VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

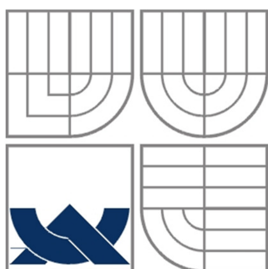
DOLOVÁNÍ SEKVENČNÍCH VZORŮ

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

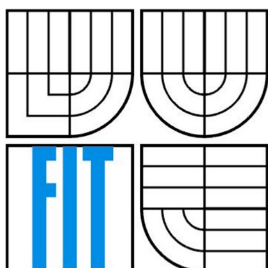
AUTOR PRÁCE
AUTHOR

Bc. ZDENĚK TISOŇ

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

DOLOVÁNÍ SEKVENČNÍCH VZORŮ
SEQUENTIAL PATTERN MINING

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE — Bc. ZDENĚK TISOŇ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN HLOSTA
SUPERVISOR

BRNO 2012

Abstrakt

Tato diplomová práce je zaměřena na problematiku získávání znalostí z databází, především pak na metody dolování sekvenčních vzorů. Jednotlivé metody dolování sekvenčních vzorů jsou zde popsány detailně. Dále se práce zabývá rozšířením analytických služeb platformy Microsoft SQL Server o nové dolovací algoritmy. V praktické části této práce jsou implementovány rozšíření pro dolování sekvenčních vzorů na platformě MS SQL Server. V poslední části jsou vytvořené algoritmy porovnány nad různými datovými sadami.

Abstract

This master's thesis is focused on knowledge discovery from databases, especially on methods of mining sequential patterns. Individual methods of mining sequential patterns are described in detail. Further, this work deals with extending the platform Microsoft SQL Server Analysis Services of new mining algorithms. In the practical part of this thesis, plugins for mining sequential patterns are implemented into MS SQL Server. In the last part, these algorithms are compared on different data sets.

Klíčová slova

Získávání znalostí z databází, dolování z dat, sekvenční vzory, frekventované podsekvence, uzavřené sekvence, GSP, PrefixSpan, SPAM, LAPIN-SPAM, BIDE, MS SQL Server, analytické služby.

Keywords

Knowledge discovery in databases, data mining, sequential patterns, frequent subsequences, closed sequences, GSP, PrefixSpan, SPAM, LAPIN-SPAM, BIDE, MS SQL Server, Analysis Services.

Citace

Tisoň Zdeněk: Dolování sekvenčních vzorů, diplomová práce, Brno, FIT VUT v Brně, 2012.

Dolování sekvenčních vzorů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Martina Hlosty. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……………………
Zdeněk Tisoň
15. května 2012

Poděkování

Na tomto místě bych rád poděkoval Ing. Martinu Hlostovi za poskytnutou pomoc, odborné připomínky a rady při tvorbě této práce.

© Zdeněk Tisoň, 2012.

Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

Obsah 1
1 Úvod 3
2 Získávání znalostí z databází 5
2.1 Proces získávání znalostí z databází 5
2.2 Druhy dat pro dolování 6
2.3 Typy dolovacích úloh 7
2.3.1 Popis konceptu/třídy 7
2.3.2 Frekventované vzory a asociační pravidla 8
2.3.3 Klasifikace a predikce 8
2.3.4 Shluková analýza 9
2.3.5 Analýza odlehlých objektů 10
2.3.6 Evoluční analýza 10
3 Dolování sekvenčních vzorů 11
3.1 Definice problému 11
3.2 Aplikace sekvenčních vzorů 12
3.3 Algoritmy pro dolování sekvenčních vzorů 14
3.3.1 Analýza existujících algoritmů 14
3.3.2 GSP 16
3.3.3 PrefixSpan 19
3.3.4 SPAM 22
3.3.5 LAPIN-SPAM 27
3.3.6 BIDE 29
3.4 Sekvenční pravidla 31
4 Dolování na platformě SQL Server 33
4.1 Microsoft SQL Server 33
4.2 Dolování z dat 34
4.2.1 Vytvoření dolovací úlohy 34
4.2.2 Vestavěné algoritmy 36
4.3 Rozšíření analytických služeb SQL Serveru 37
4.3.1 Životní cyklus algoritmu 38
4.3.2 GSP rozšíření 40
5 Návrh a implementace 42
5.1 Knihovna pro dolování sekvenčních vzorů 42
5.1.1 Implementace algoritmu PrefixSpan 44

5.1.2 Implementace algoritmu SPAM 46
5.1.3 Implementace algoritmu LAPIN-SPAM 51
5.1.4 Implementace algoritmu BIDE 52
5.2 Implementace rozšíření SQL Serveru 54
5.2.1 Prezentace sekvenčních vzorů a pravidel 57
5.2.2 Parametry rozšiřujících algoritmů 58
6 Experimenty 60
6.1 Ověření funkčnosti algoritmů 60
6.2 Porovnání algoritmů 60
6.2.1 Syntetická datová sada *D10KC8T8S8I8N1K* 61
6.2.2 Syntetická datová sada *D50KC20T20S20I10N200* 62
6.2.3 Reálná datová sada *Gazelle* 64
6.2.4 Reálná databáze virů 65
6.3 Vyhodnocení výsledků experimentů 66
7 Návrh na další postup 67
8 Závěr 68
Literatura 69
Seznam příloh 72
Příloha A: Výsledky porovnání algoritmů 73
Syntetická datová sada *D5KC20T20S20N1K* 73
Syntetická datová sada *D10KC10T2.5N10KS6I2.5* 74
Příloha B: Ukázka PMML dokumentu 75
Příloha C: Obsah přiloženého CD 77

1 Úvod

V dnešní moderní době znalost informací znamená konkurenční výhodu. Každý náš nákup, telefonát přístup na webovou stránku nebo návštěva lékaře produkují data, která jsou nejrůznějšími společnostmi zaznamenávána. A právě přeměnou těchto dat na nové znalosti se mohou organizace stát konkurenceschopné. Tradiční databázové systémy však nenabízí možnost extrakce nových znalostí z uložených dat. Proto bylo zapotřebí vyvinout nový obor, který se tímto zabývá.

Získávání znalostí z databází je obor, který tento nedostatek řeší. Tento obor si klade za cíl vyhledat z velkého množství dat skryté, netriviální a užitečné informace. Typickou aplikací tohoto oboru je analýza nákupního košíku. Ta přináší informace o tom, jaké zboží si zákazníci nejčastěji nakupují dohromady. Prodejce pak může využít této informace a dané zboží umístit vedle sebe, nebo naopak co nejdále od sebe. Možným rozšířením tohoto příkladu je zjištění, v jakém pořadí si zákazníci dané zboží koupí. Tyto znalosti jsou reprezentovány pomocí *sekvenčních vzorů*.

Druhá kapitola této práce představuje proces získávání znalostí z databází. Seznámí čtenáře s jednotlivými fázemi tohoto komplexního procesu. Shrnuje datová úložiště a úlohy, které se dají využít pro získávání znalostí. Převážně se však práce zaměřuje na získávání sekvenčních vzorů.

Sekvenční vzor je seřazená posloupnost událostí, která se v analyzovaných datech vyskytuje frekventovaně. Událost může být například návštěva lékaře, bezpečnostní průnik nebo přístup na webovou stránku. Získané sekvenční vzory mají převážně popisný charakter, mohou se však využít i pro předpověď budoucích hodnot. Například zkoumáním symptomů a nemocí pacientů můžeme v brzké fázi díky prvním příznakům detekovat a odvrátit vážnou hrozící chorobu. Třetí kapitola popisuje a analyzuje existující algoritmy pro hledání sekvenčních vzorů.

Existuje mnoho nástrojů pro nasazení procesu získávání znalostí z databází. Čtvrtá kapitola představuje robustní a flexibilní platformou Microsoft SQL Server. Jednou z částí této platformy jsou analytické služby. Tyto služby nabízí kvalitní podporu pro nasazení procesu získávání znalostí z databází. Práce seznamuje čtenáře s těmito službami a také se zabývá možnostmi rozšíření prostředí SQL Serveru o vlastní dolovací algoritmy.

Na základě analýzy existujících přístupů pro hledání sekvenčních vzorů byly pro implementaci vybrány algoritmy PrefixSpan, SPAM, LAPIN-SPAM a BIDE. Algoritmy byly vytvořeny jako rozšíření analytických služeb SQL Serveru. Pátá kapitola se zabývá postupem při implementaci těchto algoritmů.

V šesté kapitole jsou implementované algoritmy porovnány nad různými datovými sadami. Algoritmy byly také porovnány s existující implementací algoritmu GSP. Experimenty byly prováděny na syntetických i reálných datových sadách. Naměřené výsledky jsou v kapitole analyzovány a vyhodnoceny.

Na závěr jsou uvedeny možnosti dalšího pokračování projektu a jsou shrnuty poznatky z celé diplomové práce. Tato diplomová práce vznikla v návaznosti na teoretickou část řešenou v rámci semestrálního projektu. V rámci semestrálního projektu byly vypracovány kapitoly č. 2 až 4.

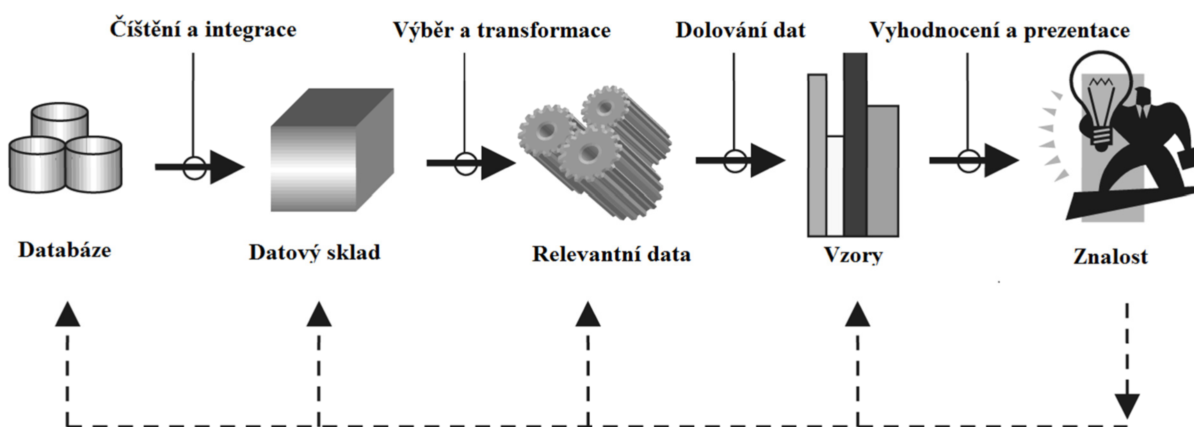
2 Získávání znalostí z databází

Získávání znalostí z databází je proces extrakce netriviálních, doposud neznámých a užitečných informací z velkého objemu dat. Informace je netriviální, když není na první pohled zřejmá a nelze ji získat jednoduchým dotazem do databáze. Získané znalosti se využívají k podpoře rozhodování. Příkladem může být zjištění informace o tom, jaké produkty si zákazníci kupují dohromady. Prodejce díky této informaci může vhodněji rozmístit zboží v prodejně nebo lépe cílit reklamu. Dalšími příklady využití procesu získávání znalostí z databází jsou finanční analýza, detekce podvodů, bioinformatika, web, zdravotnictví a mnoho dalších.

Proces získávání znalostí z databází se skládá z několika fází. Tyto fáze jsou popsány v následující podkapitole. Podkapitola 2.2 popisuje nejběžnější zdroje dat pro získávání informací a podkapitola 2.3 představuje typy dolovacích úloh, které se využívají k extrakci znalostí. Následující podkapitoly čerpají informace ze zdrojů [2, 13].

2.1 Proces získávání znalostí z databází

Získávání znalostí z databází je komplexní proces, který se skládá z řady iterativních částí. Během tohoto procesu se vstupní data zpracovávají, transformují a pomocí různých dolovacích algoritmů se z nich získávají užitečné informace. Pro dosažení přesnějších výsledků se některé fáze procesu opakují vícekrát. Obrázek 2.1 zobrazuje celý proces získávání znalostí z databází.



Obrázek 2.1: Proces získávání znalostí z databází. Převzato a upraveno z [2].

Představme si jednotlivé fáze procesu získávání znalostí z databází detailněji:

1. **Čištění dat –** Kvalita získaných znalostí mimo jiné závisí na kvalitě vstupních dat. Ve vstupních datech se mohou vyskytovat chybějící, zašuměné, popřípadě nekonzistentní hodnoty. Pro zkvalitnění získaných informací je nutné tyto neduhy odstranit nebo napravit. Během fáze čištění dochází k napravení těchto nekvalitních dat.

2. **Integrace dat** – V této fázi dochází k integraci dat z různých datových zdrojů. Právě integrace dat je hlavním zdrojem nekonzistence, a proto se často integrace a čištění dat provádí v jednom kroku. Typy zdrojů dat jsou popsány v kapitole 2.2.
3. **Výběr dat** – V tomto kroku vybereme data, která jsou relevantní pro dolovací úlohu. Data bývají často uložena v relačních databázích, proto výběr dat představuje selekci relevantních sloupců z tabulek. V případě datových skladů můžeme vybrat vhodné dimenze.
4. **Transformace dat** – Každý typ dolovací úlohy má určité požadavky na formát vstupních dat. Z tohoto důvodu se provádí transformace dat do takové podoby, která nejvíce vyhovuje zvolené dolovací úloze. Transformace může zahrnovat operace vyhlazení, generalizace, agregace, sumarizace, normalizace a konstrukce atributů.
5. **Dolování z dat** – Tato fáze je jádrem procesu získávání znalostí z databází. Aplikací vhodné metody se extrahují z dat vzory a modely, které reprezentují výsledné znalosti. Dolovací metody můžeme rozdělit do několika skupin. Tyto techniky jsou popsány v kapitole 2.3.
6. **Vyhodnocení vzorů** – Cílem tohoto kroku je určit skutečně zajímavé vzory pomocí měr užitečnosti. Mezi tyto míry například patří spolehlivost a podpora.
7. **Prezentace znalostí** – Pomocí vizualizace a technik reprezentace znalostí se srozumitelnou formou uživateli prezentují výsledné informace. Pro prezentaci se nejčastěji využívají různé grafy a tabulky.

Kroky čištění, integrace, výběr a transformace dat jsou souhrnně pojmenovány jako proces předzpracování dat. V případě datových skladů se také můžeme setkat s pojmenováním ETL - extrakce, transformace a přenos (loading). V tomto důležitém procesu se tedy data připravují pro analýzu dat.

2.2 Druhy dat pro dolování

V této kapitole si představíme nejtypičtější datová úložiště, která se využívají pro dolování z dat. Za účelem dosažení přesnějších výsledků se často kombinují data z více úložišť.

- **Relační databáze** – Datová úložiště, kde jsou data uchovávaná v tabulkách. Řádky tabulek reprezentují jednotlivé záznamy a sloupce obsahují informace o atributech těchto záznamů. Data v databázi lze zpřístupnit pomocí dotazů zapsaných v dotazovacím jazyce SQL.
- **Transakční databáze** – Toto úložiště se skládá ze souboru, ve kterém každý záznam reprezentuje jednu transakci. Transakce zahrnuje unikátní identifikátor a seznam položek, které transakci tvoří. Příkladem takové transakce může být nákup zboží zákazníkem.

- **Datový sklad** – Komplexní integrované datové úložiště, kde jsou data uložena ve struktuře, která umožňuje efektivní analýzu a dotazování. Práce s tímto úložištěm pak probíhá pomocí OLAP operací.
- **Prostorové a časoprostorové databáze** – Tyto databáze uchovávající informace vztahující se k nějakému prostoru. Typickým příkladem je geografická databáze obsahující souřadnice objektů na mapě.
- **Databáze sekvencí** – Úložiště posloupností uspořádaných událostí, kde čas může, ale nemusí, být explicitně zaznamenám. Typickým příkladem jsou sekvence nákupů zboží. Více informací o dolování nad těmito daty lze nalézt v kapitole 3.
- **Multimediální databáze** – Ukládají textové dokumenty, obrázky, audio a video. Dolováním nad textovými dokumenty se dají například zjistit autoři dokumentů, klasifikovat soubory do skupin atd.
- **Web** – Obrovská heterogenní databáze. Typickými dolovacími úlohami nad tímto úložištěm je dolování z užití webu, automatické shlukování, klasifikace webových stránek a analýza komunit v sociálních sítích.

2.3 Typy dolovacích úloh

Jádrem procesu získávání znalostí z databází je fáze dolování z dat. Tato kapitola popisuje hlavní typy dolovacích úloh, které se v této fázi mohou využít. Následující text čerpá ze zdrojů [2, 13]. V těchto zdrojích lze nalézt detailní informace o zmíněných metodách.

Dolovací úlohy lze obecně rozdělit do dvou základních skupin:

- **Deskriptivní** – Úlohy, které charakterizují obecné vlastnosti dat v databázi. Představiteli této skupiny jsou shluková analýza, dolování sekvenčních vzorů a analýza asociačních pravidel.
- **Prediktivní** – Tyto dolovací úlohy pomocí analýzy současných dat provádí předpověď budoucích hodnot. Typickými představiteli jsou klasifikace a predikce.

2.3.1 Popis konceptu/třídy

Cílem této dolovací úlohy je asociovat data s určitým konceptem nebo třídou. Pro uživatele pak může být užitečné jednotlivé třídy a koncepty popsat nějakým souhrnným, stručným a přitom přesným způsobem. Popis tříd a konceptů lze získat pomocí následujících metod:

- **Charakterizace dat** – Cílem je sumarizovat obecné rysy cílové třídy. Tyto informace lze získat pomocí jednoduchého SQL dotazu do databáze. V případě datových skladů lze sumarizace získat pomocí operace *roll-up* nad datovou kostkou.
- **Diskriminace dat** – Cílem je srovnání obecných rysů zkoumané třídy s obecnými rysy jiné třídy nebo množiny tříd.

2.3.2 Frekventované vzory a asociační pravidla

Pojmem frekventovaný vzor označujeme vzor, který se v datech vyskytuje relativně často. Mezi frekventované vzory patří podsekvence, množiny položek, podgrafy a podstromy. Z těchto vzorů pak generujeme asociační pravidla. Příkladem asociačního pravidla je:

$$koupí\ (kameru) \rightarrow koupí\ (tiskárnu)\ \ [podpora = 5\%, spolehlivost = 50\%].$$

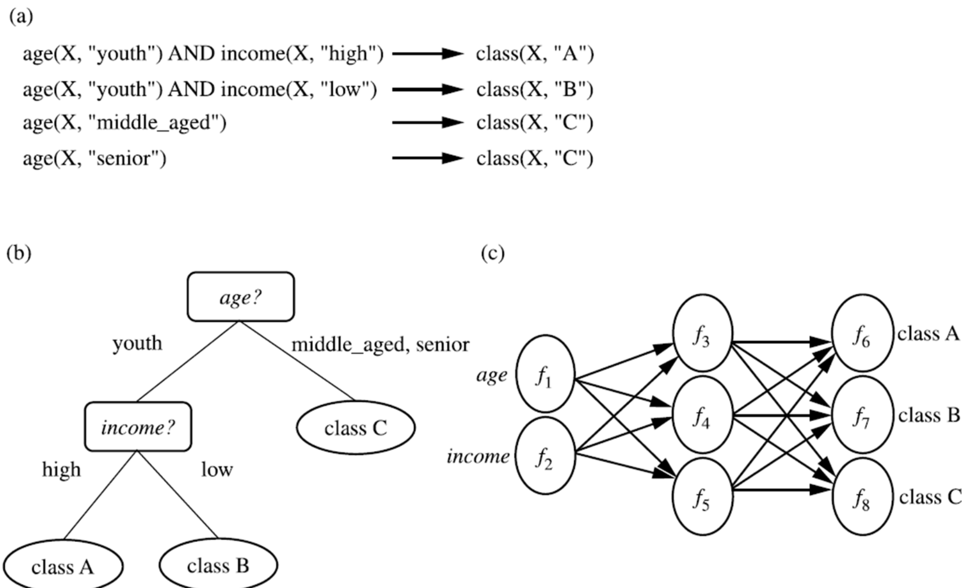
Toto pravidlo vyjadřuje, že zákazník, který si koupí kameru, si pravděpodobně koupí i tiskárnu. Důležitými měrami u asociačních pravidel jsou podpora a spolehlivost. Podpora v našem příkladu určuje, v kolika procentech všech nákupů si zákazník koupil současně kameru a tiskárnu. Spolehlivost pravidla říká, v kolika procentech zkoumaných případů byla splněna pravá část pravidla za předpokladu, že byla splněna i levá část. Tedy spolehlivost 50% značí, že v 50% nákupů, ve kterých si zákazníci koupili kameru, si také koupili i tiskárnu.

2.3.3 Klasifikace a predikce

Klasifikace je proces hledání modelu, který rozděluje data na základě jejich vlastností do konečné množiny tříd nebo konceptů. Klasifikace se používá k předpovědi hodnot kategorického charakteru. Predikce je obdobný proces jako klasifikace, jen s tím rozdílem, že se používá pro dedukci hodnot spojitého charakteru. Celý proces klasifikace a predikce se skládá z následujících kroků:

- **Trénování** – V prvním kroku se vytvoří klasifikační model. Tento model je vytvořen na základě trénovací množiny dat. U dat z této množiny známe třídu, do které patří.
- **Testování** – V tomto kroku se otestuje přesnost klasifikačního modelu na datech z testovací množiny dat. U dat z této množiny také známé třídu, do které patří. Podle výsledku testování klasifikátoru se rozhodne, jestli je model dostatečně přesný a může být použit pro předpověď budoucích hodnot.
- **Aplikace** – V této poslední fázi se získaný model využívá pro klasifikaci objektů, jejichž cílové třídy neznáme.

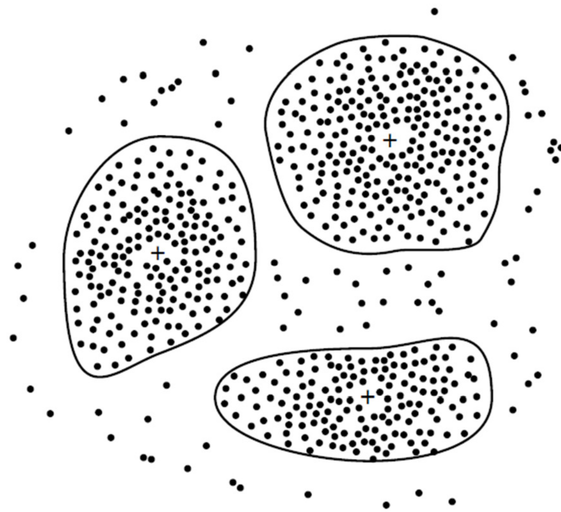
Výsledný klasifikační model může mít různou podobu. Mezi nejčastější patří klasifikační (*IF - THEN*) pravidla, rozhodovací strom nebo neuronové sítě. Různé formy klasifikačního modelu zobrazuje obrázek 2.2. Typickým příkladem využití klasifikace je model, který rozděluje zákazníky banky do dvou skupin. První skupina obsahuje zákazníky, u kterých se předpokládá, že budou schopni splácet bankovní úvěr. Druhá skupina naopak obsahuje zákazníky, u kterých je poskytnutí bankovního úvěru rizikem. Příkladem predikce může být určení výše platu nového zaměstnance na základě znalostí jeho dalších vlastností.



Obrázek 2.2: Formy reprezentace klasifikačního modelu: a) klasifikační (*IF - THEN*) pravidla, b) rozhodovací strom, c) neuronová síť. Převzato z [2].

2.3.4 Shluková analýza

Shluková analýza stejně jako klasifikace a predikce rozděluje data do různých tříd. Rozdílem však je, že shluková analýza rozděluje objekty do předem neznámých tříd (shluků). Tyto shluky jsou vytvářeny až v průběhu dolování. Cílem metody je vytvořit shluky tak, aby si objekty uvnitř stejného shluku byly co nejvíc podobné, a naopak objekty z různých shluků, aby byly navzájem co nejvíc odlišné. Pro určení podobnosti se nejčastěji využívají vzdálenostní funkce. Obrázek 2.3 ukazuje 2D graf výsledků shlukové analýzy pro polohu bydlišť zákazníků ve městě.



Obrázek 2.3: Příklad shluků poloh bydlišť zákazníků. Převzato z [2].

2.3.5 Analýza odlehlých objektů

Databáze může obsahovat datové objekty, které se neshodují s obecným chováním nebo modelem dat. Tyto data se nazývají odlehlé objekty. Většina dolovací úloh odlehlé objekty zahazuje jako datový šum nebo výjimky. Některé úlohy, jako je například detekce neobvyklého chování a podvodů, naopak považují odlehlé hodnoty za zajímavé a využívají analýzu odlehlých hodnot ve fázi dolování. Odlehlé hodnoty se získávají zpravidla použitím upravených dolovacích metod. Příkladem může být shlukování, které tyto objekty umí identifikovat. Další možností je použití metod statistického zpracování dat.

2.3.6 Evoluční analýza

Evoluční analýza popisuje a modeluje pravidelnosti nebo trendy objektů, jejichž chování se mění v čase. Typickými úlohami tohoto typu jsou analýza trendů, podobnostní analýza v časových řadách, dolování sekvenčních a periodických vzorů. Proces dolování sekvenčních vzorů je detailně popsán v kapitole 3.

3 Dolování sekvenčních vzorů

Dolování sekvenčních vzorů je proces vyhledávání frekventovaných vzorů z databáze sekvencí. Sekvence je složena z množiny uspořádaných událostí. Příkladem databáze sekvencí jsou posloupnosti nákupů zákazníků. Výsledným sekvenčním vzorem může být například zjištění, že zákazník, který si koupil počítač, si během dalšího měsíce koupí tiskárnu a digitální kameru. Získané znalosti se mohou využít například pro dokonalejší marketing.

Definice problému dolování sekvenčních vzoru je popsaná v následující podkapitole. Kapitola 3.2 představuje aplikace sekvenčních vzorů a v kapitole 3.3 jsou popsaný algoritmy pro dolování sekvenčních vzorů.

3.1 Definice problému

Problém dolování sekvenčních vzorů byl představen Agrawalem a Srikantem v roce 1995 [1] na základě studia sekvencí nákupů zákazníků. Problém je definován následovně: *Je daná množina sekvencí, kde každá sekvence obsahuje seznam událostí a každá událost je složena z množiny položek. Dolování sekvenčních vzorů je hledání všech podsekvencí, jejichž frekvence výskytu v množině všech sekvencí je větší nebo rovna uživatelem zadané minimální podpoře.*

Definice 3.1 Nechť $I = \{i_1, i_2, \dots, i_n\}$ je množinou všech položek [2]. *Sekvence* $s = < e_1 e_2 \dots e_m >$ je uspořádaný seznam *událostí*, kde událost e_1 předchází událost e_2, událost e_2 předchází e_3 atd. Událost e_i bývá také označována jako *prvek* popřípadě *element* sekvence s. Každá událost $e = (x_1 x_2 \dots x_q)$ je neprázdnou množinou položek, pro kterou platí $e \subseteq I$. Jedna položka se v rámci události vyskytuje nejvýše jednou, ale v rámci celé sekvence i vícekrát. Pokud je událost tvořena pouze jednou položkou, pak se při zápisu události vynechávají kulaté závorky. Počet instancí položek v sekvenci udává délku sekvence. Sekvence s délkou l se značí jako *l-sekvence*. Sekvence $\alpha =< a_1 a_2 \dots a_n >$ se nazývá *podsekvence* jiné sekvence $\beta =< b_1 b_2 \dots b_m >$ a naopak β je *nadsekvence* α, značeno $\alpha \sqsubseteq \beta$, pokud existují celá čísla $1 \leq j_1 < j_2 < \dots < j_n \leq m$ taková, že $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$.

Databáze sekvencí je množina dvojic $< sid, s >$, kde sid je identifikátor sekvence a s je sekvence. Dvojice $< sid, s >$ obsahuje sekvenci α v případě, že α je podsekvencí sekvence s. *Podpora* (*support*) sekvence α v databázi sekvencí S je dána počtem databázových dvojic, které obsahují sekvenci α, formálně zapsáno:

$$support_S(\alpha) = |\{< sid, s > |(< sid, s > \in S) \wedge (\alpha \sqsubseteq s)\}| \quad (3.1)$$

Mějme uživatelem zadané celé číslo značící *minimální podporu* (*min_support*), pak *sekvenční vzor* je sekvence α v databázi sekvencí S, pro kterou platí: $support_S(\alpha) \geq min_support$. Pro sekvenční vzor se také používá pojem *frekventovaná sekvence*. Je-li sekvence α frekventovaná a neexistuje-li v databázi její vlastní nadsekvence β se stejnou podporou, pak se sekvence α nazývá *uzavřený sekvenční vzor* [3].

