčová slova

asociační pravidla, dolování dat, Apriori, Closure, relační databáze, transakční databáze, minimální podpora, frekventované vzory

Keywords

association rules, data mining, Apriori, Closure, relational database, transactional database, minimal support, frequent patterns

Citace

Lukáš Matula: Nové algoritmy pro dolování asociačních pravidel, bakalářská práce, Brno, FIT VUT v Brně, 2014

Nové algoritmy pro dolování asociačních pravidel

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Vladimíra Bartíka.Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Lukáš Matula
13.5.2014

Poděkování

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Vladimíru Bartíkovi, Ph.D. za odborné vedení, vstřícnost, za pomoc a rady při zpracování této práce.

Obsah

Obsah....1
1 Úvod....3
2 Problematika dolování asociačních pravidel....4
2.1 Získávání znalostí z databází....4
2.1.1 Klasifikace....5
2.1.2 Predikce....5
2.1.3 Shlukování....6
2.2 Druhy dat pro dolování....6
2.2.1 Relační databáze....6
2.2.2 Datové sklady....7
2.2.3 Transakční databáze....9
2.3 Asociační pravidla....9
2.3.1 Metriky zajímavosti pravidla....10
2.3.2 Typy asociačních pravidel....10
2.3.3 Asociační pravidla založená na omezeních....10
3 Algoritmy....11
3.1 Apriori....11
3.1.1 Možnosti pro zvýšení efektivity algoritmu Apriori....12
3.1.2 Pseudokód algoritmu Apriori....13
3.2 Metoda FP-stromu....14
3.3 Closure....15
3.3.1 Pseudokód algoritmu Closure....16
4 Implementace....19
4.1 Konceptuální návrh aplikace....19
4.2 Popis jednotlivých tříd....20
4.2.1 Aplikace....20
4.2.2 Homepage....20
4.2.3 Apriori....20
4.2.4 Closure....20
4.2.5 Seznam....21
4.2.6 Prvek....21
4.2.7 Podseznam....21
4.2.8 Podprvek....21

4.3 Práce s aplikací....21
4.3.1 Připravení prostředí pro práci s aplikací....21
4.3.2 Snímek vzhledu aplikace....22
4.3.3 Popis aplikace....22
5 Ověřování funkčnosti algoritmů....23
5.1 Datové zdroje....23
5.2 Provádění ověřování....24
5.2.1 Zjišťování doby běhu algoritmů bez generování asociačních pravidel....24
5.2.2 Zjišťování doby běhu algoritmů s generováním asociačních pravidel....26
5.2.3 Zjišťování počtu průchodů zdrojovými databázemi....27
6 Závěr....30
Literatura....31
Seznam příloh....32
Příloha č.1....33
Příloha č.2....34

1 Úvod

Práce se zabývá problematikou získávání asociačních pravidel z databází v oboru informačních technologií. Asociační pravidla nám jednoduše řečeno říkají, co s čím souvisí a v jakém vztahu se navzájem dvě a více věcí vyskytují. Například už jen pravidelnou návštěvou supermarketu a nákupem zboží se vytváří asociační pravidla pro nákupní košík. Další téma, na které je bakalářská práce zaměřena se nazývá data mining (dolování dat). Dolování dat je proces, pro vytvoření modelů dat z databází.

Cílem této bakalářské práce je vybrat jeden z novějších algoritmů provádějící automatizovaný proces dolování dat a tento algoritmus zanalyzovat, naimplementovat, ověřit jeho činnost za pomocí experimentů nad vzorkem dat a výsledky srovnat se známým algoritmem Apriori. Pro tuto práci byl vybrán algoritmus s názvem Closure algorithm. Hlavním testovaným faktorem jsou doba zpracování dat algoritmu a počet průchodů zdrojovou databází. Podmínkou ověřování funkčnosti algoritmů je dosažení správného výsledku.

Nyní následuje krátké srhnutí obsahu této práce. Druhou (jelikož úvod je první kapitolou) kapitolou je problematika dolování asociačních pravidel a vysvětlení základních pojmů z tohoto oboru. Další kapitola se zabývá jednotlivými kroky celého procesu získávání znalostí z databáze. Jedním z kroků je i proces dolování dat z databáze a získávání asociačních pravidel. Následuje představení pojmů klasifikace, predikce a shlukování. Popsání zdrojů dat pro dolování, do kterých patří relační databáze, datové sklady a transakční databáze, které se použijí v aplikaci pro ověření funkčnosti algoritmů. V kapitole asociační pravidla je popsáno jejich využití, použití metrik pro tyto pravidla a jsou uvedeny i typy asociačních pravidel. Následuje kapitola algoritmy a v ní představení tří algoritmů pro automatizované provedení procesu dolování. Prvním z nich je metoda FP-stromu, druhým je algoritmus Apriori a třetím je algoritmus Closure. Součástí posledních dvou algoritmů jsou i pseudokódy, podle kterých se naimplementuje činnost těchto algoritmů v ověřovací aplikaci. Implementace je praktická kapitola jejíž součástí je koncepční návrh aplikace a popsání jednotlivých tříd. Další praktickou kapitolou je ověření činnosti algoritmů a prezentování výsledků pomocí grafů se stručným popisem a poukázáním na důsledky různých vlastností algoritmů. V kapitole závěr se popisují získané znalosti a hodnotí se celkový výsledek práce.

2 Problematika dolování asociačních pravidel

Dolování (získávání) asociačních pravidel je část celého procesu získávání znalostí z databází. Po rozdělení tohoto pojmu na části lépe pochopíme jeho význam. Dolování je jiným slovem získávání. Slovo asociační (asociace = vztah, vazba), takže vztahy jednotlivých dat. Pravidla jsou opakující se výskyty vazeb. Po složení těchto částí dohromady, získáme definici, že dolování asociačních pravidel je získávání modelů dat. Jako datový zdroj při provádění procesu dolování se používají relační databáze, datové sklady (víceúrovňová asociační pravidla), transakční databáze (jednoúrovňová asociační pravidla), další zdroje jsou popsány v kap. 2.2 Druhy dat pro dolování. Celá kapitola 2, včetně podkapitol byla čerpána ze zdroje [1].

2.1 Získávání znalostí z databází

Získávání znalostí z databáze je analytická metodologie, extrakce („dolování“) netriviálních, skrytých, potencionálně užitečných modelů dat a vzorů z velkého objemu dat. Proč zrovna netriviálních, skrytých a potencionálně užitečných modelů dat? Netriviální znamená, že informaci nelze získat jen pomocí jednoduchého SQL dotazu nad daty, ale musí se použít specifický sofistikovaný postup. To souvisí s další vlastností a tou je skrytost. Modely a vzory, které jsou hledány nejsou na první pohled vidět, protože databáze nebyla navržena proto, aby tyto informace přímo ukládala. Třetí vlastností je potenciální užitečnost. Tato vlastnost je závislá na zisku informací pro potřebu rozhodnutí. Setkáváme se s ní v praxi při analýze zákazníka žádajícího půjčku v bance, nebo při rozhodování rozmístění zboží v hypermarketu, apod.

S aplikací získávání znalostí z databází je možno se setkat v běžném životě. Hodně rozšířená a ve velkém počtu používaná je analýza nákupního košíku. U ní se např. zjišťuje jaké zboží v jakém období zákazníci nejvíce nakupují. Další oblast s velkým počtem využití jsou finanční analýzy a řízení rizik (predikce vývoje cen v čase, predikce rizika poskytnutí půjčky). Praktické využití se nachází i při detekci podvodů a neobvyklého chování. Za pomocí získávání znalostí z textu se detekují příchozí emailové zprávy od zájmových skupin, nebo reklamních společností a označují je jako spam. Jednou z oblastí, kde se nachází obrovské množství dat vhodných pro analýzu je rychle rozvíjející se bioinformatika a analýza biologických dat.

Získávání znalostí z databází se také velmi často vyskytuje pod názvem Dolování dat (data mining). Ve skutečnosti je dolování dat pouze jednou ze 7 částí celého procesu získávání dat z databází. Skládá se z několika kroků:

- **Čištění dat** – vyřešení chybějících dat a vyřešení nekonzistence.
- **Integrace dat** – provádí se společně s čištěním dat, protože se zpravidla berou data z více zdrojů, a tak je potřeba sjednotit je do datových skladů.
- **Výběr dat** – vybereme data, která jsou pro naše řešení užitečná (relační databáze – sloupce z tabulek, datové sklady – dimenze).
- **Transformace dat** – je potřeba transformovat data do ustálené podoby vhodné pro dolování.
- **Dolování dat –** hlavní částí procesu získávání znalostí z databáze, ve které pomocí vybrané metody a patřičného algoritmu vyextrahuje vzory ze zdrojových dat → vytvoří model dat.
- **Hodnocení modelů a vzorů** – snaha o výběr užitečných vzorů dat.
- **Prezentace znalostí –** prezentování výsledku uživateli (technikami vizualizace) z dolování dat.

2.1.1 Klasifikace

Klasifikace je pojem, který se vyskytuje u procesu dolování dat. Je chápána jako proces, kdy se dané objekty zařazují dle svých vlastností do jistých tříd. Častým příkladem pro klasifikaci a predikci je poskytnutí půjčky zákazníkovi. Např. Přijde-li zákazník do banky požádat o půjčku. Banka si nejprve zjistí od zákazníka informace o jeho zaměstnání a hlavně o výši mzdy. Vyhledá v záznamech zda-li už nemá o zákaznicích s podobnými vlastnostmi nějaké informace z dřívější doby a následně ho zařadí do odpovídající třídy. Pro dané třídy už mají banky přichystány podmínky pro zákazníka.

2.1.2 Predikce

Predikce je pojem hodně spojovaný s pojmem klasifikace. Predikce je proces, který umožní na základě vlastností objektu přiřadit danému objektu jisté hodnoty. Přiblížíme si tuto definici opět na příkladu. Do zaměstnání přijde nový zaměstnanec a zaměstnavatel se sním musí domluvit na mzdě. Jelikož zaměstnanec má v daném oboru odpracováno mnoho let, tím pádem má hodně zkušeností. Pro zaměstnance je mnohem více atraktivní, než absolvent v tomto oboru. Na základě těchto vlastností mu zaměstnavatel určí vyšší mzdu než absolventovi.

