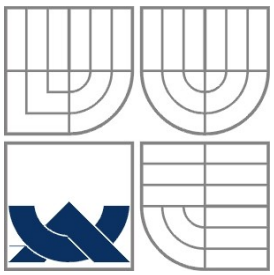
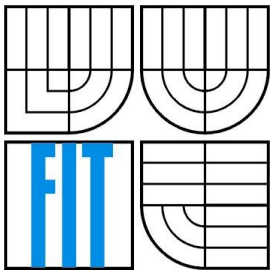


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOLOVÁNÍ ASOCIAČNÍCH PRAVIDEL

ASSOCIATION RULES MINING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL DVOŘÁK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. LUKÁŠ STRYKA

BRNO 2009

Abstrakt

Cílem této bakalářské práce je návrh a implementace aplikace umožňující porovnávat výkonnost a časovou náročnost zvolených algoritmů pro dolování frekventovaných množin a asociačních pravidel. Pro demonstraci byly vybrány dolovací algoritmy Apriori, AprioriTIDList, AprioriItemSet a metoda s využitím FP-stromu. Testování probíhalo nad různými objemy dat a s různými hodnotami minimální podpory a spolehlivosti.

Aplikace je implementována v objektově orientovaném jazyce C# a jako zdroj dat slouží relační databáze na MS SQL Server 2008.

Klíčová slova

Získávání znalostí z databází, frekventované množiny, asociační pravidla, Apriori, T-SQL, AprioriTIDList, AprioriItemSet, FP-strom, FP-growth, podpora, spolehlivost.

Abstract

The main goal of this bachelor's thesis is design and implementation of the application that provides a comparison of the performance and time consumption of given algorithms for mining of the frequent itemsets and the association rules. For demonstration, the mining algorithms Apriori, AprioriTIDList, AprioriItemSet and the method using FP-tree were chosen. The tests were executed over various amounts of data and with different minimum support and confidence values as well. The application was implemented in the object oriented language C# and the relational database provided by MS SQL Server 2008 is used as the data source.

Keywords

Knowledge discovery from databases, frequent patterns, associations rules, Apriori, T-SQL, AprioriTIDList, AprioriItemSet, FP-tree, FP-growth, support, confidence.

Citace

Dvořák Michal: Dolování asociačních pravidel. Brno, 2009, bakalářská práce, FIT VUT v Brně.

Dolování asociačních pravidel

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Lukáše Stryky.
Další informace mi poskytl pan Bartík Vladimír, Ing., Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Michal Dvořák
30. 4. 2009

Poděkování

Chtěl bych poděkovat všem lidem, kteří mě podporovali při tvorbě této práce, zvláště pak vedoucímu práce Ing. Lukáši Strykovi za poskytnutí kvalitních studijních materiálů a konzultací.

Obsah

Úvod

V dnešní době jsou data kumulována v databázích neuvěřitelnou rychlostí. Jelikož se z dat stávají informace až po přidání sémantiky člověkem, je nezbytné poskytnout uživatelům alespoň částečně automatizované prostředky k vytřízení dat potencionálně zajímavých. To vedlo ke vzniku nové vědní disciplíny – Získávání znalostí z dat, která zahrnuje získávání různých typů znalostí z rozmanitých zdrojů dat.

Lepší představu může nastínit příklad ze života. V supermarketu byla sbírána data o nákupech zákazníků. Po čase se provedla analýza sesbíraných dat a zjistilo se, že v nákupním vozíku se objevují zároveň dětské pleny a přepravka piva. Tato část analýzy je prováděna pomocí zvoleného algoritmu. Nyní ale nastoupí manažer a přemýšlí, proč to tak je a jak z této informace profitovat. Každý si může tuto asociaci vyložit jinak. Důvodem může být například fakt, že dětské pleny jsou objemné zboží a přepravka piv těžká. Tudíž ani jednu věc nebude člověk nosit v ruce a koupí je spíše v případě, když se vydá na nákup autem. Manažer se díky této nové znalosti může rozhodnout, jestli umístí tyto druhy zboží vedle sebe nebo naopak co nejdále od sebe, aby musel zákazník projít velkou část obchodu a zvýšila se tak pravděpodobnost koupě dalšího zboží po cestě.

Dolováním asociací mezi položkami nákupního košíku se věnuje i tato práce.

V první kapitole je popsán proces získávání znalostí z databází spolu s představením různých typů dolovacích úloh a motivačními problémy, které jsou řešeny.

Druhá kapitola se věnuje problematice asociačních pravidel spolu s popisem jejich získávání z frekventovaných množin. Jsou zde definovány základní termíny (podpora a spolehlivost) a představeny algoritmy pro dolování frekventovaných množin. Jedná se zejména o algoritmus Apriori a jeho vylepšené modifikace – Apriori ItemSet a Apriori TID list. Poslední část tvoří dolování bez generování kandidátů – metoda s FP-stromem.

Třetí kapitola popisuje návrh konceptu aplikace pro dolování asociačních pravidel různými algoritmy.

Čtvrtá kapitola se věnuje implementaci jednotlivých algoritmů pro dolování frekventovaných množin a asociačních pravidel.

V páté kapitole je popsáno testování na reálných datech, vynesení výsledků do grafu a komentář k získaným výsledkům.

Poslední kapitolou je závěr, ve kterém jsou shrnuty výsledky práce.

1 Získávání znalostí z databází

Tato kapitola se zabývá problematikou získávání znalostí z databází.

1.1 Motivace k získávání znalostí z databází

S rozvojem informačních technologií docházelo ke shromažďování velkého množství dat, která neměla mnohdy valnou hodnotu, a tak docházelo po určité době k jejich mazání. Tato data avšak mohou ukrývat nějaké skryté, dříve neznámé a i potencionálně užitečné znalosti. A tak vznikla nová disciplína: Získávání znalostí z databází.

V podstatě jde o získání „zajímavých" informací, které jsou:

- **Netriviální** – nelze je získat pouhým SQL dotazem.
- **Skryté** – různé modely a vzory, které nejsou na první pohled zřejmé.
- **Dříve neznámé** – objevení známé pravdy nepřináší nic zajímavého.
- **Potenciálně užitečné** – mají např. význam pro další rozhodování.

Dle této definice [1] tedy do získávání znalostí nepatří informace o počtu prodaných ovocných nanuků v jednotlivých prodejnách, ani že nářadí si kupují spíše muži než ženy.

1.2 Proces získávání znalostí

V této kapitole bude představen proces získávání znalostí, který lze dle [1] rozdělit na 4 hlavní části:

- **Pochopení aplikační domény** – nejprve je nutné vědět, co data znamenají a co reprezentují v reálném světě.
- **Předzpracování dat** – nejdůležitější fáze, která má velký vliv na výsledky analýzy.
    - **Čištění dat (Cleaning)**

      V této fázi je nutné se vypořádat s chybějícími nebo odlehlými hodnotami, které by mohly ovlivnit výsledky.

      Příkladem uvedu záznamy tělesné teploty pacientů, které se zadávají s přesností desetiny stupně. Sestra se při zadávání splete a místo 36,5°C zadá 365.
    - **Integrace dat (Integration)**

      Ne vždy jsou informace v jediné databázi, proto je nutná jejich integrace do jediného zdroje, ze kterého se bude dolovat. V této fázi se také sjednocují názvy shodných sloupců tabulek.

Můžu mít například zákazníka se zákaznickou kartou, který platí občas platební kartou. A jednou si zapomene kartu zákaznickou, jindy platí hotově a někdy použije obojí. Spojením těchto informací můžu zákazníka jednoznačně identifikovat.

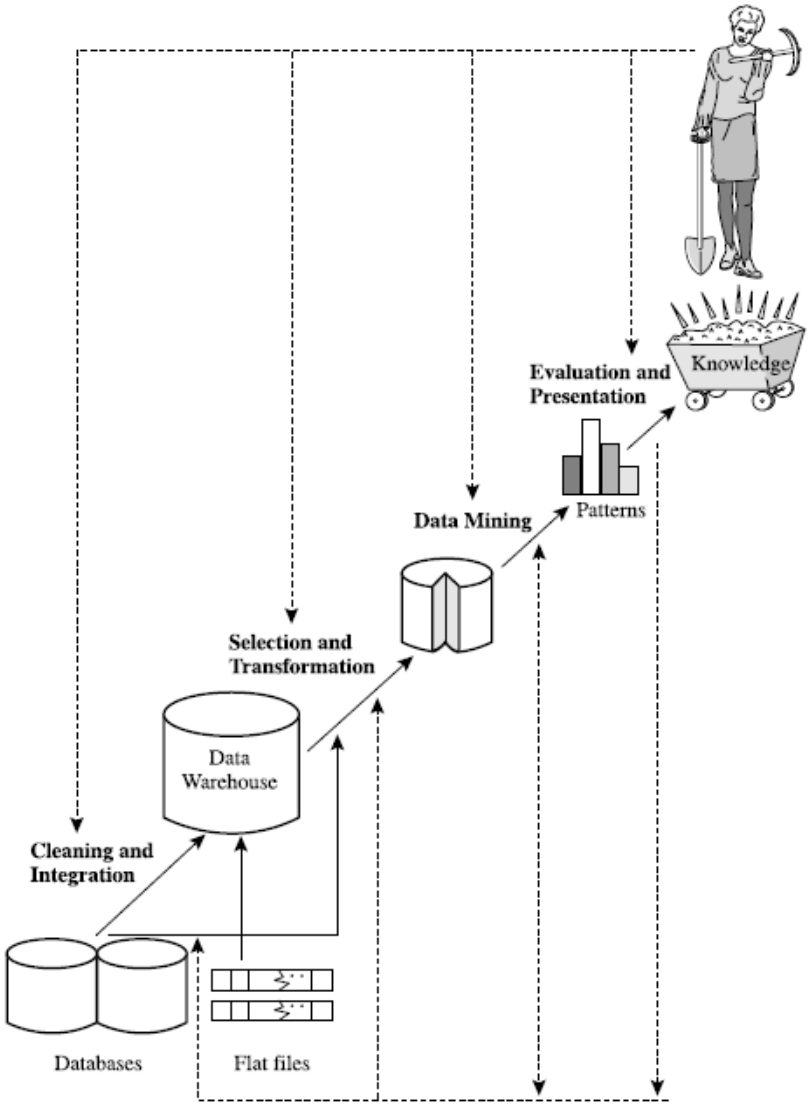
- **Výběr dat (Selection)**

  Jde o výběr jen těch sloupců, které jsou relevantní v právě řešené analýze. Například u zákazníka kupujícího potraviny mě nebude zajímat konfekční velikost, ale spíš kolik členů má jeho rodina a jestli bere své děti na nákup.

- **Transformace dat (Transformation)**

  Vybraná data se v této fázi zpracovávají do podoby vhodné pro danou dolovací úlohu.

- **Dolování z dat (Data Mining)** – jádro celého procesu získávání znalostí z databází.
- **Hodnocení modelů (Evaluation) a prezentace znalostí (Presentation)** – identifikují se zajímavé vzory a prezentují se výsledky analýzy.



Obrázek 1.1 – Proces získávání znalostí (kniha [1]*, str. 6, fig. 1.4)*

1.3 Typy dolovacích úloh

Typy dolovacích úloh se liší dle toho, jaký druh modelu dat se snažím z dat získat. Jde o úlohu řešenou v kroku dolování z dat.

Dolovací úlohy je možné rozdělit do dvou základních skupin [1]:

- **Deskriptivní** – charakterizují obecné vlastnosti analyzovaných dat. Jako příklad je možné uvést časté nákupy.
- **Prediktivní** – na základě analýzy současných dat se provádí dedukce pro předpověď budoucího chování nebo k rozhodování.

Typy dolovacích úloh se však dají také dělit podle funkcionality [1]:

- **Popis konceptu/třídy**
    - **Charakterizace dat**
    - **Diskriminace dat**
- **Úlohy zaměřené na odhalování vztahů mezi atributy**
- **Klasifikace a predikce**
- **Shluková analýza**
- **Analýza odlehlých hodnot**
- **Analýza evoluce**
- **A další.**

Tématem této práce je odhalování vztahů mezi atributy, což bude podrobně rozebráno v následujících kapitolách.

2 Asociační pravidla

Tato kapitola, stejně jako celá tato práce, se zabývá jednou z oblastí získávání znalostí, a to dolováním asociačních pravidel [1], [2] nad rozsáhlými množinami datových položek. Získaná asociační pravidla reprezentují zajímavé vztahy mezi zkoumanými položkami. Typické využití těchto znalostí je při analýze nákupního košíku, kdy je zkoumáno nákupní chování zákazníků – jsou hledány souvislosti mezi nakupovanými položkami, viz [1], [3], [4], [5].

