VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

METODY DOLOVÁNÍ SEKVENČNÍCH VZORŮ

METHODS FOR MINING SEQUENTIAL PATTERNS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARTIN FEKETE**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Student: **Fekete Martin**
Program: Informační technologie
Název: **Metody dolování sekvenčních vzorů**
Methods for Mining Sequential Patterns
Kategorie: Data mining

Zadání:

1. Prostudujte problematiku získávání znalostí z dat, zaměřte se podrobněji na dolování sekvenčních vzorů.
2. Seznamte se s existujícími implementacemi a knihovnami zahrnujícími dolování sekvenčních vzorů.
3. Navrhněte experimentální aplikaci zahrnující zvolené metody dolování sekvenčních vzorů. Návrh konzultujte s vedoucím.
4. Implementujte navrženou aplikaci.
5. Proveďte experimenty s vhodně zvolenými datovými sadami, vyhodnoťte výhody a nevýhody jednotlivých metod.
6. Zhodnoťte dosažené výsledky a další možnosti pokračování v tomto projektu.

Literatura:

- Han, J., Kamber, M.: Data Mining - Concepts and Techniques, 2nd Edition. Morgan Kaufmann Publishers, 2006.
- Mabroukeh, N. R., Ezeife, C.: A Taxonomy of Sequential Pattern Algorithms, ACM Computing Surveys, Vol. 43, No. 1, ACM, 2011.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 22. října 2020

Abstrakt

Dolovanie sekvenčných vzorov je oblasť dolovania z dát so širokým využitím. V súčasnosti existuje množstvo algoritmov a prístupov k problému dolovania sekvenčných vzorov. Cieľom tejto práce je navrhnúť a implementovať aplikáciu určenú na dolovanie sekvenčných vzorov a pomocou nej experimentálne porovnať zvolené algoritmy. Experimenty sú vykonávané ako so syntetickými, tak aj s reálnymi databázami. Výstupom práce je zhrnutie výhod a nevýhod jednotlivých algoritmov pre rôzne druhy vstupných databáz a aplikácia implementujúca vybrané algoritmy knižnice SPMF.

Abstract

Sequential pattern mining is a field of data mining with wide applications. Currently, there are a number of algorithms and approaches to the problem of sequential pattern mining. The aim of this work is to design and implement an application designed for sequential pattern mining and use it to experimentally compare the chosen algorithms. Experiments are performed with both synthetic and real databases. The output of the work is a summary of the advantages and disadvantages of each algorithm for different kinds of input databases and an application implementing the selected algorithms of the SPMF library.

Kľúčové slová

získavanie znalostí z databáz, dolovanie z dát, sekvenčné vzory, dolovanie sekvenčných vzorov, apriori, expanzia vzoru, skoré odstraňovanie kandidátov, GSP, SPADE, SPAM, CM-SPADE, CM-SPAM, LAPIN, PrefixSpan, SPMF

Keywords

knowledge discovery from databases, data mining, sequential patterns, sequential pattern mining, apriori, pattern growth, early candidate pruning, GSP, SPADE, SPAM, CM-SPADE, CM-SPAM, LAPIN, PrefixSpan, SPMF

Citácia

FEKETE, Martin. *Metody dolování sekvenčních vzorů*. Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Metody dolování sekvenčních vzorů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Fekete
6. mája 2021

Poďakovanie

Rád by som poďakoval vedúcemu práce, pánovi Ing. Vladimírovi Bartíkovi, Ph.D. za odbornú pomoc, pripomienky a cenné rady.

Obsah

Kapitola 1

Úvod

V súčasnej dobe, keď množstvo dát v úložiskách po celom svete rastie nesmiernym tempom je otázka, ako tieto dáta spracovať čoraz relevantnejšia. Takmer o každom z nás je zbierané obrovské množstvo informácií, ktoré však bez správneho spracovania neposkytujú veľkú pridanú hodnotu. Klasické databázové systémy často nedokážu zodpovedať komplexnejšie otázky. Veľké množstvo týchto otázok je však možné zodpovedať použitím metód dolovania z dát.

Dolovanie znalostí z databáz je oblasť, ktorá si kladie za cieľ nájsť skryté, implicitné informácie vo veľkom množstve dát. Dolovanie znalostí má rozsiahle využitie v zdravotníctve, bankovom sektore alebo napríklad v školstve. Algoritmy dolovania znalostí tiež zohrávajú významnú rolu v obchodnej sfére. Analýzou nákupného košíku dokáže obchodník napríklad zistiť, aké položky sú nakupované spolu naprieč veľkým množstvom zákazníkov. Ak by však obchodníka zaujímalo aj poradie nákupov rôznych produktov, bolo by vhodné použiť metódy dolovania sekvenčných vzorov.

Od predstavenia prvých algoritmov dolovania sekvenčných vzorov sa táto oblasť dolovania znalostí každým rokom rozširuje. Pribúdajú nové algoritmy a prístupy riešenia tohto problému, ktoré so sebou prinášajú určité výhody, ale aj nevýhody. Cieľom tejto práce je sumarizovať a experimentálne porovnať najvýznamnejšie algoritmy dolovania sekvenčných vzorov a sumarizovať ich klady a zápory.

Práca je rozdelená do šiestich kapitol. V kapitole 2 je popísaná problematika dolovania znalostí z databáz, proces získavania znalostí, typy dolovacích úloh a typy dolovaných dát. V kapitole 3 je pozornosť zameraná na dolovanie sekvenčných vzorov a sú v nej rozobraté najvýznamnejšie prístupy a algoritmy. Kapitola 4 popisuje návrh a implementáciu experimentálnej aplikácie na dolovanie sekvenčných vzorov, ktorá je používaná na vykonávanie experimentov. Samotným experimentom s rôznymi druhmi vstupných databáz sa venuje kapitola 5. Zhrnutie výsledkov práce a jednotlivých metód sa nachádza v kapitole 6.

Kapitola 2

Získavanie znalostí z databáz

Táto kapitola sa zameriava na teoretický úvod do dolovania znalostí z databáz. Sekcia 2.1 sa venuje definícii dolovania znalostí z databáz a motivácii vzniku tohto odboru. Proces dolovania znalostí je popísaný v sekcii 2.2. V sekcii 2.3 je pozornosť zameraná na zdroje dát, ktoré môžu byť dolované. Sekcia 2.4 sa venuje najpoužívanejším typom dolovacích úloh. Užitočnosť nájdených vzorov a modelov je popísaná v poslednej sekcii 2.5. Informácie v tejto kapitole boli čerpané z [9] a [19].

2.1 Čo to je dolovanie znalostí

Dolovanie znalostí z databáz je vedný odbor, ktorý vznikol koncom 20. storočia. Motivácia jeho vzniku bola potreba získať relevantné informácie z obrovského množstva dát, ktoré najmä kvôli príchodu internetu rástlo (a stále rastie) vysokým tempom. Dolovanie znalostí sa dá formálne definovať ako extrakcia zaujímavých, skrytých, netriviálnych a potenciálne užitočných modelov dát a vzorov z veľkého objemu dát, pričom tieto modely a vzory reprezentujú znalosti získané z dát.

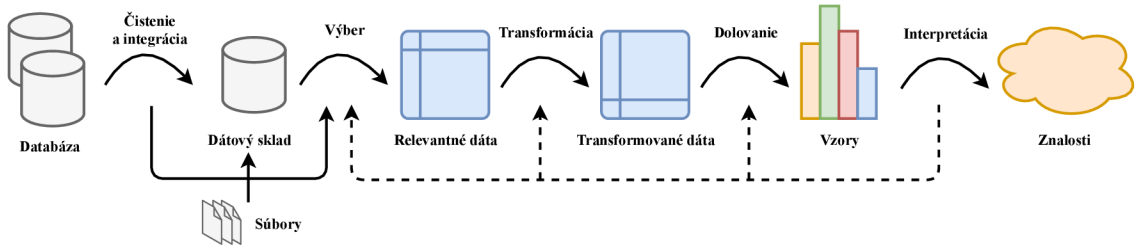
Získané znalosti by mali niesť novú informáciu, ktorá je potenciálne užitočná. Môže to byť napríklad informácia o podozrivých pohyboch na účtoch klientov, často opakujúcich sa položkách v nákupných košíkoch zákazníkov alebo pohyboch návštevníka na webových stránkach online obchodu. Na základe týchto informácií potom môže byť predídené krádeži peňazí, postavená marketingová kampaň alebo personalizovaný obsah webovej stránky.

2.2 Proces získavania znalostí z databáz

Proces dolovania znalostí je komplexný a iteratívny proces, ktorý za skladá z niekoľkých etáp. Jeho grafické znázornenie je možné vidieť na obrázku 2.1. Pre dosiahnutie čo najlepších výsledkov sa môžu niektoré etapy opakovať viackrát. Celý proces sa skladá z nasledovných etáp:

1. **Čistenie dát** – pre zaistenie čo najvyššej kvality získaných znalostí je nutné vstupné dáta očistiť o nekonzistentné, prípadne chýbajúce hodnoty.
2. **Integrácia dát** – úlohou tejto fázy je zlúčenie dát z viacerých zdrojov. Tento krok je často spojený s krokom čistenia dát, pretože práve rôzne zdroje dát sú zdrojom nekonzistencií.

3. **Výber dát** – cieľom tohto kroku je vybrať tie dáta, ktoré sú relevantné z pohľadu danej úlohy. Napríklad sú vybraté konkrétne stĺpce tabuľky relačnej databázy.
4. **Transformácia dát** – v tomto bode sú dáta transformované do podoby najvhodnejšej pre konkrétnu úlohu. Sú tu vykonávané napríklad operácie agregácie alebo sumarizácie.
5. **Dolovanie z dát** – fáza, ktorá je jadrom procesu dolovania znalostí z databáz. Pomocou vhodnej metódy sú zo vstupných dát extrahované vzory a modely, ktoré reprezentujú výsledné znalosti.
6. **Hodnotenie modelu/vzorov** – úlohou tohto kroku je identifikovať skutočne zaujímavé vzory na základe miery užitočnosti, ktorými môžu byť napr. podpora alebo spoľahlivosť.
7. **Prezentácia znalostí** – cieľom posledného kroku je prezentovať získané znalosti vhodným spôsobom, napríklad pomocou tabuliek alebo grafov.



Obr. 2.1: Jednotlivé kroky iteratívneho procesu dolovania znalostí. Prevzaté a upravené z [9].

Kroky čistenia, integrácie, výberu a transformácie dát sa súhrnne označujú ako *predspracovanie dát*. V kroku dolovania z dát môže dochádzať k interakcii s používateľom alebo databázou znalostí. Výsledok dolovania môže databázu znalostí potom obohatiť o nové znalosti, ktoré môžu byť využité v ďalších iteráciách alebo iných dolovacích úlohách.

2.3 Typy dát pre dolovanie

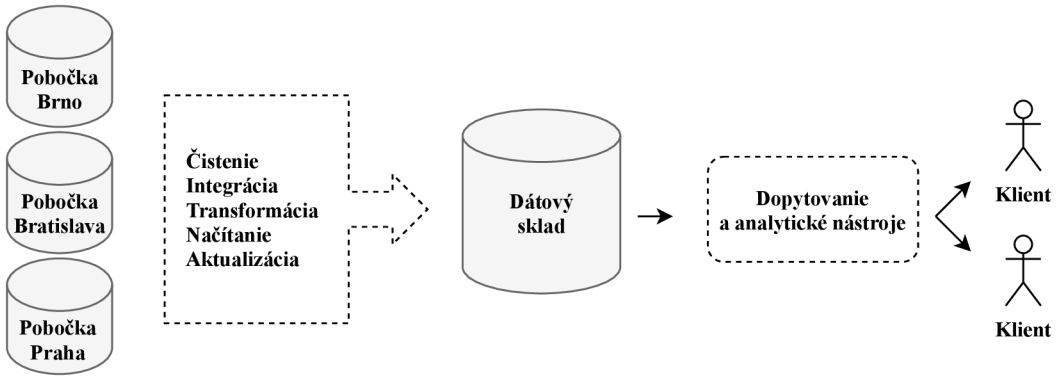
Vo všeobecnosti je možné dolovať takmer akékoľvek zmysluplné dáta, ktoré sú buď perzistentne uložené v úložiskách alebo sú tranzistentné (prúdy dát). Asi najčastejším zdrojom dát pre dolovacie úlohy sú relačné databázy. Ďalej to môžu byť dátové sklady, transakčné databázy alebo iné zdroje dát, ktoré sú predstavené na konci tejto sekcie.

2.3.1 Relačné databázy

Relačná databáza je kolekcia tabuliek, ktoré spĺňajú podmienku prvej normálnej formy, teda všetky stĺpce každej tabuľky obsahujú iba atomické hodnoty. Riadky tabuľky reprezentujú jednotlivé záznamy, stĺpce obsahujú informácie o atribútoch týchto záznamov. Každý záznam v tabuľke je jednoznačne identifikovaný primárnym kľúčom, ktorý je zložený z neprázdnej množiny atribútov. Dáta sú z relačnej databázy sprístupňované typicky príkazmi jazyka SQL. Príkaz je po zadaní transformovaný do množiny relačných operácií ako selekcia, spojenie a projekcia a následne je prípadne optimalizovaný.

2.3.2 Dátové sklady

Dátový sklad je komplexné úložisko, v ktorom sú uchovávané dáta z viacerých zdrojov. Dátové sklady sú budované procesom čistenia, integrácie, transformácie, načítania a periodického aktualizovania dát. Údaje sú v dátovom sklade typicky uložené z historickej perspektívy a zvyčajne sú sumarizované. Dátové sklady sú často modelované pomocou tzv. *multidimenzionálnych dátových kociek*. Dátová kocka je multidimenzionálna dátová štruktúra, ktorej každá dimenzia zodpovedá atribútu alebo skupine atribútov schémy databáze. Každá bunka obsahuje agregované údaje, napr. počet predaných kusov produktu alebo objem predaja.



Obr. 2.2: Koncepcia dátového skladu. Prevzaté a upravené z [9].

Dátové sklady sú vhodné na vykonávanie *OLAP* (Online Analytical Processing) operácií, ktoré umožňujú prezentovať dáta na rôznych úrovniach abstrakcie. Medzi OLAP operácie patria napr. operácie *drill-down* a *roll-up*. Prvá menovaná operácia sa používa na detailnejší pohľad na dáta v určitej dimenzii. Pri dimenzii *čas* môže byť použitá napr. na zmenu pohľadu zo štvrťročia na jednotlivé mesiace. Operácia roll-up slúži na opačný smer zmeny pohľadu, zvyšuje sa teda pomocou nej úroveň abstrakcie.

2.3.3 Transakčné databázy

Transakčná databáza je tvorená súborom, ktorého záznamy reprezentujú jednotlivé transakcie. Z pohľadu relačného modelu je takáto tabuľka nenormalizovaná. Transakcia je zvyčajne tvorená jej unikátnym identifikátorom a zoznamom položiek, ktoré ju tvoria. Príkladom transakcie môže byť zoznam položiek v jednom nákupe zákazníka. Dolovaním v takejto transakčnej databáze potom môže byť získaná množina produktov, ktoré sú často kupované spolu.

ID transakcie	Položky
t01	p3, p4, p8
t02	p1, p9
...	...

Tabuľka 2.1: Ukážka transakčnej databázy. Transakcia sa zvyčajne skladá z jednoznačného identifikátoru a zoznamu položiek, ktoré ju tvoria.

2.3.4 Ostatné typy dát

Okrem spomenutých databáz existuje aj veľké množstvo iných zdrojových dát na dolovanie znalostí:

- **Multimediálne databázy** – sú v nich uložené multimédiá ako fotografie, audio alebo videozáznamy.
- **Textové databázy** – databázy obsahujúce neštruktúrované, čiastočne štruktúrované (HTML, JSON, ...) alebo štruktúrované dokumenty (katalógy knižníc) údaje.
- **Priestorové databázy** – databázy obsahujúce informácie vzťahujúce sa k priestorovému usporiadaniu. Príkladom môže byť geografická databáza obsahujúca súradnice objektov na mape.
- **Sekvenčné databázy** – sú v nich uložené postupnosti usporiadaných udalostí. Čas vykonania udalostí nemusí byť explicitne zaznamenaný, dôležité je ich usporiadanie. Príkladom môže byť sekvencia viacerých nákupov zákazníka obchodu. Detailnejší popis dolovania tohto typu dát je rozoberaný v ďalšej časti tejto práce 3.
- **Web** – obrovská, rýchlo rastúca, heterogénna databáza. Typickými dolovacími úlohami sú napr. klasifikácia webových stránok alebo analýza webových komunít.

2.4 Typy dolovacích úloh

Ako už bolo spomenuté, dolovanie z dát je jadrom procesu získavania znalostí z databáz. Typom dolovacej úlohy rozumieme druh modelu dát, ktorý sa snažíme z dát dolovaním dostať. Dolovacie úlohy je možné rozdeliť do dvoch kategórií:

1. **Deskriptívne** – úlohy, ktoré charakterizujú všeobecné vlastnosti analyzovaných dát v databáze. Ich predstaviteľmi sú napr. zhluková analýza alebo hľadanie asociačných pravidiel.
2. **Prediktívne** – úlohy, ktoré na základe dostupných dát vykonávajú predpoveď budúcich hodnôt alebo správania. Predstaviteľmi sú klasifikácia alebo regresia.

2.4.1 Popis konceptu/triedy

Cieľom tejto deskriptívnej dolovacej úlohy je asociovať dáta s určitou triedou alebo konceptom. Napríklad firma predávajúca elektroniku môže mať triedy produktov ako *počítače* a *príslušenstvo k počítačom*. Potom je vhodné charakterizovať tieto triedy nejakým súhrnným, stručným a dostatočne presným popisom:

1. **Charakterizácia dát** – cieľom je sumarizovať všeobecné vlastnosti cieľovej triedy. Údaje analyzovanej triedy sú typicky získané dopytom nad databázou. Výstupom charakterizácie dát môžu byť napr. rôzne typy grafov alebo dátové kocky.
2. **Diskriminácia dát** – cieľom je porovnať všeobecné vlastnosti údajov cieľovej triedy so všeobecnými vlastnosťami inej triedy alebo množiny tried.

Príklad 2.4.1. Príkladom charakterizácie dát môže byť sumarizácia charakteristík skupiny zákazníkov, ktorí za posledný rok minuli u predajcu viac ako 500€. Výsledkom môže byť priemerný vek takéhoto zákazníka alebo oblasť zamestnania. Pri diskriminácii dát zase môžu byť porovnané vlastnosti skupiny zákazníkov, ktorí minú u predajcu za rok viac ako 500€ a tých, ktorí minú menej ako 50€.