2.1.3 Shlukování

Shlukování je proces rozdělování objektů na základě podobnosti do tříd. Pravidlem pro rozdělování objektů je, že v jedné třídě si objekty musí být co nejvíce podobné, ale zároveň musí být pokud možno co nejvíce odlišné od objektů jiných tříd. Se shlukovou analýzou se setkáváme při rozdělení osob na muže a ženy, nebo objektů v přírodě na živočichy a rostliny, apod. Shlukování je používané jako předzpracování pro klasifikační algoritmy. Ve srovnání s klasifikací má shlukování výhodu v tom, že se předem nemusí znát žádné předdefinované třídy ani žádné trénovací množiny, ale nevýhodou je, že vytvořené shluky nemají předem daný význam - ten se jim musí přiřadit, až následně - ale ne vždy se to podaří.

2.2 Druhy dat pro dolování

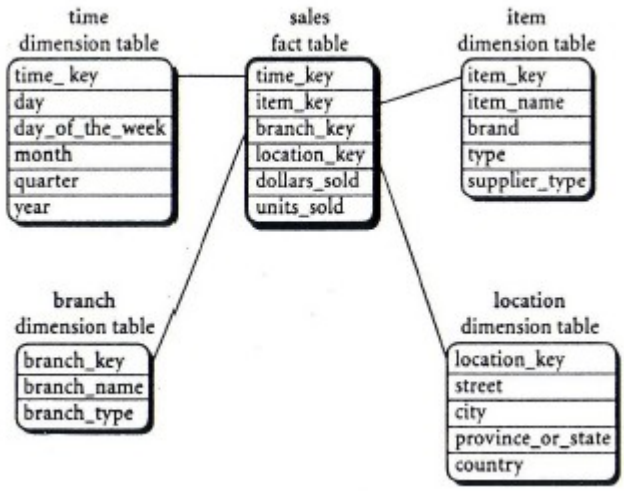
Pro data určená k dolování není definován přesný formát v jakém musí být uložena. To nám umožňuje dolovat data z různých typů zdrojů dat. Jediná podmínka, která však musí být dodržena je, aby data byla perzistentní (uložená v datových úložištích) nebo transientní (jedná se o proudy dat). Nejvíce používaným zdrojem dat pro dolování bývají relační databáze. Je to způsobeno tím, že jsou v dnešní době nejrozšířenějším úložištěm pro data. Dalším poměrně častým nositelem dat vhodným pro dolování je datový sklad, u kterého se data ukládají do datových kostek. Třetím používaným zdrojem dat jsou transakční databáze. Mezi další zdroje dat se řadí objektově-relační databáze, temporální databáze, databáze sekvencí, databáze časových řad, prostorové databáze, textové databáze, multimediální databáze, heterogenní databáze, zděděné databáze, již zmíněné proudy dat a jako poslední zdroj je web.

2.2.1 Relační databáze

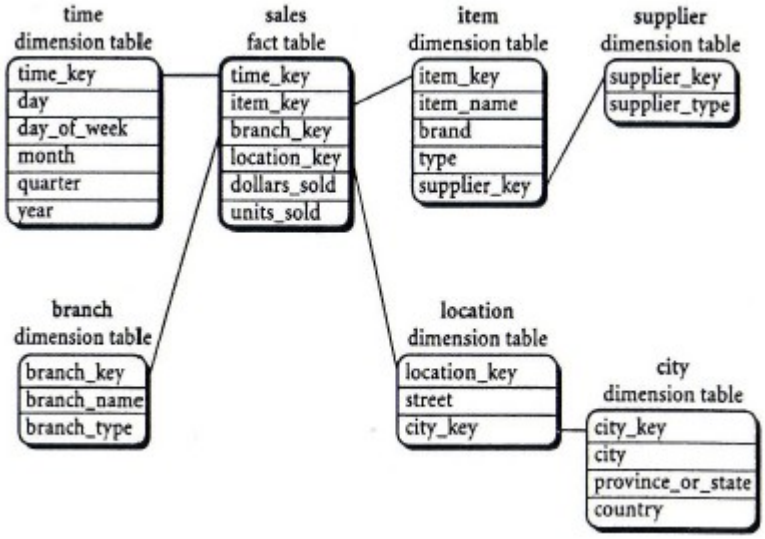
Nejrozšířenějším úložištěm dat je v dnešní době relační databáze. Relační databáze je kolekce tabulek, které znázorňují relace. Relace je dvourozměrná datová struktura tvořena záhlavím a tělem. Záhlaví obsahuje domény a v těle jsou odpovídající hodnoty k těmto doménám. V relačních databázích platí určitá pravidla. Říká se jim normální formy. Aby relační databáze mohla být použita jako zdroj dat, musí příslušná databáze splňovat podmínku první normální formy, to znamená, že data v jednotlivých sloupcích musí být atomická. Relační databáze jsou uzpůsobeny ke každodenní práci. Mezi základní operace, které se provádí nad touto databází patří vytváření nových tabulek (entit reálného světa), jejich úpravy, odstraňování a práce s daty (vkládání, upravování, odstraňování a dotazování na ně). Pomocí SQL dotazování nad relační databází je možno snadno získat triviální informace. Relační databáze patří do systému OLTP (On-line transaction processing). Tyto systémy jsou určeny pro zákazníka. Zákazník s nimi pracuje ve formě různých aplikací, kde si ani neuvědomuje, že vlastně provádí dotazování na základě vložených kritérií nad databází.

2.2.2 Datové sklady

Datový sklad je datové úložiště, u kterého nedochází k častým změnám údajů. Změny dat se zde provádí vyjímečně. Proto se taky říká, že je tvořen z historických dat, většinou pomocí sumarizací a agregací. Operace, které se provádí nad těmito úložišti jsou vytvoření datového skladu, naplnění jej daty a následně se provádí jen analýza. Oproti relační databázi se při analýze těchto dat získává vyšší úroveň informací. Příkladem může být relační databáze, kde se uchovávají informace o prodeji. Jaká položka byla prodána, ve který den, hodinu, minutu, kdo ji prodal, apod. Ale v datovém skladu se pak uchovávají jen některé informace, např. jaká je prodávanost dané položky ve všech prodejnách za určité období. Datové sklady jsou určeny pro analytiky, protože jim usnadňují práci s analýzou dat. Je tvořen multidimenzionálními datovými kostkami. Data jsou v nich uložená ve formě dimenzí. Jako dimenzi máme na mysli například entitu nebo pohled (uloženy ve formě tabulky). V databázích se tyto multidimenzionální modely ukládájí ve formě schémat. Nejpoužívanější schéma je schéma hvězdy, které je vidět na obrázku č.1. Základem tohoto schématu je tabulka faktů obsahující data bez redundance. Další tabulky už obsahují informace o jednotlivých dimenzích. Data jsou v nich už redundantní. Na obrázku č.2 je možno vidět další typ schématu pro multidimenzionální modely a tím je schéma sněhové vločky. Používá se méně častěji než předchozí schéma a to díky tomu, že je náročnější pro vytváření dotazů (musí se spojovat více tabulek). Oproti hvězdě je zde snaha snížit výskyt redundance v tabulkách dimenzí. Důsledek této snahy je zvýšení počtu tabulek. Máme datovou kostku o třech dimenzích, zákazník, položka a čas. Data jsou archivována 5 let. Nyní se analyzují data podle těchto tří dimenzí a vyvozují se z toho patřičné výsledky. Datové sklady patří do systému OLAP (On-line analytical processing). Systémy OLAP jsou orientovány na obchod a používají se pro analýzu prováděnou analytiky. Tyto systémy mají i své operace roll-up a drill-down, pomocí kterých se z dat získávají informace. Operace roll-up vytváří abstraktnější pohled na data, z měst uložených v dimenzi tak vytvoří např. kraje, nebo okresy. Operace drill-down provádí opak, snaží se získat více detailní informace z dat v rámci dimenze (z časového období čtvrtletí přejde na jednotlivé měsíce). Datový sklad se používá i jako zdroj dat pro dolování, což je hlubší, automatizovaně prováděná analýza, na rozdíl od OLAP operací. Na závěr si ukažme ještě několik rozdílů mezi OLTP (kde patří relační databáze) a OLAP systémy. Velikost těchto systémů je velmi rozdílná. Zatímco velikosti OLTP se pohybují od stovek MB do jednotek GB, u OLAP systému se pohybuje nejčastěji od stovek GB po TB. Další velký rozdíl je i v počtu uživatelů, kteří pracují s těmito systémy. U OLTP se jedná o tisíce uživatelů, jsou to OLAP systémy pracují desítky až stovky uživatelů.



Obr. 2.1 Schéma hvězdy [1]



Obr. 2.2 Schéma sněhové vločky [1]

2.2.3 Transakční databáze

Je třetím nejpoužívanějším zdrojem dat pro dolování dat a získávání asociačních pravidel. Data jsou uložena ve formě dvourozměrných tabulek, identifikátor transakce a název položky. Pokud se mluví o transakcích tak je na mysli obchodní transakce. Pro představu, jedna transakce může být jeden nákup jednoho zákazníka v obchodě. Tato data z hlediska normalizačních pravidel pro relační databáze jsou nenormalizována. Nejtypičtějším příkladem pro dolování dat z transakčních databází je analýza nákupního košíku, která je prováděna i v této bakalářské práci. Základním cílem analýzy je zjistit, jaké zboží se nejvíce prodává s jiným zbožím, a jak často k této kombinaci dochází. Na základě výsledků z příkladu analýzy nákupního košíku je možno provádět akční nabídky na různé zboží, s tím že s nějvětší pravděpodobností si zákazník ke zboží v akci přikoupí i zboží neslevněné, které nejčastěji nakupuje se zbožím v akční nabídce. Tato získaná znalost o vzájemné vazbě určitých druhů zboží se může hodit ke strategickému rozmístění těchto druhů zboží po prodejně. Jako příklad si uvedeme zákazníka, který velmi často nakupuje rohlíky a pomazánky. Bude tedy vhodné umístit na začátek prodejny pečivo a na konec prodejny pomazánky. Tím pádem musí zákazník projít celou prodejnu a s velkou pravděpodobností koupí i to co neměl v plánu. A tím může být krok se strategickým rozmístěním zboží považován jako úspěšný.

2.3 Asociační pravidla

Asociační pravidla přináší zajímavé asociace a korelace z velkého množství datových položek. Jsou podporou při rozhodování na základě analýzy. Nejčastějším příkladem pro získávání asociačních pravidel je analýza nákupního košíku. Cílem je získat souvislosti v nákupech. Jaké zboží se nejčastěji vyskytuje v nákupním košíku s jiným zbožím? Jaká je pravděpodobnost, že při nákupu jednoho zboží si zákazník koupí i jiné zboží. Na základě těchto poznatků si obchodník může efektivně rozmístit zboží v prodejně, vytvořit reklamu nebo akční nabídku.