2.1 Definice asociačního pravidla

Pravidlem se dle Agrawala (1993) rozumí „implikace“ $A \Rightarrow B$, kde A znamená předpoklad a B závěr. Nejedná se ale přímo o implikaci, jelikož jev nenastává vždy s pravděpodobností rovnou jedné. Formálně lze tato pravidla zapsat následovně:

> Nechť $\mathfrak{I} = \{I_1, I_2, \dots, I_m\}$ je množina položek. Nechť D je množina transakcí, kde každá transakce T je množina položek taková, že $T \subseteq \mathfrak{I}$. Každá transakce je asociována s unikátním identifikátorem zvaným TID. Nechť A je množina položek. Říkáme, že transakce T obsahuje A tehdy a jen tehdy, když $A \subseteq T$. Asociační pravidlo je „implikace“ tvaru $A \Rightarrow B$, kde $A \subset T, B \subset T$ a $A \cap B = \phi$ [1].

Tato definice generuje úplně všechna možná asociační pravidla. Bude jich tedy vygenerováno velké množství a navíc spousta z nich nebude zajímavá. Objektivní zajímavosti se dá ale dosáhnout omezením pomocí minimální podpory a spolehlivosti, viz následující podkapitola.

2.2 Podpora a spolehlivost

Základními objektivními charakteristikami pravidel jsou podpora (z anglického support) a spolehlivost (z anglického confidence).

Podpora udává četnost výskytu daného pravidla v databázi transakcí. Uvádí se buď jako absolutní (počet transakcí) nebo jako relativní (v % počtu všech transakcí). Vyjadřuje pravděpodobnost $P(A \cup B)$, kde $A \cup B$ označuje, že transakce obsahuje jak A, tak i B, viz [1], [2].

Výpočet hodnoty podpory se provádí pomocí následující formule:

$$podpora(A \Rightarrow B) = P(A \cup B). \qquad \text{(Vzorec 2.1 [1])}$$

Jelikož podpora neříká nic o tom, kdy se vyskytne předpoklad, ale závěr již ne, vznikla druhá objektivní charakteristika, a sice spolehlivost, která udává zajímavost asociačního pravidla.

Je definována jako podmíněná pravděpodobnost $P(B \mid A)$, tzn. pravděpodobnost B za předpokladu, že platí A, viz [1], [2].

Hodnota spolehlivosti je odvozena z hodnot podpory množiny položek A a podpory množiny $(A \cup B)$ dle vztahu:

$$spolehlivost(A \Rightarrow B) = P(B \mid A) = \frac{podpora(A \cup B)}{podpora(A)} \qquad \text{(Vzorec 2.2 [1])}$$

Pro ucelení tématu uvádím rovnici a nerovnici, které demonstrují rozdíl mezi podporou a spolehlivostí asociačních pravidel.
Pro množiny položek A, B, kde $A \subset T, B \subset T$ a $A \cap B = \phi$ platí:

$$podpora(A \Rightarrow B) = podpora(B \Rightarrow A), \qquad \text{(Vzorec 2.3 [1])}$$

což není nic překvapivého. Podpora závisí jen na tom, jestli se v transakci objevují obě množiny položek současně.
Na druhou stranu u spolehlivosti již rovnost obecně neplatí:

$$spolehlivost(A \Rightarrow B) \neq spolehlivost(B \Rightarrow A). \qquad \text{(Vzorec 2.4 [1])}$$

2.3 Proces dolování asociačních pravidel

Získávání asociačních pravidel probíhá ve třech krocích [1]:

1. Nalezení frekventovaných množin – ty splňují podmínku minimální podpory dle vzorce 2.1.
2. Pro každou frekventovanou množinu l se vygenerují všechny její neprázdné podmnožiny.
3. Pro každou podmnožinu s se vygeneruje pravidlo $s \Rightarrow (l - s)$ a dle vzorce 2.2 se vypočítá jeho spolehlivost. Pokud je spolehlivost pravidla větší než minimální, je pravidlo silné. Minimální podpora je již zajištěna tím, že se pravidla generují z frekventovaných množin.

2.4 Získávání frekventovaných množin - Apriori

Algoritmus Apriori [1] byl vyvinut pány Agrawalem a Srikantem v roce 1994 na dolování frekventovaných množin[1]. Jméno algoritmu je založeno na faktu, že algoritmus využívá předchozí

[1] Frekventovaná množina je taková množina položek, kde každá podmnožin položek má podporu vyšší než minimální.

znalosti (z anglického prior knowledge) o dříve získaných frekventovaných množinách, to znamená, že *k*-množiny[2] se používají pro generování *(k+1)*-množin.

Pro vyšší efektivitu se využívá Apriori vlastnosti, která říká, že každá neprázdná podmnožina frekventované množiny je také frekventovaná. Kdyby to tak nebylo, přidání položky k množině by mohlo způsobit zvýšení podpory, což není možné.

Algoritmus Apriori se skládá ze dvou částí, a to:

- **Spojovací krok:** K nalezení L_k je nutné nejprve získat množinu všech kandidátních *k*-množin C_k, kterou lze získat spojením L_{k-1} s L_{k-1}.

  Nechť l_1 a l_2 jsou množiny L_{k-1}, které jsou uspořádané podle libovolného, ale vhodného pravidla[3]. Notace $l_i[j]$ značí *j*-tý prvek množiny l_i. Spojení $L_{k-1} \bowtie L_{k-1}$ je proveditelné tehdy, pokud je prvních (k-2) prvků stejných, tedy pokud platí:

  $$(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1]).$$

  Podmínka $(l_1[k-1] < l_2[k-1])$ jen zajistí nevytváření duplicit. Výsledná množina, která vznikla z l_1 a l_2, je: $\{l_1[1],\ l_1[2],\ \ldots,\ l_1[k-2],\ l_1[k-1],\ l_2[k-1]\}$.
- **Vylučovací krok:** C_k je nadmnožinou L_k, to tedy znamená, že prvky C_k mohou, ale nemusí být frekventované. Zároveň všechny frekventované *k*-množiny do C_k náleží. Zjišťování podpory každé kandidátní množiny by bylo velice náročné, využívá se tedy Apriori vlastnosti, tedy: Žádná *(k-1)*-množina, která není frekventovaná, nemůže být součástí frekventované *k*-množiny. Takové *k*-množiny můžeme okamžitě odstranit z C_k.

[2] *k*-množina je množina, která má *k* prvků.
[3] Uspořádání musí být rostoucí nebo klesající: hodnoty pro uspořádání musí být po dvou různé.

Algoritmus Apriori napsaný v pseudokódu [1]:

Vstup:
D – databáze transakcí,
min sup – hodnota minimální podpory.

Výstup:
L – frekventovaná množina z D.

Pseudokód:

```
L1 = najdi_frekventovane_1_mnoziny(D)
for (k = 2; Lk-1 ≠ ϕ; k++)
{
    Ck = apriori_gen(Lk-1)
    for each transakci t ∈ D
    {
        Ct = subset(Ck, t)
        for each kandidata c ∈ Ct
         c.count++
    }
    Lk = {c ∈ Ck|c:count ≥ min_sup}
}
return L = ⋃k Lk

procedure apriori_gen(Lk-1: frekventovane (k-1)-množiny)
    for each množinu l1 ∈ Lk-1
         for each množinu l2 ∈ Lk-1
              if (l1[1]=l2[1]) ∧ (l1[2]=l2[2]) ∧ … ∧ (l1[k-2]=l2[k-2]) ∧
               (l1[k-1]<l2[k-1]) then
              {
                   c = l1 ⋈ l2 // spojovaci krok - generovani
                   kandidatu
                   if ma_nefrekventovanou_podmnozinu(c, Lk-1) then
                        odstran c // vylucovaci krok
                   else
                        pridej c do Ck
              }
return Ck;

procedure ma_nefrekventovanou_podmnozinu (c: kandidat k-mnoziny,
Lk-1: frekventovana (k-1)-mnozina) // vyuziva Apriori vlastnosti
for each (k-1)-mnozinu s z c
    if s ∉ Lk-1 then
         return TRUE
return FALSE
```

2.5 Získávání frekventovaných množin - AprioriItemset a AprioriTIDList

Jakmile se objeví nová věc, objeví se také její vylepšení. A jinak tomu nebylo ani u algoritmu Apriori. Ke zvýšení efektivity algoritmu byly vymyšleny různé techniky, které řeší množství čtení z transakční databáze, množství kandidátů na frekventované množiny a v neposlední řadě zátěž při počítání podpory kandidátů. Vznikly tak metody jako Partition, DHP, DIC, MaxMiner, Closet a další.

V této práci budou rozebrány dvě takové modifikace algoritmu Apriori, jejichž vylepšení je založeno na nalezení vhodnějších datových struktur pro frekventované množiny a informací o tom, které transakce je obsahují. Místo standardního horizontálního formátu dat se zavedl formát vertikální, kde ke každé položce z databáze je uvedeno, ve kterých transakcích se nalézá. Jedná se o metody AprioriItemset a AprioriTIDList [1].

Metoda AprioriItemset ukládá ke každé frekventované množině bitový vektor, kde jednotlivé bity udávají, jestli se tato frekventovaná množina v transakci nachází nebo ne[4]. Z toho plyne, že délka bitového vektoru je rovna počtu transakcí.

C_2	**Bitový vektor**
$\{A, B\}$	1011
$\{A, C\}$	1000
$\{B, D\}$	0110
$\{B, C\}$	0110

Tabulka 2.1 – Množina a její bitový vektor u Apriori Itemset

Pro zjištění, ve kterých transakcích se nachází množina $\{A, B, C\}$, stačí provést logický součin nad množinami $\{A, B\}$ a $\{A, C\}$: $\{1011\}\,\text{AND}\,\{1000\} = \{1000\}$. Hledaná 3-množina je v první transakci.

Pro spočítání podpory takto nalezené množiny stačí zjistit počet log. 1 ve vektoru.

Přestože logické operace nad bitovými vektory jsou velice rychlé, jejich uložení je při velkém počtu transakcí náročné na paměť. Tento nedostatek se snaží odstranit druhá metoda a to AprioriTIDList. Ta ke každé frekventované množině ukládá seznam transakcí, v nichž se nachází.

[4] Log. 1 znamená ANO, log. 0 NE.

C_2	RV
$\{A, B\}$	$\{1, 3, 4\}$
$\{A, C\}$	$\{1\}$
$\{A, D\}$	$\{2, 3\}$
$\{B, C\}$	$\{2, 3\}$

Tabulka 2.2 – Množina a její bitový vektor u Apriori Itemset

Přítomnost kandidátů v transakcích se zjišťuje průnikem množin RV u kandidátů – podmnožin daného kandidáta.

Podpora je rovna počtu prvků v množině RV.

2.6 Získávání frekventovaných množin bez generování kandidátů

Algoritmus Apriori má dva nedostatky, kvůli nimž je takřka nereálné jeho nasazení nad velkými databázemi:

- Generuje obrovské množství kandidátních množin. Je-li na počátku např. 10^4 1-množin, vygeneruje algoritmus Apriori více než 10^7 2-množin.
- Je potřeba opakovaně procházet databázi.

Snahou o vyřešení těchto problémů je metoda vzrůstu frekventovaných množin (z angl. FP-growth, tedy frequent-pattern growth).

Metoda FP-growth [1], [6], [7], [8] transformuje problém hledání dlouhých frekvenčních vzorů na rekurzivní hledání vzorů kratších a připojování sufixu. Jako sufixy používá nejméně frekventované položky.

Tato metoda značně snižuje časovou náročnost procesu dolování frekventovaných množin.

Základem dolování je transformace databáze do struktury zvané FP-strom.

FP-strom má několik vlastností, které nelze opomenout uvést:

- Nikdy neporuší dlouhou množinu z jakékoliv transakce.
- Udržuje kompletní informaci pro potřeby získávání frekventovaných množin - není tedy nutné opakované čtení databáze např. pro získání podpory dané množiny.
- Redukuje nerelevantní informace – nefrekventované položky jsou vymazány.
- Je „seřazen“ podle frekvence výskytu – vychází z myšlenky, že více frekventované položky budou spíše sdíleny.
- Nikdy nebude větší než původní databáze – dochází v ní ke kompresi informace.

Algoritmus na výstavbu FP-stromu a jeho následné dolování napsaný v pseudokódu [1]:

Vstup:
D – databáze transakcí,
min sup – hodnota minimální podpory.

Výstup:
Kompletní množina frekventovaných vzorů.

Pseudokód:

FP-strom je konstruován následovně:

a) Projdi jednou transakční databázi. Vytvoř množinu frekventovaných položek *F* a spočítej podporu těchto položek. Setřiď *F* sestupně podle podpory a vyber jen ty položky, které vyhovují minimální podpoře. Tento seznam pojmenuj *L*.
b) Vytvoř kořen FP-stromu a označ jej „NULL". S každou transakcí *Trans* z *D* udělej následující:
   1. Vyber a setřiď frekventované položky v *Trans* dle *L*. Nechť seznam seřazených frekventovaných položek v *Trans* je [*p*|*P*], kde *p* je první element a *P* je množina zbylých elementů.
   2. Zavolej *insert_tree([p|P], T)*, která dělá následující: Pokud *T* má potomka *N* takového, že *N.item-name = p.item-name*, pak inkrementuj počítadlo prvku *N* o jedničku. Jinak vytvoř nový uzel *N*, nastav jeho počítadlo na 1, jeho rodičovský uzel nasměruj na *T* a jeho „node-link" nasměruj do dalších uzlů se stejným *item-name*. Pokud není *P* prázdné, volej *insert_tree(P, N)* rekurzivně.

