VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

METÓDY DOLOVANIA SEKVENČNÝCH VZOROV

METHODS FOR MINING SEQUENTIAL PATTERNS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **IGOR MIKULA**
AUTHOR

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

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

164137

Ústav: Ústav informačních systémů (UIFS)
Student: **Mikula Igor**
Program: Informační technologie
Název: **Metody dolování sekvenčních vzorů v datech**
Kategorie: Data mining
Akademický rok: 2024/25

Zadání:

1. Seznamte se s problematikou získávání znalostí, zejména se zaměřte na problematiku dolování sekvenčních vzorů.
2. Zvolte vhodné datasety, na nichž by se daly ověřovat vlastnosti jednotlivých metod dolování sekvenčních vzorů. Volbu datasetů konzultujte s vedoucím.
3. Prostudujte existující knihovny zahrnující dolování sekvenčních vzorů a navrhněte experimentální aplikaci, která bude toto dolování provádět.
4. Dle návrhu implementujte navrženou aplikaci.
5. Proveďte experimenty, které vyhodnotí vlastnosti jednotlivých metod a výsledky experimentů zdokumentujte.
6. Zhodnoťte dosažené výsledky a další možnosti pokračování tohoto 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.

Při obhajobě semestrální části projektu je požadováno:
Body 1-3.

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: 1.11.2024
Termín pro odevzdání: 14.5.2025
Datum schválení: 22.10.2024

Abstrakt

Cieľom tejto práce je oboznámiť čitateľa s problematikou dolovania sekvenčných vzorov, priblížiť jej praktické využitia a poukázať na aspekty, ktoré túto úlohu robia náročnou z hľadiska výpočtovej zložitosti. Súčasťou práce je aj návrh a implementácia aplikácie určenej na dolovanie sekvenčných vzorov, ako aj experimentálne porovnanie vybraných algoritmov používaných v tejto oblasti.

Abstract

The aim of this thesis is to introduce the reader to the topic of sequential pattern mining, to present its practical applications, and to highlight aspects that make this task challenging in terms of computational complexity. The thesis also includes the design and implementation of an application dedicated to sequential pattern mining, as well as an experimental comparison of selected algorithms commonly used in this field.

Kľúčové slová

dolovanie sekvenčných vzorov, dolovanie dát, získavanie znalostí z databáz, dáta, predspracovanie dát, skladovanie dát, prediktívne dolovacie úlohy, deskriptívne dolovacie úlohy, minimálna podpora, sekvencia, GSP, PrefixSpan, SPAM, SPADE, CM-SPAM, CM-SPADE, LAPIN, FAST

Keywords

sequence pattern mining, data mining, knowledge discovery in databases, data, data preprocessing, data storing, predictive data mining tasks, descriptive data mining tasks, minimal support, sequence, GSP, PrefixSpan, SPAM, SPADE, CM-SPAM, CM-SPADE, LAPIN, FAST

Citácia

MIKULA, Igor. *Metódy dolovania sekvenčných vzorov*. Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Metódy dolovania sekvenčných vzorov

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.

.....
Igor Mikula
14. mája 2025

Poďakovanie

Za pomoc pri zabezpečení odbornej literatúry, usmernenie počas tvorby práce a hodnotné praktické odporúčania ďakujem vedúcemu tejto práce, pánovi Ing. Vladimírovi Bartíkovi, Ph.D.

Obsah

Kapitola 1

Úvod

V súčasnosti vznikajú a denne sa ukladajú rádovo terabajty až petabajty nových dát. V snahe získať z týchto dát hodnotné poznatky vznikla disciplína nazývaná dolovanie dát (angl. *data mining*), ktorá sa zaoberá automatickým hľadaním zaujímavých vzorov, modelov a súvislostí.

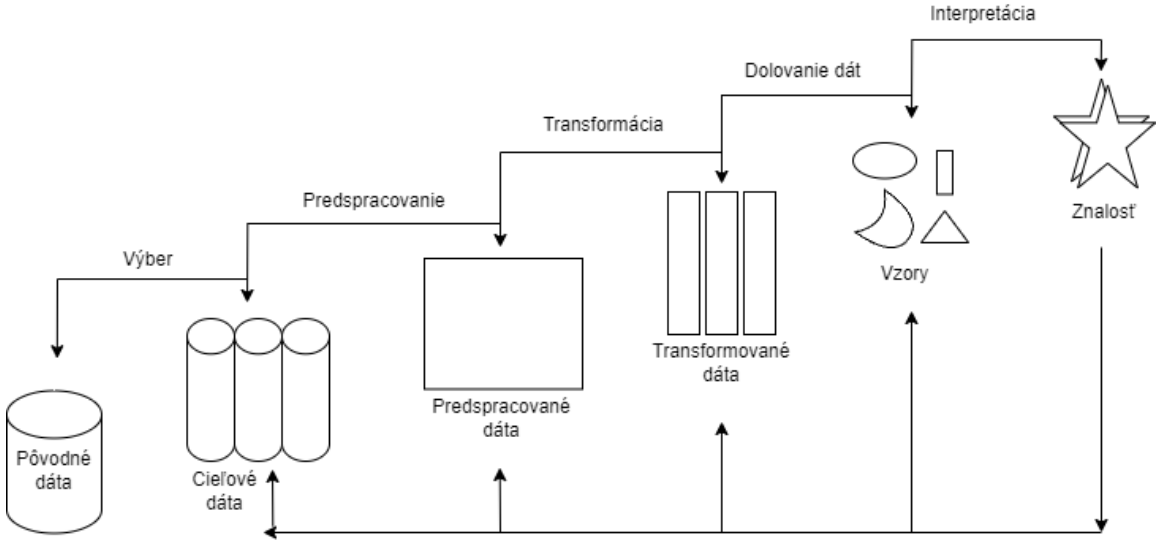
Dolovanie dát je kľúčovou súčasťou širšieho procesu získavania znalostí z databáz (angl. *Knowledge Discovery in Databases*, KDD), ktorý okrem samotnej analýzy zahŕňa aj prípravu dát, ich čistenie, výber vhodného modelu a interpretáciu výsledkov. Tento proces spája poznatky z oblastí ako štatistika, databázové systémy, strojové učenie a iných. Ilustrácia fungovania tohoto procesu je zobrazená na obrázku 1.1

Jednou z významných poddisciplín dolovania dát je dolovanie sekvenčných vzorov (angl. *sequential pattern mining*). Jeho cieľom je identifikovať opakujúce sa vzory v dátach, ktoré zachovávajú časové usporiadanie prvkov. Dolovanie sekvenčných vzorov nachádza uplatnenie v praxi napríklad pri analýzach nákupného správania zákazníkov či biologických procesov.

Táto práca sa zameriava práve na oblasť dolovania sekvenčných vzorov. Jej cieľom je predstaviť základné pojmy, princípy a algoritmy používané pri analýze sekvenčných dát, pričom sa sústreďuje najmä na porovnanie efektivity jednotlivých prístupov.

Úvodná kapitola stručne predstavuje základné pojmy a techniky z oblastí dolovania dát a získavania znalostí z databáz. Druhá kapitola práce sa venuje teoretickému popisu základných pojmov v oblasti dolovania sekvenčných vzorov, ako aj vysvetleniu fungovania základných a často používaných algoritmov pre dolovanie sekvenčných vzorov.

Ďalšie kapitoly obsahujú aj popis implementovanej aplikácie s grafickým rozhraním, ktorá umožňuje vykonávať analýzu dát pomocou rôznych algoritmov na dolovanie sekvenčných vzorov a rozbor praktických experimentov zameraný na porovnanie výkonnosti a efektivity algoritmov.



Obr. 1.1: Proces získavania znalostí z databáz

Kapitola 2

Dolovanie dát

V tejto kapitole sú popísané základné pojmy a techniky používané v kontexte dolovania dát. Sekcia 2.1 zoznamuje čitateľa s pojmom dáta, Sekcia 2.2 sa zaoberá spôsobmi, akými sú dáta skladované a Sekcia 2.3 stručne popisuje základné úlohy používané pre analýzu v oblasti dolovani dát.

2.1 Dáta

Dáta predstavujú hodnoty, ktoré samy o sebe nemajú informačný význam, no po ich zaradení do kontextu a správnej interpretácii hodnôt nám môžu poskytnúť užitočné informácie. Pre účely analýzy je dôležité vedieť aké typy hodnôt môžu dáta obsahovať, čím sa zaoberá Sekcia 2.1.1 a ako dáta vhodne pripraviť pre analýzu, čo je popísané v Sekcii 2.1.2.

2.1.1 Dátové atribúty

V dátovej analýze pracujeme s objektmi a atribútmi. Objekt môžeme chápať ako entitu, ktorú chceme analyzovať, môže ním byť zákazník, produkt alebo transakcia. Na to, aby sme objekt mohli opísať a ďalej s ním pracovať, potrebujeme údaje o jeho vlastnostiach. Tieto vlastnosti sú popísané atribútmi. Každý z atribútov nesie konkrétnu hodnotu, ktorá z určitého hľadiska objekt charakterizuje.

Cieľom tejto sekcie je predstaviť základné typy atribútov podľa toho, akými hodnotami môžu byť vyjadrené. Ich správne rozlíšenie je dôležité pre výber vhodných metód analýzy a transformácie dát.

Nominálne atribúty

Nominálne atribúty sú taktiež známe pod pojmom kategorické atribúty. Ich hodnotami sú symboly alebo mená vecí, pričom každá hodnota reprezentuje nejaký druh, stav, či kategóriu a v informatike sú tieto hodnoty známe pod pojmom enumerácie [7].

Predpokladajme, že typ vozidla a farba interiéru sú dva atribúty opisujúce objekt vozidlo. Hodnoty pre typ vozidla môžu byť sedan, kabriolet, SUV a dodávka. Atribút farba interiéru môže nadobúdať hodnoty čierna, biela, červená a béžová. Oba atribúty, typ vozidla aj farba interiéru, označujeme ako nominálne. Hodnoty nominálnych atribútov môžu byť reprezentované aj číslami, ako napríklad keby zvolíme číselné kody pre jednotlivé farby interiéru – 0 pre čiernu, 1 pre bielu, 2 pre červenú a 3 pre béžovú, podstatné však je, že tieto

číselné hodnoty nepoužívame kvantitatívne, respektíve nad takýmito číselnými hodnotami neprevádzame matematické operácie ako sčítanie či odčítanie.

Binárne atribúty

Binárny atribút možno definovať ako nominálny atribút obsahujúci iba 2 možné hodnoty, t. j. typicky 0 alebo 1. Pri binárnych atribútoch tiež rozlišujeme či sa jedná o symetrický alebo asymetrický binárny atribút. Ak sú oba stavy rovnako hodnotné, teda nezáleží, ktorý zo stavov bude zakódovaný ako 0 a ktorý ako 1, potom hovoríme, že takýto atribút je symetrický [7].

Príkladom symetrického binárneho atribútu by mohol byť stav vozidla, keby jeho možné stavy boli v pohybe a zastavené. Naopak, ak stavy nie sú rovnako hodnotné, hovoríme, že takýto binárny atribút je asymetrický. Ako príklad možno uviesť pozitívny alebo negatívny výsledok testu na prítomnosť jedu v potravine. Typicky je za dôležitejšiu považovaná tá hodnotu, ktorá je vzácnejšia, teda v tomto prípade, že sa v potravine jed skutočne nachádza a kódujeme ju typicky číslom 1, zatiaľ čo informácia o neprítomnosti jedu v potravine nie je tak dôležitá a kódujeme ju typicky číslom 0.

Ordinálne atribúty

O atribúte hovoríme, že je ordinálny, v prípade že jeho hodnoty majú medzi sebou zmysluplné poradie alebo hierarchiu, avšak nie sme schopní presne určiť veľkosť rozdielu medzi jednotlivými hodnotami [7].

Predpokladajme, že by sme chceli popísať objekt vozidlo pomocou atribútu velkosť vozidla, ktorého jedinými možnými hodnotami by boli: malé, stredné a veľké. Z príkladu sme schopní určiť, že stredne veľké vozidlo bude väčšie ako malé vozidlo, nie sme však na základe týchto hodnôt schopní presne povedať o koľko väčšie vozidlo skutočne je.

Číselné atribúty

Hodnotami číselných atribútov sú čísla, nad ktorými možno prevádzať matematické operácie. Číselné atribúty ďalej delíme na intervalovo škálovatelné a pomerovo škálovatelné.

Intervalovo škálovatelné číselné atribúty sú tie, ktorých hodnoty sú usporiadané a môžu byť kladné, záporné alebo nulové. Dôležité je však uvedomiť si, že hodnota nula pri týchto atribútoch nehovorí o neexistencii vlastnosti, pretože tieto atribúty nemajú prirodzený nulový bod. Preto pri týchto atribútoch nemôžeme tvrdiť, že jedna hodnota je násobkom druhej. Napriek tomu sme schopní hodnoty porovnávať a určiť rozdiel medzi nimi [7].

Ako príklad môžeme uviesť teplotu motora vozidla. Pri porovnávaní teploty motorov dvoch vozidiel možno určiť o koľko stupňov je motor prvého vozidla chladnejší alebo teplejší ako motor druhého vozidla. Nemožno však o motore, ktorého teplota je nula stupňov povedať, že ide o motor, ktorý nemá teplotu.

Pomerovo škálovatelné číselné atribúty majú prirodzený nulový bod a o hodnote takéhoto atribútu môžme hovoriť ako o násobku inej hodnoty. Hodnoty sú usporadané a môžeme vypočítať ich rozdiel, priemer, medián aj modus.

Ako príklad uveďme spotrebu paliva vozidla. Ak vozidlo 1 spotrebuje 15 litrov na 100 kilometrov a vozidlo 2 spotrebuje 5 litrov na 100 kilometrov, môžme povedať, že vozidlo 1 má 3-násobnú spotrebu oproti vozidlu 2.

2.1.2 Predspracovanie dát

Predspracovanie dát je proces, pri ktorom sa surové údaje zbavujú redundancií, dopĺňajú sa chýbajúce hodnoty a upravuje sa ich štruktúra tak, aby bola analýza efektívna a jej výsledky spoľahlivé. Tento proces sa zvyčajne skladá z troch hlavných fáz: čistenia, integrácie a transformácie. Táto sekcia obsahuje popis najčastejšiích metód, ktoré sa v jednotlivých fázach predspracovania používajú.

Čistenie dát

Čistenie dát je typicky prvým krokom v procese predspracovania dát. Pre surové dáta je typické, že obsahujú nedostatky, ktorými sú napríklad chýbajúce hodnoty, šum, odľahlé hodnoty, nekonzistencie a podobne. Keby tieto nedostatky zostali neošetrené, mohli by byť výsledky analýzy zavádzajúce alebo nepresné [6]. Najčastejšie problémy, ktoré sa snažíme vyriešiť pomocou čistenia dát sú:

Chýbajúce hodnoty, resp. hodnoty, ktoré sú v danom zázname vynechané.
Riešenia:

1. **Ignorovanie záznamu** je jednoduchý a intuitívny spôsob riešenia, kedy záznam, ktorý neobsahuje všetky požadované hodnoty bude ignorovaný, alebo prípadne úplne zmazaný. Nevýhodou je, že tento prístup v praxi možno použiť iba v prípade, že hodnoty chýbajú iba v malej časti záznamov, kedy predpokladáme, že by odignorovanie zopár záznamov nemalo mať dopad na výsledok analýzy.

2. **Nahrádzanie konštantou**, kedy sa zvolí jednotná a vhodná konštanta, typicky „?“ alebo „N/A“ (z angl. Not Answered), ktorou budú nahradené chýbajúce hodnoty všetkých záznamov.

3. **Nahrádzanie priemerom alebo mediánom** možno použiť v prípade, že nahrádzané hodnoty sú číselného charakteru. Logika za touto metódou je, že priemerné hodnoty nebudú mať štatistický vplyv na výslednú analýzu.

4. **Predikcia**, kedy s pomocou využitia modelov, ktorými môžu byť napríklad rozhodovacie stromy, možno na základe ostatných atribútov záznamu pomerne presne odhadnúť, akou hodnotou by sa mala chýbajúca hodnota nahradiť.

5. **Nahrádzanie najčastejšou hodnotou** je prakticky použiteľná technika, kedy do záznamu s chýbajúceho záznamu vložíme najčastejšie sa opakujúcu hodnotu.

Šum v kontexte čistenia dát predstavuje menšie odchýlky v dátach. Môže byť spôsobený meracími chybami alebo náhodnými vplyvmi a tak neodrážať skutočnú povahu údajov.
Riešenia:

1. **Klastrovanie** (Clustering) je technika, ktorej cieľom je usporiadať údaje, ktoré sú podobné do skupín, tiež nazývaných klastre. Údaje, pri ktorých je zistené, že nepatria do žiadneho z týchto klastrov sú označené za šum.

2. **Vyhladzovanie** (Smoothing) je napríklad nahrádzanie údajov mediánom hodnôt v okolí tohto údaju. Týmto prístupom možno odstrániť odchýlky.

3. **Regresná analýza** je prístup založený na modelovaní vzťahov medzi atribútmi a následným odstraňovaním hodnôt, ktoré sa výrazne odchylujú.

Odľahlé hodnoty na rozdiel od šumu predstavujú výrazné odchýlky od bežných, očakávaných hodnôt.

Riešenia:

1. **Vizualizácia hodnôt** pomocou využitia grafov, diagramov a podobných vizuálnych nástrojov pre zobrazenie údajov a odhalenie extrémnych hodnôt.

2. **Stanovenie prahových hodnôt**, resp. odhalenie hodnôt, ktoré presahujú prahové hodnoty a môžu tak byť považované za extrémne odchýlky.

Integrácia dát

Pod pojmom integrácia dát sa myslí proces spájania dát, kedy sa zlučujú dáta z niekoľkých heterogénnych zdrojov do jednoho konzistentného formátu, ktorý je vhodný pre analýzu. Cieľom je vytvoriť ucelený pohľad na dáta tak, aby umožnil efektívne dolovanie znalostí a presnejšiu interpretáciu výsledkov [6].