Tabulka 3.1: Příklad databáze sekvencí. Převzato a upraveno z [2].

sid	sekvence
s_1	$< a(abc)(ac)d(cf) >$
s_2	$< (ad)c(bc)(ae) >$
s_3	$< (ef)(ab)(df)cb >$
s_4	$< eg(af)cbc >$

Příklad 3.1 Vysvětleme si předcházející pojmy na příkladu. Předpokládejme databázi sekvencí S, danou tabulkou 3.1 a uživatelem zadanou minimální podporu *min_support* = 2. Množina všech položek v databázi S je $\{a, b, c, d, e, f, g\}$. Sekvence $s_1 = < a(abc)(ac)d(cf) >$ má pět událostí, konkrétně (a), (abc), (ac), (d), (cf). Tyto události nastaly v uvedeném pořadí, tedy událost (a) předcházela událost (abc), ta předcházela událost (ac) a tak dále. V sekvenci s_1 je devět instancí položek, proto se také nazývá 9-sekvence. V sekvenci s_1 je položka a třikrát, přidává tedy k celkové délce sekvence hodnotu tři. Nicméně k celkové podpoře sekvence $< a >$ přidává sekvence s_1 pouze hodnotu jedna. Sekvence $< a(bc)df >$ je podsekvence sekvence s_1, protože všechny její události jsou podmnožinami událostí sekvence s_1 a pořadí těchto událostí je zachováno. Sekvence s_1 a s_2 obsahují podsekvenci $s =< (ab)c >$, s má tedy podporu dva, splňuje uživatelem zadaný práh minimální podpory a proto je sekvenčním vzorem. Sekvence s je také uzavřený sekvenční vzor, protože v databázi S neexistuje její nadsekvence se stejnou podporou. Naopak sekvence $s_a = < (ab) >$ s podporou dva je sekvenční vzor, ale není uzavřený, protože v databázi existuje její nadsekvence s, která má stejnou podporou.

3.2 Aplikace sekvenčních vzorů

Metody dolování sekvenčních vzorů lze aplikovat na velkou škálu domén. Mezi tyto domény patří marketing, zdravotnictví, telekomunikace, web, bioinformatika, přírodní katastrofy, bezpečnost sítí a mnoho dalších. Tato kapitola popisuje několik příkladů aplikací sekvenčních vzorů v praxi. Následující text čerpá z dokumentu [17], kde autor M. Gupta sepsal přehled aplikací sekvenčních vzorů.

Klasifikace XML dokumentů

Článek [18] představuje metodu využití sekvenčních vzorů pro klasifikaci XML dokumentů. Dokumenty jsou do různých skupin klasifikovány na základě frekventovaných podstruktur. Převod struktu-

ry XML dokumentu do sekvence je proveden pomocí následujícího postupu. Každému uzlu v XML stromu je přiřazen identifikátor. Každý identifikátor je asociován s hloubkou ve stromě. Nakonec se nad výsledným stromem dokumentu provede průchod do šířky, čímž se obdrží výsledná sekvence. Dolováním nad těmito sekvencemi se získají frekventované strukturální podsekvence, které reprezentují danou skupinu XML dokumentů. Sekvenční vzor pro skupinu dokumentů popisující filmy by vypadal například takto:

$$< (film_0)(název_1)(filmografie_1)(režisér_2)(herci_2)(jméno_3) >.$$

Nové XML dokumenty jsou do skupin klasifikovány na základě nejdelší společné podsekvence.

Systém detekce narušení (IDS)

V [19] L. Wuu a S. Chen publikovali implementaci modulu pro software Snort. Snort je open source systém pro detekci narušení v síti. Jejich modul využívá dolovacích metod k extrakci sekvenčních vzorů ze záznamů získaných během průniků do systémů. Sekvence jsou reprezentovány posloupností hlaviček a obsahy datových paketů. Získané sekvenční vzory jsou převedeny do Snort pravidel, které se využívají pro online analýzu paketů. Výsledkem sekvenčním vzorem může být následující sekvence detekčních pravidel:

seqstart tcp SPort1 any → any 139 (flags: S+; timelen:10; sid:1;),
seqcheck tcp SPort1 any → any 139 (flags: A+; timelen:10; sid:2;),
seqcheck tcp SPort1 any → any 139 (flags: UAP+; timelen:10; sid:3;),
seqalert tcp SPort1 any → any 139 (flags: UAP+; timelen:10; sid:4;).

V případě, že sekvence příchozích paketů odpovídá této posloupnosti pravidel, detekční systém vyvolá výstrahu.

Dolování znalostí z užití webu

Dolování z užití webu extrahuje vzory mezi průchody webem jednotlivých uživatelů. Dolování se v tomto případě provádí nad webovými logy spolu se zapojením různých heuristik. Mezi nejtypičtější heuristiky patří uživatelské profily a hierarchie zboží. Získané vzory se mohou využít pro zvýšení prodejnosti zboží díky doporučujících systémů, redesignu struktury webu pro vyšší přístupnost, případně pro zrychlení odezvy aplikace, díky dopředného načítání stránek.

Během posledních let bylo vytvořeno mnoho prací na toto téma [20, 21, 22, 23, 24]. Například ve článku [20] autoři A. Pitman a M. Zarkus vytvořili doporučující systém zboží s velmi vysokou přesností. Vysoká přesnost jejich systémů je postavena na dolování vícedimenzionálních frekventovaných sekvencí. Výsledný vzor jejich systému může mít následující podobu:

$$< \big(shlédl(mléko),\ shlédl(bio\ mléko),\ koupil(bio\ mléko)\big)\big(koupil(hnědý\ chléb)\big) >.$$

Aplikací tohoto vzoru se zákazníkovi, který navštíví webovou stránku s mlékem a bio mlékem a nakonec si bio mléko přidá do nákupního košíku, zobrazí reklama na hnědý chléb. Zákazník si totiž s velkou pravděpodobností hnědý chléb také zakoupí.

3.3 Algoritmy pro dolování sekvenčních vzorů

Během posledních let vzniklo mnoho algoritmů pro dolování sekvenčních vzorů. Tato kapitola popisuje nejvýznamnější vlastnosti, ve kterých se algoritmy odlišují. Následující podkapitoly pak detailně popisují algoritmy, které byly v rámci této práce implementovány a testovány. Na základě analýzy vlastností algoritmů byl pro implementaci vždy zvolen nejefektivnější zástupce z rodiny algoritmů, která přináší novou myšlenku v oblasti dolování sekvenčních vzorů.

3.3.1 Analýza existujících algoritmů

V rámci výzkumů algoritmů pro dolování sekvenčních vzorů bylo vymyšleno mnoho nových technik, které zrychlují proces dolování sekvenčních vzorů. Techniky se zejména odlišuje v následujících způsobech [12]:

1. Generování a uložení kandidátních sekvencí. Hlavním cílem je minimalizovat počet vygenerovaných kandidátních sekvencí a minimalizace vstup/výstupních operací.
2. Počítání podpory a způsob testování, zdali jsou kandidáti frekventovaní. Klíčovou strategií zde je eliminace počtu průchodů vstupní databázi za účelem výpočtu podpory kandidátů.

Apriori vlastnost

Většina algoritmů využívá pro urychlení dolování *Apriori* vlastnost, a to buď přímo, nebo nepřímo. Tato vlastnost, představena v rámci dolování asociačních pravidel v [11], zní pro oblast dolování sekvencí následovně: *Všechny podsekvence frekventované sekvence musí být také frekventované, respektive sekvence je nefrekventovaná pokud obsahuje nějakou nefrekventovanou podsekvenci*.

Prvním algoritmem pro dolování sekvenčních vzorů je algoritmus s názvem AprioriAll [1]. Tento algoritmus je postavený na využití Apriori vlastnosti a byl vytvořen autory, kteří jako první definovali problém sekvenčních vzorů. Stejní autoři později představili *zobecněné sekvenční vzory* zavedením časových omezení, klouzavého okna a hierarchie položek. Navíc vytvořili nový algoritmus GSP, který je značně výkonnější než AprioriAll a dokáže získávat právě zmíněné zobecněné sekvenční vzory.

S Apriori vlastností souvisí také princip *generování a testování (generate-and-test)* kandidátních sekvencí. Tato metoda v každém běhu dolovacího procesu spojuje frekventované sekvence získané v předchozím běhu, pro vytvoření nových, o jeden prvek delších, kandidátů. Pomocí průchodu vstupní databází se poté stanoví, zdali jsou kandidátní sekvence frekventované.

Algoritmy postavené pouze na těchto vlastnostech generují obrovské množství kandidátních sekvencí, spotřebovávají mnoho paměti a vyžadují vícenásobný průchod databází. Z těchto důvodů nemají dostačující výkon při dolování nad velkými datovými sadami.

Reprezentace databáze

Databáze sekvencí může být reprezentována dvěma formáty: *horizontálně* nebo *vertikálně*. Při využití horizontálního formátu se sekvence skládá z uspořádaného seznamu událostí. Každá událost obsahuje seznam položek. Tuto reprezentaci využívají algoritmy AprioriAll [1], GSP [4] a PrefixSpan [5]. Algoritmy počítají podporu kandidátů průchodem vstupní databází, což je výpočetně náročné.

Druhá skupina algoritmů využívá vertikální formát. Databáze je reprezentována množinou položek. Každá položka si uchovává informaci o tom, ve kterých událostech se vyskytuje a ve kterých ne. Hlavními představiteli této skupiny jsou algoritmy SPADE [8] a SPAM [7]. Výpočet podpory kandidátů zde probíhá pomocí rychlé operace spojení (logický průnik, bitový součin atd.).

Algoritmy využívající vertikální reprezentaci databáze eliminují vstup/výstupní operace, protože v procesu dolování neprocházejí původní databázi. Pro implementaci a testování byl zvolen algoritmus SPAM. Detailní popis tohoto algoritmu lze nalézt v kapitole 3.3.4.

Brzké prořezání vstupního prostoru

Většina algoritmů prořezává prohledávací prostor pomocí Apriori vlastnosti. Algoritmy patřící do této skupiny se však snaží využít i jiných, efektivnějších technik prořezávání. Hlavními představiteli jsou algoritmy z rodiny LAPIN (LAPIN [10], LAPIN-SPAM [9], LAPIN-WEB [14]). Tyto algoritmy staví na myšlence *indukce pozice* a snaží se díky ní co nejdříve v procesu dolování odhadnout, jestli kandidát může být frekventovaný, či ne. Velmi efektivní prořezávací metodu s názvem *BackScan* přináší algoritmus BIDE [3]. Díky této metodě je BIDE nejefektivnějším algoritmem pro dolování frekventovaných sekvencí. Z této kategorie byl pro implementaci a testování zvolen algoritmus LAPIN-SPAM a BIDE. Kapitola 3.3.5 detailně popisuje algoritmus LAPIN-SPAM, kapitola 3.3.6 podrobně popisuje algoritmus BIDE.

Princip vzrůstu vzoru

Algoritmy založené na principu vzrůstu vzoru řeší problém neefektivního generování a testování kandidátních sekvencí. Hlavní myšlenkou je vyhnout se generování kandidátů a zaměřit se na vyhledávání vzorů v menších částech databáze. Vstupní databáze je během procesu dolování postupně dělena do menších částí, nad kterými se vyhledávají lokálně frekventované položky. Frekventované položky se připojí k prefixu, čímž dochází k postupnému *vzrůstu vzoru* (*pattern growth*). Hlavními představiteli této skupiny jsou algoritmy FreeSpan [15] a PrefixSpan [5]. Druhý jmenovaný algoritmus byl zvolen pro implementaci a je popsán v kapitole 3.3.3.

Uzavřené sekvenční vzory

Dolování úplné množiny frekventovaných podsekvencí může generovat obrovské množství výsledných vzorů. Možnou alternativou je dolovat *uzavřené sekvenční vzory* (*closed sequential patterns*). Připomeňme si, že jde o frekventovanou sekvenci, ke které neexistuje v databázi nadsekvence se stejnou podporou. Dolování uzavřených sekvenčních vzorů může produkovat výrazně menší počet výsledných vzorů. Nutno podotknout, že úplnou množinu sekvenčních vzorů lze i s podporou jednoduše odvodit z množiny uzavřených sekvenčních vzorů. Díky této kompresi lze uzavřené vzory získat rychlejším způsobem než úplnou množinu vzorů.

Prvním algoritmem, který přišel s myšlenkou dolování uzavřených vzorů je CloSpan [16]. Tento algoritmus si během procesu dolování uchovává množinu všech dříve nalezených uzavřených sekvencí. Tuto množinu používá pro prořezávání prohledávacího prostoru a ke kontrole uzavřenosti nově nalezených vzorů. Tento přístup však může mít malou škálovatelnost v případech, kdy je zadána nízká hodnota minimální podpory. Důvodem je, že uchovaná množina uzavřených vzorů zabírá mnoho místa v paměti a je časově náročné procházet tuto množinu k určení uzavřenosti nově nalezeného vzoru. Algoritmus BIDE [3] přichází s metodou jak určit uzavřenost sekvencí bez uchování dříve nalezených uzavřených sekvencí. Tento efektivní algoritmus byl proto zvolen k implementaci a je popsán v kapitole 3.3.6.

3.3.2 GSP

Algoritmus GSP (Generalized Sequential Pattern) [4] byl vytvořen stejnými autory jako algoritmus AprioriAll [1] v roce 1996. GSP je podobně jako AprioriAll založený na metodě generování a testování kandidátních sekvencí. Pro eliminaci počtu kandidátů využívá Apriori vlastnost. V průběhu dolování prochází vstupní databázi sekvencí vícekrát, což je jedna z příčin nedostačujícího výkonu tohoto algoritmu.

Algoritmus funguje následujícím způsobem. Prvním průchodem databází se vyhledají všechny položky, jejichž podpora je větší nebo rovna hodnotě minimální podpory. Každá taková frekventovaná položka tvoří jednoprvkový sekvenční vzor a dohromady vytváří počáteční množinu sekvenčních vzorů (*seed set*) pro další běh. Každý další běh algoritmu začíná s počáteční množinou vzorů, která byla nalezena v předchozím běhu. Počáteční množina se využívá pro generování nových sekvencí, zvaných *kandidátní sekvence*. Každá nově vygenerovaná kandidátní sekvence má vždy o jednu položku více než sekvenční vzor, z kterého byla generována. Všechny kandidátní sekvence mají v jednom běhu stejnou délku, obecně jde tedy o k-sekvence, které společně tvoří množinu značenou C_k. Následujícím průchodem databází se vypočítá podpora všech kandidátních k-sekvencí. Kandidátní k-sekvence, jejichž podpora je větší nebo rovna zadané minimální podpoře, pak tvoří množinu frekventovaných k-sekvencí L_k. Ta se použije jako základ pro další běh, k + 1. Algoritmus skončí

v případě, že v běhu nebyly nalezeny žádné nové sekvenční vzory, nebo když nemohou být vygenerovány žádné kandidátní sekvence. Pseudokód GSP je zobrazen pomocí algoritmu 3.1.

Definice 3.2 Mějme sekvenci $s =< s_1 s_2 \dots s_n >$ a podsekvenci c, pak c se nazývá *sousedící* (*contiguous*) podsekvence sekvence s v případě, že platí nějaká z následujících podmínek [4]:

1. c je odvozena ze sekvence s odstraněním položky z události s_1 nebo s_n.
2. c je odvozena ze sekvence s odstraněním položky z události s_i a zároveň událost s_i má nejméně dvě položky.
3. c je sousedící podsekvence sekvence c', a c' je sousedící podsekvence sekvence s.

Příklad 3.2 Předpokládejme sekvenci $s =< (ab)(cd)ef >$. Sekvence $< b(cd)e >$, $< (ab)cef >$ a $< ce >$ jsou sousedící podsekvence sekvence s. Naopak sekvence $< (ab)(cd)f >$ a $< aef >$ nejsou sousedící podsekvence sekvence s.

Algoritmus 3.1: GSP

Vstup:
```
    S: databáze sekvencí;
    min_support: minimální podpora;
```
Výstup:
```
    úplná množina sekvenčních vzorů;
```
Metoda:
```
    L_1 = frekventované 1-sekvence;
    for (k = 1; L_k != null; k = k + 1) do
      C_{k+1} = generuj kandidátní sekvence z L_k;
      foreach sekvence s in S do
        inkrementuj podporu všech kandidátů v C_{k+1}, kteří jsou v s;
      end;
      L_{k+1} = kandidáti v C_{k+1} s podporou ≥ min_support;
    end;
    return ∪_k L_k
```

Generování kandidátních sekvencí

Algoritmus GSP využívá pro generování nových kandidátních sekvencí dvoufázovou metodu. V první fázi zvané *Join Phase* [4] dochází ke spojení dvou k-sekvencí. Dvě sekvence s_1 a s_2 s k položkami mohou být spojeny za předpokladu, že podsekvence získána odstraněním první položky z s_1 je stejná jako podsekvence získána odstraněním poslední položky z s_2. Výsledná kandidátní $(k+1)$-sekvence je tvořena sekvencí s_1 rozšířenou o poslední položku ze sekvence s_2.

V druhé fázi zvané *Prune Phase* [4] dochází k prořezání vygenerovaných kandidátních sekvencí díky využití Apriori vlastnosti. Odstraní se všechny kandidátní sekvence, které mají nějakou nefrekventovanou sousedící $(k-1)$-podsekvenci.

Příklad 3.3 Ukažme si generování kandidátních sekvencí na příkladu. Tabulka 3.2 ukazuje počáteční množinu frekventovaných 3-sekvencí po sloučení a po prořezání. Ve slučovací fázi se spojí sekvence $< (ab)c >$ se sekvencí $< b(cd) >$ a vytvoří novou kandidátní sekvenci $< (ab)(cd) >$, dále se první sekvence spojí se sekvenci $< bce >$ a dohromady vytvoří sekvenci $< (ab)ce >$. K žádnému dalšímu spojení již nedojde. Například sekvence $< (ab)d >$ se nespojí se žádnou sekvencí, protože zde není žádná sekvence ve tvaru $< b(d_) >$ nebo $< bd_ >$. V ořezávací fázi dojde k odstranění sekvence $< (ab)ce >$, protože její sousedící podsekvence $< ace >$ není frekventovaná.

Tabulka 3.2: Příklad generování kandidátních sekvencí. Převzato a upraveno z [4].

frekventované 3-sekvence	kandidátní 4-sekvence	
	po sloučení	po prořezání
$< (ab)c >$	$< (ab)(cd) >$	$< (ab)(cd) >$
$< (ab)d >$	$< (ab)ce >$	
$< a(cd) >$		
$< (ac)e >$		
$< b(cd) >$		
$< bce >$		

Příklad 3.4 (GSP) Uvažujme databázi sekvencí S, danou tabulkou 3.1 a uživatelem zadanou minimální podporu *min_support* = 2. Algoritmus GSP nejprve projde databázi S a vypočte podporu všech položek, které se v databázi vyskytují. Z těch pak vytvoří 1-sekvence (ve tvaru: *„položka: podpora“*): $< a >$: 4, $< b >$: 4, $< c >$: 3, $< d >$: 3, $< e >$: 3, $< f >$: 3, $< g >$: 1.

Sekvence $< g >$ má podporu menší než je hodnota minimální podpory a proto je z výsledné množiny odstraněna. Odstraněním nefrekventovaných položek se získá první počáteční množina $L_1 = \{< a >, < b >, < c >, < d >, < e >, < f >\}$. Členy v množině L_1 reprezentují jednoprvkové sekvenční vzory. S využitím šesti prvkové počáteční množiny L_1 je vygenerováno $6 \times 6 + \frac{6 \times 5}{2} = 51$ kandidátních sekvencí, $C_2 = \{< aa >, < ab >, \ldots, < af >, < ba >, < bb >, \ldots, < ff >, < (ab) >, < (ac) >, \ldots, < (ef) >\}$.

Dalším průchodem databází se vypočítá podpora všech kandidátních sekvencí z množiny C_2 a odstraní se nefrekventované kandidátní sekvence, výsledek pak tvoří množinu dvouprvkových sekvenčních vzorů L_2, která se použije pro další běh.

Zobecněné sekvenční vzory

Algoritmus GSP umožňuje dolovat zobecněné sekvenční vzory [4]. Uživatel tak může pomocí vstupních parametrů zadat nová omezení:

a. Časové omezení, pomocí parametrů *max-gap* a *min-gap* lze určit maximální resp. minimální délku mezi dvěma událostmi v sekvenci. Lze tak například omezit případy, kdy nás zajímají nákupy zákazníka, které jsou prováděny vždy maximálně po měsíci.

b. Pomocí parametru *window-size*, lze specifikovat délku intervalu, v němž mohou položky z jedné události patřit do více událostí. Lze tak například řešit situace, kdy zákazník nakoupí v jeden den zboží ráno a večer. Klasicky jde o dvě události v rámci jedné sekvence. Z obchodního hlediska může být zajímavé tyto nákupy sloučit v jednu událost.
c. Uživatel může zadat na vstupu hierarchii položek. Může pak dolovat sekvenční vzory napříč hierarchiemi. Výsledem pak může být sekvenční vzor říkající, že zákazník si koupí obecně počítač a týden na to si koupí myš subkategorie XX od konkrétní firmy Y.

Dolovací algoritmy založené na principu generování a testování jako je GSP, typicky vyžadují vícenásobný průchod databází. Například GSP musí pro nalezení vzoru $< (abc)(abc)(abc)(abc) >$ projít vstupní databázi dvanáctkrát. Také generují obrovské množství kandidátních sekvencí, z nichž se některé ani nevyskytují v databázi. Například je-li v databázi 1 000 frekventovaných 1-sekvencí, pak algoritmus GSP vygeneruje $1\,000 \times 1\,000 + \frac{1\,000 \times 999}{2} = 1\,499\,500$ kandidátních sekvencí. Z těchto důvodů algoritmus GSP nestačí svým výkonem novějším, pokročilejším algoritmům, zejména při dolování dlouhých sekvenčních vzorů.

3.3.3 PrefixSpan

PrefixSpan (Prefix-Projected Sequential Pattern Growth) [5] je efektivní algoritmus pro dolování frekventovaných sekvencí metodou postupného *vzrůstu vzoru* (*pattern growth*). Hlavní předností algoritmu je, že nevyužívá princip generování a testování kandidátních sekvencí. Namísto toho PrefixSpan pomocí prefixů rekurzivně projektuje databázi sekvencí do více projektovaných databází. Projektované databáze jsou zpravidla mnohem menší než původní databáze, čímž dochází k redukci prohledávácího prostoru a k urychlení procesu dolování. V projektované databázi následně vyhledává lokálně frekventované položky, které využívá pro sestavení nových sekvenčních vzorů. Pomocí lokálně frekventovaných položek projektuje databázi a cyklus se opakuje. Touto technikou dochází k postupnému vzrůstu nalezených vzorů.

Definice 3.3 Bez újmy na obecnosti algoritmus předpokládá, že jsou položky v rámci události abecedně seřazeny [5]. Sekvence $\beta =< e'_1 e'_2 \dots e'_m >$ se nazývá *prefixem* sekvence $\alpha =< e_1 e_2 \dots e_n >$ $(m \leq n)$, tehdy a jen tehdy když platí: 1) $e'_i = e_i$ pro $(i \leq m-1)$, 2) $e'_m \subseteq e_m$ a 3) všechny položky v množině $(e_m - e'_m)$ abecedně následují za těmi v množině e'_m. Sekvence $\gamma =< e''_m e_{m+1} \dots e_n >$ se pak nazývá *sufixem* sekvence α vzhledem k prefixu β, značeno jako $\gamma = \alpha \,/\, \beta$. V případě, že výsledná množina položek $e''_m = (e_m - e'_m)$ není prázdná, označujeme sufix sekvenci také jako <(_ položky v e''_m)$e_{m+1} \dots e_n$ >. Dále uvažujme, že α je sekvenční vzor v databázi sekvencí S,

pak α-*projektovaná databáze* (α-*projected database*) je kolekce sufixů v S vzhledem k prefixu α, značeno $S|_\alpha$.

Příklad 3.5 Ilustrujme si předcházející pojmy na příkladu. Mějme sekvenci $s =< a(abc)(ac)dc >$, sekvence $< a >$, $< aa >$, $< a(ab) >$, $< a(abc) >$ a $< a(abc)(a) >$ jsou prefixy sekvence s. Naopak $< ab >$ a $< a(bc) >$ nejsou prefixy sekvence s. Sekvence $< (abc)(ac)dc >$ je sufixem s vzhledem k prefixu $< a >$, sekvence $< (_bc)(ac)dc >$ je sufixem vzhledem k prefixu $< aa >$ a sekvence $< (_c)(ac)dc >$ je sufixem sekvence s vzhledem k prefixu $< a(ab) >$. Uvažujme databázi sekvenci S danou tabulkou 3.1, pak α-projektovaná databáze $S|_\alpha$ se skládá z následujících sekvencí: $< (abc)(ac)d(cf) >$, $< (_d)c(bc)(ae) >$, $< (_b)(df)cb >$, $< (_f)cbc >$.

S využitím pojmů prefixu a sufixu lze problém dolování sekvenčních vzorů rozložit do následujících podproblémů:

1. Nechť $\{< x_1 >, < x_2 >, \dots, < x_n >\}$ je kompletní množinou jednoprvkových sekvenčních vzorů v databázi sekvencí S. Úplnou množinu sekvenčních vzorů v databázi S lze rozdělit do n disjunktních podmnožin tak, že *i-tá* podmnožina je množinou všech sekvenčních vzorů s prefixem $< x_i >$, pro $(1 \leq i \leq n)$.
2. Nechť α je sekvenční vzor o délce l a $\{\beta_1, \beta_2, \dots, \beta_m\}$ je kompletní množina všech sekvenčních vzorů s prefixem α o délce $(l + 1)$. Úplnou množinu sekvenčních vzorů s prefixem α, lze rozdělit do m disjunktních podmnožin tak, že *j-tá* podmnožina je množinou všech sekvenčních vzorů s prefixem β_j, pro $(1 \leq j \leq \text{m})$.

Na základě těchto tvrzení můžeme problém dolování sekvenčních vzorů rozdělit rekurzivně tak, že v případě potřeby lze každou podmnožinu sekvenčních vzorů dále rozdělit na další podmnožiny. A proto během dolování podmnožiny sekvenčních vzorů algoritmus PrefixSpan vytváří projektované databáze a v každé rekurzivně doluje odděleně. Důkazy o správnosti předcházejících tvrzení lze nalézt v [5].

Příklad 3.6 (PrefixSpan) Pro lepší názornost si funkčnost algoritmu PrefixSpan ukážeme na příkladu [5]. Předpokládejme databázi sekvencí S, danou tabulkou 3.1 a uživatelem zadanou minimální podporu *min_support* = 2. Dolování sekvenčních vzorů v databázi S pak probíhá následovně:

1. Pomocí prvního průchodu databází se vyhledají všechny sekvenční vzory o délce jedna. V příkladu se jedná o šest sekvencí $< a >$: 4, $< b >$: 4, $< c >$: 3, $< d >$: 3, $< e >$: 3, $< f >$: 3.

2. Kompletní množinu sekvenčních vzorů lze na základě prefixů rozdělit do šesti podmnožin: 1) sekvenčních vzory s prefixem $< \alpha >$, 2) sekvenční vzory s prefixem $< \beta >$, …, a 6) sekvenční vzory s prefixem $< f >$.
3. Podmnožiny sekvenčních vzorů lze získat pomocí konstrukce příslušných projektovaných databází. Nad každou touto databází se rekurzivně doluje následujícím způsobem:
    a. V případě podmnožiny sekvenčních vzorů začínajících prefixem $< \alpha >$ se vyhledají pouze sekvence, které obsahují $< \alpha >$. Z každé této sekvence se do $< \alpha >$-projektované databáze přidají pouze podsekvence předcházeny prvním výskytem $< \alpha >$. Například pro sekvenci $< (ef)(ab)(df)cb >$ to je podsekvence $< (_b)(df)cb >$. Sekvence v databázi S se tedy dle prefixu $< \alpha >$ projektují a vytvoří $< \alpha >$-projektovanou databázi. Databáze se skládá ze čtyř sufix sekvencí: $< (abc)(ac)d(cf) >$, $< (_d)c(bc)(ae) >$, $< (_b)(df)(cb) >$ a $< (_f)cbc >$.

        Průchodem $< \alpha >$-projektované databáze se vyhledají lokálně frekventované položky. Obdržíme tyto položky $a : 2$, $b : 4$, $_b : 2$, $c : 4$, $d : 2$ a $f : 2$. Spojením těchto položek s prefixem se sestaví všechny dvouprvkové sekvenční vzory s prefixem $< a >$, konkrétně to jsou $< aa >: 2$, $< ab >: 4$, $< (ab) >: 2$, $< ac >: 4$, $< ad >: 2$ a $< af >: 2$.