FP-strom je dolován voláním *FP_growth(FP-tree, null)*, který je implementován následovně:

```
Procedure FP_growth(Tree, α)
    Pokud Tree obsahuje pouze jednoduchou cestu P potom
        Pro každou kombinaci β uzlů v cestě P
            Generuj vzory β ∪ α s podporou = minimální podpora
            uzlů v β.
    Jinak
        Pro každé a_i v tabulce odkazů stromu Tree
            Generuj vzor β = a_i ∪ α s podporou = a_i.podpora
            Zkonstruuj podmíněný základ a podmíněný FP-strom
            Tree_β
            Pokud Tree_β ≠ ϕ pak
                Zavolej FP_growth(Tree_β, β)
```

2.6.1 Příklad výstavby FP-stromu a jeho dolování

Výstavbu a dolování FP-stromu budu demonstrovat na příkladu inspirovaném knihou [1].

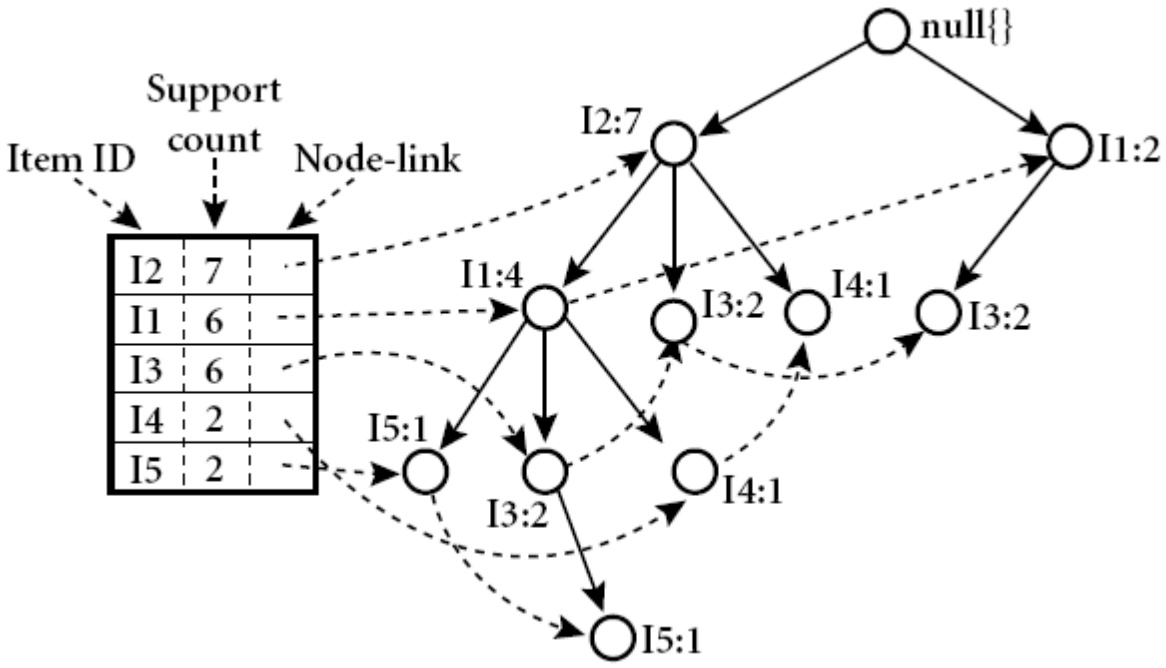
TID	**Seznam identifikátorů položek**
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Tabulka 2.3 – Transakce a její položky: data pro výstavbu FP-stromu (kniha [1]*, str. 236, tab. 5.1)*

První krok je stejný jako u algoritmu Apriori – získání frekventovaných 1-množin a podpory pro každou takto získanou 1-množinu. Nechť je minimální podpora rovna 2 a množina frekventovaných položek je seřazena sestupně podle podpory. Takto vzniklá množina či seznam je nazvána L, kde: $L = \{\{I2:7\}, \{I1:6\}, \{I3:6\}, \{I4:2\}, \{I5:2\}\}$.

FP-strom je konstruován následovně. Nejprve je vytvořen kořen stromu označený jako „null“. Projde se databáze podruhé. Položky v každé transakci jsou zpracovávány v pořadí dle seznamu L (počtu výskytů) a pro každou transakci je vložena větev, kde uzly jsou položky transakce. Například zpracování první transakce „T100: I1, I2, I5“ vede ke konstrukci první větve stromu <I2:1>, <I1:1> a <I5:1>, kde I2 je potomek kořene, I1 potomek I2 atd. Druhá transakce „T200: I2, I4“ by měla vytvořit další větev z kořene stromu obsahující I2 a I4. Tato větev ale sdílí s předchozí větví prefix I2, a proto bude pouze inkrementován počet výskytů u položky I2 a z této položky „vyroste“ nová větev s prvkem I4:1 – mateřským uzlem nyní bude I2:2. U slučování větví tedy platí, že počet výskytů je inkrementován u všech položek, které tvoří společnou cestu grafem.

Pro efektivnější průchod stromem je vytvořena tabulka odkazů na uzly a každý takový uzel obsahuje ukazatel na další uzel shodného jména, tzv. chain of node-links. Tím je problém dolování frekventovaných vzorů v databázích transformován na dolování FP-stromu.



Obrázek 2.4 – Schéma FP-stromu a tabulky odkazů na uzly (kniha [1]*, str. 244, fig. 5.7)*

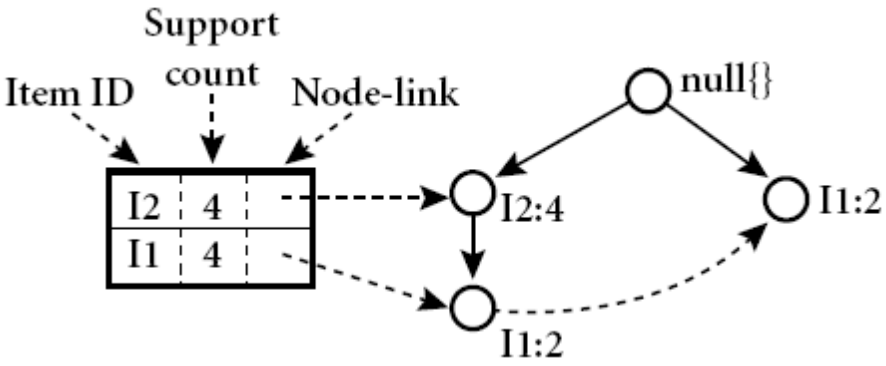
FP-strom je dolován následovně. Dolování je započato pro každý frekventovaný vzor délky 1 (iniciační sufix vzoru z tabulky odkazů na uzly) zkonstruováním jeho podmíněného základu a následně podmíněného FP-stromu. Dolování je následně rekurzivně opakováno, dokud podmíněný strom neobsahuje pouze jednoduchou cestu. Růst vzoru je dosažen konkatenací sufixu vzoru s frekventovanými vzory generovanými z podmíněného FP-stromu.

Položka	Podmíněný základ	Podmíněný FP-strom	Frekventované množiny
I5	{{I2, I1:1}, {I2, I1, I3:1}}	<I2:2, I1:2>	{I2, I5:2}, {I1, I5:2}, {I2, I1, I5:2}
I4	{{I2, I1:1}, {I2:1}}	<I2:2>	{I2, I4:2}
I3	{{I2, I1:2}, {I2:2}, {I1:2}}	<I2:4, I1:2>, <I1:2>	{I2, I3:4}, {I1, I3:4}, {I2, I1, I3:2}
I1	{{I2:4}}	<I2:4>	{I2, I1:4}

Tabulka 2.5 – První dolování FP-stromu (kniha [1]*, str. 244, tab. 5.2)*

Dolování započnu s položkou I5 – poslední ze seznamu L. I5 se nalézá ve dvou větvích FP-stromu: <I2, I1, I5:1> a <I2, I1, I3, I5:1>. Jelikož I5 je sufix, existují dvě prefixové cesty: <I2, I1:1> a <I2, I1, I3:1>, které tvoří podmíněný základ. Podmíněný FP-strom obsahuje ale již jen jednu cestu: <I2:2, I1:1>. I3 tam není obsaženo, protože jeho podpora je rovna 1 a to je méně než podpora minimální. Frekventované množiny získám připojením sufixu I5 ke všem neprázdným podmnožinám, které lze vygenerovat z podmíněného FP-stromu, který obsahuje pouze jednoduchou cestu. Pro sufix I5 dostanu následující frekventované množiny: {I2, I5:2}, {I1, I5:2} a {I2, I1, I5:2}.

I4 má dvě prefixové cesty, které tvoří podmíněný základ I4: {{I2, I1:1}, {I2:1}}. Ten generuje podmíněný FP-strom o jednom uzlu: <I2:2>. Frekventovaná množina je poté pouze jedna: {I2, I4:2}.



Obrázek 2.6 – Schéma podmíněného FP-stromu a tabulky odkazů na uzly (kniha [1]*, str. 245, fig. 5.8)*

Podmíněný základ I3 jsou množiny: {{I2, I1:2}, {I2:2}, {I1:2}}, které vytvoří dvě větve: <I2:4, I1:2>, <I1:2>. To je uvedeno na obrázku 2.6. Vygenerované frekventované množiny jsou poté {I2, I3:4}, {I1, I3:4}, {I2, I1, I3:2}.

Pro I1 se bude postupovat stejným způsobem jako u předchozích položek tabulky odkazů.

3 Návrh konceptu aplikace

Program na dolování asociačních pravidel je napsán jako klientská aplikace, která využívá databázový server jako zdroj informací pro dolování. Aplikace je napsána v objektově orientovaném programovacím jazyce C# pomocí třívrstvého modelu [9], [10], [11], [12], [13] :

- prezentační vrstva,
- výkonná vrstva a
- vrstva pro práci s daty.

Prezentační vrstva obsahuje grafické uživatelské rozhraní. Stará se o interakci aplikace s uživatelem, přebírá vstup od uživatele a zobrazuje výsledky.

Výkonná vrstva se stará o veškeré výpočty. Vytváří dotazy na databázi a zpracovává odpovědi od databázového serveru. Ale pouze skrze vrstvu pro práci s daty.

Vrstva pro práci s daty se stará o připojování k databázi, vykonávání dotazů a čtení výsledků. Pracuje také s konfiguračním XML souborem, který ušetří uživatele od opakovaného zadávání hodnot.

Toto rozdělení mi dává nejen možnost dalšího vývoje této aplikace nezávisle na dalších vrstvách, ale také například možnost implementace libovolné vrstvy do jiných systémů nebo provázání s jazyky pro popis dolovacích úloh, jako je například jazyk PMML (Predictive Model Markup Language) nebo DMSL (The Data Mining Specification Language).

Projekt je ve skutečnosti složen ze 4 částí (ale 3 vrstvy). Tou čtvrtou jsou programové objekty databázového serveru napsané v .NET CLR [9], [14], [15] – takto napsaná uložená procedura mi dává mnohem širší možnosti než deklarativní jazyk SQL, potažmo T-SQL.

Jako databázový server jsem zvolil Microsoft SQL server 2008 v edici Enterprise. Stejně dobře by ale bylo možné použít například Microsoft SQL server 2008 Express, který je volně ke stažení. (Podporu .NET CLR mají všechny edice.)

Databázi pro dolování jsem získal z volně dostupné testovací databáze AdventureWorks 2008. Data z této databáze budou odpovídat určitě více realitě, než kdybych si taková data pseudonáhodně vygeneroval.

4 Implementace

Tato kapitola popisuje implementaci algoritmů na dolování frekventovaných množin a možnosti jejich zrychlení. Dále pak dolování asociačních pravidel s využitím dřívějších znalostí pro zrychlení běhu. V neposlední řadě také práce s vlákny pro běh na pozadí a práce s grafy.

4.1 Dolování frekventovaných množin

Byly implementovány čtyři varianty algoritmu Apriori a dolování FP-stromu pro získávání frekventovaných množin.

4.1.1 Apriori v C#

Jako první jsem implementoval algoritmus Apriori přesně tak, jak byl uveden v teoretické části této práce.

