Objevování temporálních vzorů

Projekt do předmětu Vybrané problémy informačních systémů (VPD)

Martin Hlosta

Fakulta informačních technologií,
Vysoké učení technické v Brně,
Božetěchova 2, Brno

Abstrakt Tento článek je zaměřen na širší seznámení s problematikou dolování v temporálních datech, jenž bývá v literatuře označováno také jako temporal mining. Z této oblasti se konkrétně zaměřuje na objevování temporálních vzorů (temporal pattern discovery). V článku jsou zmíněny typy dat, ve kterých se dají vzory dolovat a jaká specifika a problémy přináší do dolování uvažování časové domény. Stěžejní část pak představuje konkrétní typy temporálních vzorů společně s metodami a konkrétními algoritmy pro jejich objevování.

1 Úvod

Dlouhou dobu jsou data aplikací, např. také informačních systémů, uchovávána v databázových systémech. Tyto databáze začaly časem růst takovým způsobem, že se v nich sice skrývalo obrovské množství dat, ale nebyli jsme schopni tohoto množství využít, protože jsme z nich nebyli schopni získat znalosti. Typicky toto vystihuje citát Rutherforda D. Rogerse a~Johna Naisbitta *"We are drowning in data, but lacks for knowledge"* [6], tedy "topíme se v datech ale hladovíme po znalostech". Díky této potřebě nalézt v datech souvislosti nebo vzory začaly vznikat metody získávání znalostí z databází, někdy také nazývané podle hlavní fáze procesu získávání znalostí a to dolování z dat (*Data mining*).

Typickým příkladem takovýchto metod je asociační analýza, která se snaží v datech nalézat asociační pravidla. To jsou pravidla tvaru předchůdce→následník, která se v datech vyskytují dostatečně často. V reálném světě se nachází uplatnění v analýze nákupního košíku, díky čemuž jsme schopni zjistit, jaké zboží si zákazník kupuje pohromadě a v případě internetového obchodu při umístění jedné položky do nákupního košíku nabízet takové položky, které by jej mohly zajímat. Kromě asociační analýzy existuje celá řada úloh, které získávání znalostí z databází řeší. Řadí se tam především charakterizace, diskriminace, klasifikace, shlukování a hledání odlehlých hodnot .

Potřeba uchovávat nejen aktuální stav objektů v databázi, ale také jejich historii vedla na ukládání temporálních dat. Příkladem oborů, kde se taková data uchovávají, jsou bioinformatika, biomedicína, finančnictví nebo data získaná monitorováním v prostředí Internetu. Mezi nejběžnější typ temporálních dat se

řadí **časové řady**, čímž se rozumí zaznamenávané spojité hodnoty v pravidelných časových intervalech. Typickým příkladem časových řad pochází např. z finančnictví v podobě záznamu hodnot akcií či jiných ukazatelů. Dalším typem temporálních dat jsou **sekvence**. Na rozdíl od časových řad se jedná o záznam diskrétních hodnot, často nazývaných jako událostí. Pravidelnost zaznamenávání nehraje v tomto případě žádnou roli.

Metody dolování v temporálních datech jsou tedy podmnožinou metod dolování v datech, které se zaměřují na data s temporálním charakterem. Hlavním specifikem metod je nutonost vhodně si poradit s časovou doménou, která je od ostatních atributů značně odlišná. Role časové složky je stěžejní. Pokud nás v asociační analýze zajímaly položky, které mají frekventovaný společný výskyt, tak v případě temporálního dolování by nás zajímalo, jak tyto položky následovaly po sobě. V literatuře se takovéto znalosti označují jako sekvenční vzory a budou blíže popsány v sekci 3. Hledání sekvenčních vzorů náleží do podkategorie algoritmů temporálního dolování *objevování temporálních vzorů (temporal pattern discovery)*. A právě této oblasti se blíže věnuje celý tento článek.

Článek je dále členěn takto. V sekci 2 jsou disktuovány specifika a problémy, které přináší použití časové domény. Stěžejní jsou následující sekce a 3 a 4, které popisují konkrétní dva typy zdrojů temporálních dat nad kterými se dají dolovat nové vzory. Tyto sekce, zejména sekvenční dolování, rovněž popisují metody, které slouží pro dolování představených vzorů. Následující sekce 5 stručně představuje další možný zdroj temporálních dat a tím jsou datové proudy. Celý článek je pak shrnut v závěru.

2 Problémy časové domény

V úvodu byl nastíněn problém časové domény, která je od ostatních atributů poměrně odlišná. V této části budou představena specifika, která je potřeba uvažovat, když se zabýváme v databázích časovou složkou [14].

1. **Časový model** - v temporálních databázích se rozlišují dva typy časových údajů - *valid time* a *transaction time* a může být ukládán buď jeden z nich, nebo oba dva. V případě použití obou dvou časů se mluví o *bitemporálním modelu*.
   (a) *Valid time (čas platnosti)* - ten reprezentuje platnost objektu v reálném světě. Např. *"Pan Novák byl přijat do zaměstnání jako údržbář 21.3.2004"*.
   (b) *Transaction time (transakční čas)* - označuje čas, kdy se uložila daná informace do databáze. Např. předchozí informace mohla být vložena až 1.4.2004.
2. **Typy databází** - na základě toho, jaké časové údaje se ukládají, rozlišujeme 4 typy databází. (1) *Snapshot (snímkové)* ukládají pouze aktuální stav databáze, (2) *Rollback* databáze podporují ukládat u objektu pouze transakční čas, (3) *Historické* databáze naopak pouze validní čas a (4) *Temporální* databáze ukládají časy oba dva.

3. **Granularita** - popisuje trvání jednoho časového vzorku, měření. V databázích se většinou ukládají údaje s přesností na sekundy či milisekundy, ale pro potřeby dolování z dat je často zapotřebí granularita jiná než takováto. Pro konkrétní aplikační oblast se vyžaduje různá granularita času. Např. pro hledání sezónních trendů v nákupech se budeme pohybovat na úrovní měsíců, ale pro hledání denní vytíženosti obchodů nás budou zajímat hodiny či minuty.
4. **Monotónnost** - atomický čas neustále roste. Pokud tedy zaznamenáváme např. hodnotu akcií, máme zaručeno že čas nově zaznamenané hodnoty bude větší než u všech dosud zaznamenaných.
5. **Dotazovací jazyk** - na rozdíl od dotazovacího jazyka SQL pro relační databáze neexistuje všeobecně používaný jazyk nebo rozšíření SQL pro dotazování nad temporálními daty. Takový jazyk by měl umožňovat např. dotazy na různých úrovních granularity nebo získat informaci o časovém intervalu z dat, která uchovávají informace pouze pro časové okamžiky. Takové operace nejsou zmíněny jen tak, ale jsou důležité ve fázi předzpracování dat, která předchází samotnému dolování. Při neexistenci takového obecně rozšířeného dotazovacího jazyka je pak na každém, co si vybere. Většinou se vybírá takové řešení, že na úrovni uživatelského rozhraní je zvolena vlastní reprezentace dotazů a ta je pak pomocí prostřední vrstvy (mediator) převáděna na běžně známé SQL dotazy. Často se volí jako dotazovací jazyk TSQL2, pro který se opět vytváří prostředník, který převádí dotazy v TSQL2 na běžné SQL dotazy.

3 Dolování v sekvencích

Pro lepší srozumitelnost je vhodné na tomto místě definovat pojmy, které se k dolování v sekvencích používají [5]. Nechť I=I_1,I_2,...,$I_p$je množina všech **položek**. Množina položek (*itemset*) je neprázdná množina položek. **Sekvence** je pak uspořádaný seznam **událostí** označovaný jako $\langle e_1 e_2 e_3 \dots e_l \rangle$, kde událost $e_1$nastala dříve než událost e_2, událost e_2dříve než $e_3$atd. Události tvoří množiny položek (*itemsets*) a rovněž bývají někdy označovány jako **prvky** či **elementy**. Množina položek v rámci jedné události je chápána tak, že nastala ve stejný čas. Není mezi nimi časové uspořádání. Událost jako množinu položek se zapisuje jako $(x_1 x_2 ... x_q)$ a pro připomenutí každé x_k je položka. Pokud je událost tvořena pouze jednou položkou, pak se vynechává zápis závorek. Je vhodné rovněž připomenout vlastnost matematického chápání množiny v tom smyslu, že položka se může v množině položek vyskytnout nejvýše jedenkrát. **Databáze sekvencí** je množina dvojic $\langle SID, s \rangle$, kde SID je identifikace sekvence a s je sekvence. Příklad databáze sekvencí ukazuje tabulka 1. Pro shrnutí, **sekvencí** v kontextu dolování z temporálních dat chápeme jako časově uspořádaný seznam prvků nebo událostí, u nichž může, ale také nemusí být zaznamenaný čas. Příkladem zápisu sekvence, tak jak byla výše popsána, může být $\langle acab(dc)b(aebd)a \rangle$, kde množina položek $I = \{a, b, c, d, e\}$. Typickou interpretací takovéto sekvence může být záznam nákupů jednoho zákazníka v konkrétním obchodě, kde každým nákupem

se chápe událost a ta je složena z položek zboží, které zákazník během jedné návštěvy nakoupil. Databáze sekvencí by pak byla tvořena sekvencemi všech zákazníků, kteří navštěvují daný obchod s sekvencemi nákupů.

Další pojmy, které souvisí se sekvencemi jsou:

- $l-$**sekvence**, tj. sekvence délky l, přičemž délka je dána počtem instancí položek v sekvenci,
- sekvence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ je **podsekvencí** sekvence $\beta = \langle b_1 b_2 \dots b_m \rangle$, pokud existují celá čísla $1 \leq j_1 < j_2 < \dots < j_n \leq m$ tak, že platí $a_1 \subseteq b_{j_1}, a_1 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$. Pak se značí $\alpha \sqsubseteq \beta$ a β je označováno jako **nadsekvencí** sekvence α. Například sekvence $\langle ad \rangle$ je podsekvencí sekvence 1 z tabulky 1.