        Všechny sekvenční vzory s prefixem $< \alpha >$ lze rekurzivně rozdělit do šesti podmnožin: 1) sekvenční vzory s prefixem $< aa >$, 2) sekvenční vzory s prefixem $< ab >$, …, a 6) sekvenční vzory s prefixem $< af >$. Tyto podmnožiny lze prohledat pomocí vytvoření odpovídajících projektovaných databází. Nad každou touto databází se opět doluje rekurzivně:

        i. $< \alpha\alpha >$-projektovaná databáze je složena ze dvou neprázdných sufixů: $< (_bc)(ac)d(cf) >$ a $< (_e) >$. Průchodem $< \alpha\alpha >$-projektované databáze se nenajdou žádné lokálně frekventované položky a hledání je tím v této databázi ukončeno.
        ii. Vytvoří se $< ab >$-, $< (ab) >$-, $< ac >$-, $< ad >$- a $< af >$-projektované databáze a rekurzivně se nad nimi doluje stejným způsobem.
    b. Naleznou se sekvenční vzory s prefixem $< b >$, $< c >$, $< d >$, $< e >$ a $< f >$ pomocí konstrukce $< b >$-, $< c >$-, $< d >$-, $< e >$- a $< f >$-projektovaných databází a dolováním nad nimi.
4. Kompletní množina sekvenčních vzorů se získá sloučením všech sekvenčních vzorů nalezených ve výše uvedeném rekurzivním procesu. Pseudokód PrefixSpanu je zobrazen pomocí algoritmu 3.2.

Z uvedeného rekurzivního postupu lze vidět, že algoritmus PrefixSpan během procesu dolování negeneruje žádné kandidátní sekvence. Na druhou stranu algoritmus vytváří velké množství projektovaných databází, což je hlavním výkonnostním problémem algoritmu. Autoři algoritmu proto navrhli optimalizační techniku s názvem *pseudo-projekce* popsanou v [5]. Namísto fyzického projektování sekvencí je projekce reprezentovaná *ukazatelem* na sekvenci v původní databázi a *offsetem*, který v sekvenci určuje pozici začátku projekce. Využití pseudo-projekce je však efektivní pouze v případech, kdy lze celou vstupní databázi sekvencí uchovat v paměti počítače. V opačném případě, kdy se data nevlezou do paměti, dochází s touto technikou k degradaci výkonu z důvodu náhodných přístupů na pevný disk. V dnešní době zpravidla není problém s nedostatkem paměti, a proto je tato optimalizační technika využívána.

Algoritmus 3.2: PrefixSpan

Vstup:
```
     S: databáze sekvencí;
     min_support: uživatelem zadaná minimální podpora;
```
Výstup:
```
     úplná množina sekvenčních vzorů;
```
Metoda:
```
     call PrefixSpan(<>, S);

     procedure PrefixSpan(α, S|α):
       α: prefix,
       S|α: α-projektovaná databáze, pro α != <>, jinak databáze
            sekvencí S;

       Průchodem S|α nalezni všechny frekventované položky b, které:
         -  lze připojit za poslední položku v poslední události sekvence
            α a vytvořit tak nový sekvenční vzor, nebo
         -  lze připojit za poslední událost sekvence α a vytvořit tak
            nový sekvenční vzor;
       Každou položku b připoj k prefixu α a sekvenční vzor α′ ulož;
       Pro každý vzor α′, vytvoř α′-projektovanou databázi S|α′;
       call PrefixSpan(α′, S|α′);
     end;
```

3.3.4 SPAM

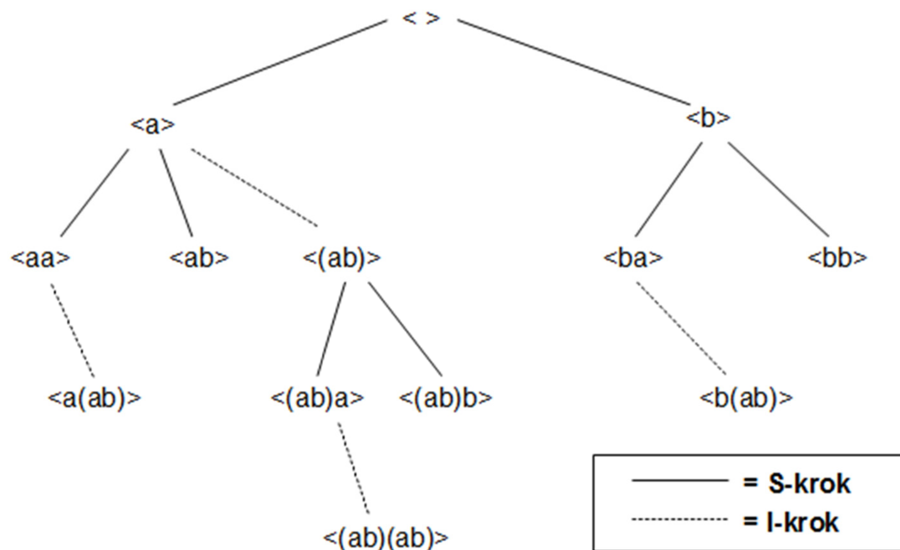
Algoritmus SPAM (Sequential Pattern Mining using A Bitmap Representation) [7] integruje principy z různých dříve navržených algoritmů. Používá techniku generování a testování kandidátů stejně jako GSP. Pro větší výkon však využívá vertikální formát databáze reprezentovaný pomocí bitmap. Bitmapová reprezentace umožňuje využít rychlé bitové operace AND pro generování kandidátních sekvencí. Navíc lze prostřednictvím této reprezentace efektivně počítat podporu vygenerovaných kandidátů. Vstupní prostor prochází pomocí lexikografického sekvenčního stromu a jako první algoritmus pro dolování sekvenčních vzorů jej prohledává metodou do hloubky.

Lexikografický sekvenční strom

Algoritmus SPAM je postaven na *lexikografickém sekvenčním stromě* [7]. Tento strom je využit pro generování a prořezávání kandidátních sekvencí. Podoba lexikografického sekvenčního stromu je podrobně popsána v následujícím textu.

SPAM stejně jako algoritmus PrefixSpan předpokládá lexikografické uspořádání na množině položek I. Je-li výskyt položky j v daném uspořádání předcházen výskytem položky i budeme vztah označovat $i \leq_I j$. Uspořádání lze dále rozšířit na sekvence. V případě, že sekvence s_a je podsekvencí s_b, pak se sekvence s_a v uspořádání vyskytuje před sekvenci s_b, značeno $s_a \leq s_b$. Lexikografický sekvenční strom T je definován následovně. Kořen stromu je označen znakem <>. Rekurzivně, je-li n uzel stromu, pak jeho synové jsou všechny uzly n' takové, pro které platí: $n \leq n'$ a $\forall m \in T: n' \leq m \Rightarrow n \leq m$.

Během průchodu stromem jsou z každého uzlu generovány dva typy poduzlů: *sekvenčně-rozšířené sekvence* (*sequence-extended sequences*) a *množinově-rozšířené sekvence (itemset-extended sequences*). Sekvenčně-rozšířená sekvence je vygenerována přidáním nové jednoprvkové události na konec rodičovské sekvence. Tento krok se označuje jako *S-krok* (*S-step*). Množinově-rozšířena sekvence je vygenerovaná přidáním nové položky do poslední události rodičovské sekvence, značeno jako *I-krok* (*I-step*). Obrázek 3.1 zobrazuje lexikografický sekvenční strom pro položky $I = \{a, b\}$ s omezením, že každá sekvence obsahuje maximálně dvě události.



Obrázek 3.1: Lexikografický sekvenční strom položek {a, b} s maximálně dvěma událostmi v sekvenci.

Prořezávací technika

V průběhu dolování frekventovaných sekvencí SPAM prochází strom do hloubky (*deep-first search*), na rozdíl od algoritmu GSP, který vstupní prostor prochází do šířky (*breadth-first search*). Průchod

do hloubky je výhodnější, protože spotřebovává méně paměti a umožňuje aplikovat efektivnější prořezávání techniky.

V každém uzlu stromu se nejprve vypočte podpora všech vygenerovaných sekvencí. V případě, že je podpora kandidátní sekvence větší nebo rovna uživatelem zadané minimální podpoře, je sekvence uložena a rekurzivně se nad odpovídajícím uzlem opakuje průchod do hloubky. V případě, že je podpora sekvence menší než minimální podpora, odpovídající uzel se na základě aplikace Apriori vlastnosti již neprohledává.

Tato metoda průchodu má velký prohledávací prostor, proto autoři algoritmu SPAM pro zlepšení výkonu navrhli prořezávací techniku, která tento prostor redukuje [7]. Technika je opět založena na Apriori vlastnosti a její snahou je pro každý uzel n stromu redukovat množinu kandidátních položek S_n pro S-krok a množinu kandidátních položek I_n pro I-krok.

První část optimalizační techniky prořezává v lexikografickém sekvenčním stromě syny S-kroku. Předpokládejme sekvenci s v uzlu n a uvažujme její sekvenčně-rozšířené sekvence $s_a =< s, \{i_j\} >$ a $s_b =< s, \{i_k\} >$. Dále předpokládejme, že s_a je frekventovaná a s_b není frekventovaná sekvence. Na základě využití Apriori vlastnosti víme, že sekvence $<s, \{i_j\}, \{i_k\} >$ a $< s, \{i_j, i_k\} >$ nemohou být taktéž frekventované, protože obsahují nefrekventovanou podsekvenci s_b. Díky této znalosti lze položku i_k odstranit z množin S_m a I_m, kde m je libovolný uzel odpovídající sekvenčně-rozšířenému potomku sekvence s.

Druhá část optimalizační techniky je zaměřena na prořezávání synů v I-kroku. Uvažujme sekvenci $s =< s', \{i_1, \dots, i_n\} >$ a její množinově-rozšířené sekvence $s_a =< s', \{i_1, \dots, i_n, i_j\} >$ a $s_b =< s', \{i_1, \dots, i_n, i_k\} >$, kde $i_j < i_k$. V případě, že s_a je frekventovaná sekvence a s_b není frekventovaná, pak na základě Apriori vlastnosti víme, že sekvence $< s', \{i_1, \dots, i_n, i_j, i_k\} >$ není také frekventovaná. Díky této znalosti můžeme z množiny kandidátních položek I_m odstranit položku i_k, kde m je jakýkoliv uzel odpovídající množinově-rozšířenému potomku sekvence s.

Tabulka 3.3: Příklad databáze sekvencí.

sid	sekvence
s_1	$< a(abc)(ac) >$
s_2	$< (ad)c(bc)a >$
s_3	$< (ab)d >$

Příklad 3.7 Ilustrujme si prořezávání kandidátních položek na příkladu. Předpokládejme, že se v sekvenčním stromě nacházíme na sekvenci $< a >$, dále uvažujme, že množina kandidátních položek pro S-krok je $S_{<a>} = \{a, b, c, d\}$ a množina kandidátních položek pro I-krok je $I_{<a>} = \{b, c, d\}$. Všechny možné sekvenčně-rozšířené sekvence jsou: $< aa >$, $< ab >$, $< ac >$ a $< ad >$. Po vypočtení jejich podpor zjistíme, že $< ac >$ a $< ad >$ nejsou frekventované. Proto pro uzly $< aa >$

a $< ab >$ není potřeba provádět S-krok a I-krok s položkami c a d, tj. $S_{<aa>} = S_{<ab>} = \{a, b\}$, $I_{<aa>} = \{b\}$, $I_{<ab>} = \emptyset$.

Množinově-rozšířené sekvence pro uzel $< a >$ jsou: $< (ab) >$, $< (ac) >$, $< (ad) >$. Po vypočtení podpory těchto kandidátů zjistíme, že sekvence $< (ac) >$ není frekventovaná. A proto sekvence $< (abc) >$ na základě Apriori vlastnosti také nemůže být frekventovaná, tj. $I_{<(ab)>} = \{d\}$. Algoritmus 3.3 ukazuje pseudokód algoritmu SPAM s aplikací prořezávacích technik.

Algoritmus 3.3: SPAM – Průchod do hloubky a aplikace prořezávacích technik.

```
procedure SPAM (n=<s_1,…,s_k>, S_n, I_n) do
    S_temp = ∅;
    I_temp = ∅;
    foreach i ∈ S_n do
      if (< s_1,…,s_k,{i} > je frekventovaná) do
        S_temp = S_temp ∪ {i};
      end;
    end;
    foreach i ∈ S_temp do
      call SPAM (< s_1,…,s_k,{i} >, S_temp, všechny položky v S_temp > i);
    end;
    foreach i ∈ I_n do
      if(< s_1,…,s_k ∪ {i} > je frekventovaná) do
        I_temp = I_temp ∪ {i};
      end;
    end;
    foreach i ∈ I_temp do
      call SPAM (< s_1,…,s_k ∪ {i} >, S_temp, všechny položky v I_temp > i);
    end;
end;
```

Vertikální reprezentace databáze

SPAM využívá efektivní bitmapovou reprezentaci databáze. Během prvního průchodu databází se pro každou frekventovanou položku vytvoří vertikální bitmapa. Bitmapa obsahuje bit příslušný ke každé události v databázi. Když se položka vyskytuje v události je odpovídající bit nastaven na jedna, jinak je bit nastaven na nulu. Myšlenku bitmap položek lze dále rozšířit na sekvence. Předpokládejme, že máme bitmapy pro položky i a j. Pak bitmapa pro sekvenci $s =< (i, j) >$ se získá aplikací operace AND nad bitmapami položek i a j.

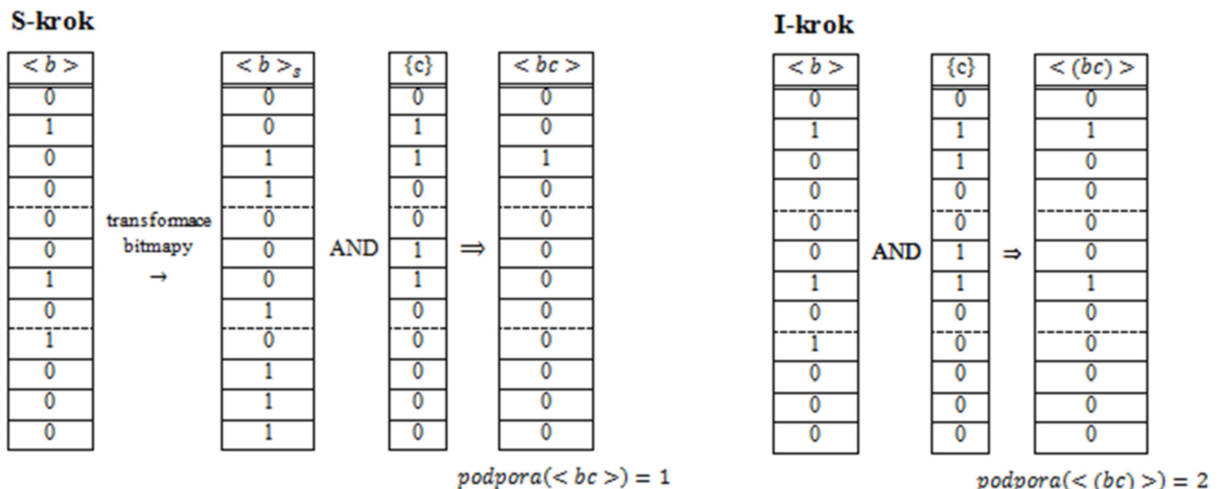
Hlavní výhodou vertikálních bitmap je schopnost efektivního počítání podpory. Autoři algoritmu SPAM roztřídili sekvence do skupin na základě jejich velikostí. Pro velikost sekvence v rozmezích $2^k + 1$ a 2^{k+1} bude výsledná část bitmapy obsahovat 2^{k+1} bitů (2^{k+1}-bit sekvence). Minimální hodnota proměnné $k = 1$ a maximální hodnota $k = 5$. Z omezení vyplývá, že SPAM podporuje sekvence s maximálně 64 událostmi. Výpočet podpory probíhá tak, že se v bitmapě zkontroluje, zdali jednotlivé 2^{k+1}-bit sekvence neobsahují všechny bity nastaveny na nulu. Obrázek 3.2 zobrazuje

bitmapovou reprezentaci databáze sekvencí S, danou tabulkou 3.3. Například položka {d} má podporu dva, protože části s sid: 2 a 3 neobsahují samé nuly.

sid	{a}	{b}	{c}	{d}	
1	1	0	0	0	4-bit sekvence
1	1	1	1	0	
1	1	0	1	0	
-	0	0	0	0	
2	1	0	0	1	4-bit sekvence
2	0	0	1	0	
2	0	1	1	0	
2	1	0	0	0	
3	1	1	0	0	4-bit sekvence
3	0	0	0	1	
-	0	0	0	0	
-	0	0	0	0	

Obrázek 3.2: Bitmapová reprezentace databáze sekvencí z tabulky 3.3.

Nakonec je potřeba zmínit, jak probíhá generování kandidátních sekvencí pomocí bitmapové reprezentace. V případě I-kroku se výsledná bitmapa generuje provedením operace AND nad bitmapou sekvence a bitmapou připojované položky. V případě S-kroku je potřeba nejdříve vytvořit transformovanou bitmapu. Ta se obdrží tak, že se v každé 2^{k+1}-bit sekvenci najde index i prvního bitu nastaveného na jedna. Všechny bity následující za indexem i se v 2^{k+1}-bit sekvenci nastaví na jedna a bity s indexem menším, nebo rovno i se nastaví na nulu. Nakonec se opět provede operace AND nad transformovanou bitmapou a bitmapou připojované položky. Proces generování bitmap je zobrazen na obrázku 3.3.



Obrázek 3.3: S-krok a I-krok pomocí bitmapové reprezentace sekvencí.

Při vzájemném porovnání výkonu algoritmu SPAM s algoritmem PrefixSpan autoři ukázali, že je SPAM rychlejší při dolování dlouhých sekvenčních vzorů nad velkými datovými sadami. Kvůli bitmapové reprezentaci databáze má však SPAM větší nároky na paměť. Dostatek paměti je důležitý,

protože algoritmus SPAM vyžaduje, aby se všechna data vzniklá v procesu dolování, vlezla do paměti počítače.

3.3.5 LAPIN-SPAM

Algoritmus LAPIN-SPAM (Last Position Induction Sequential Pattern Mining) [9] kombinuje nejvýkonnější vlastnosti z algoritmů SPAM [7] a LAPIN [10]. Z prvně jmenovaného přebírá bitmapovou reprezentaci databáze, generování kandidátních sekvencí pomocí S-kroku a I-kroku, lexikografický sekvenční strom a průchod do hloubky. Oproti SPAMu se algoritmus LAPIN-SPAM snaží zefektivnit počítání podpory kandidátních sekvencí. Místo provádění operace AND nad bitmapami a testováním, zdali jednotlivé 2^{k+1}-bit sekvence obsahují samé nuly, LAPIN-SPAM počítá podporu kandidátních sekvencí pomocí pomocné tabulky. Tato tabulka, převzata z algoritmu LAPIN, se vybuduje během prvního průchodu vstupní databází a uchovává informace o posledních pozicích položek v jednotlivých sekvencích. LAPIN-SPAM implementuje princip brzkého prořezávání kandidátních sekvencí. To znamená, že se snaží co nejdříve v procesu dolování prořezat prohledávaný prostor a snížit tak čas potřebný k získání výsledných sekvenčních vzorů.

Indukce pozice

Jak již bylo zmíněno, hlavním rozdílem LAPIN-SPAMu oproti SPAMu je počítání podpory kandidátních sekvencí. Algoritmus LAPIN-SPAM počítá podporu na základě principu indukce pozice. Princip zní následovně: *Když je poslední pozice položky α v sekvenci menší, nebo rovna pozici poslední položky v prefix k-sekvenci s, pak již položka α nemůže být připojena k sekvenci s a vytvořit tak novou frekventovanou $(k+1)$-sekvenci* [10]. Uvažujme například sekvenci s_2 z tabulky 3.3, prefix sekvenci $< (ad)c >$ a položku d. Pozice prefixu $< (ad)c >$ v sekvenci s_2 je dva, poslední pozice položky d je v sekvenci s_2 rovná jedné. Bezpečně lze tedy při počítání podpory určit, že vygenerovaný kandidát $< (ad)cd >$ se v sekvenci s_2 nevyskytuje. Díky této vlastnosti se dá vyhnout provádění mnoho náročných operací AND nad bitmapami, jak to provádí algoritmus SPAM.

Tabulka výskytů položek

Neefektivní přístup určení, zdali je kandidát frekventovaný, lze implementovat pomocí porovnání poslední pozice položky s aktuální pozicí prefixu. Tato operace by však měla stejnou časovou náročnost jako operace AND ve SPAMu. Proto autoři algoritmu LAPIN-SPAM vytvořili tabulku s názvem *ITEM_IS_EXIST_TABLE*. Tato tabulka se vytvoří během prvního průchodu vstupní databází. Během procesu dolování se pak informace o tom, jestli se položka vyskytuje za aktuální pozicí prefixu, získá právě z této tabulky. Podobu této tabulky pro sekvenci $s =< ac(bd)c(ab) >$ zobrazuje obrázek 3.4. Sekvence s obsahuje pět událostí, proto její výsledná část bitmapy má délku osm (8-bit sekvence), jak definuje algoritmus SPAM.

	a	b	c	d
1	1	1	1	1
2	1	1	1	1
3	1	1	1	0
4	1	1	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

Obrázek 3.4: Tabulka ITEM_IS_EXIST_TABLE pro sekvenci $s =< ac(bd)c(ab) >$. Převzato z [9].

První sloupec tabulky určuje číslo pozice události v sekvenci a horní řádek obsahuje identifikátor položky. Řádek tabulky je implementován pomocí vektoru bitů. Bit s hodnotou jedna představuje výskyt položky za danou pozicí, nula naopak říká, že se položka v sekvenci za danou pozicí již nevyskytuje. Velikost vektoru je dána počtem položek v databázi sekvencí. Například pro aktuální pozici dva dostaneme vektor s hodnotou 1111. Tato hodnota značí, že se všechny položky vyskytují za aktuální pozicí prefixu. Pro pozici čtyři dostaneme vektor 1100. Hodnota vyjadřuje, že se pouze položky a, b vyskytují za aktuální pozicí prefixu.

Podpora kandidáta se vypočte akumulací hodnot získaných z vektorů pro všechny sekvence. Algoritmus 3.4 ukazuje část výpočtu podpory pro algoritmus LAPIN-SPAM, zbytek algoritmu je totožný s algoritmem SPAM a proto zde není uveden. Výše zmíněné vylepšení je navrženo pouze pro S-krok.

Algoritmus 3.4: Vyhledání frekventovaných položek v **LAPIN-SPAM.**

Vstup:

```
    α: sekvenční vzor/aktuální prefix;
    S_α: seznam kandidátních položek pro S-krok;
    min_support: uživatelem zadaná minimální podpora;
```

Výstup:

```
    freqItemList_α: seznam lokálně frekventovaných položek;
```

Metoda:

```
    foreach sekvenci s in DB do
      bitV = získej bitový vektor pro aktuální pozici prefixu α v s
             z tabulky ITEM_IS_EXIST_TABLE;
      foreach položku β in S_α do
        supportList[β] += bitV[β];
      end;
    end;
    foreach položku γ in supportList do
      if (supportList[γ] >= min_support) do
        freqItemList_α->add(γ);
      end;
    end;
```

Autoři algoritmu LAPIN-SPAM provedli porovnání jejich metody s algoritmem SPAM. Dle jejich výsledků je LAPIN-SPAM dvakrát až třikrát rychlejší než SPAM. Na druhou stranu však zabírá dvakrát více paměti. Tuto informaci je potřeba respektovat v prostředí s omezenou pamětí, protože jde o algoritmus, který efektivně pracuje pouze v případě, že se všechna data vytvořená během procesu dolování vlezou do paměti.

3.3.6 BIDE

BIDE (BI-Directional Extension) [3] je aktuálně nejefektivnější algoritmus pro dolování frekventovaných uzavřených sekvencí. Připomeňme si, že sekvence je uzavřená, když v databázi neexistuje její nadsekvence se stejnou podporou. BIDE prochází vstupní prostor a získává sekvenční vzory stejným způsobem, jako algoritmus PrefixSpan. Navíc však přináší nové výkonné techniky pro určení uzavřenosti vzorů a prořezání prohledávacího prostoru. Z důvodu, že BIDE vychází z algoritmu PrefixSpan, tato kapitola popisuje pouze jeho nové vlastnosti.

Oboustranná kontrola uzavřenosti

V průběhu procesu dolování musí BIDE pro každou nalezenou frekventovanou sekvenci určit, zdali jde o sekvenci uzavřenou. Pro tuto činnost byla navržena *kontrola uzavřenosti oboustranného rozšíření* (*BI-Directional Extension closure checking*) [3]. Tato kontrola je postavena na následujícím tvrzení. V případě, že sekvence $s =< e_1 e_2 \dots e_n >$ není uzavřená, pak musí platit nejméně jedna z následujících podmínek:

1. V databázi sekvencí existuje sekvence $s' =< e_1 e_2 \dots e_n e' >$, pro kterou platí: $podpora(s) = podpora(s')$. Tedy každá sekvence v databázi, která obsahuje sekvenci s, obsahuje také událost e', kterou lze připojit na konec sekvence s a získat tak novou sekvenci s'. Událost e' se pak nazývá *událost dopředného rozšíření* (*forward-extension event*) a s' se nazývá *dopředně rozšířená sekvence* (*forward-extension sequence*).
2. V databázi existuje sekvence $s' =< e_1 e_2 \dots e_i e' e_{i+1} \dots e_n >$ nebo $s' =< e' e_1 e_2 \dots e_n >$, pro kterou platí: $podpora(s) = podpora(s')$. Sekvence s' se pak nazývá *zpětně rozšířená sekvence* (*backward-extension sequence*) a e' *událost zpětného rozšíření* (*backward-extension event*).

Pro určení uzavřenosti sekvence se tedy dle tvrzení musí zjistit, zdali nalezený sekvenční vzor neobsahuje událost dopředného ani zpětného rozšíření.

Definice 3.4 Pro určení události zpětného rozšíření je potřeba definovat následující pojmy [3]. Předpokládejme vstupní sekvenci S, která obsahuje prefix $\alpha =< e_1 e_2 \dots e_i >$. *Poslední instance prefixu* (*last instance of a prefix*) α v S je podsekvence od začátku S do posledního výskytu událostí e_i

v sekvenci S. Například poslední instance prefixu $\alpha =< ab >$ v sekvenci $S =< abbca >$ je rovna $< abb >$.

Dále přepokládejme, že sekvence S obsahuje prefix $\beta =< e_1 e_2 \dots e_n >$. Pak LL_i vzhledem k prefixu β (*i-th last-in-last appearance w.r.t. a prefix*) v sekvenci S je definováno rekurzivně takto:

1. Pro $i = n$, to je poslední výskyt události e_i v poslední instanci prefixu β v S.
2. Pro $1 \leq i < n$, to je poslední výskyt události e_i v poslední instanci prefixu β v S a zároveň LL_i se musí vyskytovat před LL_{i+1}.

Například pro sekvenci $S_1 =< caabc >$ a prefix $\alpha =< ab >$, je LL_1 vzhledem k prefixu α v S_1 roven druhému výskytu položky a v S_1. Pro sekvenci $S_2 =< cacac >$ a prefix $\beta =< cac >$ je LL_1 vzhledem k prefixu β v S_2 roven druhému výskytu položky c v S_2, LL_2 je roven druhému výskytu položky a v S_2 a LL_3 je roven třetímu výskytu položky c v sekvenci S_2.

Uvažujme sekvenci S, která obsahuje prefix $\gamma =< e_1 e_2 \dots e_n >$. Pak *i-tá maximální perioda prefixu* (*i-th maximum period of a prefix*) γ v S je definována následovně:

1. Pro $1 < i \leq n$, je to část sekvence mezi koncem prvního výskytu prefixu $< e_1 e_2 \dots e_{i-1} >$ v S a začátkem LL_i vzhledem k prefixu γ v S.
2. Pro $i = 1$, je to část sekvence v S před LL_1 vzhledem k prefixu γ.

Například pro sekvenci $S_1 =< abcb >$ a prefix $\gamma_1 =< ab >$ je první maximální perioda prefixu γ_1 v S_1 prázdná. Druhá maximální perioda prefixu γ_1 v S_1 je $< bc >$. Pro sekvenci $S_2 =< abbb >$ a prefix $\gamma_2 =< bb >$ je první maximální perioda rovna $< ab >$ a druhá maximální perioda je $< b >$.

Díky využití PrefixSpan frameworku lze snadno vyhledat události dopředného rozšíření. Množina všech událostí dopředného rozšíření pro prefix α je rovna množině lokálně frekventovaných položek v α-projektované databázi, jejichž podpora je rovna podpoře prefixu α. Události zpětného rozšíření se pro prefix $\alpha =< e_1 e_2 \dots e_n >$ obdrží prohledáním maximálních period prefixu α. Události, které se objevují ve všech maximálních periodách, tvoří množinu událostí zpětného rozšíření. V případě, že se nenalezne žádné rozšíření prefix α je uzavřený sekvenční vzor. Důkazy o správnosti předcházejících tvrzení lze nalézt v [3].