Získávání asociačních pravidel probíhá ve dvou krocích:

1. Nalezení frekventovaných množin, tj. množin splňující podmínku minimální podpory.
2. Generování silných asociačních pravidel z frekventovaných množin. Pravidla musí splňovat podmínku minimální podpory a spolehlivosti.

2.3.1 Metriky zajímavosti pravidla

Podpora procentuálně říká, v kolika případech nastala požadovaná kombinace (zákazník si koupil zboží A a zároveň zboží B).

Spolehlivost procentuálně udává, v kolika případech, pokud nastala pouze z části požadovaná kombinace, tak nastala i druhá část kombinace (kolikrát si zákazník koupil i zboží B když si kupoval zboží A).

2.3.2 Typy asociačních pravidel

- **Podle typu hodnot v pravidlech:**
  - jedná se pouze o přítomnost/nepřítomnost položky – boolovská asociační pravidla
  - pokud popisuje asociace mezi kvantitativními položkami – kvantitativní asociační pravidla
- **Podle dimenzí obsažených v pravidlech:**
  - boolovská asociační pravidla jsou jednodimenzionální
  - kvantitativní asociační pravidla jsou vícedimenzionální
- **Podle úrovní abstrakce v pravidlech:**
  - některé metody jsou schopny získávat asociační pravidla nad různými úrovněmi, říkáme jim víceúrovňová pravidla.
- **Podle dalších rozšíření asociačních pravidel:**
  - maximální vzory
  - korelační analýza
  - uzavřené množiny

2.3.3 Asociační pravidla založená na omezeních

Jedním z častých a velmi nepříjemných problémů při generování asociačních pravidel je jejich množství. Přitom z tohoto množství pravidel nás zajímá jen několik málo zajímavých, které nám přináší něco nového. Tento problém můžeme vyřešit nastavením následujících typů omezení:

- Datová omezení (určují, která data budou použita pro dolování)
- Omezení dimenzí/úrovní dat (určují dimenze dat, které budou použity pro dolování)
- Omezení zajímavosti (určení prahů pro metriky zajímavosti, např. podpora a spolehlivost)
- Omezení pravidel (určuje tvar výsledných pravidel, pomocí metapravidel)

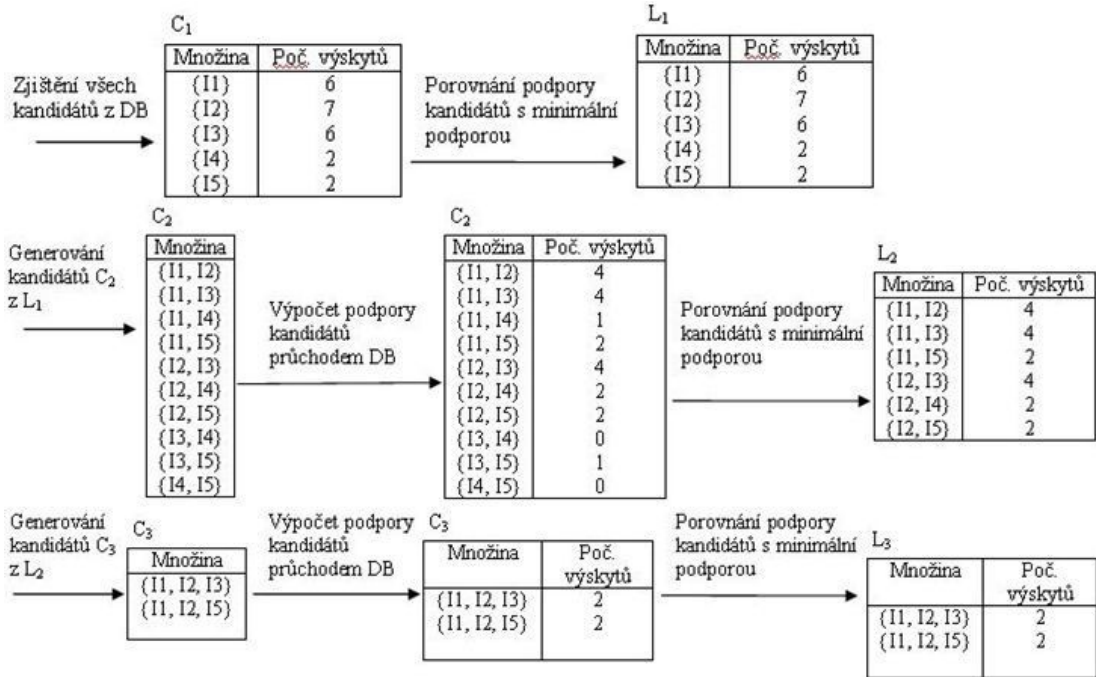
3 Algoritmy

V podkapitolách se zaměřuji na vlastnosti algoritmů a jejich postupy při zpracování dat. U algoritmů Apriori a Closure jsem zařadil i pseudokódy z implementace v testovací aplikaci.

3.1 Apriori

Algoritmus Apriori patří do skupiny boolovských jednoúrovňových algoritmů, u kterých se využívá vlastnosti zda položka existuje či nikoli. Je základním algoritmem pro generování asociačních pravidel, ze kterého se dále pomocí určitých vylepšení odvozují další algoritmy. Jeho velkou nevýhodou je počet průchodů celou zdrojovou databází. Využívá předchozí znalost o dříve získaných frekventovaných množinách, tzn. v každé iteraci se získané frekventované k-množiny použijí pro generování (k+1)-množin. Platí **pravidlo, že po přidání prvku k množině nesmí vzrůst její podpora** [1]. Algoritmus Apriori je vhodný pro data, u kterých se vyskytují asociační pravidla v množinách o malém počtu prvků. Algoritmus tvoří dva základní kroky:

1. **Spojovací** – množiny, které vyhoví minimální podpoře, se rozšíří o prvek z předcházejícího kroku iterace.
2. **Vylučovací** – proběhne výpočet podpory kandidátů, podpory se porovnají s minimální podporou a nefrekventované množiny se odstraní.



Obr. 3.1 Ukázka postupu algoritmu Apriori [1]

Pro generování asociačních pravidel z frekventovaných množin se používá vztah pro výpočet spolehlivosti:

$$conf(A \Rightarrow B) = P(B|A) = \frac{s(A \cup B)}{s(A)} \qquad (3.1)$$

Výsledná podoba asociačního pravidla v příkladu vypadá takto:

$$I1 \wedge I2 \Rightarrow I5 ; c = 2/4 = 50\% \qquad (3.2)$$

Postup při generování asociačních pravidel:

- Pro každou frekventovanou množinu l, vygeneruj všechny její neprázdné podmnožiny.
- Pro každou neprázdnou podmnožinu s, vygeneruj pravidlo s => (l - s) a podle rovnice vypočti jeho spolehlivost. Vypočtenou spolehlivost porovnej s minimální spolehlivostí a urči, která pravidla jsou silná.

3.1.1 Možnosti pro zvýšení efektivity algoritmu Apriori

Hašovací přístup

Když se prochází na začátku zdrojovou databázi a probíhá získávání jednoprvkových množin, může algoritmus generovat rovnou všechny dvouprvkové množiny pro každou transakci a přiřadit je na „hromádky“ hašovací tabulky. U každé hromádky bude uchováván počet množin.. Pokud tento počet bude nižší než minimální podpora, tak rovnou se hromádka odstraní. Zamezí se tak průchodu zbytečnými množinami, které stejně k výsledku nevedou.

Redukce transakcí

Pokud transakce nedisponuje žádnou frekventovanou k-množinou, je jisté, že nebude obsahovat žádnou (k+1)- množinu. Tím pádem tato transakce může být smazána.

Rozdělení dat

Nejprve se databáze rozdělí na několik částí. V každé z těchto částí se zjistí lokální frekventované množiny a z nich jsou vybrány globální frekventované množiny. Velkou výhodou tohoto zefektivnění je počet průchodů zdrojovou databází, který je na hodnotě 2.

Vzorkování

Při vzorkování se použije náhodně vybraný vzorek dat z databáze a v něm se vyhledají frekventované množiny. Vzorek může být nahrán do hlavní paměti, ale tím se sníží přesnost. Tu je možno zvýšit snížením požadované minimální podpory. Výsledkem je, že jsou získány i množiny, které nejsou frekventované. Nakonec se musí u získaných frekventovaných množin znovu prověřit podpora na celé databázi.

Dynamické počítání množin

Počítání kandidátů probíhá v průběhu čtení z databáze. U základní verze algoritmu Apriori se provádělo počítání kandidátů před průchodem databází.

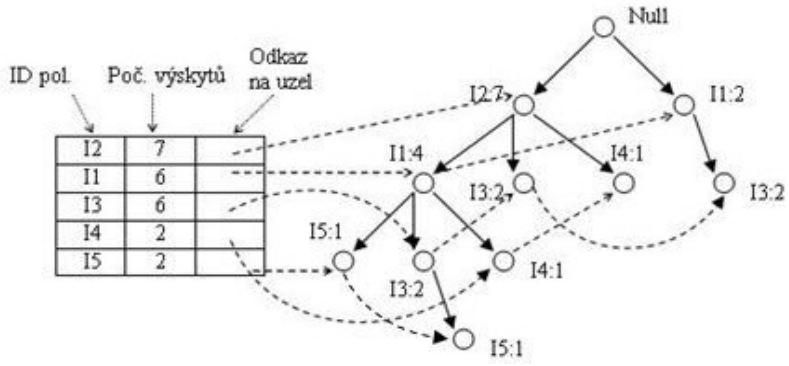
3.1.2 Pseudokód algoritmu Apriori

```
Seznam l,c;
index = 1;
naplň seznam c jednoprvkovými množinami z databáze;

/*pro jednotlivé prvky seznamu c, za pomocí sql dotazů nad databází
zjisti hodnoty jejich podpory*/
c.spoctiPodporu();


//cyklus naplní seznam l frekventovanými prvky ze seznamu c
while(c.getPrvek(i) != null){
  if(c.getPrvek(i).getPodpora() >= podpora){
    l.pridej(c.getPrvek(i));
  }
  i++;
}
index++;

//cyklus pro index > 1
while(l != null){
  c.vymaz();   //vymaže celý seznam c

  //do seznamu c se vloží nové množiny o počtu n+1 prvků
  c = l.rozgeneruj();

   /*pro jednotlivé prvky seznamu c, za pomocí sql dotazů nad
   databází zjisti hodnoty jejich podpory*/
  c.spoctiPodporu();
  l.vymaz();//vymazání celého obsahu seznamu l
  i=0;

    //cyklus pro naplneni seznamu l frekventovanými prvky
    while(c.getPrvek(i) != null){
      if(c.getPrvek().getPodpora >= podpora){
        l.pridej(c.getPrvek());
```

```
        }
        i++;
      }
      index++;
}

/*generování asociačních pravidel z podseznamů uložených v seznamu l
ve tvaru x → y, kde za x a y musí být vždy minimálně jeden prvek
podseznamu*/
i=0;
while(l.getPrvek(i)!= null){

  vytvořím veškeré kombinace prvků z l.getPrvek(i).getPodseznam;
  kombinace = ulož první kombinaci prvků z podseznamu;

  //cyklus projde všechny kombinace, končí pokud se kombinace==null
  while(kombinace != null){
    spočti hodnotu spolehlivosti levé strany pravidla;
    if(spolehlivost pravidla >= minimální spolehlivosti){
      vytiskni pravidlo na výstup aplikace;
    }
  }
  i++;
}
```