SID	Sekvence
1	$< a(abc)(ac)d(cf) >$
2	$< (ad)c(bc)(ae) >$
3	$< (ef)(ab)(df)cb >$
4	$< eg(af)cbc >$

Tabulka 1. Příklad databáze sekvencí

Podle typů hledaných vzorů lze dolování v sekvencích rozdělit do tří ktegorií. (1) **Sequence mining** - tj. hledání frekventovaných sekvenčních vzorů, (2) dolování **temporálních asociačních pravidel** a (3) objevování **frekventovaných epizod**. O první kategorii se v literatuře píše také jako o *inter-transaction pattern discovery*, vzory se hledají v databázi složené z více sekvencí. Ostatní dvě kategorie jsou nazývany jako *intra-transaction pattern discovery* a zdrojem pro dolování bývá většinou jediná sekvence.

3.1 Sequence mining

Pojem sequence mining označuje hledání frekventovaných sekvenčních vzorů. K vysvětlení tohoto pojmu je vhodné nejprve vysvětlit pojmy související. N-tice $\langle SID, s \rangle$v databázi sekvencí **obsahuje** sekvenci α, pokud α je podsekvencí sekvence s. **Podpora** sekvence (*support*) α v databázi sekvenci S je určena jako počet databázových n-tic, které obsahují sekvenci α, formálně zapsáno

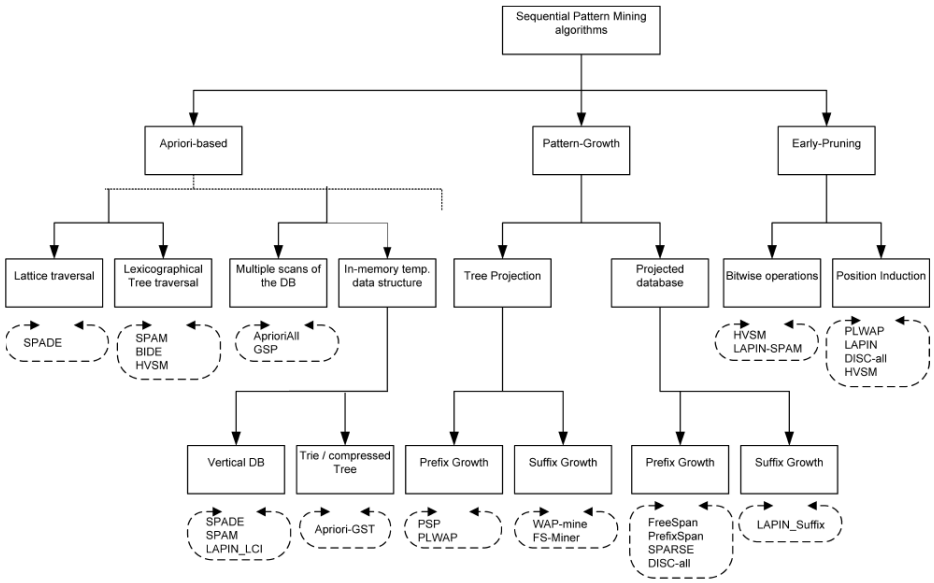
$$support_S(\alpha) = |\{\langle SID, s \rangle | (< SID, s > \in S) \wedge (\alpha \sqsubseteq s)\}|$$

Frekventované sekvenční vzory jsou pro danou databázi sekvencí takové sekvence, jejichž podpora je větší než **minimální podpora**. Tato minimální podpora přitom bývá vstupním parametrem agloritmu a zadávána dle potřeb uživatele. Pro tyto vzory se také užívá zkrácený pojem *frekventované sekvence*.

Metody, které se pro dolování sekvenčních vzorů používají lze ještě rozdělit na dvě kategorie: (1) metody, které dolují *úplnou množinu* sekvenčních vzorů

a (2) metody, které dolují pouze množinu *uzavřených sekvenčních vzor*, což je podmnožina výsledku první zmíněné kategorie. Uzavřené vzory společně s algoritmy pro jejich získávání budou popsány na konci této sekce za algoritmy pro hledání úplné množiny vzorů.

Počátkem vzniku algoritmů pro hledání sekvenčních vzorů lze označit rok 1995, kdy byl tento problém popsán Agrawalem a Srikantem na základě studia sekvencí nákupu zákazníků. Od té doby vznikla již celá řada algoritmů založených na různých principech. Taxonomii rozdělení algoritmů dle jejich vlastností dobře vystihuje obrázek 3.1. Levá větev představuje skupinu algoritmů založenou na využívání Apriori vlastnosti (*apriori-based*), prostřední skupina je založena na principu růstu vzorů (*pattern-growth*) a pravou větev tvoří algoritmy založené na brzkém ořezávání (*early-pruning*). Tyto skupiny se dělí ještě do dalších podskupin, což bude představeno v následujících podsekcích. Tato taxonomie algoritmů pochází z [12], jenž byl hlavním materiálem, ze kterého jsem v této části čerpal. Pokud nebude uvedeno explicitně u algoritmu jinak, tak byly myšlenky převzaty právě z tohoto článku.



Obrázek 1. Taxonomie algoritmů pro hledání frekventovaných sekvenčních vzorů

Metody založené na vlastnosti apriori Tato skupina algoritmů využívá apriori vlastnosti a přímo navazuje na algoritmus *Apriori*, který byl poprvé představen R. Agrawalem v [11]. Tento algoritmus slouží pro dolování frekventovaných množin a asociačních pravidel z transakčních databází. **Apriori** vlastnost

říká, že **každá neprázdná podmnožina frekventované množiny musí být rovněž frekventovaná**. Pro připomenutí frekventovaná množina znamená, že její podpora je větší než zadaná minimální podpora. Algoritmus Apriori se skládá z několika průchodů. V *prvním* průchodu jsou ve vstupní databázi transakcí nalezeny všechny frekventované množiny délky 1 (tzv. *1-množiny*), tedy frekventované položky. V *dalších* krocích se pak vygenerují nové kandidátní množiny z frekventovaných množin z předchozího kroku cyklu. Generování probíhá na základě kartézského součinu frekventovaných kandidátních sekvencí z předchozího kroku. Pro tyto kandidátní množiny se opět zjišťuje, zdali mají frekventovaný výskyt ve vstupní transakční databázi. Ty množiny, které toto nesplňují, jsou vyřazeny a dále se s nimi nepočítá. Díky výše představené apriori vlastnosti je tímto způsobem zajištěno, že bude nalezena úplná množina frekventovaných množin. Algoritmus končí ve chvíli, kdy již nejsou nalezeny další frekventované množiny splňující minimální podporu. Princip algoritmu spočívá v *generování a testování kandidátů*, a je také vidět že algoritmus prohledává prostor do šířky.

AprioriAll Právě algoritmus Apriori od stejných autorů posloužil jako základ pro první algoritmus hledající sekvenční vzory a to AprioriAll. Využívá stejné generování kandidátů pomocí kartézského součinu a dochází k ořezání takových kandidátů, jejichž podsekvence nejsou frekventované. Tato informace je známá z předchozího kroku. Podpora kandidátních sekvencí se pak počítá jako poměr n-tic v databázi sekvencí, ve kterých je kandidátní sekvence obsažena. Ostatní části algoritmu jsou stejné jako pro Apriori. Nevýhodou algritmu je exponenciální růst generovaných kandidátů s přibývajícími cykly algoritmu. Není tedy příliš vhodný pro hledání delších sekvencí. Toto není problém pouze tohoto algoritmu, ale obecně všech algoritmů založených na generování a testování kandidátů. Dále je algoritmus omezen pouze na nalezení sekvencí, jejichž prvky mají pouze jednu položku.

GSP Od stejných algoritmů pochází i algoritmus *GSP (Generalized sequential patterns)*, publikovaný v [13]. I tento algoritmus je založený na generování a testování kandidátů. Podmínka spojení dvou sekvencí z předchozího kroku pro vytvoření nové sekvence je ale odlišná. Tato operace se nazývá GSP-join a dvě sekvence s_1a s_2 s k položkami mohou vytvořit novou pokud je prvních $(k-1)$ položek ze sekvence $s_1$stejných jako posledních $(k-1)$ položek ze sekvence s_2. Sekvence *abc* tedy může vzniknout spojením sekvencí *ab* a *bc*. Při ořezávání se kontroluje, zda CONTIGOUS $(k-1)$-podsekvence nového kanditáta je frekventovaná. Po generování se v každém cyklu opět kontroluje zda jsou kandidáti frekventovaní stejně jako v AprioriAll. Algoritmus opět skončí pokud není žádný kandidát frekventovaný nebo ani jeden neprojde ořezáváním při generování. Pro urychlení se v každém průchodu algoritmu ukládají nové kandidátní sekvence do struktury hashovacího stromu, který pak umožňuje při průchodu vstupní databází sekvencí rychlejší kontrolu pro zjištění, kteří kandidáti jsou v sekvenci obsaženi.

Z názvu algoritmu Generalized sequential patterns je patrné, že se zabývá zobecněnými sekvenčními vzory. Uživatel může pomocí vstupních parametrů omezit prohledávaný prostor sekvencí dle jeho požadavků.

