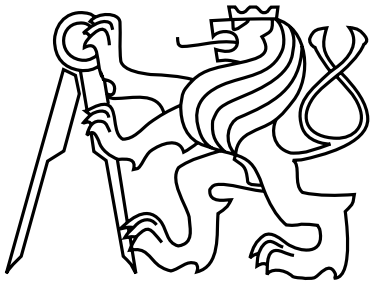
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering



Bachelor's Project

Mining of frequent subsequences in databases

Tomáš Kačur

Supervisor: Ing. Robert Kessl

Study Programme: Electrical Engineering and Information Technology

Field of Study: Computer Engineering

January 10, 2010

Aknowledgements

I would like to thank my supervisor Ing. Robert Kessl for guiding me through the process of creating this work.

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publications used.
I have no objection to usage of this work in compliance with the act §60 Zákon č. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on 8. 1. 2010 .....

Abstract

In this bachelor thesis we describe the problem of mining of frequent subsequences from databases. We describe three major algorithms: GSP, Spade, and Prefixspan. We experimentaly compare two of the algorithms: Spade and Prefixspan on synthetic and real databases for different values of minimal support. GSP is ommited from the experimental comparsion due to the proved slowness and memory use of the algorithm. The experimental results show that Spade is slower than Prefixspan, which corresponds with the experiments of the authors of the two algorithms.

Abstrakt

V tejto práci popisujeme problematiku získavania frekventovaných subsekvencií z databáz. Popis zahrňuje tri hlavné algoritmy: GSP, Spade a Prefixspan. Experimentálne porovnáme dva algoritmy: Spade a Prefixspan na umelo vygenerovaných a reálnych databázach. GSP je z porovnania vynechaný, kvôli prekazateľnej pomalosti a pamäťovej náročnosti algoritmu. Výsledky porovnania ukazujú, že Spade je pomalejší oproti Prefixspanu čo sa zhoduje s výsledkamy porovnania autorov týchto algoritmov.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

The whole variety of systems provide some kind of information. These data are collected and stored in databases. The data undergo a further analysis from which useful information can be produced providing a feedback to the system producing the data. The data can be further analyzed

Data mining involves such a procces of the analysis, where gathered data are being organized, explored and classified for particular reasons. The reasons could be general but mostly depend on the domain within which the data have been gathered, for example agriculture, bussiness, medicine and industry.

Association analysis[7] as a field of data mining coresponds to discovering interesting relationships hidden in large data sets. The uncovered relationships can be represented in the form of association rules or sets of frequent items(patterns). For example, within the business domain, given a market basket data set(filled with items bought by customers offered by market), the following relationship has been extracted: $\{Bread\} \Longrightarrow \{Butter\}$. The relationship suggests that:

- **patterns** formed from these two items of which occurancy has been found frequent within the data set. Simply put, the items had been frequently bought by customers.
- **association rule** as a strong relationship between the sale of bread and butter exists meaning that many customers who buy bread also buy butter.

Retailers can use this type of rules to help them identify new opportunities for cross-selling[1] their products to the customers.

Forming up the items using spatial or temporal relation, we get the **sequences**. For example basket market data often contains temporal information about when an item was purchased by customers. Such information can be used to piece together the sequence of transactions made by a customer over certain period of time. Adding this new dimension to the concept of itemset patterns discovery, introduced so far, the **sequential pattern discovery**(or mining of frequent subsequences) as a standalone subcategory. Resulting information may be valuable for identifying reccuring features of a dynamic system or predicting future occurrences of certain events[7].

[1] The strategy of pushing new products to current customers based on their past purchases.

1.1 Practical application of sequential mining

Besides the efficient and scalable solving of the problem of mining of frequent subsequences there is also its practical aspect that needs to be considered. The proposed solutions must be applicable within the systems that have an inherent sequential nature. The approaches are then strongly related to the character of sequence data, but may be also more general targeting more areas with similar sequential character. Having the type of sequences split according to its character as follows is reasonable:

1.1.1 Biological Sequences

[3]Biological sequences are useful for understanding the structures and functions of various molecules, and for diagnosing and treating diseases. Advances on these problems can help us to better understand life and diseases. Example of such a type of data is DNA sequence which is made up of four chemical units known as nucleotides: adenine(A), guanine(G), cytosine(C), and thymine(T),see the figure 1.1. Another example could be protein and ribonucleic acid sequences[3]. The task is to discover arrangements of regions of high occurrence of one or more units in a sequence[1]. Sequential mining of biological sequences is not a primary

```
GAATTCTCTGTAACACTAAGCTCTCTTCCTCAAAACCTTTCTAGAAGAGGTAG
```

Figure 1.1: A DNA sequence fragment

target of later described approaches of this work. The proposed solutions may be more specific because of the constraints, for example small number of types of items,. However, these approaches may be also applicable to this type of sequence data, but probably with worse results.

1.1.2 Event-based Sequences

Type of data where this work comes in. Event-based(event for short) sequence refers to an ordinaly related set of events, where event is another set of items assigned with an unique time identifier within a sequence. The are variety of systems where this type of data resides in. Hence the solutions proposed are applicable over larger spectrum of fields, not just for a specific one. The most aplicated of them are:

- **web usage mining** is the task of applying data mining techniques to discover usage patterns from Web data in order to understand and better serve the needs of users navigating on the Web[6]. For example, web server records page visits of users for a specific web site at a level of URL[2]. Seqeuence 1.2 is composed from pages visited by one user in a given time period. Finding frequent sequences then uncovers most visited sequences of pages which helps to understand and better serve the needs of user navigating on the web site.

[2]Uniform Resource Locator for example www.company.com/home

FRONTPAGE→SPORT→FOOTBALL

Figure 1.2: Web usage mining sequence

- **customer behaviour mining** used to develop marketing strategies in stores. Customers buy products in transactions. Each transaction includes the customer-id, the products bought in the transaction, and the timestamp of the transaction, e.g., see Figure 1.3. Grouping transactions by customers and sorting them in the timestamp ascending order, we get a purchase sequence database. The task is to find frequent subseqeunce that are shared by many customers. As patterns, those frequent subsequences can help to understand the behaviour of customers and identify products to be promoted which can lead to customers be retained, and stimulation of sales on other products.

```
223100, 05/26/06, 10am, CentralStation, {WholeMealBread, AppleJuice},
225101, 05/26/06, 11am, CentralStation, {Burger, Pepsi, Banana},
223100, 05/26/06, 4pm, WalMart, {Milk, Cereal, Vegetable},
```

Figure 1.3: customer purchase history

1.2 Structure of the document

The remaining of this paper is organized as follows. In Chapter 2, we introduce the problem using the stricter mathematical formulations that is used when describing later algorithms. Also an approach of hyper-lattice structure will be explained, which constitutes every algorithm for mining frequent subsequences. In the following chapters, we describe GSP algorithm as the basic algorithm, the SPADE and the PREFIXSPAN algorithms are then explained in more details. In Chapter 6, we present the experiments that we implemented on the PREFIXSPAN and SPADE algorithm and compare the performance

Chapter 2

Mathematical Foundation

This chapter describes the problem of mining frequent subsequences with respect to some mathematical background that is needed in order to understand later described algorithms.

2.1 Basic notion

In this section we introduce a basic notation that is later used for explanation of the algorithms.

Definition 2.1 (Event)**.** Let $\mathbb{I} = \{a_1, a_2, ..., a_m\}$ be a set of m distinct *items*(e.g.numbers, characters, symbols, etc.). An event is a non-empty unordered set of items $U = \{a_{u_1}, a_{u_2}, \cdots, a_{u_{|U|}}\} \subseteq \mathbb{I}$, denoted as $(a_{u_1}, a_{u_2}, \cdots, a_{u_{|U|}})$.

For example, given a set of items $\mathbb{I} = \{A, B, C\}$, a set $(AB) \subseteq \mathbb{I}$ is an *event* but (CC) is not, because item C occurs twice in the set. An event is an unordered set of items. However, without loss of generality, we will assume the items in the event are sorted in some order$(a_1 < a_2 < \cdots < a_m)$.

Definition 2.2 (Sequence)**.** Let $\mathbb{E} = \mathcal{P}(\mathbb{I})$, the powerset of $\mathbb{I}$, be a base set of events. An ordered list of N events $\alpha_i \in \mathbb{E}$, denoted as $\alpha = (\alpha_1 \rightarrow \ \alpha_2 \rightarrow \ \cdots \rightarrow \ \alpha_N)$, is called a *sequence*. Each event in the sequence, is assigned an unique identifier called the *event id*(eid for short). The set of all sequences is denoted by $\mathbb{B}$.