2.4.2 Frekventované vzory a asociačné pravidlá

Frekventované vzory sú vzory, ktoré sa v dátach vyskytujú relatívne často. Existuje mnoho druhov frekventovaných vzorov, ako napr. frekventované množiny, podsekvencie (tiež známe ako sekvenčné vzory), podstromy a podgrafy.

Frekventovaná množina je množina položiek, ktoré sa často vyskytujú spolu. Príkladom frekventovanej množiny môže byť napríklad dvojica *klávesnica* a *myš* pri analýze nákupného košíka. Pomocou získaných frekventovaných vzorov sú v údajoch odhaľované zaujímavé korelácie a asociácie. Odhaľovanie zaujímavých asociácií prebieha procesom nazývaným asociačná analýza, ktorého výsledkom sú *asociačné pravidlá*. Príklad asociačného pravidla vyzerá nasledovne:

$$kupuje(X,\ 'klávesnica') \implies kupuje(X,\ 'myš')[podpora = 0.8\ \%,\ spoľahlivosť = 55\ \%],$$

kde premenná X označuje zákazníka. Toto asociačné pravidlo hovorí, že zákazníci, ktorí si kupujú klávesnicu si zvyčajne kúpia aj počítačovú myš. *Podpora* vyjadruje, v koľkých percentách nákupov si zákazníci zakúpili naraz obe tieto položky. *Spoľahlivosť* vyjadruje, že ak si zákazník kupuje klávesnicu, existuje 55 % šanca, že si zakúpi aj myš. Uvedené asociačné pravidlo obsahuje iba jeden predikát (kupuje), preto sa nazýva *jednodimenzionálne asociačné pravidlo*.

Asociačné pravidlá môžu obsahovať aj viacero predikátov, vtedy sú nazývané *multidimenzionálne*. Multidimenzionálne asociačné pravidlo s tromi predikátmi môže vyzerať napríklad nasledovne:

$$vek(X,\ '18..24') \wedge pohlavie(X,\ 'muž') \implies kupuje(X,\ 'tlačiareň')$$
$$[podpora = 0.9\ \%,\ spoľahlivosť = 65\ \%].$$

Pravidlo hovorí, že 0.9 % analyzovaných zákazníkov mužského pohlavia vo veku 18 až 24 rokov si zakúpilo tlačiareň. Existuje 65 % pravdepodobnosť, že si zákazník, ktorý je muž v tomto veku zakúpi tlačiareň.

Za zaujímavé sú typicky považované iba tie asociačné pravidlá, ktoré spĺňajú stanovený minimálny prah podpory a spoľahlivosti.

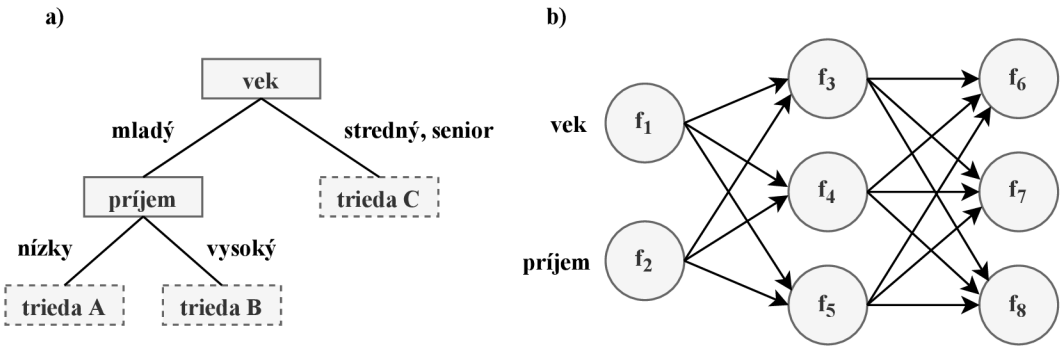
2.4.3 Klasifikácia a regresia

Klasifikácia je proces hľadania modelu, ktorý opisuje a rozdeľuje dáta na základe ich vlastností do konečnej množiny tried. Model je získaný analýzou trénovacích dát (t. j. dát, ktorých trieda zaradenia je známa) a potom použitý na predikciu tried objektov, ktorých zaradenie je neznáme.

Proces klasifikácie sa skladá z troch krokov, a to trénovanie, testovanie a aplikácia. Trénovanie a testovanie prebieha pomocou dát, ktorých trieda je známa. Aplikácia je použitie získaného modelu na klasifikáciu objektov, ktorých triedy sú neznáme.

Klasifikačný model môže mať rôznu podobu, ako napr. rozhodovacie stromy, neurónové siete alebo klasifikačné pravidlá. Ich prvé dve formy je možné vidieť na obrázku 2.3.

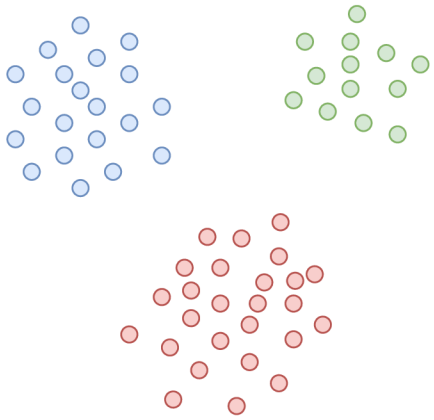
Klasifikácia sa používa na predikciu kategorických (diskrétnych, nezoradených) hodnôt. *Regresia* je používaná na predikciu numerických hodnôt spojitého charakteru. Pojem *predikcia* sa vzťahuje ako k numerickej predikcii, tak aj k predikcii kategorických hodnôt. Najčastejšou metódou numerickej predikcie je *regresná analýza*.



Obr. 2.3: Formy reprezentácie klasifikačných modelov: a) rozhodovací strom, b) neurónová sieť. Prevzaté a upravené z [9].

2.4.4 Ostatné typy dolovacích úloh

Ďalší obvyklý typ dolovacej úlohy je *zhluková analýza*. Rozdiel oproti klasifikácii a regresii je, že zhluková analýza rozdeľuje objekty do predom neznámych tried (zhlukov). Jej cieľom je maximalizovať podobnosť objektov v rámci zhluku, ale minimalizovať podobnosť objektov patriacich rozdielnym zhlukom. Výsledkom zhlukovania sú homogénne skupiny. Ukážku výstupu zhlukovej analýzy je možné vidieť na obrázku 2.4.



Obr. 2.4: Ukážka troch zhlukov vytvorených zhlukovou analýzou podľa bydliska zákazníkov. Prevzaté a upravené z [9].

Analýza odľahlých objektov na rozdiel od doteraz spomenutých dolovacích úloh považuje za zaujímavé práve také údaje, ktoré sa výrazne odlišujú od ostatných. V praxi môže byť analýza odľahlých objektov využitá na detekciu odcudzenia kreditnej karty, napr. detekciou nezvyčajne vysokých platieb alebo nezvyčajných lokácií týchto platieb.

Posledným typom dolovacej úlohy je *evolučná analýza*. Tá modeluje trendy u objektov, ktorých chovanie sa mení v čase. Evolučná analýza je vhodná najmä na vyhľadanie periodicity alebo podobnosti dát. Príkladom použitia môže byť dolovanie v dátach obsahujúcich vývoj cien akcií pre nájdenie pravidelností, ktoré by mohli odhaliť vývoj budúcich cien.

2.5 Užitočnosť nájdených vzorov

Po skočení procesu dolovania znalostí sa prirodzene nastoľuje otázka, či sú všetky nájdené vzory a modely užitočné. Všeobecne sa dá povedať, že nie. Existujú 4 vlastnosti, ktoré charakterizujú zaujímavý vzor:

- jednoduchá zrozumiteľnosť človekom,
- platnosť pre nové alebo testovacie dáta s istým stupňom istoty,
- potenciálna užitočnosť,
- neznámosť/novosť.

Zaujímavý je tiež ten vzor/model, ktorý validuje hypotézu, ktorú sa snaží užívateľ potvrdiť. Ten potom predstavuje znalosť.

Miery zaujímavosti môžu byť *objektívne* a *subjektívne*. Subjektívne miery zaujímavosti sú napríklad novosť alebo neočakávanosť (sú v kontradikcii s používateľovým názorom). Medzi objektívne miery zaujímavosti patrí už vyššie spomenutá podpora a spoľahlivosť. Vo všeobecnosti sa pred výkonom samotnej dolovacej úlohy dajú nastaviť minimálne hodnoty objektívnych mier (napr. minimálna podpora 50 %). Potom sú všetky vzory, ktoré túto hodnotu nespĺňajú považované za nezaujímavé.

Je žiaduce, aby algoritmy hľadali iba zaujímavé vzory. To je možné dosiahnuť dvojakým spôsobom. Prvý spôsob je najprv nájsť všetky vzory a následne odfiltrovať tie nezaujímavé. Tento spôsob je ale nereálny, pretože by to znamenalo úplné prehľadanie priestoru možných riešení. Druhý prístup je priamo generovať iba zaujímavé vzory, čo je optimalizačný problém. Je dôležité podotknúť, že nie vždy je možné a efektívne nájsť úplnú množinu zaujímavých vzorov a pravidiel, najmä kvôli veľkosti prehľadávaného priestoru. Preto sú používané rôzne obmedzenia na hľadané vzory.

Kapitola 3

Dolovanie sekvenčných vzorov

Táto kapitola sa zameriava na problematiku dolovania sekvenčných vzorov. V sekcii 3.1 je popísané využitie sekvenčných vzorov. Sekcia 3.2 vysvetľuje základné pojmy týkajúce sa sekvenčných vzorov a následne uvádza tri hlavné kategórie algoritmov dolovania sekvenčných vzorov. Každej kategórii je potom vyčlenená jedna sekcia, v ktorej sú predstavené najvýznamnejšie algoritmy, ktoré do nej spadajú.

Algoritmy patriace do prvej a zároveň najstaršej kategórie, popísanej v časti 3.3, sú založené na *apriori vlastnosti*. O niečo mladšia skupina algoritmov je založená na princípe *expanzii vzoru* a je popísaná v časti 3.4. Posledná podkapitola 3.5 rozoberá algoritmy založené na *skorom odstraňovaní kandidátov*, resp. skorom prerezávaní prehľadávaného priestoru.

3.1 Využitie sekvenčných vzorov

Problematika dolovania sekvenčných vzorov bola prvýkrát predstavená v 90. rokoch 20. storočia v práci R. Agrawala a R. Strikanta [1]. Dolovanie sekvenčných vzorov sa zaoberá hľadaním zaujímavých podsekvencií v množine sekvencií, pričom zaujímavosť podsekvencií môže byť merateľná viacerými kritériami – napríklad počtom jej výskytov alebo jej dĺžkou [8]. Istým spôsobom je táto problematika podobná dolovaniu frekventovaných množín. Na rozdiel od dolovania frekventovaných množín sa však pri dolovaní sekvenčných vzorov berie ohľad na poradie udalostí vo vstupnej sekvenčnej databáze.

Dolovanie sekvenčných vzorov má rozsiahle využitie. Práca [5] popisuje využitie algoritmov dolovania sekvenčných vzorov na detekciu škodlivého softvéru. Zaujímavé je, že algoritmus predstavený v tejto práci je schopný detekovať aj doposiaľ neznámy malvér na základe sekvencií kódu v spustiteľných súboroch. V práci [15] sa zase autori zamerali na dolovanie sekvencií v zdravotných záznamoch, konkrétne predpisoch liekov pre pacientov s diabetom. Vďaka získaným sekvenčným vzorom potom bolo možné predikovať ďalší predpísaný liek s vysokou presnosťou. Autori navrhujú, že na základe týchto výsledkov by bolo možné vytvoriť systém pre automatizovaný návrh ďalšieho predpísaného lieku pri zmenách liečby. Článok [4] zase predstavuje metódu na dolovanie DNA sekvencií, ktorá ale môže byť aplikovaná aj na sekvencie RNA, proteínové sekvencie alebo iné biologické sekvencie. Ďalšia oblasť využitia sekvenčných vzorov je e-learning. Autori práce [14] navrhli systém, ktorý je schopný odporúčať študijné materiály na základe preferencií a znalostí študenta. Využitiu sekvenčných vzorov v oblasti vzdelávania sa venuje aj článok [11], ktorého autori identifikovali produktívne a neproduktívne sekvencie učebných návykov, čo dosiahli porovnávaním

návykov (ako napr. čítanie, pýtanie sa otázok alebo vysvetľovanie) viac a menej úspešných študentov.

3.2 Definícia problému

Vstupom úlohy pre dolovanie sekvenčných vzorov je sekvenčná databáza. Ukážku takejto databázy je možné vidieť nižšie. Táto databáza obsahuje štyri vstupné sekvencie, ktoré môžu reprezentovať napr. nákupy vykonané štyrmi zákazníkmi obchodu.

ID sekvencie	Sekvencia
s_1	$\langle a(abc)(ac)d(cf)\rangle$
s_2	$\langle (ad)c(bc)(ae)\rangle$
s_3	$\langle (ef)(ab)(df)cb\rangle$
s_4	$\langle eg(af)cbc\rangle$

Tabuľka 3.1: Ukážková sekvenčná databáza. Prevzaté z [13].

Nech $I = \{i_1, i_2, i_3, ..., i_k\}$ je množina všetkých *položiek* vo vstupnej databáze. *Udalosť* je definovaná ako neprázdna množina položiek a je označovaná ako $(i_1 i_2 i_3 ... i_k)$[1]. Na poradí položiek v rámci udalosti nezáleží, zvyčajne sú ale usporiadané lexikograficky[2]. *Sekvencia* je usporiadaný zoznam udalostí a je označovaná ako $\langle e_1 e_2 e_3 ... e_n\rangle$, kde udalosť e_1 predchádza udalosť e_2, udalosť e_2 predchádza udalosť e_3 atď. Jedna položka sa v rámci udalosti môže vyskytnúť maximálne raz, avšak môže sa vyskytovať vo viacerých udalostiach jednej sekvencie. Sekvencia, ktorá obsahuje k položiek je označovaná ako k-sekvencia.

Príklad 3.2.1. Predpokladajme sekvenčnú databázu 3.1. Množina všetkých položiek tejto databázy je $\{a, b, c, d, e, f, g\}$. Sekvencia s_1 je tvorená piatimi udalosťami a keďže obsahuje deväť inštancií položiek, jedná sa o 9-sekvenciu. Sekvencia s_4 je tvorená šiestimi udalosťami a je to 7-sekvencia.

Sekvencia $\alpha = \langle a_1 a_2 a_3 ... a_n\rangle$ je *podsekvencia* sekvencie $\beta = \langle b_1 b_2 b_3 ... b_n\rangle$ ak existujú celé čísla také, že $1 \leq i_1 < i_2 < i_3 < ... < i_n \leq n$, pre ktoré platí $a_1 \subseteq b_{i_1}$, $a_2 \subseteq b_{i_2}$, ..., $a_n \subseteq b_{i_n}$. Zároveň môžeme povedať, že sekvencia β je *nadsekvencia* sekvencie α.

Nech je daná databáza D obsahujúca sekvenčné záznamy. *Podpora* (support) vzoru S v databáze D je definovaná ako počet sekvencií databázy D obsahujúcich vzor S vydelený celkovým počtom sekvencií v databáze D. *Absolútna podpora* označuje celkový počet sekvencií v databáze obsahujúcich vzor S. Sekvencia je označená za *frekventovanú* vtedy, ak je jej podpora rovná alebo vyššia ako používateľom špecifikovaný prah *minimálnej podpory*, skrátene označovaný ako *minsup*. Pojem *sekvenčný vzor* sa používa ako synonymum k pojmu frekventovaná sekvencia. Sekvencia je *uzavretá* práve vtedy, keď je frekventovaná a nijaká z jej nadsekvencií nemá rovnakú podporu.

Príklad 3.2.2. Sekvencia $\langle a(bc)df\rangle$ je podsekvencia sekvencie s_1, pretože je zachované poradie udalostí a zároveň platí, že $a \subseteq a$, $(bc) \subseteq (abc)$, $d \subseteq d$ a $f \subseteq (cf)$. Pri zadanom prahu minimálnej podpory $minsup = 0.5$ (50 %), môžeme povedať, že sekvencia $s = \langle (ab)c\rangle$

[1] Ak udalosť obsahuje iba jednu položku, je zvyčajne zobrazovaná bez zátvoriek.

[2] Lexikografické usporiadanie, označované ako $\succ_{lex}$, je definované ako úplné usporiadanie na množine I.

je frekventovaná, pretože sa nachádza v sekvenciách s_1 a s_3. Sekvencia s je tiež uzavretá, pretože v databáze neexistuje jej nadsekvencia, ktorá má rovnakú podporu.

Algoritmy dolovania sekvenčných vzorov využívajú pri prehľadávaní priestoru 2 základné operácie, a to *s-rozšírenie* (rozšírenie sekvencie) a *i-rozšírenie* (rozšírenie udalosti), niekedy nazývané aj ako *s-krok* a *i-krok*. Pomocou týchto operácií sú generované $(k+1)$-sekvencie z aktuálnych k-sekvencií. Sekvencia s_b je s-rozšírením sekvencie $s_a = \langle e_1 e_2 e_3 ... e_n \rangle$ o položku x, ak platí, že $s_b = \langle e_1 e_2 e_3 ... e_n x \rangle$. Sekvencia s_b je i-rozšírením sekvencie $s_a = \langle e_1 e_2 e_3 ... e_n \rangle$ o položku x, ak platí, že $s_b = \langle e_1 e_2 e_3 ... e_n \cup x \rangle$.

Príklad 3.2.3. Uveďme si príklad na s a i-rozšírenia. Sekvencie $\langle ab \rangle$ a $\langle ac \rangle$ sú s-rozšírením sekvencie $\langle a \rangle$ o položky b a c. Na druhú stranu, sekvencia $\langle a \rangle$ je i-rozšírená o položky b a c v prípade sekvencií $\langle (ab) \rangle$ a $\langle (ac) \rangle$.