- parametr **max_gap**, neboli maximální délka intervalu, který se požaduje mezi dvěma prvky sekvence. Analogicky je definován i parametr min_gap, tedy minimální délka intervalu.
- parametr **window_size** určuje velikost okna - délku intervalu, v němž jsou považovány položky tak, že náleží do jednoho elementu, tedy že nás nezajímá jejich vzájemné uspořádání v čase. Tímto lze dobře měnit časovou granularitu algoritmu. Ačkoliv ve vstupní databázi může být u položek zaznamenán čas s přesností na sekundy, lze se na data podívat s přesností dnů, týdnů atd.
- S jistou úpravou formátu vstupní databáze lze do dolování začlenit taxonomii a hledat tak vzory na různých úrovních hierarchie.

V literatuře se uvádí se, že by algoritmus GSP měl bý $2 - 20\times$ rychlejší než AprioriAll. Nicméně, z povahy algoritmu založeného na generování a testování kandidátů je špatně použitelný na sekvence větší délky, o čemž jsem se i sám přesvědčil v prostředí Microsoft Analysis Services na datech se zhruba 60 000 vstupními sekvencemi.

PSP Tento algoritmus je téměř totožný jako GSP. Oproti němu využívá pro ukládání kandidátních sekvencí strukturu *prefixový strom*. V [14] se uvádí, že je oproti GSP rychlejší, ačkoliv i on musí provést několik průchodů vstupní databází a také generovat a testovat kandidátní sekvence. Stejně jako předchozí je ale nevhodné jej používat v případě nízkého minimální podpory a je tak nepoužitelný např. pro dolování z webových logů. Obdobným typem algoritmu jako PSP je **Apriori-GST**, který místo prefixového stromu využívá *zobecněný příponový (suffixový) strom.*

SPADE Předchozí algoritmy dolovaly nad databází sekvencí v tzv. horizontální podobě, tj, takové, co byla představena v úvodu této sekce. Oproti tomu SPADE využívá *vertikálního formátu databáze* [10]. Databáze se pak skládá z n-tic v podobě $\langle mnozina\_polozek : (ID\_sekvence, ID\_udalosti)\rangle$. Pro jednu množinu položek přitom může existovat více dvojic, které identifikují konkrétní výskyt události v sekvenci. $ID\_udalosti$ si lze také chápat jako časové razítko *(timestamp)*. Například pro sekvenci 1 a 2 z tabulky 1 by mohla podmnožina databáze vypadat takto: $\{\langle a : (1,1)\rangle, \langle abc : (1,2)\rangle, \langle ac : (1,3)\rangle, \langle d : (1,4\rangle, \langle cf : (1,5)\rangle, \langle ad : (2,1)\rangle, \langle c : (2,2)\rangle, \langle bc : (2,3)\rangle, \langle ae : (2,4)\rangle\}$.

SPADE využívá *teorie mřížek (lattice theory)* pro generování kandidátních sekvencí převzatou z inkrementálního algoritmu *ISM*. Právě struktura mřížky je postupně budována v průběhu algoritmu a pak využívána pro počítání podpory a tvorbu dalších kandidátních sekvencí. Algoritmus lze rozdělit do tří fází.

1. Nejprve se vstupní databáze sekvencí transformuje do vertikálního formátu a naleznou se frekventované položky, tj. sekvence délky 1.

2. Z těchto frekventovaných položek jsou pomocí operace spojení vygenerovány sekvence délky 2.
3. Prochází se mřížková struktura, jsou generováni kandidáti větších délek a vyřazují se ti, kteří nejsou frekventovaní. Během průchodu a zejména při počítání podpory se využívá vytváření struktur *ID-list.*

Id-list je struktura, která se vytváří v průběhu algoritmu pro každou položku a kandidátní sekvenci a jedná se o seznam $(ID\_sekvence, ID\_udalosti)$, ke kterým náleží kandidátní sekvence. Nad těmito strukturami pak probíhá operace spojení pro tvorbu nových kandidátních sekvencí a při tom rovněž i počítání jejich podpory. Dva kandidáti mohou být spojeni pro vytvoření nového, pokud platí tři podmínky, tj. (1) pokud byli oba frekventovaní, (2) pokud mají stejný identifikátor sekvence a (3) pokud jejich identifikátory událostí zachovávají správné uspořádání (vzestupné).

Z důvodu možného nárůstu struktury mřížky v paměti představili tvůrci ekvivalenční třídy ve struktuře založené na společném prefixu sekvencí. Každá část struktury pak může být načtena a prohledávána v paměti odděleně. Algoritmus je považován za rychlejší než GSP či dokonce v další podčásti představený FreeSpan. I přes využití mřížkové struktury trpí algoritmus generováním a testováním kandidátů.

SPAM Algoritmus integruje myšlenky z několika algoritmů, konrkétně z GSP, SPADE a FreeSpan [3]. Pro prohledávání vstupního prostoru využívá strukturu *lexikografického stromu,* a jako první algoritmus tento prostor prohledává do hloubky *(FreeSpan).* Z GSP přebírá generování a testování kandidátů. Během průchodu stromu jsou z každého jeho uzlu generovány dva typy potomků: (1) *sekvenčně-rozšířené* sekvence *(S-krok)* a (2) *množinově-rozšířené* sekvence *(I-krok).* Stejně jako v případě algoritmu SPADE je využit vertikální formát databáze. Narozdíl podoby id-listů ve SPADE je ale zvolena efektivní bitmapová reprezentace. Bitmapová reprezentace umožňuje využít rychlé bitové operace AND nad bitmapovými reprezentacemi pro generování kandidátů a také výpočet podpory. Na tomto místě se pro SPADE používala operace spojení. Při vzájemném porovnání algoritmů byl shledán SPAM 2,5x rychlejší, ale také $5-20\times$ náročnější na spotřebu paměti. Na velkých datových sadách byl dokonce rychlejší než velmi rychlý algoritmus PrefixSpan, který bude představen v další podsekci. Na velmi malých datových sadách byl PrefixSpan sice rychlejší, ale z pohledu dolování z dat je too zandebatelná informace. Z důvodu vylepšení paměťových nároků algoritmu byl později představena vylepšená verze *I-SPAM,* v němž se povedlo maximální nároky snížit na polovinu bez ztráty výkonnosti.

Metody založené na růstu vzorů Metody založené na růstu vzorů řeší problém předchozích algoritmů a tím bylo generování a testování kandidátních sekvencí, což vadí hlavně při hledání dlouhý sekvencí v rozsáhlých databázích. Myšlenka je založena na principech *růstu FP-stromu*, což je algoritmus pro hledání asociačních pravidel v transakčních databázích. (1) Nejprve jsou opět

nalezeny frekventované položky a následně je tato informace zkomprimována do struktury *FP-stromu (Frequent Pattern, frekventovaný vzor).* Z tohoto stromu je následně generována množina projektovaných databází a nad každou pak probíhá dolování odděleně. K objekvování frekventovaných vzorů jsou vytvářeny prefixové vzory a ty konkatenovány s příponovými vzory. Tím se zamezuje generování kandidátů.

Obdobných principů se využívá i při dolování sekvenčních vzorů. Často bývá ze začátku vytvořena reprezentace vstupní databáze a způsob jak rozdělit prohledávaný prostor a generovat co nejméně kandidátů. Často se například využívá **projektovaných databází**. Podsekvence α' sekvence α se nazývá *projekce z αvzhledem k prefixu* β tehdy a jen tehdy, když (1) α' má prefix β a (2) neexistuje žádná další nadsekvence α'' sekvence α' tak, že α'' je podsekvence od αa má také prefix β. Sekvence $\beta = \langle e'_1 e'_2 \dots e'_m \rangle$se nazývá **prefixem** sekvence $\alpha = \langle e_1 e_2 ... e_n \rangle (m \leq n)$ pokud platí následující tři podmínky: (1) $e'_i = e_i$ pro $(i \leq m-1)$, (2) $e'_m \subseteq e_m$ a (3) všechny frekventované položky v $(e_m - e'_m)$ abecedně následují za těmi v (e_m). Sekvence $\gamma = \langle e''_m e''_{m+1} \dots e_n \rangle$je nazývána **suffixem (příponou)** sekvence α vzhledem k prefixu β, značí se také jako $\gamma = \alpha / \beta$.

FreeSpan Prvním představitelem této skupiny algoritmů je Frequent Pattern-Projected Sequential Pattern Mining, zkráceně FreeSpan. (1) V prvním průchodu si vytváří z frekventovaných položek seznam *f-list* a z nich sestaví strukturu *S-Matice* pro uložení podpory všech možných kandidátů délky 2. V druhém průchodu databáze se tato S-Matice naplní daty o podporách kandidátů. Ve třetím kroku jsou sestaveny pouze frekventované sekvence délky 2, které jsou navíc vhodně anotovány, aby z nich bylo možné sestavit projektované databáze a dále prohledávat pouze je, čímž se zamezí dalšímu průchodu vstupní databáze. Algoritmus testuje menší počet sekvencí než GSP díky prořezávání kandidátů v S-Matici ještě před tím, nežli jsou generováni. Problémem je výpočet a tvorba projektovaných databází. Protože podsekvence může růst na kterémkoliv místě sekvenci, je nutná kotrola všech možných kombinací růstu, což je nákladné.

PrefixSpan Tento algoritmus, představený v [7], je považován za jeden z nejznámnějších algoritmů pro hledání frekventovaných sekvenčních vzorů a společně s algoritmem SPADE bývá uváděn jako měřítko *(benchmark)* při testování výkonu jiných algoritmů. Pochází od stejných autorů jako FreeSpan a řeší problém zmíněný na závěr popisu tohoto algoritmu. PrefixSpan zkoumá místo všech kombinací možných podsekvencí pouze prefixové podeskvence a vytváří projektové databáze pouze z jejich příponových podsekvencí. Algoritmus lze popsat pomocí následujících kroků. Pro názornější popis budou u kroků uvedeny příklady postavené na vstupní databází sekvencí z tabulky 1 a zvolené miniální podpoře $minsup$=2.