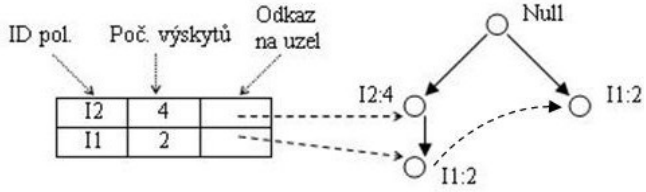
3.2 Metoda FP-stromu

Metoda FP-stromu, jinak řečeno metoda vzrůstu frekventovaných množin, je aktuálně považována za nejefektivnější algoritmus pro dolování asociačních pravidel. Řeší některé problémy vyskytující se u algoritmu Apriori. Prvním z problémů je velké množství generovaných kandidátů, zvláště u velkých databází. Druhým problémem je neustálé procházení zdrojové databáze, což je největší časovou zátěží při provádění algoritmu. Algoritmus je složen ze tří fází. První fází je komprese databáze do stromové struktury množin, zvané FP-strom.[4] Druhou fází je rozdělení stromu do podmíněných FP-stromů. Podmíněné FP-stromy se vytváří pro každou frekventovanou položku. Třetí fází je získání frekventovaných množin z těchto stromů. Na obr. 3.2 je možno vidět tabulku jednoprvkových množin s jejich podporou ve zdrojové databázi a k ní vytvořený odpovídající FP-strom. Přerušované šipky znázorňují souvislost jednotlivých uzlů.



Obr. 3.2 metoda FP-stromu [1]

Na dalším obr. 3.3 je znázorněná ukázka podmíněného FP-stromu pro frekventovaný prvek I3. Při pohledu na podmíněný FP-strom je jasné, že získané frekventované množiny jsou (za dvojtečkou je frekventovanost): **<I2 I3>: 4, <I2 I1 I3 >: 2, <I1 I3>: 2**.



Obr. 3.3 podmíněný FP-strom položky I3 [1]

3.3 Closure

Algoritmus Closure (uzavírací) je jeden z novějších algoritmů. Používá se u něj odlišný postup zpracování dat než u algoritmu Apriori a to shora dolů. Právě díky tomuto odlišnému způsobu zpracování má algoritmus předpoklad být rychlejší než algoritmus Apriori [2]. S rychlostí zpracování dat a získání správného výsledku souvisí hlavně počet průchodů zdrojové databáze, který je v ideálním případě téměř poloviční oproti Apriori. Důvod je jednoduchý, provedení průchodu celou databází je časově nejnáročnější instrukce z celého algoritmu. Proto ji potřebujeme co nejvíce redukovat. Každý algoritmus má i své nevýhody, u tohoto algoritmu je patrné už z jeho konstrukce, že pro databáze, u kterých je frekventovanost splněna pouze v množinách o malém počtu prvků bude dosahovat horších výsledků než algoritmus Apriori. Naopak pokud se použije v databázi, kde se předpokládá splnění podpory ve víceprvkových množinách, by měl algoritmus být velkým přínosem.

3.3.1 Pseudokód algoritmu Closure

```
/*oba dva druhy seznamu jsou  jednosměrně vázané, ale obsahují jiné
druhy položek*/
Seznam jednoprvek, fronta;
Podseznam frekvent, nefrekvent;

/*obě třídy charakterizují prvky seznamu, ale mají rozdílný počet
atributů i rozdílné typy atributů*/
Prvek polozka;
Podprvek podpolozka;

naplň seznam jednoprvek jednoprvkovými množinama z databáze;

/*pro každou položku seznamu vypočítej její podporu ze zdrojové
databáze*/
jednoprvek.spoctiPodporu();
polozka = jednoprvek.dejPrvniPrvek();

/*cyklus prochází seznamem jednoprvkových množin,a určuje zdali je
prvek frekventovaný nebo nefrekventovaný.
Cyklus končí svůj běh, pokud zpracuje poslední prvek v seznamu */
while(polozka != null){

   /*pokud je položka frekventovaná, tak se vytvoří příslušný
   podseznam a do položky se vloží odkaz na něj*/
   if(polozka je frekventovaná){

      //přidej položku do frekventovaného seznamu
      frekvent.pridej(polozka.getText());
      vytvoř podseznam k této položce;
      zjisti ve kterých transakcích se tato položka vyskytuje;
      vlož všechny položky ze všech těchto transakcí do
   vytvořeného podseznamu;
   }
   else{
      //vložení položky do seznamu nefrekventovaných množin
      nefrekvent.pridej(polozka.getText());
   }
   //do prvku položka se vloží odkaz na následující prvek
   polozka = polozka.dejdalsi();
}

//cyklus beží dokud nejsou zpracovány všechny položky jednoprvkového
seznamu
polozka = jednoprvek.dejPrvniPrvek();
while(polozka != null){

   if(polozka je frekventovana){
     vytvoř řetězec z celého podseznamu této polozky;

     if(obsahuje řetězec nějaký prvek z nefrekvent. seznamu){
       odstraň tento prvek z řetězce;
       výsledný řetězec ulož do fronty;
```

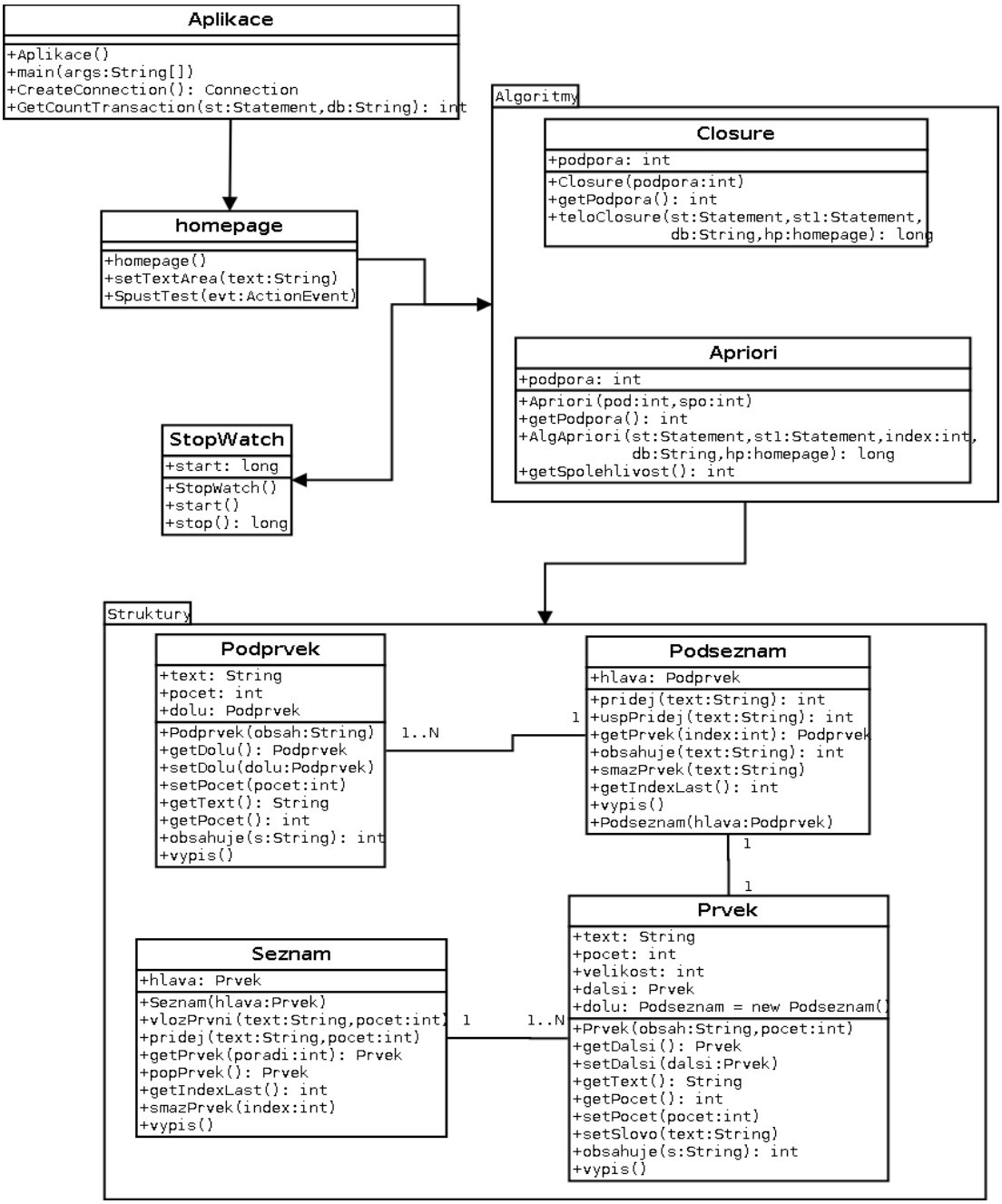
```
  }
  else{
    vytvoř z řetězce,tj. z množiny prvků sql dotaz;
    zjisti podporu celé množiny provedením sql dotazu nad
    databází;
  }

  //když je množina frekventovaná, vlož ji do seznamu frekvent
    if(je dotazovaná množina frekventovaná){
      frekvent.pridej(mnozina);
  }
  else{
    nefrekvent.pridej(mnozina);

    //rozgenerováním se myslí vytvoření množin o n-1 prvcích
    rozgeneruj tuto mnozinu do fronty;
  }
}

//cyklus beží dokud není seznam s názvem fronta prázdný
polozka = fronta.vyjmiPrvni();
while(polozka != null){
  vytvor retezec z podprvku podseznamu polozky;

    if(obsahuje retezec nejake prvky z nefrekvent seznamu){
    odstraň ty prvky z retezce;

    //řetězec se rozdělí na jednotlivé prvky a ty se uloží
    řetězec ulož do podseznamu u aktuální polozky;

    //přejdi na další krok cyklu
    continue;
  }
  else{
    vytvoř z tohoto řetězce, tj. Z množiny prvků sql dotaz;
    zjisti pomocí provedení tohoto dotazu podporu v databázi;
  }

  //ptáme se na množinu prvků, nad kterou byl proveden dotaz
  if(je řetězec frekventovaný){
    frekvent.pridej(retezec);
  }
  else{
    /*pokud už při vkládání jsme zjistili, že řetězec se už
      vyskytuje v nefrekventovaném seznamu, vyjmeme první prvek z
      fronty, uložíme jej do proměnné položka a cyklus
      spustíme nad touto položkou*/
       nefrekvent.pridej(retezec)
    if(retezec uz existuje v nefrekvent){
       polozka = fronta.vyjmiPrvni();
    continue;
    }
    else{
```

```
        /*vytvoříme z řetězce nové podřetězce s n-1 prvky a ty
uložíme do fronty*/
          řetězec rozgeneruj do fronty;
        }
      }
      //vyjmi první položku z fronty a ulož ji do proměnné položka
      polozka = fronta.vyjmiPrvni();
    }
    polozka = jednoprvek.dejDalsi();
}

na výstup aplikace vytiskni řetězce o největším počtu prvků;

//generování asociačních pravidel z řetězců o maximálním počtu prvků
while(řetězec obsahuje maximální počet prvků){

   /*podmínka pravidla je aby v každém pravidle byl vlevo a vpravo
    vždy alespoň jeden prvek*/
  vytvořím veškeré kombinace prvků v řetězci;
  kombinace = ulož první kombinaci prvků z řetězce;

 //cyklus projde všechny kombinace, končí pokud je kombinace prázdná
  while(kombinace != null){
      if(je levá část pravidla uložená v seznamu frekvent){

      spočti spolehlivost ze znalosti podpory levé části pravidla;

      /*pokud spolehlivost pravidla není menší jak minimální
    spolehlivost, tak se pravidlo vytiskne na výstup aplikace*/
      if(spolehlivost pravidla >= minimální spolehlivosti){
        vytiskni pravidlo na výstup aplikace;
      }
    }
    else{
      vytvoř sql dotaz k získání podpory levé části pravidla;
      proveď tento dotaz nad databází;
      spočti spolehlivost ze znalosti o hodnotě podpory;

      if(spolehlivost pravidla >= minimální spolehlivosti){
        vytiskni pravidlo na výstup aplikace;
      }
    }

    //pokud už žádná kombinace neexistuje tak uloží hodnotu null
    kombinace = ulož další kombinaci prvků z řetězce;
  }

}
```