Prehľadávaný priestor môže byť prechádzaný buď *do hĺbky (BFS)* alebo *do šírky (DFS)*. Algoritmy využívajúce metódu BFS nájdu najprv všetky frekventované 1-sekvencie, ktoré sú potom pomocou s a i-rozšírení rozširované na 2-sekvencie, na základe nich 3-sekvencie a tento proces pokračuje po dobu, dokedy je možné generovať nové sekvencie. Z každej úrovne sú teda najprv nájdené všetky frekventované sekvencie a až potom sa postupuje na ďalšiu úroveň. Algoritmy patriace do druhej skupiny najprv vygenerujú frekventované 1-sekvencie a potom rekurzívne vykonávajú s a i-rozšírenia na jednu z týchto sekvencií. Takto rekurzívne sú generované nové sekvencie pokiaľ to je možné. Ak žiadne ďalšie sekvencie nemôžu byť vygenerované, je vykonaný *spätný chod* (backtracking) k sekvenciám, ktoré je možné ďalej rozširovať.

Odbor dolovania sekvenčných vzorov sa každým rokom rozširuje o nové algoritmy. V práci sú predstavené a experimentálne porovnané najvýznamnejšie algoritmy, ktoré sa na problém dolovania sekvenčných vzorov pozerajú inými spôsobmi. Ich rozdelenie je založené na práci [12] do troch kategórií, a to algoritmy založené na:

1. apriori vlastnosti,
2. expanzii vzoru,
3. skorom odstraňovaní kandidátov.

3.3 Algoritmy založené na apriori vlastnosti

Základom tejto skupiny algoritmov je apriori vlastnosť, ktorá hovorí, že všetky podsekvencie frekventovanej sekvencie musia byť takisto frekventované; naopak, ak sekvencia nie je frekventovaná, tak aj všetky jej nadsekvencie nie sú frekventované. Majme napríklad sekvenciu $\langle a \rangle$ a $\langle (ab) \rangle$. Je zrejmé, že sekvencia $\langle (ab) \rangle$ nemôže mať podporu vyššiu ako sekvencia $\langle a \rangle$, pretože je viac špecifická. Ak teda sekvencia $\langle a \rangle$ nespĺňa prah minimálnej podpory, je bezpečné s ňou nepočítať.

Algoritmus GSP 3.3.1 pracuje s horizontálnym formátom databázy a priestor prehľadáva metódou BFS, teda všetky k-sekvencie sú nájdené pri k-tej iterácii algoritmu. Ako sa však ukázalo, tento prístup so sebou prináša isté nevýhody, čo bolo motiváciou pre vznik algoritmov využívajúcich vertikálny formát databázy a prehľadávanie do hĺbky. Ich najvýznamnejší zástupcovia sú SPADE 3.3.2 a SPAM 3.3.3. Medzi najnovšie algoritmy využívajúce vertikálny formát databázy patria algoritmy CM-SPADE a CM-SPAM 3.3.4.

3.3.1 GSP

Prvé algoritmy dolovania sekvenčných vzorov, AprioriAll, AprioriSome a DynamicSome, boli predstavené v práci [1]. Rovnakí autori predstavili o krátku dobu algoritmus GSP (Generalized Sequential Patterns) [2], ktorý je výrazne rýchlejší ako jeho predchodcovia. Aj napriek tomu je jeho výkon najmä pri veľkom množstve vstupných dát nedostačujúci. Je to spôsobené hlavne nutnosťou prechádzať vstupnú databázu počas procesu dolovania viacnásobne.

Na začiatku algoritmus vykoná počiatočný prechod databázou, ktorým sú nájdené všetky položky, ktoré spĺňajú prah minimálnej podpory. Tieto položky tvoria frekventované 1-sekvencie (sekvenčné vzory), ktoré sa stanú *počiatočnými položkami (seed set)* pre ďalší prechod. Pomocou počiatočných položiek sú v každom ďalšom prechode generované nové potenciálne frekventované sekvencie nazývané *kandidátne sekvencie*. Podpora kandidátnych sekvencií je počítaná prechodom databázou a tie kandidátne sekvencie, ktoré spĺňajú minimálnu podporu sa stanú počiatočnými položkami pre ďalší prechod. Algoritmus je ukončený, ak nie sú generované nijaké nové počiatočné položky alebo kandidátne sekvencie.

Definícia 3.3.1. Nech je daná sekvencia $s = \langle e_1 e_2 e_3 ... e_n \rangle$ a jej podsekvencia c. Sekvencia c je *susedná (contiguous)* podsekvencia sekvencie s ak je splnená niektorá z nasledujúcich podmienok [2]:

1. sekvencia c je odvodená zo sekvencie s odstránením udalosti buď e_1 alebo e_n,
2. sekvencia c je odvodená zo sekvencie s odstránením jednej položky z udalosti e_i ($1 \leq i \leq n$), pričom e_i má aspoň dve položky,
3. c je susedná podsekvencia c', pričom c' je susedná podsekvencia s.

Príklad 3.3.1. Nech je daná sekvencia $s = \langle (ab)(cd)ef \rangle$. Jej susedné sekvencie sú napríklad $\langle b(cd)e \rangle$, $\langle (ab)cef \rangle$ a $\langle ce \rangle$. Na druhú stranu sekvencie $\langle (ab)(cd)f \rangle$ a $\langle aef \rangle$ nie sú jej susedné sekvencie [2].

Generovanie nových kandidátnych sekvencií, teda $(k+1)$-sekvencií z aktuálnych k-sekvencií, prebieha v dvoch fázach [2]:

1. **Fáza spojenia (join phase)**. Sekvencie s_1 a s_2 sú spojené práve vtedy, ak je sekvencia s_1 po odstránení jej prvej položky zhodná so sekvenciou s_2 po odstránení jej poslednej položky. Kandidátna sekvencia je potom vytvorená rozšírením sekvencie s_1 o poslednú položku s_2. Táto položka vytvorí samostatnú udalosť vtedy, ak stála samostatne aj v sekvencii s_2. Inak sa stane súčasťou poslednej udalosti s_1. Napríklad sekvencia $s_1 = \langle (ab)c \rangle$ po spojení so sekvenciou $s_2 = \langle b(cd) \rangle$ vytvorí kandidátnu sekvenciu $\langle (ab)(cd) \rangle$, ale po spojení so sekvenciou $s_3 = \langle bce \rangle$ vytvorí kandidátnu sekvenciu $\langle (ab)ce \rangle$.
2. **Fáza odstraňovania (prune phase)**. Odstránené sú tie kandidátne sekvencie, ktorých $(k-1)$-susedné podsekvencie majú podporu menšiu ako zadaná minimálna podpora.

Príklad 3.3.2. Ukážme si na príklade princíp fungovania algoritmu GSP. Majme vstupnú databázu 3.1 a prah minimálnej podpory 100 %, teda 4 sekvencie.

Prvým prechodom databázou je nájdená množina všetkých položiek, ktoré sa v databáze nachádzajú. V našej ukážke je to množina $\{a, b, c, d, e, f, g\}$. Pre každú položku z tejto

množiny je zároveň spočítaná podpora a následne sú odstránené tie položky, ktoré nespĺňajú zadanú minimálnu podporu. Počiatočnou množinou pre ďalší prechod bude teda množina 1-sekvencií $\{\langle a\rangle, \langle b\rangle, \langle c\rangle\}$.

Spojením sekvencií z počiatočnej množiny je vytvorená množina kandidátnych sekvencií $\{\langle aa\rangle, \langle ab\rangle, \langle ac\rangle, \langle ba\rangle, \langle bb\rangle, \langle bc\rangle, \langle ca\rangle, \langle cb\rangle, \langle cc\rangle, \langle (ab)\rangle, \langle (ac)\rangle, \langle (bc)\rangle\}$. Pre položky tejto množiny je opäť nutné prejsť celú databázu, spočítať ich podporu a následne odstrániť tie, ktorých podpora je menšia ako 4. Tým je získaná nová počiatočná množina obsahujúca iba dve 2-sekvencie, a to $\langle ab\rangle$ a $\langle ac\rangle$.

Sekvencie v tejto počiatočnej množine už podľa fázy spojenia nemôžu byť spojené, teda vzniknutá množina kandidátnych sekvencií je prázdna. Tým pádom je algoritmus ukončený a konečná množina sekvenčných vzorov je množina $\{\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle ab\rangle, \langle ac\rangle\}$.

Ako je možné vidieť na príklade 3.3.2, algoritmus generuje množstvo kandidátnych sekvencií, z ktorých sa väčšina ani nemusí nachádzať vo vstupnej databáze. Toto je spôsobené práve krokom spojenia. Pre každú kandidátnu množinu je potom nutné prejsť celú databázu a spočítať podporu všetkých jej sekvencií. Práve toto je jeden z najväčších nedostatkov algoritmu GSP.

Autori algoritmu predstavujú aj obmedzenia na hľadané sekvenčné vzory, ako sú napr. časové okná, v rámci ktorých sú viaceré udalosti brané ako jedna udalosť. Ďalšie obmedzenia sa týkajú minimálnej, resp. maximálnej vzdialenosti dvoch udalostí a užívateľsky definovaných hierarchií vo vstupnej databáze. Detaily je možné nájsť v článku [2].

3.3.2 SPADE

Algoritmus SPADE [18] využíva na rozdiel od GSP *vertikálnu reprezentáciu databázy* pomocou tzv. *IDListov*. Takáto reprezentácia databázy indikuje pozíciu každej položky v rámci udalostí každej sekvencie. V tabuľke 3.2 je možné vidieť transformáciu vstupnej horizontálnej sekvenčnej databázy 3.1 na IDList reprezentáciu databázy pre položky a a b. ID sekvencie je značené ako SID a poradie udalosti v rámci sekvencie je značené ako EID. SPADE rozdeľuje prehľadávaný priestor do menších sektorov, ktoré môžu byť ďalej spracovávané nezávisle od seba, prípadne aj paralelne. Algoritmu zvyčajne stačí jeden prechod databázou, ktorým sú získané IDListy pre frekventované položky. IDList reprezentácie dlhších vzorov sú potom získané spájaním existujúcich IDListov. Názornú ukážku rozširovania je možné vidieť nižšie 3.3.3. Vďaka tomu nie je nutné viacnásobne prechádzať databázu, čo prináša obrovskú výhodu oproti algoritmu GSP. Podpora vygenerovanej sekvencie je získaná spočítaním všetkých unikátnych identifikátorov sekvencií (SID) v jej IDList reprezentácii.

a		**b**		...
SID	EID	SID	EID	
1	1	1	2	
1	2	2	3	
1	3	3	2	
2	1	3	5	...
2	4	4	5	
3	2			
4	3			

Tabuľka 3.2: Ukážka vertikálnej IDList reprezentácie vstupnej databázy 3.1 pre položky a a b

Príklad 3.3.3. Na príklade si môžeme ukázať, ako funguje generovanie kandidátnych sekvencií pomocou vertikálnej reprezentácie databázy. Pre vytvorenie sekvencie $\langle ab \rangle$ sa stačí pozrieť na vertikálny formát vstupnej databázy a nájsť také dvojice, kde SID položiek a a b je rovnaké a zároveň EID položky a je menšie ako EID položky b (teda položka a predchádza v rámci danej sekvencie položku b). Môžeme vidieť, že v tabuľke sú štyri takéto kombinácie pre štyri rôzne SID. Podpora sekvencie $\langle ab \rangle$ je teda štyri. V prípade sekvencie $\langle ba \rangle$ nájdeme v tabuľke takéto kombinácie dve, podpora tejto sekvencie je teda dva. Tabuľka nižšie zobrazuje ukážku spojenia pre lepšiu predstavu.

ab			**ba**			...
SID	EID (a)	EID (b)	SID	EID (b)	EID (a)	
1	1	2	1	2	3	
2	1	3	2	3	4	
3	2	5				...
4	3	5				

Tabuľka 3.3: Ukážka generovania sekvencie $\langle ab \rangle$ a $\langle ba \rangle$ na základe IDList reprezentácie položiek a a b.

Prístup algoritmu SPADE sa javí ako veľmi efektívny v porovnaní s prístupom algoritmu GSP, a to najmä vďaka tomu, že nie je nutné prechádzať vstupnú databázu viackrát a nevzniká veľké množstvo kandidátnych sekvencií.

Jeho modifikácia, algoritmus cSPADE [17], predstavuje obmedzenia pre hľadané sekvencie. Konkrétne sú to obmedzenia na dĺžku a šírku hľadaných sekvencií, časové obmedzenie medzi udalosťami sekvencie, časové okná pre celé sekvencie, obmedzenia na sekvencie obsahujúce konkrétne položky a pri špeciálnych databázach aj obmedzenie súvisiace s triedami sekvencií. Detaily týkajúce sa tohto algoritmu je možné nájsť v [17].

3.3.3 SPAM

Algoritmus SPAM (Sequential Pattern Mining using A Bitmap Representation) [3] využíva vertikálnu *bitmapovú* reprezentáciu databázy, ktorá umožňuje efektívne vytváranie nových kandidátnych sekvencií a počítanie podpory. Prehľadávaný priestor je reprezentovaný *lexikografickým sekvenčným stromom*, ktorý je prechádzaný metódou DFS.

Bitmapová reprezentácia databázy

Prvým prechodom databázou sú nájdené frekventované položky (1-sekvencie), pre ktoré je vytvorená vertikálna bitmapa. Každý bit tejto bitmapy, korešponduje s každou udalosťou každej sekvencie vstupnej sekvenčnej databázy. Dĺžka jedného slotu v bitmape je určená podľa najdlhšej sekvencie vstupnej databázy. Takáto reprezentácia potom umožňuje efektívny výpočet podpory jednotlivých sekvencií. Ten prebieha jednoduchou kontrolou, či jednotlivé sloty bitmapy majú všetky bity nastavené na 0 (podpora nie je inkrementovaná), alebo je aspoň jeden z bitov nastavený na 1 (podpora sa každým takýmto slotom inkrementuje). V ukážke 3.4 je možné vidieť bitmapovú reprezentáciu vstupnej databázy pre prah minimálnej podpory 50 %. Keďže najdlhšia sekvencia vo vstupnej databáze obsahuje 4 udalosti, počet bitov v jednotlivých slotoch bitmapy je 4. Celkovo obsahuje vstupná databáza 4 sekvencie, výsledné bitmapy majú teda celkovú dĺžku 16 bitov.

ID sekvencie	Sekvencia
s_1	$\langle(cd)(abc)(abf)(acdf)\rangle$
s_2	$\langle(abf)e\rangle$
s_3	$\langle(abf)\rangle$
s_4	$\langle(dgh)(bf)(agh)\rangle$

	a	b	d	f
s_1	0	0	1	0
	1	1	0	0
	1	1	0	1
	1	0	1	1
s_2	1	1	0	1
	0	0	0	0
	0	0	0	0
	0	0	0	0
s_3	1	1	0	1
	0	0	0	0
	0	0	0	0
	0	0	0	0
s_4	0	0	1	0
	0	1	0	1
	1	0	0	0
	0	0	0	0

Tabuľka 3.4: Bitmapová reprezentácia (vpravo) vstupnej databázy (vľavo, prevzaté z [18]) pre položky spĺňajúce prah minimálnej podpory 50 %. Každá sekvencia je reprezentovaná jedným slotom bitmapy, jej udalosti sú potom bity tohto slotu.

V ukážke 3.5 je zobrazený princíp fungovania s-rozšírenia. Najprv je potrebné transformovať vstupnú bitmapu. Transformácia je vykonávaná tak, že všetky bity od začiatku slotu po prvý bit (vrátane), ktorého hodnota je 1, sú nastavené na 0. Zvyšok bitov v rámci slotu je potom nastavený na 1. Táto transformácia je vykonaná pre všetky sloty vstupnej bitmapy. Medzi transformovanou bitmapou a bitmapou, o ktorú je vstupná bitmapa rozširovaná je potom vykonaný bitový *AND*. Princíp vykonávania i-rozšírenia je podobný s tým rozdielom, že nie je vykonávaná transformácia vstupnej bitmapy.

Podpora takto vygenerovaných sekvencií je potom získaná podobne ako v prípade algoritmu SPADE. V tomto prípade je ale spočítaný počet rôznych bitových sektorov, ktoré obsahujú nejaký bit nastavený na 1. Môžeme vidieť, že rozšírená sekvencia $\langle ab\rangle$ z ukážky 3.5 má tento sektor práve jeden – ten prvý. Pri pohľade na vstupnú databázu 3.4 je možné vidieť, že podsekvenciu $\langle ab\rangle$ obsahuje iba sekvencia s_1.

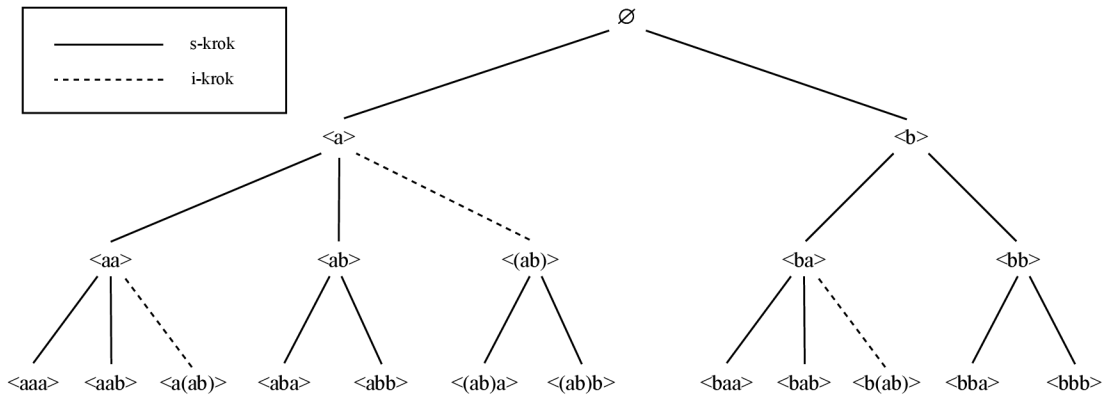
$\langle a\rangle$		$\langle a\rangle_s$		$\langle b\rangle$		$\langle ab\rangle$
0		0		0		0
1		0		1		0
1		1		1		1
1		1		0		0
1		0		1		0
0		1		0		0
0		1		0		0
0	s-rozšírenie	1	&	0	výsledok	0
1	$\longrightarrow$	0		1	$\longrightarrow$	0
0		1		0		0
0		1		0		0
0		1		0		0
0		0		0		0
0		0		1		0
1		0		0		0
0		1		0		0

Tabuľka 3.5: Ukážka transformácie vstupnej bitmapy a následného s-rozšírenia pomocou bitovej operácie *AND*.