1. Nalezenou se všechny frekventované sekvence délky 1. V příkladu se jedná o šest sekvencí $\langle a \rangle, \langle b \rangle, \langle c \rangle, \langle d \rangle, \langle e \rangle, \langle f \rangle$.

2. Prohledávaný prostor sekvenčních vzorů lze rozdělit na základě prefixu do n podmnožin, kde n značí počet nalezených frekventovaných 1-sekvencí. Z příkladu v předchozím kroku by to bylo šest skupin (1) sekvencí s prefixem $\langle a \rangle$, (2) s prefixem $\langle b \rangle$ ... a (6) s prefixem $\langle f \rangle$.
3. Prohledávají se podmnožiny vstupního prostoru pomocí sestavení *projektovaných databází.* Nad každou z databází se pak doluje rekurzivně. V případě skupiny sekvenčních vzorů začínající prefixem $\langle a \rangle$ by do projektované databáze byly zařazeny pouze takové sekvence ze vstupní databáze, které obsahují položku a. Z každé sekvence se do projektované DB musí zařadit pouze podsekvence předcházeny prvním výskytem položky a. Např. pro 1. sekvenci $\langle (ef)(ab)(df)cb \rangle$ by to bylo $\langle (\_b)(df)cb \rangle$. Zápis $(\_b)$ značí, že poslední událost v prefixu, který obsahoval položku a obsahoval také položku b. Pro položku a by *a-projektovaná databáze* obsahovala čtyři příponové podsekvence, kromě zmíněné ještě $\langle (abc)(ac)d(cf) \rangle, \langle (\_d)c(bc)(ae) \rangle, \langle (\_f)cbc \rangle$. Prohledáním této databáze by se zjistily frekventované položky, sestavily by se příslušné vzory délky 2 a na základě nich jako dalších prefixů by se analogicky prostor rozdělil na podmnožiny. Takto by se rekurzivně pokračovalo dále, dokud by bylo možné generovat další projektované databáze.

Jak je vidět, algoritmus vůbec negeneruje žádné kandidátní sekvence, na druhou stranu může generovat hodně projektovaných databází, které se dále mohou dělit. To může být hlavním výkonnostním problémem algoritmu. Každopádně je na tom mnohem lépe než GSP i FreeSpan.

Existuje několik dalších optimalizačních technik. Jedna z nich je *dvouúrovňová projekce (bilevel projection).* Místo projektovaných DB se používá S-Matice představena ve FreeSpanu. V S-Matici jsou uloženy kromě podpory také všechny 1-sekvence. S-matice se používá pro určení, které $3-$*sekvence* mohou být generovány a které ne. Druhou možností optimalizace je je použití *pseudo-projekce* a to v případě, že algoritmus má data pouze v paměti a nepřistupuje na disk. Pak je každá projekce reprezentována ukazatelem do sekvence v databázi a offsetem v této sekvenci. Na základě experimentů bylo v [7] ukázáno, že při disk-based zpracování je lepší použít *bilevel projekci*, při dolování pouze v paměti je naopak výhodnější využít *pseudo-projekce.*

K algoritmu PrefixSpan bylo publikováno později několik dalších technik k rozšíření. Jedním z nich je např. *MILE.* V něm se využívá znalostí o existujících vzorech, aby se vyhnulo redundantním průchodům v databázi. Algoritmus je tím rychlejší oproti algoritmu PrefixSpan, čím je větší počet nalezených vzorů. Uchování informací o existujících vzorech činí naopak algoritmus náročnější na paměť.

WAP-Mine Tento algoritmus, publikovaný v [8], vznikl ve stejné době jako FreeSpan a PrefixSpan a představuje techniku založenou na dolování ve stromové struktuře, tzv. *WAP-stromu.* Tento strom je velice podobný dříve představenému FP-stromu. Algoritmus je optimalizován pro dolování z webových logů a umožňuje tak na vstupu pouze sekvence s událostmi s jedinou položkou. Příklad takové databáze lze vidět v tabulce.

ID uživatele	Sekvence
100	$\langle bcbae \rangle$
200	$\langle bacbae \rangle$
300	$\langle abe \rangle$
400	$\langle abebd \rangle$
500	$\langle abad \rangle$

Tabulka 2. Ukázková databáze sekvencí z webového logu. Převzato a upraveno z [12].

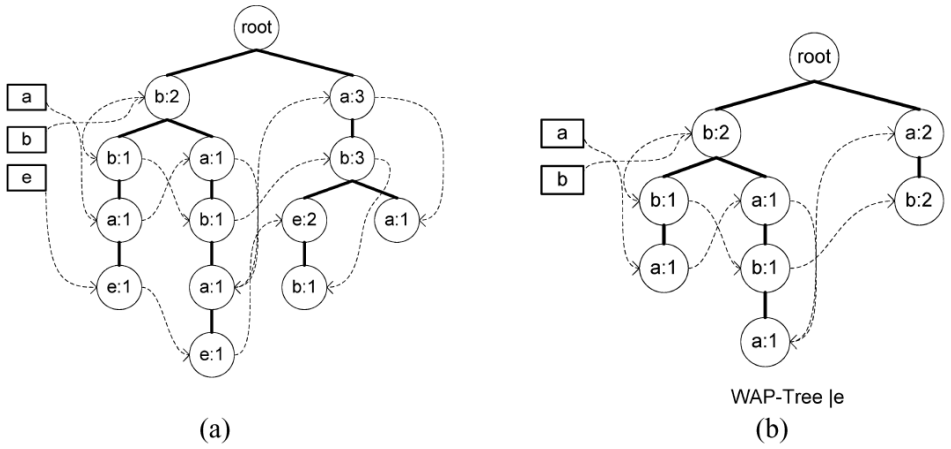
WAP-Mine vyžaduje dva průchody vstupní databází. (1) V prvním průchodu se opět najdou frekventované 1-sekvence a (2) ve druhém průchodu je vybudován WAP-strom pouze z *frekventovaných podsekvencí*. To jsou takové sekvence, které vzniknou z původních sekvencí vstupní databáze při ořezání položek nenalezených v prvním kroku. Následně už se doluje pouze nad WAP-stromem. Společně s WAP-stormem je uchovávna *hlavičková tabulka (header table)*, jenž obsahuje pro všechny položky ve stromu počet jejich výskytů a odkaz na první uzel ve stromu, který tuto položku obsahuje. V odkazovaných uzlech je pak ukazatel na další uzel, ukazatele tvoří jednosměrně vázaný seznam, což ukazuje i obrázek. Při dolování se postupně pro položky s nejnižší podporou tvoří *podmíněný základ* a přitom se vytváří dočasné WAP-stromy odvozené od tohoto základu. Takto se postupuje rekurzivně dále se zvětšujícím se podmíněným základem až jsou nalezeny všechny sekvenční vzory. Příklad WAP-stromu pro databází sekvencí z tabulky 3.1 i dočasného WAP-stromu vzniklého z podmíněného základu e je ukázán na obrázku 3.1.

Algoritmus WAP-mine je lépe škálovatelný nežli GSP, vyhýbá se generování a testování kandidátů a těží z toho, že vstupní databázi prohledává pouze dvakrát. Nicméně obdobně jako u algoritmu PrefixSpan trpí rekurzivním generováním dočasných WAP-stromů a tím i velkou spotřebou paměťového prostoru.

Tento problém byl podnětem pro vznik algoritmu **PLWAP**, který jej vyřešil pomocí toho, že u stromových uzlů ukládá binárně zakódovánou pozici a uzly propojuje na základě průchodu pre-order. Toto umožňuje algoritmu dolování vzorů v původním WAP-stromu a není nucen vytvářet velké množství dočasných WAP-stromů.

FS-Miner Tento algoritmus je podobný WAP-mineru a je zajímavý tím, že podporuje inkrementální dolování. Používá komprimovanou reprezentaci databáze v podobě FS-stromu. FS-strom je velice podobný WAP-stromu a PLWAP-stromu a je považován za variantu struktury *trie*. Oproti nim uchovává informace o podporách pro hrany mezi uzly stromu a také udržuje i částí sekvencí, které nejsou frekventované. To je sice z pohledu dolování zbytečná redundance, ale slouží to k podpoře inkrementálního učení. Tyto hrany reprezentují ve stromu 2-sekvence. Příklad FS-stromu nad databází sekvencí z tabulky 3.1 ilustruje obrázek 3.