4 Implementace

Následuje praktická část a v ní je první kapitolou grafická podoba konceptuálního návrhu aplikace.

4.1 Konceptuální návrh aplikace



Obr. 4.1 Konceptuální návrh testovací aplikace

4.2 Popis jednotlivých tříd

4.2.1 Aplikace

Je hlavní třídou celé aplikace a z jejího těla se volá třída homepage, která tímto spouští formulář pro komunikaci s uživatelem. Dalším úkolem třídy Aplikace je navázání spojení s databází.

4.2.2 Homepage

Homepage je třídou tvořící grafický vzhled aplikace. Jedinou akcí v této třídě je po stisknutí tlačítka Spustit test provedení testu vybraného algoritmu nad zadanou databází s příslušnou minimální podporou a spolehlivostí. Nakonec se po ukončení činnosti algoritmu výsledek vypíše do textové oblasti.

4.2.3 Apriori

Při implementaci této třídy jsem se držel pseudokódu, který je popsán v návrhu tohoto algoritmu. V první verzi aplikace pro bakalářskou práci jsem jednotlivé fáze při zpracování dat ukládal do databáze ve formě tabulek. Pro práci s daty a realizaci jednotlivých kroků mi stačilo používat dotazovací jazyk SQL. Práce s SQL je poměrně přehledná, ale má i své nevýhody. Jednou z nich je časová náročnost realizace jednoho příkazu SQL. Pokud se jedná pouze o desítky operací s databází, tak je časová náročnost zanedbatelná, ale když se dostaneme na stovky až tisíce takových operací, potom se už časová náročnost znatelně projeví.

V další verzi jsem tento algoritmus přepracoval a pro ukládání dat při provádění algoritmu používám interní struktury (dvourozměrné seznamy). Představivost při práci se strukturami je o něco složitější, ale dosahuji díky ním značného zrychlení.

Při spuštění algoritmu Apriori se zavolá třída StopWatch a tím se uloží čas, ve kterém se algoritmus Apriori spustil. Po dokončení celého algoritmu se zavolá metoda *stop* uložena v tentýž třídě a ta nám vrátí celkovou dobu běhu algoritmu.

4.2.4 Closure

Třída Closure realizuje činnost algoritmu Closure. V této třídě se vyskytuje metoda *teloClosure*, která provádí celý algoritmus, který je popsán v kapitole 3.3.1 Pseudokód algoritmu Closure. Tak jako třída Apriori, tak i tato třída obsahuje na začátku volání metody *teloClosure* spuštění časového intervalu pomocí třídy StopWatch. Na konci provádění metody *teloClosure* se měřený časový interval zastaví. Interní data se ukládají do instancí balíku struktury.

4.2.5 Seznam

Třída Seznam je součástí balíku Struktury a vytváří nám jednosměrně vázaný lineární seznam. Součástí třídy jsou i metody, umožňující práci s tímto seznamem. Seznam je tvořen prvky charakterizovanými třídou Prvek.

4.2.6 Prvek

Třída prvek patří taktéž do balíku Struktury a tvoří jednotlivé listy seznamu třídy Seznam. Obsahuje atributy text, pocet (což je frekventovanost daného prvku), dalsi (ukazatel na další prvek), dolu (ukazatel na podseznam prvku) a velikost (délka podseznamu). Nesmíme zapomenout, že součástí třídy jsou i metody pro práci s prvkem.

4.2.7 Podseznam

Tato třída podobně jako třída Seznam vytváří jednosměrně vázaný lineární seznam. Rozdílem těchto dvou tříd je, z jakých prvků jsou seznamy vytvořeny. Třída Podseznam je tvořena prvky vytvořené ze třídy Podprvek. Také je součástí balíku Struktury.

4.2.8 Podprvek

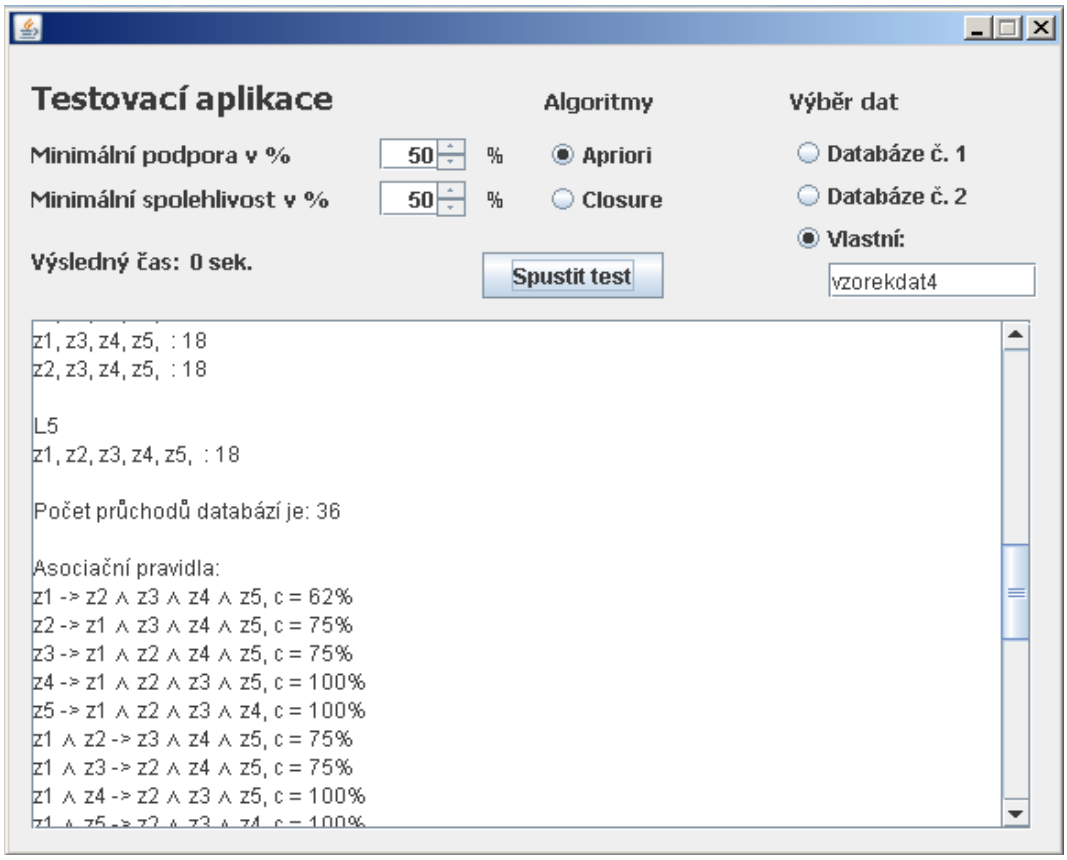
Třída Podprvek tvoří jednotlivé listy seznamu Podseznam. Obsahuje atributy text, pocet (určuje frekventovanost podprvku) a dolu (ukazatel na další podprvek v Podseznamu). Tato třída patří do balíku Struktury.

4.3 Práce s aplikací

4.3.1 Připravení prostředí pro práci s aplikací

Než začneme s aplikací pracovat musíme si připravit prostředí pro správnou funkčnost aplikace. Pro samotné spuštění aplikace musíme mít soubor JDK (java development kit), protože tato aplikace je vyvíjena v jazyku Java a bez JDK aplikaci nespustíme. Další potřebný soubor je balík mysql-conector-java.jar, sloužící pro propojení aplikace s databází. A třetí podmínkou, která je nezbytná pro práci s aplikací, je lokální databázový server a v něm vytvořené datové zdroje.