Lexikografický strom

Prehľadávaný priestor je v algoritme SPAM reprezentovaný *lexikografickým sekvenčným stromom*. Majme lexikografické usporiadanie množiny položiek I vo vstupnej databáze. Ak sa položka i udeje pred položkou j v tomto usporiadaní, označujeme to ako $i \leq_I j$. Takéto usporiadanie sa dá rozšíriť na sekvencie tak, že ak je sekvencia s_1 podsekvenciou sekvencie s_2, bude to značené vzťahom $s_1 \leq s2$. Ak nie je sekvencia s_1 podsekvenciou s_2, nemajú v tomto usporiadaní nijaký vzťah. Lexikografický sekvenčný strom T je potom definovaný nasledovne. Koreň stromu je označený $\emptyset$. Rekurzívne, ak je n uzol stromu, potom pre všetky jeho synovské uzly n' platí $n \leq n'$ a $\forall m \in T : n' \leq m \implies n \leq m$ [3].

Každý uzol stromu je rekurzívne rozširovaný pomocou s alebo i-rozšírení. Na obrázku 3.1 je možné vidieť lexikografický strom pre množinu frekventovaných položiek $\{a, b\}$ zobrazený do tretej úrovne.



Obr. 3.1: Tri úrovne zanorenia v lexikografickom strome pre frekventované položky a a b.

Algoritmus prehľadáva lexikografický strom metódou prehľadávania do hĺbky. Pre každý uzol sú vygenerovaní jeho potomkovia a je otestovaná ich podpora. Ak vygenerovaná sekvencia spĺňa prah minimálnej podpory *minsup*, je uložená a rekurzívne sa pokračuje na ňu. Ak vygenerovaná sekvencia prah minimálnej podpory nespĺňa, podľa apriori pravidla nie je nutné pokračovať na jej potomkov. Ak nijaká z vygenerovaných sekvencií nespĺňa prah minimálnej podpory, algoritmus vykoná spätný chod na predchodcov aktuálneho uzla.

3.3.4 CM-SPADE a CM-SPAM

Medzi najnovšie a veľmi efektívne algoritmy patria algoritmy založené na *prerezávaní priestoru na základe spoločného výskytu* (co-occurence pruning). Sú to algoritmy CM-SPADE a CM-SPAM, predstavené v práci [7]. Ako z ich názvu vyplýva, tieto algoritmy iba rozširujú algoritmy SPADE a SPAM, preto sú v tejto časti iba popísané hlavné rozdiely medzi nimi. Najprv definujme pojem *prefix sekvencie*.

Definícia 3.3.2. Nech je daná sekvencia $\alpha = \langle e_1 e_2 ... e_n \rangle$. Sekvencia $\beta = \langle e'_1 e'_2 ... e'_m \rangle (m \leq n)$ sa nazýva *prefixom sekvencie* α vtedy a len vtedy, keď platí:

1. $e'_i = e_i$ pre $(i \leq m - 1)$,
2. $e'_m \subseteq e_m$,
3. všetky frekventované položky v množine $(e_m - e'_m)$ lexikograficky nasledujú za položkami v množine e'_m.

Ako príklad môžeme uviesť sekvenciu $S = \langle a(abc)(ac)d(cf)\rangle$. Prefixami tejto sekvencie sú napríklad sekvencie $\langle a\rangle$, $\langle aa\rangle$, $\langle a(ab)\rangle$ a $\langle a(abc)\rangle$. Sekvencie $\langle ab\rangle$ a $\langle a(bc)\rangle$ na druhú stranu nie sú prefixami sekvencie S.

Mapa spoločného výskytu

Mapa spoločného výskytu (co-occurence MAP), skrátene CMAP, je štruktúra mapujúca každú položku $k \in I$ (pričom I je množina všetkých položiek vstupnej databázy) na množinu položiek, ktoré za ňou v sekvenciách databáze nasledujú. Tieto algoritmy využívajú dve takéto štruktúry, nazývané $CMAP_i$ a $CMAP_s$. $CMAP_i$ mapuje každú položku k na množinu $cm_i(k)$. Táto množina obsahuje všetky položky $j \in I$ také, ktoré nasledujú položku k v rámci rovnakej udalosti, a to minimálne v takom počte sekvencií, aký je prah minimálnej podpory. Obdobne, $CMAP_s$ mapuje každú položku k na množinu $cm_s(k)$. Tá obsahuje všetky položky $j \in I$ také, ktoré nasledujú položku k v rámci iných udalostí, a to minimálne v takom počte sekvencií, aký je prah minimálnej podpory.

Príklad 3.3.4. Na príklade si môžeme ukázať, ako bude vyzerať takáto množina pre vstupnú databázu 3.1 a prah minimálnej podpory 2 sekvencie. Napríklad položka a je v rámci rovnakých udalostí nasledovaná (v dvoch a viacerých sekvenciách) iba položkou b. Platí teda, že $cm_i(a) = \{b\}$. V rámci iných udalostí je položka a nasledovaná všetkými položkami z I okrem položiek e a g. Platí teda, že $cm_s(a) = \{a, b, c, d, f\}$.

$CMAP_i$	
Položka	Je nasledovaná (i-rozšírenie)
a	{b}
b	{c}
c	∅
d	∅
e	∅
f	∅
g	∅

$CMAP_s$	
Položka	Je nasledovaná (s-rozšírenie)
a	{a,b,c,d,f}
b	{a,c,d,f}
c	{a,b,c,e}
d	{b,c}
e	{a,b,c,f}
f	{b,c}
g	∅

Tabuľka 3.6: Štruktúra $CMAP_i$ a $CMAP_s$ pre vstupnú databázu 3.1 a minimálnu podporu 2 sekvencie.

Prerezávanie priestoru na základe spoločného výskytu

Štruktúra CMAP môže byť použitá na prerezávanie kandidátnych sekvencií na základe nasledujúcich pravidiel [7]:

1. **Prerezávanie i-rozšírení.** Nech existuje sekvenčný vzor S a položka k. Ak existuje v poslednej udalosti sekvencie S položka j taká, že položka k nepatrí množine $cm_i(j)$, tak platí, že i-rozšírenie sekvencie S o položku k nie je frekventované.
2. **Prerezávanie s-rozšírení.** Nech existuje sekvenčný vzor S a položka k. Ak existuje položka $j \in S$ taká, že položka k nepatrí množine $cm_s(j)$, tak platí, že s-rozšírenie sekvencie S o položku k nie je frekventované.

3. **Prerezávanie prefixu.** Spomenuté vlastnosti môžu byť použité na prerezávanie sekvencií začínajúcich konkrétnym prefixom. Nech existuje sekvenčný vzor S a položka k. Ak existuje položka $j \in S$ (resp. j v poslednej udalosti S) taká, že položka $k \notin cm_s(j)$ (resp. $k \notin cm_i(j)$), tak platí, že všetky nadsekvencie S_n, ktorých prefix je S a zároveň položka k nasleduje položku j s-rozšírením (resp. i-rozšírením) nie sú frekventované.

Integrácia do algoritmov SPADE a SPAM

Nech S je sekvencia, ktorá je aktuálne rozširovaná o položku x. Posledná položka sekvencie S nech je označená a. Ak sa položka x nenachádza v množine $cm_s(a)$, resp. $cm_i(a)$, tak výsledný vzor určite nie je frekventovaný. Týmto spôsobom sa algoritmy efektívne vyhýbajú operáciám spojenia a počítania podpory, čo urýchľuje celý proces dolovania.

3.4 Algoritmy založené na expanzii vzoru

Krátku dobu po predstavení prvých apriori algoritmov začali byť evidentné ich hlavné nedostatky. Najväčším z nich je neefektívne generovanie kandidátnych sekvencií, ktorých počet môže v najhoršom prípade narastať exponenciálne. Tieto negatívne vlastnosti sú citeľné najmä pri veľkých vstupných databázach. Algoritmy založené na expanzii vzoru predstavujú riešenie týchto problémov rozdelením vstupnej databázy na menšie celky, v ktorých sú vyhľadávané lokálne frekventované položky. Tieto položky sú potom pripojené k vstupnému prefixu, čím dochádza k expanzii vzoru. Medzi najvýznamnejších predstaviteľov tejto skupiny algoritmov patria algoritmy FreeSpan [10] a jeho nasledovník PrefixSpan [13].

3.4.1 PrefixSpan

Algoritmus PrefixSpan (Prefix-Projected Sequential Pattern Mining) je najvýznamnejší algoritmus z kategórie algoritmov založených na expanzii vzoru. Algoritmus je postavený na koncepte algoritmu FreeSpan a využíva rekurzívne *projektovanie* databázy na základe *prefixov* sekvencií. Vďaka použitiu projektovaných databáz je algoritmus vo väčšine prípadov rýchlejší a pamäťovo úspornejší ako algoritmy založené na apriori vlastnosti. Ďalšou prirodzenou vlastnosťou projektovaných databáz je to, že ich veľkosť každou ďalšou projekciou zmenšuje.

Definícia 3.4.1. Nech je daná sekvencia $\alpha = \langle e_1 e_2 ... e_n \rangle$ a jej prefix $\beta = \langle e_1 e_2 ... e_{m-1} e'_m \rangle$ $(m \leq n)$. Sekvencia $\gamma = \langle e''_m e_{m+1} ... e_n \rangle$ sa nazýva *sufixom sekvencie* α vzhľadom k prefixu β, označované ako $\gamma = \alpha / \beta$, kde $e''_m = (e_m - e'_m)$.

Ako príklad môžme uviesť sekvenciu $S = \langle a(abc)(ac)d(cf) \rangle$. Sekvencia $\langle (abc)(ac)d(cf) \rangle$ je sufixom sekvencie S vzhľadom na prefix $\langle (a) \rangle$, sekvencia $\langle (\_bc)(ac)d(cf) \rangle$ je jej sufix vzhľadom na prefix $\langle aa \rangle$ a $\langle (\_c)(ac)d(cf) \rangle$ je jej sufix vzhľadom na prefix $\langle a(ab) \rangle$.

Pomocou prefixu a sufixu je možné rozdeliť problém dolovania sekvenčných vzorov do nasledujúcej množiny podproblémov [13]:

1. Nech $\{\langle x_1 \rangle, \langle x_2 \rangle, ..., \langle x_n \rangle\}$ je úplná množina jednoprvkových sekvenčných vzorov v sekvenčnej databáze D. Úplnú množinu sekvenčných vzorov v databáze D je možné rozdeliť do n disjunktných podmnožín, pričom i-ta podmnožina $(1 \leq i \leq n)$ je množina sekvenčných vzorov s prefixom $\langle x_i \rangle$.

2. Nech α je sekvenčný vzor o dĺžke l a $\{\beta_1, \beta_2, ..., \beta_n\}$ je množina všetkých $(l+1)$-sekvenčných vzorov s prefixom α. Úplnú množinu sekvenčných vzorov s prefixom α, s výnimkou samotnej sekvencie α, je možné rozdeliť do m disjunktných podmnožín tak, že j-ta podmnožina $(1 \leq j \leq m)$ je množina všetkých sekvenčných vzorov s prefixom β_j.

Na základe týchto tvrdení (ich dôkazy je možné nájsť v [13]) môže byť problém dolovania sekvenčných vzorov rozdelený rekurzívne, teda každá podmnožina sekvenčných vzorov môže byť v prípade potreby ďalej rozdeľovaná. Pre pochopenie algoritmu je na nasledujúcom príklade ukázaný spôsob jeho fungovania [13]. Uvažujme vstupnú databázu 3.1 a prah minimálnej podpory $minsup = 2$, resp. 50 %.

1. Prvým priechodom databázou sú nájdené všetky sekvenčné vzory s dĺžkou jedna. Konkrétne sú to vzory: $\langle a \rangle : 4$, $\langle b \rangle : 4$, $\langle c \rangle : 4$, $\langle d \rangle : 3$, $\langle e \rangle : 3$ a $\langle f \rangle : 3$. Zápis vo forme *„$\langle vzor \rangle$: počet“* reprezentuje sekvenčný vzor a k nemu prislúchajúcu podporu.
2. Úplná množina sekvenčných vzorov je rozdelená do šiestich podmnožín na základe prefixov nasledovne: 1) sekvenčné vzory s prefixom $\langle a \rangle$, 2) sekvenčné vzory s prefixom $\langle b \rangle$, ..., 6) sekvenčné vzory s prefixom $\langle f \rangle$.
3. Podmnožiny sekvenčných vzorov sú nájdené vytvorením korešpondujúcej množiny projektovaných databáz a dolovaním každej z nich rekurzívne nasledujúcim spôsobom:
   (a) V prípade podmnožiny sekvenčných vzorov, ktoré začínajú prefixom $\langle a \rangle$ sú vyhľadané iba tie sekvencie, ktoré ho obsahujú. Do úvahy sú potom brané tie podsekvencie nájdených sekvencií, ktorých prefix vznikol prvým výskytom $\langle a \rangle$. Napríklad v prípade sekvencie $\langle (ef)(ab)(df)cb \rangle$ to je podsekvencia $\langle (\_b)(df)cb \rangle$. Sekvencie v databáze D sa potom projektujú podľa prefixu $\langle a \rangle$, čím vzniká $\langle a \rangle$-*projektovaná databáza*. Táto databáza sa skladá zo štyroch sufixových sekvencií: $\langle (abc)(ac)d(cf) \rangle$, $\langle (\_d)c(bc)(ae) \rangle$, $\langle (\_b)(df)cb \rangle$ a $\langle (\_f)cbc \rangle$.

   Priechodom $\langle a \rangle$-projektovanou databázou zistíme, že jej lokálne frekventované položky sú $a : 2$, $b : 4$, $\_b : 2$, $c : 4$, $d : 2$ a $f : 2$. Sú teda nájdené všetky sekvenčné vzory dĺžky 2, ktorých prefix je $\langle a \rangle$, konkrétne: $\langle aa \rangle : 2$, $\langle ab \rangle : 4$, $\langle (ab) \rangle : 2$, $\langle ac \rangle : 4$, $\langle ad \rangle : 2$ a $\langle af \rangle : 2$.

   Rekurzívnym spôsobom je množina sekvenčných vzorov s prefixom $\langle a \rangle$ rozdelená do nasledovných podmnožín: 1) sekvenčné vzory začínajúce prefixom $\langle aa \rangle$, 2) sekvenčné vzory začínajúce prefixom $\langle ab \rangle$,... a 6) sekvenčné vzory začínajúce prefixom $\langle af \rangle$.

   i. $\langle aa \rangle$-projektovaná databáza pozostáva z dvoch neprázdnych sufixov, a to $\langle (\_bc)(ac)d(cf) \rangle$ a $\langle (\_e) \rangle$. Priechodom $\langle aa \rangle$-projektovanej databázy nie sú nájdené žiadne lokálne frekventované položky, čím je dolovanie v tejto časti databázy ukončené.
   ii. Podobne sú vytvorené $\langle ab \rangle$, $\langle (ab) \rangle$, $\langle ac \rangle$, $\langle ad \rangle$ a $\langle af \rangle$-projektované databázy, ktoré sú dolované rovnakým spôsobom.

   (b) Obdobným spôsobom ako pri prefixe $\langle a \rangle$ je postupované s prefixami $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$, $\langle e \rangle$, $\langle f \rangle$. Najprv sú teda vytvorené ich projektované databázy, ktoré sú potom dolované.
4. Úplná množina sekvenčných vzorov je získaná zlúčením všetkých nájdených sekvenčných vzorov vo vyššie popísanom rekurzívnom procese.

Z popisu algoritmu vyplýva, že PrefixSpan iba rozširuje, resp. expanduje nájdené sekvenčné vzory. Týmto sa efektívne vyhýba nutnosti generovať a testovať kandidátne sekvencie, čím výrazne zmenšuje prehľadávaný priestor. Ďalšou vlastnosťou algoritmu je, že projektované databázy sú vždy menšie ako pôvodné databázy, z ktorých vznikli. Je to spôsobené faktom, že sú do nich projektované iba sufixové podsekvencie frekventovaného prefixu.

Konštrukcia projektovaných databáz však môže byť v niektorých prípadoch pamäťovo nákladná. Riešenie tohto problému predstavuje optimalizácia pomocou tzv. *pseudoprojekcií* [13]. Pri použití tejto techniky sú fyzické projekcie nahradené *ukazovateľom* na pôvodnú sekvenciu a *offsetom*, ktorý určuje pozíciu začiatku projekcie v sekvencii. V prípade $\langle a \rangle$-projekcie sekvencie $s_1 = \langle a(abc)(ac)d(cf) \rangle$ by ukazovateľ na sekvenciu mohol obsahovať jej *id*, s_1, a offset množinu celých čísel $\{2, 3, 6\}$.

3.5 Algoritmy založené na skorom odstraňovaní kandidátov

Skupina algoritmov založená na skorom odstraňovaní kandidátov je najmladšia z troch menovaných skupín. Jej základom je snaha o čo najskoršie prerezanie/odstránenie kandidátnych sekvencií, čím sa minimalizuje čas potrebný na počítanie ich podpory a zmenšuje sa prehľadávaný priestor.

3.5.1 LAPIN

Významný algoritmus z tejto kategórie je algoritmus, LAPIN (Last Position Induction) [16]. Hlavným rozdielom medzi algoritmom LAPIN a predošlými algoritmami je, že LAPIN prehľadáva iba časť celého priestoru. PrefixSpan prehľadáva všetky projektované databázy, SPADE zase dočasne spája IDList reprezentácie kandidátnych sekvencií. Podobne ako SPAM, aj LAPIN využíva na reprezentáciu prehľadávaného priestoru lexikografický strom, ktorého uzly sú rozširované pomocou s a i-rozšírení. Strom je prechádzaný metódou DFS.

ID sekvencie	Sekvencia
s_1	$\langle ac(bc)d(abc)ad \rangle$
s_2	$\langle b(cd)ac(bd) \rangle$
s_3	$\langle d(bc)(ac)(cd) \rangle$

ID sekvencie	Posledná pozícia položky
s_1	$b_{posl} = 5, c_{posl} = 5, a_{posl} = 6, d_{posl} = 7$
s_2	$a_{posl} = 3, c_{posl} = 4, b_{posl} = 5, d_{posl} = 5$
s_3	$b_{posl} = 2, a_{posl} = 3, c_{posl} = 4, d_{posl} = 4$

Tabuľka 3.7: Ukážka vstupnej databázy (a) a tabuľky posledných pozícií jej položiek (b). Prevzaté a upravené z [16].