V praxi je zvyčajné, že dáta pochádzajú z viacerých rôznych zdrojov, ako napríklad z rôznych súborov či databáz. Tieto zdroje môžu využívať inú terminológiu alebo iné formáty, ktoré je potreba zjednotiť.

Problémy, ktoré sa snažíme vyriešiť v procese integrácie dát:

Rozdielne formáty – Typickým príkladom sú rôzne formáty ukladania dátumov, napríklad formát YYYY-MM-DD oproti formátu DD/MM/YYYY.

Riešenie:

1. **Zjednotenie formátov:** Postup riešenia tohto problému je typicky priamočiarý, ide o automatizované, prípadne ručné konverzie formátov do zvoleného štandardu.

Konfliktné údaje – Rôzne zdroje dát môžu obsahovať odlišné hodnoty, ktoré sa napríklad vzájomne vylučujú. Napríklad rovnaký zákazník môže mať vo dvoch zdrojoch rôzne telefonné číslo.

Riešenie:

1. **Spájanie konfliktov**, kedy je určený jeden zo zdrojov, ktorý bude preferovaný a ktorého údaje budú zvolené prioritne nad údajmi ostatných zdrojov.

2. **Agregácia hodnôt všetkých zdrojov**

3. **Manuálna korekcia údajov**

Redundantné údaje – Rôzne zdroje dát môžu obsahovať rovnaké informácie. V tomto prípade je potreba zlúčenia dát takým spôsobom, aby sa predišlo viacnásobnému opakovaniu rovnakej informácie.

Riešenie:

1. **Zlúčenie viacnásobných záznamov do jedného**

Ako z popisu problémov a riešení tejto časti procesu čistenia dát vyplýva, typickými technikami, ktoré sú využívané pri integrácií dát sú využitie vhodných automatizovaných nástrojov (softvéru), kontrola údajov spolu s ručným vykonávaním požadovaných zmien.

Transformácia dát

Zatiaľ čo pri procesoch čistenia a integrácie dát je cieľom odstrániť z dát nedostatky a správne spojiť dáta pochádzajúce z viacerých rôznorodých zdrojov, v prípade transformácie dát je úlohou premena dát do takej formy, ktorá bude vhodná, prípadne až vyžadovaná, na to aby sa mohli podrobiť analýze.

Medzi najčastejšie techniky transformácií dát patria normalizácia, diskretizácia, kompresia a metódy vzorkovania:

Normalizácia dát Normalizácia je proces, ktorého úlohou je transformovať hodnoty atribútov tak, aby mali porovnateľný vplyv na výsledky analýzy.

Napríklad výška človeka (150–200 cm) a jeho váha (50–120 kg) sú v rôznych rozsahoch. Ak by sme tieto atribúty spracovávali bez úpravy, výška by mohla mať neprimeraný vplyv na výsledky analýzy.

Normalizácia (škálovanie) upravuje hodnoty do spoločného rozsahu, napríklad [0, 1].

Bežné techniky:

- **Min-max škálovanie**:
$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$
- **Z-score štandardizácia**:
$$x' = \frac{x - \mu}{\sigma},$$
kde μ je priemer a σ smerodajná odchýlka.

Diskretizácia dát Diskretizácia je transformácia, pri ktorej sa číselné hodnoty premieňajú na kategórie.

Príklad: hodnoty IQ možno rozdeliť na úrovne:

- 85–114: priemerná inteligencia,
- 115–129: nadpriemerná,
- 130–144: mierne nadaný,
- 145–159: vysoko nadaný.

Spôsoby diskretizácie:

- Pevný počet intervalov,
- Rovnaký počet hodnôt v každom intervale,
- Dozorované metódy založené na známych triedach.

Kompresia dát Kompresia znižuje veľkosť dát, ideálne bez straty informácie, čím sa znižuje výpočtová náročnosť pri analýze.

Príkladmi sú agregácia (namiesto práce s každým záznamom jednotlivo) alebo obrazová kompresia, pri ktorej mierne odchýlky nemenia význam pre analýzu.

Vzorkovanie Vzorkovanie predstavuje výber menšej podmnožiny z veľkej množiny dát tak, aby výsledná vzorka reprezentovala pôvodné dáta.

Techniky vzorkovania:

- Náhodné vzorkovanie,
- Stratifikačné vzorkovanie (na základe kategórií),
- Systematické vzorkovanie (napr. každý n-tý záznam).

2.2 Skladovanie dát

Cieľom tejto sekcie je stručne popísať základné, v praxi najčastejšie využívané princípy a formy skladovania dát, ktoré sú spojené s problematikou dolovania dát a dolovania sekvenčných vzorov. Sekcia 2.2.2 popisuje relačné databázy, ktoré sú základnou súčasťou systémov OLTP (Online Transaction Processing). V Sekcii 2.2.3 sú uvedené princípy fungovania dátových skladov, ktoré sú často používané, v súčasnosti až nevyhnutné pre podporu analytických operácií v rámci OLAP (Online Analytical Processing) a v Sekcii 2.2.4 je popísasná problematika sekvenčných databáz, ich vzniku a spôsobu využitia.

2.2.1 Úvod

Skladovanie dát je komplexná a stále sa rozširujúca oblasť, ktorej hlavnou motiváciou je vynachádzanie nových a zdokonalovanie existujúcich spôsobov, akými sú dáta ukladané, aby bolo možné efektívne ukladať neustále väčšie objemy dát.

Dáta môžu byť v praxi skladované rôznymi spôsobmi. Niekedy postačia jednoduché textové alebo tabuľkové súbory ako napríklad poznámkové bloky, formáty CSV a podobne. V iných prípadoch komplexnejšieho, či väčšieho objemu dát sú vhodné a často využívané pokročilejšie formy dátových skladísk, ktoré pomáhajú zefektívniť a zjednodušiť manipuláciu s komplexnejšími dátami.

2.2.2 Relačné databázy

Jedným z najrozšírenejších typov dátových skladísk sú relačné databázy. Slúžia ako základný stavebný blok systémov pre spracovanie online transakcií (OLTP systems – Online Transaction Porcessing systems). Systém relačných databáz zaviedol E. F. Codd v roku 1970. Databáza je definovaná ako množina tabuliek (resp. relácií), pričom platí, že každá tabuľka sa skladá zo stĺpcov, ktoré nazývame atribúty, a riadkov, ktoré nazývame záznamy. Každý riadok predstavuje iný záznam a každý záznam je identifikovaný jedinečným kľúčom. Vďaka tejto jedinečnej identifikácii je zabezpečená integrita údajov a umožnené efektívne vyhľadávanie [2].

Manipulácia s dátami je typicky v rámci relačných databáz vykonávaná pomocou Structured Query Language (SQL). SQL umožňuje vykonávanie základných operacií nad relačnou databázou – vytvárania, čítania, aktualizovania a mazania dát, ktoré sú tiež známe pod pojmom CRUD (Create, Read, Update, Delete). Vykonávanie týchto operacií nad relačnou databázou je zvyčajne rýchle aj efektívne, vďaka čomu sú práve relačné databázy ideálnym prostriedkom pre spracovávanie transakcií v reálnom čase. V praxi ich teda využívame napríklad v bankových systémoch pre ukladanie informácií o zákazníkoch, ich účtoch, transakciách, ktoré vykonali a podobne.

Medzi iné výhody relačných databáz patrí aj ich predvídateľná a jasne definovaná štruktúra, ktorá uľahčuje zachovávanie konzistentnosti a integrity dát, ďalej možnosť zabezpečenia transakcií pomocou mechanizmov akými sú napríklad ACID vlastnosti (Atomicita, Konzistentnosť, Izolácia, Trvanlivosť) a rôzne možnosti kontroly prístupu k dátam.

Výraznejším obmedzením relačných databáz je problém ukladania a spracovávania dát, ktoré sú neštrukturované. Takýmito dátami môžu byť rozsiahle textové súbory, obrázky či videá. Iným typickým problémom je obmedzená škálovateľnosť. Tento problém sa prejavuje v prípade, že je v databáze uložený priveľký objem dát a je potreba databázu rozdeliť do viacerých. Tento proces rozdeľovania databáz je známy pod pojmom database sharding.

Najznámejšie a najpoužívaniejšie technológie relačných databáz sú napríklad:

- PostgreSQL[1]
- MySQL[2]
- Oracle Database[3]
- SQLite[4]

2.2.3 Dátové sklady

Základnou myšlienkou za vznikom dátových skladov je potreba oddeliť dotazy určené na spracovanie transakcií (OLTP dotazy) od analytických dotazov, ktoré slúžia na podporu vykonávania manažérskych rozhodnutí a analýzu podniku. Dôvodom je, že vykonávanie komplexnejších analytických (OLAP) dotazov nad dátami uloženými v databáze by mohlo znížiť výkon a narušiť jej primárnu činnosť, ktorou je spracovanie každodenných transakcií v reálnom čase.

Dátový sklad možno definovať ako centrálny repozitár integrovaných dát, ktoré sú zozbierané s rôznych heterogénnych zdrojov a sú organizované a optimalizované pre podporu efektívnych analytických procesov a rozhodovania [8]. Dáta v dátových skladoch sú v praxi zvyčajne spracované, vyčistené a transformované, vďaka čomu sú kvalitným zdrojom pre spolahlivú analýzu.

Štruktúra dátového skladu je optimalizovaná pre dotazy typu OLAP, ktoré umožňujú analyzovať dáta z rôznych perspektív a dimenzí, ako napríklad podľa kategórie produktu, času, geografickej lokality a podobne. V rámci dátovych skladov je typické využitie dimenzionálneho modelovania, čo znamená, že štruktúra skladu je orientovaná napríklad do hviezdicovej schémy (star scheme), alebo schémy snehovej vločky (snowflake scheme), ktoré umožňujú rýchlejšie a efektívnejšie vykonávanie analytických dotazov.

Hviezdicová schéma pozostáva z jednej centrálnej tabuľky, nazývanej tabuľka faktov, ktorá obsahuje numerické ukazatele, nazývane fakty, a ďalej viacerých dimenzionálnych tabuliek, vďaka ktorým je umožnené vykonávanie detailnej analýzy podľa rôznych kriterií.

Schéma snehovej vločky je variantom hviezdicovej schémy, kde sú už spomínané dimenzionálne tabuľky rozdelené do ďalších viacerých vzájomne prepojených tabuliek. Táto vlastnosť pomáha znižovať redundanciu dát, môže však viesť k zníženiu rýchlosti dotazov.

Charakteristickými vlastnosťami uložených dát sú: integrita, historickosť, nemennosť a orientovanosť k predmetu. Orientovanosť dát k predmetu znamená, že dáta sú organizované

[1] https://www.postgresql.org/
[2] https://www.mysql.com/
[3] https://www.oracle.com/database/
[4] https://www.sqlite.org/

okolo oblastí záujmu daného podniku, môžu nimi byť produkty, zákazníci alebo iné. Nemennosť dát značí, že po tom, ako boli dáta uložené, sa už nemenia. Táto vlastnosť umožňuje získavať z analýzy spolahlivejšie výsledky a takisto presnosť pri opakovaní analytických dotazov. Historickosť dát znamená, že dáta sú skladované počas dlhšieho obdobia, vďaka čomu možno napríklad pozorovať ako sa menia trendy.

Dátové sklady sú tvorené za účeľom optimalizácie výnosov podniku, analyzovanie správania zákazníkov, vyhodnocovanie finančných výsledkov a podobne. Medzi oblasti, v ktorých sú využívané dátove sklády patria napríklad marketingové analýzy.

Najznámejšie a najpoužívaniejšie technológie dátových skladov sú napríklad:

- Snowflake[5]
- Amazon Redshift[6]
- Google BigQuery[7]
- Microsoft Azure Synapse Analytics[8]
- Teradata[9]

2.2.4 Sekvenčné databázy

Sekvenčná databáza nie je bežne využívaným úložiskom dát, avšak v kontexte dolovania sekvenčných vzorov má svoje jasné použitie. Dáta sa v sekvenčnej databáze uchovávaju vo forme postupnosti udalostí, či položiek, pričom dôležitým faktorom je poradie týchto udalostí. Sekvenčnými dátami v praxi môžu byť napríklad nákupné transakcie užívateľov, bioinformatické dáta – sekvencie aminokyselín tvoriacich bielkovinu, DNA a podobne [4].

Na rozdiel od relačnej databázy, je sekvenčná databáza špecificky navrhnutá pre podporu sekvenčných operácií, to znamená hľadanie často sa opakujúcich vzorov, vyhľadávanie subsekvencií, porovnávanie sekvencií a podobne. Tento databázový formát umožňuje efektívnejšiu analýzu dát pri procese dolovania sekvenčných vzorov, keď je vyžadované brať ohľad na poradie prvkov.

Záznam v sekvenčnej databáze má zvyčajne podobu identifikátora sekvencie (SID) a sekvencie, ktorá je tvorená udalosťami. Pre lepšie pochopenie na praktickom príklade formy sekvencií a udalostí, si možno predstaviť internetový obchod, ktorý zaznamenáva nákupy svojich zákazníkov za cieľom lepšie pochopiť ich správanie. Udalosť v tomto príklade môže predstavovať jeden produkt, ktorý si zákazník zakúpil, pričom sekvencia bude tvorená všetkými produktami, ktoré boli daným zákazníkom zakúpené a bude zachované poradie ich nákupu.

Algoritmy, ktoré sa používajú pre dolovanie sekvenčných vzorov typicky požadujú, aby ich vstupom bola práve sekvenčná databáza. Je potreba tiež spomenúť, že štruktúra sekvenčnej databázy môže byť závislá aj od implementácie algoritmov, ktoré plánujeme použiť pre vyhľadávanie vzorov v danej sekvenčnej databáze. Napríklad algoritmy implementované v knižnici SPMF, ktorá je jednou z najpoužívanejších knižníc v oblasti dolovania sekvenčných vzorov, vyžadujú špecifický formát databázy, kde každá sekvencia musí byť zapísaná

[5] https://www.snowflake.com/
[6] https://aws.amazon.com/redshift/
[7] https://cloud.google.com/bigquery
[8] https://azure.microsoft.com/en-us/products/synapse-analytics/
[9] https://www.teradata.com/

na samostatnom riadku a zakončená hodnotou '-2', ďalej musí byť každá udalosť transformovaná na hodnotu pozitívneho celého čísla a ďalšie opatrenia, ktorých je potreba byť si vedomý, pri transformácií požadovaných dát do formy sekvenčnej databázy.

Medzi technológie, ktoré sú často využívané pre uchovávanie, či analýzu sekvenčných dát možno zaradiť:

- SPMF[10]
- TimescaleDB[11]

2.3 Úlohy dolovania dát

Dolovanie dát je rozsiahla oblasť, v rámci ktorej sa riešia rôzne typy analytických úloh. Tieto úlohy sa líšia podľa toho, aký druh poznatkov chceme získať a aký je účel analýzy. Sekcia 2.3.1 popisuje základnú kategorizáciu dolovacích úloh, na základe toho, aký je cieľ ich analýzy. V sekciách 2.3.2 a 2.3.3 sú popísané základné typy dolovacích úloh patriace do jednotlivých kategórií.

2.3.1 Rozdelenie úloh podľa cieľov analýzy

Úlohy dolovania dát možno obecne rozdeliť do dvoch kategórií na základe ich cieľa. Týmito kategóriami sú **deskriptívne** a **prediktívne** úlohy.

Deskriptívne dolovacie úlohy sa pokúšajú odpovedať na otázku `"Čo sa v daných dátach deje?"`. To znamená objavovanie vzorov a iných vlastností v existujúcich dátach. Príkladom môže byť hľadanie spoločných vzorov v sekvenciách aminokyselín bielkovín, ktoré môžu ukázať príbuznosť medzi bielkovinami.

Cieľom **prediktívnych** úloh je vykonanie odhadov budúcnosti, na základe dostupných dát, resp. otázkou je `„Čo sa stane na základe existujúcich dát?"`. Snaha je teda predpovedať hodnoty budúcich dát. Typickými modernými príkladmi sú predikcie pohybu cien kryptomien, burzových akcií a podobne.

Rozdiel medzi deskriptívnymi a prediktívnymi úlohami nie je presne definovaný, preto v niektorých prípadoch môžu byť prediktívne modely využité aj na vykonanie a zistenie deskriptívnych výsledkov a to isté platí aj naopak.

2.3.2 Prediktívne dolovacie úlohy

Prediktívne dolovanie sa zameriava na vytváranie modelov, ktoré umožňujú odhadovať budúce alebo neznáme hodnoty na základe existujúcich údajov.

Základným predpokladom je existencia historických alebo aktuálnych dát, ktoré slúžia ako tréningová množina, vďaka ktorej sa model naučí vzory a vzťahy medzi premennými, ktoré následne využíva na predikciu nových prípadov. Medzi najzákladnejšie prediktívne úlohy patria:

Klasifikácia

Prediktívna úloha, ktorej cieľom je vytvorenie modelu, ktorý dokáže priradzovať nové objekty do jednej zo známych kategórií na základe atribútov tohto objektu. Typicky funguje

[10] http://www.philippe-fournier-viger.com/spmf/
[11] https://www.timescale.com/

na princípe porovnávania nového objektu s ostatnými objektami, ktorých kategórie sú už známe.

Model sa učí aké sú vzťahy medzi atribútmi objektov a ich kategóriami a následne na základe týchto vzťahov tvorí odhad, kam zaradiť nový objekt. Jednoducho pochopiteľným príkladom môže byť klasifikácia objektu na obrázku, kedy sa model snaží odhadnúť, či ide o človeka, domáce zviera, hmyz alebo inú kategóriu.

Využívané algoritmy:

- **Rozhodovací strom**, kde sa na základe atribútov objektu vytvára postupnosť rozhodnutí, ktoré určia ako objekt klasifikovať.
- **Algoritmus k-najbližších susedov**, ktorý klasifikuje objekt na základe podobnosti k známym objektom.
- **Naïve Bayes klasifikátor** odhaduje pravdepodobnosť, že objekt patrí do určitej kategórie, na základe podmienenej pravdepodobnosti jednotlivých atribútov.