Redukce vstupního prostoru

Autoři algoritmu navrhli velmi efektivní metodu pro prořezání vstupního prostoru. Tato metoda s názvem *BackScan* využívá obousměrného rozšíření a maximálních period pro určení, zdali projektovaná databáze může obsahovat doposud nenalezené frekventované uzavřené sekvence. V případě, že je nemůže obsahovat, lze v brzké fázi procesu dolování ukončit prohledávání dané projektované databáze. Více informací o této velmi účinné metodě lze nalézt v literatuře [3]. Algoritmus 3.5 ukazuje pseudokód algoritmu BIDE s aplikací kontroly uzavřenosti a také s využitím techniky *BackScan.*

Algoritmus 3.5: BIDE

Vstup:

```
    S: databáze sekvencí;
    min_support: uživatelem zadaná minimální podpora;
```

Výstup:

```
    closedSeqPatternList: seznam uzavřených sekvenčních vzorů;
```

Metoda:

```
    call BIDE (<>, S);

    procedure BIDE (α, S|α):
      α: prefix,
      S|α: α-projektovaná databáze pro α != <>, jinak databáze sekvencí S;

      locFreqItems = vyhledej lokálně frekventované položky v S|α;
      forwardExtCount = |{i in locFreqItems | sup(i) == sup(α)}|;
      backwardExtCount = najdi v S|α události zpětného rozšíření;

      if (forwardExtCount == 0 && backwardExtCount == 0) do
        closedSeqPatternList->add(α);
      end;

      foreach i in locFreqItems do
        α' = < α, {i} >;
        S|α' = vytvoř α'-projektovanou databázi;
        if (!BackScan (α', S|α')) do
          call BIDE (α', S|α');
        end;
      end;
    end;
```

Při porovnání BIDE s algoritmy, které hledají úplnou množinu sekvenčních vzorů, je BIDE na reálných datech výkonnější. Během procesu dolování z dat BIDE spotřebovává velmi málo paměti. Je to způsobeno převážně tím, že si nemusí uchovávat historii nalezených uzavřených sekvenčních vzorů, aby ověřil uzavřenost nově nalezeného vzoru.

3.4 Sekvenční pravidla

Získané sekvenční vzory mají především popisný charakter. Vzory lze však využít i pro předpověď budoucích hodnot. K tomuto účelu se ze sekvenčních vzorů generují tzv. *sekvenční pravidla* (*sequential rules*). Podobu sekvenčních vzorů definoval M. Zaki v [8]. Sekvenční pravidlo je složeno z levé a pravé strany, kde na každé straně leží frekventovaná sekvence:

$$< (pc, kamera) > \Rightarrow < (pc, kamera)(tiskárna) > [\, podpora = 5\%, spolehlivost = 75\%].$$

Toto pravidlo vyjadřuje, že v 5% všech nákupů si zákazníci koupili dohromady pc a kameru a v nějakém *následném* nákupu si koupili tiskárnu. Spolehlivost pak říká, že zákazník, který si v jednom nákupu koupí pc a kameru si s pravděpodobností 75% v příštím nákupu zakoupí tiskárnu.

Algoritmus 3.6 zobrazuje pseudokód metody pro generování pravidel ze sekvenčních vzorů. Tento algoritmus publikoval M. Zaki v [8].

Algoritmus 3.6: Generování sekvenčních pravidel.

Vstup:

```
    SP: množina sekvenčních vzorů;
    min_conf: uživatelem zadaná minimální spolehlivost;
```

Výstup:

```
    seqRules: seznam sekvenčních pravidel;
```

Metoda:

```
    foreach β in SP do
      foreach subsequence α ⊑ β do
        conf = support(β)/support(α);
        if (conf ≥ min_conf) do
          seqRules->add(α ⇒ β, conf);
        end;
      end;
    end;
```

4 Dolování na platformě SQL Server

Dolování z dat lze provádět na mnoha komerčních i volně dostupných platformách. Tato kapitola představuje Microsoft SQL Server 2008 jako nástroj pro dolování z dat. Kapitola 4.1 popisuje architekturu tohoto komplexního systému. Kapitole 4.2 ukazuje, jak lze pomocí analytických služeb SQL Serveru dolovat data. SQL Server obsahuje několik vestavěných algoritmů pro analýzu dat. Seznam těchto algoritmů však není vyčerpávající a proto tvůrci systému navrhli rozhraní pro rozšíření SQL Serveru o vlastní dolovací algoritmy. Více informací o tvorbě vlastního algoritmu pro tento systém lze nalézt v kapitole 4.3.

4.1 Microsoft SQL Server

Microsoft SQL Server je robustní databázový a analytický systém vyvinutý společností Microsoft. Platforma nabízí kvalitní podporu pro nasazení procesu získávání znalostí z dat. Tato kapitola popisuje jednotlivé části, které systém nabízí. Následující text čerpá ze zdrojů [24, 25, 26].

Obrázek 4.1: Architektura Microsoft SQL Serveru. Převzato a upraveno z [24].

Platforma Microsoft SQL Server se skládá z následujících částí:

- **Relační databáze** – Klíčová a nejvyužívanější komponenta SQL Serveru, která umožňuje ukládat, načítat, zpracovávat a zabezpečit data. V tomto úložišti se nachází data, nad kterými se provádí analýza dat ve formě databázových tabulek.
- **Integrační služby** – Tato služba obsahuje sadu nástrojů pro integraci, čištění, transformaci a výběr dat. Využívá se převážně v prvních fázích procesu získávání znalostí z databází.

- **Replikační služby** – Služba slouží pro distribuci kopie dat. Díky ní mohou mobilní a odpojení uživatelé používat na cestách místní sady dat, provádět lokální změny a poté všechny změny najednou synchronizovat se serverem.
- **Reportovací služby** – Slouží pro zpřístupnění dat a výsledků analýz uživatelům ve vhodné a srozumitelné formě. Výsledné reporty prezentují získané znalosti převážně pomocí tabulek a různých grafů.
- **Analytické služby** – Zásadní komponenta pro dolování z dat. Tato služba navíc obsahuje funkce OLAP pro rychlou a pokročilou analýzu velkých a složitých datových sad s využitím vícedimenzionálních úložišť. Možnosti analytických služeb se zaměřením na dolování z dat jsou popsány v kapitole 4.2.
- **Nástroje pro správu –** Systém SQL Server se dodává spolu s několika nástroji, které umožňují správu databázových objektů a interakci s daty. Hlavním nástrojem je SQL Server Management Studio, pomocí kterého lze spravovat, ladit a monitorovat jednotlivé služby systému.
- **Nástroje pro vývojáře** – SQL Server nabízí integrované nástroje pro vývojáře určené pro extrakci, transformaci a načítání dat, dolování z dat, funkce OLAP a vytváření sestav. Pro vytváření a správu projektů spojených s procesem získávání znalostí se využívá nástroj Business Intelligence Development Studio (BIDS).

4.2 Dolování z dat

Představili jsme si jednotlivé komponenty Microsoft SQL Serveru, které se využívají pro pokročilou analýzu dat. V této kapitole si ukážeme možnosti využití analytických služeb SQL Serveru se zaměřením na dolování z dat. Také jsou v kapitole stručně popsány vestavěné dolovací algoritmy, které jsou součástí SQL Serveru.

4.2.1 Vytvoření dolovací úlohy

Chceme-li využít dolovacích nástrojů, které nám poskytuje SQL Server, je potřeba pomocí Business Intelligence Development Studia (BIDS) vybudovat analytický projekt a definovat dolovací model. Tato podkapitola popisuje jednotlivé kroky budování analytického projektu na platformě Microsoft SQL Server 2008 [25, 27]:

1. **Definice zdroje dat/pohledů** – V prvním kroku je potřeba definovat datový zdroj a pohledy na data (*data source*, *data source views*). Pohled je definován výběrem databázových tabulek a pohledů, eventuálně lze napsat složitější pojmenovaný SQL dotaz.

2. **Dolovací struktura** – Struktura (*mining structure*) popisuje podobu business problému, který budeme analyzovat. Obvykle jeden databázový záznam reprezentuje jednu zkoumanou entitu reálného světa a primární klíč pak jednoznačně identifikuje tento záznam. V tomto případě zvolenou tabulku z datového pohledu označíme jako *tabulku případů* (*case table*). Může však nastat situace, kdy máme data rozložena ve více tabulkách. Typickým příkladem jsou transakční databáze a databáze sekvencí. V tomto případě musíme jako tabulku případů označit tu, která uchovává identifikátor transakce nebo sekvence. Navíc však musíme vybrat *vnořenou tabulku* (*nested table*), která nese položky jednotlivých transakcí nebo událostí sekvencí.

   V tomto kroku se dále jednotlivým sloupcům z datového pohledu vymezí jejich datový a obsahový typ. Obsahový typ (*content type*) definuje, jak bude dolovací mechanismus zacházet s daným sloupcem. Možné hodnoty jsou: klíč, diskrétní typ, spojitý typ a speciálně pro sekvence je zde typ s názvem *sekvenční klíč* (*key sequence*). Tento klíč určuje uspořádání událostí v sekvenci. Například to může být čas nákupu zboží nebo doba přístupu na webovou stránku. Výběr typů pro dolování sekvenčních vzorů ukazuje obrázek 4.2.

3. **Dolovací model** – Pro dolovací strukturu lze vytvořit jeden nebo více *dolovacích modelů* (*mining model*). Každý model je reprezentován dolovacím algoritmem a výběrem sloupců z dolovací struktury, které jsou relevantní pro daný typ algoritmu. Pro každý sloupec se navíc musí určit, zdali jde o vstupní nebo predikovaný sloupec. Díky možnosti využití více modelů nad stejnou strukturou lze podle výsledků analýz snadno určit, který algoritmus je nejvhodnější pro zkoumaný problém.

   Před spuštěním samotného dolování může být potřeba nastavit parametry dolování. Mezi typické parametry patří minimální podpora, minimální spolehlivost, počet výstupních shluků atd.

4. **Prezentace modelů/vzorů** – Po dokončení dolování se získané vzory a modely zobrazí v BIDS na záložce s prohlížeči výsledků (*mining model viewer)*. Tato záložka obsahuje vždy generický stromový prohlížeč výsledků (*generic content tree viewer*). Pro vestavěné algoritmy v SQL Serveru lze využít i bohatší grafické prohlížeče, které zobrazují data ve formě různých grafů a tabulek. Obrázek 4.3 zobrazuje získané sekvenční vzory v generickém prohlížeči pro databázi sekvencí danou tabulkou 3.3 a minimální podporu *min_support* = 2.

5. **Predikce** – V případě algoritmů, které provádí klasifikaci nebo predikci, lze výsledný model aplikovat na nová data a předpovídat tak budoucí hodnoty.

Obrázek 4.2: Výběr datových a obsahových typů sloupců pro dolování sekvenčních vzorů.

Mining Structure | Mining Models | Mining Model Viewer | Mining Accuracy Chart | Mining Model Prediction

Mining Model: PrefixSpan   Viewer: Microsoft Generic Content Tree Viewer

Node Caption (Unique ID)

(PrefixSpan)
- (00003 - <'a'>)
- (00002 - <'a' 'a'>)
- (00002 - <'a' 'b'>)
- (00002 - <'a' 'b' 'a'>)
- (00002 - <'a' ('b' 'c')>)
- (00002 - <'a' ('b' 'c') 'a'>)
- (00002 - <'a' 'c'>)
- (00002 - <'a' 'c' 'a'>)
- (00002 - <'a' 'c' 'c'>)
- (00002 - <('a' 'b')>)
- (00003 - <'b'>)
- (00002 - <'b' 'a'>)
- (00002 - <('b' 'c')>)
- (00002 - <('b' 'c') 'a'>)
- (00002 - <'c'>)
- (00002 - <'c' 'a'>)
- (00002 - <'c' 'c'>)
- (00002 - <'d'>)
- (00002 - <'a' 'b'> => 00002 - <'a' 'b' 'a'>)
- (00002 - <'a' ('b' 'c')> => 00002 - <'a' ('b' 'c') 'a'>)
- (00002 - <'a' 'c'> => 00002 - <'a' 'c' 'a'>)
- (00002 - <'a' 'c'> => 00002 - <'a' 'c' 'c'>)
- (00002 - <('b' 'c')> => 00002 - <('b' 'c') 'a'>)
- (00002 - <'c'> => 00002 - <'c' 'a'>)
- (00002 - <'c'> => 00002 - <'c' 'c'>)

Node Details

MODEL_CATALOG	ZTDIP SequenceDatabase
MODEL_SCHEMA	
MODEL_NAME	PrefixSpan
ATTRIBUTE_NAME	
NODE_NAME	PrefixSpan
NODE_UNIQUE_NAME	PrefixSpan
NODE_TYPE	1 (Model)
NODE_GUID	
NODE_CAPTION	
CHILDREN_CARDINALITY	25
PARENT_UNIQUE_NAME	
NODE_DESCRIPTION	SEQUENCES_COUNT=18; MIN_SUPPORT=2; MAX_SUPPORT=3; MIN_SEQUENCE_SIZE=3; MAX_SEQUENCE_SIZE=2; MIN_SEQUENCE_LENGTH=3; MAX_SEQUENCE_LENGTH=2; RULE_COUNT=7; MIN_CONFIDENCE=1; MAX_CONFIDENCE=1;
NODE_RULE	
MARGINAL_RULE	
NODE_PROBABILITY	1
MARGINAL_PROBABILITY	1
NODE_DISTRIBUTION	ATTRIBUTE_NAME ATTRIBUTE_VALUE SUPPORT PROBABILITY VARIANCE VALUET
NODE_SUPPORT	3
MSOLAP_MODEL_COLUMN	
MSOLAP_NODE_SCORE	0

Obrázek 4.3: Výsledek dolování sekvenčních vzorů v prostředí BIDS.

4.2.2 Vestavěné algoritmy

SQL Server poskytuje několik vestavěných algoritmů pro dolování z dat. Následuje stručný popis nejpoužívanějších algoritmů [25, 27]:

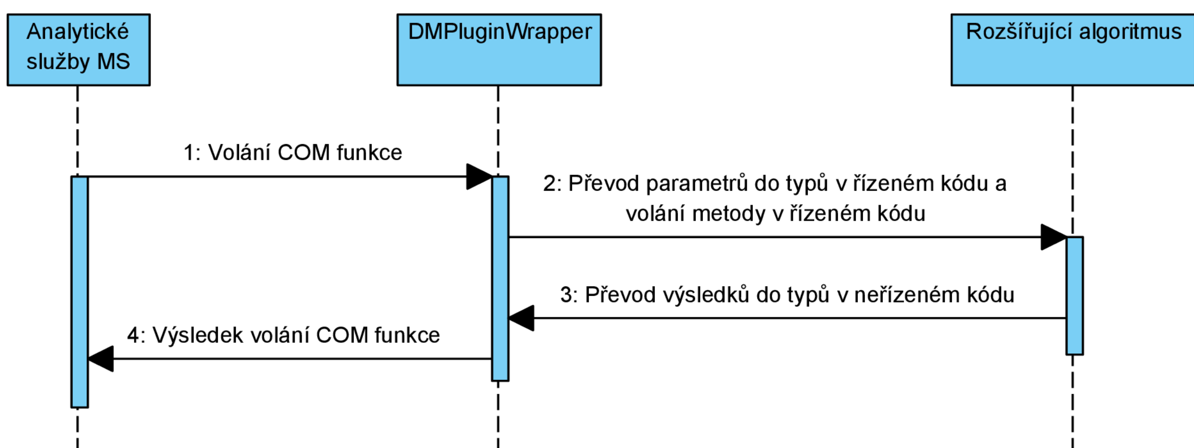
- **Microsoft Decision Trees** – Tento algoritmus na základě vlastností trénovacích dat vytváří rozhodovací strom. Výsledný strom lze využít pro klasifikaci, predikci a asociační analýzu.
- **Microsoft Association Rules** – Algoritmus pro vyhledávání frekventovaných položek v datech a také pro vyhledávání asociací mezi položkami.
- **Microsoft Clustering** – Pro shlukovou analýzu byl tvůrci SQL Serveru vytvořen tento algoritmus. V případě nalezení shluků, lze jednotlivé shluky dále analyzovat odděleně a zkoumat je tak do větších detailů.

- **Microsoft Linear Regression** – Algoritmus se používá pro modelování lineárních vztahů mezi dvěma spojitými hodnotami. Hlavním využitím je predikce spojitých hodnot.
- **Microsoft Naive Bayes** – Velmi rychlý a poměrně přesný algoritmus založený na Bayesově větě, pomocí kterého je možné pracovat přímo s pravděpodobnostmi. Tento algoritmus kombinuje nová data s předcházejícími znalostmi. Používá se pro klasifikaci a predikci.
- **Microsoft Neural Network** – Algoritmus, který pomocí neuronových sítí provádí klasifikaci a predikci hodnot.

4.3 Rozšíření analytických služeb SQL Serveru

Microsoft SQL Server poskytuje mechanismy k rozšíření analytických služeb o vlastní dolovací algoritmy a prohlížeče výsledků. Algoritmy komunikují se serverem pomocí stejných rozhraní jako vestavěné algoritmy firmy Microsoft. Rozšiřující dolovací algoritmy (*plugins*) lze implementovat následujícími způsoby:

1. pomocí *COM* (*Component Object Model*) objektů v jazyce C++, nebo
2. pomocí *řízeného kódu* (*managed code*) v prostředí .NET. V tomto případě se však jedná pouze o nadstavbu, která pomocí rozhraní a bázových tříd zastiňuje práci s COM objekty. Implementace pomocí tohoto přístupu sice zjednodušuje práci, díky možnosti využití vysokoúrovňového programovacího jazyku C#, na druhou stranu je potřeba počítat s výkonnostními ztrátami. Ztráty jsou především způsobeny převáděním dat mezi řízeným a neřízeným kódem. Obrázek 4.4 zobrazuje komunikaci mezi analytickými službami SQL Serveru a rozšiřujícím algoritmem. Převod volání mezi řízeným a neřízeným kódem zařizuje volně dostupná knihovna *DMPluginWrapper*.



Obrázek 4.4: Diagram komunikace mezi analytickými službami a rozšiřujícím algoritmem. Převzato z [28].

4.3.1 Životní cyklus algoritmu

Tato kapitola stručně představuje hlavní činnosti, které musí každý rozšiřující algoritmus implementovat. Informace podané v této kapitole se vztahují na algoritmy programované v řízeném kódu. Detailnější informace lze nalézt v literatuře [27, 28].

Každý rozšiřující algoritmus musí provést následující tři činnosti:

- Popsat své vlastnosti a funkcionalitu.
- Vyhledat vzory a modely.
- Prezentovat nalezené znalosti.

Popis vlastností a funkcionality algoritmu

V první fázi je potřeba dodat SQL Serveru informace o vlastnostech, omezeních a funkcionalitě rozšiřujícího algoritmu. Pro tuto činnost se musí implementovat třída, která dědí od abstraktní třídy *AlgorithmMetadataBase*. Tato třída pak pomocí přetížení abstraktních a virtuálních metod definuje název algoritmu, vstupní parametry, podporované obsahové typy atd. Navíc také validuje správnost uživatelem zadaných atributů a vytváří instanci samotného dolovacího algoritmu.

Vyhledání vzorů a modelů

Vlastní dolování z dat se provádí v této fázi. Pro tuto funkci je potřeba vytvořit třídu, která dědí od abstraktní třídy *AlgorithmBase*. SQL Server pak v případě potřeby dolování vzorů volá abstraktní metodu *InsertCases()*. V implementaci této metody musí programátor poskytnout mechanismus pro dolování vzorů z trénovací množiny dat. Metoda by navíc měla informovat SQL Server o postupu zpracování dat pomocí notifikačních zpráv. Třída navíc může implementovat metody pro uložení/načtení vzorů do/z databáze pomocí metod *SaveContent()* a *LoadContent()*. Pro predikci budoucích hodnot slouží metoda *Predict()*.

Pro načtení a zpracování vstupních dat je potřeba implementovat rozhraní *ICaseProcessor*. Toto rozhraní má metodu s názvem *ProcessCase()*. Metoda je postupně volána pro každý vstupní případ (*case)*. Obyčejně jeden případ obsahuje atributy jednoho záznamu v tabulce. V situacích, kdy byla vybrána i vnořená (*nested*) tabulka, jeden případ navíc obsahuje všechny vázané záznamy z vnořené tabulky. Tabulka 4.1 ukazuje vstupní data z pohledu uložení v databázi.

Tabulka 4.2 4.2 zobrazuje podobu dat zpřístupněných dolovacímu algoritmu. Podoba dat pro dolovací algoritmus je ovlivněna nastavením obsahových typů sloupců, proto je v hlavičce tabulky 4.1 uveden zvolený obsahový typ. Například sloupec s obsahovým typem *key* není algoritmu vůbec zpřístupněn. Uživatelský pohled na případy, vnořené tabulky a obsahové typy popisuje kapitola 4.2.1.

Tabulka 4.1: Podoba dat uložených v relační databázi.

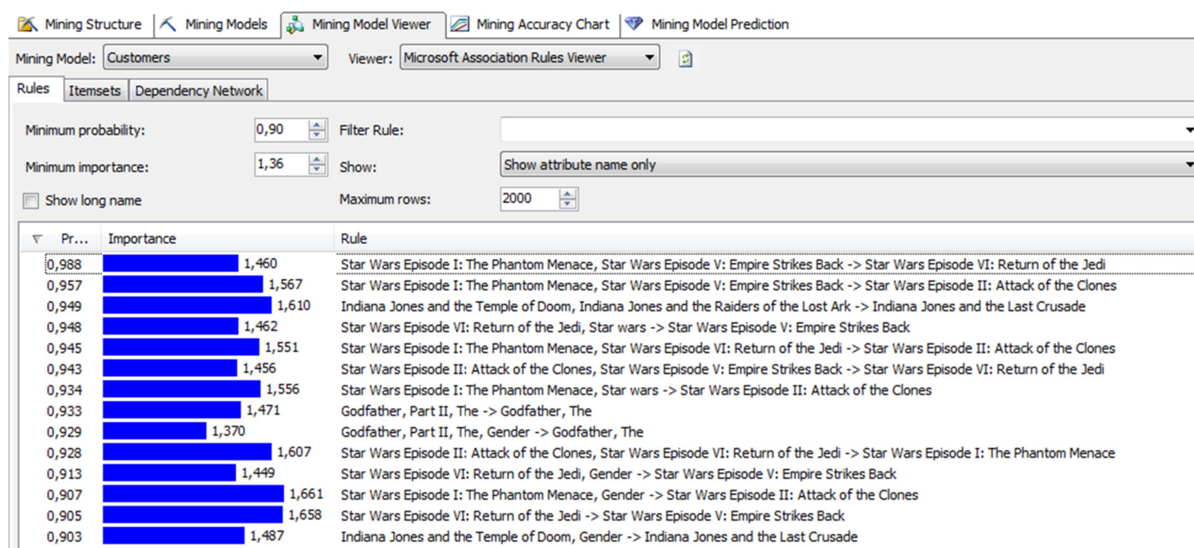
Sekvence [*key*]	Uživatel [*discrete*]	Událost [*sequence key*]	Položka [*discrete*]
1	Tom	1	A
1	Tom	1	B
1	Tom	2	B
2	Alice	1	A
2	Alice	1	C
2	Alice	2	B
2	Alice	2	A

Tabulka 4.2: Podoba dat, která je dostupná dolovacímu algoritmu.

<table>
<tr><td>Tom</td><td>1</td><td>A</td><td>1</td><td>B</td><td>2</td><td>B</td><td></td><td></td></tr>
<tr><td>Alice</td><td>1</td><td>A</td><td>1</td><td>C</td><td>2</td><td>B</td><td>2</td><td>A</td></tr>
</table>

Prezentace nalezených znalostí

Poslední nezbytnou součástí rozšíření je prezentace získaných znalostí. Pro tuto činnost je potřeba vytvořit třídu, která dědí od bázové třídy *AlgorithmNavigationBase*. SQL Server tuto třídu využívá pro postupný průchod přes všechny nalezené vzory. Třída musí, poskytnou detailní informace o jednotlivých vzorech, jako je podpora, spolehlivost, textová podoba vzoru, distribuce položek atd. Architektura systému vyžaduje, aby vzory byly serveru zpřístupněny ve stromové struktuře. Je také výhodné informace o nalezených vzorech podávat ve stejném formátu jako vestavěné dolovací algoritmy a to z důvodu možnosti využití zabudovaných prohlížečů výsledků.



Obrázek 4.5: Vestavěný prohlížeč firmy Microsoft pro zobrazení asociačních pravidel. Převzato z [27].

SQL Server nabízí možnost vytvoření vlastního prohlížeče výsledků. Vlastní prohlížeče lze navrhovat v prostředí .NET *WinForm* pomocí programovacího jazyku C#. Více informací o vytvoření navigátoru a tvorbě vlastních prohlížečů pro prezentaci znalostí lze nalézt v literatuře [27, 28]. Obrázek 4.3 ukazuje příklad prezentace vzorů, která se zobrazí koncovému uživateli v prostředí BIDS

v generickém prohlížeči výsledků. Obrázek 4.5 zobrazuje výsledky dolování asociačních pravidel v databázi videopůjčovny. Pro zobrazení asociací mezi výpůjčkami filmů byl využit zabudovaný prohlížeč asociačních pravidel.

PMML

Další variantou prezentace znalostí je využití *PMML* (*Predictive Model Markup Language*) dokumentů [31]. PMML je značkovací jazyk založený na XML, který se využívá pro sdílení vydolovaných modelů mezi různými nezávislými aplikacemi. PMML dokument se skládá nejméně ze dvou sekcí. První sekce popisuje informace o atributech dat a druhá sekce uchovává vydolované vzory nebo modely. SQL Server umožňuje načítání i generování PMML dokumentů ve verzi 2.1. Informace o atributech dat generuje samotný SQL Server. Programátor rozšiřujícího algoritmu může implementovat metodu *RenderPMMLContent()* pro generování nalezených znalostí a metodu *LoadPMMLContent()* pro načtení znalostí.

Pro práci s PMML dokumenty slouží nástroj *SQL Server Management Studio* (*SSMS*). Tento nástroj je součástí instalace SQL Serveru a je využívám pro správu databázových objektů. Uživatel se prvně pomocí SSMS připojí k analytickým službám, na kterých chce vytvořit dolovací model. Následně může vytvořit model z PMML dokumentu pomocí následujícího příkazu:

CREATE MINING MODEL <model> FROM PMML <xml string>;

Obsah PMML dokumentu musí být serveru předán ve formě textového řetězce. V případě, že uživatel dokončil dolování a chce nalezené vzory přenést na jiný systém, může využít následující příkaz pro export PMML dokumentu:

SELECT MODEL_PMML FROM <model>.PMML;

Příklad výsledné podoby PMML dokumentu pro dolování sekvenčních vzorů je dodán v příloze B.

4.3.2 GSP rozšíření

V rámci výzkumného projektu na VUT FIT bylo vytvořeno rozšíření SQL Serveru pro dolování sekvenčních vzorů metodou GSP. Projekt se zaměřuje na analýzu škodlivého kódu a jedním z typů analýzy je vyhledávání sekvenční vzory. GSP rozšíření implementovali M. Hlosta, J. Kupčík a M. Šebek. Mým úkolem je se na implementaci navázat a vytvořit efektivnější rozšíření SQL Serveru pro dolování sekvenčních vzorů. Více informací o algoritmu GSP popisuje kapitola 3.3.2. Tato kapitola popisuje vlastnosti existujícího GSP rozšíření.

Validace vstupních atributů

Algoritmus vyžaduje, aby uživatel během tvorby dolovací struktury kromě identifikátoru sekvence definoval také vnořenou tabulku a v té vybral sekvenční klíč a sloupec, který nese informace o položkách. Algoritmus provádí kontrolu předchozího požadavku v metodě *ValidateAttributeSet()* třídy *Metadata*. Pro rozlišení sloupce s položkami byl navržen příznak *MiningModelingFlags.KeyAttribute*. V situaci, kdy vstupní data obsahují více diskrétních atributů, musí uživatel využít tento příznak k označení sloupce s položkami.

Logování

Algoritmus byl dodán s logovací knihovnou. Ta umožňuje zápis logů do textových souborů. Při prvním zápisu knihovna automaticky vytvoří unikátní název logovacího souboru. Název je složen z názvu dolovacího algoritmu a unikátního čísla vytvořeného na základě aktuálního času. Text každého zápisu je automaticky obohacen o aktuální datum a čas.

Funkcionalita

Algoritmus umí dolovat zobecněné sekvenční vzory metodou GSP. Navíc implementuje metody pro uložení/načtení sekvenčních vzorů do/z databáze. Výsledné vzory loguje a ukládá do XML souboru pro další využití. Predikce budoucích hodnot nebyla implementována.