For example a sequence (A→BC) is considered as a 3-sequence.
A sequence with M items is called *M-sequence* , i.e., $M = \sum_{1 \leq j \leq N} |\alpha_j|$. The eids are used for ordering the events in the sequence. They mostly represent the time at which they have occurred. For a sequence α, if the event α_i occurs before or precedes event α_j, we denoted it as $\alpha_i < \alpha_j$, where $i < j$.

Definition 2.3 (Subsequence)**.** [10]Let $\alpha = (\alpha_1 \rightarrow \ \alpha_2 \rightarrow \ \cdots \rightarrow \ \alpha_n), \beta = (\beta_1 \rightarrow \ \beta_2 \cdots \rightarrow \beta_m)$ be two sequences. The sequence α is called the *subsequence* of the sequence β, or α *contains* β, denoted as $\alpha \preceq \beta$, if and only if there exists a one-to-one order preserving function f that maps events in α to events in β, that is:

1. $\alpha_i \subseteq \beta_l = f(\alpha_i)$ and
2. if $\alpha_i < \alpha_j$ then $f(\alpha_i) < f(\alpha_j)$, i.e., $\beta_k = f(\alpha_i), \beta_l = f(\alpha_j)$ such that $\beta_k < \beta_l$

For instance (A→BC) is a subsequence of (A→DE→BC) or (D→AB→BC) but not of (ABC) or (BC→A).

Definition 2.4 (Supersequence)**.** Given the sequences α and β. Let α be a subsequence of β, then β is called the *supersequence* of α.

Definition 2.5 (Prefix)**.** [8] Let $\alpha = (\alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_n), \beta = (\beta_1 \rightarrow \beta_2 \cdots \rightarrow \beta_m), m \leq n$, be two sequences. The sequence β is called a *prefix* of α if and only if:

- $\beta_i = \alpha_i$ for $(i \leq m-1)$
- $\beta_m \subseteq \alpha_m$
- all the items in $(\alpha_m - \beta_m)$ are alphabetically after those in β_m.

For example, sequences (A→B) and (A →BC→A) are prefixes of sequence (A→BC→ABC→C), but neither (A→A) nor (A→BC→B) is considered as a *prefix*.

Definition 2.6 (Suffix)**.** [8] Let $\alpha = (\alpha_1 \rightarrow \alpha_2 \rightarrow \cdots \rightarrow \alpha_n)$, $\beta = (\beta_1 \rightarrow \beta_2 \cdots \rightarrow \beta_m)$, $m \leq n$, be two sequences where β is the prefix of α. Sequence $\gamma = (\gamma_m \rightarrow \gamma_{m+1} \rightarrow \cdots \rightarrow \gamma_n)$ is called the *suffix* of α with regards to prefix β, denoted as $\gamma = \alpha/\beta$ or $\alpha = \beta.\gamma$, where $\gamma_m = (\alpha_m - \beta_m)$ such that if γ_m is not empty, the suffix is also denoted as $(\_\gamma_m \rightarrow \gamma_{m+1} \rightarrow \cdots \rightarrow \gamma_n)$.

For instance, given a sequence α=(A→ABC→AC→D→CF), (ABC→AC→D→CF) is the *suffix* with regards to the prefix (A), (_BC→AC→D→CF) is the *suffix* with regards to the prefix (A→A), and (_C→AC→D→CF) is the *suffix* with regards to the prefix (A→AB).

Definition 2.7 (Transaction)**.** Let $\alpha \in \mathbb{I}$ be a sequence and sid an integer called the *sequence identifier*. The pair (sid, α), is called a *transaction*.

Definition 2.8 (Database)**.** Let $1 \leq i \leq n, \alpha_i \in \mathbb{B}$ be a sequence, sid_i be a sequence id and $t_i = (sid_i, \alpha_i)$, be a transactions. The set $\mathcal{D} = \{t_i | sid_i \neq sid_j \; for \; i \neq j\}$ is called the *sequence database*, denoted by $\mathcal{D}$.

The database defined in definition 2.8 is often called a *horizontal database*. However, we can store the database in the so called *vertical* format. Thus, there are two primary formats of database used within this work:

Vertical database:
Each row consists of collection of items, i.e., event, and two numbers, i.e., sid, and eid uniquely identifying the collection within the database, see fig. 2.1(b).

Horizontal database:
In principle, each row corresponds to a sequence and a sequence id. Event id is omitted, because the events are sorted according to the order in which they were in put into the sequence, see fig. 2.1(a).

sid	sequence
1	A→ABC→ABD
2	D→EF→AB
3	C→AD

(a) Horizontal

sid	eid	event
1	1	A
1	2	ABC
1	3	ABD
2	1	D
2	2	EF
2	3	AB
3	1	C
3	2	AD

(b) Vertical

Figure 2.1: Two different representations of the same database

Definition 2.9 (Support)**.** [10] Let α be a sequence and $\mathcal{D}$ a sequence database. Support or frequency, denoted as $\sigma(\alpha, \mathcal{D})$, is the total number of sequences in the database $\mathcal{D}$ that contain α as a subsequence.

Definition 2.10 (Frequent Sequence)**.** [10] Let α be a sequence and $\mathcal{D}$ a database. Given a user-specified threshold called the *minimum support*, denoted as min_supp , the sequence α is frequent if $\sigma(\alpha, \mathcal{D}) \geq min\_supp$.

We denote a set of all frequent k-sequences as $\mathcal{F}_k$. The set of all frequent sequences is denoted by $\mathcal{F}$, i.e., $\mathcal{F} = \bigcup_k \mathcal{F}_k$.

Definition 2.11 (Maximal Frequent Sequence)**.** Let $\mathcal{D}$ be a database and $\mathcal{F}$ be a set of all frequent sequences in the database $\mathcal{D}$. A frequent sequence $\alpha \in \mathcal{F}$ is called *maximal frequent sequence* if for each $\beta \in \mathcal{F}, \alpha \neq \beta$, and $\beta \not\preceq \alpha$.

Definition 2.12 (Problem definition)**.** Let $\mathcal{D}$ be a sequence database, and min_sup a user-specified threshold. The problem of mining of all frequent subsequences is to find all frequent sequences $\mathcal{F}$ in the $\mathcal{D}$.

Algorithms for mining of frequent subsequences(FS in short) take a database and a min_sup as the input arguments. Output of algorithms is a set of frequent sequences contained in the database. Finding frequent sequences is a difficult computational task. For a given items that can be contained in a sequence, there is a vast number of combinations arising that can make up a sequence. For instance, having only two items A and B, the following set of distinct sequences containing two events can be made up from: (A→B),(A→A),(A→AB),(B→A),(B→B),(B→AB),(AB→A),(AB→B)(AB→AB). The total number of sequences of lenght at most k that can be comprimised from n items is n^k [10].

2.2 Hyper-Lattice based approach

In this section, we describe the basic approach that is used when decomposing the problem of mining frequent subsequences into smaller subproblems. Every set of all sequences, $\mathbb{B}$, forms the so called hyper-lattice structure. Within this structure, the sequences are being layered, meaning that each sequence is composed by adding a new item to the sequence from previous layer.

At first, without a mention of sequences problem, we excerpt the basics of lattice theory, see [2].

Definition 2.13 (Join)**.** Let P be an ordered set, and let $S \subseteq P$. An element $X \in P$ is an upper bound of S if for all $s \in S, s \leq X$. The minimum upper bound of S is called the *join*, denoted as $\bigvee S$.

Definition 2.14 (Meet)**.** Let P be and ordered set, and let $S \subseteq P$. An element $X \in P$ is an lower bound of S if for all $s \in S, s \geq X$. The maximum lower bound of S is called the *meet*, denoted as $\bigwedge S$.

Definition 2.15 (Lattice)**.** [10] Let L be an ordered set. L is called a *join (meet) semilattice*, if and only if $x \bigvee y$ $(x \bigwedge y)$ exists for all $x, y \in L$. L is called a *lattice* if it is a join and meet semilattice. An ordered set $M \subset L$ is a *sublattice* of L if $x, y \in M$ implies $x \bigvee y \in M \wedge x \bigwedge y \in M$.

Every lattice must have one minimum upper bound and maximum lower bound for every of its two elements, while semilattice must have just minimum upper bound(i.e.join semilattice) or just maximum lower bound(i.e.meet semilattice). Each ordered set has its greatest element, called the *top element*, denoted by $\top$, and conversely the lowest element, called the *bottom element*, denoted by $\bot$.

Definition 2.16 (Atom)**.** [10] Let L be a lattice and let $x, z, y \in L$. We say x is *covered* by y, if $x < y \wedge x \leq z < y$, implies $z = x$. Let $\bot$ be the bottom element of lattice L. Then $x \in L$ is called an *atom* if x covers $\bot$. The set of atoms of L is denoted by $\mathcal{A}(L)$.