Nejprve bylo potřeba vygenerovat frekventované 1-množiny. Jelikož databázový server umí pracovat s tabulkovými daty mnohem rychleji, napsal jsem si SQL dotaz, který mi vrátí frekventované 1-množiny.

```
SELECT      productID
FROM        ARMTable
GROUP BY    productID
HAVING      COUNT(productID) > @min_support
ORDER BY    productID
```

Z takto vygenerovaných 1-množin jsem začal skládat s pomocí vlastnosti Apriori delší a delší množiny.

Apriori vlastnost umožní rychle zjistit, jestli se takto vygenerovaná množina stane kandidátní množinou. Je nutné ale nejprve vygenerovat všechny její podmnožiny o jedna kratší a ty nalézt mezi již dříve získanými frekventovanými množinami, jinak se kandidátní nestane. Generování všech (k-1)-podmnožin dané k-množiny jsem provedl tak, že jsem vygeneroval k množin, kde v každé chybí jeden prvek z k-množiny.

Bylo potřeba také opakovaně zjišťovat podporu jednotlivých kandidátů, proto jsem si napsal takzvanou „inteligentní property“, která zjišťovala minimální podporu kandidáta pomocí na míru vygenerovaného SQL dotazu, pokud tato hodnota nebyla ještě zjištěna.

Z popisu implementace je tedy patrné, že klientská aplikace neustále odesílala požadavky na databázový server, aby zjistila podporu daného kandidáta.

4.1.2 Apriori v T-SQL

Napadlo mě, že když jsem si ulehčil práci při generování frekventovaných 1-množin použitím SQL dotazu, proč by nešlo napsat celý algoritmus Apriori pomocí jazyka T-SQL (Transact-SQL). Tím by se výpočet přesunul na server, klient by nebyl vůbec namáhán a ulehčení by nastalo také na síti. Počítal jsem také s tím, že operace JOIN bude rychlejší v SŘBD, než když jeho obdobu píšu pomocí jazyka C#. Jazyk T-SQL ale není tak mocný, aby bylo možné obecně zapsat celý algoritmus. Napsal jsem si tedy generátor T-SQL kódu, který jej vytvoří dle zadaných parametrů. Pak jej stačí pouze odeslat na SŘBD a ten mi vrátí frekventované množiny.

Nastává však jeden problém. Po celou dobu výpočtu musí být klient připojen k databázovému serveru. Pokud se spustí výpočet a trvá dlouho, spojení se přeruší a nedozvím se odpověď. Řešením by bylo skriptem vytvořit uloženou proceduru, tu spustit a nechat ji běžet. Výsledek by uložená procedura uložila do tabulky a odtud bych si jej poté mohl převzít. Zvolil jsem ale jinou cestu: zvětšil jsem timeout spojení. Není to ideální řešení, protože se tím na druhou stranu oddaluje zjištění nefunkčního spojení. A v případě, že spojení spadne po delší době, ztratím tím pouze čas a výsledky se nedostaví.

Zjišťování frekventovaných 1-množin a 2-množin pomocí T-SQL:

```
-- frekventovane 1-mnoziny
SELECT
      productID AS AproductID
INTO #frekventovane_1_mnoziny
FROM ARMTable
GROUP BY productID
HAVING COUNT(productID) > @min_support
ORDER BY AproductID


-- frekventovane 2-mnoziny
SELECT
      A.AproductID AS AproductID,
      B.AproductID AS BproductID
FROM
      #frekventovane_1_mnoziny A
      INNER JOIN #frekventovane_1_mnoziny B ON A.AproductID < B.AproductID
WHERE
      @min_support <
      (
            SELECT
                  COUNT(*)
            FROM
                  ARMTable TA
                  LEFT JOIN ARMTable TB ON TA.shoppingCart =
TB.shoppingCart
            WHERE
                  TA.productID = A.AproductID
                  AND
                  TB.productID = B.AproductID
      )
ORDER BY 1, 2
```

Při testování této metody mě napadl ještě jeden způsob, jak by se dala metoda zrychlit. Po každém vygenerování frekventované k-množiny odstraním z databáze všechny záznamy, které nepatří do žádné z frekventovaných množin. Při spojování tabulek pomocí JOIN tím ušetřím velké množství operací. Nejedná se ale o známou techniku mazání takových transakcí, které neobsahují žádné frekventované k-množiny, ale o mazání řádků tabulky takových, které se nevyskytují v žádné frekventované množině. Tento způsob jsem také naimplementoval, ale do finální verze jsem jej nezakomponoval. Zrychlení nebylo patrné, pravděpodobně kvůli režii s mazáním.

Při testování jsem zjistil, že výsledky nejsou nikterak dobré. Nevyužíval jsem vůbec Apriori vlastnosti, která by mi vyřadila spoustu vygenerovaných kandidátních množin, ale počítal jsem pro každého kandidáta jeho podporu a tu porovnával s minimální. Proto jsem se rozhodl tento složitý JOIN (viz úryvek kódu výše) nahradit uloženou procedurou v .NET CLR, která by již respektovala Apriori vlastnost. Zrychlení se dostavilo.

4.1.3 Apriori Itemset

AprioriItemset je vylepšením standardního algoritmu Apriori, kde se k růstu množin používá bitové pole, které má každá položka. Toto bitové pole má právě tolik prvků, kolik obsahuje tabulka databáze transakcí. Aby tato metoda fungovala, je potřeba mít data přichystána transformací tak, aby čísla transakcí vytvářela posloupnost od 0 s krokem 1. (Transformaci jsem si provedl pro zkušební databázi. Jelikož v procesu získávání znalostí se počítá s čištěním, integrací, výběrem a také transformací dat před samotným dolováním z dat, nezabýval jsem se dále převodem.) Jinak by nebylo možné jednoduše indexovat bitové pole pomocí čísla transakce.

V kapitole 5.3.3 jsem provedl měření, které ukazuje důležitost takové transformace.

4.1.4 Apriori TID list

Metoda Apriori TID list již tímto problémem netrpí; místo bitového pole používá seznam transakcí, ve kterých se jednotlivé položky nalézají. Při rozmýšlení, proč by tato metoda měla být rychlejší než Apriori Itemset, jsem dospěl k závěru, že by se dal rozdíl přirovnat k souboru (Itemset) a řídkému souboru (TID list). Na tuto myšlenku mě přivedla technologie Database Snapshot v MS SQL serveru – proč dělat Full backup databáze (jako bitové pole u Itemset), když můžu ukládat pouze rozdíly ve stránkách.

Implementačně se tato metoda liší velice málo od Apriori nebo Aprioti Itemset. Rozdíl je akorát v ukládání kandidátních a frekventovaných množin a v počítání podpory.

4.1.5 Konstrukce a dolování FP-stromu

Tato metoda je založena na rekurzivním dolování FP-stromu, který je ale potřeba nejprve vytvořit. Základem tedy bylo získat z databáze tabulku, která obsahuje čísla nákupního košíku, čísla položek a podporu každé položky. To jsem provedl následující funkcí (pro lepší čitelnost jsem použil Common Table Expressions):

```
CREATE FUNCTION [dbo].[udf_FPTreeBase]
(
      @support INT
)
RETURNS TABLE
AS
RETURN
(
      WITH productWithSupport_CTE (productID, support)
      AS
      (
            SELECT
                  productID,
                  COUNT(productID) AS support
            FROM ARMTable
            GROUP BY productID
            HAVING COUNT(productID) >= @support
      )

      SELECT
            A.shoppingCart,
            A.productID,
            O.support
      FROM
            ARMTable A
            INNER JOIN
            productWithSupport_CTE O
            ON A.productID = O.productID
)
```

Proces výstavby stromu a následně jeho dolování jsem provedl striktně dle algoritmu. Spuštění na malých testovacích datech bylo rychlostí běhu slibné. Pokud jsem ale algoritmus spustil na větších množinách, stal se výrazně pomalejším než Apriori (především u menších podpor). Hledal jsem příčinu a dospěl jsem k tomu, že rekurzivní dolovací algoritmus doluje i položky, které nemají perspektivu. Bylo tedy potřeba strom nějakým způsobem prořezat. Nejlepší metoda, která mě napadla, bylo odstranění těch záznamů z tabulky odkazů, které nesplňují minimální podporu. (Není třeba mazat žádné části stromu a algoritmus dolování se neomezuje žádnou další podmínkou, která by jej akorát brzdila.) Došlo tím nejen k rapidnímu zrychlení při menších hodnotách podpory, ale rovněž při hodnotách vyšších – tam však již není zisk tak významný. V kapitole 5.3.5 je empirický důkaz mého zjištění.

4.2 Omezení délky frekventovaných množin

Algoritmus Apriori nalezne všechny frekventované množiny libovolné délky. Já jsem ale nastavil do implementace omezení, kdy se vždy generují nejvýše k-množiny. Není mi znám žádný postup, kterým by bylo možné předpovědět počet vygenerovaných k-množin a tím i čas běhu výpočtu např. na základě počtu transakcí a selektivitě položek. Proto jsem zvolil možnost volby maximální délky frekventované množiny.

U dolování FP-stromu žádné omezení zvoleno není, protože funguje na úplně jiném principu a také proto, že je to metoda o poznání rychlejší než prvotní Apriori.

4.3 Dolování asociačních pravidel

Generování asociačních pravidel z frekventovaných množin je v podstatě proces generování všech podmnožin dané množiny bez prázdné a sebe sama. Otestování, jestli takto vygenerované pravidlo splňuje minimální spolehlivost a je tudíž silné, již není problém.

Generování pravidel jsem implementoval tak, že jsem vygeneroval pouze polovinu pravidel a druhou polovinu vytvořil převrácením první. Plyne z toho, že jsem musel do seznamu asociačních pravidel vkládat i pravidla slabá, která jsem musel po vygenerování druhé poloviny pravidel smazat. U vytváření druhé poloviny pravidel si již můžu dovolit vkládat pouze silná pravidla.

Tato operace byla ale velice časově náročná. Zrychlení o 60% jsem dosáhl tím, že jsem přestal zjišťovat podporu levé strany pravidla standardním způsobem. Místo toho si podporu vyhledám mezi již získanými frekventovanými množinami.

Výsledek dolování asociačních pravidel je umístěn v tabulce pod záložkou Asociační pravidla včetně spolehlivosti daného pravidla.

4.4 Práce s vlákny

Aby aplikace zůstala i při složitých výpočtech interaktivní, použil jsem na výpočet frekventovaných množin a asociačních pravidel samostatná vlákna. Jelikož GUI aplikace je postaveno na Windows Forms, využil jsem vestavěné třídy BackgroundWork ke spuštění úloh na pozadí a ne standardní třídu Thread. Práce s BackgroundWorkerem mi přišla jednodušší než u vláken, kde bylo potřeba testovat, jestli vlákno ještě běží. Pomocí metody RunWorkerComplete jednoduše zavolám rutinu, která se má provést, až výpočet skončí, a nemusím se dále o nic starat. Abych zajistil korektnost měření délky běhu různých algoritmů, nespouštím nikdy více než jedno vlákno současně pro výpočty frekventovaných množin nebo asociačních pravidel.

Spuštění dolování asociačních pravidel probíhá následovně:

```
BackgroundWorker bw = new BackgroundWorker();

if (rbAprioriA.Checked)
    bw.DoWork += bw_AprioriA;
else if (rbAprioriB.Checked)
    bw.DoWork += bw_AprioriB;
else if (rbAprioriC.Checked)
    bw.DoWork += bw_AprioriC;
else if (rbAprioriD.Checked)
    bw.DoWork += bw_AprioriD;
else if (rbFPTreeA.Checked)
    bw.DoWork += bw_FPTreeA;

bw.DoWork += bw_ARMining;
bw.RunWorkerCompleted += bw_MiningCompleteShowAR;

bw.RunWorkerAsync();
```

4.5 Grafy

Na tvorbu grafů jsem použil volně dostupný nástroj Microsoft Chart Controls, který integrací do vývojového prostředí Microsoft Visual Studio vytvoří komponentu pro práci s grafy. Jako jeden z mála volně dostupných nástrojů, které jsem našel, umí vytvářet trojrozměrné grafy (osy metoda, podpora a čas běhu). A to včetně průhlednosti, která je pro čitelnost grafu velkým přínosem.

5 Spuštění na reálných datech a zhodnocení výsledků

V této kapitole jsou podrobeny jednotlivé modifikace algoritmů testům, které empiricky dokazují tvrzení a předpoklady z předchozích kapitol. Nejedná se pouze o porovnání algoritmů mezi sebou, ale také porovnání jednoho algoritmu samého se sebou s a bez konkrétní optimalizace.

5.1 Testovací data

Jako zdroj dat jsem použil tabulku Sales.SalesOrderDetail (sloupce SalesOrderID a ProductID) z databáze AdventureWorks2008, kterou Microsoft poskytuje zdarma ke stažení. Data jsou zde již velice dobře předpřipravená, takže není potřeba provádět žádné čištění, integraci více zdrojů, redukce nebo transformace. Tabulka obsahuje přes 121 tisíc řádků, kde je přes 31 tisíc transakcí a 266 položek. Pro testování si vybírám vždy pouze část této tabulky.