4.3.2 Snímek vzhledu aplikace



Obr. 4.2 Snímek vzhledu aplikace

4.3.3 Popis aplikace

Ovládání této aplikace je naprosto jednoduché a intuitivní. Při prvním pohledu na aplikaci je nám jasné, že musíme vybrat algoritmus, který budeme spouštět, nastavíme minimální podporu, spolehlivost a hlavně vybrat zdroj dat pro testovaný algoritmus. Zdroj dat si můžeme zvolit, buď ze dvou předem připravených tabulek uložených pod názvy databáze č.1 a 2, nebo si sami vytvoříme zdrojovou databázi ve formě tabulky a následně přes volbu vlastní databáze vložíme její název do aplikace. Jedinou podmínkou vytvořené tabulky je, aby obsahovala sloupce transakce a itemset0.

Po stisknutí tlačítka Spustit test, se nastaví všechny parametry potřebné pro práci vybraného algoritmu a patřičný algoritmus se spustí. Jakmile algoritmus doběhne do konce, zobrazí se celkový čas jeho běhu a v textové oblasti se vypíše zadaná minináln podpora v počtu prvků, postupné mezivýsledky generovaných množin (pouze u algoritmu Apriori), výsledné maximální množiny, počet průchodů databází a asociační pravidla vygenerovaná z výsledných maximálních množin.

Pokud algoritmus pracuje příliš dlouho, tak jej zastaví maximální časový interval nastavený na 5 minut.

5 Ověřování funkčnosti algoritmů

V kapitole ověřování funkčnosti algoritmů bude probíhat aplikování vytvořených algoritmů na různé datové zdroje. Nejprve v podkapitole 5.1 Datové zdroje představím datové zdroje, na kterých budu provádět ověřování. Následně provedu praktické ověření za pomocí ověřovací aplikace a v další podkapitole budu ze získaných výsledků dat vytvářet grafy. Díky těmto grafům budu moci lépe popsat získané vědomosti o algoritmech Apriori a Closure.

5.1 Datové zdroje

Pro ověřování funkčnosti algoritmů v aplikaci používám transakční databáze. Vytvořil jsem si tři různé tabulky v požadovaném formátu se sloupci transakce (číselné hodnoty) a itemset0 (obsahuje název položky). Všechny tři jsem uložil na lokální databázový server Mysql.

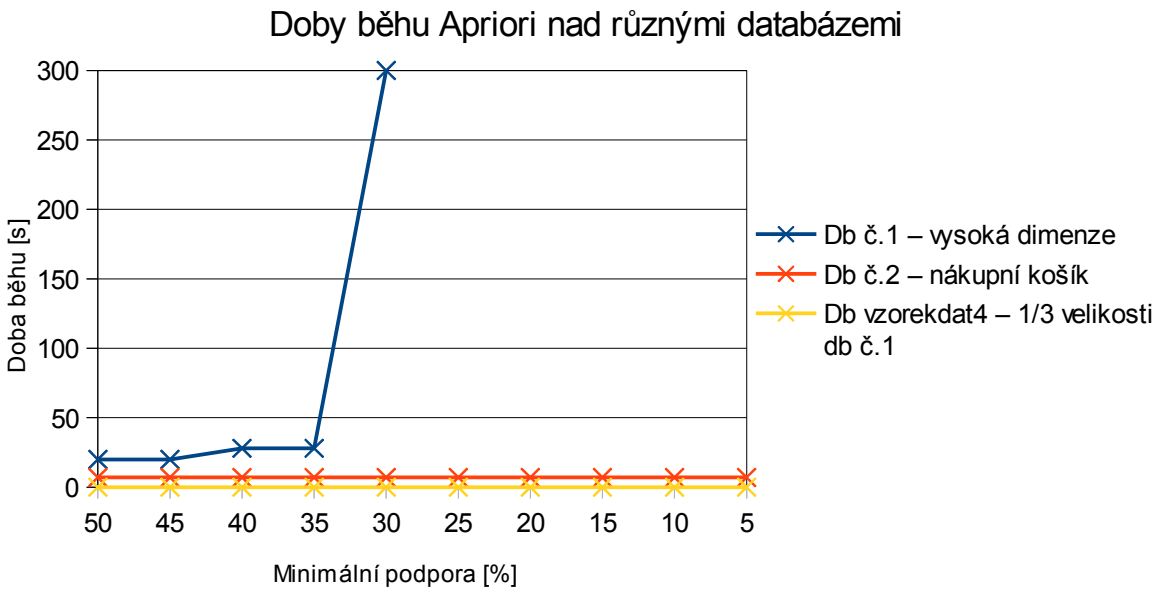
První tabulkou určenou pro ověřování je tabulka s názvem databaze4. Tato tabulka se v aplikaci skrývá pod názvem databáze č.1. Byla uměle vytvořena a obsahuje 465 záznamů se 49 transakcemi. Frekventovanost položek v této tabulce se pohybuje v množinách o vysokém počtu prvků.

Druhou zdrojovou databází je tabulka s názvem databaze3. Databaze3 se v aplikaci vyskytuje pod položkou databáze č.2. Je názorným příkladem transakčního zdroje dat pro analýzu nákupního košíku supermarketu. Obsahuje 3459 záznamů při 274 transakcích. Tato tabulka bohužel obsahuje frekventovanost pouze u jedno nebo dvou prvkových množin. Ale i díky tomuto typu zdroje dat si můžeme názorně ukázat výhody a nevýhody algoritmů.

Jako třetí zdrojovou databázi jsem si zvolil tabulku s názvem vzorekdat4, která je přibližně třetinové velikosti oproti databázi č.1. Obsahuje 149 záznamů o 29 transakcích. Pro ověřování je tato databáze střední cestou mezi databází č.1 a č.2.

5.2 Provádění ověřování

5.2.1 Zjišťování doby běhu algoritmů bez generování asociačních pravidel



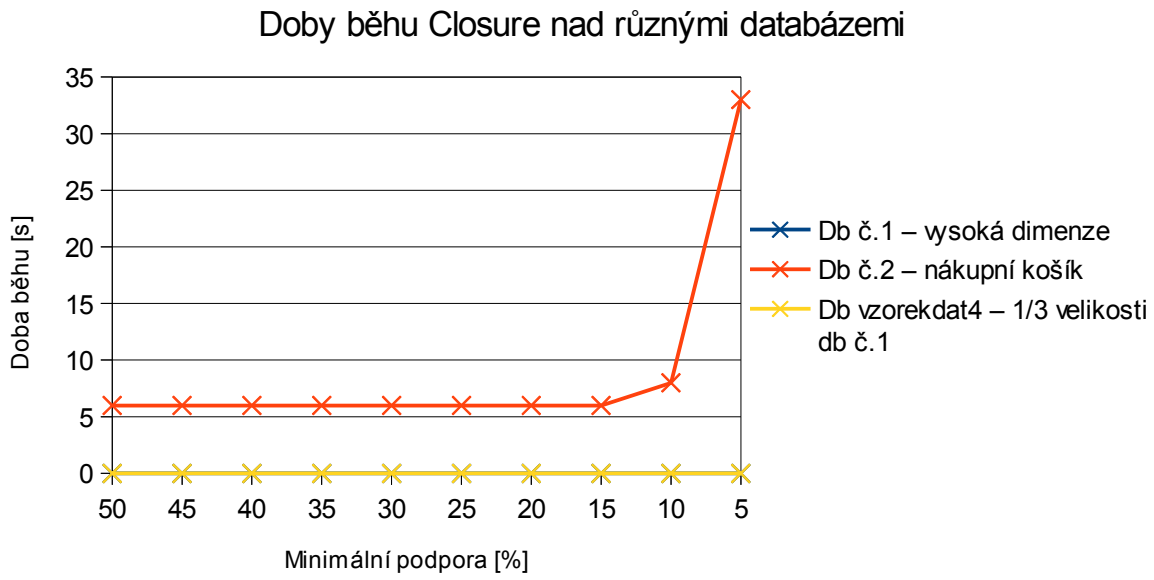
Obr. 5.1 Graf doby běhu algoritmu Apriori nad jednotlivými databázemi

Graf 5.1 nám znázorňuje výsledek testu, který byl proveden s algoritmem Apriori (dále už jen Apriori). Experiment obsahoval zpracování databáze pomocí Apriori bez následného generování asociačních pravidel (výsledkem byly pouze frekventované množiny) z maximálních frekventovaných množin nad třemi různými datovými zdroji popsány v kapitole 5.1. Jelikož Apriori generuje a zpracovává množiny od nejmenších (jednoprvkových) po největší, tak se od tohoto postupu odvíjí i různé doby zpracování, dle typu databáze.

Nejhorším případem je databáze č.1, která obsahuje víceprvkové množiny (přibližně s 10 prvky) s vysokou frekventovaností. Od hodnoty minimální podpory 30% se doba zpracování nevešla do předem stanoveného limitu 300 sekund, tak proto jsem ji už nezaznamenával v grafu.

U databáze č.2 (nákupní košík), u které se mnohokrát nevyskytují ani dvojice stejných položek v transakcích, Apriori s jejím zpracováním nemá větší problémy. Jediným náznakem zpomalení je při hodnotě minimální podpory menší než 5%. Důvodem je splnění minimální podpory dvouprvkovými množinami. Toto zpomalení graf nezachycuje.

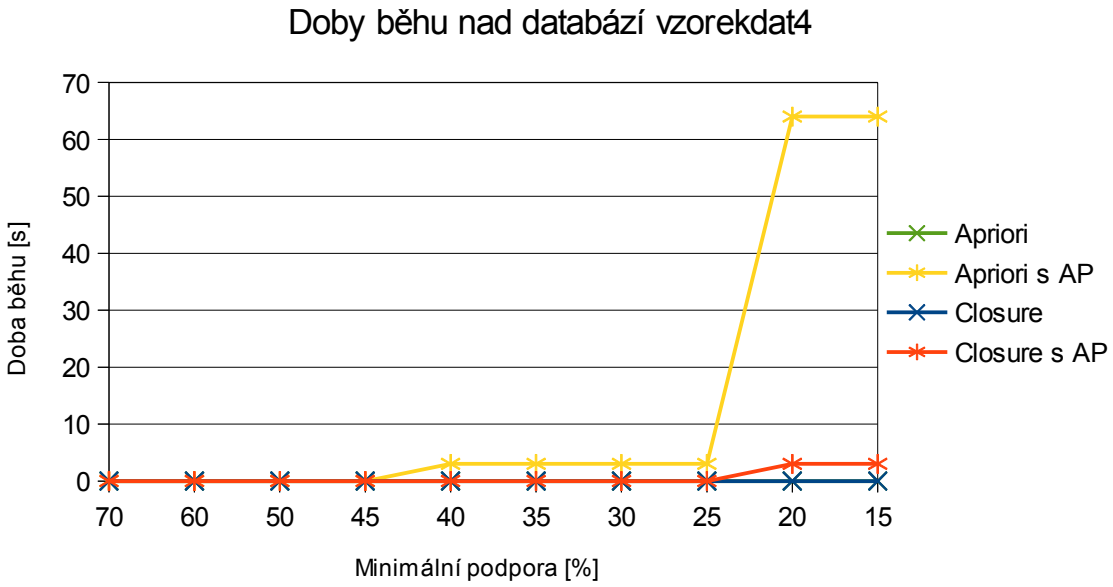
Třetí zpracovávaná databáze s názvem vzorekdat4 obsahuje malý počet záznamů, takže Apriori s tímto zdrojem neměl žádné časové zdržení.