Regresia

Zatiaľ čo klasifikácia sa snaží o odhad kategórie objektu, cieľom regresie je odhadnúť číselné hodnoty na základe vstupných dát.

Regresný model sa snaží nájsť vzťah alebo funkciu, ktorý najlepšie vyhovuje vstupným dátam. Týmto spôsobom možno odhadovať napríklad rast alebo pokles ceny nehnuteľností. Regresný model bude hľadať vzťahy medzi atribútmi ako lokalita nehnuteľnosti, rozloha, počet izieb a na základe toho určí cenu nehnuteľnosti.

Príklady algoritmov:

- **Lineárna regresie** - Najjednoduchší typ regresnej úlohy, kedy hľadaná funkcia určujúca vzťah hodnôt má formu priamky.
- **Regresný strom** - Regresný strom je typom rozhodovacieho stromu, ktorý je spomínaný v časti algoritmov využívaných pri klasifikácii 2.3.2. Na rozdiel od typického rozhodovacieho stromu, ktorého priechod je riadený odpoveďami na otázky typu pravda/nepravda, obsahuje regresný strom číselné hodnoty. Prechod regresným stromom sa riadi podľa pozorovania vlastností tejto číselnej hodnoty.

Iné typy prediktívnych úloh

Okrem klasifikácie a regresie existuje mnoho iných prediktívnych úloh, ktoré sú často využívané v reálnych aplikáciách:

- **Doporučovacie systémy** — predikujú, aký obsah, produkt alebo služba môže byť pre konkrétneho používateľa najzaujímavejší na základe jeho správania alebo preferencií.
- **Predikcia časových radov** — cieľom je predpovedať budúce hodnoty na základe historických časovo zoradených dát, napríklad predpoveď teploty alebo cien akcií.
- **Odhaľovanie podvodov** — modely predikujú pravdepodobnosť, že určitá udalosť alebo transakcia je podozrivá alebo neautentická.

2.3.3 Deskriptívne dolovacie úlohy

Deskriptívne úlohy sa zameriavajú na objavovanie štruktúr, vzťahov a vzorov v dátach bez potreby predpovedať konkrétny výstup. Medzi hlavné úlohy tejto kategórie patria:

Klastrovanie

Klastrovanie je úlohou radenou do skupiny deskriptívnych úloh, ktorej cieľom je zoskupiť objekty do skupín nazývaných klastre tým spôsobom, že objekty patriace do jedného z klastrov sú si vzájomne čo najviac podobné a zároveň čo najviac odlišné od objektov v iných klastroch.

Dolovanie vzorov

Dolovanie vzorov je úloha zameraná na objavovanie vzorov, ktoré sa v dátach často opakujú. Ide teda o hľadanie často sa opakujúcich kombinácií položiek, pričom častosť kombinácie je vyjadrená prahovou hodnotou nazývanou minimálna podpora. Kombinácie, ktoré presahujú túto hodnotu sú považované za časté.

Podrobnejší popis dolovania vzorov, zameraný primárne na dolovanie sekvenčných vzorov je obsiahnutý v kapitole 3.

subsubsectionIné typy deskriptívnych úloh

Okrem klastrovania a dolovania vzorov medzi deskriptívne úlohy patria aj:

- **Asociačné pravidlá** — objavujú pravidlá typu „ak sa vyskytne A, často sa vyskytne aj B“, typické napríklad v analýze nákupných košíkov.
- **Detekcia anomálií** — zameriava sa na odhaľovanie objektov alebo udalostí, ktoré sa výrazne odlišujú od bežného správania v dátach.
- **Zhrnutie dát** — úloha, ktorej cieľom je zjednodušiť veľké množstvo dát pomocou štatistických ukazovateľov alebo agregácií.
- **Vizualizačné techniky** – pomáhajú zobraziť štruktúru dát.

Kapitola 3

Dolovanie sekvenčných vzorov

Táto kapitola slúži na uvedenie čitateľa do problematiky procesu dolovania sekvenčných vzorov a popis často využívaných algoritmov. Sekcia 3.2 popisuje základné pojmy v tejto oblasti, ktoré je potreba poznať ako pre pochopenie problematiky, tak aj pre pochopenie princípov funkcionality algoritmov. Sekcia 3.3 uvádza do problematiky dolovania sekvenčných vzorov a v sekcii 3.4 sú popísané základné algoritmy dolovania sekvenčných vzorov.

3.1 Úvod

Tradičné dolovanie vzorov je oblasťou dolovania dát, ktorá sa zameriava na objavovanie zaujímavých, užitočných alebo neočakávaných vzorov v databázach. Vývoj tejto disciplíny sa začal v 90. rokoch 20. storočia s publikáciou algoritmu Apriori, ktorý bol navrhnutý na objavovanie častých množín položiek. Príkladom využitia tohto algoritmu môže byť identifikácia skupín položiek v maloobchodoch, ktoré sa často nakupujú spolu [4].

Jedným z obmedzení dolovania vzorov a dôvodom za vznikom dolovania **sekvenčných** vzorov je **ignorovanie poradia udalostí**. Napríklad spomínaný algoritmus Apriori je schopný identifikovať položky, ktoré sa často vyskytujú spoločne, neberie však ohľad na ich poradie. Dolovanie sekvenčných vzorov je odvetvím tradičného dolovania vzorov, ktorého cieľom je nájdenie subsekvencií v množine sekvencií. Dôležitým faktorom je, že sekvencie sú usporiadanými množinami, teda berú ohľad na poradie udalostí.

3.2 Základné pojmy

- **Položka (Item)**: Najzákladnejší a najmenší prvok sekvencie – napr. produkt, akcia, udalosť atď.
- **Množina položiek (Itemset)**: Neprázdna množina položiek, ktoré sa vyskytli spoločne v jednej udalosti. Notácia napr. $(A\ B)$ znamená, že položky A a B sa vyskytli v rovnakom čase.
- **Udalosť (Event)**: Súbor položiek, ktoré nastali naraz. V tejto notácii je každá udalosť zapísaná ako zátvorka, napr. (C) alebo $(B\ D)$. Pojmy udalosť a množina položiek sú v tomto kontexte často používané ako synonymá.

- **Sekvencia (Sequence)**: Usporiadaný zoznam udalostí. Sekvencia zachováva poradie výskytu udalostí v čase a je zapísaná v uhlových zátvorkách. Príklad:

$$\langle (A)(B\ C)(D) \rangle$$

znamená, že sa najskôr vyskytlo A, potom naraz B a C, a nakoniec D.

- **Sekvenčná databáza**: Kolekcia sekvencií, z ktorých každá má jedinečný identifikátor SID. Príklad: `(SID = 1, sekvencia =` $\langle (A)(C) \rangle$`)`
- **Podsekvencia a nadsekvencia**: Sekvencia α je podsekvenciou β, ak možno nájsť všetky jej udalosti v β v rovnakom poradí (nie nevyhnutne po sebe). Príklad: $\langle (A)(C) \rangle$ je podsekvenciou $\langle (A)(B)(C)(D) \rangle$
- **Podpora (Support)**: Počet sekvencií v databáze, ktoré obsahujú danú podsekvenciu. Napr. ak sa α nachádza v 3 z 5 sekvencií, jej podpora je 3.
- **Minimálna podpora (Minimum support)**: Typicky percentuálne vyjadrená prahová hodnota, ktorá definuje, čo považujeme za „častý vzor“. Sekvencia je považovaná za častú len vtedy, ak jej podpora je väčšia alebo rovná minimálnej podpore.
- **Častý sekvenčný vzor (Frequent Sequential Pattern)**: Sekvencia, ktorej podpora presahuje zvolený prah.
- **Kandidátna sekvencia (Candidate sequence)**: Sekvencia, ktorá je generovaná ako možný častý vzor, ale zatiaľ nebolo overené, či spĺňa požiadavku minimálnej podpory. V texte je tento pojem niekedy označovaný skrátene ako kandidát.
- **Uzavretá sekvencia (Closed sequence)**: Častá sekvencia, ktorá nemá žiadnu nadsekvenciu so zhodnou podporou.
- **Maximálna sekvencia (Maximal sequence)**: Častá sekvencia, ktorá nie je podsekvenciou žiadnej inej častej sekvencie.

3.3 Popis problematiky

Po zoznámení sa so základnými pojmami v oblasti sme schopní lepšie pochopiť dôvodu vzniku a zoznámiť sa s problematikou dolovania sekvenčných vzorov.

Cieľom dolovania sekvenčných vzorov je hľadanie a objavovanie zaujímavých vzorov v sekvenčných databázach. Tieto sekvencie môžu byť tvorené položkami alebo udalosťami, ktoré sú usporiadané v čase. Ako príklady možno uviesť transakcie zákazníkov, biologické dáta a podobné **časovo orientované údaje** [4].

Podstatou problému pri dolovaní sekvenčných vzorov je nájdenie všetkých subsekvencií, ktoré sa vyskytujú v databáze dostatočne často na to, aby bolo možné ich považovať za relevantné. Táto častosť výskytu subsekvencií je v praxi vyjadrovaná mierou, či hodnotou podpory, pričom platí, že častý sekvenčný vzor, resp. často sa vyskytujúca subsekvencia je tá, ktorej hodnota podpory presahuje stanovený minimálny prah podpory [9].

Spoločnými prvkami algoritmov sú vždy ich vstup a výstup, pričom vstupom je sekvenčná databáza a výstupom je množina častých sekvencií.

Kategorizovať algoritmy možno z viacerých úhlov pohľadu. Najčastejším je kategorizácia podľa prístupu využívanému ku generovaniu kandidátnych sekvencií. Z tohto hľadiska rozdeľujeme algoritmy do 2 kategórií:

- Algoritmy založené na **Apriori vlastnosti**, ktorá udáva, že ak sekvencia nie je častá, potom ani žiadna jej nadsekvencia nebude častá. Vďaka tejto vlastnosti sú algoritmy schopné efektívne prechádzať vyhľadávací priestor, keďže odoberá potrebu skúmať nadsekvencie, ktoré by boli odvodené z tých sekvencií, ktoré už nesplnili minimálny prah podpory. Algoritmy, ktoré sú založené na Apriori prístupe generujú kandidátne sekvencie iteratívne. Základným problémom týchto algoritmov je veľké množstvo vygenerovaných kandidátnych vzorov a potreba viacnásobne prechádzať databázu [4]. Zároveň sú však tieto algoritmy typicky jednoduché na implementáciu a pochopenie.
- Algoritmy využívajúce prístup **rastu vzoru** (pattern-growth) sú založené na princípe generovania kandidátnych sekvencii pomocou rozširovania už existujúcich častých sekvencií, čím sa zmenšuje počet generovaných kandidátnych sekvencií a zefektívňuje proces dolovania.

Aj napriek stálym pokrokom v tejto oblasti dolovania dát, existujú neustále sa opakujúce problémy, medzi ktoré patria hlavne škálovatelnosť algoritmov a efektívne spracovanie rozsiahlych databáz.

3.4 Algoritmy

Táto sekcia popisuje vybrané algoritmy pre dolovanie sekvenčných vzorov, ktoré patria medzi najzákladnejšie a najčastejšie používané. Pri jednotlivých algoritmoch sú popísané ich charakteristické znaky, postup a ukážka na pochopiteľnom príklade.

3.4.1 GSP

Algoritmus GSP (Generalized Sequential Patterns) bol predstavený Srikantom a Agrawalom ako rozšírenie predchádzajúcich prístupov k dolovaniu sekvenčných vzorov [13]. V dnešnej dobe je tento algoritmus považovaný za jeden zo základných algoritmov v oblasti dolovania sekvenčných vzorov.

Základný princíp:

Algoritmus GSP pracuje iteratívne. V každej iterácii sa najskôr hľadajú časté sekvencie s $k-1$ položkami (označované ako L_{k-1}), ktoré sa následne používajú na generovanie kandidátnych sekvencií C_k. Tie sa potom overujú v databáze a na základe hodnoty ich podpory sa určí, ktoré z nich sú skutočne časté.

Kroky algoritmu:

1. **Prvý prechod sekvenčnou databázou:** Získajú sa všetky časté 1-sekvencie (sekvencie pozostávajúce z jednej položky alebo jednej udalosti).
2. **Generovanie kandidátov:** Pre každý pár sekvencií z L_{k-1}, ktoré sa dajú spojiť (napr. majú rovnaký prefix), sa vytvárajú nové kandidátne sekvencie. Tento krok je založený na **Apriori vlastnosti**.
3. **Prerezávanie (Pruning):** Kandidátne sekvencie, ktorých niektorá z podsekvencií nie je častá, sú zo zoznamu vyradené.

4. **Výpočet podpory:** Prechode databázou za cieľom vypočítania hodnôt podpory kandidátnych sekvencií. Na tento účel sa používa **hash-tree** pre efektívne vyhľadávanie kandidátov a technika **alternatívnej reprezentácie sekvencií** pomocou zoznamov časových výskytov položiek.

5. **Iterácia:** Tento proces sa opakuje pre sekvencie dĺžky $k+1$, až kým nebudú generované žiadne ďalšie časté sekvencie.

Príklad:

Majme sekvenčnú databázu so štyrmi sekvenciami definovanú nasledovne:

ID	Sekvencia
S1	$\langle(A)(B\ C)(D)\rangle$
S2	$\langle(A\ B)(C)\rangle$
S3	$\langle(A)(B)(C\ D)\rangle$
S4	$\langle(A\ C)(E)\rangle$

Tabuľka 3.1: Sekvenčná databáza príkladu

Predpokladajme hodnotu minimálnej podpory rovnú 50 %, teda častá sekvencia sa bude musieť vyskytovať aspoň v dvoch rôznych sekvenciách.

GSP najskôr prejde databázu a nájde všetky časté 1-sekvencie, v tomto prípade $\langle(A)\rangle$, $\langle(B)\rangle$, $\langle(C)\rangle$ a $\langle(D)\rangle$. Na základe nájdených častých 1-sekvencií sa vygenerujú kandidátne 2-sekvencie, napríklad $\langle(A)(B)\rangle$, $\langle(A)(C)\rangle$, $\langle(B)(C)\rangle$, $\langle(C)(D)\rangle$ alebo $\langle(B\ C)\rangle$. Každá z týchto kandidátnych sekvencií je následne overená v celej databáze, pričom sa zaznamenáva, v koľkých sekvenciách sa vyskytuje.

Napríklad sekvencia $\langle(A)(B)\rangle$ sa nachádza v troch sekvenciách (S1, S2 a S3), takže je častá. Naopak, sekvencia $\langle(B\ C)\rangle$ sa vyskytuje iba v sekvencii S1, preto nie je zahrnutá do výsledku.

GSP bude týmto spôsobom iteratívne rozširovať dĺžku sekvencií a opakovať tento proces, až dokým nebude možné vygenerovať nové časté sekvencie.

3.4.2 PrefixSpan

PrefixSpan (Prefix-projected Sequential pattern mining) je algoritmus na dolovanie sekvenčných vzorov, ktorý využíva prístup rastu vzorov s využitím prefixovo projektovaných databáz. Bol navrhnutý ako reakcia na neefektivitu algoritmov založených na prístupe Apriori vlastnosti, ako napríklad GSP, ktoré generujú veľké množstvo kandidátnych sekvencií a viacnásobne prechádzajú sekvenčnú databázou [10].

Základnou myšlienkou algoritmu je, že sa databáza sekvencií rekurzívne projektuje na menšie databázy (tzv. *prefix-projected databases*), z ktorých sa následne generujú sekvenčné vzory rozširovaním častých prefixov.

Postup algoritmu:

1. **Zistenie častých 1-sekvencií:** Prechod sekvenčnou databázou a identifikácia častých jednoprvkových sekvencií, ktoré spĺňajú minimálny prah podpory. Tieto položky budú predstavovať časté prefixy v ďalšom kroku.

2. **Vytváranie projektovaných databáz:** Pre každý častý prefix sa vytvorí projektovaná databáza obsahujúca iba sufixy sekvencií, ktoré sa nachádzajú za prvým výskytom tohto prefixu.

3. **Rekurzívne dolovanie:** V každej projektovanej databáze sa opäť dolujú časté položky, ktoré sa môžu pripojiť k prefixu, čím sa vytvárajú dlhšie časté sekvencie. Tento proces sa opakuje, až kým už nie je možné nájsť nové časté vzory.

Príklad:

Uvažujme nasledujúcu sekvenčnú 3.2 databázu so štyrmi sekvenciami a predpokladajme minimálnu podporu 50 %:

Tabuľka 3.2: Sekvenčná databáza príkladu

ID	Sekvencia
S1	$\langle(A)(B\ C)(D)\rangle$
S2	$\langle(A\ B)(C)\rangle$
S3	$\langle(A)(B)(C\ D)\rangle$
S4	$\langle(B\ C)(E)\rangle$

PrefixSpan vykoná počiatočný prechod databázou a identifikuje položky A, B, C a D ako časté 1-sekvencie, z ktorých vytvorí počiatočné prefixy $\langle(A)\rangle$, $\langle(B)\rangle$, $\langle(C)\rangle$, $\langle(D)\rangle$. Pre tieto prefixy sa následne vytvára projektovaná databáza, ktorá obsahuje sufixy – teda časti sekvencií, ktoré nasledujú bezprostredne po prvom výskyte daného prefixu:

Tabuľka 3.3: Projektovaná databáza pre prefix $\langle(A)\rangle$

ID	Projektovaný sufix
S1	$\langle(B\ C)(D)\rangle$
S2	$\langle(B)(C)\rangle$
S3	$\langle(B)(C\ D)\rangle$

Z tejto projektovanej databázy algoritmus vyhľadá časté položky, ktoré je možné pripojiť k prefixu $\langle(A)\rangle$, čím vzniknú nové rozšírenia ako $\langle(A\ B)\rangle$, $\langle(A)(B)\rangle$, $\langle(A)(C)\rangle$ a podobne. Pre každé takéto rozšírenie sa následne vytvorí nová projektovaná databáza a celý proces sa rekurzívne opakuje.

Takýmto spôsobom PrefixSpan prehľadáva iba časté vetvy vyhľadávacieho priestoru bez potreby generovania všetkých možných kandidátnych kombinácií.