LAPIN využíva tzv. *indukciu pozície*. Prvým prechodom databázou si algoritmus vytvorí pomocnú tabuľku posledných pozícií položiek jednotlivých sekvencií, ktorá umožňuje skoré odstránenie kandidátnych sekvencií. Jej ukážku je možné vidieť v tabuľke 3.7. Princíp indukcie pozície hovorí, že ak je posledná pozícia položky v sekvencii menšia alebo rovná pozícii poslednej položky aktuálneho prefixu, položka sa nemôže vyskytnúť za týmto prefixom v rámci aktuálnej sekvencie [16]. Vytváranie kandidátnych sekvencií a počítanie ich podpory je ukázané na príklade nižšie 3.5.1.

Príklad 3.5.1. Uvažujme prefix sekvencie $\langle a \rangle$, ktorého pozície v tabuľke 3.7 (a) sú $s_1 : 1$, $s_2 : 3$ a $s_3 : 3$ (zapísané vo formáte $ID\ sekvencie : poradie\ udalosti$). V tabuľke (b) sú

zobrazené všetky prvé položky, ktorých posledné pozície sú väčšie ako pozícia prefixu $\langle a \rangle$. Konkrétne sú to položky $s_1 : b$, $s_2 : c$, $s_3 : c$. Od týchto položiek (vrátane) do konca každého riadku tabuľky (b) je spočítaná podpora každej položky. Podpora jednotlivých položiek bude $a : 1$, $b : 2$, $c : 3$ a $d : 3$. Po s-rozšírení prefixu $\langle a \rangle$ o tieto položky bude podpora vzniknutej sekvencie rovnaká, ako je spočítaná podpora položiek. Teda napríklad vzniknutá sekvencia $\langle aa \rangle$ má podporu 1 (vidíme, že sa nachádza iba v sekvencii s_1) a sekvencia $\langle ab \rangle$ má podporu 2 (nachádza sa v sekvenciách s_1 a s_2). Takmer identicky by vyzeral aj postup pre i-rozšírenie, detailnejší popis je možné nájsť v [16].

Ako je ale možné vidieť, takéto porovnávanie je pomerne náročné. Autori tento problém vyriešili tabuľkou ITEM_IS_EXISTS_TABLE. Tabuľka je vytvorená pre každú sekvenciu vo vstupnej databáze a obsahuje informáciu o poslednej pozícii každej položky v rámci danej sekvencie. Na ukážke 3.8 je možné vidieť takúto tabuľku pre sekvenciu $\langle ac(bc)d(abc)ad \rangle$.

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

Tabuľka 3.8: Tabuľka ITEM_IS_EXISTS_TABLE pre sekvenciu $\langle ac(bc)d(abc)ad \rangle$. Prevzaté z [16].

Prvý stĺpec tabuľky označuje poradie udalosti v sekvencii, záhlavie je tvorené frekventovanými položkami, ktoré boli nájdené prvým prechodom databázou. Hodnota 1 v tabuľke značí, že sa daná položka vyskytuje za korešpondujúcou udalosťou v sekvencii. Naopak bit s hodnotou 0 hovorí, že položka sa za danou udalosťou už nevyskytuje. Ak je teda napríklad pozícia aktuálneho prefixu 5, z tabuľky je prečítaná hodnota vektoru 5, t. j. 1001. To znamená, že sa v rámci danej sekvencie za aktuálnym prefixom nachádzajú iba položky a a d. Podporu kandidátnej sekvencie je potom možné spočítať kontrolou korešpondujúcich vektorov naprieč všetkými sekvenciami v databáze.

Autori predstavili aj optimalizovanú verziu tejto tabuľky pomocou použitia tzv. *kľúčových pozícií*. Kľúčová pozícia je tá, ktorej prislúchajúci vektor je odlišný od predchádzajúceho (okrem vektorov s hodnotami 0). Na príklade tabuľky 3.8 sú to pozície 5 a 6, pretože všetky vektory pred pozíciou 5 majú na každej pozícii jednotky, naopak vektor 7 má na každej pozícii hodnotu nula. Vďaka tejto optimalizácii sú potom jednotlivé tabuľky priestorovo úspornejšie.

Na rozdiel od algoritmu SPAM prehľadáva LAPIN vďaka týmto optimalizáciám iba časť lexikografického stromu. Je ale nutné podotknúť že v množstve prípadov je práve tvorba pomocných štruktúr časovo náročná, a preto môže mať algoritmus v konečnom dôsledku nižšiu výkonnosť.

Kapitola 4

Návrh a implementácia

V tejto kapitole je popísaný návrh a implementácia aplikácie na dolovanie sekvenčných vzorov a zber štatistík o jednotlivých algoritmoch. Sekcia 4.1 popisuje návrh aplikácie a požiadavky na ňu. Takisto je tu popísaná knižnica, ktorá implementuje všetky algoritmy popísané v predošlej časti práce. Sekcia 4.2 sa venuje implementácii aplikácie a zvoleným technológiám.

4.1 Návrh aplikácie

Cieľom aplikácie je experimentálne porovnanie algoritmov dolovania sekvenčných vzorov. Bolo by teda vhodné, aby aplikácia spĺňala nasledujúce požiadavky:

- jednoduché grafické používateľské rozhranie,
- meranie doby behu algoritmu,
- meranie maximálnej spotreby pamäte algoritmom,
- zobrazovanie nájdených sekvenčných vzorov, ako tabuľkové tak aj grafické,
- export výsledkov.

4.1.1 Knižnica SPMF

Pre dolovanie sekvenčných vzorov bola vybratá knižnica SPMF [6], ktorá je v súčasnosti jednou z najpoužívanejších knižníc v oblasti dolovania znalostí. Knižnica je implementovaná v jazyku Java a v dobe písania práce obsahuje 201 algoritmov. Okrem algoritmov pre dolovanie sekvenčných vzorov implementuje aj iné algoritmy pre dolovanie znalostí, napríklad algoritmy pre dolovanie asociačných pravidiel, sekvenčných pravidiel, periodických vzorov atď.

Formát vstupu a výstupu

Vstupom každého algoritmu dolovania sekvenčných vzorov je textový súbor obsahujúci sekvenčnú databázu v SPMF formáte. Formát je definovaný nasledovne:

- každá sekvencia je popísaná na jednom riadku súboru,
- každá položka je reprezentovaná celým kladným číslom a položky sú od seba oddelené medzerou,

- každá udalosť je ukončená číslom -1,
- každá sekvencia je ukončená číslom -2 a znakom nového riadku.

Vstup pre ukážkovú databázu 3.1 môže vyzerať nasledovne:

```
1 -1 1 2 3 -1 1 3 -1 4 -1 3 6 -1 -2
1 4 -1 3 -1 2 3 -1 1 5 -1 -2
5 6 -1 1 2 -1 4 6 -1 3 2 -1 -2
5 7 -1 1 6 -1 3 2 3 -1 -2
```

Formát výstupu (nájdených sekvenčných vzorov) je podobný, navyše je ale na konci každého sekvenčného vzoru pridaná jeho absolútna podpora. Výstup algoritmu PrefixSpan so vstupnou ukážkovou databázou pre minimálnu podporu väčšiu ako 0.75 vyzerá nasledovne:

```
1 -1 #SUP: 4
1 -1 2 -1 #SUP: 4
1 -1 3 -1 #SUP: 4
2 -1 #SUP: 4
3 -1 #SUP: 4
```

Knižnica tiež umožňuje definovať textový popis jednotlivých položiek v sekvenciách. V tomto prípade sú potom položky na výstupe reprezentované týmto popisom a nie celými číslami. Vstupný súbor s popisom položiek musí začínať riadkom `@CONVERTED_FROM_TEXT`. Popis sekvencie s položkami 1, 2 a 3 potom vyzerá napríklad nasledovne:

```
@ITEM=1=the
@ITEM=2=best
@ITEM=3=things
1 2 -1 1 2 3 -1 1 3 -1 -2
```

Grafické používateľské rozhranie

Grafické používateľské rozhranie bude implementované pomocou open-source platformy JavaFX[1]. Hlavnou motiváciou využitia tejto platformy je fakt, že podobne ako Java je aj JavaFX multiplatformová. JavaFX umožňuje definovať grafické rozhranie pomocou značkovacieho jazyka FXML, čo oddeľuje vzhľad a logiku aplikácie. Vďaka tomu je v prípade potreby možné upraviť vzhľad aplikácie bez zásahu do samotnej logiky aplikácie. JavaFX taktiež poskytuje nástroj s názvom Scene Builder[2], ktorý umožňuje efektívnu tvorbu grafického rozhrania metódou drag&drop.

Meranie doby behu a spotreby pamäte

Knižnica SPMF poskytuje aj aplikáciu s grafickým rozhraním, ktorá má však viacero nedostatkov. Najvýraznejším z nich je fakt, že jednotlivé algoritmy nemajú zjednotený spôsob merania spotreby času. Napríklad algoritmy GSP, SPADE a CM-SPADE do celkového času nezapočítavajú načítanie vstupnej databázy zo vstupného súboru do hlavnej pamäte a ani nájdenie frekventovaných položiek (1-sekvencií). V niektorých prípadoch, najmä pri väčších vstupných databázach, tento fakt spôsobuje, že je zistená doba behu algoritmu iba zlomkom skutočného času potrebného na nájdenie sekvenčných vzorov, pretože väčšinu času

[1] https://openjfx.io/
[2] https://gluonhq.com/products/scene-builder/

spotrebuje algoritmus načítaním databázy do hlavnej pamäte a hľadaním frekventované 1-sekvencií.

Ďalší nedostatok je spôsobený samotným programovacím jazykom Java a jeho správou pamäte. Každá aplikácia napísaná v jazyku Java je najprv preložená do Java bajtkódu, ktorý je spúšťaný pomocou virtuálneho stroja – Java Virtual Machine (ďalej iba JVM). Toto umožňuje platformovú nezávislosť Java aplikácií, pretože o jednotlivé rozdiely medzi prostrediami sa už postará JVM. Java spolu s JVM, na rozdiel od jazykov ako napr. C alebo C++, ponúka automatickú správu pamäte, ktorú zabezpečuje *Garbage Collector* (vo voľnom preklade zberač odpadkov). Ten sa stará o uvoľňovanie objektov, na ktoré už neexistuje referencia. To, kedy je Garbage Collector zavolaný má na starosti JVM.

Na zistenie využitia pamäte poskytuje Java triedu `Runtime` a jej metódy `totalMemory()` a `freeMemory()`. Existujúca SPMF aplikácia používa na zistenie využitia pamäte práve tieto prostriedky, ako sa však pri experimentovaní ukázalo, vo veľkom množstve prípadov sa výsledky o spotrebe pamäte môžu výrazne líšiť. Napríklad pri algoritme GSP bola pri jednom behu algoritmu pozorovaná spotreba pamäte 287 MB, pri druhom behu to bolo iba 204 MB a pri treťom behu až 501 MB. Je to spôsobené práve tým, že správa pamäte je odtienená od programátora a úplne prenechaná pod správu JVM.

Pri experimentovaní s knižnicou bola tiež v prípade algoritmu PrefixSpan nájdená chyba v knižnici SPMF. Táto chyba sa prejavuje tak, že v prípade, keď mal naposledy spustený algoritmus väčšiu spotrebu pamäte ako aktuálne bežiaci PrefixSpan, tak je spotreba predošlého algoritmu zaznamenaná aj pre algoritmus PrefixSpan[3].

Navrhovaná aplikácia preto zjednotí spôsob merania času spotrebovaného jednotlivými algoritmami tak, aby bolo pri všetkých algoritmoch brané do úvahy aj načítanie vstupnej databázy a nájdenie frekventovaných 1-sekvencií. Navrhovaná aplikáciu tiež bude poskytovať možnosť automatického viacnásobného opakovania behu algoritmu, čo umožní zistiť priemernú hodnotu spotreby pamäte a času algoritmom, čím sa minimalizujú negatívne dôsledky automatickej správy pamäte. Pred spustením algoritmu bude mať užívateľ možnosť výberu počtu opakovaní behu algoritmu. Aplikácia tiež poskytne aj grafickú reprezentáciu spotreby pamäte a času v jednotlivých behoch algoritmu. Tieto grafy budú implementované pomocou vstavaných funkcií knižnice JavaFX. V neposlednom rade aplikácia tiež vyrieši chybu s nepresným meraním spotreby pamäte pri algoritme PrefixSpan.

Reprezentácia a export výsledkov

Nájdené sekvenčné vzory budú zobrazované v tabuľke vo formáte z ukážky 4.1. Obsah tabuľky bude možné filtrovať jednoduchým textovým filtrom. Tabuľka bude vytvorená pomocou vstavaných funkcií knižnice JavaFX.

Absolútna podpora	Sekvenčný vzor
...	...

Tabuľka 4.1: Tabuľková reprezentácia nájdených sekvenčných vzorov v navrhovanej aplikácii. Obsah tabuľky bude možné filtrovať jednoduchým textovým filtrom.

[3]Túto chybu sa mi podarilo v knižnici SPMF v rámci implementácie opraviť. V novších verziách knižnice by sa už nemala vyskytovať.

Ďalšia funkcia, ktorou bude experimentálna aplikácia disponovať bude grafická vizualizácia sekvenčných vzorov. Grafická vizualizácia bude implementovaná pomocou open-source knižnice JavaFXSmartGraph[4]. Zobrazený graf bude obsahovať iba aktuálny pohľad na tabuľku, teda ak používateľ použil filter na obsah tabuľky, budú v grafe viditeľné iba relevantné sekvenčné vzory.

Výsledky bude tiež možné exportovať buď do textového súboru v SPMF alebo CSV formáte, kde každý riadok bude obsahovať jeden sekvenčný vzor a jeho absolútnu podporu. Rovnako ako pri grafickej vizualizácii bude exportovaný iba aktuálny pohľad na tabuľku.

4.2 Implementácia aplikácie

Aplikácia je implementovaná v jazyku Java, grafické rozhranie je postavené na platforme JavaFX. Pre dolovanie sekvenčných vzorov je použitá knižnica SPMF a výsledné vzory sú graficky zobrazené pomocou knižnice JavaFXSmartGraph. Pre správu závislostí je použitý nástroj Maven[5], ktorý automatizuje výber verzií platformy JavaFX pre rôzne operačné systémy. Adresárová štruktúra so zdrojovými súbormi vyzerá nasledovne:

```
src
└── main
    ├── java
    │   ├── tools
    │   ├── gui
    │   └── libraries
    │       ├── spmf
    │       └── smartgraph
    └── resources
```

Všetky zdrojové súbory sa nachádzajú v priečinku `src`. V jeho podpriečinku `libraries` sa nachádzajú externé knižnice SPMF a JavaFXSmartGraph. V priečinku sa `resources` sa nachádzajú FXML súbory definujúce grafické užívateľské rozhranie a CSS súbory.

4.2.1 Vstup aplikácie

Vstupom aplikácie sú textové súbory v nasledujúcich formátoch:

- **SPMF.** Formát podporovaný algoritmami knižnice SPMF. Nižšie spomenuté formáty sú pred procesom dolovania tohto formátu konvertované.
- **IBM.** Okrem databáz v SPMF formáte budú vykonávané experimenty aj so syntetickými databázami. Na ich generovanie bude používaný program IBMGenerator[6]. Výstupom tohto programu je textový súbor, ktorého formát je rozdielny od SPMF formátu. Na konvertovanie IBM formátu do SPMF formátu je v aplikácii implementovaná trieda `IbmConverter`.
- **TEXT.** Aplikácia takisto podporuje vstupné súbory obsahujúce texty. Pri tomto type vstupu je každá veta vstupného textu uvažovaná ako jedna sekvencia a každé slovo ako samostatná udalosť. Na konvertovanie tohto typu súboru je použitá trieda

[4] https://github.com/brunomnsilva/JavaFXSmartGraph
[5] https://maven.apache.org/
[6] https://github.com/zakimjz/IBMGenerator

`TextConverter`. V tomto prípade sú položky v nájdených sekvenčných vzoroch reprezentované pôvodnými slovami, a nie celými číslami ako je to v prípade SPMF alebo IBM formátu.

Súbory vo formáte IBM alebo TEXT sú metódami príslušných tried konvertované do SPMF formátu a uložené do dočasného súboru, s ktorým aplikácia ďalej pracuje. Dočasné súbory sú vytvárané pomocou vstavaných nástrojov jazyku Java, konkrétne metódy `createTempFile` z triedy `File`. Táto metóda umožňuje vytvárať súbory, ktoré budú automaticky zmazané po ukončení aplikácie, resp. ukončení JVM.

4.2.2 Beh algoritmu a štatistiky

Pred každým spustením algoritmu má užívateľ možnosť zadať počet opakovaní behu algoritmu. Čím viac behov algoritmus vykoná, tým presnejšie budú štatistiky o spotrebe pamäte a času. V priečinku `tools` sa nachádzajú súbory definujúce dve triedy potrebné pre beh algoritmov a zber štatistík.

Prvým je súbor definujúci triedu `RunAlgorithm`, ktorá zapúzdruje algoritmy knižnice SPMF do jednotného rozhrania. Hlavnou úlohou tejto triedy je spúšťať algoritmy a následne uchovať informácie o spotrebe pamäte, času a samotných výsledkoch (nájdených sekvenčných vzoroch). V tejto triede je pri každom spustení tiež obnovená inštancia triedy `MemoryLogger`, čím je maximálna hodnota spotreby pamäte algoritmom je nastavená na 0. Meranie doby behu algoritmu prebieha tak, že je do celkového času započítaný čas potrebný na načítanie vstupnej databázy do hlavnej pamäte, nájdenie frekventovaných 1-sekvencií a nájdenie dlhších sekvenčných vzorov. V prípade formátov IBM a TEXT je súbor najprv konvertovaný do SPMF formátu, pričom čas potrebný na konverziu nie je započítaný do celkového spotrebovaného času.

Druhou dôležitou triedou je trieda `Stats`, ktorej úlohou je uchovávanie a výpočet štatistík o jednotlivých behoch algoritmu, konkrétne:

- spotreba času a pamäte v jednotlivých behoch,
- aritmetický priemer spotreby času a pamäte,
- medián spotreby času a pamäte.

