UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

Techniques for the Discovery of Temporal Patterns

Técnicas para el Descubrimiento de Patrones Temporales

D. Antonio Gomariz Peñalver
2014

UNIVERSIDAD DE MURCIA

Departamento de Ingeniería de la Información y las Comunicaciones

UNIVERSITEIT ANTWERPEN

Departement Wiskunde - Informatica

Ph.D. Thesis

Techniques for the Discovery of Temporal Patterns

Author

Antonio Gomariz Peñalver

Advisors

Roque Marín Morales

Manuel Campos Martínez

Bart Goethals

December 2013

Techniques for the Discovery of Temporal Patterns
Título en español: Técnicas para el Descubrimiento de Patrones Temporales
Nederlandse titel: Technieken voor de Ontdekking van Temporele Patronen

The research in this thesis was completely financed by a PhD grant from the Seneca Foundation (Regional Agency for Science and Technology of the Region de Murcia).

UNIVERSIDAD DE MURCIA

Departamento de Ingeniería de la Información y las Comunicaciones

UNIVERSITEIT ANTWERPEN

Departement Wiskunde - Informatica

Tesis Doctoral

Técnicas para el Descubrimiento de Patrones Temporales

Autor

Antonio Gomariz Peñalver

Directores

Roque Marín Morales

Manuel Campos Martínez

Bart Goethals

Diciembre 2013

Técnicas para el Descubrimiento de Patrones Temporales
English title: Techniques for the Discovery of Temporal Patterns
Nederlandse titel: Technieken voor de Ontdekking van Temporele Patronen

Este trabajo debe agradecerse a la financiación otorgada por la Fundación Séneca (Agencia de Ciencia y Tecnología de la Región de Murcia) a través de una beca FPI que ha permitido, al autor, la dedicación exclusiva a la investigación.

Agradecimientos. Acknowledgements

Si echo la vista atrás, sobre este tiempo que he dedicado a la realización de la tesis, una extraña mezcla de sentimientos y sensaciones vienen a mi mente. Por encima de haber pasado por momentos magníficos o por algunos otros en los que he tenido ganas de enviarlo todo a "tomar viento fresco", lo que puedo afirmar, sin lugar a dudas, es que he sido ayudado por personas extraordinarias. Todas ellas me han animado y empujado a completar este trabajo.

Entre todas estas personas, me gustaría empezar agradeciendo la labor de mis dos directores de tesis de la Universidad de Murcia, Roque Marín y Manuel Campos. A Roque tengo que agradecerle todas las molestias que se ha tomado, a lo largo de estos cuatro años, en mostrarme el mundo de la investigación. A Manolo, directamente, tengo que darle las gracias por todo: por trabajar conmigo, codo con codo, durante todo este tiempo; por saber gestionar mis estados de ánimo consiguiendo que afrontara algo positivamente cuando me vencía el desánimo; por siempre ofrecerme una buena alternativa cuando algo empezaba a complicarse; y por involucrarte tanto en mi trabajo diario. Has sido un enorme apoyo y te aseguro que te estaré por siempre agradecido.

Muchas gracias, también, a José Tomás Palma por darme la oportunidad, junto a Roque, de iniciarme en el mundo de la investigación. Muchas gracias, igualmente, a Pepe Juárez por todo su apoyo tanto en mis inicios en el laboratorio como en el plano personal.

No puedo olvidarme tampoco de aquellas personas con las que he tenido el placer de compartir tantísimas horas: mis compañeros de Laboratorio. Gracias a aquellos que siguen estando: Morales y su sigilo; Eduardo, a pesar de que siga maldiciendo sus malos chistes; y Latifa, aportando al laboratorio un componente exótico. Gracias, con la misma intensidad, a aquellos compañeros que estuvieron: la pareja Tamara y Miki, el enigmático José Salort, el jovenzuelo Rubio y, particularmente, mi querida amiga Bea.

Mención aparte merece mi amiga Ana, la pequeña rubita de la tercera planta, compañera de fatigas en todo momento. Es difícil imaginarme esta tesis sin ti, desde su inicio hasta su final, compartiendo todo tipo de ilusiones, dudas, quejas y lamentos. Te puedo asegurar que es a ti a quien corresponde el mejor recuerdo de todo este tiempo.

Dank u wel! Bart Goethals, my third advisor, since you make me possible starting a fantastic collaboration with the University of Antwerp, leading to a Joint PhD. Thank you very much for your support in the (almost) eight months that I spent with you and for making my stay easier.