Na rozdíl od všech dosud popsaných algoritmů uchovává dva typy podpory a má tedy i dva parametry minimální podpory. Prvním (1) je $MSuppC^{seq}$, minimální podpora pro sekvenci, čili totožná s tím, jak byla podpora chápána



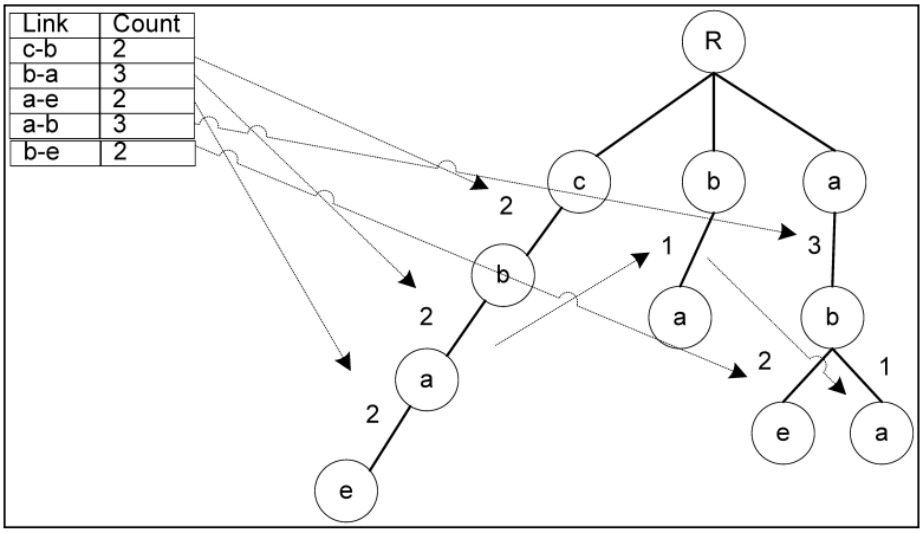
Obrázek 2. (a) Ukázka WAP-stromu a (b) WAP-stromu vzniklého z podmíněného základu e. Převzato z [12].

dosud. Druhá (2) je $MSuppC^{link}$- minimální popodpora pro *spoje (hrany)*. Hrana h s podporou $Supp^{link}(h)$ se považuje za *frekventovanou hranu*, pokud $Supp^{link}(h) \geq MSuppC^{seq}$ a *potenciálně frekventovanou hranu,* pokud platí podmínka $MSuppC^{link} \leq Supp^{link}(h) \leq MSuppC^{seq}$. Jestliže je podpora hrany menší než obě minimální podpory, pak se jedná o *nefrekventovanou hranu.* Právě uchovávání potenciálně frekventovaných hran slouží v algoritmu pro možnost inkrementálního dolování.

Průběh algoritmů je podobný jako WAP-mine, ačkoliv FS-miner využívá reprezentace v podobě hran. V tuto chvíli už se předpokládá vytvořený FS-strom. Z uložených frekventovaných hran (1) se nejprve odvodí *cesty,* které slouží pro (2) tvorbu podmíněných sekvenčních bází a ty zase pro tvorbu (3) podmíněného FS-stromu. Z těchto pomocných stromů se pak extrahují výsledné *frekventované sekvence.*

Výhodou algoritmu je zmíněná podpora inkrementálního dolování a také schopnost dobře si poradit s rozsáhlými daty s různými *(distinct)* položkami. Porovnávání s algoritmy WAP-mine nebo PLWAP bohužel nebylo provedeno, autoři se pouze zmiňují o porovnání s variantou Apriori pro sekvenční vzory a o očekávaných lepších výsledcích jimi vytvořeného algoritmu. Nevýhodou FS-mineru je zejména množství informací, které se uchovává ve stromu.

Metody založené na brzkém ořezání Zatím poslední novou skupinou algoritmů pro hledání sekvenčních vzorů jsou metody založené na *brzkém ořezání kandidátních sekvencí.* Ještě více než ostatní metody se snaží co nejdříve v procesu dolování ořezat prohledávaný prostor a snížit délku výpočtu podpory. Algoritmy staví na myšlence *indukce pozice*, která má následující znění: *Pokud je poslední pozice položky menší než aktuální pozice prefixu, položka se už nemůže*



Obrázek 3. FS-strom pro vstupní datbáze, převzato z [12].

objevit za aktuálním prefixem ve vstupní databázové sekvenci [17]. Pro uchování posledních pozic algoritmy většinou využívají tabulky. Další fáze algoritmů bývají obdobné jako u růstu vzorů.

LAPIN Zřejmě nezjnámějším představitelem je algoritmus LAPIN z [18]. Pro uchování posledních pozic položek využívá seznam *posledních pozic položek (item-last-position)*. Ten je vytvořen už při prvním průchodu databáze při hledání frekventovaných 1-sekvencí.

Pro frekventované 1-sekvence se na základě seznamu zjistí *hranice prefixu*. Ta udává pro každou sekvenci v databázi nejnižší možný index elementu, od kterého lze uvažovat další frekventovanou položku. Následně se na základě této informace hledají další výskyty položek v sekvencích a inkrementují se jejich podpory. Z nich vzniknou frekventované sekvence délky 2, které slouží jako nový *podmíněný prefix*, aktualizuje se seznam posledních pozic položek a pokračuje se dále dokud nejsou nalezeny všechny frekventované sekvence.

Uvádí se, že díky brzkému ořezání prohledávací prostor snížen oproti algoritmu PrefixSpan téměř na polovinu. LAPIN byl rychlejší než PrefixSpan při minimální podpoře větší než 1% a pro větší datové sady.

Obdobným algoritmem jako LAPIN je **LAPIN-SPAM** a vznikla jako optimalizační technika pro algoritmus SPAM. Namísto náročného počítání bitových AND operací si algoritmus vytvoří v prvním průchodu databáze pomocnou tabulku *Item_is_exist_table*, která plní obdobnou funkci jako v seznam posledních pozic položek v předešlém algoritmu LAPIN. Oproti algoritmu LAPIN se liší v tom, že využívá bitmapovou reprezentaci vstupní databáze stejně jako SPAM.

Zrychlení algoritmu oproti metodě SPAM je vykoupeno zvýšenými paměťovými nároky.

Dalším zajímavým kandidátem z této kategorie je algoritmus **HVSM**, který oproti LAPIN zdůrazňuje důležitost použití bitmapové reprezentace vstupní databáze sekvencí a generuje nové kandidáty tak, že považuje své sourozenecké uzly za své synovské uzly. Tímto se ožezané množiny sekvencí neobjeví v dalších vrstvách. Na rozdíl od všech předešlých algoritmů navíc délku sekvence počítá jako počet prvků (událostí, elementů) a nikoliv jako počet výskytů položek. Algoritmu **Disc-All** zase vedle běžně používaného lexikografického uspořádání položek a množin položek využívá i uspořádání temporálního. Další jeho vlastností je použití modifikace projektovaných databází pro rozdělení vstupního prostoru, která závistí na pořadí umístění frekventovaných 1-sekvencí.

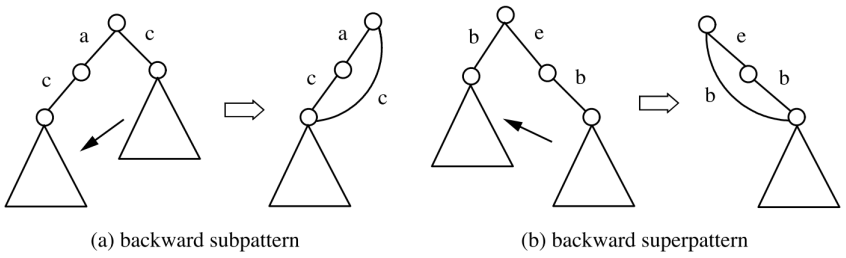
Hybridní algoritmy Některé algoritmy využívají vlastnosti, které nenáleží právě do jedné ze zmíněných tří kategorií, ale spíše je kombinují. V [12] se pak řadí mezi hybridní algoritmy, ačkoliv v jiných publikacích jsou zařazeny do některé z předešlých kategorií. Příkladem je algoritmus *SPADE,* popsaný v tomto článku v kategorii generování a testování kandidátů. Dalším představitelem je *PLWAP*, představený jako vylepšená varianta algoritmu *WAP-mine.* Tento algoritmus by šlo zařadit na pomezí *růstu vzorů* a *brzkého ořezání.*

Uzavřené sekvenční vzory Dolování sekvenčních vzorů může způsobit ve výsledku velké množství výsledných vzorů. To může být pro uživatele velmi nepřehledné. Jednou z možností je využít bezztrátové komprese v podobě dolování pouze podmnožiny úplné sady vzorů a to tzv. *uzavřených sekvenčních vzorů.* Dle definice z [5], sekvenční vzor s se nazývá *uzavřený*, pokud neexistuje žádná pravá nadsekvence $s' \sqsupset s$ tak, že $sup(s) = sup(s')$. Jinak řečeno, k dané sekvenci neexistuje supersevence se stejnou podporou. Tato kompresní metoda je bezztrátová, to znamená, že v případě potřeby lze z množiny nalezených uzavřených vzorů vygenerovat úplnou množinu všech sekvenčních vzorů.

Naivním způsobem, jak vydolovat pouze uzavřené sekvenční vzory, je použít některý z existujících algoritmů pro hledání úplné množiny vzorů a následným průchodem zjistit a vyřadit takové, které nejsou uzavřené [1]. Takový přístup ale na první pohled není vhodný, například i proto, že testování, zda je vzor uzavřený není triviální. Proto vznikly nové algoritmy, z nichž dva nejznámější *CloSpan* a *BIDE* budou představeny blíže. Dalšími představiteli jsou algoritmy *MAFIA*, *DepthProject*, *CLOSET* a *CHARM,* stručně popsané v [14].

CloSpan Algoritmus CloSpan, publikovaný v [16], je odvozen od algoritmu PrefixSpan a je založen na vlastnosti ekvivalence projektovaných databází: *Dvě projektované databzáze $S|_\alpha = S|_\beta, \alpha \sqsubseteq \beta$ (tj. α je podsekvencí β), jsou ekvivalentní tehdy a jen tehdy, když celkový počet položek v $S|_\alpha$ je totožný s celkovým počtem položek v $S|_\beta$.* Na základě této vlastnosti dokáže algoritmus v průběhu vyřadit

neuzavřené sekvence. Pokaždé, když najde dvě totožné projektované databáze založené na prefixu, lze zakázat, aby se v jedné z nich pokračovalo dále. Buď se nepokračuje v aktuálně nalezené nebo je ořezána některá z dříve nalezených větví. Tyto dva způsoby ořezání jsou ilustrovány na obrázku 4. Na konci algoritmu je ještě vyžadován krok pro smazání neuzavřených sekvencí, které mohly vzniknout v odvozených větvích. Experimenty ukázaly, že CloSpan je schopen vydolovat mnohem menší množinu výsledků nežli PrefixSpan a to v kratším čase.