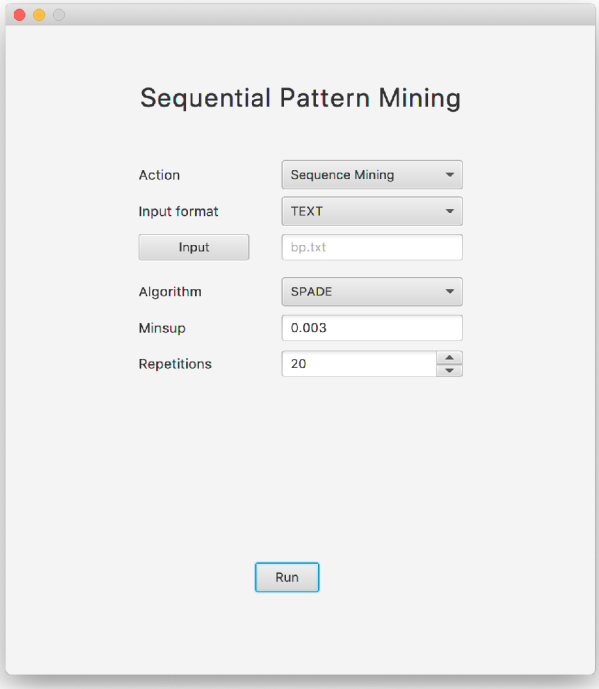
Aplikácia tiež umožňuje zber štatistík o vstupnej databáze. Na tento účel je použitá modifikovaná verzia triedy `SequenceStatsGenerator` knižnice SPMF, do ktorej bola pridaná štatistika o hustote vstupnej databázy. Tá je počítaná ako pomer priemerného počtu položiek v udalostiach a celkového počtu jedinečných položiek vo vstupnej databáze. Sledované vlastnosti vstupnej databázy sú:

- počet sekvencií,
- počet jedinečných položiek,
- priemerný počet jedinečných položiek v sekvencii,
- priemerný počet udalostí v sekvencii,
- priemerný počet položiek v udalosti,
- hustota vstupnej databázy.

4.2.3 Grafické užívateľské rozhranie

Grafické užívateľské rozhranie aplikácie bolo navrhnuté v aplikácii Scene Builder a všetky vygenerované FXML súbory sa nachádzajú v priečinku `resources`. Každý FXML súbor definuje jedno okno aplikácie. V rovnakom priečinku sa tiež nachádzajú CSS súbory, v ktorých je definovaný štýl niektorých častí aplikácie. Triedy definujúce logiku zobrazovaných komponent sa nachádzajú v priečinku `gui`.

Po spustení aplikácie je najprv spustená hlavná metóda `main`, ktorá vytvorí inštanciu triedy `MainController`. Tá sa stará o logiku komponent hlavného okna aplikácie a tiež má na starosti spúšťanie behu algoritmu. Hlavné okno aplikácie je možné vidieť na obrázku 4.1. Prejdením kurzorom nad položky *Minsup* alebo *Repetitions* je zobrazená detailnejšia nápoveda. Obsah hlavného okna sa dynamicky mení podľa vybranej akcie *Action*, kde si môže užívateľ vybrať medzi dolovaním sekvenčných vzorov alebo analýzou vstupnej databázy. V prípade niektorých algoritmov sú tiež zobrazené voliteľné vstupné polia, ktorými môžu byť definované obmedzenia na hľadané sekvenčné vzoru (ako napr. minimálna alebo maximálna dĺžka sekvenčných vzorov).

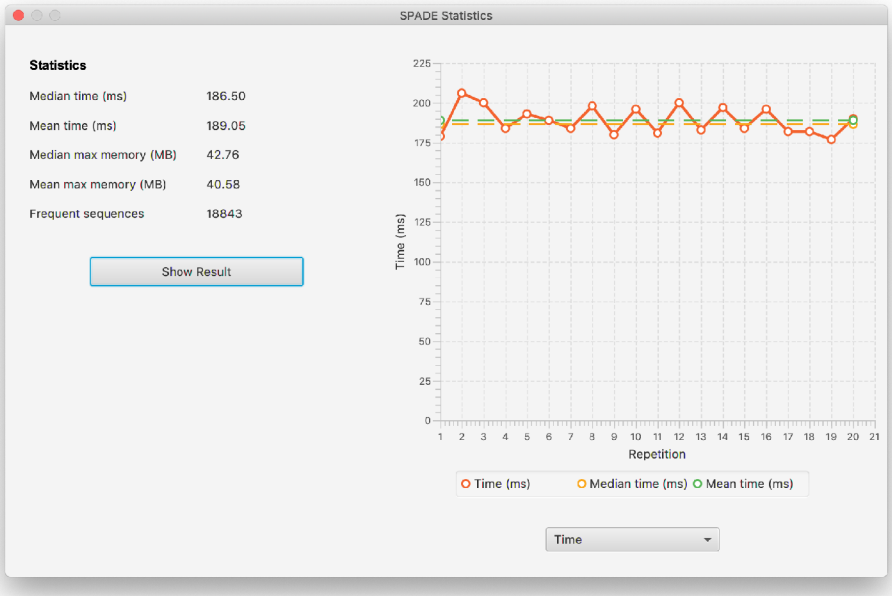


Obr. 4.1: Hlavné okno aplikácie. Jeho obsah sa mení dynamicky podľa vybranej akcie *Action* a zvoleného algoritmu. Po dokončení behu algoritmu je otvorené okno 4.2 so štatistikami.

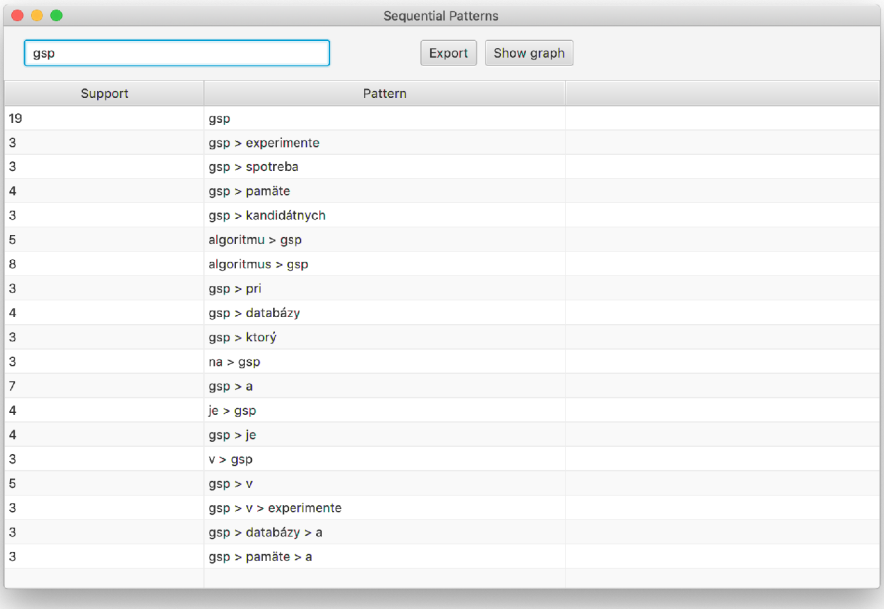
Okno so štatistikami algoritmu spravuje trieda `StatController`. V pravej časti okna sa nachádza graf časovej spotreby alebo pamäťovej spotreby v jednotlivých behoch algoritmu. Tieto grafy je možné meniť pomocou rozbaľovacej ponuky v dolnej časti okna. Po kliknutí na tlačidlo *Show Result* je otvorené okno 4.3 zobrazujúce nájdené sekvenčné vzory.

O zobrazenie nájdených sekvenčných vzorov sa stará trieda `ResultController`. Tabuľková reprezentácia (4.3) obsahuje aj jednoduchý filter, ktorý filtruje všetky sekvencie

obsahujúce zadaný reťazec. Ak je v aktuálnom pohľade na tabuľku maximálne 100 sekvencií, je ich možné zobraziť aj graficky, kliknutím na tlačidlo *Show graph*.



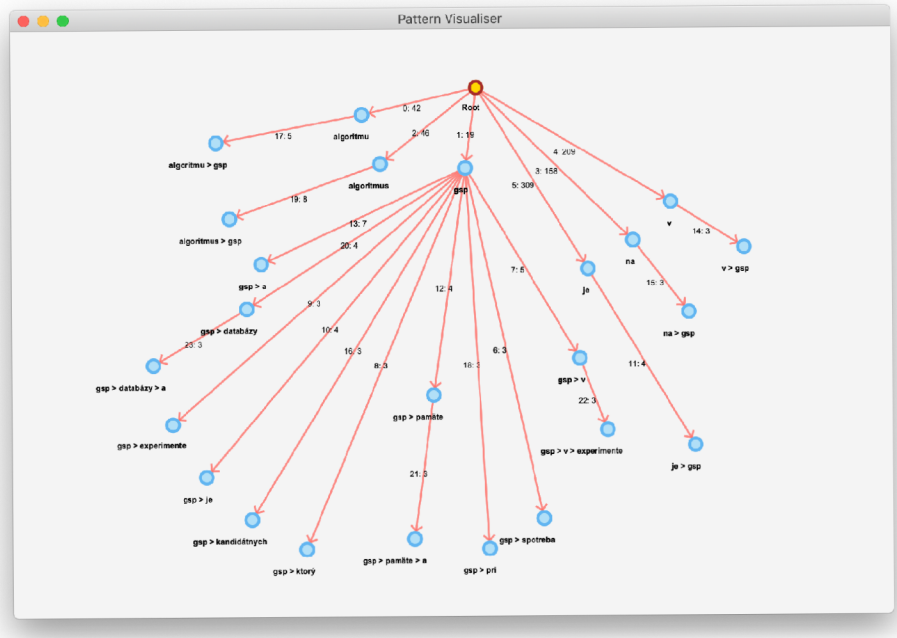
Obr. 4.2: Okno so štatistikami o behu algoritmu. Pomocou rozbaľovacej ponuky v spodnej časti okna je možné zobraziť graf časovej alebo pamäťovej spotreby. Po kliknutí na tlačidlo *Show Result* je otvorené okno 4.3.



Support	Pattern
19	gsp
3	gsp > experimente
3	gsp > spotreba
4	gsp > pamäte
3	gsp > kandidátnych
5	algoritmu > gsp
8	algoritmus > gsp
3	gsp > pri
4	gsp > databázy
3	gsp > ktorý
3	na > gsp
7	gsp > a
4	je > gsp
4	gsp > je
3	v > gsp
5	gsp > v
3	gsp > v > experimente
3	gsp > databázy > a
3	gsp > pamäte > a

Obr. 4.3: Tabuľková reprezentácia nájdených sekvenčných vzorov. Sekvencie je možné filtrovať jednoduchým textovým filtrom alebo exportovať do textového súboru.

Grafická reprezentácia sekvenčných vzorov obsahuje koreňový uzol *Root*. Všetky nadväzujúce uzly obsahujú udalosti jednotlivých sekvencií. Hrany grafu obsahujú názov vo formáte *ID hrany : absolútna podpora sekvenčného vzoru*. Graf je možné zväčšovať alebo zmenšovať, prípadne meniť polohu jednotlivých uzlov.



Obr. 4.4: Grafická reprezentácia sekvenčných vzorov. Graf je možné zväčšovať alebo zmenšovať, prípadne meniť polohu jednotlivých uzlov.

Kapitola 5

Experimenty

Táto kapitola popisuje experimenty vykonávané s algoritmami popísanými v teoretickej časti práce pomocou implementovanej aplikácie. Sekcia 5.1 rozoberá experimenty vykonávané so syntetickými databázami a výkonnosť algoritmov s meniacimi sa parametrami vstupných databáz. Sekcia 5.2 popisuje experimenty vykonávané s reálnymi vstupnými databázami. Tie obsahujú buď prepisy kníh do vhodného formátu alebo sekvencie prístupov k webovým stránkam. Sledované premenné sú spotreba času (v sekundách) a spotreba pamäte (v MB).

Experimenty boli vykonávané na počítači s operačným systémom MacOS 10.14, dvojjadrovým procesorom Intel Core i5 taktovanom na 2.7GHz a pamäťou RAM o veľkosti 8GB (1866MHz LPDDR3). Pre JVM boli vyčlenené 2GB pamäte. V prípade, že pre vstupnú databázu neexistuje výsledok experimentu, znamená to, že algoritmus ukončil svoj beh kvôli nedostatku pamäte.

5.1 Syntetické databázy

Hlavnou motiváciou experimentovania so syntetickými databázami je možnosť modifikácie jednotlivých parametrov. Databázy sú generované pomocou programu IBMGenerator. Pri experimentoch sa vždy mení iba aktuálne sledovaný parameter, pričom ostatné parametre vstupnej databázy a hodnota minimálnej podpory ostávajú nemenné. Sledované parametre experimentov je možné vidieť v tabuľke nižšie 5.1.

Parameter	Popis
D	Počet sekvencií vo vstupnej databáze
N	Počet jedinečných položiek (ovplyvňuje hustotu databázy)
C	Priemerný počet udalostí v sekvenciách

Tabuľka 5.1: Sledované parametre syntetických databáz. Každý experiment je zameraný na jeden z nich, pričom ostatné ostávajú nemenné.

Každý experiment obsahuje tabuľku s informáciami o vstupných databázach algoritmov. Dáta uvedené v tabuľkách sa môžu líšiť pre každú databázu v rámci experimentu v rozsahu jednotiek percent vzhľadom k tomu, že databázy sú generované náhodne. Každý algoritmus bol spustený nad každou vstupnou databázou 5- až 20-krát a do úvahy je braný medián časovej a pamäťovej spotreby.

5.1.1 Experiment 1

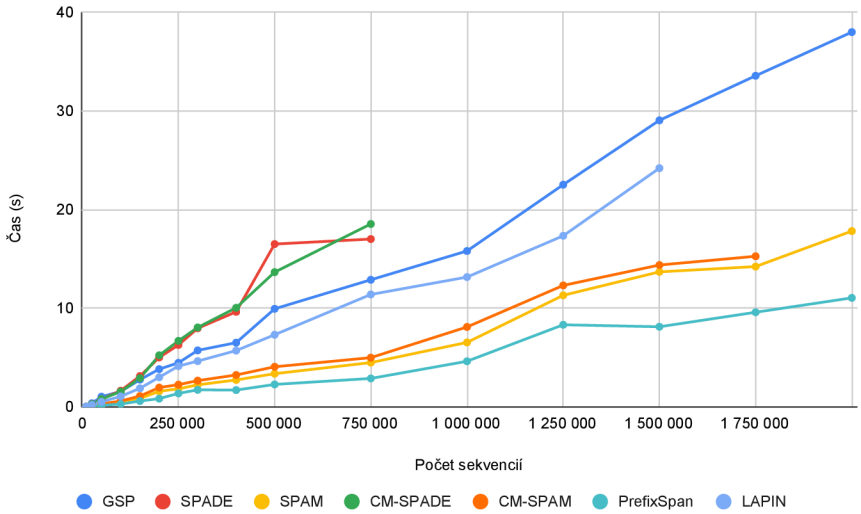
Prvý experiment je zameraný na škálovateľnosť algoritmov s ohľadom na **počet sekvencií vo vstupnej databáze**. Pri generovaní vstupnej databázy ostali všetky parametre okrem počtu sekvencií rovnaké. Parametre vstupných databáz sú zobrazené v tabuľke nižšie 5.2.

Sekvencie	Jedinečné položky	Udalosti v sekvencii	Položky v udalosti
n	1 000	5	4

Tabuľka 5.2: Parametre syntetických databáz experimentu zameraného na zmenu počtu sekvencií. Parameter n sa pre každú vstupnú databázu mení.

Počet sekvencií (n) pre vstupné databázy stúpa od 1 000 až po 2 000 000. Hodnota minimálnej podpory bola nastavená na 0.01, resp. 10 %.

Ako je možné vidieť na grafe 5.1, časová náročnosť rastie pre každý algoritmus lineárne. Zaujímavé je, že algoritmus GSP v tomto experimente prekonáva výkonnosť algoritmov SPADE a CM-SPADE. Toto môže byť pripísané faktu, že všetky sekvenčné vzory v každej databáze sú 1-sekvencie. Nie je teda vytvárané obrovské množstvo kandidátnych sekvencií a pre každú z nich prechádzaná databáza, čo je všeobecne najväčším nedostatkom tohto algoritmu. Na druhú stranu, algoritmy SPADE a CM-SPADE sú spomaľované implementáciou, ktorá nepracuje priamo so vstupnou databázou v textovom súbore, ale obsah tohto súboru je najprv prevedený do internej reprezentácie `SequenceDatabase`. Tento úkon zaberá v rámci tohto experimentu väčšinu času, najmä pri väčších databázach. Implementácia SPAM a CM-SPAM naopak pracuje priamo s obsahom vstupného súboru, čím sa výrazne redukuje čas potrebný na vytvorenie vertikálnej reprezentácie databázy.

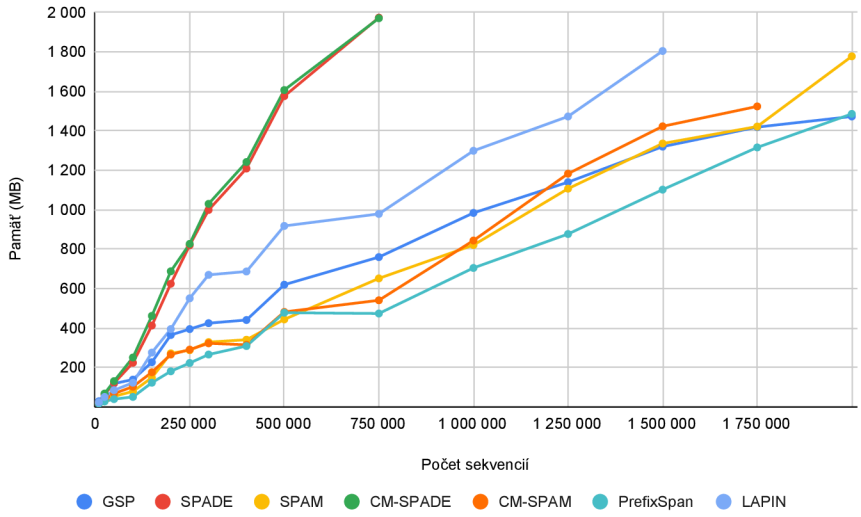


Obr. 5.1: Časová spotreba algoritmov s ohľadom na počet sekvencií vo vstupnej databáze. Ostatné parametre a prah minimálnej podpory ostávajú nemenné.

Na grafe 5.2 môžeme pozorovať, že spotreba pamäte stúpa pre všetky algoritmy, podobne ako spotreba času, lineárne. Najväčšiu spotrebu majú algoritmy SPADE a CM-SPADE, za

čím stojí už spomenuté ukladanie vstupnej databázy do internej reprezentácie. Vysokú spotrebu má tiež LAPIN, najmä kvôli pomocným dátovým štruktúram, ako je napríklad tabuľka ITEM_IS_EXISTS_TABLE. Spotreba algoritmu GSP sa drží relatívne nízko hlavne kvôli malému počtu kandidátnych sekvencií, ktoré sú generované.