Thanks a lot to the whole ADReM group, composed of great people with whom I have shared a really good time. Without wanting to forget anybody, I thank Floris, Nikolaj, *Michael, Jilles, Koen Smets, "New" Koen, Álvaro, the sleepy Cheng, the smiling Sandy and the young and charming Tayena.*

In a separate paragraph, three people deserve a special mention: Boris, Emin and Jeremy. Thanks Boris, for all your help in managing my bureaucracy and for all the good moments that we spent watching the "Reds". I still remember some rewarding conversations that we had during my stays or this last summer in Almería. Thanks Emin, for all your help and support in the Lab, and, especially, all the good moments and conversations that we shared tasting beers after working or during some weekends. Thanks Jeremy for the great time that we had several nights while we talked about very different topics while you practised your (Colombian) Spanish. All of you are the kind of people that you remember and keep as friends for the whole life.

Gracias, también, al señor jefe Randalf, por ser siempre una via de escape a mis sofocos mientras degustabamos un buen café. Muchas gracias, especialmente, por tu ayuda en la maquetación de la tesis, y así poder aliviar algo mis agujereados bolsillos.

Gracias, muchísimas gracias, a mis padres, José y Virtudes, por lo pesados que han sido y su insistencia en que me formara. Ellos son los principales responsables de que hoy pueda estar agradeciéndoos a todos vuestra ayuda en la realización de esta tesis doctoral. Mamá, Papá... vosotros, desgraciadamente, no tuvisteis la oportunidad de estudiar cuando erais jóvenes, pero, en algunos aspectos, podríais dar lecciones de vida a muchísima gente.

Gracias a Virtu, mi querida hermana, por su constante apoyo y por creer, ella más que yo, que esto lo acabaría terminando algún día.

Finalmente, y como se espera, la persona más importante de todas: Laura, la chica de mi vida. Muchas gracias por tu apoyo, cariño y entrega. Gracias por aguantar todos mis altibajos, cambios de humor, y todo tipo de tonterías. Siempre has sido mi mayor creyente, animándome e iluminándome cuando las cosas las veía muy negras. Gracias también, por acompañarme y sacrificar tanto tiempo libre estos últimos años, cosa que espero poder devolverte con creces en no mucho tiempo.

¡Gracias!

Thanks!

Abstract

One of the problems that information technologies have had to confront in recent years is the analysis of the huge amount of data that originates during the daily activities of organisations or people. This analysis may consist of searching for models or patterns that will assist in understanding the data or behaviour of these organisations or people. One essential component in this kind of knowledge is the temporal dimension. When time is included in the patterns, they provide much more information but also become more complex.

Sequence Data Mining (SDM) is an area in the field of Knowledge Discovery whose aim is to extract sets of frequent patterns that occur, ordered in time, in a database. SDM techniques have been used in a wide array of application domains, such as the discovery of motifs in DNA sequences, the analysis of customer purchase sequences, web click streams, and so forth.

The patterns obtained in these domains depend on the nature of the data under analysis and the purpose of the analysis. On the one hand, there are simple patterns that only contain point events ordered in time. For example, a pattern can model the behavior of a person who, during the night, wakes up, drinks water, goes to the bathroom, and then goes back to bed again. On the other hand, much more complex patterns include interval events with temporal distances between them. For example, a person sleeps for 3 hours, then spends 3 minutes in the kitchen and, after watching TV for 40 minutes, goes back to sleep for 2 more hours. There is a wide range of patterns between these two extremes.

This thesis presents a number of contributions to the SDM field. Firstly, we propose a clear categorisation of patterns and algorithms within SDM. We principally study three different dimensions: the representation of the patterns, their expressiveness and the search strategy used to mine patterns. In this categorisation we have found certain gaps in the state-of-the-art algorithms. Secondly, in order to complete those gaps that have not yet been explored, we provide five new algorithms that use different representations and strategies. Finally, we discuss the convenience of using a particular algorithm depending on the properties of the database and the patterns that we are interested in finding.

Resumen en español

Uno de los problemas a los que las tecnologías de la información han tenido que enfrentarse en los últimos años es el análisis de una enorme cantidad de datos originada en las actividades cotidianas de organizaciones o personas. Este análisis puede consistir en la búsqueda tanto de modelos como patrones que ayuden en la comprensión de los datos o el comportamiento de estas organizaciones o personas. Una componente esencial asociada a este tipo de conocimiento es la dimensión temporal, que cuando es tenida en cuenta en los patrones, no sólo proporciona mucha más información, sino también los convierte en más complejos.

La minería de datos de secuencias (SDM) es un área en el campo de la detección de conocimiento en bases de datos (KDD) cuyo objetivo es extraer los conjuntos de patrones frecuentes que se encuentran, ordenados en el tiempo, en una base de datos. Algunas técnicas de SDM han sido empleadas en una amplia variedad de dominios de aplicación, tales como el descubrimiento de patrones en secuencias de ADN, el análisis de secuencias de compras de clientes, número de clics en una web, etcétera.