Obrázek 4. Dva způsoby ořezávání v algoritmu CloSpan, převzato z [5].

BIDE Algoritmus BIDE [9] je zkratkou pro *BI-Directional Extension*, tj. oboustranné rozšíření a ve své základní verzi hledá pouze sekvenční vzory s prvky, které jsou tvořené jedinou položkou. Na základě oboustaného rozšíření kontroluje v průběhu, zda jsou vzory uzavřené. Pokud není vzor $p = e_1 e_2 \dots e_n$ uzavřený, tak musí platit nejméně jedna z následujících podmínek.

1. Existuje sekvenční vzor $s' = e_1 e_2 \dots e_n e'$ tak, že $sup(s) = sup(s')$, tedy v každé sekvenci která obsahuje sekvenci s se někde za ním vyskytuje i událost e'. Sekvence s'se nazývá *dopředně rozšířená sekvence (forward-extension sequence)* a e'se nazývá *událost dopředného rozšíření (forward-extension event).*
2. Existuje sekvenční vzor $s_1 = e_1 e_2 \dots e_i e' e_{i+1} \dots e_n$ nebo vzor $s_1 = e' e_1 e_2 \dots e_n$ tak, že $sup(s) = sup(s_1)$. S_1 se nazývá *zpětně rozšířená sekvence (backward-extension sequence)* a e' *událost zpětného rozšíření (backward-extension event).*

Z toho plyne, že vzor je uzavřený pokud neobsahuje jak *backward-extension*, tak ani *forward-extension* události. Klíčový je tedy způsob, jak hledat tyto události. (1) Při kontrole *forward-extension* událostí se pro vzor udělá p-projektovaná databáze, tj. databáze sekvencí, které obsahují společný prefix p. Pokud se ve všech sekvencích v p-projektované databázi nachází událost e', tak vzor p není uzavřený. (2) Při hledání *backward-extension* událostí je potřeba počítat s tím, že se událost může vyskytovat v sekvenci vícekrát. Pro to, abychom zjistitli jestli se mezi událostmi e_i a e_{i+1} *nachází backward-extension* událost, je potřeba prohledat prostor mezi prvním výskytem podsekvence $e_1 \dots e_i$ a posledním výskytem

podsekvence $e_{i+1}e_n$. Pokud se v tomto prostoru nachází událost e' v každé sekvenci obsahující vzor p, pak není uzavřený.

Stačí tedy zjistit při generování vzoru p založeném na hloubce p, zda je uzavřený. Zde se využívá projektovaných databází dostupných v algoritmu PrefixSpan a jejich rozšíření tak, aby databáze obsahovaly celé sekvence, nikoliv pouze přípony sekvencí po předponě p.

Backward-extension události jsou využity i pro ořezání prpohledávaného podprostoru ve chvíli, kdy není nalezen žádný uzavřený vzor. V případě, že vzor $p = e_1 \dots e_n$ není uzavřený a existuje $q = e_1 \dots e_i e' e_{i+1} \dots e_n$, lze prostor ořezat o kandidáty, kteří jsou tvořeny prefixem p a podsekvencemi obsahujícími sekvenci q ve svých projektovaných databázích. Tato optimalizace se nazývá *BackScan.*

Algoritmus pak probíhá v následujících krocích. Nejprve jsou nalezeny frekventované 1-sekvence a je vytvořena pseudo projekční databáze obdobně jako v PrefixSpanu. Pak je procházena každá položka, tedy prefix, a testuje se zda jej lze ořezat pomocí optimalizace *BackScan.* Pokud to není možné, volá se rekurzivně procedura nad danou projektovanou databází. Tato procedura zjišťuje, zda nejsou přítomny forward a backward extension události, zjišťuje lokálně frekventované položky v projektované databázi a volá se rekurzivně nad nově vzniklými proječními databázemi z lokálně frekventovaných položek.

Výkonnost algoritmu byla porovnána s algoritmem CloSpan a na testovaných datových sadách byl BIDE ve většině případů rychlejší, v některých dokonce až o řád. Co se týče paměťové náročnosti, tak potřeboval k výpočtu ve všech případech nejméně o jeden řád méně paměti. Mnohem menší pameťová náročnost tkví v tom, že na rozdíl od CloSpan si BIDE nemusí uchovávat historii nalezených uzavřených vzorů k ověření, zda nově nalezený vzor je uzavřený nebo zda může zneplatnit uzavřenost dosud některých dosud nalezených uzavřených vzorů. Co se týče porovnání s algoritmy hledající úplnou množinu sekvenčních vzorů, tak už CloSpan byl mnohem rychlejší než PrefixSpan, BIDE tento rozdíl ještě zvyšuje.

Další typy algoritmů pro sekvenční vzory Kromě výše uvedených typů existují ještě další typy algoritmů pro hledání sekvenčních vzorů. **Inkrementální** dolovací algoritmy využívají k dolování jistou část modelu, kterou si vybudovaly v předchozích spuštěních. Tento typ algoritmů je vhodné použít, pokud chceme, aby algoritmus nemusel proběhnout napoprvé celý nebo nám přibývají v databázi data a nechceme, aby výpočet bylo nutné provádět celý od znova. Příkladem těchto algoirmtů je *ISM* a také výše představený *FS-Miner.* Metody dolování **založené na omezeních** na základě uživatelsky definovaných omezeních dokáží ještě více ořezat prohledávaný prostor sekvnečních vzorů. Navíc jsou schopné nalézat jen takové vzory, které uživatele zajímají. Příkladem takových omezení může být maximální doba trvání sekvence, minimální délka sekvence, minimální doba mezi dvěma po sobě následujícími událostmi a další. Takováto omezení je možné začlenit do implemntace několika algoritmů, několik z těchto omezení je např. podporováno v algoritmu GSP.

Zhodnocení hledání frekventovaných sekvenčních vzorů Co se týče oblíbenosti, tak mezi nejpoužívanější algoritmy lze z představených zařadit GSP, SPADE, PrefixSpan a zástupce ze skupiny algoritmů pro hledání uzavřených vzorů BIDE. V [12] bylo rovněž prezentováno srovnání výkonnosti několika vybraných algoritmů nad dvěmi datovými sadami - jedna středně velká s počtem 200 000 sekvencí a druhá velká s počtem 800 000 sekvencí. Dále byly testovány algoritmy pro dvě různé hodnoty minimální podpory, 0.1% a 1%. Výsledky z těchto experimentů ukazuje tabulka 3. Lze na ní vidět, že pro středně velké datové sady se střední hodnotou nízké podpory si nejlépe vedly PrefixSpan a LAPIN, přičemž pro nízkou podporu LAPIN ani nedoběhl. Lze si také všimnout, že průměrně nejlepší výsledky dával hybridní algoritmus PLWAP. Algoritmy založené na Apriori vlastnosti dopadly v testu nejhůře a často ani nedoběhly do konce.

Algoritmus	Velikost datové sady	Minimální podpora	Čas běhu algoritmu [s]	Spotřebovaná paměť [MB]
GSP (Apriori-based)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	>3600	800
		Střední (1%)	2126	687
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	-	-
		Střední (1%)	-	-
SPAM (Apriori-based)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	-	-
		Střední (1%)	136	574
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	-	-
		Střední (1%)	674	1052
PrefixSpan (růst vzoru)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	31	13
		Střední (1%)	5	10
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	1958	525
		Střední (1%)	798	320
WAP-mine (růst vzoru)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	-	-
		Střední (1%)	27	0.556
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	-	-
		Střední (1%)	50	5
LAPIN_Suffix (Brzké ořezání)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	>3600	-
		Střední (1%)	7	8
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	-	-
		Střední (1%)	201	300
PLWAP (Hybridní)	Střední ($\|D\| = 200000$)	Nízká (0.1%)	23	5
		Střední (1%)	10	0.556
	Velká ($\|D\| = 800000$)	Nízká (0.1%)	32	9
		Střední (1%)	21	2

Tabulka 3. Srovnávací analýza výkonnosti vybraných algoritmů pro dolování sekvenčních vzorů. Znak "-" značí, že algoritmus se zadanými parametry skončil s chybou. Data získána z [12].

3.2 Hledání frekventovaných epizod

Předchozí problém předpokládal na vstupu databázi mnoha sekvencí. Oproti tomu při hledání frekventovaných epizod se na vstupu objevuje jediná sekvence událostí. Sekvence n událostí je složena z párů událostí a korespondujícími časovými razítky, tj.$(e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n)$. Epizoda je definována jako sekvence událostí vyskytujících se ve specifickém pořadí v rámci specifického časového okna. Epizoda *náleží* do sekvence, pokud se události v epizodě vyskytují ve stejném pořadí jako v sekvenci. Epizoda může být buď *sériová,* kde záleží na pořadí událostí, nebo *paralelní*, kde na pořadí nezáleží. Příkladem sériové epizody je např. $(A \rightarrow B \rightarrow C)$, příkladem paralelní (ABC). Třetím případem typu epizody je *kompozitní,* která je tvořena sériovou kombinací paralelních epizod [14].