Z experimentu vyplýva, že dobrú výkonnosť preukazujú algoritmy SPAM a CM-SPAM. Ako najefektívnejší algoritmus sa javí PrefixSpan, či už z pohľadu výkonnosti, tak aj spotreby pamäte.



Obr. 5.2: Pamäťová spotreba algoritmov s ohľadom na počet sekvencií vo vstupnej databáze. Ostatné parametre a prah minimálnej podpory ostávajú nemenné.

5.1.2 Experiment 2

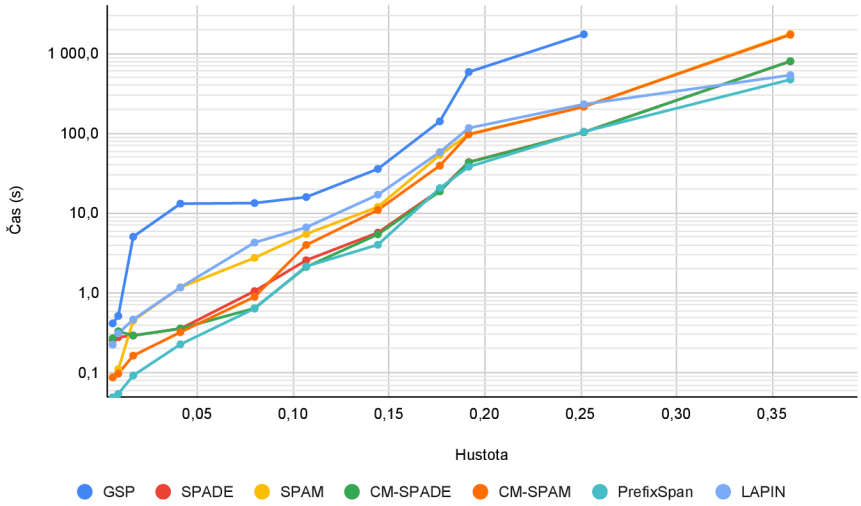
V druhom experimente je pozornosť zameraná na **hustotu vstupnej databázy**. Pripomeňme si, že hustota vstupnej databázy je počítaná ako pomer priemerného počtu položiek v udalostiach a celkového počtu jedinečných položiek v databáze. Hodnota minimálnej podpory bola nastavená na 0.1 (10 %). Hustota vstupných databáz (n) postupne stúpa od 4 % až po 36 %, resp. počet jedinečných položiek v databázach klesá od 905 až po 15. Parametre vstupných databáz sú zobrazené v tabuľke 5.3.

Sekvencie	Jedinečné položky	Udalosti v sekvencii	Položky v udalosti
10 000	n	5	4

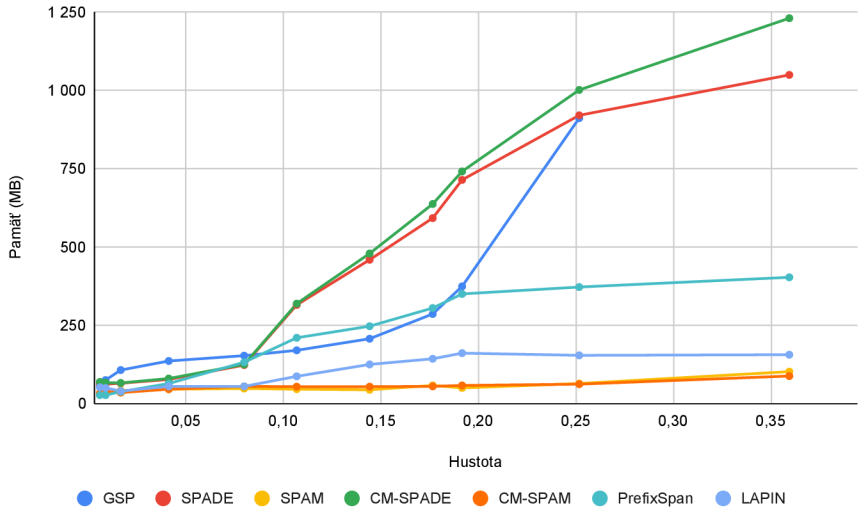
Tabuľka 5.3: Parametre syntetických databáz experimentu zameraného na zmenu hustoty vstupnej databázy. Parameter n sa pre každú vstupnú databázu mení.

Na základe výsledkov experimentu môže byť povedané, že so stúpajúcou hustotou databázy rastie časová náročnosť každého algoritmu exponenciálne (5.3). Je to kvôli tomu, že klesá počet jedinečných položiek, čo spôsobuje (pri stálej hodnote minimálnej podpory), že stúpa počet frekventovaných sekvencií. Najviac evidentné to je pri algoritme GSP,

ktorý musí pre všetky kandidátne sekvencie prechádzať databázu. Za pripomienku stojí, že vstupná databáza obsahuje iba 10 000 sekvencií a pri väčších databázach by bola jeho výkonnosť podstatne nižšia. Výkonnosť algoritmu SPADE a CM-SPADE je podstatne lepšia ako v prvom experimente, práve kvôli relatívne malej vstupnej databáze. V prípade algoritmu LAPIN môžeme pozorovať, že je jeho výkonnosť pre nízke hustoty v porovnaní s ostatnými algoritmami pomerne slabá. Pre databázy s nízkym počtom jedinečných položiek sa už ukazuje efektivita použitia tabuľky posledných pozícií položiek a LAPIN sa dostáva na úroveň algoritmu PrefixSpan. Ten sa však, rovnako ako v prvom experimente, javí ako najefektívnejší algoritmus.



Obr. 5.3: Časová spotreba algoritmov s ohľadom na hustotu vstupnej databázy. Časová osa má logaritmickú stupnicu.



Obr. 5.4: Pamäťová spotreba algoritmov s ohľadom na hustotu vstupnej databázy.

Pri pohľade na graf 5.4 je možné vidieť, že SPADE a CM-SPADE majú podobne ako v prvom experimente najväčšiu spotrebu pamäte. V prípade GSP začína spotreba pamäte narastať vysokým tempom v prípade, keď hustota vstupnej databázy prevyšuje 15 % a je nutné udržiavať explozívne rastúce množstvo kandidátnych položiek v pamäti. Podobne PrefixSpan má v porovnaní s prvým experimentom relatívne k ostatným algoritmom vyššiu spotrebu pamäte, pretože je vytvárané veľké množstvo projektovaných databáz, ktoré sú udržiavané v pamäti. Algoritmy LAPIN, SPAM a CM-SPAM majú nízku spotrebu vďaka uchovávaniu reprezentácie databáze pomocou bitových vektorov, ktorých počet s klesajúcim počtom jedinečných položiek takisto klesá.

Napriek tomu, že väčšina algoritmov má odlišné výsledky ako v experimente 1, PrefixSpan sa aj v tomto experimente ukazuje ako najviac efektívny. Pokiaľ je k dispozícii veľké množstvo pamäte, tiež sú vhodnou voľbou algoritmy SPADE a CM-SPADE, nakoľko majú na základe výsledkov experimentu identickú, v niektorých prípadoch mierne lepšiu, výkonnosť ako algoritmus PrefixSpan. Ak je hustota vstupnej databázy veľmi vysoká (viac ako 30 %) a pracuje sa s obmedzenou pamäťou, stojí za zváženie použitie algoritmu LAPIN, nakoľko má nižšiu spotrebu pamäte, ale takmer identickú výkonnosť ako PrefixSpan, resp. SPADE alebo CM-SPADE.

5.1.3 Experiment 3

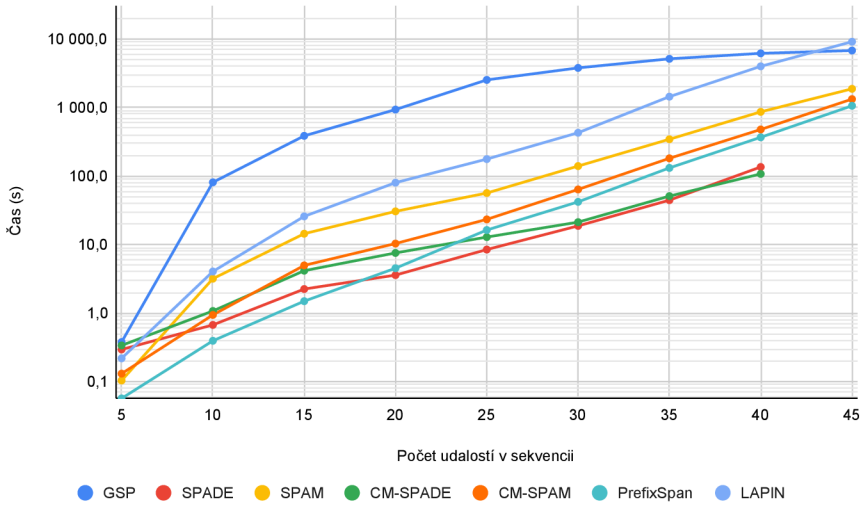
Tretí experiment testuje škálovateľnosť algoritmov s ohľadom na **počet udalostí v sekvenciách**, pričom jednotlivé udalosti obsahujú priemerne rovnaký počet položiek. Minimálna podpora bola nastavená rovnako ako v minulých experimentoch na hodnotu 10 %. Parametre vstupných databáz sa nachádzajú v tabuľke 5.4. Počet udalostí v sekvenciách (n) stúpa pre jednotlivé databázy od hodnoty 5 až po hodnotu 45. Ostatné parametre a prah minimálnej podpory ostávajú nemenné.

Sekvencie	Jedinečné položky	Udalosti v sekvencii	Položky v udalosti
10 000	1 000	n	4

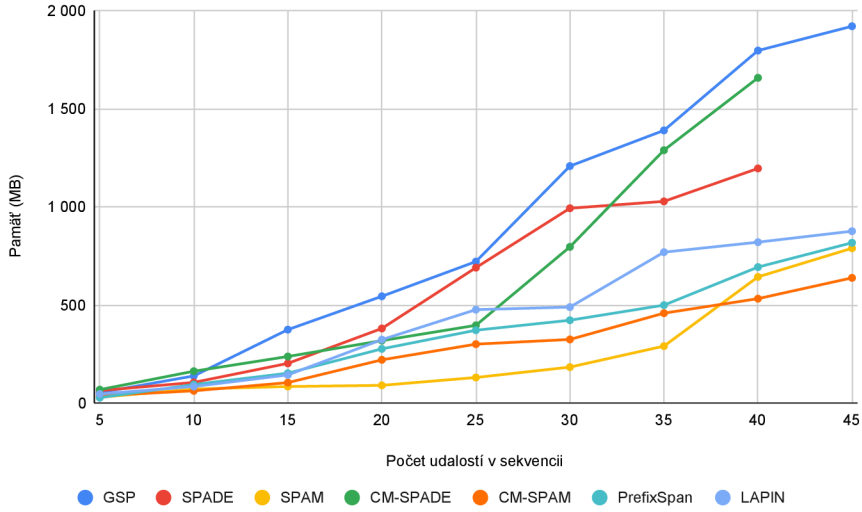
Tabuľka 5.4: Parametre syntetických databáz experimentu zameraného na zmenu počtu udalostí v sekvenciách vstupnej databázy. Parameter n sa pre každú vstupnú databázu mení.

Na základe výsledkov experimentu zobrazených na grafe 5.5 môže byť povedané, že pre sekvencie s malým počtom udalostí je najvýkonnejší opäť algoritmus PrefixSpan. Pre sekvencie s počtom sekvencií väčším ako 20 už začínajú PrefixSpan prekonávať algoritmy SPADE a CM-SPADE. SPAM a CM-SPAM majú pre viac udalostí o takmer rád slabšiu výkonnosť ako SPADE a CM-SPADE. Podobne aj LAPIN ukazuje pomerne slabú výkonnosť, za čím opäť stojí nutnosť prechádzať každú sekvenciu pre zostrojenie tabuľky posledných pozícií. GSP je aj v tomto experimente najmenej výkonný kvôli už spomenutým nedostatkom.

Čo sa týka pamäťovej náročnosti zobrazenej na grafe 5.6, najväčšie množstvo pamäte spotrebúva opäť algoritmus GSP. Podobne vysokú spotrebu majú pre väčší počet udalostí aj algoritmy SPADE a CM-SPADE, ktoré dokonca pre poslednú vstupnú databázu nedokončili svoj beh, pretože vyčerpali všetku dostupnú pamäť. Pamäťovo najúspornejšie sú algoritmy používajúce bitovú reprezentáciu vstupnej databázy spolu s algoritmom PrefixSpan.



Obr. 5.5: Časová spotreba algoritmov s ohľadom na počet udalostí v sekvenciách vstupnej databázy. Časová osa má logaritmickú stupnicu.



Obr. 5.6: Pamäťová spotreba algoritmov s ohľadom na počet udalostí v sekvenciách vstupnej databázy.

Ako najvýkonnejšie algoritmy pre databázy so sekvenciami s veľkým počtom udalostí sa ukazujú algoritmy SPADE a CM-SPADE, ktoré však majú aj veľmi vysokú spotrebu pamäte. Ak je v rámci úlohy limitovaná spotreba pamäte, je vhodné zvoliť algoritmus PrefixSpan alebo CM-SPAM.

5.2 Reálne databázy

Druhá časť experimentovania bola zameraná na reálne databázy, ktoré sa dajú rozdeliť do dvoch hlavných skupín:

1. textové (prepisy kníh do SPMF formátu),
2. sekvencie prístupov k webovým stránkam.

Všetky vstupné databázy sú voľne dostupné z webových stránok knižnice SPMF[1].

Pri experimentoch s reálnymi dátovými sadami bola v každom experimente menená hodnota minimálnej podpory. Pôvodné hodnoty boli nastavené tak, aby bola doba behu algoritmov prijateľná a najdlhšie behy algoritmov sa pohybovali v rádoch hodín. Každý algoritmus bol podobne ako v prvom experimente spustený pri každej databáze 5- až 20-krát.

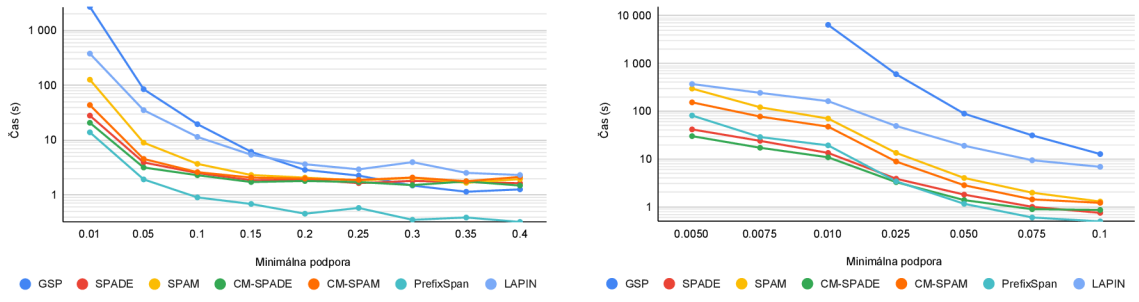
5.2.1 Textové sekvencie

Prvou dvojicou experimentov s reálnymi dátovými sadami sú experimenty s prepismi kníh, konkrétne ide o knihy Biblia a Leviathan. Pri týchto databázach sú slová uvažované ako udalosti s jednou položkou a vety ako jednotlivé sekvencie. Presné detaily vstupných databáz je možné vidieť v tabuľke 5.5.

Názov	Sekvencie	Jedinečné položky	Udalosti v sekvencii	Položky v udalosti
Bible	36 369	13 905	22	1
Leviathan	5 834	9 025	34	1

Tabuľka 5.5: Parametre textových databáz. Parameter udalosti v sekvencii udáva priemerný počet udalostí v sekvenciách.

Na grafe 5.7a je znázornená časová náročnosť algoritmov pre databázu Bible, graf 5.7b zase zobrazuje časovú náročnosť pre databázu Leviathan. Môžeme vidieť, že pri databáze Bible je najvýkonnejší algoritmus PrefixSpan, najmä pri vyšších hodnotách minimálnej podpory. Na druhú stranu v prípade databázy Leviathan prekonávajú PrefixSpan pri nižších hodnotách minimálnej podpory algoritmy CM-SPADE a SPADE. Tento rozdiel je spôsobený rozdielnou veľkosťou vstupných databáz. V oboch databázach si pre nižšie hodnoty minimálnej podpory o celý rád horšie počína algoritmus GSP, za ktorým s podstatne lepším výsledkom nasleduje LAPIN.

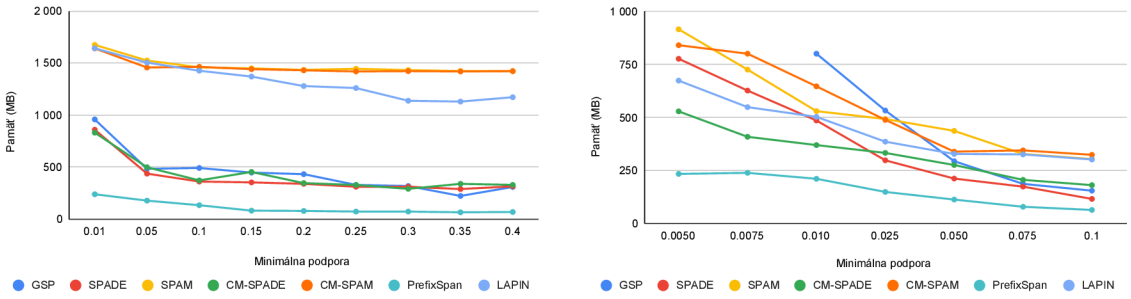


(a) Bible (b) Leviathan

Obr. 5.7: Časová spotreba algoritmov pri databázach Bible a Leviathan a zmene hodnoty minimálnej podpory. Časová osa má logaritmickú stupnicu.

[1] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

Zaujímavý je výsledok porovnania pamäťovej náročnosti algoritmov (5.8). Môžeme vidieť, že pri databáze Bible majú najväčšiu spotrebu algoritmy využívajúce bitovú reprezentáciu sekvencií – SPAM, CM-SPAM a LAPIN. Je to spôsobené veľkým počtom jedinečných položiek v databáze, kvôli čomu majú bitové vektory reprezentujúce databázu veľkú veľkosť. Túto úvahu tiež potvrdzuje experiment 2 (5.1.2), v ktorom mala naopak vstupná databáza veľmi nízky počet jedinečných položiek a pamäťová spotreba bola pri týchto algoritmoch najnižšia. Najlepšie dopadol pri oboch databázach algoritmus PrefixSpan.