Definition 2.17 (Hyper-lattice)**.** Let L be an ordered set. L is called *hypper-lattice* if its meet and join represent a set of maximal lower bounds and a set of minimal upper bounds.

Now, let us put hyper-lattice introduced so far into the sequence context.

Proposition 2.1 (Sequence hyper-lattice)**.** Let $\mathbb{B}$ be a set of all sequences over the set of items $\mathbb{I}$. Using the *subsequence* relation $\preceq$, a *sequence hyper-lattice* structure is defined on $\mathbb{B}$, in which the meet of a set of sequences is the set of maximal common subsequences and the join of a set sequences is the set of minimal common supersequences. We denote the sequence hyper-lattice strucutre by L.

Example 1 (Sequence hyper-lattice)**.** Given a set of items as $\mathbb{I} = \{A, B, C\}$. Figure 2.2 shows the fragment of sequence hyper-lattice structure spanned from the $\mathbb{I}$ using the subsequence relation. The items from $\mathbb{I}$ are also the atoms of the sequence hyper-lattice, because they cover the bottom element $\{\}$. The top element $\top$ is undefined beacuse the sequence hyper-lattice is infinite[10]. To see why the set of all sequences forms a hyper-lattice, consider the join of A and B; $A \bigvee B = \{(A \rightarrow\ B), (AB), (B \rightarrow\ A)\}$. As we can see, the join produces three minimal upper bounds, i.e., minimal common supersequences. Similary, the meet of two or more sequences produces a set of maximal lower bounds, i.e., maximal common subsequences, e.g.$(B \rightarrow\ AB) \bigwedge (AB \rightarrow\ A) = \{(AB), (B \rightarrow\ A)\}$.

Definition 2.18 (Equivalence relation)**.** Let $S \subseteq \mathbb{B}$ be a set of sequences, N be a set of non-negative integers. We define a function $p : (S, N) \rightarrow\ S$, and $p(\alpha, k) = \alpha[1 : k]$, In other words, $p(\alpha, k)$ returns the k length prefix of α. Define an *equivalence relation*, denoted as θ_k on the hyper-lattice as follows: $\forall \alpha, \beta \in L$, we say that α is related to β under θ_k, denoted as $\alpha \equiv_{\theta_k} \beta$ if and only if $p(\alpha, k) = p(\beta, k)$.

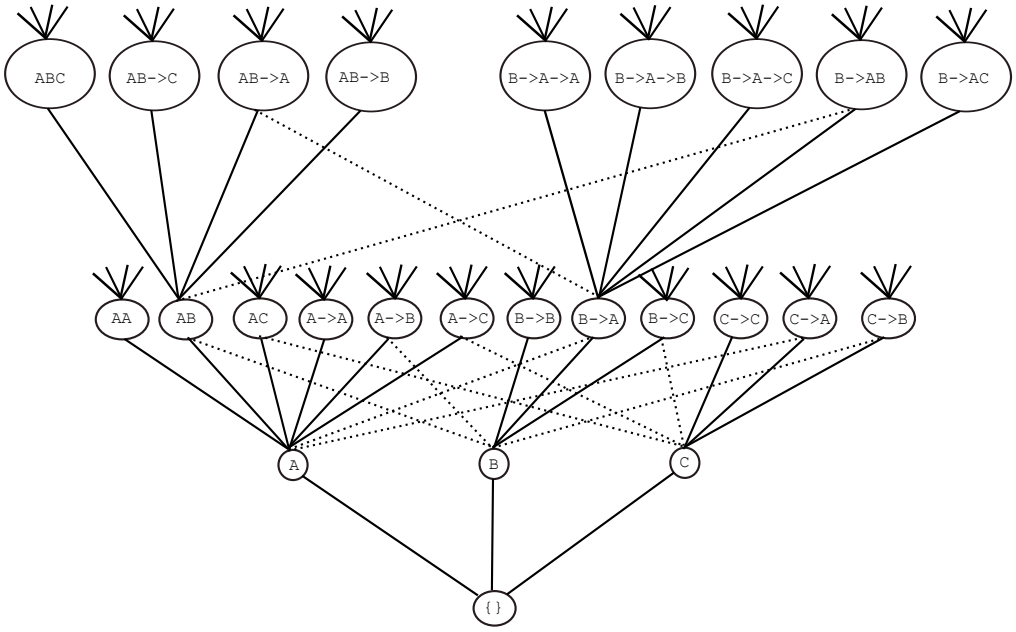


Figure 2.2: fragment of sequence hyper-lattice structure

Equivalence class is a set of sequences that share a common k length prefix, i.e., sequences related under equivalence relation θ_k. Each equivalence class $[\alpha]_{\theta_k}$ induced by the equivalence relation θ_k is a *sub-hyper-lattice* of L[10]. For instance, consider the hyper-lattice structure shown in the figure 2.2. For item A, a set of sequences induced by $[A]_{\theta_1}$ forms a hyper-lattice structure on the set, with its own set of atoms, i.e., $\mathcal{A}([A]_{\theta_1}) = \{$(AA), (AB), (AC), (A→A), (A→B), (A→C)$\}$, and the bottom element is $\perp = A$.

Definition 2.19 (Parent class)**.** Let $\alpha \in L$ be a sequence, and S be a sequence hyper-lattice structure formed from the set of sequences induced by $[\alpha]_{\theta_k}$. The set of atoms of $\mathcal{A}(S)$ is called a *parent class.*

Definition 2.20 (Anti-monotone property)**.** [7] Let $\mathbb{B}$ be a set of all sequences, and L a hyper-lattice on $\mathbb{B}$. A measure f is *anti-monotone* if $\forall \alpha, \beta \in L : (\alpha \preceq \beta) \Longrightarrow f(\beta) \leq f(\alpha)$.

If we apply support as a measure for this property, we get a powerful leverage for enumerating frequent sequences. That is:

Proposition 2.2. All subsequences of frequent sequence are frequent.

For instance, given a sequence hyper-lattice structure shown in the figure 2.2. Consider only the sequences outlined in the figure as a set of frequent sequences. It is obvious that if the sequence (ABC) is frequent, then all its subsequences, i.e., $\{(AB), (A), (B), (C)\}$, must be frequent as well. Vice versa, if the sequence $(B \rightarrow \ A)$ has been found infrequent, then all its supersequences, i.e., the sequences that contain $(B \rightarrow \ A)$, must be infrequent as well. This observation leads to very powerfull pruning strategy, where we eliminate all the sequences of whose subsequences are infrequent.

Chapter 3

The GSP Algorithm

GSP is the first FS algorithm and was introduced by *Srikant& Agrawal* in 1996. We will briefly describe this algorithm, for more details refer to[9].

The GSP algorithm is a breadth-first search algorithm that leverages the anti-monotone property, i.e., all subsequences of a frequent sequence must be also frequent. The algorithm works in phases, making multiple passes over the database. In phase k creating all frequent k-sequences, denoted by $\mathcal{F}_k$. In the first phase, all the frequent 1-sequences, i.e., items, are found. After, a candidate set of 2-sequences(C_2 for short) is made by joining $\mathcal{F}_1$ with itself, i.e., in the k-phase $C_k = \mathcal{F}_{k-1} \bigvee \mathcal{F}_{k-1}$. Another pass is made to gather the support of all sequences in the C_k. The set of all frequent sequences $\mathcal{F}_k$ is created from C_k by prunning all infrequent sequences from C_k, i.e., $\mathcal{F}_k = \{\alpha | \alpha \in C_k \text{ and } \sigma(\alpha, \mathcal{D}) \geq min\_sup\}$. This process repeats until no more frequent sequences are found.

Algorithm 1 GSP(**In**: Database $\mathcal{D}$,**In**: Integer min_supp, **In/Out**: Set F)

```
F_1 ← {frequent 1-sequences}
for k ← 2; F_{k-1} ≠ 0; k ← k + 1 do
  F_k ← {} {set of frequent k-sequences}
  C_k ← F_{k-1} ⋁ F_{k-1} {Generating of candidates set}
  for all β ∈ C_k do
    for all α ∈ D do {pass over database}
      if β ⪯ α then
        β.support ← β.support + 1
      end if
    end for
    if β.support ≥ min_supp then
      F_k ← F_k ⋃ β
    end if
  end for
  F ← F ⋃ F_k
end for
```

One of the pitfalls of the GSP is that it generates large number of candidates that is, with increasing length of sequences, the number of frequent sequences has tendency to decrease,

where the number of candidates generated by GSP is still enormous. Additionally, the GSP algorithm performs multiple scans of the database, which also slows down the process.