Při hledání frekventovaných epizod uživatel zadává *šířku časového okna*, čímž určuje, jak moc mohou být od sebe v čase vzdáleny první a poslední událost epizody. Druhým parametrem je opět *minimální podpora* pro rozlišení frekventovaných epizod. Při hledání frekventovaných epizod se rovněž uplatňuje Apriori vlastnost. Využívá toho např. algoritmus *WINEPI* [2], který nejprve nalezne frekventované epizody s jednou událostí, z nich se hledají frekventované epizody s dvěma událostmi atd. Tento algoritmus má nevýhodu v tom, že při počítání podpory epizod může některé počítat zbytečně vícekrát. Tento problém řeší algoritmus *MINEPI [2]* od stejných autorů zavedením *minimálního výskytu (minimal occurence).* Minimální výskyt epizody αje definovaný jako interval, jehož žádné podokno neobsahuje epizodu α. Např. pro sekvenci $S = (A, 1), (A, 2), (B, 3), (B, 4)$ má podporu minimálního výskytu 2 (intervaly $[1, 1]$ a $[2, 2]$). U předchozího algoritmu by byla vypočítána podpora $sup = 4$. Na základě těchto algoritmů vzniklo i několik dalších vylepšení, např. *MINEPI+*, těm se ale už článek nevěnuje.

3.3 Dolování temporálních asociačních pravidel

Temporální asociační pravidlo lze definovat jako dvojice (R, T), kde R je asociační pravidlo a T temporální rys, např. perioda nebo kalendář [14]. Existují tři míry pro hodnocení běžných asociačních pravidel a ty lze uplatnit i při hledání v temporálních datech: (1) *podpora* - pravděpodobnost, že bude pravidlo nalezeno v databázi, (2) *spolehlivost* - podmíněná pravděpodobnost výskytu jednoho prvku ze znalosti výskytu druhého a (3) *informativnost* - míra, která nám udává užitečnost pravidla a často se používá tzv. *J-hodnota.*

Existuje několik metod dolování temporálních asociačních pravidel. Byly představeny algoritmy založené na použití vlastnosti Apriori, dále na použití *genetického programování* a specializovaného hardware anebo na hledání *fuzzy temporálních asociačních pravidel.* Více se lze dozvědět v [14].

4 Dolování v časových řadách

Pro připomenutí, časová řada je tvořena seznamem spojitých hodnot, které jsou zaznamenávány v pravidelných časových intervalech. Typickým typem dolovací

úlohy nad časovými řadami je hledání periodických vzorů, které budou představeny blíže. Ostatním zástupcům bude věnována pozornost menší. Řadí se k nim *hledání motivů (motif discovery)* a *hledání anomálií (anomaly discovery).*

4.1 Periodické vzory

Periodické vzory jsou vzory, které se pravidelně vyskytují v určitých časových intervalech v temporálních datech, nejčastěji v časových řadách. Jednou z možností, jak lze kategorizovat periodické vzory, je rozdělení na *plně periodické (full periodic)* a *částečně periodické (partially periodic) [5].*

- **Plně periodické vzory** - chápe se, že každý bod, každá zaznamenaná hodnota, v čase přispívá (ať už přesně nebo přibližně) do cyklického chování časové řady. Například všechny dny v roce přibližně přispívají do ročního období.
- **Částečně periodické vzory** - zde už není nutno, aby každý bod v čase přispíval do cyklického chování v čase. Příkladem vzoru je pravidelné ranní dojíždění do práce jistého zaměstnance v čase od 7:00-7:45, nicméně jeho další chování v zaměstnání už se každý den může lišit podle aktuálních povinností. Takovéto vzory najdeme v reálném světě mnohem častěji než plně periodické vzory, nicméně je také složitější je nalézt.

Další možností rozdělení je na *synchronní* a *asynchronní.* První vyžadují, aby události nastávaly v poměrně pevných pravidelných intervalech (např. 15:00 každý den), zatímco u asynchronních může kolísat v definované periodě.

Pro definici problému hledání periodických vzorů je potřeba nejprve definovat několik pojmů [15]: Časovou řadou se nazývá sekvence $S = D_1 D_2 \dots D_n$ s délkou n, značí se $|S| = n$. Její prvky D_i jsou tvořeny množinou rysů vyskytujících se v časovém okamžiku i. Ačkoliv se časové řady většinou chápou jako sekvence spojitých hodnot, zde jsou definovány tímto obecnějším způsobem. To umožňuje aplikovat metody i na jiný typ vstupních dat. *Periodickým vzorem* se nazývá sekvence množiny rysů, tj. $s = s_1 s_s \dots s_p$, kde každý s_i je buď množina rysů nebo znak $*$. Tento znak reprezentuje libovolnou událost. *Periodou* vzoru se s označuje $|s|$. Frekvence výskytu vzoru s v časové řadě S se definuje jako $freq\_count(s) = |\{i | 0 \leq i < m,\ s \text{ je validní v } D_{i|s|+1} \dots D_{i|s|+s|s|}\}|$, m je definováno jako celočíselná hodnota tak, že platí $m|s| \leq n < (m+1)|s|$, n je velikost časové řady. Dále se definuje *spolehlivost (confidence)* jako $conf(s) = freq\_count(s)/m$. Pokud je spolehlivost větší než *zadaná minimální spolehlivost,* hovoří se o *frekventovaném periodickém vzoru.*

Problém hledání periodických vzorů je pak zadán následovně. Na vstupu máme zadanou časovou řadu *S, periodu* (případně množinu period) a *minimální spolehlivost.* Úkolem dolování je nalézt všechny *frekventované periodické vzory* v časové řadě *S.* Následně bude představeno několik metod pro hledání periodických vzorů.

Úplné periodické vzory Techniky používané v této kategorii se týkají zpracování numerických hodnot a čerpají ze zpracování signálu a statistiky. Nejčastěji se používají metody jako *FFT (fast fourier transformation, rychlá Fourierova transformace)* pro převod dat z časové domény do frekvenční domény a tam pak probíhá analýza. Ta může být založena například na konvoluci. Jiné algoritmy z této oblasti jsou založeny na využití autokorelace, např. metoda *AUTOPERIOD* [5].

Částečné periodické vzory U částečně periodických vzorů se míchají dohromady jak periodické, tak neperiodické události a nelze tak použít metody transformace do frekvenční domény. Ty totiž považují časové řady jako nerozdělitelný tok hodnot.

První algortimus pro dolování částečně periodických vzorů je založen na vlastnosti **Apriori**. Na vstupu předpokládá známou periodu p. V prvním kroku spočte množinu frekventovaných *1-vzorů (vzor délky 1)* periody p. To se provede akumulací frekvence výskytu pro každý *1-vzor* v každé celém segmentu periody a poté se mezi nimi vyberou takové, jejichž frekvence výskytu není menší než $minconf \times m$. Ve druhém kroku jsou nalezeny všechny frekventované *i-vzory* periody p a algoritmus je ukončen, pokud je kandidátní množina frekventovaných *i-vzorů* prázdná. Ačkoliv takový přístup je vhodný pro dolování asociačních pravidel, zde se počet frekventovaných i-vzorů příliš nezmenšuje a ořezávání na základě Apriori vlastnosti není tak efektivní.

Tento problém Apriori přístupu byl vyřešen v dalším algoritmu z [4] pomocí tzv. *max-subpattern hit set* vlastnosti, která umožňuje provést dolování shora-dolů. Z frekventovaných 1-vzorů se zjistí kandidátní max-vzor C_{max}. Jedná se o maximální vzor, který může být vygenerován z množiny frekventovaných 1-vzorů F_1. Pro $F_1 = \{a**, b**, ***c\}$ by bylo $C_{max} = \{ab\}**c$. Potom pro každý segment periody D_i ze vstupní časové řady S se zjistí maximální podvzor vzoru C_{max}, který se nazývá *hit*. Množina *hitů* ze všech segmentů D_i tvoří *množinu hitů (hit set)*. Důležitou vlastností této množiny je, že na základě frekvencí výskytu vzorů v této množině lze odvodit všechny frekventované částečné vzory. Z těchto podvzorů je sestaven strom, ve kterém jsou uloženy i počty *hitů*. Díky tomuto stromu je schopen získat všechny frekventované vzory v celkem dvou průchodech celé časové řady.

Jeden z dalších algoritmů se zabývá redundancí částečně periodických vzorů. Autoři staví na tom, že mnoho vzorů je redundantních, jako např. "Potřebujeme zásobu pití každé 3 dny" a "Potřebujem zásoby pití každých 9 dní". Cílem přístupu je vyhnout se generování těchto redundatních vzorů. Toho je dosaženo transformací původní časové řady do bitové reprezentace a uložením bitového vektoru do hlavní paměti.

Oblast metod pro hledání periodických vzorů je mnohem větší, než předpokládá tento článek. Kromě výše uvedených přístupů zahrnuje metody pro hledání *asynchronních* periodických vzorů, hledání vzorů s *neznámou periodou* nebo dolování vzorů s *požadavky na časové mezery*. Dolování periodických vzorů může

někdy směřovat k dolování *cyklických a periodických asociačních pravidel.* Takové pravidla pak dávají do souvislosti události, které se objevují periodicky. Příkladem může být: *"Na základě denních transakcí se zjistí, že pokud skončí přednáška ve 11:40 - 12:00 bude ve 12:15 - 12:40 potřeba vydat 300 obědů."*

4.2 Objevování motivů

Potřeba objevovat motivy v temporálních datech pochází z posledních pokroků v oblasti bioinformatiky, kde se to týká hledání v DNA sekvencích. Motiv je definován jako frekventovaný vzor v časové řadě nebo také jako přibližné opakování se podsekvencí v časové řadě. Slovo *"přibližné"* napovídá, že nemusí jít vždy o stejné podsekvence, ale stačí podobné, mající např. obdobný trend řady. Algoritmy často využívají převedení časové řady do diskretizované podoby a extrakce podsekvencí pomocí posuvného časového okna. Pak mohou využívat tzv. *náhodnou projekci* nebo modelování motivů pomocí *skrytých markovských řetězců (HMM).* To se využívá hlavně u vícerozměrných časových řad [14].