Výhody algoritmu:

- Nevyžaduje generovanie kandidátov ako algoritmy využívajúce Apriori-prístup.
- Efektívne redukuje veľkosť databázy pomocou projekcie.

3.4.3 SPADE

SPADE (Sequential PAttern Discovery using Equivalence classes) je algoritmus, ktorý využíva vertikálny formát databázy a rozklad vyhľadávacieho priestoru pomocou tried ekvivalencie. Hlavnými výhodami sú nízky počet prechodov databázou a využitie jednoduchých dátových štruktúr [15].

Základné vlastnosti

SPADE využíva **vertikálny formát databázy**, kde ku každej položke (alebo vzoru) uchováva tzv. `ID-zoznam (ID-list)`, čo je zoznam dvojíc (SID, TID), ktoré označujú identifikátor sekvencie a pozíciu udalosti, v ktorej sa vzor vyskytuje. Následne algoritmus počíta podporu kandidátnych vzorov pomocou jednoduchých `dočasových spojení (temporal joins)` medzi týmito ID zoznamami.

Kroky algoritmu

1. **Konverzia do vertikálneho formátu:** Databáza sa prevedie z horizontálneho do vertikálneho formátu vytvorením ID-zoznamov pre všetky jednotlivé položky, ktoré predstavujú 1-sekvencie.

2. **Zistenie častých 1- a 2-sekvencií:** Na základe ID zoznamov sa identifikujú všetky časté sekvencie dĺžky 1. Následne sa spojením dvojíc týchto zoznamov hľadajú časté 2-sekvencie.

3. **Rozdelenie vyhľadávacieho priestoru:** Použitím **prefixovo založených ekvivalenčných tried** sa vyhľadávací priestor rozdelí na menšie podpriestory. Dve sekvencie patria do rovnakej triedy, ak zdieľajú spoločný prefix dĺžky $k-1$.

4. **Rekurzívne rozširovanie vzorov:** V rámci každej triedy sa generujú nové kandidátne sekvencie kombináciou existujúcich vzorov a ich ID zoznamov. Podpora novovzniknutých sekvencií sa určuje spojením ID zoznamov. Tento proces sa rekurzívne opakuje, teda sekvencie sa ďalej rozširujú, pokiaľ majú dostatočnú podporu.

5. **Ukončenie algoritmu:** Dolovanie sa ukončí v momente, keď nie je možné vygenerovať žiadne nové časté sekvencie. To znamená, že v danej ekvivalenčnej triede už neexistujú kombinácie, ktorých podpora by spĺňala požadovaný minimálny prah.

Príklad:

Uvažujme sekvenčnú databázu pozostávajúcu zo štyroch sekvencií, s minimálnou podporou 50 %.

Tabuľka 3.4: Sekvenčná databáza k príkladu

ID	Sekvencia
S1	⟨(A)(B)⟩
S2	⟨(A)(C)⟩
S3	⟨(A)(B C)⟩
S4	⟨(B)(C)⟩

Najskôr algoritmus vytvorí ID zoznamy pre jednotlivé položky:

- A: { (S1, 1), (S2, 1), (S3, 1) }
- B: { (S1, 2), (S3, 2), (S4, 1) }
- C: { (S2, 2), (S3, 2), (S4, 2) }

Všetky unikátne položky sa nachádzajú aspoň v dvoch sekvenciách, preto sú časté 1-sekvencie zaradené do množiny L_1. Následne sa spájaním ID zoznamov identifikujú časté 2-sekvencie. Napríklad pre kandidáta $\langle(A)(B)\rangle$ sa zisťuje, v koľkých sekvenciách sa B nachádza po výskyte A. To platí pre S1 a S3, takže podpora je 2 a vzor je pridaný do množiny L_2.

Vyhľadávací priestor sa následne rozdelí do tried podľa spoločného prefixu, napríklad všetky vzory začínajúce prefixom $\langle(A)\rangle$ patria do jednej triedy. V rámci tejto triedy sa ďalej generujú a testujú dlhšie sekvencie, až kým nie je možné nájsť nové časté vzory.

SPADE týmto spôsobom efektívne prehľadáva priestor bez potreby opakovaného prechádzania celej sekvenčnej databázy.

3.4.4 CM-SPADE

CM-SPADE (Co-occurrence Map-enhanced SPADE) je rozšírením algoritmu SPADE 3.4.3, ktorého kľúčovými vlastnosťami sú využívanie vertikálneho formátu sekvenčnej databázy a spájanie tzv. *ID zoznamov* (IdLists). Novou vlastnosťou v algoritme CM-SPADE je zavedenie dátovej štruktúry **CMAP** (Co-occurrence Map), ktorá umožňuje včasné prerezávanie (pruning) nekvalitných kandidátnych sekvencií ešte pred výpočtom ich podpory [3].

CMAP štruktúra: Algoritmus počas prvého prechodu databázou vytvára dve dátové štruktúry:

- $\text{CMAP}_i(a)$ uchováva množinu položiek, ktoré sa často vyskytujú v rovnakej udalosti ako položka a.
- $\text{CMAP}_s(a)$ uchováva množinu položiek, ktoré sa často vyskytujú po položke a v rovnakej sekvencii.

Tieto mapy obsahujú len také položky, ktoré sa s a vyskytujú spoločne vo viac sekvenciách, ako je hodnota minimálnej podpory. CMAP môže byť efektívne implementovaná ako hashovacia tabuľka množín a v niektorých prípadoch môže výrazne znižovať pamäťovú náročnosť.

Pravidlá pre prerezávanie kandidátnych sekvencií:

- **Pravidlo i-rozšírenia:** Ak $b \notin \text{CMAP}_i(a)$, potom i-rozšírenie položky a o b nie je časté.
- **Pravidlo s-rozšírenia:** Ak $b \notin \text{CMAP}_s(a)$, potom s-rozšírenie položky a o b nie je časté.
- **Generalizácia:** Ak akýkoľvek prvok sekvencie nemá b v príslušnej CMAP, potom celá nadsekvencia obsahujúca b nemôže byť častá.

CM-SPADE funguje ako pridaná vrstva nad klasickým SPADE. Počas generovania nového kandidáta $r = A \cup \{b\}$ sa algoritmus najskôr pozrie, či b patrí do príslušného $\text{CMAP}(a)$ pre niektorú položku a v A. Ak nie, kandidátna sekvencia sa vôbec negeneruje.

Príklad:

Uvažujme jednoduchú sekvenčnú databázu, zobrazenú v tabuľke 3.5, so štyrmi sekvenciami a minimálnou podporou 50 %:

Pri prvom prechode databázou algoritmus nájde 1-sekvencie a zistí ich podporu. V tomto prípade budú častými 1-sekvenciami položky A, B a C. Následne sa pre tieto položky vytvoria štruktúry CMAP:

Tabuľka 3.5: Sekvenčná databáza pre príklad

ID	Sekvencia
S1	⟨(A B)(C)⟩
S2	⟨(A)(C)⟩
S3	⟨(B)(C)⟩
S4	⟨(A B)(D)⟩

- $\text{CMAP}_i(A) = \{B\}$, pretože A sa často vyskytuje v tej istej udalosti ako B (v S1 a S4),
- $\text{CMAP}_s(A) = \{C\}$, pretože C sa často objavuje po A (v S1 a S2),
- $\text{CMAP}_s(B) = \{C\}$, pretože C sa často objavuje po B (v S1 a S3).

Na základe týchto informácií vieme, že napríklad sekvencia ⟨(A)(C)⟩ je častá (nachádza sa v S1 a S2), a teda je vhodná na ďalšie rozširovanie. Najskôr sa algoritmus pokúsi o i-rozšírenie vzoru ⟨(A)(C)⟩ o položku B. Na to skontroluje, či $B \in \text{CMAP}_i(C)$. Ak táto podmienka nie je splnená, sekvencia ⟨(A)(C B)⟩ sa vôbec negeneruje. Rovnako sa následne pokúsi o s-rozšírenie, teda vytvoriť sekvenciu ⟨(A)(C)(B)⟩ – no ak $B \notin \text{CMAP}_s(C)$, tak bude aj toto rozšírenie vylúčené.

Algoritmus sa ukončí, keď pre žiadnu položku x neplatí $x \in \text{CMAP}_i(a)$ ani $x \in \text{CMAP}_s(a)$ pre žiadnu častú položku a.

3.4.5 SPAM

Algoritmus **SPAM** (Sequential PAttern Mining) bol predstavený v roku 2002. Spája hĺbkové prehľadávanie vyhľadávacieho priestoru (DFS) s vytváraním bitových vektorov na reprezentáciu položiek a výpočet podpory [1].

Základné vlastnosti:

- **Bitmapová reprezentácia:** Dáta sú uložené vo forme vertikálnych bitových vektorov, pričom každá položka má vlastný bitový vektor, ktorého bity reprezentujú prítomnosť položky v jednotlivých sekvenciách. Týmto spôsobom je výpočet podpory sekvencií vykonávaný hardvérovo veľmi efektívnymi bitovými operáciami.
- **Hĺbkové prehľadávanie (DFS):** SPAM prehľadáva priestor možných sekvencií rekurzívne, od najkratších po dlhšie sekvencie. Na rozdiel od algoritmov založených na prístupe Apriori, pre ktoré je typické prehľadávanie vyhľadávacieho priestoru do šírky (BFS), SPAM postupuje do hĺbky vyhľadávacieho stromu.
- **Dva typy rozšírení:**
  - **S-krok (rozšírenie sekvencie):** Pridanie novej položky ako samostatnej udalosti.
  - **I-krok (rozšírenie množiny položiek):** Pridanie položky do poslednej udalosti sekvencie.
- **Prerezávanie (pruning):** Pre možnosť dynamického skracovania kandidátnych množín využíva SPAM vlastnosti Apriori (S-krok aj I-krok).

Bitmapová reprezentácia a operácie:

Každá položka je reprezentovaná bitovým vektorom m, kde každý bit označuje prítomnosť v konkrétnej sekvencii. Bitové AND operácie medzi bitovými vektormi umožňujú rýchle overenie, či položky spolu existujú v rovnakej sekvencii (I-krok), alebo vo vyhovujúcom poradí (S-krok).

- Pri **I-krok** je bitový vektor výslednej sekvencie ziskaný bitovou operáciou AND medzi bitovým vektorom pôvodnej sekvencie a bitovým vektorom novej položky.
- Pri **S-krok** je vytvorený tzv. *transformovaný bitový vektor*, kde všetky bity pred prvým výskytom poslednej udalosti sú nastavené na 0 a ostatné na 1. Tento bitový vektor sa potom AND-uje s bitovým vektorom novej položky.

Príklad:

Uvažujme minimálnu podporu 50 % a jednoduchú sekvenčnú databázu 3.6:

ID	Sekvencia
S1	⟨(A)(B)⟩
S2	⟨(A C)(B)⟩
S3	⟨(A)(C)⟩
S4	⟨(B)(C)⟩

Tabuľka 3.6: Sekvenčná databáza pre príklad

Položky A, B a C majú tieto bitové vektory:

- A: (1 1 1 0)
- B: (1 1 0 1)
- C: (0 1 1 1)

Každá z položiek má podporu aspoň 2, takže tvoria počiatočné časté vzory.

Algoritmus začne so vzorom ⟨(A)⟩ (podpora 3) a pokúsi sa ho rozšíriť pomocou I-kroku, teda pridaním položky do rovnakej udalosti. Ako prvá sa bude pridávať položka B:

$$(1\ 1\ 1\ 0)\ \text{AND}\ (1\ 1\ 0\ 1) = (1\ 1\ 0\ 0)$$

Z výsledného vektoru (1 1 0 0) možno určiť, že ⟨(A B)⟩ má podporu 2 a je teda častým vzorom.

Rovnako môže vzniknúť vzor ⟨(A C)⟩:

$$(1\ 1\ 1\ 0)\ \text{AND}\ (0\ 1\ 1\ 1) = (0\ 1\ 1\ 0)$$

⟨(A C)⟩ by mal rovnako podporu 2 a radil by sa za medzi časté vzory.

Po ukončení I-krokov sa SPAM pokúsi o S-krok, napr. pre ⟨(A)⟩. Posunutý vektor (A) je (0 1 1 0), ktorý sa AND-uje s vektorom položky C:

$$(0\ 1\ 1\ 0)\ \text{AND}\ (0\ 1\ 1\ 1) = (0\ 1\ 1\ 0)$$

Výsledok má podporu 2, teda ⟨(A)(C)⟩ je častý.

Ak výsledný AND vektor nemá dostatočnú podporu, vetva sa prerezáva.

Algoritmus pokračuje rekurzívnym rozširovaním vzorov, pričom sa v každej vetve pokúša o I-krok a S-krok s ďalšími položkami. Pre každý takto vytvorený nový vzor opäť vypočíta podporu pomocou bitových operácií a ak je podpora dostatočná, vzor sa ďalej rozširuje.

Proces sa opakuje dovtedy, kým nie je možné vykonať žiadne ďalšie rozšírenie (žiadne nové položky nemajú dostatočnú podporu po AND operácii), alebo dokým neboli prerezané všetky vetvy.

3.4.6 CM-SPAM

CM-SPAM (Co-occurrence Map SPAM) je optimalizovanou verziou algoritmu SPAM 3.4.5. Princíp optimalizácie je rovnaký ako v prípade už popísaného algoritmu CM-SPADE, resp. na základe využitia štruktúr CMAP 3.4.4 [3].

Princíp fungovania

Rovnako ako klasický SPAM, aj CM-SPAM využíva reprezentáciu položiek pomocou **bitových vektorov**, **hĺbkové prehľadávanie (DFS)** a dva typy rozšírení:

- **I-krok** – rozšírenie poslednej udalosti o ďalšiu položku.
- **S-krok** – pridanie novej udalosti s jednou položkou.

Rozdiel medzi SPAM a CM-SPAM je v tom, že CM-SPAM vďaka CMAP štruktúram umožňuje kontrolovať, či položka b patrí do $\text{CMAP}_i(a)$ alebo $\text{CMAP}_s(a)$, vďaka čomu nepotrebuje počítať hodnoty podpory generovaných sekvencií.

Príklad:

Uvažujme túto sekvenčnú databázu s minimálnou podporou 50 %:

ID	Sekvencia
S1	⟨(A B) (C)⟩
S2	⟨(A) (C)⟩
S3	⟨(B) (C)⟩
S4	⟨(A B) (D)⟩

Tabuľka 3.7: Sekvenčná databáza pre príklad

Na začiatku algoritmus vytvorí bitové vektory pre všetky položky, teda A bude reprezentované ako (1 1 0 1), B ako (1 0 1 1), C ako (1 1 1 0) a D ako (0 0 0 1).

Následne dôjde ku zostaveniu štruktúr CMAP_i a CMAP_s, konkrétne: $\text{CMAP}_i(A) = \{B\}$, keďže A a B sa spolu vyskytujú v jednej udalosti (S1, S4), $\text{CMAP}_s(A) = \{C, D\}$, pretože C alebo D sa objavujú po A (S1, S2, S4), $\text{CMAP}_i(B) = \{A\}$, pretože A sa spolu s B vyskytuje v jednej udalosti v S1 a S4, $\text{CMAP}_s(B) = \{C\}$, keďže C sa vyskytuje po B v S1 aj S3, a $\text{CMAP}_s(C) = \emptyset$, pretože po položke C už v žiadnej sekvencii nesleduje žiadna iná položka.

Algoritmus začne vzorom ⟨(A)⟩, ktorého podpora je 3. Pri I-rozšírení algoritmus skontroluje $\text{CMAP}_i(A)$ a zistí, že jediným platným kandidátom je B. Vznikne teda vzor ⟨(A B)⟩, ktorého podpora sa vypočíta bitovou AND operáciou medzi A = (1 1 0 1) a B = (1

0 1 1), z ktorej vznikne vektor (1 0 0 1). To znamená, že vzor ⟨(A B)⟩ sa nachádza v S1 a S4, jeho podpora je teda 2 a vzor je častý.

Vzor ⟨(A B)⟩ sa stane novým prefixom a algoritmus sa ho pokúsi ďalej rozširovať pomocou I-krokov a S-krokov, pričom bude opäť najskôr overovať, či nové položky patria do príslušných CMAP štruktúr.

Po spracovaní všetkých rozšírení vzoru ⟨(A B)⟩ sa algoritmus pokúsi o S-rozšírenie vzoru ⟨(A)⟩, pričom sa na základe $\text{CMAP}_s(A)$ vytvoria len vzory ⟨(A)(C)⟩ a ⟨(A)(D)⟩, pretože iné položky nie sú v $\text{CMAP}_s(A)$.

Takto algoritmus pokračuje rekurzívne pre všetky časté vzory a v každom kroku ešte pred výpočtom podpory vykoná kontrolu pomocou CMAP. Vetva sa ukončí, ak nie je žiadne j také, že $j \in \text{CMAP}_i(a)$ alebo $j \in \text{CMAP}_s(a)$ pre posledný prvok a vo vzore.

3.4.7 FAST

Algoritmus **FAST** (Fast Sequence Mining Algorithm Based on Sparse ID-Lists) bol navrhnutý ako efektívna metóda pre dolovanie sekvenčných vzorov z veľkých sekvenčných databáz. Princípom jeho efektivity je odstránenie opakovaných prechodov sekvenčnej databázy a efektívnejšie generovanie kandidátnych sekvencii [12].

Hlavnými technikami, ktoré robia FAST efektívnym sú:

1. **Riedke ID-zoznamy (Sparse ID-Lists, SIL):** dátová štruktúra pre ukladanie výskytov položiek v jednotlivých sekvenciách počas fázy rozširovania množín položiek.
2. **Vertikálne ID-zoznamy (Vertical ID-Lists, VIL):** využívajú sa vo fáze rozširovania sekvencií na efektívne zisťovanie, či položka b následuje po položke a v tej istej sekvencii.
3. **Lexikografické stromy:** stromová reprezentácia vyhľadávacieho priestoru sekvencií, v ktorom každý uzol zodpovedá jednej častej sekvencii.