Los patrones que se obtienen en estos dominios dependen de la naturaleza de los datos que son objeto de análisis y del propósito de dicho análisis. Por un lado, hay patrones sencillos que sólo contienen eventos que denotan puntos ordenados en el tiempo. Por ejemplo, un patrón puede modelar el comportamiento de una persona que, durante la noche, se despierta, toma agua, va al baño, y luego regresa a la cama. Por otra parte, otros patrones mucho más complejos incluyen eventos que denotan intervalos con distancias temporales entre ellos. Por ejemplo, una persona duerme durante 3 horas, seguidamente pasa 3 minutos en la cocina y, después de ver la televisión durante 40 minutos, vuelve a dormir durante 2 horas más. Entre estos dos extremos existe una amplia gama de diferentes patrones.

Esta tesis supone distintas aportaciones al campo de la SDM. En primer lugar, proponemos una clasificación clara de los patrones y algoritmos dentro de la SDM. Hacemos un estudio claramente diferenciado en tres distintas dimensiones: representación de los patrones, su expresividad y la estrategia de búsqueda utilizada para la extracción de patrones frecuentes. En esta clasificación hemos encontrado algunas lagunas en los algoritmos existentes en el estado del arte. En segundo lugar, con el fin de completar las lagunas que aún no han sido exploradas, ofrecemos cinco nuevos algoritmos que utilizan diferentes representaciones y estrategias. Finalmente, discutimos la conveniencia de utilizar un algoritmo determinado en función de las propiedades de la base de datos y los patrones que son objeto de nuestro interés.

Nederlandse samenvatting

Een van de problemen die de informatietechnologie de afgelopen jaren moest confronteren is de analyse van de enorme hoeveelheid data die ontstaat tijdens de dagelijkse activiteiten van organisaties of mensen. Deze analyse kan bestaan uit het zoeken naar modellen of patronen die helpen bij het begrijpen van de gegevens of het gedrag van deze organisaties of personen. Een essentieel onderdeel van dit soort kennis is de tijdsdimensie. Wanneer de tijd wordt opgenomen in de patronen, bieden ze veel meer informatie, maar ze worden ook complexer.

Sequence Data Mining (SDM) is een gebied van Knowledge Discovery waarvan het doel is om verzamelingen van frequente patronen die zich, in een bepaalde volgorde, in een database voordoen te vinden. SDM technieken worden gebruikt in een breed scala van toepassingsgebieden, zoals de ontdekking van motieven in DNA sequenties, de analyse van klantenaankoop sequenties, web click streams, enzovoort.

De patronen gevonden in deze gebieden zijn afhankelijk van de type van de gegevens die geanalyseerd en het doel van de analyse. Aan de ene kant zijn er eenvoudige patronen die alleen gebeurtenissen geordend in de tijd bevatten. Bijvoorbeeld, een patroon kan het gedrag van een persoon die, tijdens de nacht, wakker wordt, water drinkt, naar de badkamer gaat, en dan weer terug naar bed gaat, modelleren. Anderzijds, veel complexer patronen kunnen gebeurtenissen met temporele afstanden daartussen beschrijven. Bijvoorbeeld, een persoon slaapt 3 uur lang, verblijft dan 3 minuten in de keuken, en, na 40 minuten lang TV te kijken, gaat terug nog 2 uur lang slapen. Er bestaat een breed scala aan patronen tussen deze twee uitersten.

Dit proefschrift presenteert een aantal bijdragen aan het SDM veld. Ten eerste beschrijven we een duidelijke indeling van patronen en algoritmes binnen SDM. We bestuderen voornamelijk drie verschillende dimensies: de representatie van de patronen, hun expressiviteit, en de zoekstrategie gebruikt om de patronen te mijnen. In deze indeling hebben we bepaalde hiaten gevonden in de state-of-the-art algoritmen. Ten tweede, om deze hiaten, die nog niet zijn onderzocht, in te vullen, bieden we vijf nieuwe algoritmen die verschillende representaties en strategieën gebruiken. Tenslotte bespreken we het gemak van het gebruik van de voorgestelde algoritmen, afhankelijk van de eigenschappen van de database en de patronen die we willen vinden.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and motivation

This thesis has been developed in the context of AI-SENIOR research project. The main goal of this project is to establish the behavioural patterns of subjects who are continuously monitored. More specifically, these subjects, are elderly people, many of whom may suffer from diseases that determine their quality of life. The project is focused on three groups of elderly people and the development and validation of results. These three groups consist of: (1) a population of people who have some form of Cardiovascular Disease, (2) another of people suffering from some form of Chronic Pulmonary Diseases, and (3) one formed of elderly people who, although they do not suffer from any of these diseases, live alone and require the monitoring of their daily activities in order to recognise situations that occur on a daily basis.