(a) Bible

(b) Leviathan

Obr. 5.8: Pamäťová spotreba algoritmov pri databázach Bible a Leviathan a zmene hodnoty minimálnej podpory.

Obe vstupné databázy sú dostupné s popismi položiek, t. j. ku každému číslu vo vstupnej databáze je priradené aj slovo, ktoré je ním reprezentované. Sekvencia s najväčším výskytom v oboch databázach je 1-sekvencia $\langle(the)\rangle$ s podporou takmer 50 % pri databáze Bible a vyše 69 % pri databáze Leviathan. V prípade obidvoch databáz a minimálnej podpore 10 % obsahujú takmer všetky sekvenčné vzory neplnovýznamové slová. Sú to napr. sekvencie ako $\langle(and)(the)\rangle$ alebo $\langle(of)(the)\rangle$. Jediné podstatné mená v nájdených sekvenčných vzoroch sú *lord* pre databázu Bible a *man* pre Leviathan.

5.2.2 Webové sekvencie

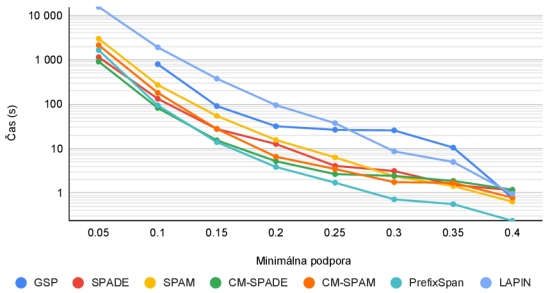
Druhý experiment testuje algoritmy s databázami obsahujúcimi sekvencie prístupov k webovým stránkam. Detaily týchto databáz je možné vidieť v tabuľke 5.6. Podobne ako pri prepisoch kníh, všetky sekvencie v týchto databázach obsahujú udalosti s maximálnou veľkosťou jedna, ktorá reprezentuje konkrétnu webovú stránku, ktorú v danej chvíli navštívil užívateľ. Ako sekvencie sú uvažované postupnosti jednotlivých webových stránok, ktoré používateľ navštívil v určitom časovom okne. Experimenty boli vykonávané s nasledujúcou trojicou databáz:

- Fifa – sekvencie prístupov k webovým stránkam turnaju Majstrovstiev sveta vo futbale 1998,
- Kosarak – dataset so sekvenciami prístupov k webovým stránkam online spravodajskému portálu,
- BMS – sekvencie prístupov na webové stránky elektronického obchodu.

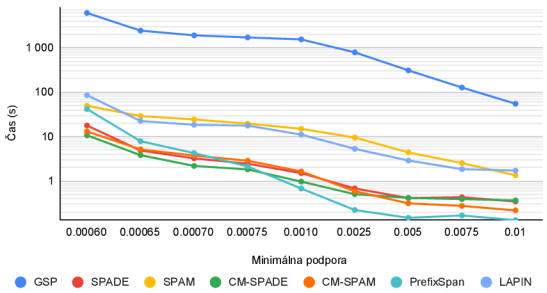
Názov	Sekvencie	Jedinečné položky	Udalosti v sekvencii	Položky v udalosti
FIFA	20 450	2 990	36	1
BMS	59 601	491	2.5	1
Kosarak	25 000	14 804	8	1

Tabuľka 5.6: Parametre databáz sekvencií prístupov k webovým stránkam. Parameter udalosti v sekvencii udáva priemerný počet udalostí vo všetkých sekvenciách.

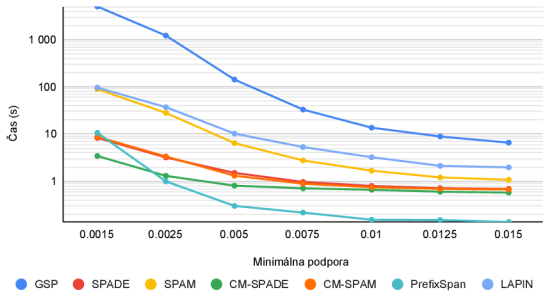
Na základe výsledkov výkonnosti algoritmov zobrazených na grafoch 5.9 môže byť povedané, že pre vyššie hodnoty minimálnej podpory si opäť najlepšie počína PrefixSpan. Pri nízkych hodnotách ho v prípade každej vstupnej databázy prekonáva algoritmus CM-SPADE, pri databáze BMS aj algoritmus SPADE a CM-SPAM. Môžeme tiež pozorovať, že relatívne zle si počínajú algoritmy používajúce bitmapy, konkrétne SPAM a LAPIN. Na druhú stranu algoritmus CM-SPAM si počína aj napriek použitiu bitmáp takmer o rád lepšie ako SPAM či LAPIN, čo v tomto prípade dokazuje efektivitu optimalizácie pomocou štruktúry CMAP.



(a) FIFA



(b) BMS

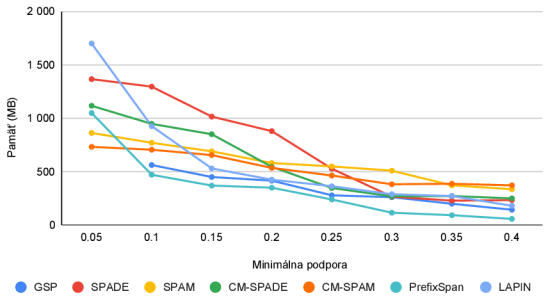


(c) Kosarak

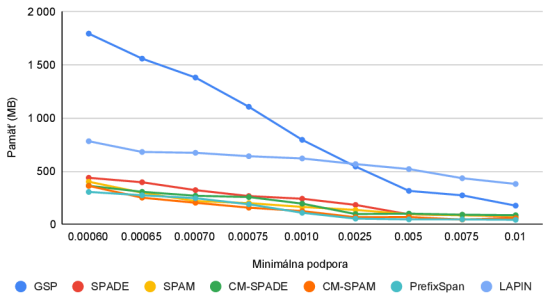
Obr. 5.9: Časová spotreba algoritmov pri databázach FIFA, BMS a Kosarak a zmene hodnoty minimálnej podpory. Časová osa má logaritmickú stupnicu.

Pri porovnaní spotreby pamäte (5.10) si najlepšie pri každej vstupnej databáze počínal opäť PrefixSpan so spotrebou menšou ako 500 MB. Pri databázach BMS a Kosarak mali podobnú spotrebu aj algoritmy CM-SPADE a SPADE. Najviac pamäte spotreboval algoritmus GSP, ktorý pri najmenších hodnotách minimálnej podpory spotreboval cez 1 500 MB pamäte pri každej databáze. Za týmto môže stáť veľké množstvo kandidátnych sekvencií, ktoré sú počas dolovania algoritmom generované. V prípade databázy BMS mal vysokú

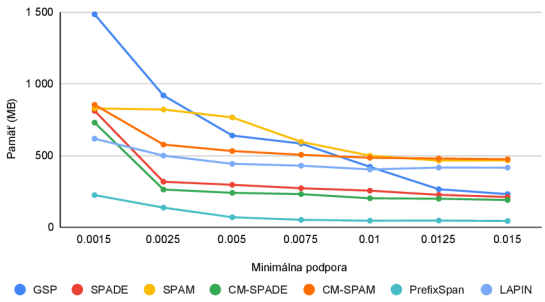
spotrebu pamäte aj algoritmus LAPIN, za čím stojí veľké množstvo sekvencií, pre ktoré je nutné uchovávať posledné pozície položiek.



(a) FIFA



(b) BMS



(c) Kosarak

Obr. 5.10: Pamäťová spotreba algoritmov pri databázach FIFA, BMS a Kosarak a zmene hodnoty minimálnej podpory.

5.3 Zhodnotenie experimentov

Experimenty boli vykonávané s čo najširším spektrom možných vstupných databáz, čo umožnilo objektívne zhodnotenie výhod a nevýhod rôznych metód dolovania sekvenčných vzorov.

Všeobecne je pred samotným dolovaním sekvenčných vzorov vhodné urobiť analýzu vstupnej databázy, ktorá zaberá iba zlomok času potrebného na dolovanie sekvenčných vzorov. Podľa výsledkov analýzy je potom vhodné vybrať algoritmus, čo môže ušetriť najmä pri väčších databázach veľké množstvo času.

Na základe výsledkov experimentov môže byť povedané, že vo väčšine prípadov ponúka algoritmus PrefixSpan veľmi dobrý výkon, pričom má relatívne nízku spotrebu pamäte. To je dosiahnuté najmä vďaka optimalizácii pomocou pseudoprojekcií. Výrazná prevaha algoritmu PrefixSpan je evidentná najmä pri databázach s veľkým množstvom sekvencií, kde spomedzi všetkých algoritmov ponúka najlepšiu ako časovú, tak aj priestorovú škálovateľnosť. Pri databázach s veľkou hustotou stojí za zváženie použitie algoritmu LAPIN, ktorý ponúka takmer identickú výkonnosť ako PrefixSpan, avšak spotrebúva zhruba polovičné množstvo pamäte. Algoritmus GSP bol takmer v každom experimente najmenej efektívny, rovnako aj jeho pamäťová spotreba sa pohybovala vo vysokých hodnotách. Ak obsahuje vstupná databáza veľké množstvo udalostí a nie je výrazne limitované množstvo pamäte, je vhodné siahnuť po algoritme CM-SPADE, resp. SPADE, ktorý mal vo väčšine experimen-

tov podobnú výkonnosť a pamäťovú spotrebu. Algoritmy SPAM a CM-SPAM preukazujú v rámci všetkých experimentov priemernú výkonnosť. Pamäťová spotreba týchto algoritmov závisí najmä od počtu jedinečných položiek vo vstupnej databáze. Ďalšie zaujímavé zistenie z experimentov je, že aj napriek tomu, že algoritmy CM-SPADE a CM-SPAM mali vďaka štruktúre CMAP mierne lepšiu výkonnosť ako SPADE a SPAM, tento rozdiel nebol vo väčšine experimentov veľmi výrazný. Rovnako tak aj pamäťová spotreba sa použitím štruktúry CMAP veľmi nelíšila.

Kapitola 6

Záver

Táto bakalárska práca popisuje a rozoberá metódy dolovania sekvenčných vzorov a jej cieľom bolo zvolené metódy experimentálne porovnať. Na účel ich porovnania bola implementovaná aplikácia, ktorá okrem zberu štatistík o spotrebe pamäte a času jednotlivými algoritmami poskytuje aj vizualizáciu nájdených sekvenčných vzorov, ich filtrovanie a analýzu vstupnej sekvenčnej databázy. Pozornosť bola zameraná na najvýznamnejšie algoritmy dolovania sekvenčných vzorov, ktoré priniesli do tejto oblasti dolovania znalostí nové myšlienky.

Prvá časť práce sa venuje problematike dolovania znalostí, pričom väčšina pozornosti je zameraná na dolovanie sekvenčných vzorov a vysvetlenie základných princípov algoritmov tejto oblasti dolovania dát. V ďalšej časti práce je popísaný návrh a implementácia experimentálnej aplikácie. Posledná časť práce je zameraná na experimenty s algoritmami, pričom je sledovaná časová a priestorová náročnosť jednotlivých algoritmov.

Experimenty boli rozdelené do dvoch hlavných skupín, a to experimenty so syntetickými a reálnymi databázami. Pred samotným dolovaním sekvenčných vzorov je vhodné urobiť analýzu vstupnej databázy a na jej základe vybrať vhodný algoritmus pre dolovaciu úlohu. Na základe výsledkov experimentov môže byť povedané, že všeobecne najvýkonnejší algoritmus pre dolovanie sekvenčných vzorov je algoritmus PrefixSpan. V prípadoch hustých databáz je tiež vhodné použiť algoritmus LAPIN, ktorý má takmer identickú výkonnosť ako PrefixSpan, no zhruba polovičnú spotrebu pamäte. Ak obsahuje vstupná databáza veľké množstvo udalostí v sekvenciách a nie je výrazne obmedzené množstvo pamäte, je vhodné použiť algoritmus CM-SPADE.

Dolovanie sekvenčných vzorov je iba malá oblasť dolovania znalostí. Prácu je ďalej možné rozšíriť o dolovanie uzavretých sekvenčných vzorov, pre ktoré platí, že sú frekventované a nijaká z ich nadsekvencií nemá rovnakú podporu. Ďalej je možné prácu rozšíriť o algoritmy pre dolovanie sekvenčných pravidiel. Sekvenčné pravidlá obsahujú na rozdiel od sekvenčných vzorov okrem prahu minimálnej podpory aj prah spoľahlivosti, ktorého význam je rovnaký ako v prípade asociačných pravidiel. Získané sekvenčné pravidlá je potom možné použiť na predpoveď budúcich hodnôt.

Literatúra

[1] Agrawal, R. a Srikant, R. Mining sequential patterns. In: *Proceedings of the Eleventh International Conference on Data Engineering*. 1995, s. 3–14. DOI: 10.1109/ICDE.1995.380415. ISBN 0-8186-6910-1.

[2] Agrawal, R. a Srikant, R. Mining sequential patterns: Generalizations and performance improvements. In: Apers, P., Bouzeghoub, M. a Gardarin, G., ed. *Advances in Database Technology – EDBT '96*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, s. 1–17. ISBN 978-3-540-49943-5.

[3] Ayres, J., Flannick, J., Gehrke, J. a Yiu, T. Sequential PAttern Mining Using a Bitmap Representation. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: Association for Computing Machinery, 2002, s. 429–435. KDD '02. DOI: 10.1145/775047.775109. ISBN 158113567X.

[4] Deng, N., Chen, X., Li, D. a Xiong, C. Frequent Patterns Mining in DNA Sequence. *IEEE Access*. 2019, zv. 7, s. 108400–108410. DOI: 10.1109/ACCESS.2019.2933044. ISSN 2169-3536.

[5] Fan, Y., Ye, Y. a Chen, L. Malicious sequential pattern mining for automatic malware detection. *Expert Systems with Applications*. 2016, zv. 52, s. 16–25. DOI: 10.1016/j.eswa.2016.01.002. ISSN 0957-4174.

[6] Fournier Viger, P., Lin, C., Gomariz, A., Gueniche, T., Soltani, A. et al. The SPMF Open-Source Data Mining Library Version 2. In: Berendt, B., Bringmann, B., Fromont, É., Garriga, G., Miettinen, P. et al., ed. *Proceedings. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III*. Springer LNCS 9853, 2016, s. 36–40. ISBN 978-3-319-46131-1.

[7] Fournier Viger, P., Gomariz, A., Campos, M. a Thomas, R. Fast vertical mining of sequential patterns using co-occurrence information. In: Tseng, V. S., Ho, T. B., Zhou, Z.-H., Chen, A. L. P. a Kao, H.-Y., ed. *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Cham: Springer International Publishing, 2014, s. 40–52. ISBN 978-3-319-06608-0.

[8] Fournier Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S. a Thomas, R. A survey of sequential pattern mining. *Data Science and Pattern Recognition*. 2017, zv. 1, č. 1, s. 54–77.

[9] Han, J. *Data mining : concepts and techniques*. 3rd ed. Waltham: Morgan Kaufmann, 2012. Morgan Kaufmann series in data management systems. ISBN 978-0-12-381479-1.

[10] Han, J., Pei, J., Mortazavi Asl, B., Chen, Q., Dayal, U. et al. 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: Association for Computing Machinery, 2000, s. 355–359. KDD '00. DOI: 10.1145/347090.347167. ISBN 1581132336.

[11] Kinnebrew, J. S., Loretz, K. M. a Biswas, G. A contextualized, differential sequence mining method to derive students' learning behavior patterns. *Journal of Educational Data Mining.* ERIC. 2013, zv. 5, č. 1, s. 190–219.

[12] Mabroukeh, N. R. a Ezeife, C. I. A Taxonomy of Sequential Pattern Mining Algorithms. *ACM Computing Surveys.* New York, NY, USA: Association for Computing Machinery. 2011, zv. 43, č. 1. DOI: 10.1145/1824795.1824798. ISSN 0360-0300.

[13] Pei, J., Han, J., Mortazavi Asl, B., Wang, J., Pinto, H. et al. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering.* 2004, zv. 16, č. 11, s. 1424–1440. DOI: 10.1109/TKDE.2004.77.

[14] Salehi, M., Kamalabadi, I. N. a Ghoushchi, M. B. G. Personalized recommendation of learning material using sequential pattern mining and attribute based collaborative filtering. *Education and Information Technologies.* Springer. 2014, zv. 19, č. 4, s. 713–735.

[15] Wright, A. P., Wright, A. T., McCoy, A. B. a Sittig, D. F. The use of sequential pattern mining to predict next prescribed medications. *Journal of Biomedical Informatics.* 2015, zv. 53, s. 73–80. DOI: 10.1016/j.jbi.2014.09.003. ISSN 1532-0464.

[16] Yang, Z., Wang, Y. a Kitsuregawa, M. LAPIN: Effective Sequential Pattern Mining Algorithms by Last Position Induction for Dense Databases. In: Kotagiri, R., Krishna, P. R., Mohania, M. a Nantajeewarawat, E., ed. *Advances in Databases: Concepts, Systems and Applications.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 1020–1023. ISBN 978-3-540-71703-4.

[17] Zaki, M. J. Sequence Mining in Categorical Domains: Incorporating Constraints. In: *Proceedings of the Ninth International Conference on Information and Knowledge Management.* New York, NY, USA: Association for Computing Machinery, 2000, s. 422–429. CIKM '00. DOI: 10.1145/354756.354849. ISBN 1581133200.

[18] Zaki, M. J. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning.* Springer Science and Business Media LLC. 2001, zv. 42, 1/2, s. 31–60. DOI: 10.1023/a:1007652502315.

[19] Zendulka, J., Bartík, V., Lukáš, R. a Rudolfová, I. *Získávaní znalostí z databází. Studijní opora.* Fakulta informačních technologií VUT v Brně, 2009.

Príloha A

Obsah priloženého pamäťového média

Súčasťou tejto práce je aj pamäťové médium s nasledujúcim obsahom:

- `app` – adresár obsahujúci zdrojové súbory aplikácie,
- `bin` – adresár so spustiteľnými verziami aplikácie,
- `data` – adresár s experimentálnymi databázami,
- `tex` – adresár obsahujúci zdrojové súbory bakalárskej práce v LaTeXu,
- `xfeket00.pdf` – bakalárska práca vo formáte `pdf`,
- `README.md` – dokumentácia s návodom na preklad a spustenie aplikácie.