5 Návrh a implementace

Tato kapitola popisuje návrh datových struktur algoritmů a postup implementace rozšíření SQL Serveru pro dolování sekvenčních vzorů. V rámci práce byly implementovány algoritmy PrefixSpan, SPAM, LAPIN-SPAM a BIDE. Algoritmy byly vytvořeny v programovacím jazyce C# za pomocí vývojového prostředí Visual Studio 2010.

Implementace byla rozdělena na dvě části. V první části byla vytvořena nezávislá knihovna pro dolování sekvenčních vzorů. Tato knihovna neobsahuje žádné závislosti na SQL Serveru, a proto ji lze využít i v jiných aplikacích. Podkapitola 5.1 detailně popisuje datové struktury a implementaci výše zmíněných algoritmů v této knihovně.

Druhá část kapitoly popisuje postup při tvorbě rozšíření SQL Serveru. Jsou popsány implementace abstraktních tříd a metod, které jsou definovány v knihovně *DMPluginWrapper*. Třídy a metody z této knihovny jsou využívány SQL Serverem pro komunikaci s rozšiřujícími algoritmy.

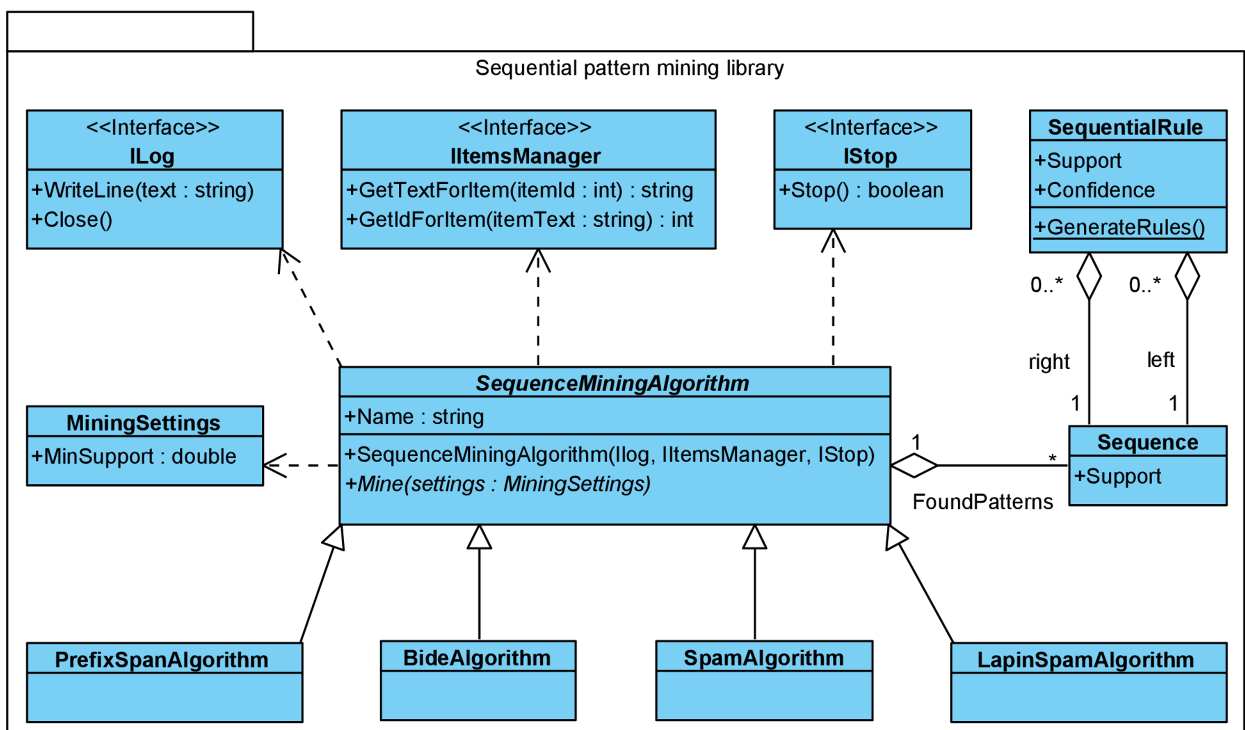
5.1 Knihovna pro dolování sekvenčních vzorů

Knihovna pro dolování sekvenčních vzorů byla pro větší využitelnost implementována nezávisle na platformě SQL Server. Nezávislá implementace přináší výhodu v tom, že knihovnu lze využít v jiných aplikacích a lze ji snadno testovat. Tento přístup však vyžaduje, aby knihovna neobsahovala žádné závislosti na aplikaci, ve které bude použita. Mezi nejčastější závislosti patří služby, které aplikace využívá pro logování, přístup do databáze, šifrování apod.

Pro odstranění závislostí jsem využil techniku *vkládání závislostí* (*dependency injection*) [29]. Tato technika vkládá závislosti mezi jednotlivými komponentami programu tak, aby jedna komponenta mohla používat druhou, aniž by na ni měla v době kompilace programu referenci. Toho se dosáhne tak, že konzument služby nezávisí na implementaci, tedy nevytváří konkrétní instanci služby, ale závisí na abstrakci, která mu bude dodána z vnějšku. Abstrakci lze definovat pomocí abstraktních tříd nebo rozhraní. Programovací jazyk C# umožňuje vytvářet rozhraní a proto jsem využil tento přístup.

Jelikož konzument nevytváří instanci služby, je potřeba mu konkrétní implementaci služby nějak dodat. Pro tuto činnost M. Fowler v [29] navrhl tři techniky a to *vkládání rozhraním* (*interface injection*), *setter metodou* (*setter injection*) a *injekci konstruktorem* (*constructor injection*). Pro dolovací knihovnu jsem využil injekci konstruktorem. Ta probíhá tak, že konzument služby v konstruktoru definuje své závislosti a tím vyžaduje, aby mu při konstrukci bylo předáno vše, co potřebuje ke svému korektnímu chování. Tato technika má výhodu v tom, že komponenta bude vždy sestavena se všemi potřebnými závislostmi, což nemusí být pravda u zbylých dvou technik. Knihovna pro dolování sekvenčních vzorů vyžaduje služby, které jsou definovány pomocí následujících rozhraní:

- **ILog** – Rozhraní definuje metody pro zápis do logovacího souboru a uzavření souboru. Pro účely testování knihovny jsem implementoval rozhraní pomocí objektu, který logy zapisuje do konzole. Na druhou stranu rozšíření SQL Serveru dodává knihovně objekt, který loguje data do textového souboru.
- **IItemsManager** – Dolovací algoritmy reprezentují každou položku v databázi pomocí celého čísla. To zajišťuje jednotnost a efektivnost řešení. Toto pravidlo není nijak omezující, protože položky vstupní databáze lze přetransformovat na celá čísla pomocí jednoduché mapovací funkce. Funkce může například vždy pro novou položku vracet vzrůstající hodnotu. Algoritmy pro hledání frekventovaných sekvencí postupně v procesu dolování vypisují nalezené vzory do logovacího souboru. Je však vhodné, aby zapisované vzory byly v původním textovém formátu. Pro tuto činnost slouží rozhraní *IItemsManager*, které definuje metody pro převod čísla položky na jeho textovou podobu a naopak.
- **IStop** – Toto rozhraní definuje metodu s názvem *Stop()*. Tato metoda je využita pro přerušení běhu algoritmu. V případech, kdy uživatel zadá velmi nízkou hodnotu minimální podpory, může algoritmus běžet velmi dlouho. Je vhodné mít možnost zastavit běh dlouho trvajících operací. Dolovací algoritmus během procesu hledání pravidelně volá metodu *Stop()* a zjišťuje tak, zdali běh algoritmu nebyl uživatelem přerušen.



Obrázek 5.1: Diagram tříd knihovny pro dolování sekvenčních vzorů.

Všechny implementované algoritmy pro hledání sekvenčních vzorů mají společný základ vlastností a funkcionality. Proto byla vytvořena bázová abstraktní třída *SequenceMiningAlgorithm*,

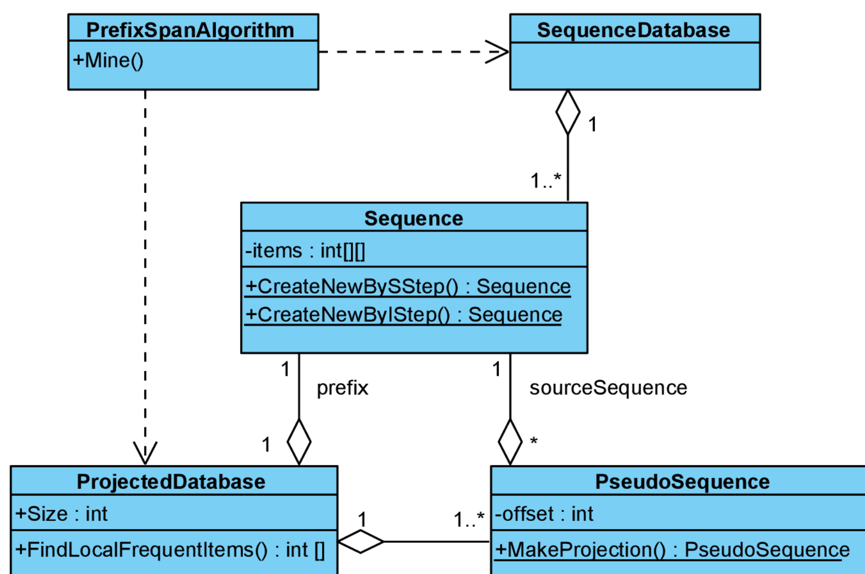
která implementuje všechnu společnou funkcionalitu. Třída se stará o zajištění všech závislostí pomocí injekce konstruktorem a definuje metodu *Mine()* pro spuštění dolování. Metoda pro spuštění dolování přebírá instanci třídy *MiningSettings*. Pomocí této třídy jsou algoritmům předány vstupní parametry, jako je minimální podpora a nastavení logování. Bázová třída zpracuje zadané vstupní parametry a poté volá abstraktní metodu pro dolování, kterou již musí implementovat každý algoritmus. Po dokončení běhu jsou klientovi přes vlastnost *FoundPatterns* třídy *SequenceMiningAlgorithm* zpřístupněny všechny nalezené sekvenční vzory. Obrázek 5.1 zobrazuje diagram tříd implementované knihovny pro dolování sekvenčních vzorů.

5.1.1 Implementace algoritmu PrefixSpan

Tato kapitola popisuje postup při implementaci algoritmu PrefixSpan, který slouží pro dolování sekvenčních vzorů metodou postupného vzrůstu. Nejprve jsou popsány datové struktury pro reprezentaci dat, následně jsou postupně rozebrány způsoby implementace jednotlivých operací algoritmu. Popis algoritmu PrefixSpan lze nalézt v kapitole 3.3.3.

Reprezentace dat

Algoritmus PrefixSpan využívá horizontální formát databáze sekvencí. Databáze, která se využívá pro hledání sekvenčních vzorů, je reprezentována třídou *SequenceDatabase*. Tato třída obsahuje neprázdný seznam sekvencí. Každá sekvence je složena z dvourozměrného pole celých čísel, kde první rozměr určuje událost a druhý rozměr zpřístupňuje položku v události. Index události definuje pořadí výskytu události v sekvenci. Tedy událost na indexu nula předchází událost na indexu jedna, ta předchází událost na indexu dva atd. V jedné události se každá položka vyskytuje maximálně jednou. Položky jsou v události lexikograficky uspořádány.



Obrázek 5.2: Diagram tříd algoritmu PrefixSpan.

PrefixSpan během dolování pro aktuálně vydolovaný sekvenční vzor vytváří projektovanou databázi a v té vyhledává lokálně frekventované položky. Nalezené položky využívá pro generování nových vzorů. Z důvodu menší spotřeby paměti a větší efektivitě jsem pro tvorbu projektovaných databází využil pseudo-projekci. To znamená, že při projekci sekvence není kopírován celý sufix, ale pouze se ukládá reference na původní sekvenci a offset, který určuje místo začátku projekce. Jedna projektovaná sekvence je reprezentovaná třídou *PseudoSequence*. Třída *ProjectedDatabase* představuje projektovanou databázi a je složena z prefixu projekce a seznamu projektovaných sekvencí. Popisované datové struktury zobrazuje diagram tříd na obrázku 5.2.

Dolování sekvenčních vzorů

V počáteční fázi dolování se průchodem vstupní databází vyhledají všechny frekventované položky. Tyto položky tvoří počáteční množinu sekvenčních vzorů a jsou využity pro vytvoření inicializačních projektovaných databází. V každé projektované databázi se poté vyhledají lokálně frekventované položky, které se postupně připojují k prefixu projekce a vytváří tak nové sekvenční vzory. Pomocí nalezených položek se opět vytváří projektované databáze. Tento cyklus se rekurzivně opakuje, jak ukazuje pseudokód algoritmu 3.2. Klíčovou roli v tomto rekurzivním procesu hrají metody pro vyhledání lokálních položek *ProjectedDatabase.FindLocalFrequentItems()* a vytvoření projekce *PseudoSequence.MakeProjection()*.

Metoda pro hledání lokálně frekventovaných položek v projektované databázi je časově nejnáročnější operací algoritmu PrefixSpan. Komplexnost metody spočívá v tom, že se musí v každé sekvenci projít každá událost, kde každá událost obsahuje pole položek. V rámci průchodu se také musí určit, zdali lze nalezenou položku připojit za poslední událost prefixu a vytvořit tak sekvenčně-rozšířenou sekvenci, nebo lze položku připojit do poslední události prefixu a sestavit tak množinově-rozšířenou sekvenci. Toto rozhodnutí záleží na poslední položce prefixu projektované databáze a na prohledávané sekvenci. Poslední položka mohla být k prefixu připojena dvěma způsoby, a to buď pomocí sekvenčního-rozšíření, značeno y, nebo pomocí množinového-rozšíření, značeno $(_y)$. Prohledávaná sekvence může nabývat tří tvarů: $< xyz >$, $< (xyz) >$, $< (_xyz) >$, kde y je jedna položka a proměnné x, z obsahují libovolný počet položek. Výraz x_i pak značí i-tou položku v kolekci x. Více informací o tvarech sufixu lze nalézt v definicích 3.1 a 3.3. Zařazení položky do dříve zmíněných dvou skupin je určeno na základě následujících pravidel:

1) y: $< xyz >$ => x_i, y, z_i,
2) y: $< (xyz) >$ => $x_i, y, z_i, (_z_i)$,
3) y: $< (_xyz) >$ => $(_x_i), (_y), (_z_i)$,
4) $(_y)$: $< xyz >$ => x_i, y, z_i,
5) $(_y)$: $< (xyz) >$ => x_i, y, z_i,
6) $(_y)$: $< (_z) >$ => $(_z_i)$.

První sloupce značí poslední položku prefixu, druhý sloupec určuje prohledávanou sekvenci a poslední sloupec definuje výsledná rozšíření. Tyto informace nejsou součástí článku o algoritmu PrefixSpan a jejich zjištění bylo nezbytnou znalostí pro správnou implementaci algoritmu.

Pro každou rozšiřující položku se zároveň počítá její podpora. Pro tuto činnost jsem využil generickou datovou strukturu *Dictionary<int, int>*. Tato struktura je součástí .NET frameworku a uchovává data ve tvaru klíč-hodnota. Jako klíč jsem využil samotnou položku a hodnota určuje podporu položky. Tuto strukturu jsem zvolil z důvodu její efektivnosti. Kolekce *Dictionary* je interně implementovaná pomocí hashovací tabulky. Časová složitost operace získání hodnoty dle zadaného klíče je konstantní.

Po vyhledání všech lokálních položek v projektované databázi se musí odstranit položky, které se nevyskytují v dostatečném počtu sekvencí a nesplňují tak uživatelem zadanou minimální podporu. Poté se každá frekventovaná položka α připojí k prefixu pomocí statické metody *Sequence.CreateNewBySStep()*, nebo *Sequence.CreateNewByIStep()*. Touto operací se vygeneruje nový sekvenční vzor, který se uloží do výsledné kolekce nalezených vzorů. Následně se na základě položky α vytvoří α-projektovaná databáze pomocí metody *ConstructProjectedDatabase()*. V této metodě se prochází aktuální projektovaná databáze a vyhledávají se sekvence, které obsahují položku α. Položky jsou v události lexikograficky setříděny, a proto jsem pro urychlení prohledávání implementoval metodu pro binární vyhledávání. Binární vyhledávání v seřazené posloupnosti má oproti sekvenčnímu logaritmickou časovou složitost. Obsahuje-li sekvence položku α, provede se projekce pomocí statické metody *PseudoSequence.MakeProjection()* a získaná sekvence se vloží do α-projektované databáze. Poté se doluje nad získanou α-projektovanou databází a celý proces se opakuje.

5.1.2 Implementace algoritmu SPAM

Tato kapitola popisuje postup při implementaci algoritmu SPAM. V kapitole jsou nejprve popsány datové struktury pro reprezentaci dat, následně jsou postupně rozebrány způsoby implementace jednotlivých operací algoritmu. Pro dosažení nejlepšího výkonu byl algoritmus implementován ve více iteracích. V každé iteraci jsem se zaměřoval na zefektivnění implementace z předchozí verze. V kapitole jsou popsány optimalizace, které byly provedeny pro větší efektivnost algoritmu. Detailní popis tohoto algoritmu SPAM lze nalézt v kapitole 3.3.4.

Reprezentace dat

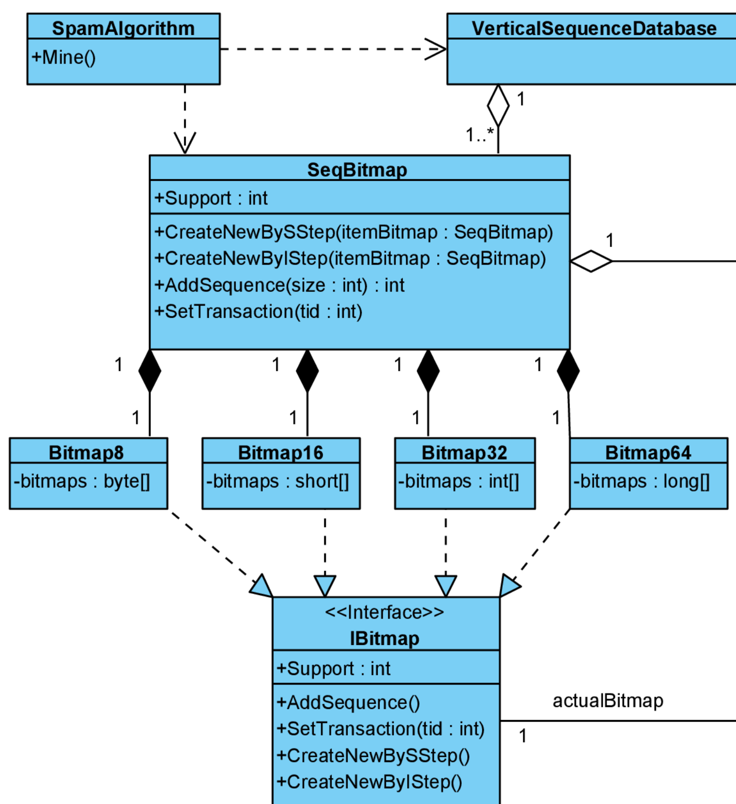
Efektivnost algoritmu SPAM je postavena na vhodné implementaci bitmap. Bitmapy se v algoritmu využívají pro určení výskytů položek v událostech sekvencí a jsou složeny z bitových vektorů.

V první implementaci jsem pro reprezentaci bitového vektoru využil třídu *BitArray* knihovny .NET. Třída *BitArray* obsahuje potřebné metody pro nastavení hodnoty bitu na zadané pozici a provedení logické operace AND s jinou instancí této třídy. Interně je tato struktura implementovaná pomocí pole celých čísel, kde velikost pole závisí na počtu bitů ve vektoru. Jedno celé číslo v poli se

skládá z 32 bitů. Každý bitový vektor tedy zabírá minimálně 4 bajty v paměti. Nastavení nebo získání hodnoty bitu na určité pozici se provádí pomocí operací bitový součet, součin a posun vlevo.

Tato implementace byla funkční a měla výhodu v tom, že algoritmus pracoval správně i s libovolně dlouhými sekvencemi. Referenční implementace algoritmu SPAM totiž podporuje sekvence s maximálně 64 událostmi, jak je popsáno v kapitole 3.3.4. Nevýhodou tohoto řešení byla paměťová a časová náročnost. Pro velké vstupní databáze se muselo na haldě alokovat obrovské množství objektů typu *BitArray*, přičemž alokace a dealokace objektů v řízeném kódu je poměrně netriviální úloha. Například pro databázi, která měla sto tisíc sekvencí a deset tisíc položek, se musela na haldě alokovat miliarda objektů typu *BitArray*.

Dále bylo časově velmi náročné počítat podporu vygenerovaných bitmap. Podpora se v případě bitmap počítá tak, že se zjišťuje, zdali jednotlivé vektory v bitmapě neobsahují všechny bity nastaveny na nulu. Třída *BitArray* však neobsahuje metodu pro rychlé zjištění, zdali jsou všechny bity vektoru nastaveny na nulu. Proto se tato operace musela provádět tak, že se postupně procházel každý bit ve vektoru a hledal se první bit nastaven na jedna. Časová asymptotická složitost této operace byla lineární, přestože lze tuto informaci získat v konstantním čase, jak je popsáno níže. Během následné iterace vývoje jsem se zaměřil na optimalizaci této implementace. Navrhl jsem několik řešení, která proces dolování urychlila až desetkrát. Přesto byl algoritmus SPAM ve většině experimentech pomalejší než dříve vytvořený algoritmus PrefixSpan. Proto jsem se rozhodl implementaci zcela přepracovat.



Obrázek 5.3: Diagram tříd algoritmu SPAM.

V druhé implementaci jsem se zaměřil na zefektivnění nejslabších částí z předešlé verze. Pro reprezentaci bitmap jsem navrhl struktury, které minimalizují počet vytvářených objektů na haldě a velmi efektivně počítají podporu. Diagram tříd algoritmu SPAM zobrazuje obrázek 5.3.

Bitmapa položek a sekvence je reprezentována třídou *SeqBitmap*. Tato třída je složena z dílčích tříd *Bitmap8*, *Bitmap16*, *Bitmap32* a *Bitmap64*. Každá tato část obsahuje bitové vektory pro sekvence s odpovídajícím počtem událostí. Například třída *Bitmap8* obsahuje všechny bitové vektory pro sekvence s maximální délkou osmi událostí. Dílčí části bitmapy pak reprezentují vektor pomocí nejvhodnějšího datového typu, čímž dochází k účinnějšímu využití paměti. Například třída *Bitmap8* využívá pro vektor osmibitový datový typ *byte*. Na druhou stranu třída *Bitmap16* využívá pro reprezentaci vektorů šestnáctibitový datový typ *short*.

Nevýhodou řešení je, že algoritmus neumí pracovat s libovolně dlouhými sekvencemi, což je avšak ve shodě s referenční implementací algoritmu SPAM. V programovacím jazyku C# je možné vytvořit vlastní datový typ a rozšířit tak současnou implementaci o podporu delších sekvencí. Například lze vytvořit datový typ *Int128*, který bude reprezentovat 128bitový vektor, pomocí dvou 64bitových čísel typu *long*.

Výhodou této implementace je, že se nevytváří obrovské množství objektů na haldě a neplýtvá se s pamětí. Datové typy *byte*, *short*, *int* a *long* jsou hodnotového typu a mohou být alokovány na zásobníku. Výpočet podpory pak probíhá tak, že se porovná hodnota vektoru s nulou. Oproti předešlému řešení má tato operace konstantní časovou složitost.

Načtení vertikální databáze

Před začátkem dolování se nejdříve musí vytvořit databáze sekvencí ve vertikálním formátu. Databáze v tomto formátu je implementována pomocí třídy *VerticalSequenceDatabase*. Třída uchovává pro každou položku v databázi bitmapu reprezentovanou třídou *SeqBitmap*. Pro načtení vstupní sekvence se musí zavolat metoda *SeqBitmap.AddSequence()*. Tato metoda přebírá na vstupu délku sekvence. Třída *SeqBitmap* se pak dle zadané délky rozhodne, na kterou dílčí část bitmapy deleguje volání metody *AddSequence()*. Je-li tedy délka sekvence například 14, třída *SeqBitmap* deleguje volání metody *AddSequence* na bitmapu *Bitmap16*. Po této delegaci si třída *SeqBitmap* uloží do proměnné *actualBitmap* referenci na použitou část bitmapy. Proměnná *actualBitmap* je typu *IBitmap*. Toto rozhraní definuje metody pro bitmapu a implementují jej všechny dílčí bitmapy, jak ukazuje diagram tříd na obrázku 5.3.

Po přidání sekvence do vertikální databáze se pro všechny položky v sekvenci volá metoda *SeqBitmap.SetTransaction()*. Pomocí této metody se nastavuje výskyt položky v jednotlivých událostech sekvence. Třída *SeqBitmap* deleguje toto volání na dílčí bitmapu, kterou má aktuálně uloženou v proměnné *actualBitmap*. Tento proces se opakuje pro každou vstupní sekvenci, čímž vytvoříme bitmapy pro všechny položky ve vstupní databázi.

Dolování sekvenčních vzorů

Dolování sekvenčních vzorů se provádí průchodem lexikografického sekvenčního stromu metodou do hloubky. V prvním kroku se z vertikální databáze získají bitmapy položek. Tyto bitmapy se pomocí S-kroku a I-kroku postupně spojují, čímž vytváří bitmapy kandidátních sekvencí. Pro každého kandidáta se musí vypočítat podpora a rozhodnout se, zdali je frekventovaný. Je-li kandidát frekventovaný, uloží se do kolekce nalezených sekvenčních vzorů a rekurzivně se doluje nad tímto vzorem. Pseudokód tohoto procesu je popsán v algoritmu 3.3. Více informací o S-kroku a I-kroku lze nalézt v kapitole 3.3.4.

S-krok, I-krok a výpočet podpory se provádí pomocí metod *SeqBitmap.CreateNewBySStep()* a *SeqBitmap.CreateNewByIStep()*. Metody na vstupu přijímají bitmapu připojované kandidátní položky. Připojení pak probíhá tak, že třída *SeqBitmap* volá metodu *CreateNewBySStep()*, respektive *CreateNewByIStep()* pro každou svou dílčí část. V S-kroku v dílčích bitmapách se postupně pro každý bitový vektor získá transformovaný vektor, který se pomocí operace logický součin spojí s vektorem připojované položky. Poté se provede test, zdali je výsledná hodnota vektoru různá od nuly. V případě, že je hodnota různá od nuly, inkrementuje se hodnota podpory nově vygenerované bitmapy. I-krok probíhá obdobně, jen se nemusí vytvářet transformovaný bitový vektor. Vzniklé dílčí části se využijí pro vytvoření bitmapy kandidátní sekvence. Více informací o důvodu vytváření transformovaných bitmap lze nalézt v kapitole 3.3.4.

Optimalizace řešení

I když byla tato implementace algoritmu SPAM dostatečně efektivní, během experimentů se ukázalo, že provedení S-kroku trvá poměrně déle než vykonání I-kroku. Proto jsem se zaměřil na optimalizaci této operace. Na základě profilace kódu jsem vypátral, že nejvíce času S-kroku zabere vytváření transformovaných bitmap.

Transformace se provádí tak, že se v každém bitovém vektoru hledá pozice prvního bitu nastaveného na jedna. Všechny bity ve vektoru následující za touto pozicí se pak musí nastavit na hodnotu jedna. Tato operace tedy vyžadovala průchod každým bitem ve vektoru. Jak jsem již zmínil, hodnota bitu se získává pomocí operací logický posun vlevo a bitový součin. Pro nastavení bitu se využívá operace logický posun vlevo a binární součet. Tyto operace jsou sice velmi rychlé, ale protože se provádí pro všechny bity v každé bitmapě, spotřebovávají mnoho procesorového času.

Optimalizaci tvorby transformované bitmapy jsem rozdělil na dva kroky. V první kroku jsem se zaměřil na efektivní zjištění pozice prvního bitu vektoru nastaveného na hodnotu jedna. Pro tuto činnost jsem využil *De Bruijnův* algoritmus [30]. Tento velmi rychlý algoritmus dokáže v konstantním čase za pomocí *De Bruijnovy sekvence* a binárních operací získat pozici nejnižšího nastaveného bitu v čísle. Algoritmus 5.1 zobrazuje pseudokód této metody pro 32bitové číslo. Před prvním voláním metody *GetFirstSetBit()* třídy *DeBruijn* se musí předpřipravit pole s De Bruijn sekvencí pomocí procedury *CreateDeBruijnSequence()*. Metoda *GetFirstSetBit()* pak postupně na zadané

číslo aplikuje tyto operace: dvojkový komplement, binární násobení, aritmetické násobení a binární posun vpravo. Výsledná hodnota se využije jako index do pole s De Bruijn sekvencí, čímž se obdrží pozice prvního nastaveného bitu ve vstupním čísle.