All three of these groups are at some risk, owing either to their living alone or to their diseases, and assistance is necessary to improve their quality of life. What is more, all the groups represent a large proportion of the older population, and the cost of assisting this population is recognised as being a particularly important problem for public administration.

Sensory devices were therefore installed in every home, consisting of various sensors, depending on the group studied, which were both wearable and non-invasive. There are several types of sensors, some of which are binary such as those that detect presence, others of which are numerical such as those that detect temperature, and others of which are multi-variable, such as those that detect activity (which provides an acceleration in each axis). Some are high frequency (the people in the first two scenarios), while others have very low frequencies (those more generally found in the third scenario). Some of these sensors additionally provide a fixed frequency time series, while others only provide a signal when activity is detected. In this context, the amount of time that a person remains in a room may have different meanings. We think it is therefore necessary to model both simple events and interval events.

One of the subobjectives of the project is to apply a knowledge discovery task that will allow us to do the following: i) determine the level of monitoring that a person must have in his/her home; ii) tune the alarm system as accurately as possible. We believe that sequential patterns are very useful in these tasks as regards modelling user behaviour. It is necessary to define these patterns at different levels of complexity in order to obtain a more detailed view when necessary. It is thus possible to begin by obtaining simple patterns such as a person who, during the night, wakes up, drinks water, goes to the bathroom, and then goes back to bed again. On the other hand, much more complex patterns include interval events with temporal distances between them, and other complex patterns can also be defined such as the fact that, a person sleep for 3 hours, then spends 3 minutes in the kitchen and, after watching TV for 40 minutes, goes back to sleep for 2 more hours. There is a wide range of patterns between these two extremes.

The objective of this Thesis, is to propose a general framework that is capable of dealing with databases from ambient intelligence, in addition to finding interesting algorithms that may be of great value as regards this problem.

1.2 Temporal Data Mining

One of the problems that information technologies have had to confront in recent years is the analysis of the huge amount of data that originates during the daily activities of organisations or people. The general process during which this analysis is carried out is called Knowledge Discovery in Database (KDD), which is defined as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" [Piatetsky-Shapiro and Frawley, 1991]. This process is required to obtain useful and valuable information for organisations or people. For example, it permits us to analyse a medical problem and extract information that may be relevant for a decision making process in order to: study manifestations and signs during the course of a disease, analyse the causes of mortality of a group of patients with a particular problem, etc.

The essential step in KDD is the Data Mining (DM) phase, which incorporates very different techniques from the fields of machine learning, statistics, decision-making systems and artificial intelligence in general, along with other ideas from computing and information management systems. DM consists of applying data analysis and discovery algorithms that produce a particular enumeration of structures over the data [Fayyad et al., 1996], in which these structures can be patterns or models [Mannila, 2002]. DM is used in several tasks such as classification, clustering, prediction or the discovery of interesting patterns (e.g. hidden patterns, trends or other relations present in the data).

In most of the techniques proposed in DM literature, data analysis is carried out without taking into account the temporal component of the data. Nevertheless, there are fields of application in which the data to be analysed have temporal interdependencies and the order between them is fundamental to the analysis. Temporal Data Mining (TDM) has emerged as an important branch of DM and can be defined as the activity of searching for interesting associations or patterns in large sets of temporal data accumulated for other purposes [Bettini et al., 1996]. TDM is capable of mining activity, and of inferring associations of contextual and temporal proximity, some of which may also indicate a cause-effect association. This important kind of knowledge can be overlooked when the temporal component is ignored or treated as a simple numeric attribute [Roddick and Spiliopoulou, 2002].

Within TDM, there is an important task called Temporal Pattern Mining (TPM) which searches for the different kinds of patterns that can be obtained in at least a minimum number of entries, called support, from a temporal database. These patterns may be very different depending on the context being studied and the type of information being sought. The two main approaches in this area are called Itemset Data Mining (IDM) and Sequence Data Mining (SDM). While IDM represents the temporal concept of synchronicity and extracts a set of itemsets or transactions (items which occur simultaneously) from input databases (composed of itemsets), SDM represents the temporal concept of order and uses a sequence input database, in which each sequence is a list of itemsets. While IDM is the simplest case, SDM [Agrawal and Srikant, 1995] is much more complex. In SDM, the space search is bigger than in the IDM case, since in addition to searching for frequent itemsets, it is also necessary to consider the temporal relation between these itemsets. The research into SDM has generated several algorithms which are applied in different applications, such as discovering of motifs in DNA sequences, the analysis of customer purchase sequences and Web click streams, and so on.