5.2 Metodika měření

Pro měření času jsem si vytvořil třídu Timer, která odečítá aktuální čas při začátku a konci výpočtu. Rozdíl v milisekundách následně vrátí. Není možné měřit pouze běh aplikace, protože spousta operací se provádí v databázi a to by ovlivnilo výsledky. Takto je čas sice započítán včetně běhu systému a dalších aplikací, ale je to u všech „stejně“ (všechny testy byly provedeny v jeden čas), tudíž poměr rychlostí jednotlivých metod je zachován.

Další sledovanou veličinou je počet vygenerovaných frekventovaných množin, popřípadě asociačních pravidel.

Testy byly prováděny na procesoru Intel Core 2 Duo Mobile P8400 s operační pamětí 4GB na OS Windows 7 RC1.

5.3 Měření

Vypracoval jsem celkem 6 měření, pomocí kterých jsou srovnávány jednotlivé metody dolování jak frekventovaných množin, tak asociačních pravidel.

5.3.1 Měření 1 – Dolování asociačních pravidel

V prvním měření se budu zabývat dolováním asociačních pravidel. Budu sledovat celkem 4 parametry:

- Doba výpočtu frekventovaných množin,
- počet frekventovaných množin,
- doba výpočtu asociačních pravidel a
- počet asociačních pravidel.

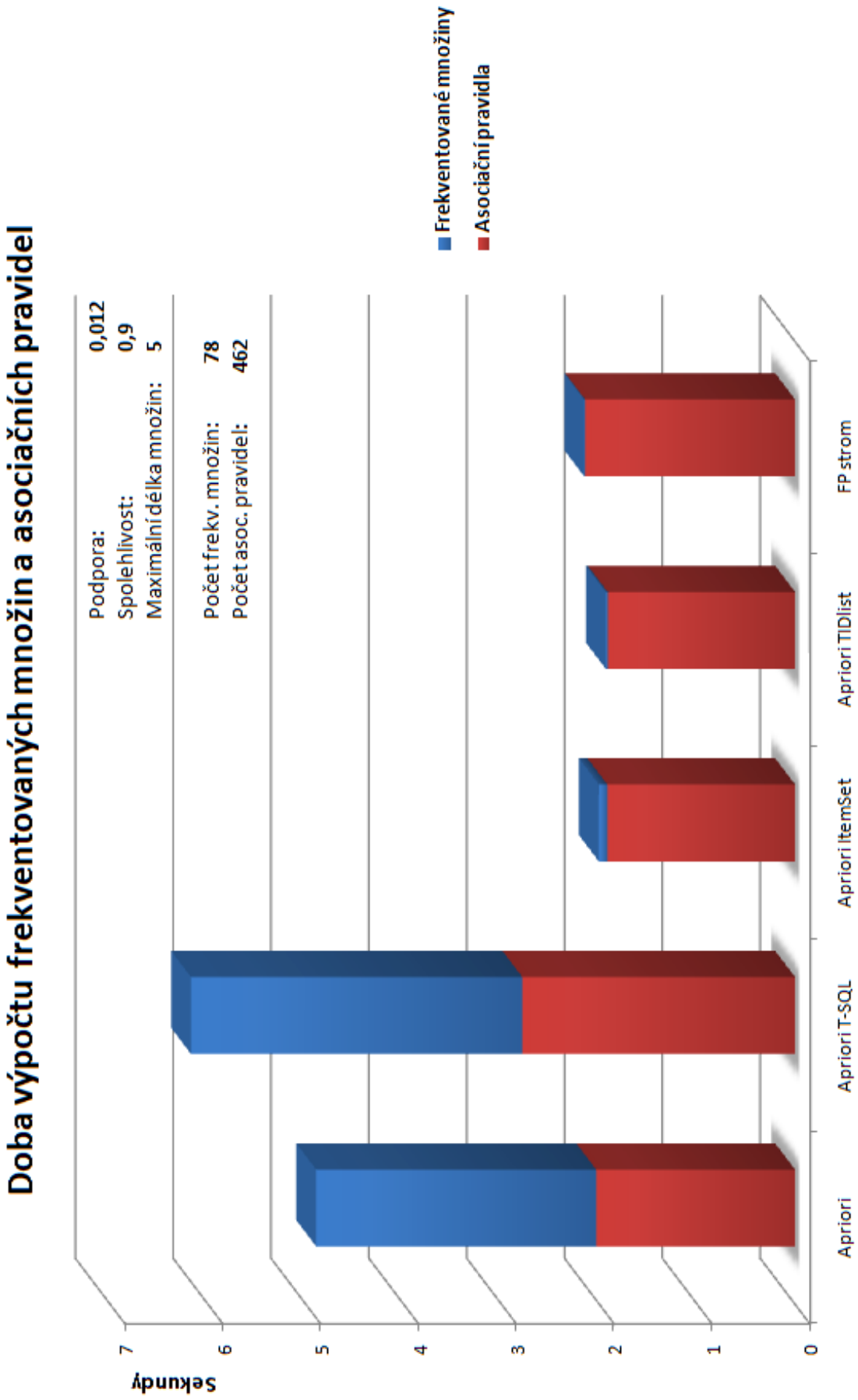
Všechny měřené veličiny jsou závislé na vstupních parametrech:

- Minimální podpora frekventovaných množin,
- minimální spolehlivost asociačních pravidel,
- maximální délka frekventované množiny (kromě FP-stromu) a
- data pro dolování (pokud nebude uvedeno jinak, vzorek dat se nemění).

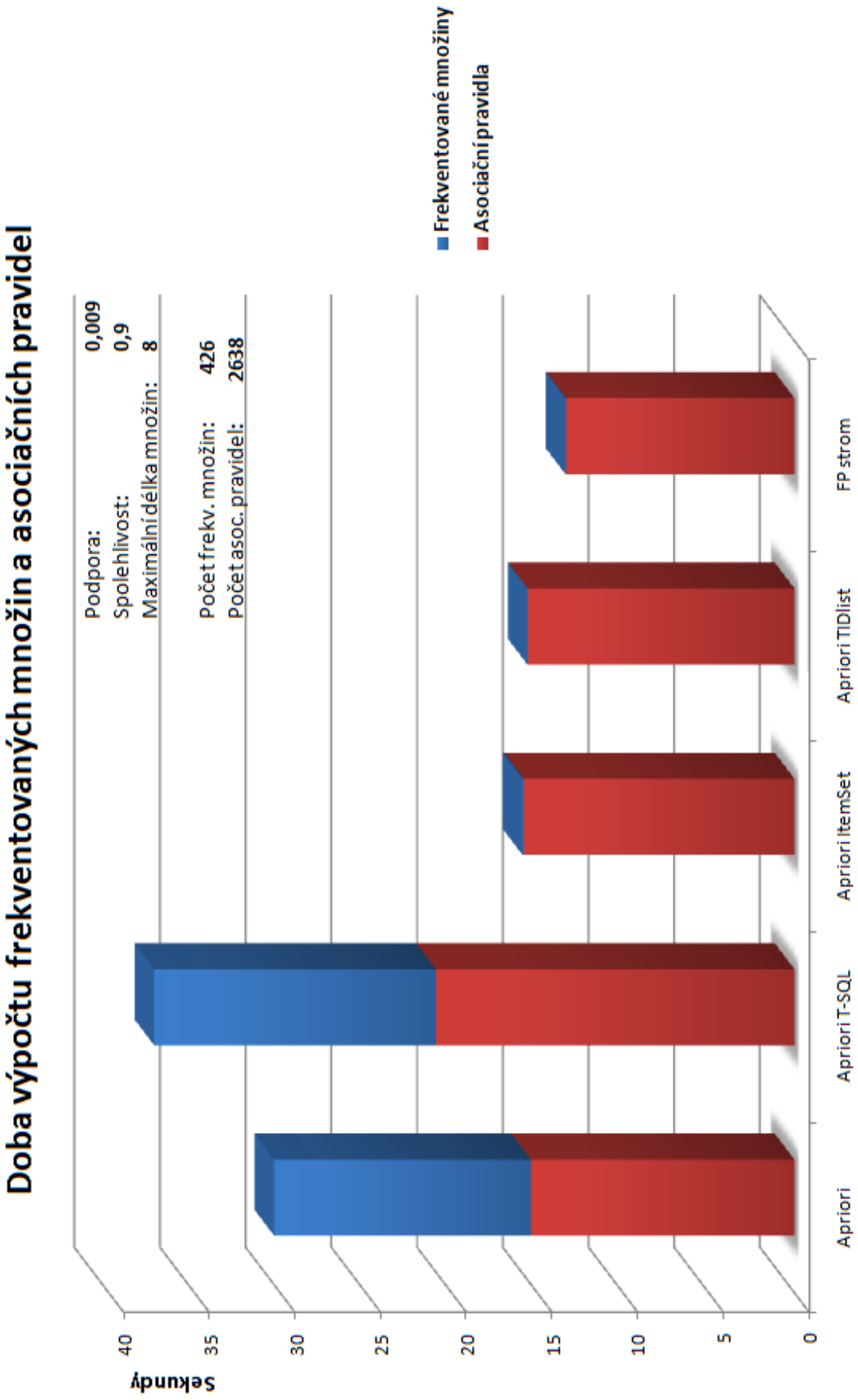
Pouze doba výpočtu frekventovaných množin je závislá na zvolené metodě pro dolování.

Doba výpočtu asociačních pravidel by měla být stejná pro různé metody dolování frekventovaných množin. S ohledem na tento předpoklad je sestrojen i graf 6.1, který ukazuje podobnost časů výpočtu frekventovaných množin. Čas u Apriori T-SQL se však liší od ostatních. Je to způsobeno tím, že tato metoda mi nevrací podporu jednotlivých frekventovaných množin. Je potřeba podporu spočítat zvlášť. Díky návrhu aplikace (takzvaná „inteligentní property“ pro výpočet podpory) nebylo potřeba nijak upravovat aplikaci.

U grafu 6.2 se změnou podpory změnil počet frekventovaných množin a tím počet asociačních pravidel. Nastal zde vzrůst doby trvání výpočtu asociačních pravidel. Opět jsou ale doby výpočtu vesměs stejné až na Apriori T-SQL.



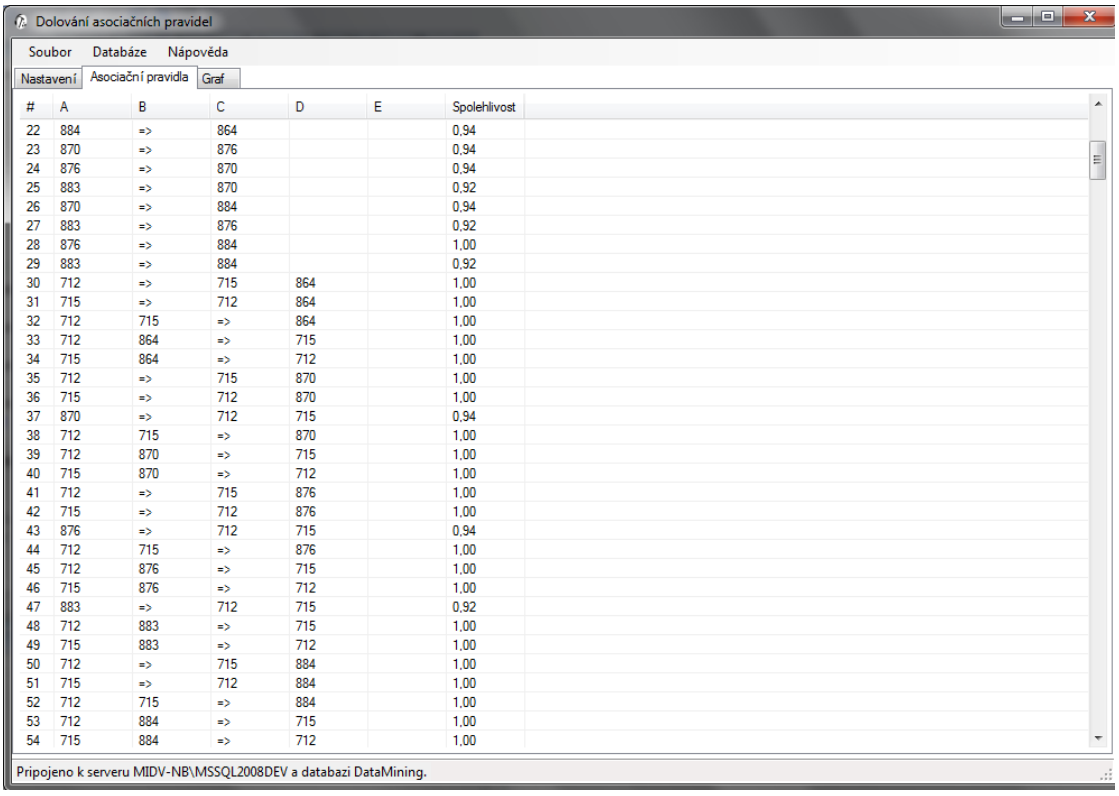
Graf 6.1 – Doba výpočtu frekventovaných množin a asociačních pravidel



Graf 6.2 – Doba výpočtu frekventovaných množin a asociačních pravidel

5.3.2 Měření 2 – Dolování asociačních pravidel

Druhé měření se bude také zabývat asociačními pravidly, tentokráte již ale ne statistikou. Obrázek ukazuje výběr pár asociačních pravidel. Pro úplný přehled je potřeba si spustit aplikaci.