Obr. 5.2 Graf doby běhu algoritmu Closure nad jednotlivými databázemi

Algoritmus Closure (dále jen Closure) prochází databáze od množin o maximálním počtu prvků, až po množiny dvouprvkové (jednoprvkové množiny si musí vygenerovat už na začátku). S databázemi č. 1 (je skrytý pod průběhem db vzorekdat4) a vzorekdat4 neměl Closure při zpracování žádný problém. Databázi č.2 algoritmus zpracovával průměrně za 7 sekund, než došel na minimální podporu 10 a méně procent, kde se začaly více objevovat frekventované jednoprvkové množiny a následně od 5 % dolů i dvouprvkové. Při této minimální podpoře si zpracování databáze pomocí Closure vyžádalo více času.

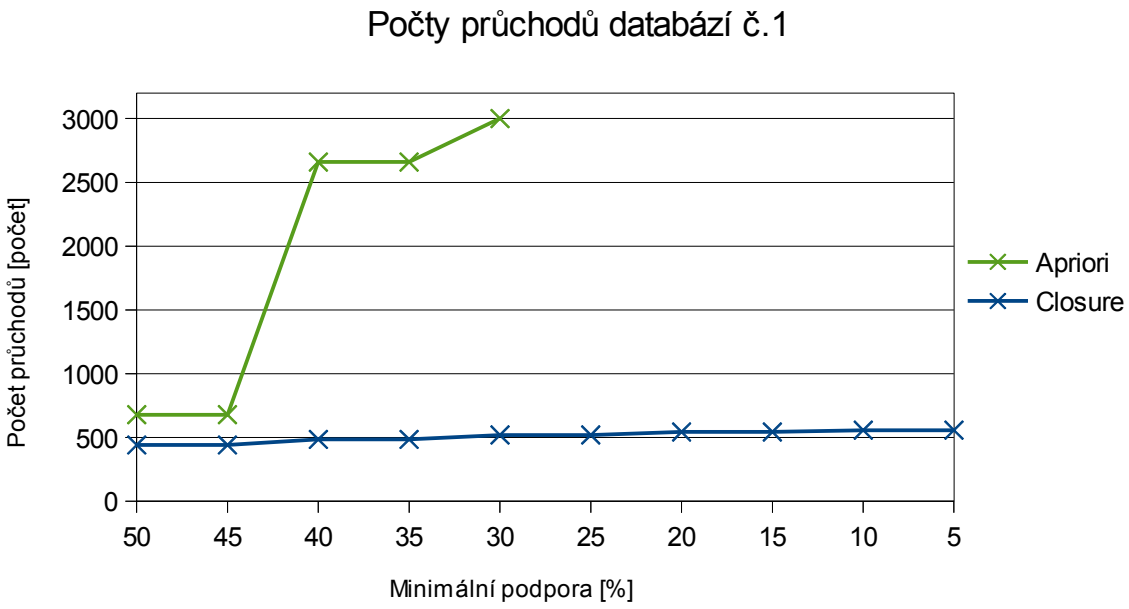
5.2.2 Zjišťování doby běhu algoritmů s generováním asociačních pravidel



Obr. 5.3 Graf doby zpracování databáze vzorekdat4

Graf 5.3 ukazuje, jak dokáže generování asociačních pravidel razantně snížit (hlavně u algoritmu Apriori) výkonnost algoritmu v pohledu na dobu zpracování. Zeleně označený průběh algoritmu Apriori bez generování asociačních pravidel je totožný s modrým průběhem algoritmu Closure.

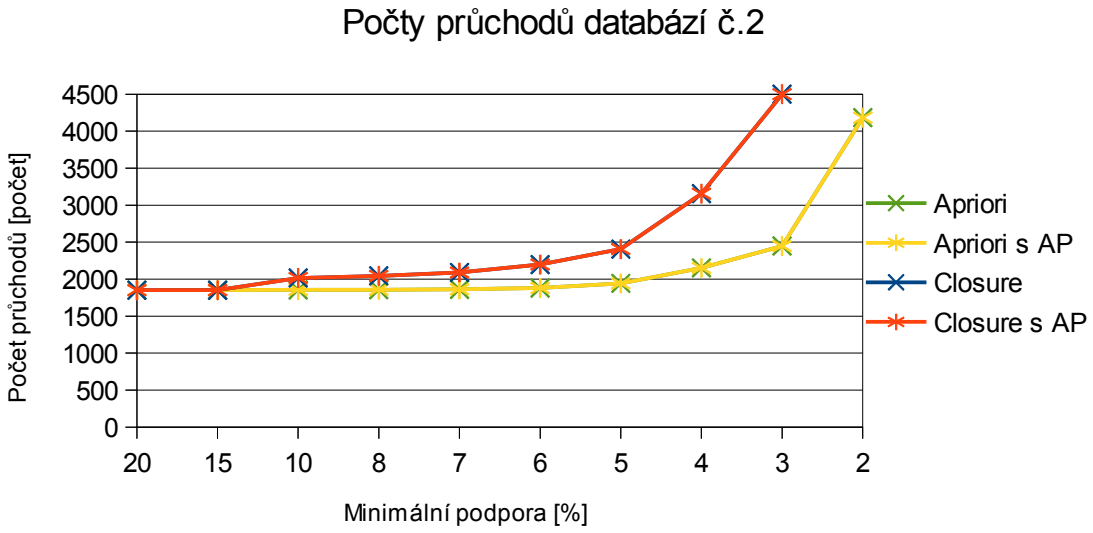
5.2.3 Zjišťování počtu průchodů zdrojovými databázemi



Obr. 5.4 Graf počtu průchodů databází č.1

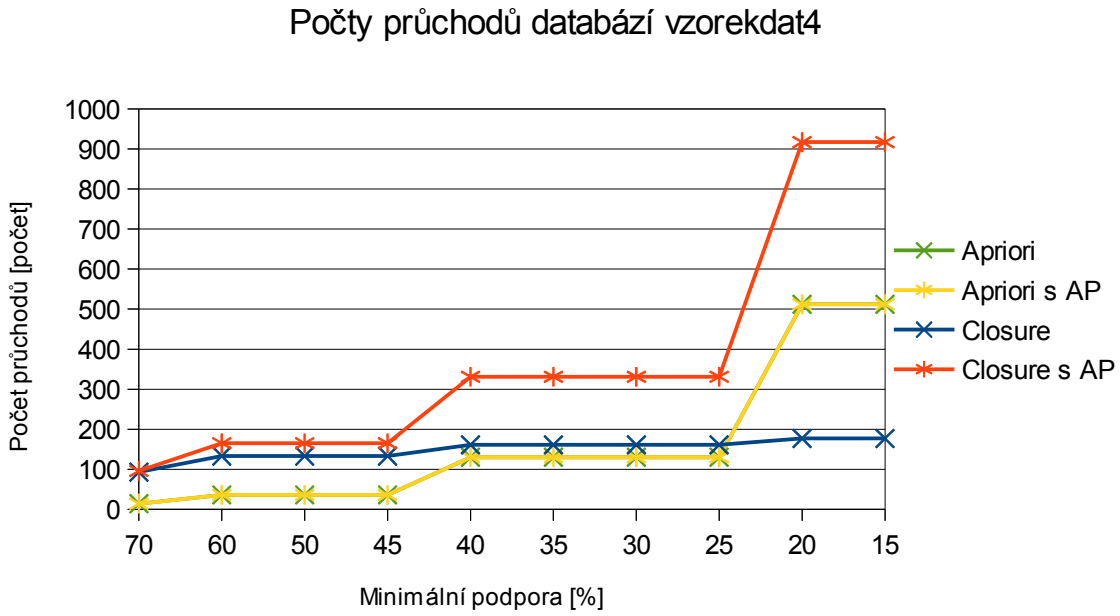
V tomto grafu nenajdeme průběhy z provedeného pokusu, kdy se při zpracování databáze generovala i asociační pravidla. Důvodem je, že algoritmus Apriori nezvládal tuto činnost provést při minimální podpoře 50% do 5 minut. Algoritmus Closure na tom byl podobně, ale pro něho se stalo hranicí 40%. Proto nebyl důvod tyto průběhy vkládat do grafu.

Z těchto dvou průběhů v grafu 5.4 můžeme říci, že vztah mezi dobou zpracování databáze a počtem průchodů databáze je přímo úměrný. Toto tvrzení je podloženo tím, že časově nejnáročnější operací je průchod zdrojové databáze. Pokud rozdíl v počtu průchodů dvou algoritmů databází je např. 2000 a více průchodů, tak tato doba se musí nutně projevit ve výsledném čase běhu algoritmu. U algoritmu Apriori jsme se od 30% dolů nedočkali výsledku počtu průchodů, protože doba běhu algoritmu se nevešla pod hranici 300 s.



Obr. 5.5 Graf počtu průchodů databází č.2

V grafu 5.5 vidíme, že ne vždy se musí počet průchodů databází při generování asociačních pravidel zvýšit. V případě algoritmu Apriori se počty průchodů databází po vygenerování asociačních pravidel nikdy nezvýší, protože veškeré podmožiny frekventovaných množin s nejvyšším počtem prvků budou již vygenerovány. U algoritmu Closure se při generování asociačních pravidel setkáváme ve většině případů s nárustem počtu průchodů zdrojovou databází. Důvod je jednoduchý, pokud najde algoritmus frekventovanou množinu, tak už negeneruje její podmnožiny. Z toho vychází, že při generování asociačních pravidel nám tyto podmnožiny budou chybět. Proč teda u tohoto příkladu nám podmnožiny nechybí? Protože frekventované množiny při čtyřech a méně procentech v tomto případě obsahují pouze každá 2 prvky. Veškeré podmnožiny, tj. jednoprvkové množiny už máme vygenerovány.