The TPM problem can be analysed from different perspectives: pattern representation and expressiveness. With regard to the representation of a pattern, two main decisions are typically made: whether to consider the events as points or as intervals. In the first case, for a same event, each appearance is considered as a time point. For instance, if we are working with a granularity of hours in a hospital environment and we consider an event that denotes whether a particular patient has a fever ($fever\_present = true$), we can consider that we have points of that event every hour at which the fever is present.

Unfortunately, this kind of simple abstraction is not always sufficient to express all of the events. In many real world scenarios such as medical, multimedia, meteorology or finance domains, the events tend to persist for periods of time rather than occurring instantaneously, and the data are usually sequences of events with both a beginning and an end time. Each interval event therefore has an associated duration as regards the time at which this pattern is activated. To continue with the previous example, with an interval representation we consider an appearance of the event during the whole time period in which that event is activated, i.e. if, for example, we detect fever in a patient for five consecutive hours, we consider those five hours to be a single appearance of the pattern $fever\_present$.

With regard to the expressiveness of the event relations in the patterns, there are several options, but most algorithms are usually focused on qualitative or quantitative relations. Qualitative relations only represent the order relation that appears between two events. For instance, if we consider two event points A and B that appear at times 1 and 2, respectively, there will be the relation "before" between them, and, similarly, if these two points appear at time 1 and 100, the relation between A and B will be exactly the same, and it will not be possible to distinguish any nuances in it. On the other hand, when it is necessary to express the exact relation, then a numeric distance is also included in the temporal relation. For instance, if we were to consider the same example as before, we would discover two different relations: one appearing before, with a time unit of temporal distance ($A\ before[1]\ B$), and a second one appearing before, with 99 time units of difference ($A\ before[99]\ B$).

Most of the efforts made to date have determined the use of a type of representation and expressiveness to mine an input database under such constraints. Most works therefore differentiate between point-based or interval-based databases, considering either qualitative or quantitative relations. Unfortunately, there are some environments in which the use of mixed cases would be very convenient. For instance, it would be possible to cite some examples in which certain attributes should be recorded as intervals, while others could occasionally occur, as points. In these cases, it is necessary to simultaneously include points and intervals, both in the database and in the patterns that we find. For example, in the case of a medical database there may be diagnosis and treatment events, in which a diagnosis is established at one moment, but a treatment has a duration.

In SDM this has traditionally been considered as a point representation with qualitative relation as a data model. This point-based representation generates patterns with three different temporal relations (*before*, *equals* and *after*. Nevertheless, if an interval based representation is used, all the common problems in SDM are particularly complex. While in the point case we only use the point relations to express the temporal order within a pattern, in the time interval representation there are different ways in which to relate intervals, such as Allen's interval algebra [Allen, 1983] or Freksa's semi-interval relations [Freksa, 1992] representation framework. While the point algebra only uses three different relations [Vilain, 1982], Allen's interval algebra has thirteen relations that configure a very expressive and complete language. This language makes the pattern representation and the tasks related to temporal reasoning much more complicated. These relations are actually crucial bottlenecks when we are interested in developing an efficient and effective algorithm with which to mine complex patterns, since these relations may lead to the generation of a much larger number of candidate sequences than when only points are considered. To avoid all the drawbacks resulting from interval relations, several pattern representations which provide the key features have been proposed: compactness, legibility and non-ambiguity.

All of the qualitative representations can be converted into quantitative representations in order to show the temporal distance between the different events, signifying that the relations become more complicated and their processing is more costly. This leads to a greater pattern explosion since, for a given support, when we consider this quantitative relation we find less occurrences of more patterns with only qualitative relations. It is therefore necessary to emphasise the scalability of the algorithms used to mine these patterns. This scalability will be reflected in the search strategies, pattern representation, the candidate generation, the support counting and the pruning methods.

Several works have been carried out for SDM, among which three main search strategies can be clearly identified: 1) Apriori-Style [Srikant and Agrawal, 1996], 2) Vertical Database Format [Zaki, 2001] and 3) Pattern Growth [Pei et al., 2004] strategies. While Apriori-based algorithms consist of executing a continuous loop which carries out a candidate generation followed by a support checking phase for each generated candidate, the Vertical Database Format and Pattern growth strategies are more direct methods which, in general, obtain better results as regards time execution and memory consumption.