Chapter 4

The Spade Algorithm

Spade utilizes the prefix-based equivalence classes that decompose the original problem into smaller sub-problems that can be solved independently in main memory using simple join operations. All sequences are discovered in only three database scans[10]. It uses vertical representation of the database, i.e., each row consists of event uniquely identified by sequence id(sid for short) and event id(eid for short). We assume, that within the sequence an event with smaller eid occurred before the event with greater eid.

sequence	support
Frequent 1-Seqeucnes	
A	4
B	4
C	3
Frequent 2-Seqeucnes	
AB	2
A→B	4
A→C	2
BC	2
B→C	2
B→B	3
C→A	2
C→B	3
Frequent 3-Seqeucnes	
AB→B	2
A→BC	2
A→B→C	2
BC→B	2
BC→C	2
C→AB	2
C→A→B	2
C→B→B	2
Frequent 4-Seqeucnes	
A→BC→B	2
BC→C→B	2
C→AB→B	2

(a) Frequent sequences

SID	EID	Items
1	1	A
1	2	BC
1	3	CF
1	4	B
2	5	C
2	6	AB
2	7	B
2	8	E
3	9	BC
3	10	C
3	11	AB
3	12	BC
3	13	C
4	14	A
4	15	B
4	16	D

(b) Database

Figure 4.1: a sample vertical database

Definition 4.1 (id-list)**.** Let $a \in \mathbb{I}$ be an item. A set of pairs (sid_i, eid_i) such that event with eid_i in a sequence sid_i contains the item a is called *id-list*, denoted by $\mathcal{L}(a) = \{(sid_i, eid_i)\}$.

For example the figure 4.2 shows the id-lists for the atoms A,B and C from our sample database 4.1. One can see that atom A occurs in the following sequences and event identifier pairs called the *id-list* for item A :$\mathcal{L}(A)$ ={(1,1),(2,6),(3,11),(4,14)}.

Definition 4.2 (Support counting)**.** Let $a \in \mathbb{I}$ be an item and $\mathcal{L}(a)$ an id-list for item a. *Support* of the item a represents the number of distinct *sid* values in the $\mathcal{L}(a)$, denoted by $|\mathcal{L}(a)|$ also called the *cardinality* of the id-list.

A	
SID	EID
1	1
2	6
3	11
4	16

(a) Id-list of atom A

B	
SID	EID
1	2
1	4
2	6
2	7
3	9
3	11
3	12
4	15

(b) Id-list of atom B

C	
SID	EID
1	2
1	3
2	5
3	9
3	10
3	12
3	13

(c) Id-list of atom C

Figure 4.2: id-lists of atoms from the sample database

4.1 Temporal join of id-lists

Temporal join is a basic operation that is used when enumerating of frequent sequences. We now describe how we perform the id-list joins for two sequences. Consider an equivalence class [A→B] with atom set {(A→BC),(A→BD),(A→B→A),(A→B→C)}. Let P be the prefix (A→B), then we can rewrite the class to get [P] = {(PC),(PD),(P→A),(P→C)}. One can observe, that the class has two kind of atoms:

- **The event atoms** {PC,PD}
- **The sequence atoms** {(P→A),(P→C)}

We assume without loss of generality that the event atoms of class precede the sequence atoms. To extend the class it is suffiecient to join the id-list of all pairs of atoms. However depending on the pairs of atoms being joined, there can be up to three possible resulting sequences, i.e., three possible minimal common super-sequences:

1. **Event Atom with Event Atom:** If we are joining PC with PD, then only possible outcome is new event atom PCD.
2. **Event Atom with Sequence Atom:** If we are joining (PC) with (P→A), then only possible outcome is new sequence atom (PC→A).
3. **Sequence Atom with Sequence Atom:** If we are joining (P→A) with (P→C) what can produce up to three possible outcomes: a new event atom (P→AC), and two sequence atoms (P→A→C) and (P→C→A). A special case arises when we join (P→A) with itself, which can produce only the ew sequence atom (P→A→A).

Example 2 (Temporal Join)**.** Let us desribe how the actual join of id-lists is performed. Consider Figure 4.3 which shows hypothetical id-lists for the sequence (P→A) and (P→C). To compute the new id-list for the resulting event atom (P→AC), we simply need to check for *equality* of (sid,eid) pairs. In our example, the only matching pair are {(8,30),(8,50),(8,80)}.

This forms the id-list for (P→AC). To compute the id-list for the new sequence atom (P→A→C), we need to check for a *temporal* relationship, i.e., for a given pair (sid, eid_1) in $\mathcal{L}(P \rightarrow A)$, we check wheter there exists a pair (sid, eid_2) in $\mathcal{L}(P \rightarrow C)$ with the same sid, but with $eid_2 > eid_1$. If such a pair exists, it means that item C follows item A for the sequence identified by sid. The pair (sid, eid_2) is then added to the resulting id-list of the new sequence atom (P→A→C). The atom (P→C→A) is constructed in the same manner by reveresing the roles of (P→A) and (P→C). Since we join only sequences within a cass, which have the same prefix, we need only to keep track of the last item's eid for determining the equality and temporal relationship.

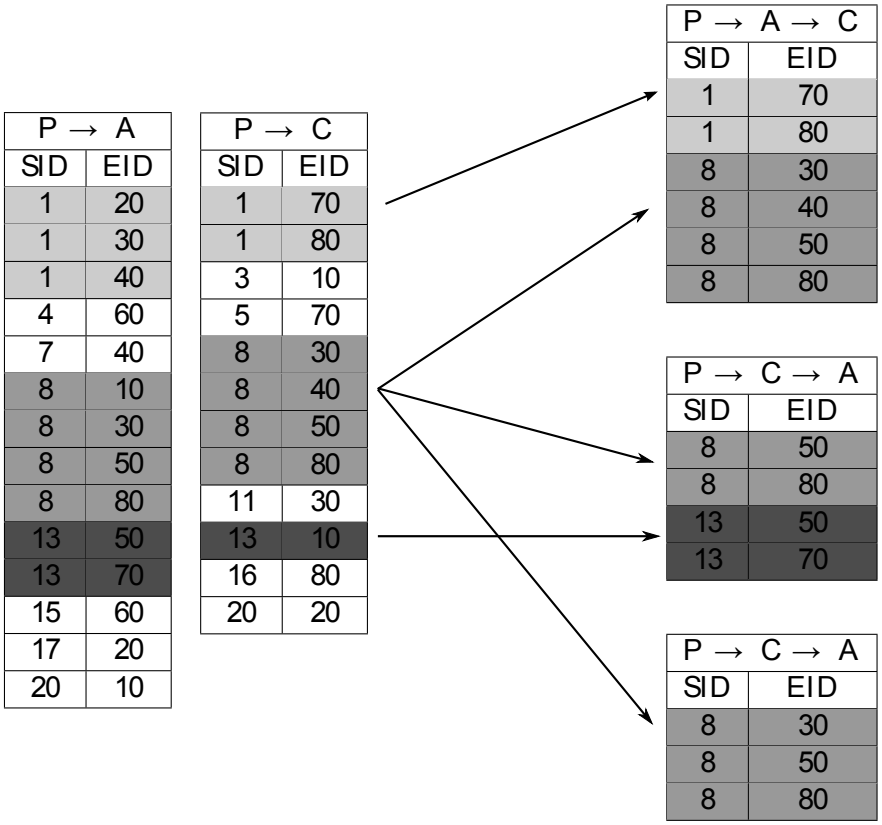


Figure 4.3: Temporal id-list join

Proposition 4.1. [10] For any $\alpha \in \mathbb{B}$, let $J = \{x \in \mathcal{A}(L) | x \preceq \alpha\}$. Then $\alpha = \bigvee_{x \in J} x$, and $\sigma(\alpha) = |\bigcap_{x \in J} \mathcal{L}(x)|$, where $\bigcap$ denotes a *temporal join*(see 4.1) of the id-lists

Above proposition states that a support of any sequence in $\mathbb{B}$ can be obtained as a temporal join of some atoms of the hyper-lattice.

Example 3 (Support Counting)**.** Given a set of atoms $J = \{A, B, C\}$ and its id-lists shown in figure 4.2. Let's say we want to compute the support of sequence $\alpha = (A \rightarrow BC)$. We start with joining the id-lists of atoms A and B by finding all occurancies of atom B after atom

A within the same sequence and store the corresponding (sid,eid) to obtain $\mathcal{L}(A \rightarrow B)$.From the figure 4.2, one can see that the pair (1,2) corresponds to such an occurance. Next we join the id-list $\mathcal{L}(A \rightarrow B)$ with the id-list of atom C by finding all equal occurancies, i.e., the same eids, in the id-lists within the same sequence, and compose the id-list $\mathcal{L}(A \rightarrow BC)$ by storing the values like (1,2). Finally, support of the sequence α is obtained by counting the number of distinct sid values in the id-list, i.e., $\sigma(A \rightarrow BC) = |\mathcal{L}(A \rightarrow BC)|$.