Princíp fungovania

FAST využíva dvojfázový prístup, kedy najskôr nájde všetky časté položky a množiny položiek pomocou tzv. *rozšírenia množiny položiek (itemset-extension)* a následne hľadá časté sekvencie s využitím *rozšírenia sekvencie (sequence-extension)*. Tieto dva tipy rozšírenia v princípe zodpovedajú I-kroku a S-kroku, ktoré boli spomenuté pri algoritme SPAM 3.4.5

1. Fáza rozširovania množín položiek (Itemset-Extension phase) V prvej fáze dochádza k jednomu prechodu sekvenčnou databázou, na základe ktorého sa vytvoria **riedke ID-zoznamy (SIL)** pre jednotlivé položky. SIL_i je vektor, v ktorom každý prvok predstavuje zoznam pozícií (transaction-ID), kde sa položka i vyskytuje v konkrétnej sekvencii.

- Ak sa položka i nenachádza v sekvencii s_j, potom $\text{SIL}_i[j] = \texttt{null}$.
- Dĺžka SIL_i je fixná a rovná počtu sekvencií v databáze.

Časté množiny položiek sú získavané kombinovaním SIL viacerých položiek — sú ponechané iba tie pozície, kde sa všetky položky v rámci množiny položiek vyskytujú spoločne.

2. Fáza rozširovania sekvencií (Sequence-Extension phase) V druhej fáze sa vytvorí **lexikografický strom sekvencií**, v ktorom každý uzol reprezentuje jednu častú sekvenciu. Kandidátne sekvencie sa generujú pomocou rozšírenia prefixov sekvencií. Používajú sa dva druhy rozšírenia:

- **Rozšírenie množiny položiek**: pridanie položky do poslednej množiny položiek.
- **Rozšírenie sekvencie**: pridanie novej položky, ako novej množiny položiek.

Na podporu výpočtu frekvencie sekvencií využíva FAST štruktúru **Vertikálneho ID-zoznamu (Vertical ID-Lists)**, ďalej označovanú ako VIL, ktorá umožňuje efektívne zistiť, či sa položka b vyskytuje po položke a v tej istej sekvencii.

Príklad:

Uvažujme sekvenčnú databázu so štyrmi sekvenciami a minimálnou podporou 50 %:

ID	Sekvencia
S1	⟨(A B) (C)⟩
S2	⟨(A) (B)⟩
S3	⟨(A B C)⟩
S4	⟨(B) (C)⟩

Tabuľka 3.8: Sekvenčná databáza pre príklad

V prvom kroku sa pre každú položku vytvorí SIL, ktorého hodnoty budú ukazovať, v ktorých sekvenciách a na akých pozíciách sa položka nachádza. Konkrétne, SIL(A) bude mať hodnoty [1, 1, 1, null], SIL(B) bude [1, 2, 1, 1], SIL(C) [2, null, 1, 2]. Z týchto zoznamov algoritmus zistí, že položky A, B aj C sa nachádzajú aspoň v dvoch sekvenciách a sú teda častými 1-sekvenciami.

Následne sa algoritmus pokúsi rozšíriť časté 1-sekvencie o ďalšie položky v rámci tej istej množiny položiek pomocou I-rozšírení. Bude kontrolovať, či sa napríklad A a B vyskytujú v rovnakých sekvenciách a na rovnakých pozíciách. V prípade kombinácie SIL(A) a SIL(B) budú zistené spoločné výskyty v S1 a S3, teda podpora vzoru ⟨(A B)⟩ bude 2 a pôjde o častý vzor. Algoritmus vytvorí výsledný SIL(A B), ktorý bude mať hodnoty [1, null, 1, null].

Kombinácie ako ⟨(A C)⟩ či ⟨(B C)⟩ pri I-rozšírení nebudú spĺňať prah minimálnej podpory, takže v týchto vetvách algoritmus ukončí rozširovanie a prejde do fázy S-rozšírení.

V tejto fáze bude snaha pridať ku každej častej sekvencii ďalšiu položku ako novú udalosť. Za týmto účelom si algoritmus vytvorí pre danú sekvenciu VIL, ktorý bude obsahovať pozície, v ktorých položka b nasleduje po položke a v tej istej sekvencii.

Napríklad pre S-rozšírenie vzoru ⟨(A)(B)⟩ vznikne zoznam VIL(⟨(A)(B)⟩) s hodnotami [null, 2, null, null]: v S1 sú A aj B na prvej pozícii, teda v tej istej udalosti a rozšírenie nie je platné; v S2 je A na pozícii 1, B na pozícii 2, čo je platné pre S-rozšírenie; v S3 sú A a B v tej istej udalosti; v S4 sa A vôbec nenachádza.

Výsledná podpora vzoru ⟨(A)(B)⟩ bude teda 1, čo je pod prahom, a algoritmus v danej vetve nepokračuje.

Ďalej sa vykoná S-rozšírenie pre vzor ⟨(A)(C)⟩ a tento proces sa opakuje pre všetky možné rozšírenia všetkých častých sekvencií. Algoritmus sa ukončí, keď už nebude možné vytvoriť žiadne nové časté sekvencie, teda keď všetky vetvy v lexikografickom strome skončia buď kvôli nedostatočnej podpore, alebo preto, že nie sú dostupné ďalšie S-rozšírenia.

3.4.8 LAPIN

LAPIN (LAst Position INduction) je algoritmus pre dolovanie sekvenčných vzorov, ktorý sa zameriava na zefektívnenie dolovania v hustých databázach. Hlavnou charakteristikou tohto algoritmu oproti iným prístupom je využitie tzv. **poslednej pozície výskytu položky**, na základe ktorej sa rozhoduje, či sa daná položka môže použiť na rozšírenie existujúceho častého vzoru [14].

Základné vlastnosti

- **Indukcia pozície:** sledovanie posledného výskytu každej položky v rámci sekvencie.
- Rozšírenia sa vykonávajú iba vtedy, ak sa nová položka nachádza za poslednou pozíciou aktuálneho prefixu.

Kroky algoritmu

1. **Počiatočný prechod databázou:** Vytvoria sa zoznamy pozícií výskytu položiek a tabuľky posledných výskytov pre každú položku v každej sekvencii.
2. **Identifikácia častých 1-sekvencií:** Určia sa všetky položky, ktoré spĺňajú podmienku minimálnej podpory.
3. **Rekurzívne generovanie vzorov:**
   - Pre každú častú sekvenciu α sa identifikujú jej výskyty v jednotlivých sekvenciách.
   - Pomocou binárneho vyhľadávania sa určí, ktoré položky sa ešte nachádzajú za poslednou pozíciou prefixu.
   - Každá z týchto položiek sa môže použiť na vytvorenie (k+1)-sekvencie.
   - Funkcia sa rekurzívne volá pre každý novo vzniknutý vzor.
4. **Rozšírenia vzoru:**
   - **I-rozšírenie:** pridanie položky do poslednej udalosti.
   - **S-rozšírenie:** pridanie novej udalosti s položkou (iba ak sa nachádza po poslednej pozícii).

Príklad:

Uvažujme sekvenčnú databázu 3.9 a minimálnu podporu 50 %.

ID	Sekvencia
S1	⟨(A) (C) (B C) (D) (A B C) (A) (D)⟩
S2	⟨(B) (C D) (A) (C) (B D)⟩
S3	⟨(D) (B C) (A C) (C D)⟩

Tabuľka 3.9: Sekvenčná databáza pre príklad

Pre každú položku sa určí jej posledná pozícia v každej sekvencii:

- S1: $A = 6$, $B = 5$, $C = 5$, $D = 7$

- S2: $A = 3$, $B = 5$, $C = 4$, $D = 5$
- S3: $A = 3$, $B = 2$, $C = 4$, $D = 4$

Na základe frekvencie výskytu jednotlivých položiek algoritmus identifikuje časté 1-sekvencie. Napríklad položka A sa vyskytuje v každej zo sekvencií, preto tvorí častý vzor $\langle(A)\rangle$.

Algoritmus následne vyhodnocuje, ktoré položky sa nachádzajú za výskytom vzoru $\langle(A)\rangle$ v jednotlivých sekvenciách. V S1 je prvý výskyt A na pozícii 1. Keďže v tejto sekvencii sa položky B, C a D vyskytujú až po tejto pozícii, sú kandidátmi na rozšírenie vzoru. V S2 sa A nachádza na pozícii 3, za ktorou nasledujú ďalšie výskyty C, B a D. Aj tu je teda možné vzor rozšíriť o tieto položky. V S3 sa A vyskytuje v tretej udalosti a aj tu nasleduje ďalší výskyt C a D.

Z týchto zistení vyplýva, že všetky položky B, C aj D sa nachádzajú po prvom výskyte A v aspoň dvoch sekvenciách. Preto môžeme rozšíriť pôvodný vzor $\langle(A)\rangle$ napríklad na vzory $\langle(A)(B)\rangle$, $\langle(A)(C)\rangle$ a $\langle(A)(D)\rangle$ pomocou S-rozšírenia, alebo na $\langle(A\ B)\rangle$ či $\langle(A\ C)\rangle$ pomocou I-rozšírenia, ak sa položka vyskytuje v tej istej udalosti ako A.

Po vytvorení každého rozšíreného vzoru algoritmus znovu zisťuje jeho podporu a pokračuje rekurzívne. V každom ďalšom kroku sa opäť sleduje, ktoré položky sa v jednotlivých sekvenciách nachádzajú za aktuálnym prefixom.

Algoritmus ukončí vetvu vtedy, keď pre daný vzor neexistuje žiadna položka, ktorá by sa nachádzala za prefixom aspoň v dvoch rôznych sekvenciách. Takáto situácia znamená, že vzor už nemožno ďalej častejšie rozšíriť bez porušenia podmienky minimálnej podpory.

Kapitola 4

Aplikácia

V rámci práce je implementovaná desktopová aplikácia v jazyku Python[1], ktorá užívateľovi poskytuje prostredie pre dolovanie sekvenčných vzorov. Hlavnými úlohami aplikácie je umožniť užívateľovi nahratie vstupného súboru, spustenie analýzy (dolovania sekvenčných vzorov) so zvoleným algoritmom a určenými parametrami. Po úspešnej analýze je vytvorený súbor s nájdenými častými vzormi.
Adresárová štruktúra aplikácie je nasledovná:

```
implementacia/
├── datasets/
├── lib/
├── outputs/
├── resources/
└── src/
    ├── data/
    ├── algo/
    ├── gui/
    └── database/
```

Implementačne najpodstatnejšími časťami sú súbory uložené v zložke src/, ktorých funkcionalita je popísaná v Sekciách 4.1 až 4.3. Na záver kapitoly je v sekcii 4.4 uvedené ako aplikáciu používať a aký je očakávaný formát vstupov a výstupov aplikácie.

4.1 Spracovanie dát

Spracovanie užívateľom nahraného súboru je v aplikácii implementované v zložke `src/data/` a pozostáva z dvoch hlavných krokov: načítania a predspracovania surového súboru, a transformácie do formátu sekvenčnej databázy, ktorý je vyžadovaný algoritmami dolovania sekvenčných vzorov.

Načítanie súboru

Aplikácia dokáže načítať súbory formátov CSV, TXT alebo DATA. Modul `data_loader.py` využíva knižnicu *pandas*[2] na načítanie vstupného súboru a prevod dát do tabuľkovej štruk-

[1] https://www.python.org/
[2] https://pandas.pydata.org/

túry `DataFrame`, ktorá umožňuje jednotnú a jednoduchú prácu s rôznymi formátmi pôvodných vstupných dát.

Konverzia na sekvenčnú databázu

V prípade CSV súboru sa po úspešnom načítaní posiela vytvorený `DataFrame` do modulu `sequence_db_converter.py`, kde sa vykonáva následovný proces:

1. **Kódovanie unikátnych prvkov sekvencie:** Všetky položky (resp. množiny položiek) v stĺpci so sekvenciami sa zoskupia a premapujú na kladné celé čísla..
2. **Oddelenie prvkov, množín prvkov a sekvencií:** Užívateľ po nahraní súboru udá akým spôsobom sú v danom súbore oddelené prvky (užívateľom zadaná hodnota *Item Separator*), sekvencie a množiny prvkov sekvencie (užívateľom zadaná hodnota *Itemset Separator*). Za ukončenie sekvencie sa berie koniec riadku danej sekvencie a vo vytvorenej sekvenčnej databáze je na koniec každej sekvencie pridaná značka `-2`, prvky sekvencie sú transformované na kladné celé čísla, oddelené medzerou a za každú množinu prvkov je vložené značka `-1`, tak aby bola vytvorená sekvenčná databáza v súlade s formátom SPMF.
3. **Ukladanie sekvenčnej databázy:** Nakoniec sa výsledné riadky vypisujú do súboru, ktorý predstavuje sekvenčnú databázu a je vhodný pre aplikovanie algoritmov. Taktiež sa ukladá slovník kódovania, ktorý obsahuje informácie o tom akým spôsobom boli prvky sekvencie zakódované, aby bolo vo neskôr možné spätné dekódovanie nájdených vzorov na pôvodné prvky.

Tabuľky 4.2 a 4.1 ilustrujú praktický príklad spôsobu, akým aplikácia kóduje a prekladá pôvodné prvky sekvencie do formátu sekvenčnej databázy.

Pôvodné dáta	**Sekvenčná DB**
ABCDE	1 -1 2 -1 3 -1 4 -1 5 -1 -2
BCDEF	2 -1 3 -1 4 -1 5 -1 6 -1 -2
AABBC	1 -1 1 -1 2 -1 2 -1 3 -1 -2
CDABE	3 -1 4 -1 1 -1 2 -1 5 -1 -2
BCCAD	2 -1 3 -1 3 -1 1 -1 4 -1 -2

Tabuľka 4.1: Transformácia pôvodných dát do formátu sekvenčnej databázy (Použitý oddelovač prvkov = „“ a oddelovač množín prvkov = „“)

Znak	**Kód**
A	1
B	2
C	3
D	4
E	5
F	6

Tabuľka 4.2: Kódovanie prvkov na kladné celé čísla

4.2 Implementácia algoritmov

Implementácia algoritmov je v rámci aplikácie realizovaná s využitím knižnice **SPMFpy**, ktorá slúži ako prepojovací modul (tzv. wrapper), ktorý umožňuje využívať knižnicu SPMF [5] z prostredia Pythonu. Táto knižnica je v praxi najčastejšie využívanou knižnicou v oblasti dolovania sekvenčných vzorov, ktorá ponúka k dátumu vyhotovenia práce cez štyridsať rôznych algoritmov podporujúcich tento druh dolovania, pričom osem z nich slúži pre základný typ dolovania sekvenčných vzorov, teda pre hľadanie podskenvecií, ktoré sa často objavujú sekvenciách. Ostatné algoritmy sú zamerané na dolovanie uzavretých sekvenčných vzorov, maximálnych sekvenčných vzorov, top-k sekvenčných vzorov a podobne. Aplikácia podporuje práve týchto osem základných algoritmov dolovania sekvenčných vzorov, ktoré sú tiež popísané v sekcii 3.4.

Hlavnou myšlienkou je, že vytvorená sekvenčná databáza4.1 s pripravenými dátami slúži ako vstup pre spustenie externého Java [3] procesu, ktorý obsahuje implementáciu algoritmu.

Spustenie algoritmu:

Spúšťanie algoritmu sa deje cez príkazový riadok typicky pomocou `subprocess.Popen` s parametrami

```
java -jar spmf.jar run <algoritmus> <vstup> <výstup> <min_podpora>
```

Z užívateľského hľadiska je pre spustenie požadované zvolenie názvu algoritmu a minimálnej podpory, ktoré sú predané ako parametre `<algoritmus>` a `<min_podpora>`. Parametrom `<vstup>` je názov súboru, ktorý užívateľ nahral a `<výstup>` je názov súboru vygenerovaný na základe názvu vstupného súboru.

Izolácia procesu:

Samotný Java proces je spúšťaný v rámci aplikácie vo vlastnom `multiprocessing.Process`. Dôvodom za týmto riešením je, že algoritmy dolovania sekvenčných vzorov môžu byť výpočetne aj časovo náročné, čo často spôsobuje nepríjemnosti pri používaní aplikácie. Výhodou obalenia Java procesu do nového procesu, nad ktorým máme plnú kontrolu je, že sme schopní pomocou zdieľaného slovníka priebežne kontrolovať, či užívateľ žiada o zrušenie procesu. Toto riešenie tiež užívateľovi umožňuje užívateľovi plynulejšiu interakciu s aplikáciou počas behu algoritmu a zabraňuje zasekávaniu sa aplikácie.

4.3 Grafické užívateľské rozhranie

Pre tvorbu grafického užívateľského rozhrania aplikácie je využitá knižnica PyQt5, ktorá je pythonovským wrapperom pre Qt5, čo je moderný multiplatformový framework pre tvorbu desktopových aplikácií, ktorý poskytuje systém widgetov, layoutov a mechanizmov pre reakcie na udalosti [11].