Most of the sequential pattern mining algorithms developed to date have been designed for the specific database configurations present in a number of real datasets; for example datasets with short sequences, with a limited number of events, with sequences in which the events are not repeated, etc. In general, when these conditions change, the performance of these algorithms decreases dramatically. Even when the algorithms can complete their executions, the number of frequent patterns that are usually found is extremely large, and there are a lot of non-meaningful patterns.

One interesting approach used in IDM in order to solve the aforementioned problem consists of searching for patterns with concrete properties such as closed itemsets [Pasquier et al., 1999]. Searching for closed itemsets provides two benefits at the same time: a reduction in the number of candidates, and the obtaining of a more compact output while maintaining the maximum amount of information. Since mining closed sequences is quite similar to mining closed itemsets, a closed pattern mining strategy can also be used in SDM.

In this thesis we propose to make a comprehensive study and comparison of all the alternatives previously set out. We also propose several algorithms, all of which are capable of working under different representation, expressiveness and database configurations.

1.3 Objective and sub-objectives

The main goal of this thesis is to define a general framework for Sequential Data Mining (SDM) methods, strategies, data structures, and expressiveness of patterns, and to develop new algorithms at all the possible levels of SDM. In order to achieve this goal, we have defined the following objectives:

- We show, through a survey of the state-of-the-art, the problems resulting from SDM and we make an in-depth comparison between all the strategies applied to SDM.

- We have structured and organised the knowledge about SDM with regard to the following dimensions: pattern representation, pattern expressiveness and search strategies. This organisation is summarised in Tables 1.1, 1.2 and 1.3, which are a guide to the objectives of this thesis shown below.

- We show and compare all of the problems derived from the different levels of the pattern representation (points, intervals and points and intervals).

- We visualise and contrast the issues associated with the different levels of expressiveness (qualitative and quantitative patterns). For both representation and expressiveness we bridge some of the gaps found in Table 1.1. We specifically deal with the patterns highlighted in the first, second, fourth and fifth rows.

- We define four new algorithms that are capable of mining point-based, interval-based and point and interval-based databases. Each algorithm implements a concrete strategy and can find both qualitative and quantitative relations. The names of these algorithms are PaGAPIS, FaSPIP, BreadthPIS and DepthPIS for qualitative patterns, and PaGAPIMS, FaSPIMP, BreadthPIMS and DepthPIMS for quantitative patterns. We additionally study the pruning methods that can be applied, and the convenience of using them. These algorithms are used to bridge some of the gaps in Table 1.2, thus making new combinations possible in terms of representation, expressiveness and data representation, which have not yet been covered. Note that while PaGAPIS and PaGAPIMS are based on the Pattern Growth strategy, the remaining algorithms that we propose are based on the Vertical Database Format strategy. We implement algorithms from both strategies in order to make a fair comparison between only the strategies, in a comprehensive manner.

- We discuss the convenience of using each search strategy for pattern representation and expressiveness in a given input database. We also analyse which strategy is most suitable for application when dealing with specific features in the input database. Concretely, we study the convenience of using the different algorithms proposed in this thesis, making a broad comparison.

- We propose a new algorithm with which to manage pattern explosion problems by means of mining Closed sequences and to bridge some of the gaps in Table 1.3. This new algorithm is the first to be based on the Vertical Database Format strategy which, by means of several pruning methods, avoids the generation of non-Closed sequences.

To date, and to the best our knowledge, no previous work has addressed a comprehensive framework for sequential pattern mining by considering an incremental complexity of the patterns applied to a single domain. There are several works with algorithms that address a single problem with a single type of target patterns, showing its advantages over other approaches to the problem addressed. We wish to point out that the databases with which we have experience, cover the entire spectrum of possibilities with patterns.

In order to facilitate the reading of this thesis we now provide two tables which show the dimensions analysed with the aforementioned objectives. Table 1.1 depicts two groups: the point-based patterns, which appear in the first three rows; and the interval-based patterns in the last three rows. Each group can have two kinds of relations: qualitative and quantitative. There are also two kinds of quantitative patterns: those with exact temporal distances and those in which the distance is bounded by a range from a lower to an upper value.

Type of pattern	Representation
Point-based qualitative	$A < B$
Exact point-based quantitative	$A < [5]B$
Bounded point-based quantitative	$A < [5,7]B$
Interval-based qualitative	$A\ o\ B$, $A < C$, $B\ o\ C$
Exact Interval-based quantitative	$A[4]\ o[2]\ B[7]$, $A[4] < [3]C[8]$, $B[7]\ o[2]\ C[8]$
Bounded Interval-based quantitative	$A[4,6]\ o[2,4]\ B[7,10]$, $A[4,6] < [1,3]C[8,12]$, $B[7,10]\ o[2,5]\ C[8,12]$

Table 1.1: Classification of the different types of patterns.