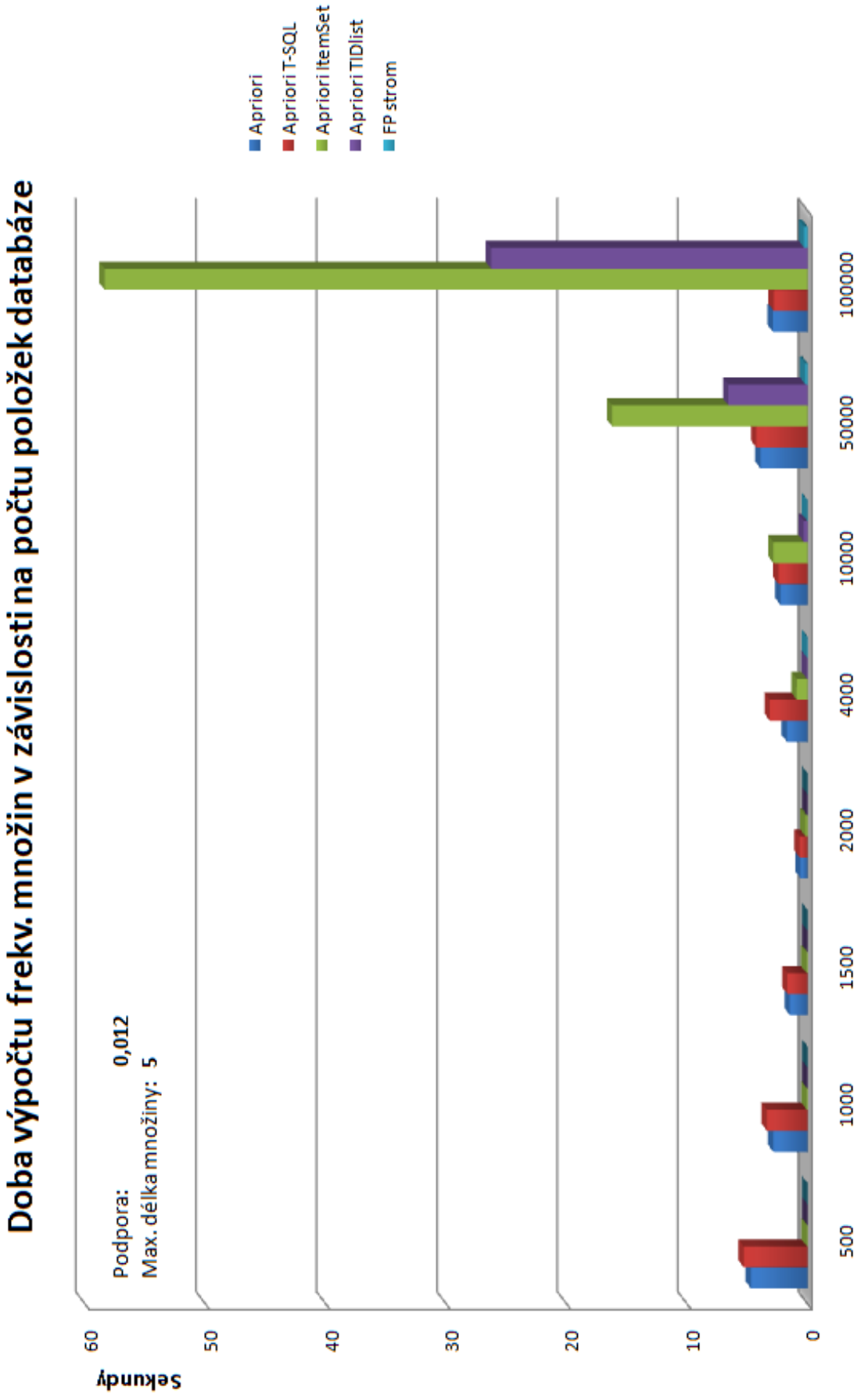


#	A	B	C	D	E	Spolehlivost
22	884	=>	864			0,94
23	870	=>	876			0,94
24	876	=>	870			0,94
25	883	=>	870			0,92
26	870	=>	884			0,94
27	883	=>	876			0,92
28	876	=>	884			1,00
29	883	=>	884			0,92
30	712	=>	715	864		1,00
31	715	=>	712	864		1,00
32	712	715	=>	864		1,00
33	712	864	=>	715		1,00
34	715	864	=>	712		1,00
35	712	=>	715	870		1,00
36	715	=>	712	870		1,00
37	870	=>	712	715		0,94
38	712	715	=>	870		1,00
39	712	870	=>	715		1,00
40	715	870	=>	712		1,00
41	712	=>	715	876		1,00
42	715	=>	712	876		1,00
43	876	=>	712	715		0,94
44	712	715	=>	876		1,00
45	712	876	=>	715		1,00
46	715	876	=>	712		1,00
47	883	=>	712	715		0,92
48	712	883	=>	715		1,00
49	715	883	=>	712		1,00
50	712	=>	715	884		1,00
51	715	=>	712	884		1,00
52	712	715	=>	884		1,00
53	712	884	=>	715		1,00
54	715	884	=>	712		1,00

Obrázek 6.3 – Snímek aplikace: vydolovaná asociační pravidla

5.3.3 Měření 3 – Dolování frekventovaných množin

Minulým měřením jsem empiricky dokázal, že doba výpočtu asociačních pravidel se nemění s různými metodami dolování frekventovaných množin při stejných vstupních parametrech. Tudíž další měření budu provádět pouze na frekventovaných množinách. Předchozí pokusy jsem prováděl na testovacím vzorku 1000 záznamů v databázi. Nyní provedu test všech metod nad vzrůstajícím počtem záznamů s konstantní relativní podporou a maximální délkou množin 5. Nyní se teprve ukáže, jak rychlé jsou jednotlivé algoritmy.



Graf 6.4 – Doba výpočtu frekventovaných množin v závislosti na počtu položek databáze

Konstantní relativní podpora se podepsala na výsledku tím způsobem, že se délka výpočtu u většiny metod příliš nelišila (Zpracovávalo se sice více dat, ale vzrostla adekvátně také absolutní podpora.). Razantní vzestup složitosti se ale projevil na metodě Apriori Itemset. Je z ní patrné, že zvětšující se bitové pole se výrazně podílí na délce výpočtu. Abych si ověřil lépe tento závěr, zjistil jsem si pomocí jednoduchého SQL dotazu počet transakcí při různých velikostech zdrojové tabulky.

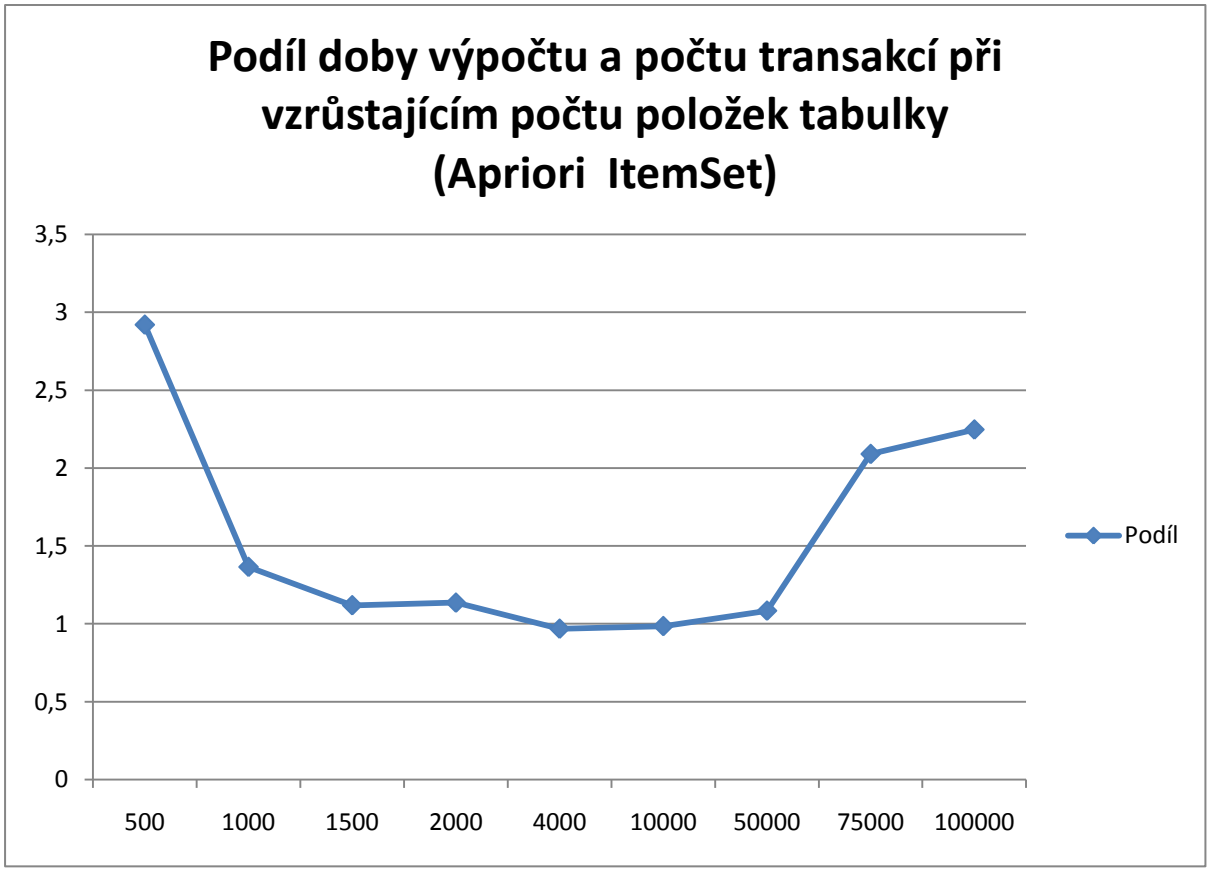
```
SELECT COUNT(DISTINCT shoppingCart) FROM ARMTable
```

Výsledek tohoto dotazu pro vybrané počty řádků tabulky:

Počet řádků tabulky	**Počet transakcí**
1 000	63
10 000	2889
50 000	14 978
100 000	25 983

Tabulka 6.5 – Výsledek dotazu na získání počtu unikátních transakcí v databázi

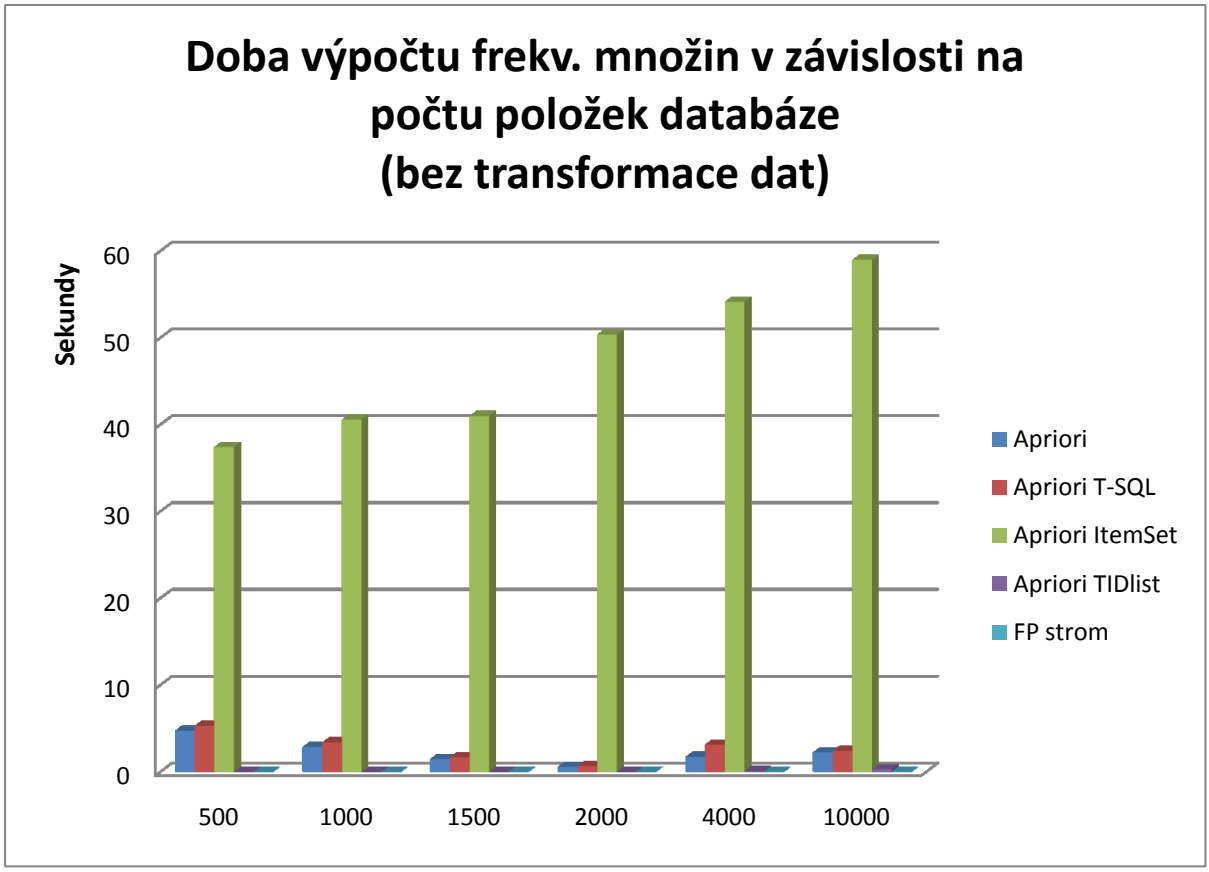
Vezmu-li první a poslední řádek této tabulky a porovnám je, tak počet transakcí se zvedl 412 krát, zatímco délka výpočtu 678 krát (tento údaj není v tabulce uveden). Rozhodl jsem se tedy sestrojit graf, který by zobrazil podíl doby výpočtu a počtu transakcí při vzrůstajícím počtu položek tabulky.