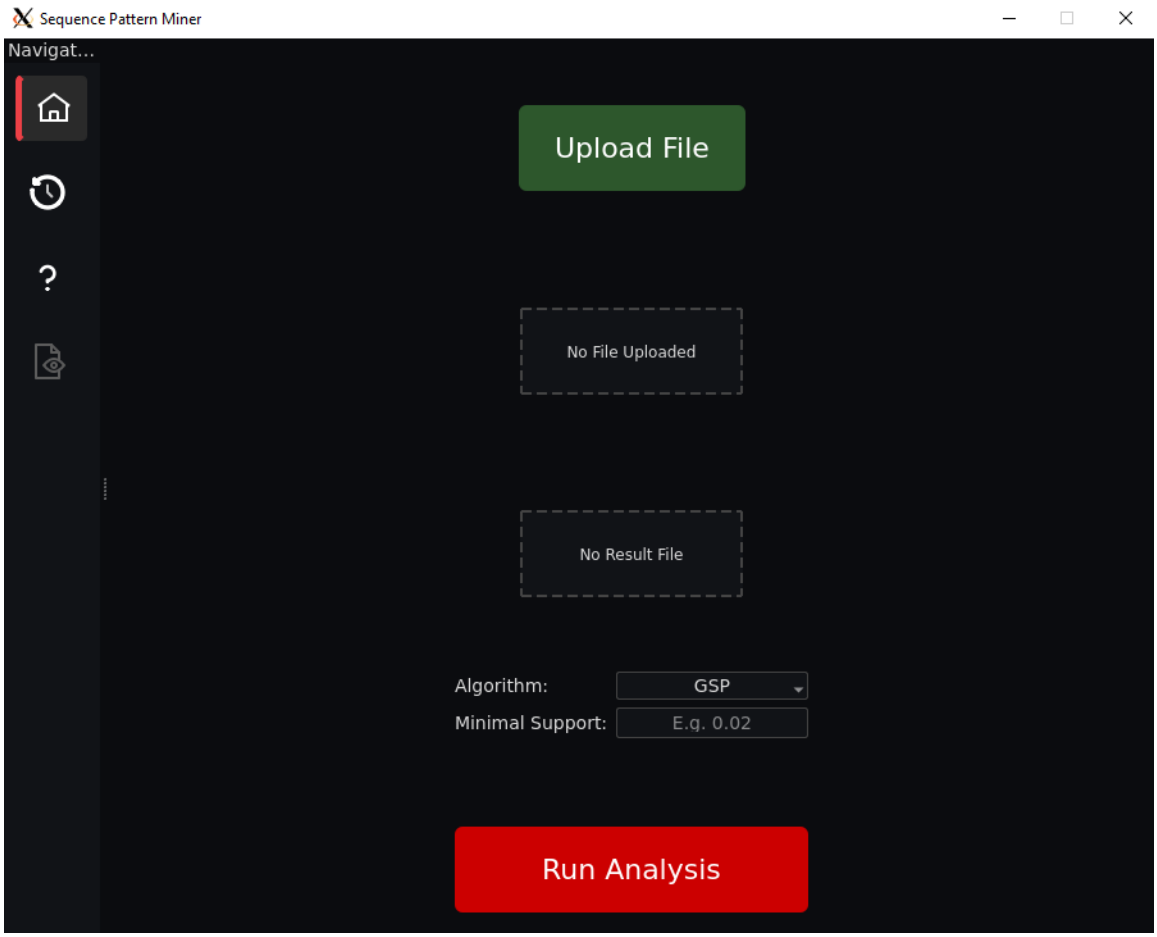
Najdôležitejšiou stránkou vrámci užívateľského prostredia je domovská stránka, ktorá slúži na vykonávanie analýzy. Užívateľ tu môže:

- nahrať vstupný súbor,

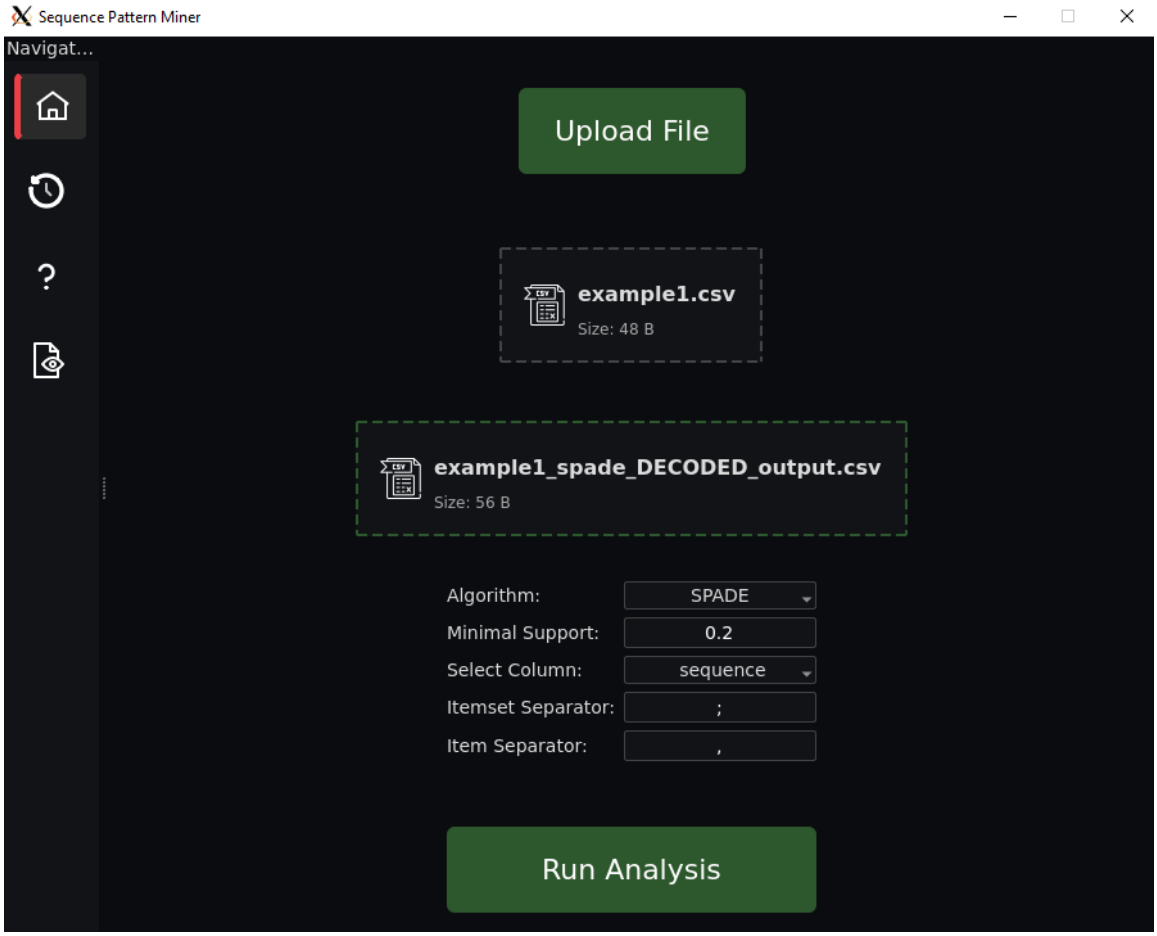
[3] https://www.java.com/en/

- zvoliť parametre analýzy -– minimálnu podporu a názov algoritmu,
- určiť informácie o súbore (v prípade CSV) – názov stĺpca so sekvenciami, oddeľovač prvkov v sekvencii a oddeľovač množín položiek.

Na obrázkoch 4.1 a 4.2 je ukážka výzoru domovskej stránky po spustení aplikácie a po úspešnom priebehu analýzy.



Obr. 4.1: Domovská stránka pred nahratím súboru.



Obr. 4.2: Domovská stránka po úspešnom vykonaní analýzy.

Okrem toho ponúka užívateľské rozhranie aj stránky pre:

- Zobrazenie nápovedy k používaniu
- Zobrazenie histórie nahratých súborov
- Zobrazenie metadát súborov, spolu s výsledkami analýz, ktoré boli na daných súboroch prevedené
- Zobrazenie výsledného súboru analýzy

4.4 Vstupy a výstupy

Aplikácia je navrhnutá primárne na spracovanie CSV súborov, pričom podporuje aj formáty TXT a DATA. Pri analýze TXT alebo DATA súborov aplikácia predpokladá, že sú vo formáte sekvenčnej databázy podľa špecifikácie SPMF [5].

Pre CSV vstupy aplikácia automaticky vygeneruje súbor so sekvenčnou databázou (`.txt`) v tom istom adresári, kde sa nachádza pôvodný CSV súbor. Týmto spôsobom sa naskytuje možnosť pri viacnásobnej analýze rovnakého súboru využívať už vygenerovanú

sekvenčnú databázu a ušetriť čas, ktorý aplikácia potrebuje na transformáciu pôvodného CSV súboru do formátu sekvenčnej databázy, čo môže byť vhodné najmä pri väčších vstupných súboroch.

Po úspešnom dokončení analýzy CSV súboru aplikácia vytvorí dva výstupné súbory:

1. **Textový SPMF výstup**: obsahuje sekvenčné vzory zakódované kladnými celými číslami (ukončenie položky „`-1`“, ukončenie sekvencie „`-2`“).
2. **Dekódovaný CSV výstup**: počas konverzie CSV na sekvenčnú databázu sa vytvorí slovník, v ktorom sa uchováva informácia o tom, ako sú prvky zakódované, ktorý sa po analýze CSV súboru použije pri dekódovaní výsledkov analýzy späť na pôvodné hodnoty.

Pri priamej analýze súborov vo formáte TXT alebo DATA sa dekódovanie nevykonáva.

Postup inštalácie závislých knižníc a správny spôsob spustenia aplikácie sú popísané v súbore `README.md`, ktorý sa nachádza v koreňovom adresári aplikácie.

Kapitola 5

Experimenty

Cieľom nižšie popísaných experimentov je pozorovanie výkonnosti rôznych algoritmov pre dolovanie sekvenčných vzorov. Každý z experimentov porovnáva výkonnosť algoritmov s využitím iných vstupných dát, ktoré boli zozbierané buď z reálnych, verejne dostupných dát, alebo vygenerované pomocou generátora sekvenčných databáz, ktorý je súčasťou knižnice SPMF. Tieto umelo vygenerované sekvenčné databázy, tiež nazývané syntetické databázy, sú jedným z najvhodnejších zdrojov pre vykonávanie experimentov, keďže umožňujú vytvoriť databázu s ľubovoľnými požadovanými parametrami. Na druhej strane však neodrážajú charakter typických reálnych databáz, pretože sú príliš predvídatelné.

Základnými vlastnosťami vstupnej sekvenčnej databázy, ktoré majú najväčší vplyv na výkonnosť algoritmov, sú:

1. Priemerná dĺžka sekvencií v databáze
2. Počet sekvencií v databáze
3. Počet jedinečných položiek v databáze
4. Priemerný počet prvkov v množinách položiek

Experimenty sú zamerané na menšie až stredne veľké sekvenčné databázy s rôznymi hodnotami vyššie uvedených vlastností, pričom skúmanými vlastnosťami algoritmov sú:

1. Čas behu
2. Počet nájdených častých sekvenčných vzorov
3. Maximálna pamäťová záťaž

Počas experimentov bol každý algoritmus spustený na každej z databáz niekoľkonásobne (konkrétny počet opakovaní závisel od experimentu) a pre každý prípad bol z nameraných hodnôt vypočítaný aritmetický priemer, ktory bol následne použitý vo výsledkoch.

Aby bola zabezpečená maximálna presnosť merania, boli hodnoty časov behu a maximálnej pamäťovej záťaže zbierané priamo zo štandardného výstupu algoritmov knižnice SPMF.

5.1 Experiment 1

Tento experiment je zameraný na jeden zo základných problémov dolovania sekvenčných vzorov, ktorým je prípad, kedy vstupná databáza obsahuje malý počet unikátnych prvkov v

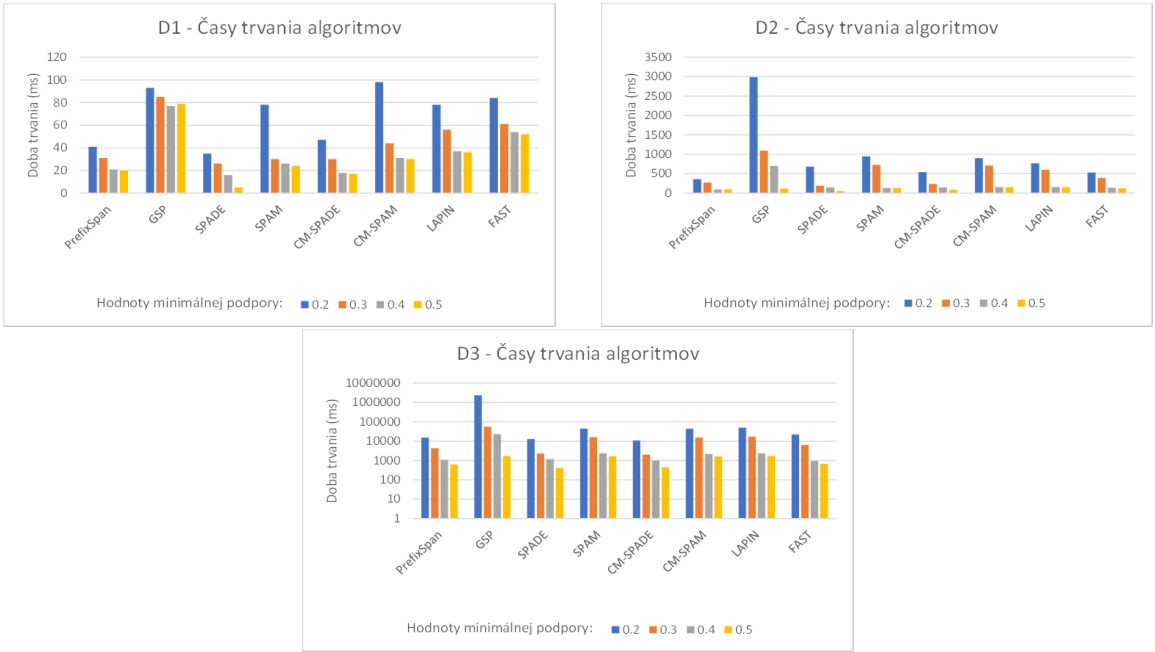
kombinácii s dlhými sekvenciami. Pre takúto kombináciu vlastností je typická vysoká miera opakovania sa položiek, čo vedie k veľkému počtu nájdených vzorov a exponenciálnemu zvyšovaniu výpočetnej náročnosti algoritmov.

V experimente boli testované tri vygenerované databázy, pričom každá z databáz obsahuje celkovo 200 sekvenciií, 20 unikátnych položiek a medzi databázami stúpa počet prvkov v sekvencii vždy o 20. Presné parametre každej z databáz sú zapísané v tabuľke 5.1. Každý z algoritmov bol spustený na každej z databáz desaťkrát.

Tabuľka 5.1: Parametre vygenerovaných sekvenčných databáz

Parameter	**D1**	**D2**	**D3**
Počet sekvencií	200	200	200
Počet prvkov v množinách položiek	1	1	1
Unikátne položky	20	20	20
Počet prvkov v sekvencii	20	40	60

Porovnanie rýchlostí algoritmov pri jednotlivých databázach je znázornené na obrázku 5.1. Z výsledkov experimentu vyplýva, že efektivita algoritmu GSP výrazne klesá s rastúcou dĺžkou sekvencií, čo môže byť priradené využívaniu apriori-prístupu, ktorý generuje veľké množstvo kandidátov, ktorých podporu je potreba vždy overiť. Čím sú sekvencie dlhšie a ich položky viac frekventované, tým väčší je vyhľadávací priestor a tým viac kombinácií musí algoritmus spracovať. Neefektivita GSP pri tomto druhu škálovania je vo výsledkoch experimentu jasne viditeľná.

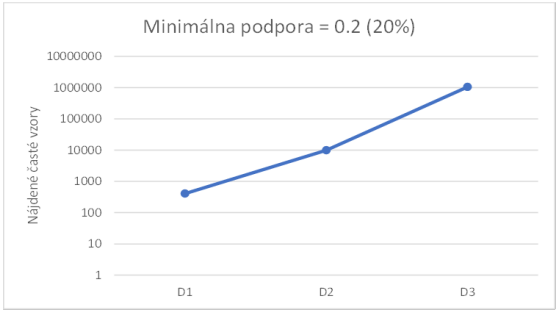


Obr. 5.1: Porovnanie rýchlostí algoritmov pre databázy D1, D2, D3

Naopak, algoritmy ako PrefixSpan, SPADE a CM-SPADE, ktoré nevyužívajú generovanie kandidátov alebo pracujú s vertikálnym formátom, si s narastajúcou zložitosťou databáz poradili podstatne lepšie. PrefixSpan bol najrýchlejší v prípade databáz D1 a D2, zatiaľ čo v prípade najnáročnejšej databázy D3 ho mierne prekonal CM-SPADE. Napríklad v prípade hodnoty minimálnej podpory rovnej 20 % boli v databáze D1 časy behov algoritmov

ešte relatívne vyrovnané a najrýchlejšie algoritmy boli približne dvojnásobne rýchlejšie ako najpomalšie algoritmy. Tento rozdiel sa prehĺbil v prípade databázy D2 na osemnásobne nižšiu hodnotu času behu pri porovnaní najrýchlejšieho (PrefixSpan) s najpomalším (GSP) algoritmom a v databáze D3 možno pozorovať ešte väčšie zmeny medzi časmi behov algoritmov, pričom najmenej efektívnym zostáva algoritmus GSP, ktorého beh bol viac ako 220-krát pomalší ako beh algoritmu CM-SPADE.

Rast výpočtovej náročnosti úzko súvisí s prudkým nárastom počtu častých vzorov. Pri minimálnej podpore 20 % sa v databáze D1 vyskytovalo len niekoľko stoviek častých vzorov, zatiaľ čo v D3 ich počet presiahol jeden milión. Exponenciálny nárast možno v tomto experimente jasne priradiť zväčšujúcemu sa počtu prvkov v jednotlivých sekvenciách v spojení s pomerne nízkym počtom unikátnych prvkov, čo spôsobuje, že sa prakticky každá podsekvencia vyskytne viackrát. Rast počtu nájdených vzorov je zobrazený na obrázku 5.2.