This process can be generalized for any set of sequences as follows:

Corollary 4.1. Let α be a sequence and $J \subseteq \mathbb{B}$ such that $\alpha = \bigvee_{\beta \in J} \beta$, then support of α can be obtained as $|\mathcal{L}(\alpha)| = |\bigcap_{\beta \in J} \mathcal{L}(\beta)|$

The spade algorithm uses a special case of this corollary, where α is a k-sequence and $J = \{\beta, \gamma\}$ contains β, γ, the two (k-1)-subsequences of α, i.e., $\alpha = \beta \bigvee \gamma$. The support of the sequence α is then obtained as a temporal join of two id-lists of the subseqeunces, i.e., $\sigma(\alpha) = |\mathcal{L}(\beta) \bigcap \mathcal{L}(\gamma)|$.

Using equivalence relation, we recursively decompose the hyper-lattice into smaller pieces which can be solved independently in the main memory. Each piece represents a parent class of the equivalence class induced by θ_k, which is recursively decomposed into smaller classes. We use *depth-first search* strategy for enumerating the frequent sequences within each parent class.

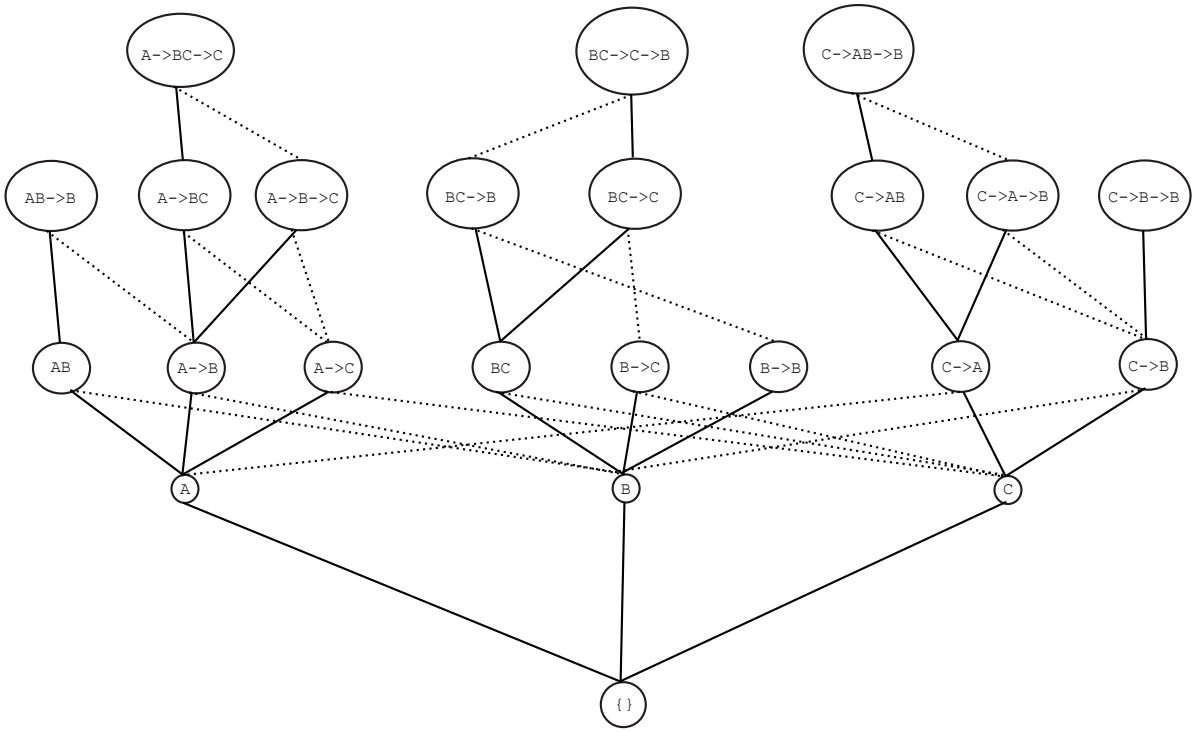


Figure 4.4: hyper-lattice structure of frequent sequences

Figure 4.4 shows the hyper-lattice structure formed from the set of frequent sequences of our sample database shown in Figure 4.1. Filled lines in the figure show the path along

which the algorithm moves within the hyper-lattice. Dotted lines represents the second subsequence that is used when composing the sequence by joining its two subsequences. The spade algorithm is summarized in Algorithm 2.

Algorithm 2 Spade(**In**: Database $\mathcal{D}$,**In**: Integer min_supp, **In/Out**: Set $\mathcal{F}$)

$\mathcal{F}_1 \leftarrow$ {frequent 1-sequences}
$\varepsilon \leftarrow$ {parent classes $[\{\}]_{\theta_1}$}
Enumerate-Frequent-Sequences(ε,min_supp,$\mathcal{F}$)
$\mathcal{F} \leftarrow \mathcal{F} \bigcup \mathcal{F}_1$

The main function of the spade algorithm is simple. At first, it scans the database $\mathcal{D}$ and find the frequent 1-sequences, which refers to the atoms of the sequence hyper-lattice. From the frequent 1-sequences, a set of atoms is generated by assigning an id-list to every of the frequent 1-sequence. The set of atoms represents a parent class for hyper-lattice induced by $[\{\}]_{\theta_1}$.

Function 1 Enumerate-Frequent-Sequences(**In**: AtomSet ε,**In**: Integer min_supp, **In/Out**: Set $\mathcal{F}$)

for all atoms $A_i \in \varepsilon$ **do**
  $T_i \leftarrow \{\}$
  for all atoms $A_j \in \varepsilon, j \geq i$ **do**
    $\alpha = A_i \bigvee A_j$
    $\mathcal{L}(\alpha) = \mathcal{L}(A_i) \bigcap \mathcal{L}(A_j)$
    if $\sigma(\alpha) \geq min\_supp$ **then**
      $T_i \leftarrow T_i \bigcup \{\alpha\}$
      $F = F \bigcup \alpha$
    end if
  end for
  Enumerate-Frequent-Sequences(T_i,min_supp,$\mathcal{F}$)
end for

The *Enumerate-Frequent-Sequences* function passes over the atom set and process each atom A_i with the other A_j from the set exactly once from which new sequence is composed. The process referes to a temporal join of the atoms, where the new atom α is obtained representing the sequence. Then a check on cardinality of the resulting atom's id-list reveals whether the sequecne is frequent. If it is true, the atom α is added to the set of atoms T_i, which represents the parent class for the next hyper-lattice induced by $[A_i]_{\theta_1}$. After the A_i pass, the function is recursively called with the T_i as a parameter.

Example 4 (Spade)**.** Let's consider the set of frequent atoms $J = \{A, B, C\}$ in Figure 4.4. The algorithm first takes the item A and subsequently applies temporal join on the rest of atoms from J, from which the set of atoms arises representing the frequent sequences set $K = \{(AB), (A \rightarrow B), (A \rightarrow C)\}$. The algorithm then moves up to the new atom set K, taking the atom (AB) applies a temporal join on the rest of the atoms, i.e., $(A \rightarrow B)$,$(A \rightarrow C)$. The new atom set originates containing only one atom representing the sequence $L = (AB \rightarrow B)$. The algorithm recursively moves to the atom set L and after

finding out that no other frequent sequence can arise from the set L, it tracks back to the previous atom set K, takes the next atom , i.e., $(A \rightarrow\ B)$ and temporaly joins it with the atom $(A \rightarrow\ C)$ in order to obtain the next atom set $N = \{(A \rightarrow\ BC), (A \rightarrow\ B \rightarrow\ C)\}$. The rest of the process is followed in the similar manner.

Chapter 5

The Prefixspan algorithm

Prefixspan is a projection-based, sequential pattern-growth algorithm, where a sequence database is recursively projected into set of smaller projected databases, and sequential pattern(i.e.frequent sequences) are grown in each projected database by exploring only locally frequent items. It is based on the initial study of the pattern-growth sequential pattern mining introduced in *FreeSpan* algorithm(see [4]). An essential adventage of the prefixspan is that no candidate sequence needs to be generated. Prefixspan uses a horizontal database representation, i.e., each row consists of (sid,sequence) pair.

SID	sequence
1	(A→BC→CF→B)
2	(C→AB→B→E)
3	(BC→C→AB→BC→C)
4	(A→B→D)

Figure 5.1: a sample horizontal database

Definition 5.1 (Sequence projection)**.** Let $\alpha, \beta, \gamma \in \mathbb{B}$ be two sequences. We say that γ is α*-projected* sequence in β iff $\alpha.\gamma$ is a maximal subsequence of β, denoted by $\beta|_\alpha$.