Graf 6.6 – Podíl doby výpočtu a počtu transakcí při vzrůstajícím počtu položek tabulky

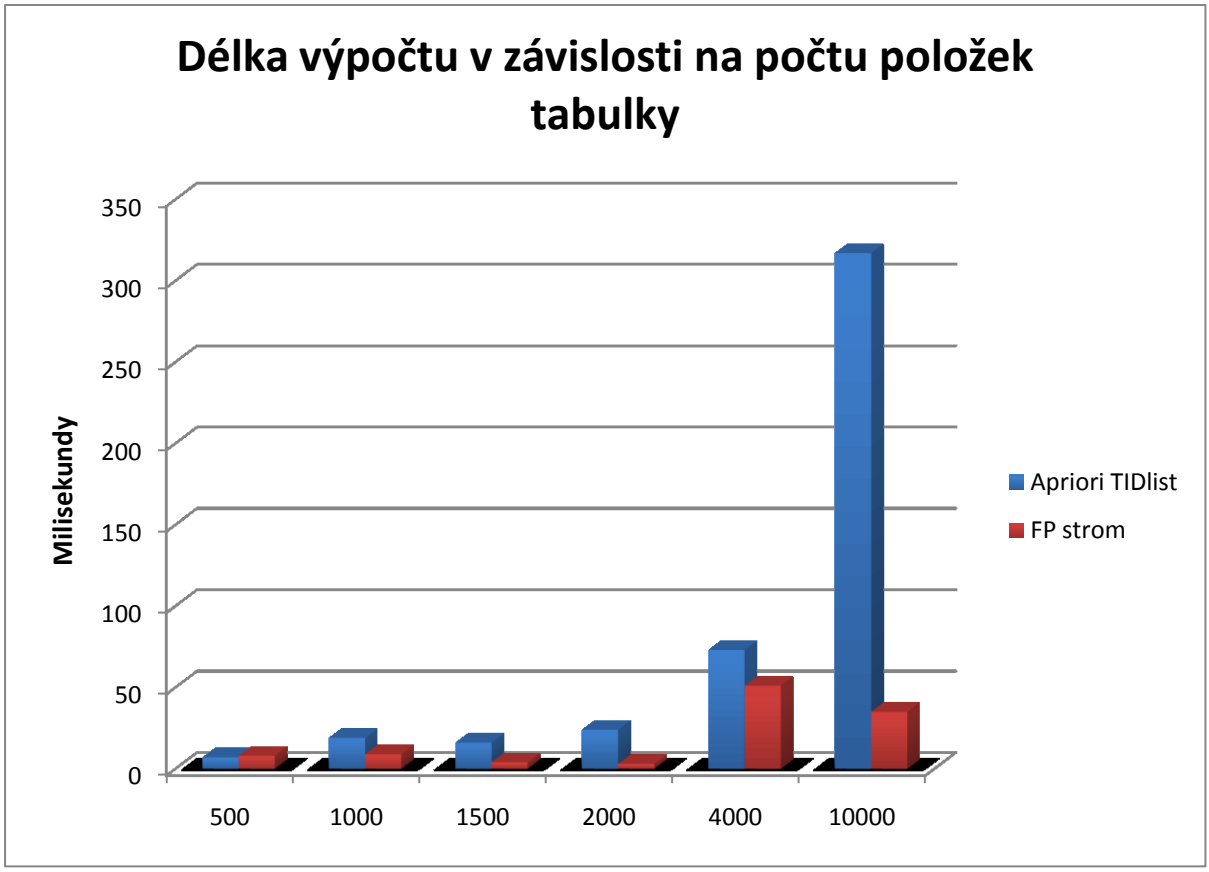
Jelikož graf vyjadřuje podíl, zobrazí se na něm přímá úměra jako konstantní funkce. To by odpovídalo střední části grafu. Krajní hodnoty tomu ale neodpovídají, nelze tedy o přímé úměře, ani jiném lineárním trendu uvažovat. Byl by potřeba mnohem větší vzorek dat, aby se dala usoudit nějaká závislost.

Výše jsem uvedl, jak je důležité před použitím Apriori Itemset udělat transformaci tabulky tak, aby čísla transakcí tvořila vzestupnou posloupnost od nuly s krokem 1. Nechtěně jsem jednou při kopírování dat ze záložní tabulky do pracovní zapomněl spustit transformační proceduru. Výsledky byly v tu chvíli překvapující. Uvedu zde tedy příklad, jak se projeví neprovedení transformace na výsledných časech.



Graf 6.7 – Doba výpočtu frekv. množin v závislosti na počtu položek databáze (bez transformace dat)

Další metoda, u které byl patrný růst, je Apriori TID list. Zkusil jsem tedy tuto metodu porovnat s FP-stromem. Jelikož výše uvedený graf zaujímá příliš velký rozsah, vytvořil jsem graf pouze s Apriori TID list a FP-stromem a to pouze do 10 000 položek, aby byly dobře patrné i hodnoty u FP-stromu.



Graf 6.8 – Doba výpočtu v závislosti na počtu položek tabulky

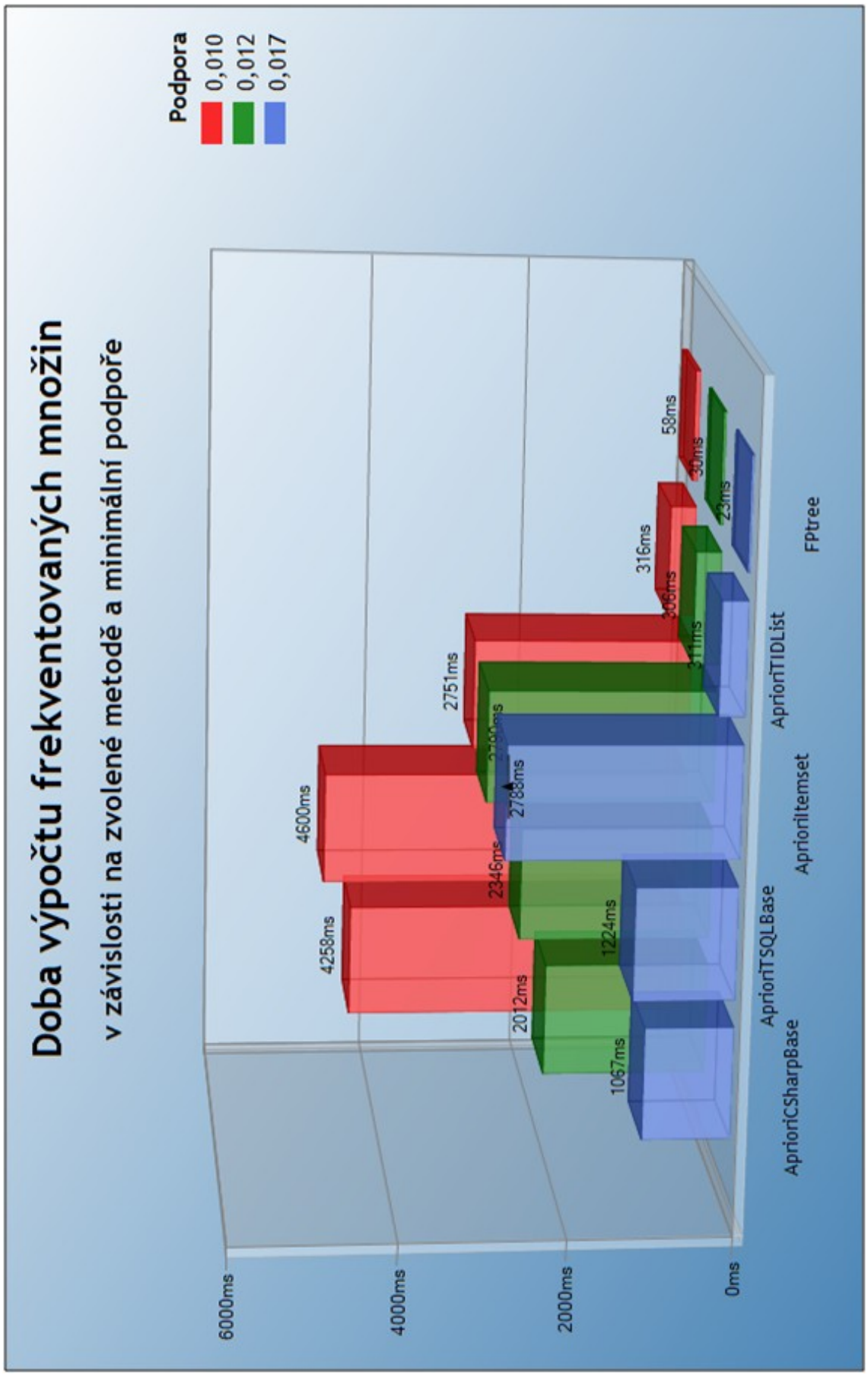
Pro malý počet položek tabulky se stává, že Apriori TID list je rychlejší než metoda založena na dolování FP-stromu. Zatímco pro větší hodnoty hraje prim metoda s pomocí FP-stromu.

5.3.4 Měření 4 – Dolování frekventovaných množin

Samotná aplikace umí vykreslovat z mého pohledu nejzajímavější graf, a to dobu výpočtu frekventovaných množin v závislosti na zvolené metodě a minimální podpoře.

Jelikož se výpočet provádí nad shodnou množinou dat, ukáže se na tomto grafu dobře rychlost jednotlivých algoritmů. Ty jsou srovnány v jednom řádku a označeny shodnou barvou.

S rostoucí podporou by také měly klesat časy, anebo zůstat alespoň stejné. To se právě děje u algoritmu Apriori Itemset. Jeho tvůrci využili bitové pole a nad ním velmi rychlé bitové operace. Výstavba takového pole je již ale náročná a to se projevilo právě v grafu (vždy se musí vytvořit bitové pole rovné počtu transakcí v tabulce databáze a to je nezávislé na zvolené podpoře).