Obr. 5.2: Rast počtu nájdených vzorov pre databázy D1, D2, D3, pri minimálnej podpore 20 %

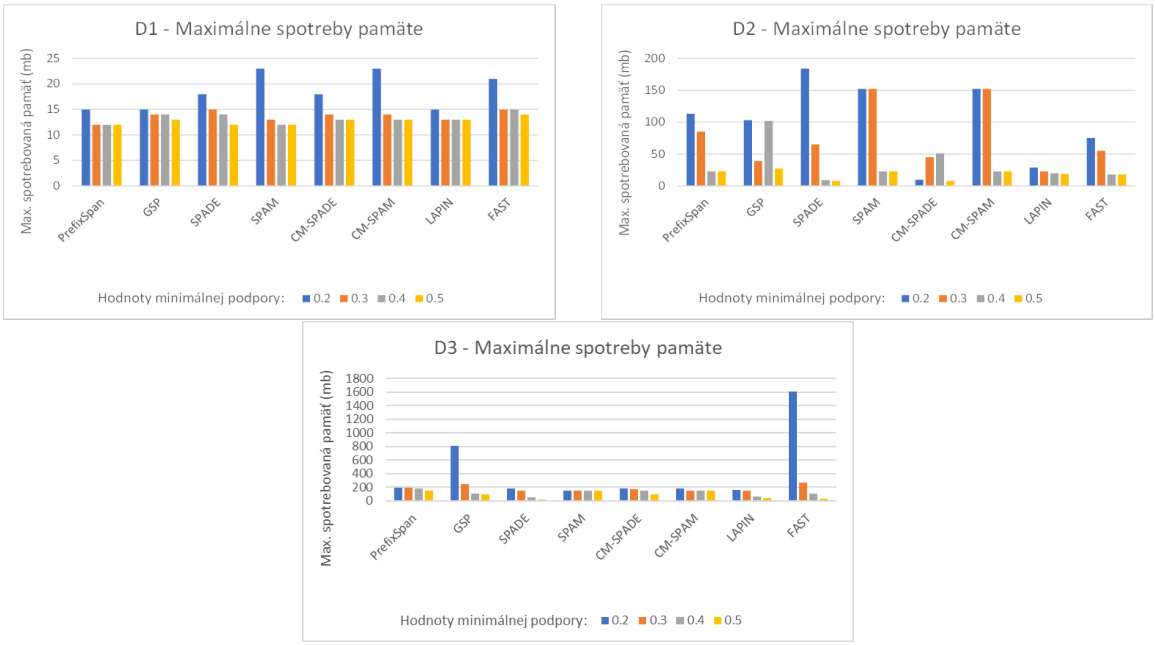
Pamäťová náročnosť algoritmov sa podobne ako čas spracovania zvyšovala s náročnosťou databáz, no jednotlivé algoritmy sa v tomto ohľade správali rozdielne. Porovnanie možno vidieť na obrázku 5.3. Za zmienku stojí algoritmus FAST, ktorý v prípade databázy D3, pri minimálnej podpore 20 %, naberá oproti ostatným algoritmom oveľa väčšiu pamäťovú záťaž, čo možno priradiť veľkému nárastu v počte vzorov, ktorý databáza obsahuje, v dôsledku čoho FAST generuje obrovské množstvo SIL a VIL zoznamov. Medzi efektívnejšie algoritmy sa z hľadiska pamäťovej záťaže radia CM-SPADE a LAPIN.

Podstatným poznatkom tiež je, že čím je databáza zložitejšia na dolovanie, tým väčšie sú rozdiely v efektivite medzi algoritmami. V prípade jednoduchej databázy D1 sú všetky algoritmy použiteľné s pomerne podobnými časmi spracovania aj pamäťovej, zatiaľ čo v databáze D3 už algoritmy založené na generovaní kandidátov, ako je GSP, prudko strácajú na svojej efektivite, zatiaľ čo algoritmy využívajúce prístup rastu vzorov a vertikálne algoritmy si zachovávajú akceptovateľný výkon.

Výsledky experimentu ukazujú, že pri databázach s malým počtom unikátnych položiek a dlhými sekvenciami, ktoré vedú k vysokému prekrývaniu vzorov, je potrebné zvoliť algoritmus, ktorý dokáže pracovať efektívne, bez potreby generovať veľké množstvo kandidátov. V takýchto prípadoch sú algoritmy ako PrefixSpan, SPADE a CM-SPADE vhodnejšou voľbou ako tradičné apriori-prístupy.

5.2 Experiment 2

Pri tomto experimente sú porovnávané výkonnosti algoritmov na dvoch rôznych databázach, ktorých spoločným znakom je netypicky veľký počet prvkov v množinách prvkov jednotli-



Obr. 5.3: Porovnanie maximálnej spotreby pamäte algoritmov pre databázy D1, D2, D3

vých sekvencií. Algoritmy boli spustené pre každú databázu pri každej úrovni minimálnej podpory päťkrát.

Internetový obchod s oblečením (D1)

V prvej časti experimentu bola použitá sekvenčná databáza založená na reálnych dátach o kliknutiach používateľov na internetovom obchode predávajúcom oblečenie. Základné vlastnosti databázy sú zhrnuté v tabuľke 5.2.

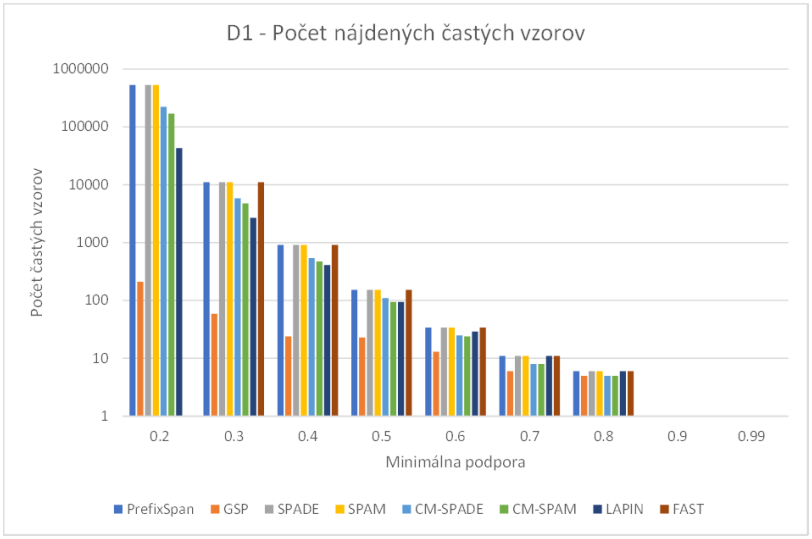
Počet sekvencií	24 026
Počet unikátnych položiek	317
Priemerný počet položiek v množine položiek	9.0
Priemerný počet položiek množín v sekvencii	6.8

Tabuľka 5.2: Vlastnosti sekvenčnej databázy D1

Zaujímavou charakteristikou tejto databázy je najmä fakt, že v rámci jednej množiny položiek obsahuje až deväť položiek, pričom typicky sa v sekvenčných databázach pohybuje táto hodnota medzi nulou a troma položkami. V rámci experimentu bola na tejto databáze sledovaná výkonnosť algoritmov v závislosti od meniacej sa hodnoty minimálnej podpory v rozsahu od 20 % do 99 %.

Rozdiel v počte nájdených častých vzorov jednotlivými algoritmami je pozorovateľný už pri minimálnej podpore nastavenej na 80 %, kedy niektoré z algoritmov našli päť častých vzorov a iné šesť. S klesajúcou minimálnou podporou sa tento rozdiel neustále zväčšuje, až pri 20 % je možné vidieť, že algoritmy ako PrefixSpan, SPADE a SPAM vygenerovali viac ako 520 000 až 650 000 vzorov, zatiaľ čo CM-SPADE, CM-SPAM a LAPIN našli podstatne menej (od 42 000 do 222 000). Extrémnym prípadom je GSP, ktorý našiel len 209 vzorov, čo je viac ako 2 500-krát menej než PrefixSpan.

Tieto rozdiely možno pripísať odlišnému prístupu algoritmov k reprezentácii vzorov, napríklad PrefixSpan, SPADE a SPAM generujú všetky časté vzory, teda aj podvzory, ktoré sú z pohľadu informačnej hodnoty redundantné, ale splňujú prah podpory. Algoritmy ako CM-SPADE a CM-SPAM sú schopné vďaka CMAP štruktúram odstraňovať nekvalitné kandidátne sekvencie a vyhýbať sa tak nájdeniu redundantných častých vzorov. GSP je apriori-založený algoritmus, ktorý vykazuje v hustých databázach slabý výkon. Kvôli rýchlemu prerezávaniu kandidátov sa mnohé časté vzory ani nevygenerujú a nájdených častých vzorov tak môže byť extrémne málo oproti ostatným algoritmom. Rozdiely medzi počtami nájdených častých vzorov pri rôznych algoritmoch sú zobrazené na obrázku 5.4.



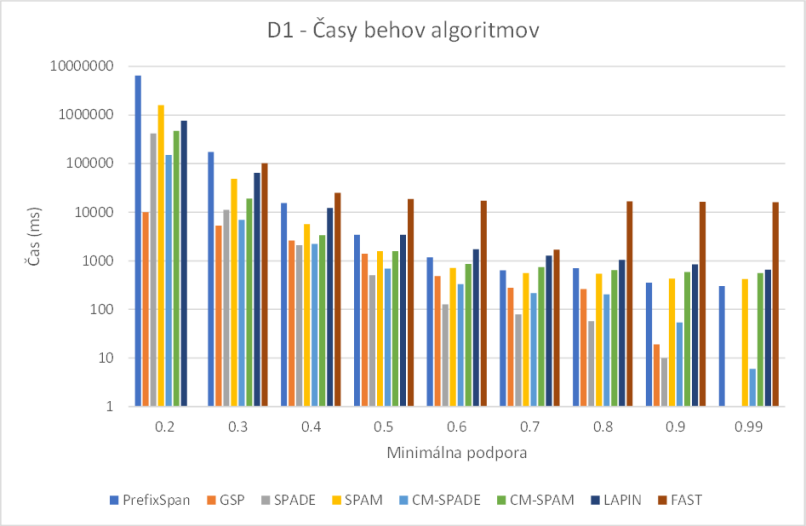
Obr. 5.4: Počty nájdených vzorov v D1, pri meniacej sa minimálnej podpore

Z hľadiska rýchlosti boli najefektívnejšími algoritmy GSP, SPAM a CM-SPADE, no ich výkonnosť závisela od úrovne podpory, ako možno vidieť na obrázku 5.5. GSP bol veľmi rýchly najmä v prípadoch, keď našiel málo vzorov, avšak nie preto, že by bol výkonný, ale preto, že spracoval len zlomok dát. CM-SPADE a SPAM si udržali dobrý výkon aj pri nižšej podpore, čo možno prisúdiť efektivite vertikálneho formátu a projekcie. Najpomalším algoritmom bol PrefixSpan a to najmä pri najnižšej testovanej hodnote minimálnej podpory, 20 %, kedy výpočet trval približne 108 minút, pričom pre porovnanie algoritmus GSP pracoval približne 10 sekúnd.

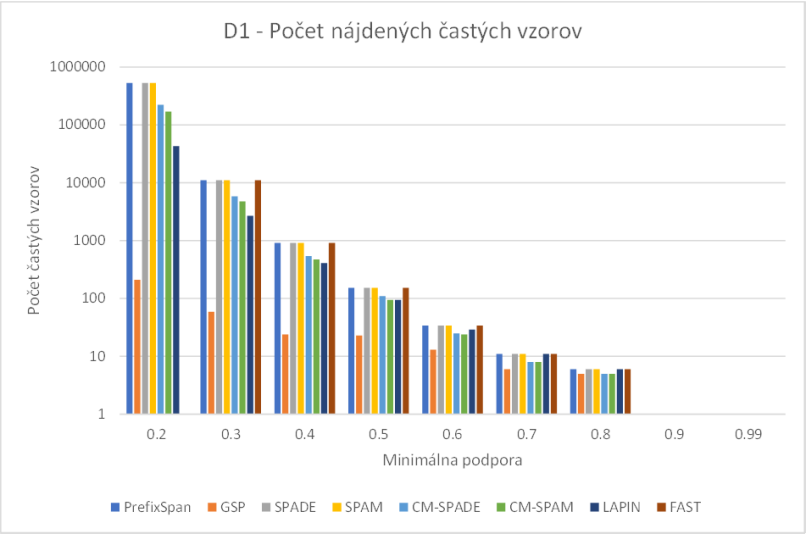
V rámci spotreby pamäte sa medzi najmenej efektívne radí FAST, ktorý konzistentne spotrebovával najväčšie množstvo pamäte a v prípade minimálnej podpory rovnej 20 % dokonca kvôli presiahnutej miere spotreby pamäte nebol schopný dáta spracovať. Naopak, najefektívnejším algoritmom z pohľadu spotreby pamäte bol SPAM. Hodnoty sú zobrazené na obrázku 5.6

D2 – Internetový obchod s darčekovým sortimentom

V rámci pokračovania experimentu bola testovaná druhá sekvenčná databáza, ktorej údaje predstavujú transakcie v maloobchode s darčekovým sortimentom. Táto databáza na rozdiel od D1 z predošlej časti experimentu, obsahuje oveľa väčší počet unikátnych položiek, čo má za následok, že sa položky tak často neopakujú a databáza D2 je tak oveľa ľahšia pre



Obr. 5.5: Časy behov algoritmov v D1, pri meniacej sa minimálnej podpore



Obr. 5.6: Hodnoty spotreby pamäte algoritmov v D1, pri meniacej sa minimálnej podpore

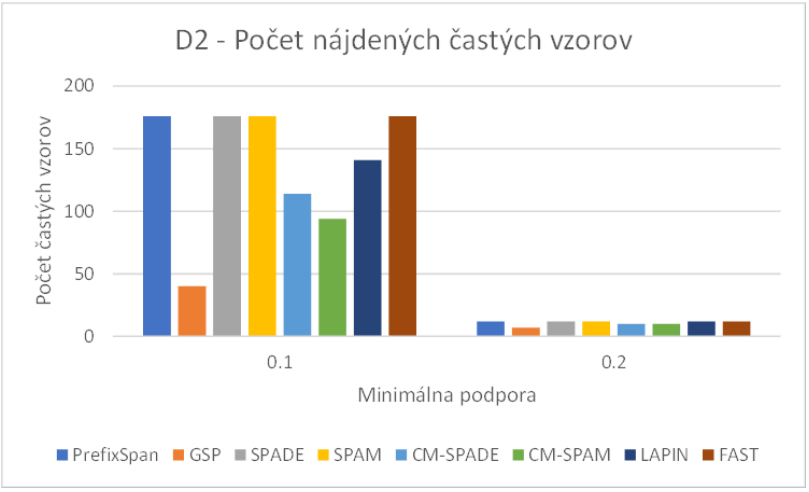
spracovanie algoritmami. Hodnota minimálnej podpory sa pri testovaní algoritmov menila v rozmedzí od 10 % do 99 %.

Počet sekvencií	4 383
Počet unikátnych položiek	41 431
Priemerný počet položiek v množine	9.0
Priemerný počet položiek množín v sekvencii	5.4

Tabuľka 5.3: Vlastnosti sekvenčnej databázy D2

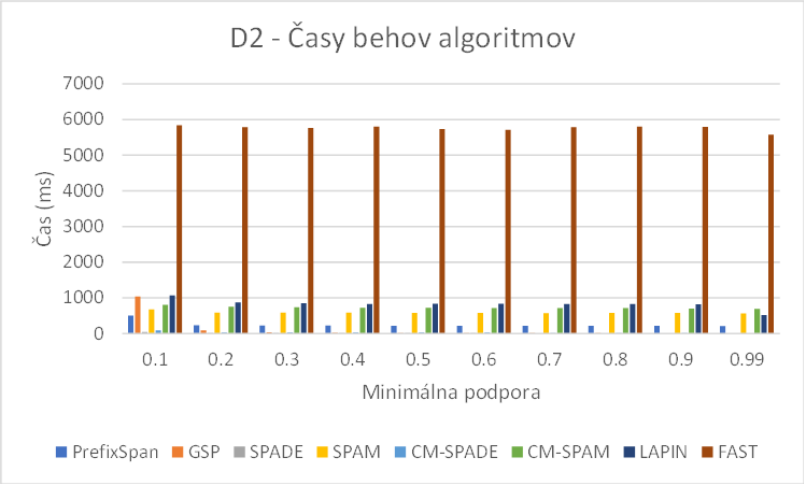
Pri analýze počtu nájdených vzorov 5.7 je zjavné, že algoritmy v tejto databáze nachádzali celkovo podstatne menší počet častých vzorov než v prípade databázy D1, avšak pri dostatočne nízkej minimálnej podpore je tiež možné pozorovať, že sa počty nájdených vzorov medzi jednotlivými algoritmami líšia. PrefixSpan, SPADE, SPAM a LAPIN aj v

tomto prípade nachádzajú najviac častých vzorov a poradie sa oproti databáze D1 nemení ani v prípade algoritmu GSP, ktorý nachádza vzorov najmenej.



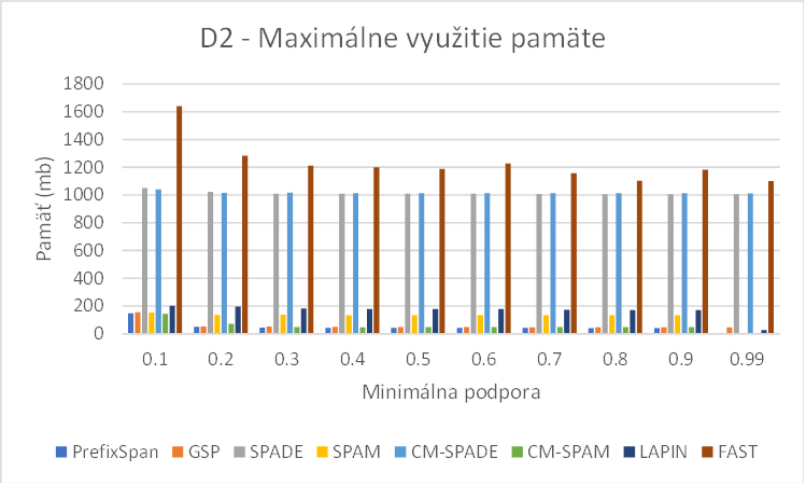
Obr. 5.7: Počty nájdených vzorov v D2, pri meniacej sa hodnote minimálnej podpory

Pri minimálnej podpore 20 % našli jednotlivé algoritmy v rozmedzí od 7 do 12 častých vzorov, čo je veľký rozdiel oproti viac ako pol miliónu vzorov v databáze D1, ktorý je zapríčinený primárne oveľa väčším počtom unikátnych položiek. Tento rozdiel sa prejavil aj na rýchlostiach algoritmov, ktoré sú zobrazené na obrázku 5.8. Z hľadiska výpočtových časov si v tejto databáze najlepšie viedli algoritmy SPADE a CM-SPADE. Zaujímavé tiež je, že algoritmus GSP, ktorý našiel najmenej častých vzorov a patrí pri minimálnych podporách 20 % až 99 % medzi najrýchlejšie algoritmy, je pri najnižšej úrovni minimálnej podpory 10 % už násobne pomalší, ako SPADE alebo CM-SPADE.