For instance, given the sequences β =(A→B→A→B→AC→D), α =(A→B). α*-projected* sequence in β is γ=(A→B→AC→D).

Definition 5.2 (Projected Database)**.** [8] Let $\alpha \in \mathbb{B}$ be a sequence, and $\mathcal{D}$ a sequence database . The α*-projected database* denoted as $\mathcal{D}|_\alpha$ is the collection of α-projected sequences contained in $\mathcal{D}$.

For example, consider a sequence database $\mathcal{D}$ shown in Figure 5.1. Let's say we want to project a database for a sequence $\alpha = (A)$. We find all the maximal subsequences of the sequences in $\mathcal{D}$ that contain α as a prefix, i.e., $\mathcal{D}|_\alpha$={(BC→CF→B), (_B→B→E), (_B→BC→C), (B→D)}.

Definition 5.3 (Support counting in projected database)**.** Let $\alpha \in \mathbb{B}$ be a sequence and $\mathcal{D}$ a sequence database, and β be a sequence with prefix α. The *support count* of β in α-projected database $\mathcal{D}|_\alpha$ denoted as $\sigma_{\mathcal{D}|_\alpha}(\beta)$, is the number of sequences γ in $\mathcal{D}|_\alpha$ such that $\beta \preceq \alpha.\gamma$.

The above definition states that if we have α-projected database $\mathcal{D}|_\alpha$, then the support of every subsequence β in $\mathcal{D}|_\alpha$ is the support of sequence $\alpha.\beta$ in $\mathcal{D}$.

The prefixspan utilizes a divide-and-conquer principle such that sequence databases are recursively projected into a set of smaller projected databases that represent the equivalence classes induced by the current sequential pattern, and sequential patterns are grown in each projected databases by exploring only locally frequent items. An essential requirement for this principle to work is that the order of items within an event remains to be fixed during the process. It follows the principe outlined in Spade, such that the hyper-lattice structure is being DFS searched, but with the difference in data structures used. Additionally, PrefixSpan count the support of an item (frequent or not frequent) directly in the projected database in a single scan of the projected database. Spade generates idlists for *all* items (frequent and not frequent).

Algorithm 3 Prefixspan(**In**: Database $\mathcal{D}$, **In**: Integer min_supp, **In/Out**: Set $\mathcal{F}$)

$\alpha \leftarrow \{\}$
Prefixspan-Recursive($\mathcal{D}$, α, min_supp, $\mathcal{F}$)

Function 2 Prefixspan-Recursive(**In**: Database $\mathcal{D}|_\alpha$, **In**: Sequence $\alpha = (\alpha_1 \rightarrow \cdots \rightarrow \alpha_n)$, **In**: Integer min_supp, **In/Out**: Set $\mathcal{F}$)

$\mathcal{F}_1 \leftarrow$frequent items in $\mathcal{D}|_\alpha\}$
for all items $i \in \mathcal{F}_1$ **do**
  $\beta = (\alpha_1 \rightarrow \cdots \rightarrow (\alpha_n \bigcup i))$
  $\gamma = (\alpha_1 \rightarrow \cdots \rightarrow \alpha_n \rightarrow (i))$
  if $\sigma_{\mathcal{D}|_\alpha}(\beta) \geq$min_supp **then**
    $\mathcal{F} \leftarrow \mathcal{F} \bigcup \beta$
    $\mathcal{D}' \leftarrow (\mathcal{D}|_\alpha)|_\beta$
    Prefixspan-Recursive($\mathcal{D}'$, β, min_supp, $\mathcal{F}$)
  end if
  if $\sigma_{\mathcal{D}|_\alpha}(\gamma) \geq$min_supp **then**
    $\mathcal{F} \leftarrow \mathcal{F} \bigcup \gamma$
    $\mathcal{D}' \leftarrow (\mathcal{D}|_\alpha)|_\gamma$
    Prefixspan-Recursive($\mathcal{D}'$, γ, min_supp, $\mathcal{F}$)
  end if
end for

Example 5 (Prefixspan)**.** For the sequence database $\mathcal{D}$ shown in Figure 5.1 with *min_sup* =2, sequential patterns in S can be mined by a prefix-projection method in the following steps:

1. **Find frequent 1-sequences** Scan $\mathcal{D}$ once to find all the frequent items in sequences. Each of these frequent items is a frequent 1-sequences, representing an equivalence class induced by its prefix. They are: (A):4,(B):4,(C):3, where the notation *pattern:count* represents the pattern and its associated support count.

2. **Find subsests of frequent sequences**: the subsets of frequent sequences can be mined by constructing the corresponding set of projected databases and mining each

recursively. The complete set of frequent sequences found in them are shown in Figure 4.1 or as a hyper-lattice structure in Figure 4.4, while the mining process is explained as follows:

(a) **Find sequential patterns with prefix (A)**: only the sequences containing (A) should be collected. Moreover, in a sequence containing (A), only the subsequence prefixed with the first occurrence of (A) should be considered. So the sequences in $\mathcal{D}$ containing (A) are projected with regards to (A) to form (A)-projected database, which consists of four suffix sequences: {(BC→CF→B), (_B→B→E), (_B→BC→C), (B→D)}. (_B) means that the last element in the prefix, which is (AB), forms one element. By scanning the (A)-projected database once, its locally frequent items are: (B):4, (_B):2, (C):2. Thus, all the frequent 2-sequences prefixed with (A) are found, and they are: {(A→B):4,(AB):2,(A→C):2}.

Recursively, all frequent sequences with prefix (A) can be partitioned into three subsets, i.e., equivalence classes:

i. those prefixed with (A→B), i.e., equivalence class induced by $[A \rightarrow B]_{\theta_1}$
ii. those prefixed with (AB), i.e., equivalence class induced by $[AB]_{\theta_1}$
iii. those prefixed with (A→C), i.e., equivalence class induced by $[A \rightarrow C]_{\theta_1}$

These subsets can be mined by constructing respective projected databases and mining each recursively as follows:

i. The (A→B)-projected database consists of three suffix sequences: {(_C→CF→B), (E), (_C→C)}. Recursively mining (A→B)-projected database returns 2 sequential patterns: {(_C),(C)}, i.e., {(A→BC), (A→B→C)}. Aditionally mining of (A→BC)-database returns one frequent sequence: (A→BC→C). After, the processing of the (A→B)-projected database termintes.
ii. The (AB)-projected database contains three sequences: {(B→E), (BC→C), (D)} of which subsequent mining return one frequent sequence: (AB→B).
iii. The (A→C)-projected database which contains no suffix sequence, and process terminates.

(b) **Find frequent sequences with prefix**: (B) and (C) respectively. This can be done by constructing the (B) and (C)-projected databases and mining them, respectively.

5.1 PseudoProjection

The major cost of Prefixspan is the construction of projected databases[8]. If the number and/or the size of projected databases can be reduced , the performance of the minig process can be improved. Pseudoprojection is such a technique that may reduce the size and number of projected databases.

Pseudo Projection[8]

Instead of performing physical projection, one can register the index(or identifier) of the corresponding sequence and starting position of the projected suffix in the sequence. Then, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point .

Physical projection may lead to repeated copying of different suffixes of the sequence. An index position pointer save physical projection of the suffix and, thus save both space and time of generating numerous physical projected databases.

Example 6 (Pseudo projection)**.** Given a sequence database $\mathcal{D}$ in Figure 5.1, frequent sequences can be mined using pseudoprojection method as follows: Suppose the sequence database $\mathcal{D}$ can be held in main memory. Instead of constructing (A)-projected database, we can represent the projected suffix sequences using pointer(sid) and offset(s). Given a sequence α ={(BC→C→AB→BC→C)}, the projection of sequence α with regard to the (A)-projection consists of two pieces of information:

1. a pointer to α within the database,
2. a set of integers indicating at which position the projection starts in the sequence,

The pseudoprojected databases for prefixes (A),(B),(C),(AB) are shown in Table 5.1, where \$ indicates an empty suffix and $\emptyset$ indicates no occurance of the prefix in corresponding sequence.

SID	sequence	(A)	(B)	(C)	(AB)
1	(A→BC→CF→B)	2	3,\$	4,5	$\emptyset$
2	(C→AB→B→E)	3	4,5	2	4
3	(BC→C→AB→BC→C)	5	2,6,7	3,4,8,\$	6
4	(A→B→D)	2	3	$\emptyset$	$\emptyset$

Table 5.1: sequence database and some of its pseudo projected databases

Chapter 6

Experimental evaluation

To evaluate the performance of the described algorithms, we have implemented the Prefixspan and the Spade algorithm and performed the experiments on real and synthetic datasets. In this chapter we describe the experiments. The algorithms have been implemented in C++ and compiled under `GNU G++ 4.3.2`. All experiments were performed on the *STAR computer cluster*[1] running *Linux Gentoo* with kernel version 2.6.23 and equipped with Sun Grid Engine with *Dual-Core AMD Opteron Processor@2600Mhz*.