Algoritmus 5.1: De Bruijn algoritmus pro nalezení prvního nastaveného bitu v čísle.

Vstup:

```
bitVector: celé číslo reprezentující bitový vektor;
```

Výstup:

```
firstPosition: pozice prvního nastaveného bitu ve vektoru;
```

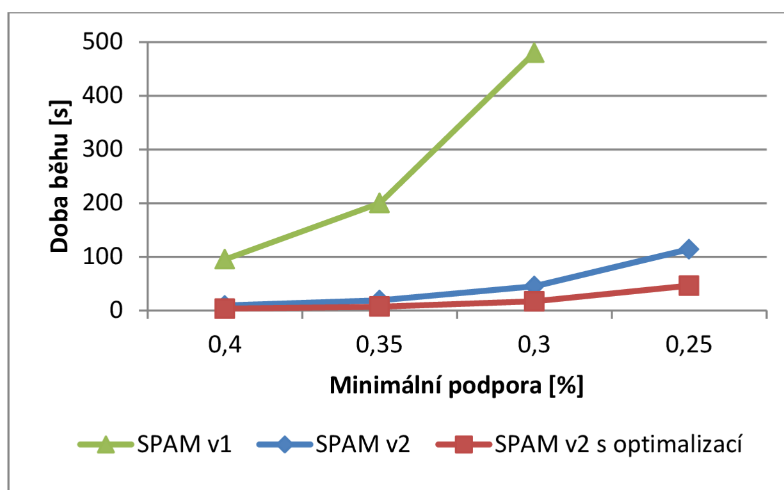
Metoda:

```
debruijn32Constant = 0x077CB531U;

procedure GetFirstSetBit (bitVector) do
  bitVector &= ~bitVector;
  bitVector *= debruijn32Constant;
  bitVector >>= 27;
  return deBruijnSequence32[bitVector];
end;

procedure CreateDeBruijnSequence() do
  for (i = 0; i < 32; i++) do
    deBruijnSequence32[(debruijn32Constant << i) >> 27] = i;
  end;
end;
```

Namísto vytváření transformovaných bitových vektorů za běhu jsem si připravil krátké pole s již transformovanými vektory. Velikost pole je rovna počtu bitů ve vektoru. Jako index do tohoto pole využiju hodnotu, kterou mi vrátí procedura *GetFirstSetBit()*. Díky této optimalizaci S-kroku se dolování metodou SPAM zrychlilo 2-3krát, jak lze vidět v grafu na obrázku 5.4. Pro lepší srovnání jednotlivých verzí byl do grafu zanesen také algoritmus SPAM v první verzi.



Obrázek 5.4: Výkon algoritmu SPAM v různých iteracích vývoje.

5.1.3 Implementace algoritmu LAPIN-SPAM

Tato kapitola popisuje implementaci algoritmu LAPIN-SPAM. Tento algoritmus je postaven na algoritmu SPAM, a proto byl navržen tak, aby využíval datové struktury tohoto algoritmu. LAPIN-SPAM se oproti SPAMu odlišuje ve způsobu získávání frekventovaných položek v S-kroku. Pro tuto činnost se využívá tabulka výskytů položek ITEM_IS_EXIST_TABLE. Tato kapitola se zabývá implementací této tabulky a také jejím využitím během dolování sekvenčních vzorů. Více informací o použitých datových strukturách lze nalézt v kapitole 5.1.2, kde je popsána implementace algoritmu SPAM.

Implementace tabulky výskytů položek

Tabulka výskytů položek ITEM_IS_EXIST_TABLE se využívá pro obdržení informace o tom, zdali se kandidátní položka vyskytuje v sekvenci za aktuální pozicí prefixu. Prefix je frekventovaná sekvence, která se generuje během průchodu lexikografickým sekvenčním stromem, jak je popsáno v kapitole 3.3.4. Postupně se tedy prochází každá vstupní sekvence, zjišťuje se pozice prefixu v dané sekvenci a inkrementuje se podpora všech kandidátních položek, které se vyskytují za prefixem. Tento proces je popsán pomocí algoritmu 3.4.

Hlavním požadavkem implementace tabulky ITEM_IS_EXIST_TABLE je, aby se získaná hodnota dala rovnou přičíst k podpoře kandidáta. V případě, že by obdržená hodnota byla například datového typu *bool*, muselo by se prvně provést porovnání a poté až podmíněná inkrementace podpory. Tato operace navíc by celý proces zpomalovala a výsledný čas by se přibližoval času algoritmu SPAM, kde se výpočet podpory provádí pomocí operací logický součin a porovnání.

Volba vhodného datového typu je důležitá nejen kvůli časové složitosti, ale také kvůli spotřebě paměti. Tabulka výskytů položek totiž uchovává informace o všech sekvencích, jejich událostech a výskytech všech položek v každé události. V případě, že by byla tabulka implementována pomocí trojrozměrného pole celých čísel, průměrná spotřeba paměti by byla rovna výrazu: počet sekvencí * průměrný počet událostí v sekvenci * počet položek * velikost celého čísla (4 bajty). Nejmenší datový typ v programovacím jazyce C#, jehož hodnotu lze přičíst k číslu je *byte*. V paměti zabírá jeden bajt, což je čtyřikrát méně než datový typ pro celé číslo *int*.

Tabulka výskytů položek je implementována třídou *ItemIsExistTable* a je složena z trojrozměrného pole datového typu *byte*. První dimenze určuje sekvence, druhá pak určuje události sekvence a třetí rozměr určuje položky. Tato implementace umožňuje efektivně počítat podporu a také snižuje spotřebu paměti.

Naplnění tabulky výskytů položek

Nezbytnou součástí algoritmu LAPIN-SPAM je správné vyplnění tabulky výskytů položek. Tato činnost se provádí na začátku procesu dolování z dat. Pro naplnění tabulky je využita vertikální databáze sekvencí implementovaná třídou *VerticalSequenceDatabase*. Tato třída, představená v imple-

mentaci algoritmu SPAM, poskytuje všechny potřebné informace pro naplnění tabulky výskytů položek.

Vyplnění tabulky je implementováno v metodě *ItemIsExistTable.FillTable()*. Zde se postupně prochází každá položka z vertikální databáze. Pro každou její vstupní sekvenci je potřeba získat pozici posledního výskytu položky v dané sekvenci. K této činnosti jsou využity bitové vektory z třídy *SeqBitmap*. Poslední nastavený bit ve vektoru udává pozici posledního výskytu položky v dané sekvenci a jeho pozici lze získat pomocí operace logický posun vpravo.

Po obdržení informace o pozici posledního výskytu položky v sekvenci se vyplní odpovídající informace ve třídě *ItemIsExistTable*. Pro konkrétní sekvenci a položku se nastaví všechny události s indexem menším než je zjištěná pozice na hodnotu jedna. Pozice, za kterými se položky nevyskytují, se nijak nenastavují, protože mají výchozí hodnotu nula. Během výpočtu podpory se přičtení hodnoty nula nijak neprojeví na výsledné podpoře kandidátní položky.

Dolování sekvenčních vzorů

Proces dolování sekvenčních vzorů je obdobný jako v případě algoritmu SPAM. Odlišuje se pouze v tom, jak se vyhledávají frekventované položky pro S-krok. Pseudokód této činnosti zobrazuje algoritmus 3.4.

Vyhledávání frekventovaných položek je implementováno metodou *GetFrequentItems()* třídy *ItemIsExistTable*. Tato metoda přebírá na vstupu bitmapu prefixu a seznam kandidátních položek. Pro každou vstupní sekvenci se nejdřív musí zjistit pozice prefixu v dané sekvenci. Tuto informaci lze zjistit pomocí bitového vektoru sekvence z bitmapy prefixu. Pozice prvně nastaveného bitu ve vektoru stanovuje pozici prefixu v sekvenci. Pro rychlé zjištění této pozice je využita efektivní metoda *GetFirstSetBit()* třídy *DeBruijn*, která vznikla během implementace algoritmu SPAM.

Nakonec se pro konkrétní sekvenci a aktuální pozici prefixu získá jednorozměrné pole z třídy *ItemIsExistTable*. Kandidátní položky se využijí pro indexaci tohoto pole a získané hodnoty se přičtou k podporám těchto položek.

Po průchodu všech sekvencí se z metody vrátí pouze položky, jejichž podpora je větší nebo rovna uživatelem zadané minimální podpoře.

5.1.4 Implementace algoritmu BIDE

Tato kapitola popisuje postup při implementaci algoritmu BIDE, který se využívá pro dolování uzavřených sekvenčních vzorů. Z důvodů, že BIDE prochází vstupní prostor a získává sekvenční vzory stejným způsobem jako algoritmus PrefixSpan, kapitola popisuje pouze implementaci nové metody pro kontrolu oboustranného rozšíření. Algoritmus BIDE využívá tuto metodu pro určení, zdali je nalezený sekvenční vzor uzavřený. Více informací o algoritmu lze nalézt v kapitole 3.3.6. Popis využitých datových struktur, které jsou společné s algoritmem PrefixSpan, lze nalézt v kapitole 5.1.1.

V článku o algoritmu BIDE [3] se autoři zaměřují na dolování sekvenčních vzorů s maximálně jednou položkou v události. My jsme však vyžadovali správnou funkčnost algoritmu pro obecně dlouhé události. Proto jsem implementaci algoritmu diskutoval s autorem doporučujícího systému zboží [20], který byl představen v kapitole 3.2. Autor M. Pitman totiž pro účely systému také implementoval algoritmus BIDE pro obecně dlouhé události.

Implementace kontroly oboustranného rozšíření

Algoritmus BIDE využívá pro dolování uzavřených sekvenčních vzorů datové struktury a funkce algoritmu PrefixSpan. Během procesu dolování navíc testuje, zdali je nalezený sekvenční vzor uzavřený. Pro test uzavřenosti byla vytvořena statická třída *ClosureChecker*.

Z důvodu, že je test uzavřenosti poměrně náročná operace, přeuspořádal jsem jednotlivé operace algoritmu BIDE, jinak než je popsáno v algoritmu 3.5. V pseudokódu algoritmu BIDE se prvně vyhledají lokálně frekventované položky. Z těch se následně vyberou položky, které mají stejnou podporu jako prefix projektované databáze. Tímto postupem se získají události dopředného rozšíření. Dále se vyhledávají události zpětného rozšíření. V případě, že nebyla nalezena událost dopředného ani zpětného rozšíření, je prefix prohlášen za uzavřený sekvenční vzor.

V mé implementaci algoritmu, ihned po nalezení první frekventované položky se stejnou podporou jako má prefix, přerušuji test uzavřenosti. Protože byla nalezena první rozšiřující událost, prefix není uzavřený a je zbytečné vyhledávat další dopředně a zpětně rozšiřující události.

V případech, kdy se nenalezne žádná dopředně rozšiřující událost, dochází k volání statické metody *ClosureChecker.BackwardExtensionCheck().* Tato metoda prohledává zadanou projektovanou databázi a vrací informaci o tom, zdali se v databázi vyskytuje nějaká zpětně rozšiřující událost.

V metodě *BackwardExtensionCheck()* se postupně pro každou událost e_i prefixu projektované databáze prohledávají maximální periody všech sekvencí. Zde se nejprve získá pozice LL_i. Tuto pozici získám tak, že procházím projektovanou sekvenci od konce a hledám první výskyt události e_i. Získaná pozice se navíc uloží pro příští běh, kdy se bude vyhledávat LL_{i-1}. V příštím běhu se nebude začínat prohledávat sekvence od konce, ale od dříve nalezené pozice LL_i.

Po zjištění LL_i zpětně procházím sekvenci od zjištěné pozice do prvního výskytu události e_i a ukládám si události zpětného rozšíření. Zpětná rozšíření jsou dvojího typu, kde první typ vytváří sekvenčně-rozšířenou sekvenci a druhý typ množinově-rozšířenou sekvenci. Pro množinové rozšíření se navíc při průchodu maximální periodou musí kontrolovat, zdali procházená událost je nadmnožinou události e_i.

V případě, že se najde zpětné rozšíření události e_i, které se vyskytuje ve všech maximálních periodách, je zjištěno, že vzor není uzavřený a kontrola uzavřenosti končí. V opačném případě, se prohledávají maximální periody události e_{i-1} a celý proces se opakuje.

Pro urychlení testu uzavřenosti jsem využil generickou datovou strukturu *HashSet<int>*. Tato struktura je součástí knihovny .NET a je optimalizovaná pro množinové operace, jako jsou sjednocení

a průnik. Během průchodu maximální periodou první sekvence si do struktury *HashSet<int>* přidávám nalezená rozšíření pomocí operace sjednocení. V následných maximálních periodách do struktury přidávám rozšíření pomocí operace průnik. Takto lze velmi rychle během průchodu maximálních period zjistit, že výsledná množina rozšíření je prázdná a nemá proto význam dále v průchodu pokračovat.

5.2 Implementace rozšíření SQL Serveru

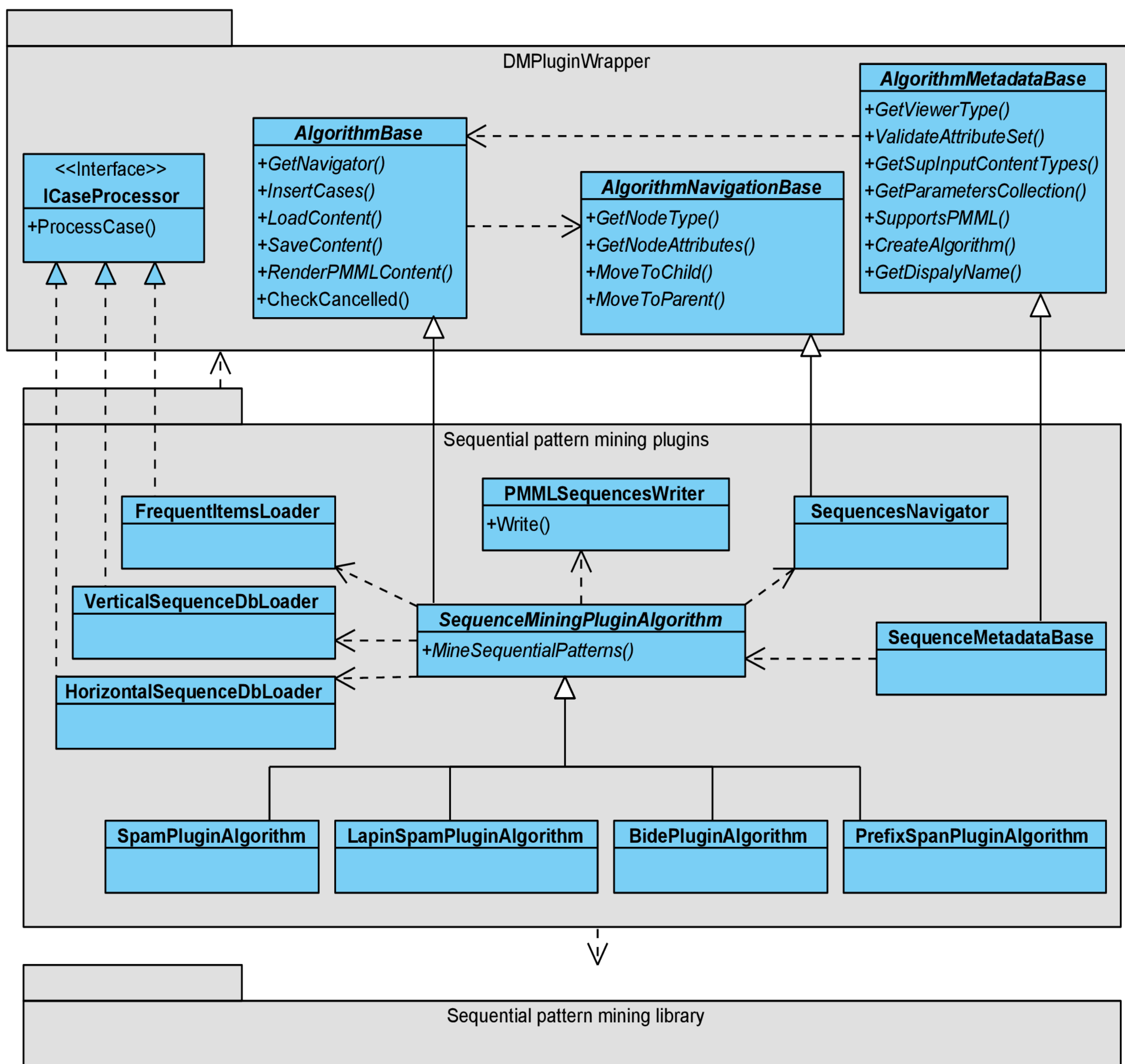
Tvorba rozšíření analytických služeb SQL Serveru v řízeném kódu je založena na implementaci abstraktních tříd a rozhraní, které jsou součásti knihovny *DMPluginWrapper*. Tato knihovna se stará o převod dat a volání funkcí mezi řízeným a neřízeným kódem. SQL Server pak využívá tuto knihovnu pro volání všech potřebných metod rozšiřujícího algoritmu, jako jsou metody pro dolování z dat, popis vlastností algoritmu atd. Obrázek 5.5 zobrazuje abstraktní třídy a rozhraní z knihovny *DMPluginWrapper* a jejich abstraktní metody, které musí každý rozšiřující algoritmus implementovat. Více informací o tvorbě rozšíření lze nalézt v kapitole 4.3.

Za účelem minimalizace tvorby duplicitního kódu jsem vytvořil společné třídy, které jsou využívány všemi rozšiřujícími algoritmy pro dolování sekvenčních vzorů. Obrázek 5.5 zobrazuje diagram těchto tříd a jejich vztahy ke komponentám v knihovně *DMPluginWrapper*. Rozšiřující algoritmy pro dolování sekvenčních vzorů využívají následující komponenty:

- ***SequenceMiningPluginAlgorithm*** – Hlavní třída knihovny, která implementuje společnou funkcionalitu všech rozšiřujících algoritmů. Tato abstraktní třída obsahuje metody pro zpracování vstupních parametrů, filtrování a třídění výsledných sekvenčních vzorů a jejich uložení do/z databáze. Také implementuje abstraktní metodu *InsertCases()* z třídy *AlgorithmBase* knihovny *DMPluginWrapper*. SQL Server volá metodu *InsertCases()* v případech, když uživatel přes Business Intelligence Development Studio (BIDS) spustí proces dolování z dat.

  V implementaci této metody se postupně načtou a zpracují parametry zadané uživatelem, vytvoří se soubor pro logování, zapíší se do něj statistické údaje o vstupních datech a nakonec se volá abstraktní metoda *MineSequentialPatterns()*. Tělo metody již závisí na typu algoritmu a proto ji musí každý rozšiřující algoritmus implementovat samostatně.

  Třída *SequenceMiningPluginAlgorithm* také implementuje rozhraní *IItemsManager* a *IStop*. Tyto rozhraní využívá knihovna pro dolování sekvenčních vzorů a byly popsány v kapitole 5.1. Pro kontrolu přerušení běhu algoritmu je poskytnuta SQL Serverem metoda *CheckCancelled()*. V případě, že uživatel přes BIDS přeruší proces dolování, metoda *CheckCancelled()* vyvolá výjimku, ta je odchycena a informace je dolovacímu algoritmu předána přes metodu *Stop()* rozhraní *IStop*.



Obrázek 5.5: Diagram tříd rozšíření SQL Serveru.

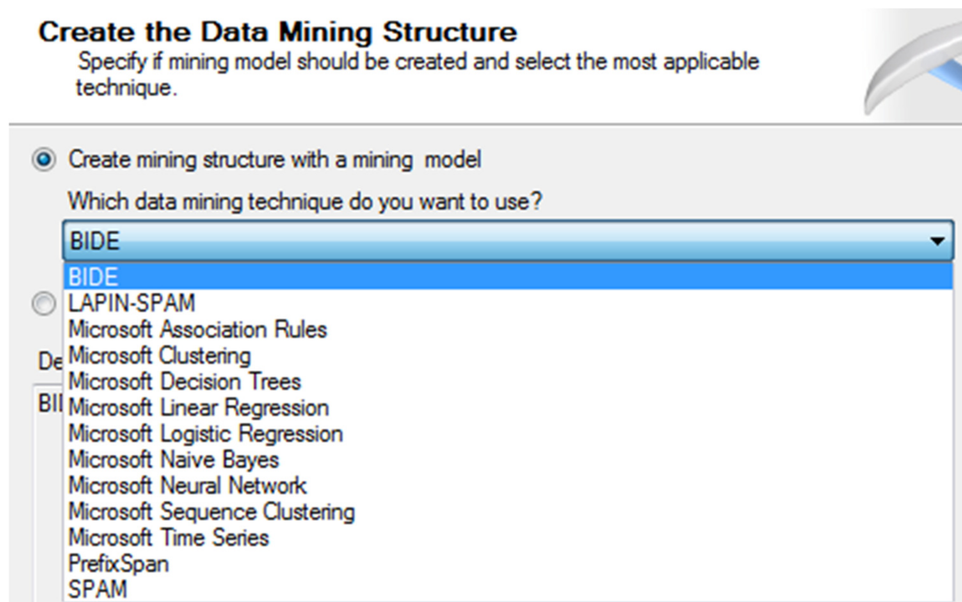
- ***SequenceMetadataBase*** – Třída, která se využívá pro popis vlastností rozšiřujících algoritmů. Mezi společné vlastnosti algoritmů pro dolování sekvenčních vzorů patří: seznam vstupních parametrů, jejich validace, podporované obsahové typy vstupních dat, prohlížeče výsledků a podpora PMML formátu.

  Třída se navíc musí postarat o vytvoření konkrétní instance algoritmu a také o zpřístupnění názvu a popisu algoritmu. Tyto operace jsou již závislé na konkrétním rozšiřujícím algoritmu. Proto je pro každý algoritmus vytvořena třída, která dědí od *SequenceMetadataBase* a zpřístupňuje dodatečné funkce rozšiřujícího algoritmu. Pro větší přehlednost však nejsou zakresleny do diagramu tříd na obrázku 5.5.

- ***FrequentItemsLoader*** – Tato třída se stará o načtení všech frekventovaných položek ze vstupní databáze. Třída implementuje rozhraní *ICaseProcessor*, které je součásti knihovny

DMPluginWrapper. SQL Server pro každý vstupní případ (case), v našem případě pro každou sekvenci, volá metodu tohoto rozhraní *ProcessCase()*. Třída prochází vstupní sekvenci a ukládá si informace o výskytech položek, čímž se současně počítá i jejich podpora. Po průchodu přes všechny vstupní sekvence třída vrací položky, které mají podporu větší nebo rovnou uživatelem zadané minimální podpoře.

- ***HorizontalSequenceDbLoader*** – Třída, která se využívá pro průchod přes vstupní data. Během průchodu daty vytváří databázi sekvencí v horizontálním formátu. Pro ukládání sekvencí využívá datové struktury, které byly představeny v implementaci algoritmu PrefixSpan v kapitole 5.1.1.
- ***VerticalSequenceDbLoader*** – Třída prochází vstupní data a vytváří databázi sekvencí ve vertikálním formátu. Pro uložení sekvencí využívá bitmapy, které byly představeny v implementaci algoritmu SPAM v kapitole 5.1.2.



Obrázek 5.6: Seznam algoritmů pro tvorbu dolovací struktury a modelu v BIDS.

Rozšiřující algoritmy SQL Serveru

Jednotlivá rozšíření SQL Serveru pro dolování sekvenčních vzorů, pomocí různých typů algoritmů, dědí od třídy *SequenceMiningPluginAlgorithm* a odlišně implementují metodu *MineSequentialPatterns()*. V těle této metody se pak využívají různé třídy pro načítání vstupních dat a odlišné třídy z knihovny pro dolování sekvenčních vzorů, která byla popsána v kapitole 5.1.

Například rozšíření pro dolování sekvenčních vzorů metodou PrefixSpan je implementováno třídou *PrefixSpanPluginAlgorithm*. V metodě *MineSequentialPatterns()* se nejprve využije třída *FrequentItemsLoader* pro načtení všech frekventovaných položek ze vstupních dat. Dále se vytvoří databáze sekvencí v horizontálním formátu pomocí třídy *HorizontalSequenceDbLoader*. Nakonec se

vytvoří instance třídy *PrefixSpanAlgorithm* z knihovny pro dolování sekvenčních vzorů a zavolá se metoda *Mine()*. Postup tvorby ostatních rozšíření je obdobný a jsou implementovaná pomocí následujících tříd: *SpamPluginAlgorithm*, *LapinSpamPluginAlgorithm* a *BidePluginAlgorithm*.

Po instalaci algoritmů do SQL Serveru uživatel může pro tvorbu dolovacích modelů využít, kromě vestavěných algoritmů i nové algoritmy pro dolování sekvenčních vzorů, jak ukazuje obrázek 5.6. Postup instalace vlastního rozšíření do SQL Serveru je dodán na přiloženém CD.

Generování sekvenčních pravidel

Mnou implementované rozšiřující algoritmy z nalezených sekvenčních vzorů generují sekvenční pravidla. Tato funkcionalita nebyla požadována v zadání práce, jedná se tedy o rozšíření zadání. Sekvenční pravidla je vhodné generovat, aby se získané znalosti daly využít i pro předpověď budoucích hodnot.

Sekvenční pravidlo je reprezentováno třídou *SequentialRule*. Tato třída je součástí knihovny pro dolování sekvenčních vzorů, jak ukazuje diagram tříd na obrázku 5.1. Třída je složena z dvou sekvencí, které reprezentují levou a pravou stranu pravidla. Statická metoda *SequentialRule.GenerateRules()* pak slouží pro generování sekvenčních pravidel z množiny zadaných vzorů. Pseudokód této metody zobrazuje algoritmus 3.6.

5.2.1 Prezentace sekvenčních vzorů a pravidel

O prezentaci sekvenčních vzorů se stará třída *SequencesNavigator*. Tato třída přebírá na vstupu seznam vydolovaných sekvenčních vzorů. Tyto vzory jsou postupně poskytovány SQL Serveru ve dvouúrovňové stromové struktuře. Kořen stromu obsahuje souhrnné informace o všech nalezených sekvenčních vzorech a pravidlech. Listové uzly stromu představují jednotlivé vydolované sekvenční vzory a pravidla. Mezi informace, které navigátor poskytuje, patří: textová podoba vzoru a jeho podpora, seznam všech položek, z kterých se vzor skládá, jejich podpora a pravděpodobnost výskytu. Prvně jsou zobrazeny všechny vzory a poté až sekvenční pravidla.

Průchod přes vzory a pravidla SQL Server provádí pomocí volání metod *MoveToChild()*, *MoveToParent()*, *GetNodeAttributes()* atd. Výslednou podobu dat, která se zobrazí uživateli v BIDS, zobrazuje obrázek 4.3.

Generování PMML dokumentů

Mnou implementovaná rozšíření SQL Serveru umí navíc prezentovat nalezené sekvenční vzory a pravidla ve formě PMML dokumentů verze 2.1. O tuto činnost se stará třída *PMMLSequencesWriter*.

Generování PMML dokumentů se provádí pomocí metody *RenderPMMLContent()*. Vstupním parametrem metody je objekt třídy *PMMLWriter*. Třída *PMMLWriter* je součástí knihovny *DMPluginWrapper* a je využita rozšiřujícími algoritmy pro zápis informací o nalezených sekvenčních

vzorech a pravidel ve formě XML fragmentů. Příklad výsledné podoby PMML dokumentu pro dolování sekvenčních vzorů je dodán v příloze B.

Tato funkcionalita není součástí zadání práce. Rozhodl jsem se však, že PMML dokumenty budu generovat pro účely přenosu vydolovaných znalostí mezi různými aplikacemi.