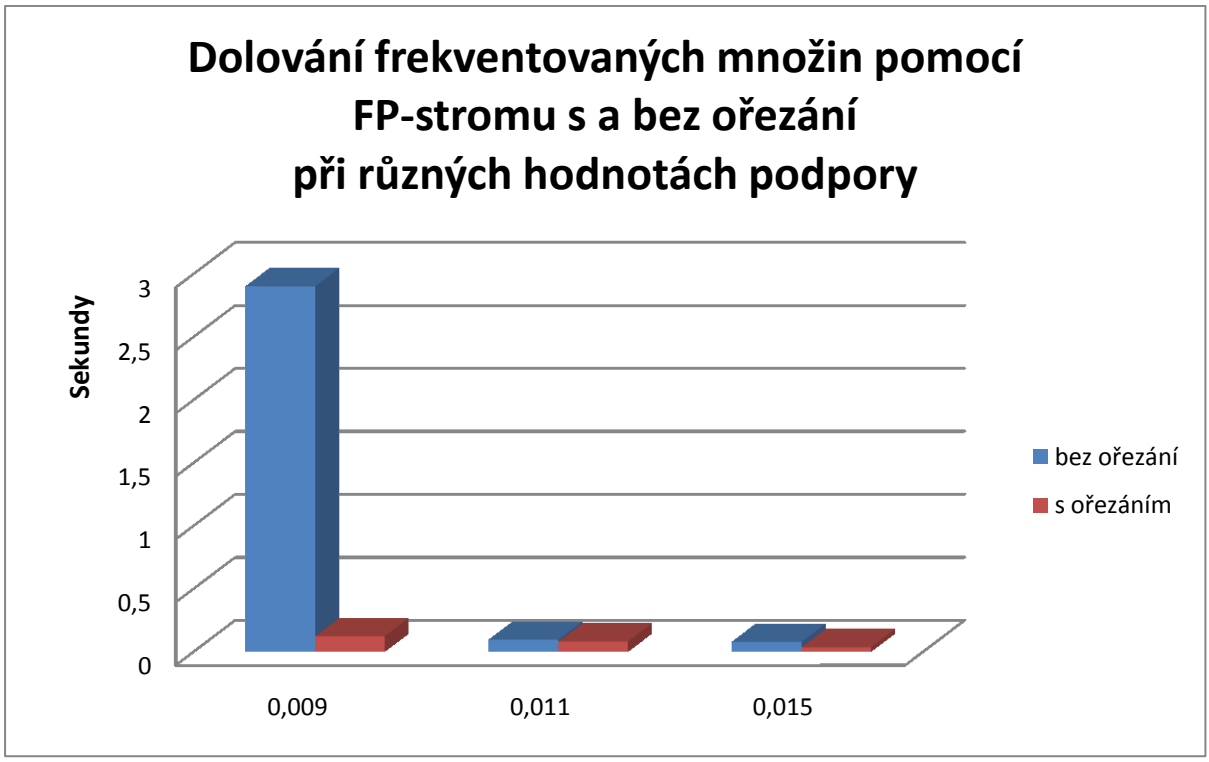


Graf 6.9 – Doba výpočtu frekventovaných množin v závislosti na zvolené metodě a minimální podpoře

5.3.5 Měření 5 – Dolování frekventovaných množin

Jak jsem uvedl dříve, obecný algoritmus dolování frekventovaných množin pomocí FP-stromu nemá při malých podporách dobré výsledky. Prořezáním stromu jsem ale docílil výrazného zrychlení algoritmu při malých hodnotách podpory. Pokud nedojde k prořezávání, vytvářejí se zbytečně podmíněné FP-stromy, ze kterých nikdy frekventované množiny nevzniknou.

Empirickým důkazem tohoto tvrzení je tento graf, který zobrazuje dobu trvání dolování frekventovaných množin pomocí FP-stromu s a bez ořezání při různých hodnotách podpory.



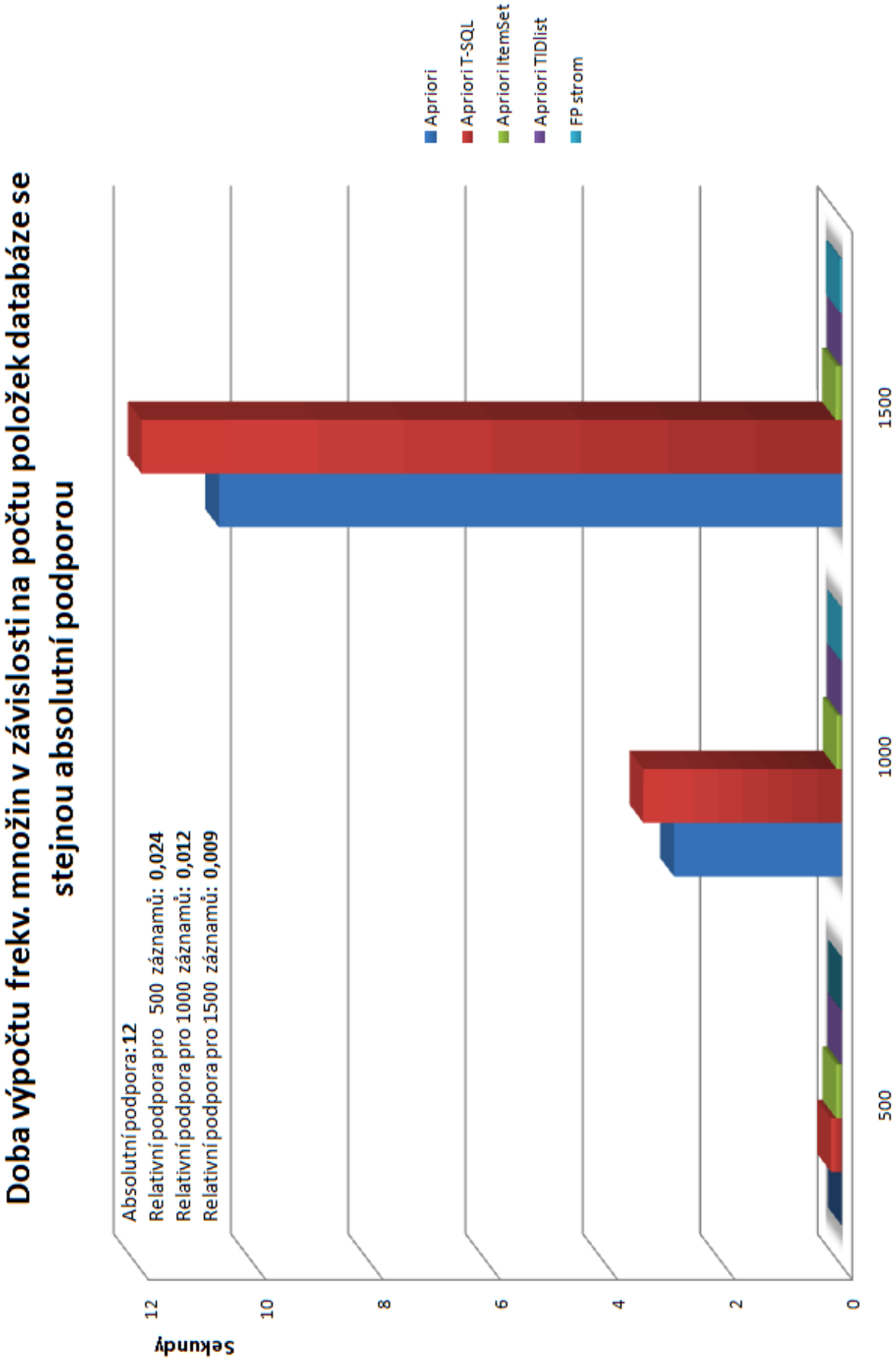
Graf 6.10 – Dolování frekventovaných množin pomocí FP-stromu s a bez ořezání při různých hodnotách podpory

5.3.6 Měření 6 – Dolování frekventovaných množin

Všechna předchozí měření byla prováděna s relativní podporou. Tzn., že absolutní hodnota minimální podpory se měnila s měnícím se počtem řádků tabulky.

V tomto měření jsem se zaměřil na vztah mezi časem a metodami, pokud se bude měnit počet řádků tabulky a hodnota podpory bude stále stejná. Jde vlastně o dvojí ztížení podmínek. Na jednu stranu se zvedá počet řádků, které se musí zpracovat, na druhou stranu se nezvětšuje adekvátně minimální podpora, která by snížila počet frekventovaných množin. Toto měření zvýraznilo rozdíly, které mezi jednotlivými metodami jsou.

Jako odrazový můstek pro měření jsem zvolil tabulku s 1 000 řádky a relativní podporu 0,012. Absolutní podpora tedy je 12.



Graf 6.11 – Doba výpočtu frekv. množin v závislosti na počtu položek databáze se stejnou absolutní podporou

6 Závěr

Tato práce se zabývala problematikou získávání asociačních pravidel z databází. Cílem práce bylo vytvořit aplikaci, která by umožňovala porovnávat výkonnost a časovou náročnost několika zvolených algoritmů pro dolování těchto pravidel. Jednalo se zejména o algoritmus Apriori a jeho modifikace a dále pak o algoritmus využívající FP-strom.

Součástí práce je tedy modulární aplikace, která se skládá ze tří vrstev: prezentační, výkonná a datová. Prezentační vrstva se stará o grafické uživatelské rozhraní – zadávání parametrů výpočtů a následné zobrazení výsledků. Parametry se z prezentační vrstvy předávají výkonné vrstvě, která provede samotné dolování frekventovaných množin a asociačních pravidel a navrátí odpovídající výsledek. Nejnižší z vrstev, datová, se stará o získávání dat z databáze. Díky této modularitě je možné aplikaci rozšiřovat o další algoritmy nebo jednotlivé vrstvy nahradit jinými.
Aplikace umí vytvářet grafy, které je možné využít k živým ukázkám při výuce dolování asociačních pravidel.

Nejslibnější budoucnost z mnou implementovaných algoritmů má metoda dolování FP-stromu. Proto se také první námět na zlepšení bude týkat této metody.

FP-strom se vždy doluje od spodu, ale staví se seshora – v každém uzlu je seznam jeho následníků. Seznam následníků nám sice ulehčí vytváření stromu, ale také zabírá zbytečné místo. Dle článku [6] bych rád v budoucnu přepracoval tvorbu FP-stromu bez seznamu následníků u každého uzlu. Dosáhl bych tím ušetření paměti, které může být při velké vstupní databázi nedostatek. Bude nezbytně nutné experimentálně ověřit, zda snížení prostorové náročnosti nepovede k výraznému nárůstu časové náročnosti výstavby stromů.

Při implementaci Apriori T-SQL se kompletní výpočet prováděl na straně databázového serveru. To server samozřejmě velmi zatěžuje. Např. v nočních hodinách by to nad produkční databází nebyl problém, ale přes den (např. v pracovní době) by to bylo nežádoucí. Bylo by tedy dobré využít Resource Governor v MS SQL serveru 2008, který by omezil výpočetní výkon a operační paměť této náročné operaci.

Jelikož některé výpočty mohou běžet velice dlouho, bylo by také dobré umožnit uživateli výpočet zastavit. Kdyby bylo možné předpovědět délku běhu ze vstupních parametrů, mohly by se parametry před spuštěním výpočtu upravit tak, abych dostal žádoucí výsledek. Problém se zastavením je ale v tom, že kvůli neznalosti pokročilosti výpočtu může zastavit výpočet chvíli před dokončením.

Literatura

1. **Han, Jiawei a Kamber, Micheline.** *Data Mining - Concepts and Techniques.* 2006. ISBN: 1-55860-901-6 .

2. **Berka, Petr.** *Dobývání znalostí z databází.* místo neznámé : Academia, 2003. ISBN: 80-200-1062-9.

3. **Berry, Michael J. A. a Linoff, Gordon.** *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management.* místo neznámé : Wiley Computer Publishing, 2004. ISBN: 978-0471470649.

4. **MacLennan, Jamie, Tang, ZhaoHui a Crivat, Bogdan.** *Data Mining with Microsoft SQL Server 2008.* místo neznámé : Wiley, 2008. ISBN: 978-0470277744.

5. **Linoff, Gordon.** *Data Analysis Using SQL and Excel.* místo neznámé : Wiley, 2007. ISBN: 978-0470099513.

6. **Borgelt, Christian.** An Implementation of the FPgrowth. *Christian Borgelt's Webpages.* [Online] [Citace: 15. 4 2009.] http://www.borgelt.net/papers/fpgrowth.pdf.

7. **Tan, Pang-Ning, Steinbach, Michael a Kumar, Vipin.** *Introduction to Data Mining.* místo neznámé : Addison Wesley, 2005. ISBN: 978-0321321367.

8. **Goethals, Bart.** Survey on Frequent Pattern Mining. *Mathematics and Computer Science Department, University of Antwerp.* [Online] [Citace: 20. 3 2009.] http://www.adrem.ua.ac.be/~goethals/software/survey.pdf.

9. **Agarwal, Vidya Vrat, a další.** *Beginning C# 2008 Databases.* místo neznámé : Apress, 2008. ISBN: 978-1590599006.

10. **Marshall, Donis.** *Programming Microsoft® Visual C#® 2008: The Language.* místo neznámé : Microsoft Press, 2008. ISBN: 978-0735625402.

11. **Nagel, Christian, a další.** *Professional C# 2008.* místo neznámé : Worx, 2008. ISBN: 978-0470191378.

12. **Troelsen, Andrew.** *Pro C# 2008 and the .NET 3.5 Platform, Fourth Edition.* místo neznámé : Apress, 2007. ISBN: 978-1590598849.

13. **Wagner, Bill.** *More Effective C#.* místo neznámé : Addison-Wesley Professional, 2008. ISBN: 978-0321485892.

14. **Stanek, William.** *Microsoft SQL Server 2005 Administrator's Pocket Consultant.* místo neznámé : Microsoft Press, 2005. ISBN: 978-0735621077.

15. **Vieira, Robert.** *Beginning SQL Server 2005 Programming.* místo neznámé : Wrox, 2006. ISBN: 978-0764584336.

Seznam obrázků, vzorců a tabulek

Seznam příloh