Obr. 5.8: Časy behov algoritmov v D2, pri meniacej sa minimálnej podpore

Najpomalším a zároveň pamäťovo najnáročnejším algoritmom bol FAST, ktorý sa v prípade tejto databázy ukázal ako veľmi neefektívne riešenie, zatiaľ čo pamäťovo najefektívnejšími boli algoritmy PrefixSpan a SPAM. Spotreba pamäte algoritmov v databáze D2 je zobrazená na obrázku 5.9.



Obr. 5.9: Hodnoty spotreby pamäte algoritmov v D2, pri meniacej sa hodnote minimálnej podpory

5.3 Experiment 3

Tento experiment je založený na porovnávaní výkonnosti algoritmov pri štyroch vygenerovaných databázach, pričom spoločnou vlastnosťou týchto databáz je, že obsahujú rovnaký celkový počet položiek. Každý z algoritmov bol spustený pre každú z databáz päťkrát pri minimálnej podpore 5 % a 10 %. Úlohou tohto experimentu je ukázať, ako sa jednotlivé algoritmy správajú pri rôznych druhoch škálovania vstupnej sekvenčnej databázy.

Databázy D1 a D2 sú škálované so zvýšením počtu sekvencií a obsahujú päťnásobne viac sekvencií ako databázy D3 a D4. Databázy D3, D4 sú škálované so zameraním na počet prvkov v sekvenciách a obsahujú 3-násobne väčší počet položiek v každej sekvencií. Podrobnejší popis parametrov databáz možno vidieť v tabuľkách 5.4 a 5.5.

Rozbor výkonnosti algoritmov je v tomto experimente rozdelený na 2 časti. Prvou je časť popisujúca algoritmy pri databázach s väčším počtom sekvencií (D1, D2) a druhou je časť popisujúca algoritmy pri databázach s väčším počtom prvkov v sekvenciách (D3, D4).

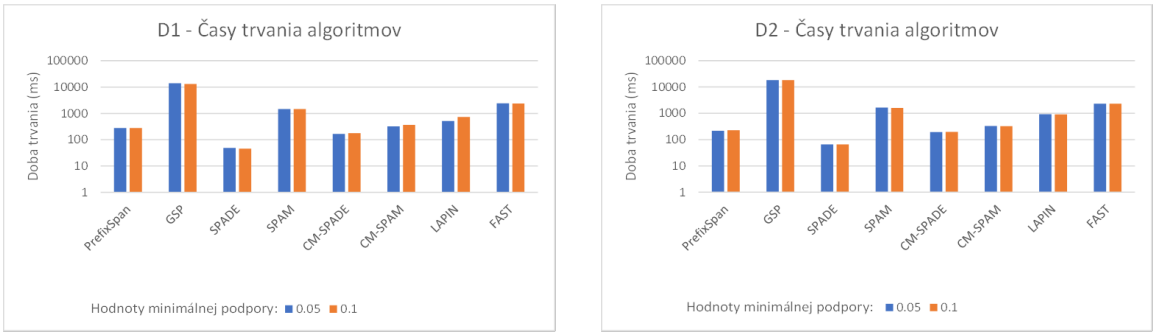
Databázy s väčším počtom sekvencií

Parameter	D1	D2
Počet sekvencií	10 000	10 000
Počet prvkov v množinách položiek	3	1
Unikátne položky	100	100
Počet množín položiek v sekvencií	5	15
Celkový počet položiek	150 000	150 000

Tabuľka 5.4: Parametre vygenerovaných sekvenčných databáz D1, D2

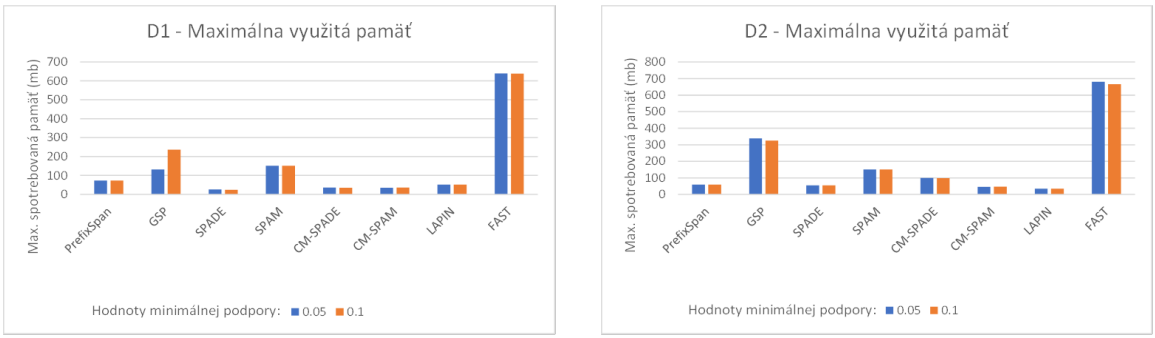
Z výsledkov experimentov znázornených na obrázku 5.10 možno pozorovať, že najrýchlejším algoritmom bol v prípade oboch databáz SPADE. Táto efektivita vyplýva z využitia vertikálneho formátu údajov a ID-zoznamov, ktoré umožňujú vykonávať výpočty bez potreby generovania veľkého množstva kandidátov. Naopak, algoritmus GSP dosahuje výrazne najhoršie časy behov. Dôvodom je jeho apriori-prístup, ktorý vyžaduje generovanie všet-

kých kandidátskych sekvencií a ich opakované overovanie, čo je časovo veľmi náročné najmä pri väčšom počte sekvencií a hustejších databázach.



Obr. 5.10: Porovnanie časov vykonania algoritmov pre databázy D1 a D2

Na obrázku 5.11 sú zobrazené hodnoty maximálnej spotrebovanej pamäte jednotlivými algoritmami. Výsledky ukazujú konzistentne najväčšie pamäťové zaťaženie v prípade algorimus FAST a inými menej efektívnymi algoritmami z tohto hľadiska sú GSP a SPAM, aj keď oba dosahujú násobne menšie hodnoty ako tomu je v prípade algoritmu FAST. Najlepšie výsledky dosahujú SPADE a LAPIN, aj keď algoritmy CM-SPADE a CM-SPAM vykazujú iba nevýrazne vyššiu pamäťovú náročnosť.



Obr. 5.11: Porovnanie maximálnej spotreby pamäte pre databázy D1 a D2

Počet nájdených častých vzorov v oboch databázach, pri oboch hodnotách podpory bol 100.

Databázy s väčším počtom prvkov v sekvenciách

Táto časť je zameraná na databázy, ktoré boli vygenerované s rovnakým počtom celkových a unikátnych položiek ako pri databázach D1 a D2, obsahujú však päťnásobne menší počet sekvencií, pričom každá sekvencia obsahuje podstatne viac položiek. Detailnejší popis parametrov databáz D3 a D4 možno vidieť v tabuľke 5.5.

Vo výsledkoch experimentu možno vidieť, že dolovanie takýchto databáz je podstatne zložitejšie z dôvodu vyššej zložitosti jednotlivých sekvencií. Hlavnou príčinou tejto zväčšenej zložitosti je, že dlhé sekvencie majú väčší počet kombinácií podsekvencií, čo vedie k podstatnému zväčšeniu vyhľadávacieho priestoru.

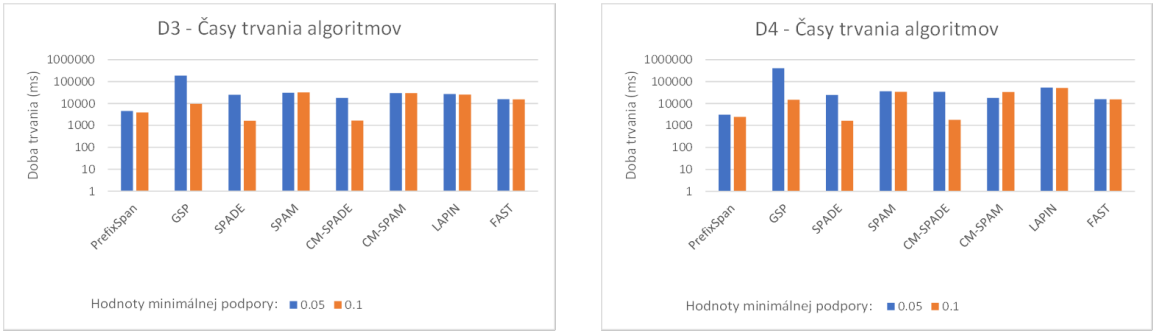
Pre porovnanie – predchádzajúce dve databázy D1, D2 obsahovali pri oboch hodnotách podpory iba 100 častých vzorov. V prípade databáz D3 a D4 bolo nájdených všetkými

Parameter	D3	D4
Počet sekvencií	2 000	2 000
Počet prvkov v množinách položiek	3	1
Unikátne položky	100	100
Počet množín položiek v sekvencií	25	75
Celkový počet položiek	150 000	150 000

Tabuľka 5.5: Parametre vygenerovaných sekvenčných databáz D3, D4

algoritmami pri minimálnej podpore 10 % až 10 100 vzorov, zatiaľ čo pri minimálnej podpore 5 % narástol ich počet na 13 985 (D3) a 21 920 (D4).

Prvým pozorovaným parametrom algoritmov sú ich rýchlosti, ktoré sú zobrazené na obrázku 5.12. Pri hodnote minimálnej podpory 0.1 sa ako najrýchlejšie preukázali algoritmy SPADE a CM-SPADE. Toto správanie možno pripísať tomu, že tieto algoritmy využívajú vertikálny formát, čo je výhodné pri dlhších sekvenciách, s častým opakovaním prvkov. Veľmi dobré výsledky dosahuje aj PrefixSpan, ktorý dokonca v prípade minimálnej podpory 5 % z hľadiska rýchlosti výrazne prekonáva všetky ostatné algoritmy.



Obr. 5.12: Porovnanie časov trvania algoritmov pre databázy D3 a D4

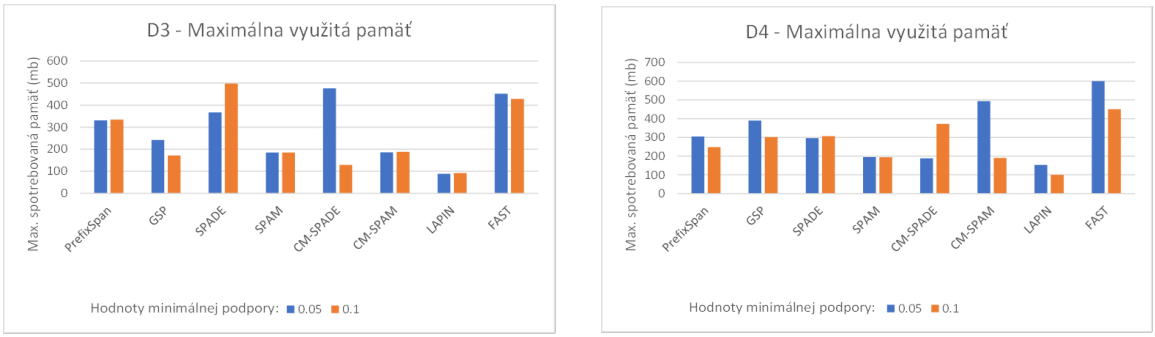
Zaujímavým prípadom je aj algoritmus GSP, ktorý pri minimálnej podpore 10 % vykazuje nadpriermné rýchlosti, no následne pri znížení podpory na 5 % jeho výkon veľmi prudko klesá a stáva sa konzistentne najpomalším algoritmom. Tento jav môže byť spôsobený tým, že GSP využíva apriori-generovanie kandidátov, čo vedie k obrovskému zvýšeniu kombinácií v prípade väčšieho počtu vzorov.

Z pohľadu maximálnej záťaže pamäte, ktorý je zobrazený na obrázku 5.13 možno vidieť, že aj pri týchto dvoch databázach je konzistentne najmenej efektívnym algoritmus FAST. Naopak, ako pamäťovo najefektívnejšie algoritmy sa v tomto prípade prejavujú LAPIN a SPAM.

5.4 Vyhodnotenie experimentov

Na základe troch uskutočnených experimentov možno pozorovať, že výkonnosť algoritmov na dolovanie sekvenčných vzorov je výrazne ovplyvnená štruktúrou a veľkosťou vstupnej databázy.

Algoritmy založené na generovaní kandidátov preukázali nízku efektivitu v hustých alebo rozsiahlych databázach, kde počet častých vzorov rástol exponenciálne. Naopak, algoritmy



Obr. 5.13: Porovnanie maximálnej spotreby pamäte pre databázy D3 a D4

ako PrefixSpan, SPADE, CM-SPADE a CM-SPAM sa ukázali ako spoľahlivejšie pri škálovaní, najmä v prípadoch dlhých sekvencií alebo veľkého množstva údajov.

SPADE a CM-SPADE boli často najefektívnejšie z hľadiska pamäťovej náročnosti a času, zatiaľ čo FAST bol vo väčšine prípadov pamäťovo najnáročnejší.

Z výsledkov tiež možno pozorovať, že nie je ľahké nájsť univerzálne najlepší algoritmus. Výber vhodného algoritmu závisí poznania vlastností vstupných dát, akými sú napríklad počet unikátnych položiek, dĺžka sekvencií a celková hustota výskytov.

Kapitola 6

Záver

Táto práca sa venovala problematike dolovania sekvenčných vzorov, pričom jej hlavnými cieľmi bolo zoznámenie čitateľa so základnými a často používanými pojmami v tejto oblasti, ako aj oboznámenie s problematikou širšieho kontextu procesu dolovania dát. Pozornosť bola venovaná formálnej definícii pojmov ako sekvencia, množina položiek, častý vzor a podpora, ktoré tvoria základ pre správne pochopenie fungovania jednotlivých algoritmov.

Dôležitou časťou práce bol popis vybraných algoritmov určených na dolovanie sekvenčných vzorov, medzi ktoré patrili klasické aj moderné prístupy. Algoritmy ako GSP, PrefixSpan, SPADE, či novšie optimalizácie ako CM-SPADE, CM-SPAM, FAST a LAPIN boli rozobrané najskôr po teoretickej stránke a neskôr otestované praktickými experimentmi.

V experimentálnej časti boli výkonnosti algoritmov testované a pozorované na rôznych typoch vstupných databáz, pričom boli porovnávané z hľadiska výpočtového času, pamäťovej náročnosti a počtu nájdených vzorov. Výsledky experimentov ukázali, že vhodnosť použitia konkrétneho algoritmu závisí najmä od vnútornej štruktúry databázy.

Súčasťou práce bol aj popis aplikácie, ktorá umožňuje užívateľovi vykonávať analýzu dát s použitím jednoduchého grafického rozhrania.

Práca tak poskytuje nielen prehľad možností v oblasti dolovania sekvenčných vzorov, ale aj praktický pohľad na ich aplikáciu a výkonnosť v rôznych podmienkach. Výsledky experimentov môžu byť užitočné napríklad pri výbere vhodného algoritmu pre konkrétne dolovacie úlohy.

Literatúra

[1] Ayres, J.; Gehrke, J.; Yiu, T. a Flannick, J. Sequential PAttern Mining using A Bitmap Representation. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, s. 429–435.

[2] Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM*. ACM New York, NY, USA, 1970, zv. 13, č. 6, s. 377–387.

[3] Fournier Viger, P.; Gomariz, A.; Campos, M. a Thomas, R. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*. Springer, 2014, sv. 8443, s. 40–52. Lecture Notes in Artificial Intelligence.

[4] 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.

[5] Fournier Viger, V. *SPMF: A Java Open-Source Pattern Mining Library* http://www.philippe-fournier-viger.com/spmf. 2014.

[6] Han, J.; Kamber, M. a Pei, J. *Data Mining: Concepts and Techniques*. 3rd. Burlington, MA, USA: Morgan Kaufmann, 2011. ISBN 9780123814791.

[7] Han, J.; Pei, J. a Kamber, M. *Data Mining: Concepts and Techniques*. 4th. Elsevier, 2022.

[8] Inmon, W. H. *Building the data warehouse*. John wiley & sons, 2005.

[9] Mabroukeh, N. R. a Ezeife, C. I. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys (CSUR)*. ACM New York, NY, USA, 2010, zv. 43, č. 1, s. 1–41.

[10] 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*. IEEE, 2004, zv. 16, č. 10, s. 1424–1440.

[11] Riverbank Computing Limited. *PyQt5*. 2016. Dostupné z: https://www.riverbankcomputing.com/software/pyqt/. Python bindings for the Qt application framework.

[12] Salvemini, E.; Fumarola, F.; Malerba, D. a Han, J. FAST Sequence Mining Based on Sparse ID-Lists. In: *International Symposium on Methodologies for Intelligent Systems (ISMIS)*. Springer, 2011, sv. 6804, s. 316–325. Lecture Notes in Computer Science.

[13] Srikant, R. a Agrawal, R. Mining Sequential Patterns: Generalizations and Performance Improvements. In: *Proceedings of the 5th International Conference on Extending Database Technology (EDBT)*. Springer, 1996, sv. 1057, s. 3–17. Lecture Notes in Computer Science.

[14] Yang, Z.; Wang, Y. a Kitsuregawa, M. LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. In: Springer. *Advances in Databases: Concepts, Systems and Applications: 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007. Proceedings 12*. 2007, s. 1020–1023.

[15] Zaki, M. J. SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*. Springer, 2001, zv. 42, č. 1, s. 31–60.

Príloha A

Obsah priloženého pamäťového média

```
xmikul74-bp/
├── implementacia/ ...Zdrojový kód implementácie
├── text/ ...Latex zdrojový kód textu
└── xmikul74-bp.pdf ...Text bakalárskej práce
```