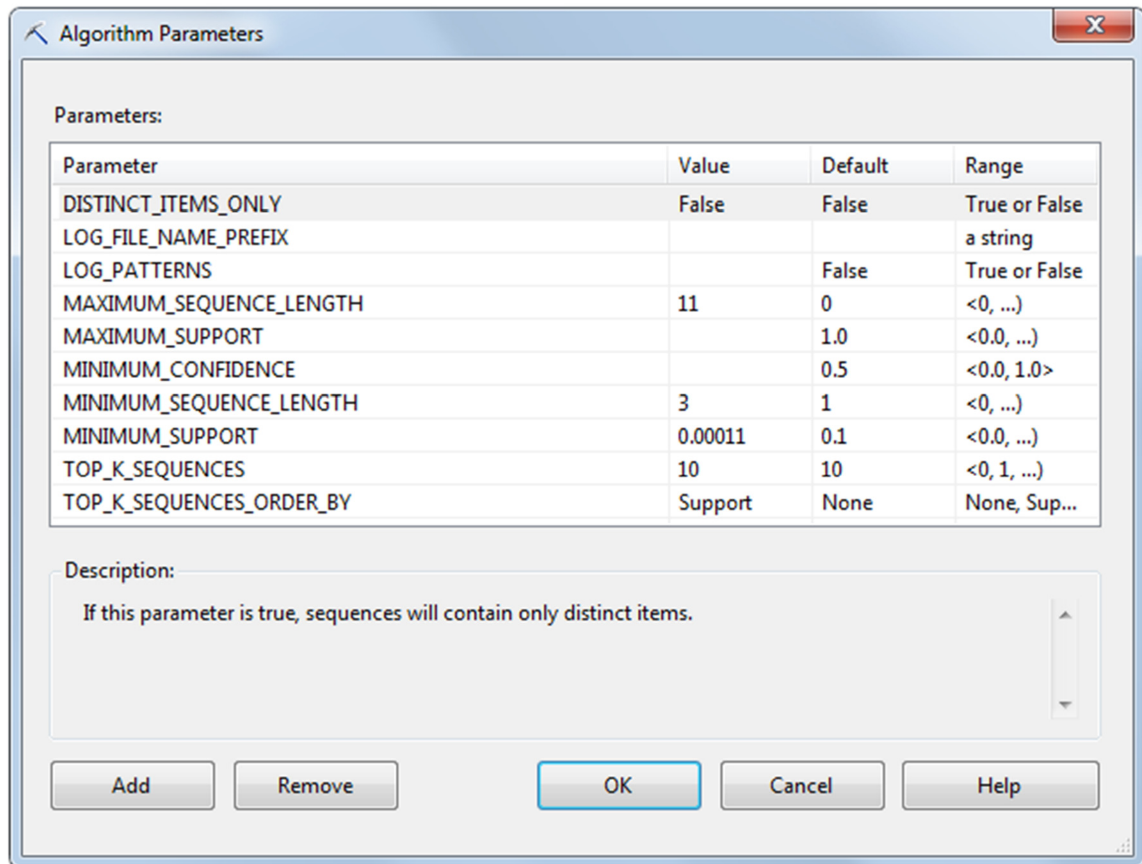
5.2.2 Parametry rozšiřujících algoritmů

Při zadání nízké hodnoty minimální podpory může být výsledný seznam nalezených vzorů velmi obsáhlý a tím i pro uživatele těžko čitelný. Proto jsem pro omezení vydolovaného seznamu sekvenčních vzorů do rozšiřujících algoritmů zavedl následující vstupní parametry:

- **MINIMUM_SUPPORT** – Parametr určuje minimální počet sekvencí, ve kterých se sekvenční vzor musí vyskytovat. Hodnotu lze zadat buď procentuálně, nebo absolutně.
- **MAXIMUM_SUPPORT** – Parametr určuje maximální počet sekvencí, ve kterých se vzor může vyskytovat. Využívá se především pro odfiltrování krátkých vzorů, které jsou obsaženy v delších sekvenčních vzorech.
- **MAXIMUM_SEQUENCE_LENGTH** – Maximální délka sekvenčního vzoru.
- **MINIMUM_SEQUENCE_LENGTH** – Minimální délka sekvenčního vzoru.
- **DISTINCT_ITEMS_ONLY** – V případě, že uživatel aktivuje tento parametr, bude při načítání vstupní databáze do sekvence vložen jen první výskyt položky. Ostatní výskyty položky v sekvenci se budou ignorovat.
- **TOP_K_SEQUENCES** – Zobrazí se jen prvních *k* nalezených sekvenčních vzorů. Výsledek je ovlivněn typem třídění nalezených vzorů.
- **TOP_K_SEQUENCES_ORDER_BY** – Parametr souvisí s předchozím parametrem a ovlivňuje způsob setřídění výsledných vzorů. Nalezené sekvenční vzory lze setřídit dle podpory, počtu události ve vzoru nebo dle počtu položek.
- **MINIMUM_CONFIDENCE** – Minimální spolehlivost sekvenčních pravidel.

Algoritmy z knihovny pro dolování sekvenčních vzorů umí pracovat pouze s minimální podporou. Proto se ostatní vstupní parametry aplikují na výslednou množinu vzorů až po dokončení dolování. Filtrace je provedena pomocí integrovaného dotazovacího jazyku LINQ [32], který je součástí .NET frameworku. Vstupní parametry lze zadat v BIDS přes nastavení algoritmu v dolovacím modelu, jak zobrazuje obrázek 5.7.

Seznam vstupních parametrů, jejich datové typy, rozsah hodnot a výchozí hodnoty jsou definovány ve třídě *SequenceMetadataBase*. SQL Server pro načtení vstupních parametrů volá metodu této třídy s názvem *GetParametersCollection()*.



Obrázek 5.7: Vstupní parametry rozšiřujících algoritmů pro dolování sekvenčních vzorů.

6 Experimenty

Tato kapitola se zabývá porovnáváním výkonu implementovaných algoritmů. Podkapitola 6.1 popisuje metody, které jsem využil pro ověření správné funkčnosti algoritmů. Podkapitola 6.2 popisuje provedené experimenty. V rámci experimentů byly mnou implementované algoritmy porovnány také s existující implementací algoritmu GSP. V poslední podkapitole jsou zhodnoceny výsledky experimentů.

6.1 Ověření funkčnosti algoritmů

Před samotným porovnáváním algoritmů jsem musel ověřit, zdali algoritmy pracují správně, tedy zdali vrací úplnou množinu sekvenčních vzorů. Pro prvotní ověření výsledků jsem využil existující GSP rozšíření, které je popsáno v kapitole 4.3.2. Toto rozšíření SQL Serveru ale není moc efektivní pro velké datové sady, a proto jsem musel najít i jinou alternativu. Navíc GSP neumí dolovat uzavřené sekvenční vzory, čímž jsem nemohl ověřit správnost implementace algoritmu BIDE.

Pro ověření správné funkčnosti algoritmů jsem si opatřil spustitelné soubory s referenční implementací algoritmů PrefixSpan a CloSpan z projektu *IlliMine* [33]. Tento open-source projekt poskytuje různé nástroje pro dolování z dat a strojové učení. Algoritmus CloSpan [16] doluje uzavřené sekvenční vzory a využil jsem jej pro ověření správné implementace algoritmu BIDE.

Pomocí generátoru jsem si vytvářel různě velké datové sady a ověřoval jsem výstupy mých a výše uvedených algoritmů. Po různých opravách algoritmů jsem docílil toho, že mnou implementované algoritmy vyhledávají stejné množiny vzorů jako referenční implementace.

6.2 Porovnání algoritmů

Experimenty byly prováděny na počítači Intel Core i3 540 @ 3.07GHz s 4 gigabajty paměti. Na počítači byl nainstalován 64bitový operační systém Windows 7 a SQL Server 2008 R2 enterprise edice. Pro testování algoritmů byly využity stejné reálné a syntetické datové sady jako ve vědeckých článcích, které se zabývají dolováním sekvenčních vzorů.

Pro generování syntetických databází byl využit IBM generátor [1, 4]. Podobu vygenerované databáze lze ovlivnit pomocí různých vstupních parametrů. Význam jednotlivých parametrů generátoru popisuje tabulka 6.1. Generátor ukládá výstupní databázi v binárním formátu. Pro převod binárního formátu do SQL skriptu jsem si vytvořil jednoduchou aplikaci.

Dobu běhu algoritmů jsem měřil pomocí třídy *Stopwatch* knihovny .NET. Tato třída vrací délku intervalu mezi voláním metod *Start()* a *Stop()*. Pro měření spotřeby paměti jsem využil třídu *GC* (garbage collector). Tato třída je součástí běhového prostředí .NET a stará se o správu objektů v

paměti. Navíc však obsahuje metodu *GetTotalMemory()* pro zjištění aktuálního stavu paměti. Během dolování jsem pak v pravidelných intervalech zjišťoval spotřebu paměti a ukládal si největší naměřenou hodnotu. Tato operace zpomaluje rychlost dolování, a proto jsem vždy prováděl buď test doby běhu algoritmu, nebo test spotřeby paměti.

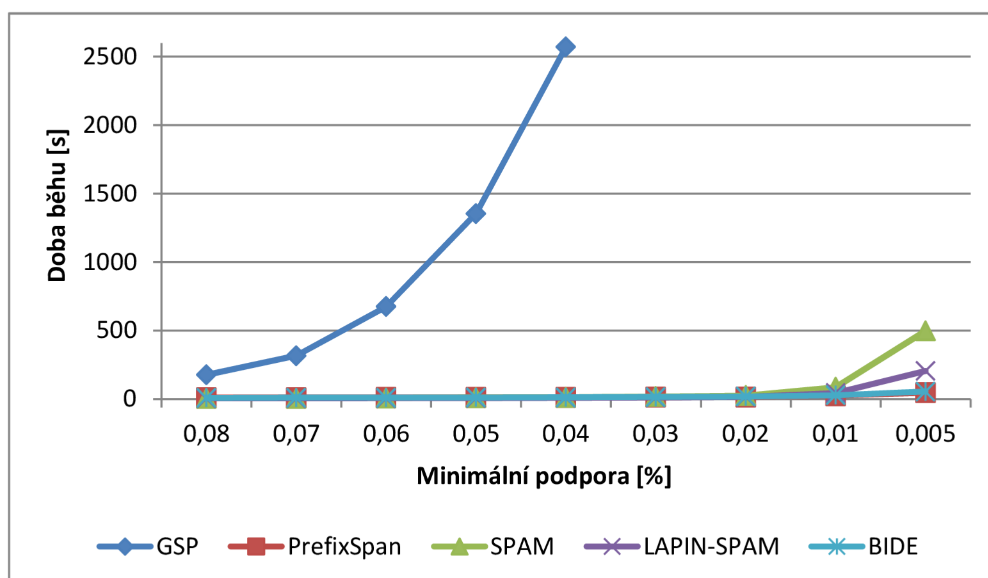
Tabulka 6.1: Parametry IBM generátoru pro tvorbu databází sekvencí.

Symbol	Význam
D	Počet sekvencí v databázi.
C	Průměrný počet událostí v sekvenci.
T	Průměrný počet položek v události.
S	Průměrná délka frekventovaných sekvencí.
I	Průměrná délka událostí ve frekventované sekvenci.
N	Počet unikátních položek v databázi.

6.2.1 Syntetická datová sada *D10KC8T8S8I8N1K*

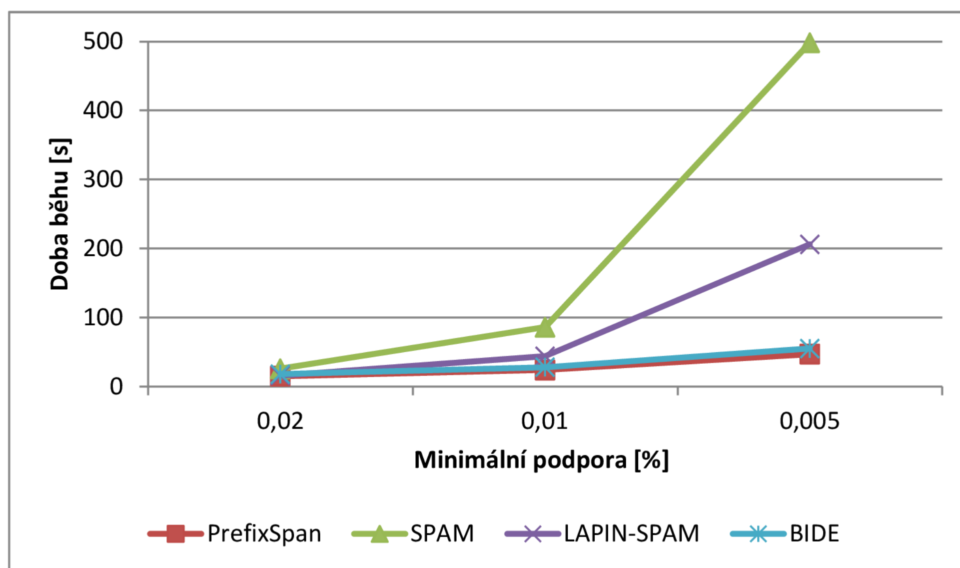
Pro první test algoritmů jsem využil datovou sadu *D10KC8T8S8I8N1K*. Tato synteticky generována databáze obsahuje deset tisíc sekvencí a tisíc položek. Průměrná délka sekvence, události a sekvenčního vzoru je osm. Obrázek 6.1 a obrázek 6.2 zobrazují grafy dob běhů všech implementovaných rozšíření pro různé hodnoty minimální podpory. Z grafu na obrázku 6.1 vyplývá, že algoritmus GSP je nesrovnatelně pomalejší než mnou implementovaná rozšíření. Doba běhu GSP algoritmu je i pro relativně vysokou hodnotu minimální podpory v řádech stovek až tisíců sekund. Zatímco algoritmy PrefixSpan, SPAM, LAPIN-SPAM a BIDE dokážou získávat sekvenční vzory v jednotkách sekund.

Nízký výkon rozšiřujícího algoritmu GSP je způsoben generováním obrovského množství kandidátních sekvencí, které se v databázi ani nevyskytují a vícenásobným průchodem vstupní databází. Pro detailnější analýzu ostatních algoritmů jsem výkon GSP dále do grafů nezakresloval.



Obrázek 6.1: Graf dob běhů rozšiřujících algoritmů na datové sadě *D10KC8T8S8I8N1K*.

Obrázek 6.2 zobrazuje detailnější pohled na výsledky měření dob běhů algoritmů na datové sadě *D10KC8T8S8I8N1K*. Nejvýkonnějším algoritmem pro dolování nad tímto typem dat je PrefixSpan. Algoritmus BIDE měl podobný výkon, ale z důvodu, že generovaná data neobsahují mnoho uzavřených sekvencí, test uzavřenosti byl většinou neúspěšný, což proces dolování zpomalovalo. Při zadání menších hodnot minimální podpory se zvětšoval počet frekventovaných položek v databázi, to zapříčinilo generování mnoho bitmap v algoritmu SPAM a jeho výkon se tím výrazně snižoval.



Obrázek 6.2: Graf dob běhů rozšiřujících algoritmů na datové sadě *D10KC8T8S8I8N1K*.

Během testování rozšíření SQL Serveru se ukázalo, že stejný test proběhne poprvé mnohem rychleji než v dalších opakováních. Proces dolování se postupně zpomaloval z jednotek sekund až na desítky minut. Tento problém byl diskutován na oficiálním fóru Microsoftu. Z různých vláken fóra [1] vyplynulo, že problém pravděpodobně způsobuje špatná správa paměti v knihovně *DMPluginWrapper*. Autoři analytických služeb doporučili psát rozšiřující algoritmy v neřízeném kódu pomocí COM objektů a vyhnout se tak využití knihovny *DMPluginWrapper*. Musel jsem tedy po každém testu SQL Server restartovat, abych mohl provádět plnohodnotné porovnávání.

Dále jsem pro účely automatizovaného a přesnějšího měření vytvořil konzolovou testovací aplikaci. Tato aplikace umí načíst synteticky generovanou databázi a provést dolování pomocí algoritmů z knihovny pro dolování sekvenčních vzorů.

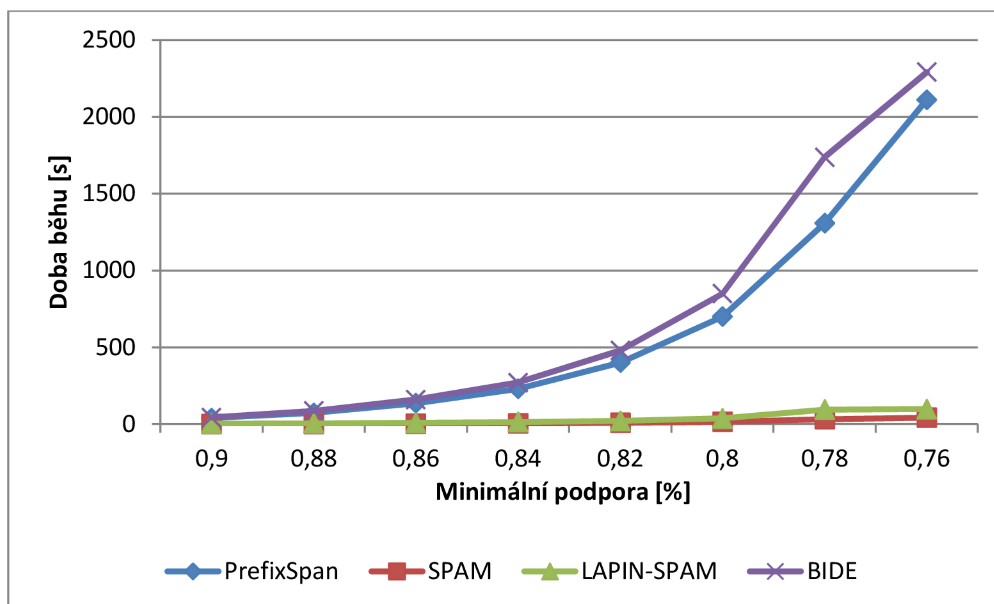
6.2.2 Syntetická datová sada *D50KC20T20S20I10N200*

Pro druhý test algoritmů jsem využil datovou sadu *D50KC20T20S20I10N200*. Tato synteticky generovaná databáze je mnohem větší než předchozí, obsahuje padesát tisíc sekvencí, kde každá sekvence obsahuje průměrně dvacet událostí. Databáze obsahuje pouze dvě stě položek. Jedná se o tzv. *hustá*

[1] http://social.msdn.microsoft.com/Forums/en/sqldatamining/thread/a483c39f-9ad9-45a4-838a-d14cb8431de3

(*dense*) data. Tato data jsou charakteristická tím, že obsahují relativně málo položek, ale dlouhé sekvence. Pod tuto skupinu dat spadají například DNA a proteinové sekvence.

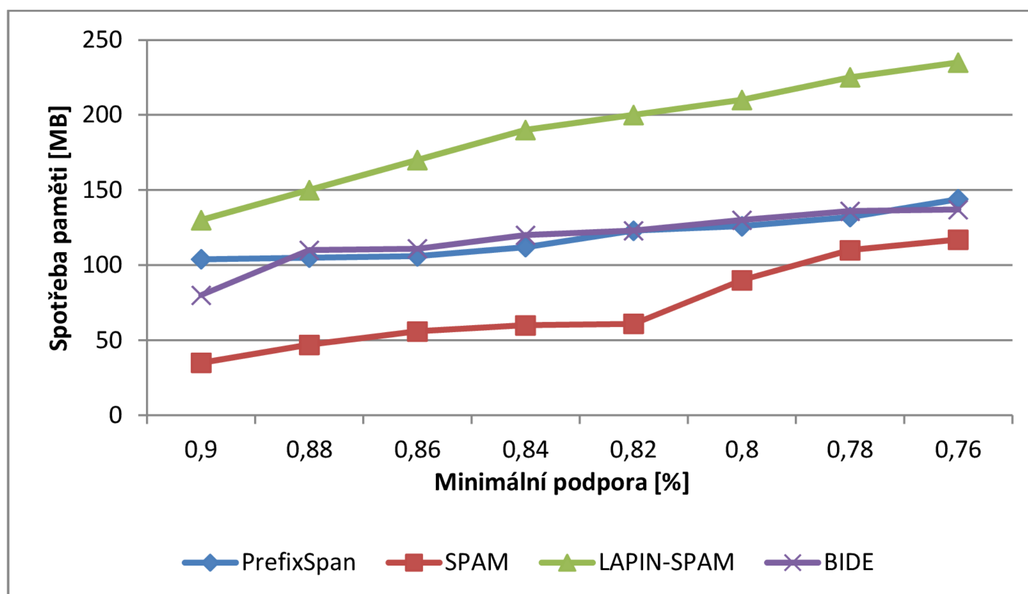
Pro tento typ dat se nejvíce hodí algoritmy SPAM a LAPIN-SPAM, jak lze vyčíst z grafu na obrázku 6.3. Vysoký výkon těchto algoritmů je způsobem tím, že data obsahují málo položek, což redukuje počet vytvářených bitmap kandidátů během procesu dolování. Dále se zde projevuje efektivní počítání podpory kandidátů. Pomocí rychlé operace logický součin nad bitmapou lze v konstantním čase určit, zdali se kandidátní položka v dlouhé sekvenci vyskytuje.



Obrázek 6.3: Graf dob běhů algoritmů na datové sadě *D50KC20T20S20I10N200*.

Experiment ukázal, že algoritmy PrefixSpan a BIDE jsou pro nízké hodnoty minimální podpory až o tři řády pomalejší než algoritmus SPAM. Algoritmus BIDE, který se momentálně považuje za jeden z nejrychlejších algoritmů, byl dokonce nejpomalejší. Důvodem je, že během dolování se vytváří obrovské množství projektovaných databází. Vstupní databáze však obsahuje málo položek a dlouhé sekvence, to způsobí, že sekvence v projektované databázi jsou jenom o kousek kratší. Pro získání lokálně frekventovaných položek se tedy prohledává velká část sekvence. BIDE navíc pro každý nalezený sekvenční vzor prochází i dlouhé maximální periody.

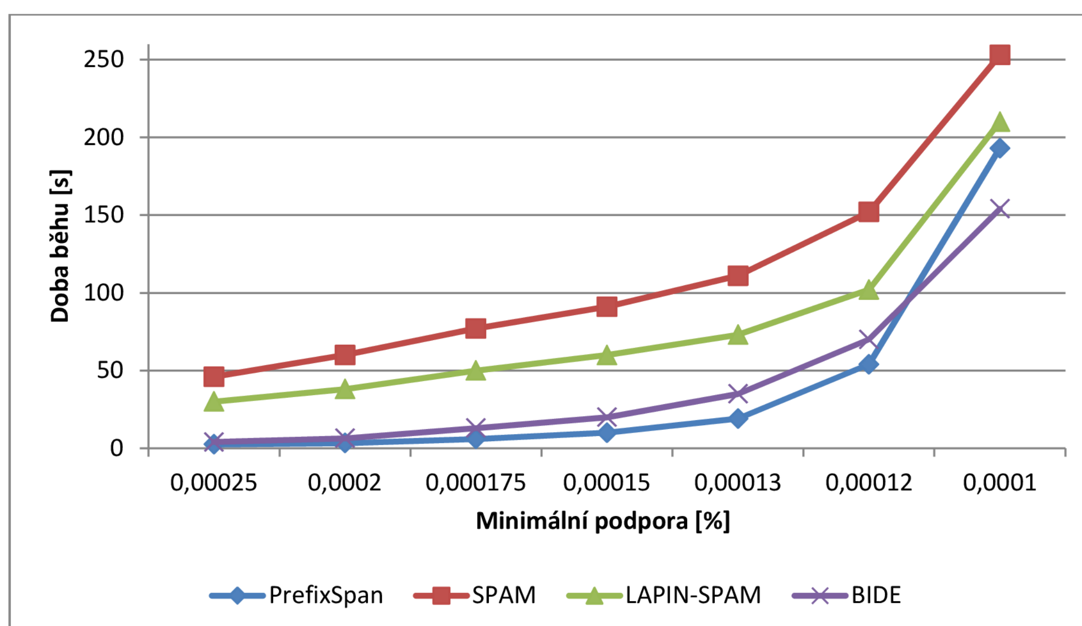
Obrázek 6.4 zobrazuje graf spotřeby paměti jednotlivých algoritmů během procesu dolování. Z grafu vyplývá, že pro hustá data spotřebovává nejméně paměti algoritmus SPAM. To je způsobeno generováním malého počtu kandidátů a k paměti šetrné implementaci bitmap. Nejvíce paměti spotřebovává algoritmus LAPIN-SPAM. U tohoto algoritmu je spotřeba paměti vždy největší, protože se kromě vertikální databáze v paměti udržuje tabulka s informacemi o výskytech všech položek ve všech událostech v databázi. Výsledky dalších testů nad různými synteticky generovanými databázemi lze nalézt v příloze A.



Obrázek 6.4: Graf spotřeby paměti algoritmů na datové sadě *D50KC20T20S20I10N200*.

6.2.3 Reálná datová sada *Gazelle*

Pro experiment nad reálnými daty jsem využil databázi *Gazelle* [2]. Databáze obsahuje záznamy o průchodech uživatelů webem společnosti Gazelle. Tato databáze byla poprvé využita v soutěži KDD-Cup 2000 a je nyní pravidelně využívána v článcích o algoritmech pro dolování sekvenčních vzorů. Detailní informace o této veřejně dostupné datové sadě lze nalézt v [34]. Databáze obsahuje 55 235 sekvencí a 1 436 unikátních položek. Jedná se o tzv. *řídká* (*sparse*) data. Tato data jsou charakteristická tím, že obsahují hodně frekventovaných položek a krátké sekvence.

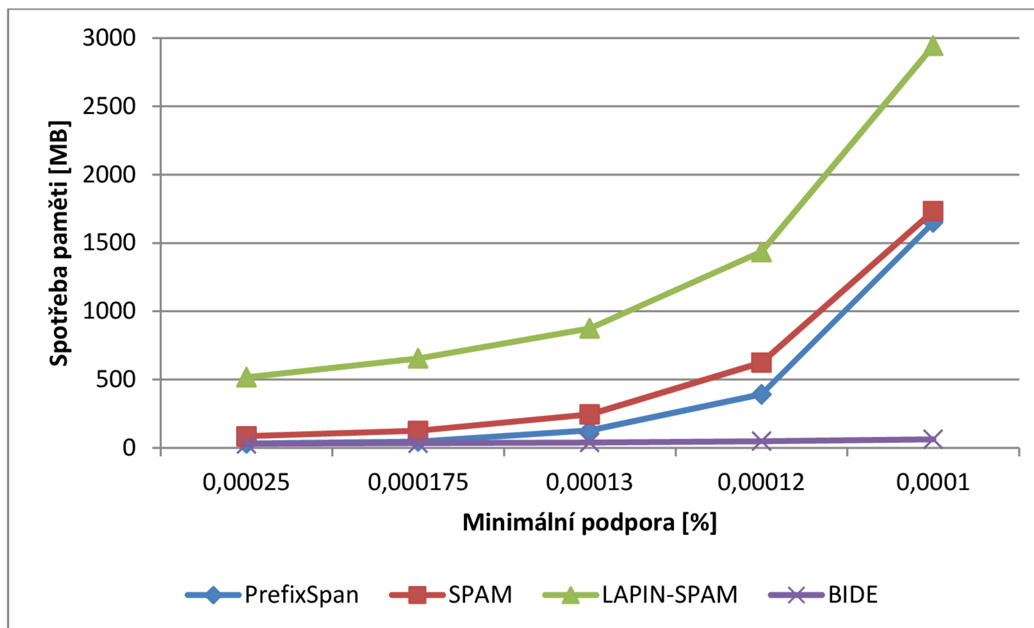


Obrázek 6.5: Graf dob běhů algoritmů na datové sadě *Gazelle*.

[2] http://www.sigkdd.org/kddcup/index.php?section=2000&method=data

Obrázek 6.5 zobrazuje graf dob běhů algoritmů na datové sadě Gazelle. Výsledky ukazují, že pro řídká data jsou vhodnější algoritmy PrefixSpan a BIDE. Datová sada Gazelle obsahuje velké množství uzavřených sekvencí, což je podmínkou aplikace efektivní metody pro ořezávání vstupního prostoru v algoritmu BIDE.

Algoritmus BIDE také spotřebovává velmi málo paměti. Spotřeba je dokonce téměř konstantní, jak lze vidět na obrázku 6.6. Naopak algoritmus LAPIN-SPAM spotřebovává mnohem více paměti než ostatní algoritmy.



Obrázek 6.6: Graf spotřeby paměti algoritmů na datové sadě *Gazelle*.

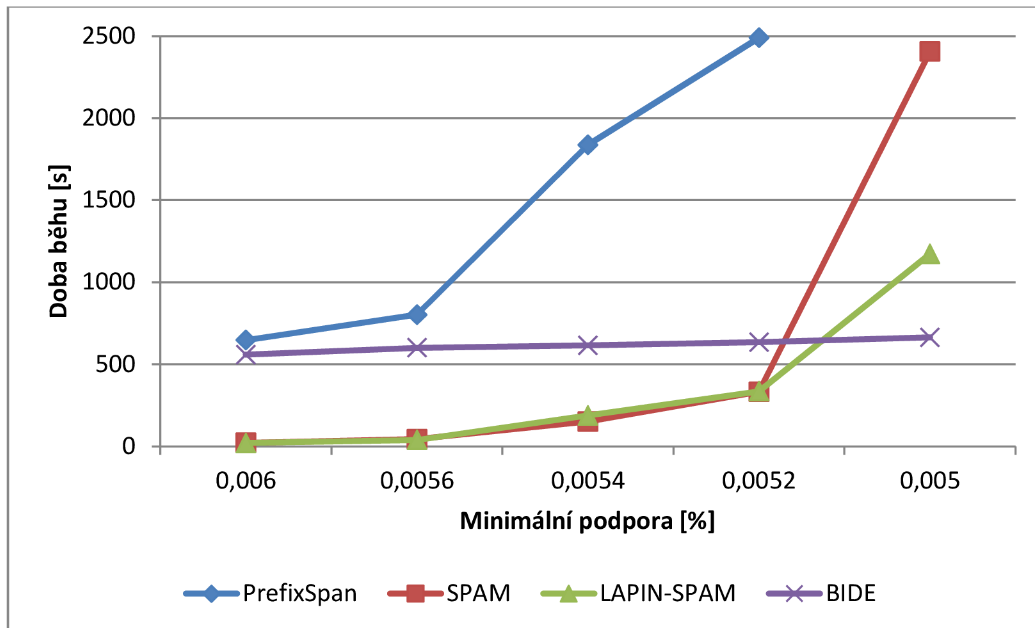
6.2.4 Reálná databáze virů

Poslední test algoritmů jsem prováděl na databázi virů. Tato obrovská databáze slouží pro účely výzkumného projektu, který probíhá na VUT FIT a zabývá se analýzou škodlivého kódu. Databáze obsahovala kolem 89 000 000 řádků. V databázi bylo přibližně 300 000 položek a 25 000 sekvencí. Sekvence reprezentuje vir a položky pak představují IP adresy sítí, z kterých vir pocházel. Test byl prováděn na počítači Intel Xeon X5660 @ 2.8GHz s 64 GB paměti.