4.3 Objevování anomálií

Hledání anomálií je zcela jiný pohled na dolování z dat. Zatímco ve většině případů se hledají vzory, které jsou frekventované, v této oblasti jsou pro nás zajímavé vzory, které se v datech vyskytují velice zřídka. Většinou se nazývají *anomálie* nebo také *odlehlé hodnoty (outliers).* V časových řadách to jsou hodnoty lišící se od běžných vzorů. Typickým přístupem hledání anomálií je vytvořit model *normálního chování* buď za pomocí vhodné trénovací datové množiny nebo za pomocí expertních znalostí. Pak se hledají hodnoty, které se liší výrazně od tohoto normálního chování [14].

V článcích se pak autoři zabývají možností určit pravděpodobnost, že se podezřelá podsekvence vyskytne zcela náhodně. K tomu užívají termín *pravděpodobnost falešného poplachu (false alarm).* Jiní autoři zkoumají hledání tzv. *time discords,* tj. časové neshody. Tím označují podsekvencez dlouhých časových řad, které se maximálně liší od ostatních podsekvencích v časové řadě. Ty jsou pak dobře použitelné pro hledání anomálií a jsou také poměrně oblíbené v této oblasti. Dalším používaným konceptem je vizualizační rámec VizTree pro převedení na suffixový strom, nad nímž pak probíhá interaktivní hledání za pomoci uživatele. Tento přístup byl např. použit ve společnosti *Aerospace Corporation* pro pomoc inženýrů při rozhodování zda vypustit či nevypustit *vesmírný satelit (space vehicle).* Rozhodnutí bylo založeno na datech z *telemetrie* sbíraných před vypuštěním. Své uplatnění v hledání anomálií v časových řadách nachází i technika používaná v dolování z relačních dat, *metoda podpůrných vektorů (Support Vector Machines, SVM)* [14].

5 Datové proudy

Dalším možným zdrojem pro temporální data jsou datové proudy. Ty se od předchozích dvou poměrně hodně ve způsobu, jak je potřeba je zpracovávat,

nicméně samotná data, která z datových proudů získáme mohou mít podobu buď sekvencí nebo časových řad. Ve skutečnosti právě například časové řady mají často podobu datových proudů, například při monitorování počítačového provozu na síti nebo ve finančnictví.

Datové proudy chápeme jako data, která přicházejí velice rychle, jsou časově uspořádaná, rychle se měnící, obrovská a potenciálně nekonečná [5]. Množství dat, které rychlé přichází je zpravidla tak obrovské, že není možné jej celé na některé úložiště. Algoritmy, které nad datovými proudy operují, se musí tedy vyrovnat s tím, že není možné udělat více průchodů databází. Veškeré výpočty musí být jednoprůchodové. Přístupy dolování v proudech dat se rovněž někdy uplatňují i v datech, která máme uložena v obrovských úložištích (např. v řádu tera nebo petabytů). I v těchto případech je vhodnější použít jednoprůchodové algoritmy, protože vícero průchodů by mohlo být časově velice náročné. Dalším specifikem datových proudů bývá nízká úroveň abstrakce, protože často přicházejí přímo z měřících senzorů. Analytici jsou naopak zvyklí spíše zpracovávat pojmy na vyšší úrovni abstrakce.

V oblasti hledání vzorů v datových proudech je známo především množství algoritmů pro hledání frekventovaných vzorů. Tyto vzory však v sobě neobsahují časovou informaci, nejedná se tedy o sekvenční vzory, ale pouze o takové, které se v datech vyskytují dostatečně často. Tyto algoritmy se pak dělí do tří hlavních skupin na (1) *algoritmy založené na čítači,* (2) *kvantilové algoritmy* a (3) *algoritmy založené na lineární projekci (sketches).* Známými představiteli jsou pak z první skupiny algoritmus *LossyCounting*, z druhé skupiny *GK algoritmus* a ze skupiny třetí *CountSketches* [4]. Příklady objevování temporálních vzorů pak pochází z chápání datových proudů jako časových řad.

5.1 Dolování v proudech časových řad

Problém hledání vzorů v datových proudech časových řad se sestává z průběžného monitorování proudů a hledá podsekvence, tak aby odpovídaly množině potenciálních vzorů. To, zda odpovídá, je založeno na vzdálenostní míře D a prahu δ [14].

Algoritmus *SPIRIT* se snaží nalézt skryté proměnné a trendy v proudech časových řad. Činí tak pomocí analýzy hlavních komponent *(Principal Component Analysis, PCA)* a je schopen detekovat změny v proudech a počtu skrytých proměnných. Algoritmus *Statstream* zase hledá v datech skryté silné korelace mezi páry časových řad datových proudů. Je založený na diskrétní Fourierově transformaci a velice dobře paralelizovatelný, což lze dobře využít pro zvýšení výkonu.

Algoritmus *MUSCLES* se zabývá hledáním odlehlých hodnot a snaží se předpovídat budoucí, chybějící nebo odložené hodnoty. Předpověď probíhá jak na základě aktuální sekvence, tak i sekvencí jiných. Algoritmus byl porovnáván s jinými metodami (např. autokorelace) a dával lepší výsledky. Algoritmus *AWSOM* se snaží odhalovat v proudech periodické vzory a vypořádat se při tom se šumem. K tomu využívá *vlnkovou transformaci* z oblasti zpracování časových řad

díky tomu, že ji lze spočítat rychle a inkrementálně. Navíc díky této transformaci je algoritmus schopen poměrně dobře odhalit periodicitu.

6 Závěr

Cílem tohoto článku bylo širší seznámení s metodami objevování temporálních vzorů v datech. Představuje dolování ze tří různých zdrojů, a to sekvencí, časových řad a datových proudů. Nejvíce se přitom zaobírá množstvím přístupů a algoritmů z dolováním *frekventovaných sekvenčních vzorů.* Detailně shrnout všechny oblasti by vyžadovalo mnohem více prostoru.

Motivací pro výběr tohoto tématu je aktuálně řešený projekt ve spolupráci se společností AVG. Zde máme k dispozici obrovské množství dat, ve kterých se předpokládají předem neznámé vzory. Protože tato data mají temporální charakter, je jednou z možností objevování temporálních vzorů. Právě zabývání se aktuálně frekventovanými sekvenčními vzory je důvod, proč jsou ve článku rozebrány více než ostatní. Další motivací pro článek je souvislost s tématem disertační práce v oblasti Dolování z časoprostorových dat, kde zpravidla čas bývá tou složitější doménou, se kterou je potřeba se vypořádat. Nastudované přístupy z oblasti temporálního dolování mohou být přínostné i pro oblast časoprostorových dat.

Článek dále poslouží jako základ pro aktuálně tvořený článek popisující nový algoritmus pro hledání vícevrstvých sekvenčních vzorů založeném na algoritmu GSP. Algoritmus má v současné době pracovní název hGSP, tj. hierarchické GSP.

Reference

1. Dong G. and Pei J. Frequent and closed sequence patterns. In Ahmed K. Elmagarmid, editor, *Sequence Data Mining*, volume 33 of *The Kluwer International Series on Advances in Database Systems*, pages 15–46. Springer US, 2007.
2. Mannila H., Toivonen H., and Inkeri V. A. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1:259–289, January 1997.
3. Ayres J., Gehrke J., Yiu T., and Flannick J. Sequential pattern mining using a bitmap representation. pages 429–435. ACM Press, 2002.
4. Han J. Efficient mining of partial periodic patterns in time series database. In *Proc. Int. Conf. on Data Engineering*, pages 106–115, 1999.
5. Han J. and Kamber M. *Data mining: concepts and techniques.* The Morgan Kaufmann series in data management systems. Elsevier, 2006.
6. Naisbitt J. *Megatrends : ten new directions transforming our lives.* Warner Books, New York, 1982.
7. Pei J., Han J., Asl M. B., Pinto H., Chen Q., Dayal U., and Hsu M. C. PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth. In *Proc.17th Int'l Conf. on Data Eng.*, pages 215–226, 2001.
8. Pei J., Han J., Mortazavi-Asl B., and Zhu H. Mining access patterns efficiently from web logs. In *PAKDD*, pages 396–407, 2000.
9. Wang J. and Han J. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 79–, Washington, DC, USA, 2004. IEEE Computer Society.

10. Zaki M. J. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*, 42(1/2):31–60, Jan/Feb 2001. special issue on Unsupervised Learning.
11. Agrawal R. and Srikant R. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases*, pages 478–499. Morgan Kaufmann, Los Altos, CA, 1994.
12. Mabroukeh N. R. and Ezeife C. I. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43:3:1–3:41, December 2010.
13. Ramakrishnan S. and Agrawal R. Mining sequential patterns: Generalizations and performance improvements. pages 3–17, 1996.
14. Mitsa T. *Temporal Data Mining*. Chapman & Hall/CRC data mining and knowledge discovery series. Chapman & Hall/CRC, 2009.
15. Hsu W., Lee M.L., and Wang J. *Temporal and spatio-temporal data mining*. IGI Pub., 2008.
16. Yan X., Han J., and Afshar R. CloSpan: Mining Closed Sequential Patterns in Large Datasets. In *In SDM*, pages 166–177, 2003.
17. Yang Z. and Kitsuregawa M. Lapin-spam: An improved algorithm for mining sequential pattern. In *Proceedings of the 21st International Conference on Data Engineering Workshops*, ICDEW '05, pages 1222–, Washington, DC, USA, 2005. IEEE Computer Society.
18. Yang Z., Wang Y., and Kitsuregawa M. Lapin: effective sequential pattern mining algorithms by last position induction for dense databases. In *Proceedings of the 12th international conference on Database systems for advanced applications*, DASFAA'07, pages 1020–1023, Berlin, Heidelberg, 2007. Springer-Verlag.