Table 1.2 shows a classification of the different algorithms developed for SDM to date (details of these algorithms will be provided in the state-of-the-art section). In this table, the dimensions are the kind of input database (point, intervals or point and interval databases), the expressiveness (qualitative or quantitative relations), and the mining strategy chosen (Apriori-style, Vertical Database Format and Pattern Growth). As will be observed, there are still some gaps that need further study. Besides, Table 1.3 shows the point-based algorithms for mining closed sequences already developed in SDM. As can be seen, there is not any algorithm that follows a Vertical Database Format strategy.

Database	Representation	Distances	Strategies		
			Apriori	Pattern Growth	Vertical Database Format
Points		qualitative	Apriori-All, GSP, PSP, TSET	FreeSpan, PrefixSpan, Memisp	SPADE, SPAM, Prism
		quantitative	I-Apriori, Yoshida	MisTA, QprefixSpan, i-PrefixSpan	
Intervals	Points	quantitative		QTPrefixSpan	
		qualitative		T-PrefixSpan	
	Intervals	quantitative	QTempIntMiner	QTiPrefixSpan	
		qualitative	IEMiner, Karmalego	Armada	H-DFS
Points and Intervals	Points	quantitative	ASTPminer		
		qualitative	HTPM	CTMiner, CEMiner	
	Intervals	quantitative			
		qualitative			

Table 1.2: Classification of the different algorithms already developed for SDM.

1.4 Structure of the thesis

The remainder of the work is organised as it follows:

Apriori	Pattern Growth	Vertical Database Format
	CloSpan, Bide	

Table 1.3: Classification of the different point-based algorithms for closed patterns already developed for SDM.

Chapter 2 introduces a survey concerning previous work carried out in SDM and the issues that we are dealing with. Concretely, we provide details about Sequence Data Mining, its structure and its problems, and how each of the strategies used in SDM (Apriori-style, Vertical Database Format and Pattern Growth) works. We also explain and compare the most relevant algorithms for each strategy in both point and interval cases. We additionally show the pattern explosion problem associated with SDM, and we explain the principal algorithms for closed patterns.

Chapter 3 describes two new algorithms called PaGAPIS and FaSPIP, both of which are capable of finding qualitative patterns from point and intervals databases. While PaGAPIS is based on the Pattern Growth strategy, FaSPIP is based on the Vertical Database Format strategy. Their quantitative versions, called PaGAPIMS and FaSPIMP, are then described in Chapter 4. In both the qualitative and the quantitative cases, the algorithms use a representation based on boundary points, i.e., each input sequence with points and intervals is translated into another sequence with only points. These algorithms are inspired by the original point-based algorithms and deal with intervals by means of only their boundary points. Finally, we compare the convenience, advantages and drawbacks of using the different strategies to mine point and interval patterns.

In Chapter 5 we describe two new algorithms called BreadthPIS and DepthPIS, which are also capable of finding qualitative patterns in points and intervals databases. As before, in Chapter 6 we define their quantitative versions. Both algorithms belong to the Vertical Database Format strategy. These algorithms use Triangular Matrix Representation (TriMax) in order to represent both qualitative and quantitative point and interval patterns. The algorithms use a temporal reasoning method in order to obtain the minimum number of candidates, and only the frequent patterns are selected from among them. Although both algorithms use the same pattern representation and the same method to count the support, they are based on different search methods, candidate generation and pruning methods. As before, we conclude by comparing the convenience, advantages and drawbacks of the algorithms in both the qualitative and the quantitative cases.

In Chapter 7 we introduce the ClaSP algorithm, the first Vertical Database Format Algorithm for Closed Sequences. Furthermore, we show that, for some database configurations, the execution of a Vertical Database algorithm is more convenient than any other strategy. We also show that in the database configurations considered, ClaSP clearly outperforms other state-of-the-art algorithms.