The Experiments were performed using *python* scripts that executed the algorithms and automatically created the graphs. To verify the correctness of the implemented algorithms, we run the both programs with the same input parameters and compared the outputs. As the outputs were identical, we suggest that the implementation of the algorithms is correct.

6.1 Data sets

For the data sets used in our performance study, we use two kinds of data sets:

1. **Real data sets**

   For real data sets we have obtained the *msnbc* data set[2]. This data represents the page visits of users who visited msnbc.com on September 28, 1999. Visits are recorded at the level of URL category and are recorded in time order, for example (HOME→ NEWS→ SPORT→ FOOTBALL). In fact, each event in the data set here contains exactly up to one item, since there is no possible to visit two or more pages at once. Furthermore, the data set consists of 989818 sequences, with 5.7 average number of events within a sequence and 17 items represented by numbers. Each number represents exactly one URL. Each row in the dataset corresponds to the page visits of a user within a twenty-four hour period. Each item of a row corresponds to a request of a user for a page.

[1] http://star.felk.cvut.cz
[2] http://kdd.ics.uci.edu/databases/msnbc/msnbc.html

2. **Synthetic data sets**

   For synthetic data sets we used synthetic data generated by the *IBM data generator* [5]. The generator has been designed especially for testing sequential pattern mining algorithms. The input parameters for the generator are:

   - **The number of transactions**, i.e., sequences, denoted by C, quoted in thousands, i.e., C10 referes to 10000 sequences in a database
   - **Average number of events** within a transaction, denoted by S, i.e., S8 referes to 8 events in a sequence in average.
   - **Average number of items** within an event, denoted by E, i.e., E4 referes to 4 items in an event in average.
   - **The number of items**, denoted by I, quoted in thousands, i.e., I0.1 referes to 100 items used.

   Naming convention

   Consider a database with 10000 transactions, 8 events per transaction, 4 items per event and 1000 items. We denote the data set of this database by *C10S8E4I1*. This naming convention has been used for labeling of the data sets used in our experiments.

6.2 Results of the experiments

In our experiments, we are measuring for each dataset the time of our implementation of the Spade and Prefixspan algorithm for different values of min_supp. The input parameters of our programs are:

1. Data set file
2. Min_supp threshold

The python script executed our program and the output of the program, the execution time in seconds, was automatically drawn as a graph. Because we were loading the data sets over the network, the time doesn't include the loading of the database into the main memory. Table 6.1 summarizes our experiments, i.e., the used data set and its basic characteristics: min_supp values, time used by the run of a single experiment, and the size of the dataset in megabytes.

The results of the experiments are shown on the graphs in the Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6. For example the measurement of C10S20E2I1 in Figure 6.3. We can see that with the increasing number of min_support, run time of the algorithm decreases which is mainly due to the decreasing number of frequent sequences found in the data set. So the largest sets of frequents sequences are found at smallest min_supports. Additionally, the graphs of the experiments on the synthetic data sets show out similar behavior, while the graph of the experiment with the real data set shows different behavior. So we assume that the character of the data in the real data set is different than the character of the data in the synthetic data sets.

From the graphs, we can see that Prefixspan outperformed the spade algorithm by order of magnitude. This conclusion has been also shown in the work of authors of Prefixspan, see [8]. However, one of the reasons may be the ineffecient implementation of Spade that we have done.

Data set	min_supp thresholds range	run time range in seconds	size of data set in MB
msnbc	2000 - 10000 (2 - 10%)	158 - 1186(≈ 20 minutes)	80
C10S20E2I1	20 - 1000 (0.2 - 10%)	3 - 1800(30 minutes)	15
C10S8E8I1	20 - 800 (0.2 - 8%)	5 - 3000(50 minutes)	16.4
C10S2E2I1	1 - 10 (0.01 - 0.1%)	0 - 5	0.4
C10S20E2I0.1	50 - 1000(0.5 - 10%)	70 - 51202(14.2 hours)	8.4
C10S10E3I0.1	20 - 500(0.2 - 5%)	34 - 9181(2.5 hours)	12