Obr. 5.6 Graf počtu průchodů databází vzorekdat4

V grafu 5.6 se již potvrdil klasický nárůst počtu průchodů databází při generování asociačních pravidel u algoritmu Closure. Při zpracování zdroje dat s generováním asociačních pravidel, oproti zpracování databáze bez generování asociačních pravidel. Další zajímavostí, které si v grafu můžeme všimnout je skřížení průběhů algoritmu Apriori s algoritmem Closure bez generování asociačních pravidel. Hlavním důvodem tohoto křížení je nárůst počtu průchodů u obou algoritmů. Zatímco u algoritmu Closure roste počet průchodů velmi pozvolna, tak u algoritmu Apriori je nárůst silně ovlivněn zvyšujícím se počtem vygenerovaných množin. Tento značný rozdíl v nárůstu počtu průchodů je vidět i v grafu 5.4. Zde se sice oba průběhy neprotínají, ale rozdíl v nárustu je viditelný.

6 Závěr

Na základě získaných výsledků z ověření funkčnosti mohu říci, že algoritmus Closure má ve většině případů lepší výsledky než algoritmus Apriori. Najdou se i specifické případy zdrojové databáze, pro kterou je algoritmus Apriori lepší než algoritmus Closure, ale těchto je minimum, oproti těm kde dominuje algoritmus Closure. Velkou výhodou algoritmu Closure je, že negeneruje tak velký počet množin jako Apriori, od toho se odvíjí i menší počet průchodů zdrojovou databází a v neposlední řadě i doba běhu algoritmu pro zpracování zdroje dat. Předpoklady pro rychlejší zpracování Closure algoritmu se potvrdily, a proto mohu říci na základě provedených testů, že je pro datové zdroje, kde se nepředpokládá frekventovanost na množinách o velmi malém počtu prvků, lepším algoritmem než Apriori.

Closure algoritmus obsahuje spoustu možností pro optimalizaci a tím zvýšení jeho výkonu při zpracování databází. Jak jsem zjistil při ladění obou algoritmů, tak největším podílem na výkonu samotného algoritmu je jeho optimalizace. Algoritmy, bez jakýchkoliv optimalizačních úprav nám nic neřeknou, budou se chovat podobně, každý pro svá vhodná data bude dobrý, ale pro méně vhodná data bude nedostačující. Výkonný a zároveň použitelný algoritmus musí být pro méně vhodná data dostačující (musí je zvládnout zpracovat, i když za mírně horší dobu běhu) a pro vhodná data musí pracovat bez jakéhokoliv zaváhání. V této práci jsem se optimalizací příliš nezabýval, provedl jsem jen několik drobných úprav, abych mohl vůbec ověřit funkčnost těchto dvou algoritmů na předem připravených databázích. Pokud bych se dále věnoval algoritmu Closure, zabýval bych se jeho optimalizací výkonu.

Literatura

[1] ZENDULKA, J., BARTÍK, V., LUKÁŠ, R., RUDOLFOVÁ, I. *Získávání znalostí z databází.* Brno, 2009. Studijní opora. Fakulta informačních technologií, Vysoké učení technické v Brně.

[2] GHORECHA, Vimal. Comparative Evaluation of Association Rule Mining Algorithms with Frequent Item Sets. *IOSR Journal of Computer Engineering (IOSR-JCE).* Mar.-Apr. 2013, Volume 9, Issue 5, s. 08-14. e-ISSN: 2278-0661, p- ISSN: 2278-8727. Dostupné z : http://www.iosrjournals.org

[3] CRISTOFOR, Dana, CRISTOFOR, Laurentium, SIMOVICI, A., Dan. Galois Connections and Data Mining. *Journal of Universal Compter Science* [online]. 2000, vol. 6, no. 1, s. 60 – 73, [cit. 2014-05-13]. Dostupné také z : http://www.jucs.org/jucs_6_1/galois_connections_and_data/Cristofor_D.pdf

[4] TAN, Pang-Ning, STEINBACH, Michael, KUMAR, Vipin. *Introduction to Data Mining*, Chapter 6. Instock: Addison-Wesley, 2005. ISBN-10: 0321321367, ISBN-13: 9780321321367. Dostupné také z : http://www-users.cs.umn.edu/~kumar/dmbook/ch6.pdf

Seznam příloh

Příloha 1. Ukázka ověření funkčnosti algoritm Closure
Příloha 2. Ukázka ověření funkčnosti algoritm Apriori
Příloha 3. CD.

- písemnou zprávu bp-xmatul18.pdf
- zdrojový tvar písemné zprávy bp-xmatul18.odt ve formátu OpenDocument
- soubor README s popisem instalace
- adresář aplikace, obsahující hierarchií zdrojových textů
- soubor mysql-connector-java-5.0.8-bin.jar pro propojení s databázovým serverem
- adresář jdk1.7.0_11, potřebný pro spuštění aplikace napsané v Javě
- adresář zdrojove databaze, obsahující zdrojové databáze ve formátu *.csv, na kterých bylo prováděno ověřování
- instalační soubor lokálního databázového serveru EasyPHP-DevServer-13.1VC11-setup.exe
- adresář overovaci aplikace obsahuje spustitelný soubor aplikace.exe

Příloha č.1

Provedení ověření funkčnosti algoritmu Closure

Vstupy:

Minimální podpora: 60%

Minimální spolehlivost: 60%

Algoritmus: Closure

Zdrojová databáze: vzorekdat4

Výstupy:

Doba zpracování: 0 sekund (nejmenší jednotkou jsou sekundy)

Minimální podpora je: 17 prvků.

Frekventované položky (za dvojtečkou je frekventovanost této množiny v počtu prvků):

z1, z2, z3, z4, z5 : 18

Počet průchodů databází je: 165

Asociační pravidla:

z1 -> z2 ∧z3 ∧z4 ∧z5, c = 62%

z2 -> z1 ∧z3 ∧z4 ∧z5, c = 75%

z3 -> z1 ∧z2 ∧z4 ∧z5, c = 75%

z4 -> z1 ∧z2 ∧z3 ∧z5, c = 100%

z5 -> z1 ∧z2 ∧z3 ∧z4, c = 100%

z1 ∧z2 -> z3 ∧z4 ∧z5, c = 75%

z1 ∧z3 -> z2 ∧z4 ∧z5, c = 75%

z1 ∧z4 -> z2 ∧z3 ∧z5, c = 100%

z1 ∧z5 -> z2 ∧z3 ∧z4, c = 100%

z2 ∧z3 -> z1 ∧z4 ∧z5, c = 75%

z2 ∧z4 -> z1 ∧z3 ∧z5, c = 100%

z2 ∧z5 -> z1 ∧z3 ∧z4, c = 100%

z3 ∧z4 -> z1 ∧z2 ∧z5, c = 100%

z3 ∧z5 -> z1 ∧z2 ∧z4, c = 100%

z4 ∧z5 -> z1 ∧z2 ∧z3, c = 100%

z1 ∧z2 ∧z3 -> z4 ∧z5, c = 75%

z1 ∧z2 ∧z4 -> z3 ∧z5, c = 100%

z1 ∧z2 ∧z5 -> z3 ∧z4, c = 100%

z1 ∧z3 ∧z4 -> z2 ∧z5, c = 100%

z1 ∧z3 ∧z5 -> z2 ∧z4, c = 100%

z1 ∧z4 ∧z5 -> z2 ∧z3, c = 100%

z2 ∧z3 ∧z4 -> z1 ∧z5, c = 100%

z2 ∧z3 ∧z5 -> z1 ∧z4, c = 100%

z2 ∧z4 ∧z5 -> z1 ∧z3, c = 100%

z3 ∧z4 ∧z5 -> z1 ∧z2, c = 100%

z1 ∧z2 ∧z3 ∧z4 -> z5, c = 100%

z1 ∧z2 ∧z3 ∧z5 -> z4, c = 100%

z1 ∧z2 ∧z4 ∧z5 -> z3, c = 100%

z1 ∧z3 ∧z4 ∧z5 -> z2, c = 100%

z2 ∧z3 ∧z4 ∧z5 -> z1, c = 100%

Příloha č.2

Provedení ověření funkčnosti algoritmu Apriori

Vstupy:

Minimální podpora: 4%

Minimální spolehlivost: 20%

Algoritmus: Apriori

Zdrojová databáze: Databáze č.2

Výstupy:

Doba zpracování: 9 sekund

Minimální podpora je: 10 prvků.

Počet průchodů databází je: 2152

L1

A Caj 0.90 2 : 27

A Caj_lemon 0.90 2 : 23

A Caj_maracuja 0.90 2 : 15

A Caj_ovocna40 0.90 2 : 13

A ChlebKminVek 6.90 2 : 12

A Cokolada 2.90 1 : 34

A Cokolada 5.80 1 : 15

A Deli_Super 7.90 1 : 11

A EH_JogY4x125 0.90 2 : 12

A Ins.polevka 8.90 2 : 13

A Ins._polevka 8.90 2 : 16

A KeCup 6.90 2 : 35

A Krmivo 9.90 2 : 10

A NT_MlekTNT1l 2.30 2 : 13

A OkurkaSalat. 2.50 2 : 13

A OL_Pudink200 3.80 2 : 11

A OL_Pudink200 6.90 2 : 19

A Persil 9.00 1 : 17

A Rohlik_hlad. 3.00 2 : 11

A Rohlik_hlad. 6.50 2 : 12

A Rohlik_hlad. 7.80 2 : 11

A Testoviny 3.90 2 : 11

A UN_Flora_250 5.90 2 : 11

A UN_ViennC600 5.90 2 : 11

A VD_RybiPo250 1.90 2 : 16

L2

A Caj 0.90 2, A Caj_lemon 0.90 2 : 10

A Ins.polevka 8.90 2, A Ins._polevka 8.90 2 : 11

Asociační pravidla:

A Caj 0.90 2 -> A Caj_lemon 0.90 2, c = 37%

A Caj_lemon 0.90 2 -> A Caj 0.90 2, c = 43%

A Ins.polevka 8.90 2 -> A Ins._polevka 8.90 2, c = 84%

A Ins._polevka 8.90 2 -> A Ins.polevka 8.90 2, c = 68%