Obrázek 6.7 zobrazuje výsledky experimentu nad databází virů. Experiment ukázal, že algoritmy dokážou i z tak velkých databází vyhledávat nové znalosti v desítkách až stovkách sekund.

Databáze obsahovala relativně málo frekventovaných položek a delší sekvence, proto byly nejvýkonnější algoritmy SPAM a LAPIN-SPAM. Při menších hodnotách minimální podpory počet frekventovaných položek vzrůstal a výkon algoritmu SPAM se prudce zhoršoval. Experiment také ukazuje, že pro větší počet frekventovaných položek je efektivnější počítat podporu kandidátů pomocí tabulky výskytů položek, jak to dělá LAPIN-SPAM, než pomocí binárního součinu bitmap, jak to provádí algoritmus SPAM.

Databáze obsahovala velké množství uzavřených sekvencí, což zapříčinilo, že pro velmi nízké hodnoty minimální podpory byl algoritmus BIDE nejvýkonnější. Díky vysokému počtu uzavřených sekvencí se velmi často aplikovala technika prořezávání vstupního prostoru a tím byl výkon algoritmu BIDE téměř konstantní.



Obrázek 6.7: Graf dob běhů algoritmů na databázi virů.

6.3 Vyhodnocení výsledků experimentů

Experimenty ukázaly, že každý algoritmus je vhodnější pro dolování v jiném typu dat. Algoritmy PrefixSpan a BIDE se hodí především pro dolování nad řídkými daty. Tedy takovými, která mají mnoho frekventovaných položek a kratší sekvence, jako jsou například webové logy. Algoritmus BIDE je velmi efektivní v případech, když data obsahují mnoho uzavřených sekvencí. V opačném případě je výkonnější algoritmus PrefixSpan. BIDE také spotřebovává povětšinu nejméně paměti ze všech implementovaných algoritmů.

Algoritmy LAPIN-SPAM a SPAM byly rychlejší při dolování nad hustými daty. Tedy takovými, která obsahují relativně málo frekventovaných položek a dlouhé sekvence, jako jsou DNA nebo proteinové sekvence. SPAM je efektivnější než LAPIN-SPAM, když data obsahují maximálně dvě stě frekventovaných položek. Naopak LAPIN-SPAM je vhodnější pro hustá data, která obsahují větší počet frekventovaných položek. Nejvíce paměti vždy spotřebovával algoritmus LAPIN-SPAM. Spotřeba byla 2-3krát větší než v případě algoritmu SPAM. Algoritmus GSP je nesrovnatelně pomalejší než mnou implementované algoritmy.

7 Návrh na další postup

Vizualizace sekvenčních vzorů

Nalezené sekvenční vzory se nyní uživateli zobrazují v textové podobě. Pro větší přehlednost by bylo vhodné vytvořit grafickou vizualizaci těchto vzorů. Microsoft SQL Server umožňuje vytvářet vlastní prohlížeče výsledků, které jsou uživatelům dostupné přes Business Intelligence Development Studio. Sekvenční vzor lze zobrazovat například ve formě acyklického grafu, kde každý uzel představuje jednu událost vzoru. Více informací o možnostech vizualizace sekvenčních vzorů lze nalézt v [35].

Zobecněné sekvenční vzory

Mnou implementované algoritmy umí dolovat jednoduché sekvenční vzory. V dalším postupu by mohla být do algoritmů přidána podpora dolování zobecněných sekvenčních vzorů. Zobecněné sekvenční vzory byly představeny v kapitole 3.3.2. Zobecnění spočívá v tom, že lze dolovat hierarchické, vícedimenzionální a časově omezené sekvenční vzory.

Přepis rozšíření SQL Serveru do neřízeného kódu

Během experimentů se ukázalo, že rozšíření SQL Serveru implementovaná v řízeném kódu nemají stabilní výkon. Doba běhů algoritmů se postupně prodlužuje z jednotek sekund až na desítky minut. Tento problém způsobuje knihovna *DMPluginWrapper*, která provádí převod dat a volání funkcí mezi řízeným a neřízeným kódem rozšiřujících algoritmů. Jediným řešením tohoto problému je po každém dolování SQL Server restartovat. Toto řešení, ale není uživatelsky přívětivé a často uživatel ani nemá dostatečná práva pro restart systému.

V dalším postupu by se tomuto problému dalo vyhnout tím, že by se mnou implementovaná rozšíření SQL Serveru přepsala do neřízeného kódu. V neřízeném kódu se využívají COM objekty a programovací jazyk C++. Pomocí této techniky jsou naprogramovány i vestavěné dolovací algoritmy SQL Serveru. Více informací o tvorbě rozšíření v neřízeném kódu lze nalézt v [36].

8 Závěr

Cílem této diplomové práce bylo rozšířit analytické služby MS SQL Serveru o algoritmy pro dolování sekvenčních vzorů. Sekvenční vzor je seřazená posloupnost událostí, která se v datech vyskytuje frekventovaně. Událost může být například bezpečnostní průnik nebo přístup na webovou stránku.

Po seznámení se s problematikou získávání znalostí z databází jsem se zaměřil na metody dolování sekvenčních vzorů. Zadané dolovací algoritmy PrefixSpan, SPAM, LAPIN-SPAM a BIDE jsem implementoval jako rozšíření analytických služeb MS SQL Serveru. Algoritmy jsem implementoval nezávisle na prostředí SQL Serveru, lze je tedy využít i v jiných aplikacích. Implementaci jsem prováděl ve více iteracích, přičemž jsem se v následných iteracích snažil o optimalizaci výkonu předešlých verzí. Vydolované sekvenční vzory prezentuji ve stejném formátu jako vestavěné algoritmy SQL Serveru. Uživatel tak může využít nástroj Business Intelligence Development Studio pro prohlížení získaných znalostí.

Nad rámec řešení jsem implementoval generování sekvenčních pravidel. Získané znalosti lze tak použít i pro předpověď budoucích hodnot. Dalším rozšířením zadání je export nalezených sekvenční vzorů do PMML dokumentů. Tuto funkcionalitu jsem se rozhodl přidat z důvodu, aby se získané znalosti daly sdílet mezi různými aplikacemi a zvýšila se tak využitelnost projektu.

Pro testování algoritmů byly využity stejné reálné a syntetické datové sady jako ve vědeckých článcích, které se zabývají dolováním sekvenčních vzorů. Ověřil jsem tak správnost implementace a očekávanou časovou složitost vytvořených algoritmů. Škálovatelnost algoritmů jsem ověřil na rozsáhlé databázi, která obsahovala informace o škodlivém kódu. Na základě výsledků experimentů jsem vyhodnotil typy dat, na které jsou jednotlivé algoritmy nejvhodnější.

V předposlední kapitole jsem představil a navrhnul změny pro další vývoj projektu. Závěrem lze říci, že tato diplomová práce splnila všechny požadavky zadání.

Literatura

[1] AGRAWAL, R. a R. SRIKANT. Mining Sequential Patterns. In *Proceedings of the 11th International Conference on Data Engineering*. Taipei, Taiwan: IEEE Computer Society, 1995. s. 3-14. ISBN 0-8186-6910-1.

[2] HAN, J. a M. KAMBER. *Data Mining: Concepts and Techniques*. Second Edition. Morgan Kaufmann Publishers, 2006, 770 s. ISBN 1-55860-901-6.

[3] WANG, J., J. HAN a C. LI. Frequent Closed Sequence Mining without Candidate Maintenance. *IEEE Transactions on Knowledge and Data Engineering*. vol. 19, no. 8, s. 1042-1056.

[4] AGRAWAL, R. a R. SRIKANT. Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*. London, UK: Springer-Verlag, 1996. s. 3-17. ISBN 3-540-61057-X.

[5] PEI, J., HAN, J., ASL, MOZARTAVI B., PINTO, H., CHEN, Q., DAYAL, U., HSU, M. C. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In *Proceedings of the 17th International Conference on Data Engineering*. IEEE Computer Society, 2001. s. 215-226.

[6] MITSA, Theophano. *Temporal data mining*. Boca Raton: CRC Press, 2010, 373 s., Chapman. ISBN 978-1-4200-8976-9.

[7] AYRES, J., FLANNICK, J., GEHRKE, J., YIU, T. Sequential PAttern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Alberta, Canada: Edmonton, 2002. s. 429-435.

[8] ZAKI, Mohammed J. SPADE: An Efficient Algorithm for Mining Frequent Sequences. In: *Machine Learning*. Volume: 42, Issue: 1. Hingham, MA, USA: Kluwer Academic Publishers, 2001, s. 31-60.

[9] YANG, Z. a M. KITSUREGAWA. LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern. In: *Proceedings of the 21st International Conference on Data Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2005. ISBN 0-7695-2657-8.

[10] YANG, Z., Y. WANG a M. KITSUREGAWA. LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. In: *Proceedings of the 12th international conference on Database systems for advanced applications*. Berlin, Heidelberg: Springer-Verlag, 2007, s. 1020-1023. ISBN 978-3-540-71702-7.

[11] AGRAWAL, R. a R. SRIKANT. Fast Algorithms for Mining Association Rules in Large Databases. In: *Proceedings of the 20th International Conference on Very Large Data Ba-*

ses. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, s. 487-499. ISBN 1-55860-153-8.

[12] MABROUKEH, Nizar R. a C. I. EZEIFE. A taxonomy of sequential pattern mining algorithms. In: *ACM Computing Surveys*: *2010*. Volume: 43, Issue: 1. New York, NY, USA: ACM, 2010, 3:1-3:41.

[13] ZENDULKA, J., BARTÍK, V., LUKÁŠ, R. a I. RUDOLFOVÁ. *Získávání znalostí z databází. Studijní opora*, 2006, Fakulta informačních technologií VUT v Brně.

[14] YANG, Z., Y. WANG a M. KITSUREGAWA. *An Effective System for Mining Web Log*. Springer, 2006, Volume: 3841 NCS, s. 40-52.

[15] HAN, J., J. PEI, B. MOZARTAVI, Q. CHEN, U. DAYAL a M. HSU. FreeSpan: frequent pattern-projected sequential pattern mining. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA: ACM, 2000, s. 355-359. ISBN 1-58113-233-6.

[16] YAN, X., J. HAN a R. AFSHAR. CloSpan: Mining closed sequential patterns in large datasets. In: *Proc of SIAM Int Conf on Data Mining*. MIT Press, 2003, s. 166-177.

[17] GUPTA, M. a J. HAN. Applications of Pattern Discovery Using Sequential Data Mining. In: *Pattern Discovery Using Sequence Data Mining: Applications and Studies*. IGI Global, 2011, s. 1-23. ISBN 9781613500569.

[18] GARBONI, C., F. MASSEGLIA a B. TROUSSE. Sequential Pattern Mining for Structure-Based XML Document Classification. In: *Advances in XML Information Retrieval and Evaluation, The Fourth International Workshop of the Initiative for the Evaluation of XML Retrieval*. Vol. 3977 / 2006. Schloss Dagstuhl, Germany: Springer, 2006.

[19] WUU, L., C. HUNG a S. CHEN. Building intrusion pattern miner for Snort network intrusion detection system. In: *Journal of Systems and Software*. Volume 80, Issue 10. New York, NY, USA: Elsevier Science Inc., 2007, s. 1699-1715.

[20] PITMAN, A. a M. ZANKER. An Empirical Study of Extracting Multidimensional Sequential Rules for Personalization and Recommendation in Online Commerce. In: *Wirtschaftinformatik Proceedings 2011*. 2011.

[21] BERENDT, B. a M. SPILIOPOULOU. Analysis of navigation behaviour in web sites integrating multiple information systems. In: *The VLDB Journal*. Volume 9, Issue 1. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2000, s. 56-75.

[22] BÜCHNER, A. a M. MULVENNA. Discovering Internet marketing intelligence through online analytical web usage mining. In: *ACM SIGMOD Record*. Volume 27, Issue 4. New York, NY, USA: ACM, 1998, s. 54-61.

[23] SPILIOPOULOU, M. Web usage mining for Web site evaluation. In: *Communications of the ACM*. Volume 43, Issue 8. New York, NY, USA: ACM, 2000, s. 127-134.

[24] HOTEK, Mike. *Microsoft SQL Server 2008: Krok za krokem*. Brno: Computer Press, a. s., 2009. ISBN 978-80-251-2466-6.

[25] LACKO, Luboslav. *Business Intelligence v SQL Serveru 2008: Reportovací, analytické a další datové služby*. Brno: Computer Press, a.s., 2009. ISBN 978-80-251-2887-9.

[26] MICROSOFT CORPORATION. *Přehled produktu SQL Server 2005* [online]. 2005-11-07 [cit. 2012-01-06]. Dostupné z URL: http://www.microsoft.com/cze/windowsserversystem/sql/prodinfo/overview/default.mspx

[27] MACLENNAN, J., Z. TANG a B. CRIVAT. *Data Mining with Microsoft SQL Server 2008*. Indianapolis, Indiana: Willey Publishing, 2008. ISBN 978-0470277744.

[28] MICROSOFT CORPORATION. *Creating Plug-in Algorithms for SQL Server 2005 Data Mining* [online]. 2008-08-05 [cit. 2012-01-06]. Dostupné z URL: http://www.sqlserverdatamining.com/ssdm/Default.aspx?tabid=94&Id=163.

[29] FOWLER, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003, 533 s. ISBN 03-211-2742-0.

[30] LEISERSON, Charles, Harald PROKOP a Keith RANDAL. *Using de Bruijn Sequences to Index a 1 in a Computer Word*. 1998.
Dostupné z URL: http://supertech.csail.mit.edu/papers/debruijn.pdf.

[31] Predictive Model Markup Language. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2012-02-25 [cit. 2012-04-21].
Dostupné z URL: http://en.wikipedia.org/wiki/Predictive_Model_Markup_Language.

[32] MICROSOFT CORPORATION. LINQ (Language-Integrated Query). In: *MSDN Library* [online]. 2012 [cit. 2012-05-06].
Dostupné z URL: http://msdn.microsoft.com/en-us/library/bb397926.aspx.

[33] HAN, J. a DATA MINING RESEARCH GROUP. *IlliMine* [online]. 2005, 2006-03-01 [cit. 2012-05-07]. Dostupné z URL: http://illimine.cs.uiuc.edu/.

[34] KOHAVI, R., C. BRODLEY, B. FRASCA, L. MASON a Z. ZHENG. KDD-Cup 2000 organizers' report: peeling the onion. In: *Proceedings of the SIGKDD Explorations*. New York, NY, USA: ACM, 2000, s. 86-93. ISSN 1931-0145.

[35] ANDERSCH, Christian. *Graphical Presentation of Sequential Patterns*. Böblingen, 2006. Diplomová práce. Hochschule Wismar.

[36] MICROSOFT CORPORATION. SQL Server Data Mining: Plug-In Algorithms. In: *MSDN Library* [online]. 2005-05-01 [cit. 2012-05-15].
Dostupné z URL: http://msdn.microsoft.com/en-US/library/ms345133.

Seznam příloh

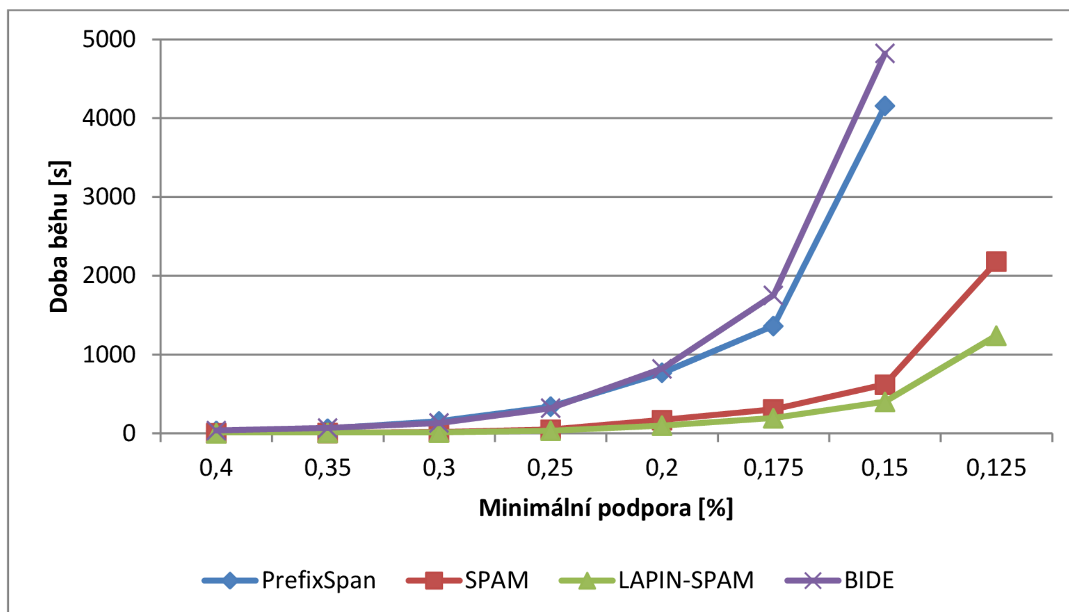
A. Výsledky porovnání algoritmů

B. Ukázka PMML dokumentu

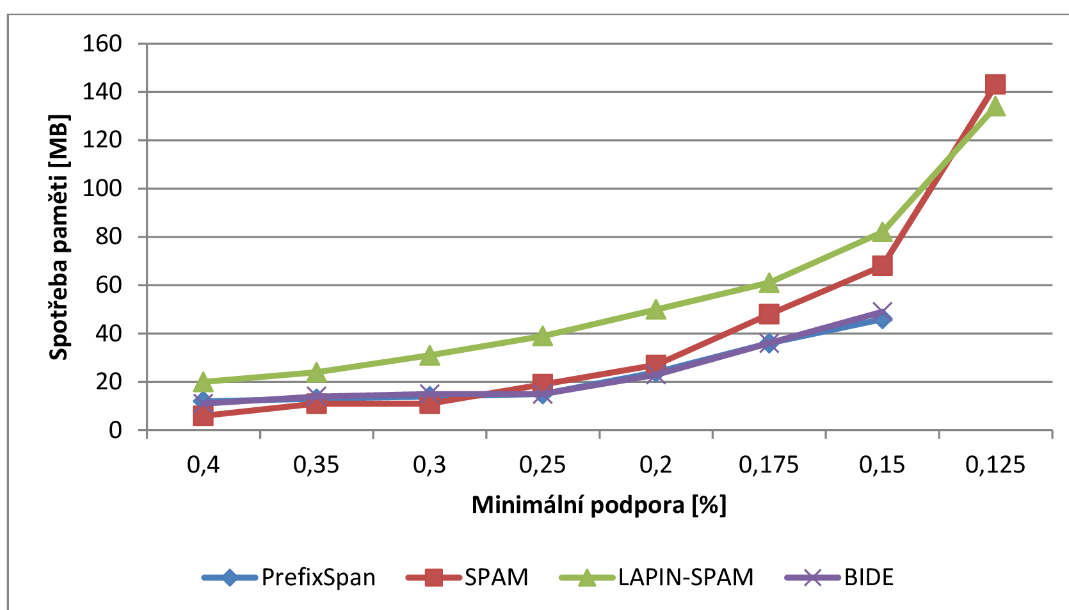
C. Obsah přiloženého CD

Příloha A: Výsledky porovnání algoritmů

Syntetická datová sada *D5KC20T20S20N1K*

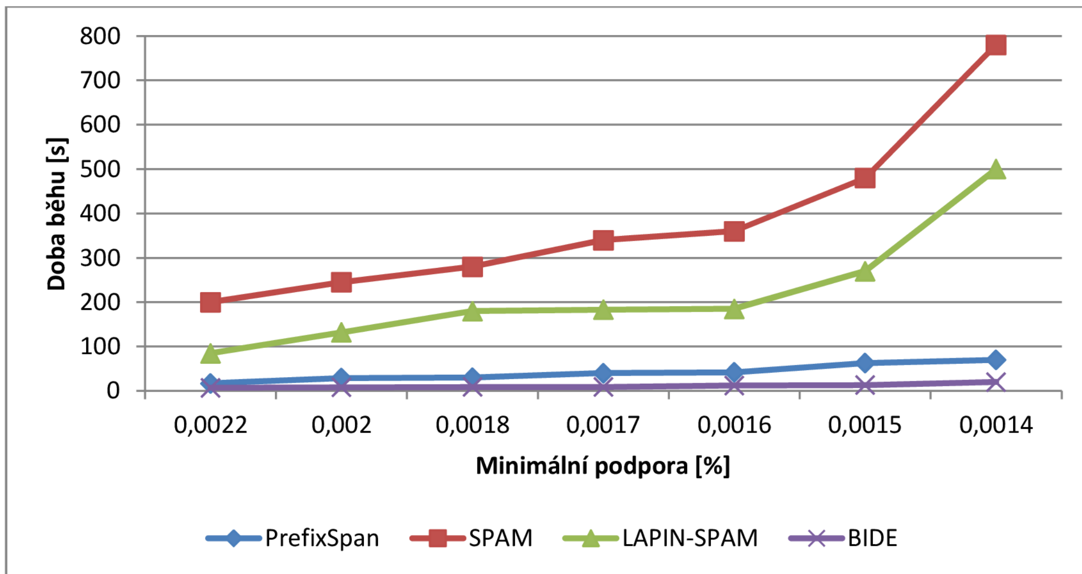


Obrázek A: Graf dob běhů algoritmů na datové sadě *D5KC20T20S20N1K*.

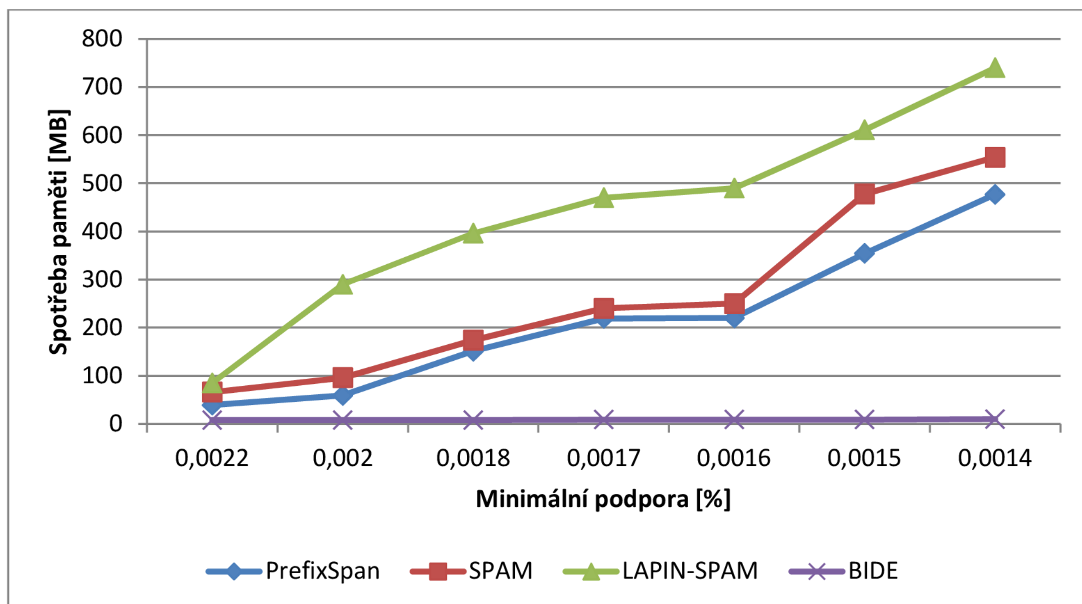


Obrázek B: Graf spotřeby paměti algoritmů na datové sadě *D5KC20T20S20N1K*.

Syntetická datová sada *D10KC10T2.5N10KS6I2.5*



Obrázek C: Graf dob běhů algoritmů na datové sadě *D10KC10T2.5N10KS6I2.5*.



Obrázek D: Graf spotřeby paměti algoritmů na datové sadě *D10KC10T2.5N10KS6I2.5*.

Příloha B: Ukázka PMML dokumentu

```
<PMML version="2.1" xmlns="http://www.dmg.org/PMML-2_1"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Header copyright="Copyright (c) 2003 Microsoft Corporation, All Rights
    Reserved">
    <Application name="Microsoft Analysis Services" version="9.0"/>
  </Header>

  <DataDictionary numberOfFields="2">
    <Extension extender="Microsoft Analysis Services"
      name="NestedTablesDefinition">
      <NestedTable name="SequenceDb">
        <NestedKey name="Tid" type="integer"/>
      </NestedTable>
    </Extension>
    <DataField name="Tid" displayName="Tid" optype="continuous"
      isCyclic="0" dataType="integer">
      <Extension extender="Microsoft Analysis Services"
        name="NestedAttributeExtension">
        <compound-attribute name="SequenceDb" value="Tid">
          <key-val name="Tid" value="Tid"/>
        </compound-attribute>
      </Extension>
    </DataField>
    <DataField name="Item" displayName="Item" optype="categorical"
      isCyclic="0" dataType="string">
      <Extension extender="Microsoft Analysis Services"
        name="NestedAttributeExtension">
        <compound-attribute name="SequenceDb" value="Item">
          <key-val name="Tid" value="Item"/>
        </compound-attribute>
      </Extension>
    </DataField>
  </DataDictionary>

  <SequenceModel modelName="PrefixSpan" functionName="sequences"
    numberOfSequences="4" numberOfSets="4"  numberOfItems="3"
    minimumSupport="2" minimumConfidence="0,75" numberOfRules="1">

    <SetPredicate id="sp1" field="transaction" operator="supersetOf">
      <Array n="1" type="string"> 'laptop' </Array>
    </SetPredicate>
    <SetPredicate id="sp2" field="transaction" operator="supersetOf">
      <Array n="1" type="string"> 'kamera' </Array>
    </SetPredicate>
    <SetPredicate id="sp3" field="transaction" operator="supersetOf">
      <Array n="1" type="string"> 'tiskarna' </Array>
    </SetPredicate>
    <SetPredicate id="sp4" field="transaction" operator="supersetOf">
      <Array n="2" type="string"> 'laptop' 'kamera' </Array>
    </SetPredicate>

    <Sequence id="seq1" numberOfSets="1" occurrence="4" support="1,0">
      <SetReference setId="sp1"/>
    </Sequence>

    <Sequence id="seq2" numberOfSets="1" occurrence="3" support="0,75">
      <SetReference setId="sp2"/>
```

```
    </Sequence>

    <Sequence id="seq3" numberOfSets="1" occurrence="2" support="0,5">
      <SetReference setId="sp3"/>
    </Sequence>

    <Sequence id="seq4" numberOfSets="1" occurrence="3" support="0,75">
      <SetReference setId="sp4"/>
    </Sequence>

    <Sequence id="seq5" numberOfSets="2" occurrence="2" support="0,5">
      <SetReference setId="sp4"/>
      <Delimiter delimiter="acrossTimeWindows" gap="unknown"/>
      <SetReference setId="sp3"/>
    </Sequence>

    <SequenceRule id="rule1" numberOfSets="2" occurrence="2" support="0,5"
      confidence="0,75">
      <AntecedentSequence>
        <SequenceReference seqId="seq4"/>
      </AntecedentSequence>
      <ConsequentSequence>
        <SequenceReference seqId="seq5"/>
      </ConsequentSequence>
    </SequenceRule>

  </SequenceModel>
</PMML>
```

Příloha C: Obsah přiloženého CD

Adresářová struktura přiloženého CD je následující:

- **thesis** – Text diplomové práce.
- **src –** Zdrojové soubory algoritmů a rozšíření SQL Serveru.
- **datasets –** Synteticky generované datové sady využité pro testování algoritmů.
- **docs –** Programová dokumentace k projektu.
- **plugins-install.pdf –** Postup instalace implementovaných rozšíření SQL Serveru.