Table 6.1: summarry of the tests

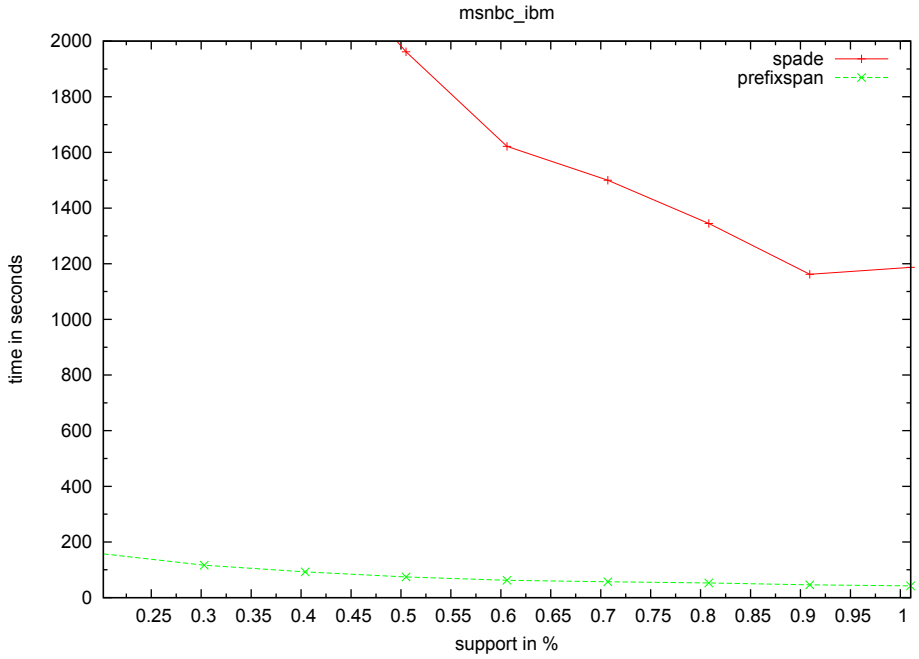


Figure 6.1: msnbc

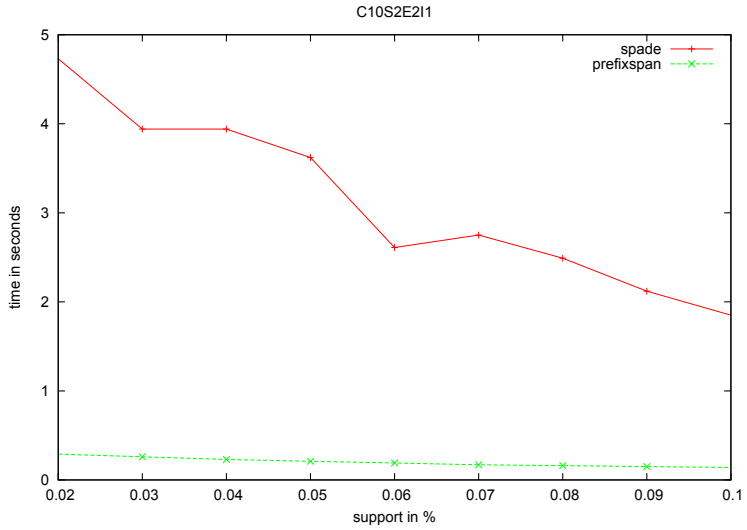


Figure 6.2: C10S2E2I1

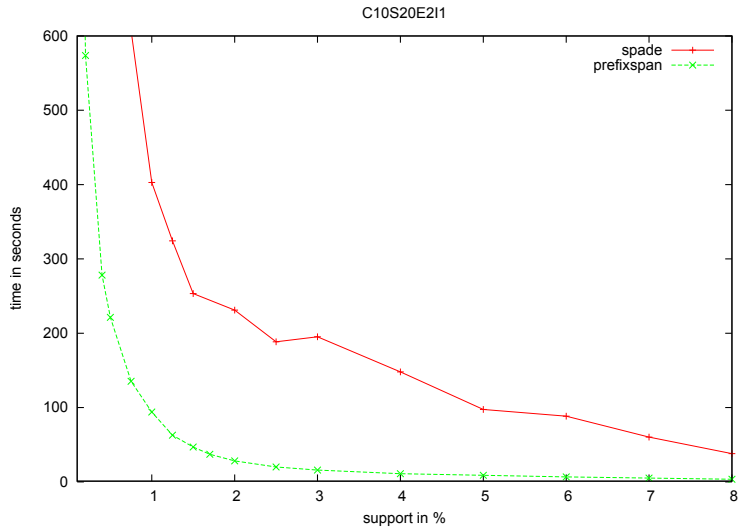


Figure 6.3: C10S20E2I1

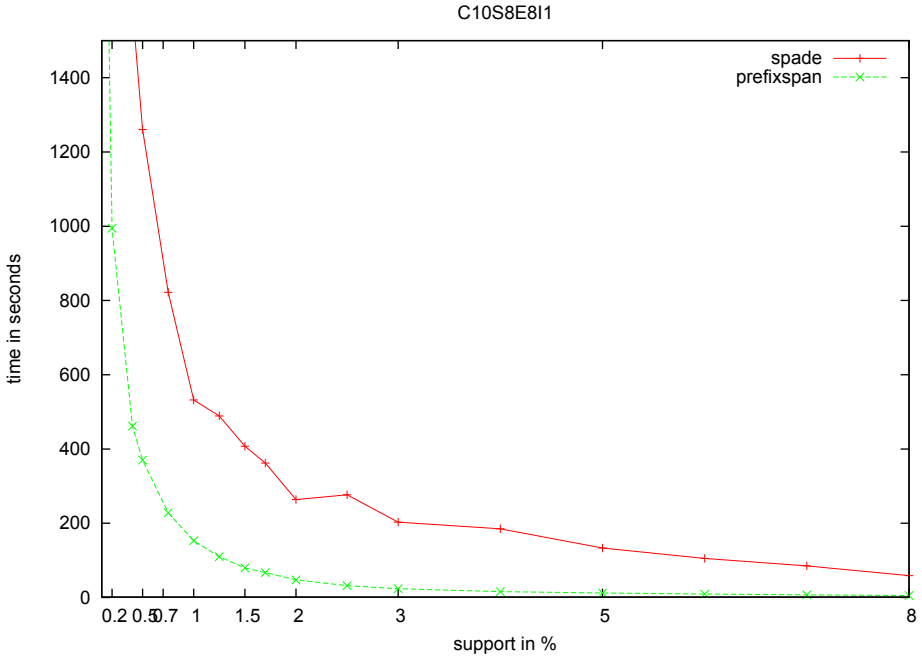


Figure 6.4: C10S8E8I1

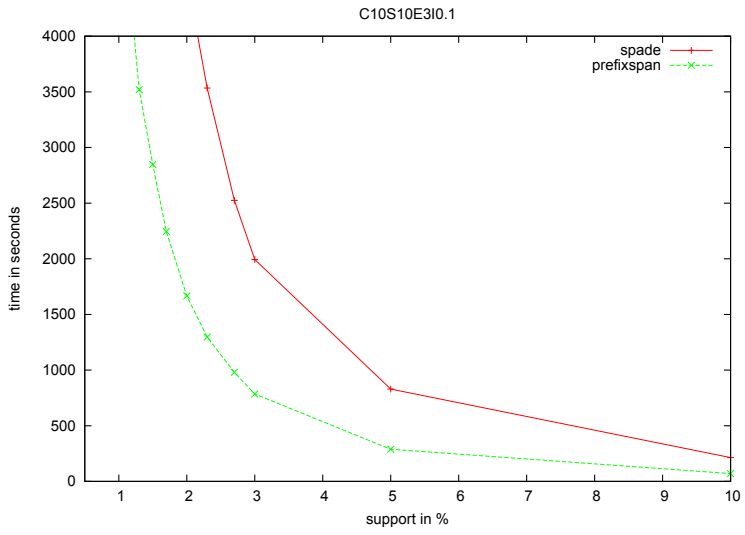


Figure 6.5: C10S20E2I0.1

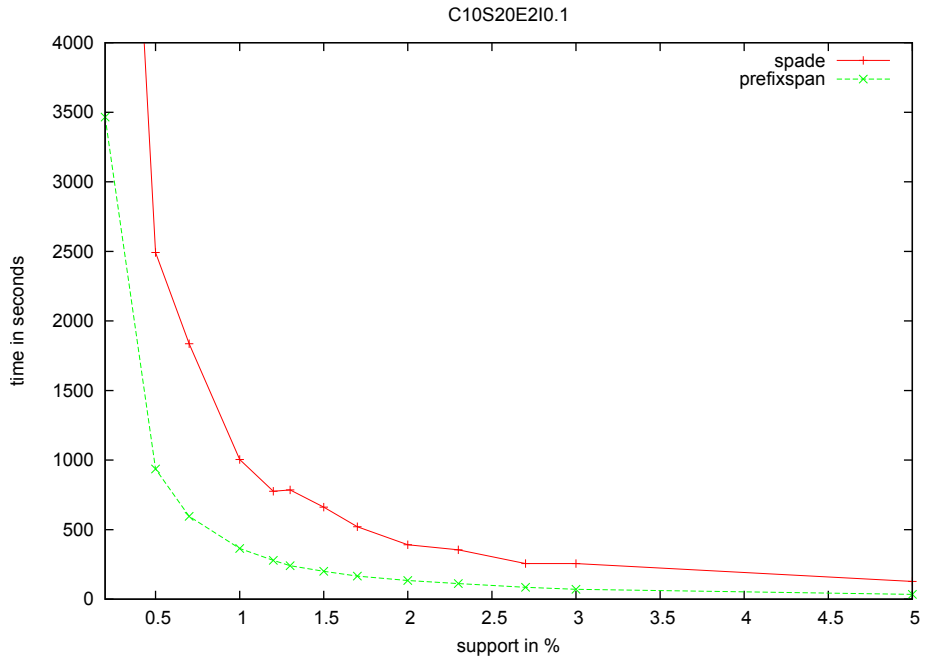


Figure 6.6: C10S10E3I0.1

Chapter 7

Conclusion

In this work, we described the problem of mining frequent subsequences and three major algorithms that have been proposed for the problem: GSP, Prefixspan, Spade. We implemented two algorithms: Spade, Prefixspan and performed experiments on the synthetic and real databases. The experiments showed that Spade is outperformed by Prefixspan by order of magnitude, which could be also due to inefficient implementation. It was quite difficult to choose elementary data structures that were used in more complex data structures used in the algorithms. It has always been a trade-off between programming comfort and the performance.

Bibliography

[1] P. P. G. Benson and G. Kollios. Discovering frequent poly-regions in dna sequences. In *Proceeding of IEEE ICDM 2006 Workshop on Data Mining in Bioinformatics*, pages 94–98. Departments of Biology and Computer Science,Boston University, 2006.

[2] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[3] G. Dong and J. Pei. *Sequence Data Mining*. Springer, 2007.

[4] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of ACM SIGKDD 2000 Sixth International Conference on Knowledge Discovery and Data Mining*, pages 355–359, 2000.

[5] IBM. Ibm quest market-basket synthetic data generator. `http://www.cs.nmsu.edu/~cgiannel/assoc_gen.html`.

[6] R. Iváncsy and I. Vajk. Frequent pattern mining in web log data. *Acta Polytechnica Hungarica, Journal of Applied Science at Budapest Tech Hungary, Special Issue on Computational Intelligence*, 3(1):77–90, Jan 2006.

[7] Pan-Ning Tan, M. Steinbach, and V. Kumar. *Introduction To Data Mining*. Addison Wesley, 2006.

[8] J. Pei, J. Han, and B. Mortazavi-Asl. Prefixspan: Mining sequential patterns effeciently by prefix-projected pattern growth. In *Proceeding of ICDE 2001 17th International Conference on Data Engineering*, pages 215–226, 2001.

[9] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th Internation Conference EDBT on Extending Database Technology*, 1996.

[10] M. J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning Journal*.

Appendix A

Compilation and usage

Both implemented algorithms have its own source directory, i.e., spade and prefixspan directories. Each of the directories contains a makefile which can be used to compile the program. To compile the program using makefile type following into command line:

```
make -all
```

The compilation process generates the output program file, i.e., spade or prefixspan, into the build directory within the each of the source directories.

The programs can be executed with the following arguments:

- **Spade**

  ```
  ./spade
  ```

  - `[--filename|-f <string>] - input file name containg a database`
  - `[--support-threshold|-s <integer>] - minimum support threshold as an abs. num.`
  - `[--testing|-t] - prints only the run time`
  - `[--print-freq-seqs|-p] - prints the result set of frequent sequences`

- **Prefixspan**:

  ```
  ./prefixspan <filename> <support_threshold> [-show-database] [-show-patterns]
  [-t]
  ```

  - `<filename> - input file name containg a database`
  - `<support_threshold> - minimum support threshold as an abs. number`
  - `[-t] - prints only the run time`
  - `[--show-patterns] - prints a result set of frequent sequences`
  - `[--show-database] - prints the input database`

Only vertical database format is accepted by the programs. Where each line of the input data file consists of: `sequence_id event_id item`

Appendix B

CD Content