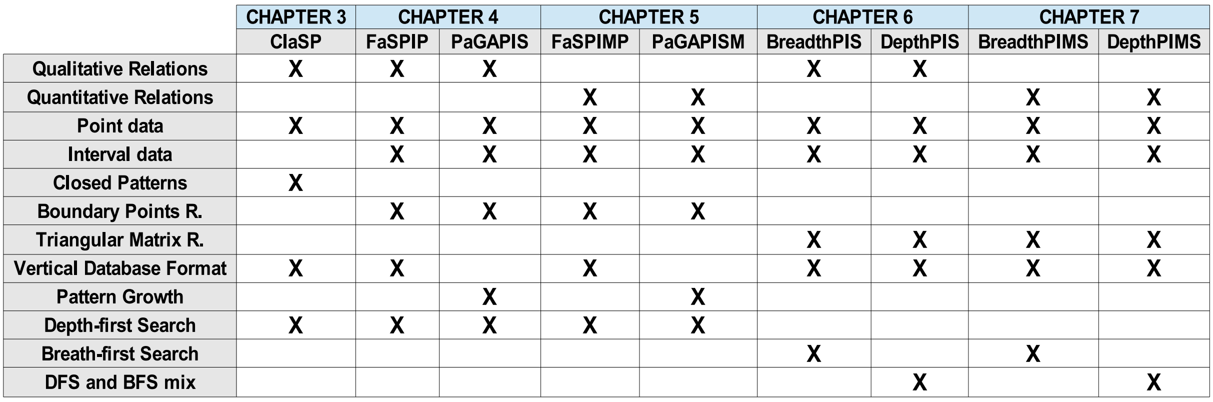
Chapter 8 shows an overall comparison of the four algorithms developed to mine both qualitative and quantitative patterns in databases with points and intervals. These comparisons allow us to show what advantages there are between using one or another representation, and the domain of Vertical Database Format strategy over the Pattern Growth strategy when mining point and interval databases.

Chapter 9 provides a general discussion on the key points of the algorithms previously described. Firstly, we show the advantages and drawbacks, in terms of execution time and memory consumption, that are encountered when working with points, intervals or points and intervals; and also when using qualitative or quantitative relations. We also check the benefits as regards the different pattern representations used in the algorithms, and the problems associated with each particular algorithm. We then analyse which search strategy is the most adequate for the different problems as regards resolving depth-first search (DFS), breadth-first search (BFS) or a mixture of the two. Finally, we discuss which candidate generation strategy is the best in general terms: that used in Apriori-based, in Vertical Database Format or in Pattern-Growth algorithms.

Finally, this thesis concludes in Chapter 10, which shows our conclusions. We highlight the contributions that we have made, and then raise some open questions that could be relevant as possible future works.

In order to help understand the different algorithms developed in this Thesis, Figure 1.1 shows the main characteristics taken into account in each Chapter and in each algorithm. These characteristics are:

- The algorithm deals with qualitative patterns.
- The algorithm deals with quantitative patterns.
- The algorithm mines time point data.
- The algorithm mines time interval data.
- The algorithm finds the final set of closed patterns.
- The algorithm uses a Triangular Matrix to represent the relations between the different events.
- The algorithm uses a sequence of boundary points to represent a pattern.
- The algorithm follows the Vertical Database Format strategy.
- The algorithm follows the Pattern Growth strategy.
- The algorithm uses a depth-first search method.
- The algorithm uses a breadth-first search method.
- The algorithm uses a mixture of both depth-first and breadth-first searches.



	CHAPTER 3	CHAPTER 4		CHAPTER 5		CHAPTER 6		CHAPTER 7	
	ClaSP	FaSPIP	PaGAPIS	FaSPIMP	PaGAPISM	BreadthPIS	DepthPIS	BreadthPIMS	DepthPIMS
Qualitative Relations	X	X	X			X	X		
Quantitative Relations				X	X			X	X
Point data	X	X	X	X	X	X	X	X	X
Interval data		X	X	X	X	X	X	X	X
Closed Patterns	X								
Boundary Points R.		X	X	X	X				
Triangular Matrix R.						X	X	X	X
Vertical Database Format	X	X		X		X	X	X	X
Pattern Growth			X		X				
Depth-first Search	X	X	X	X	X				
Breath-first Search						X		X	
DFS and BFS mix							X		X

Figure 1.1: Topics considered in the Chapters (and in their algorithms) of this Thesis.

Chapter 2

State of the Art

In this Chapter, we provide a survey in SDM strategies as well as an explanation of the main algorithms and improvements with respect to the pattern representation for the mining of both qualitative and quantitative patterns. After the general introduction to SDM, we begin by defining the main concepts and notations in Section 2.2. Section 2.3 provides a wide overview of strategies, algorithms, improvements and comparatives between the different algorithms for qualitative and quantitative patterns that are commonly used with point based databases. Then, Section 2.4 makes the same overview as in the previous Section, but with a time interval representation. Finally, in Section 2.5, we establish our major conclusions regarding the state of the art in TPM.

2.1 Introduction

One of the problems that information technologies have been facing the last years is the analysis of the huge amount of data originated in the daily activity of organizations. The general process in which this analysis is carried out is called Knowledge Discovery in Database (KDD) and is defined as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data" [Piatetsky-Shapiro and Frawley, 1991]. This process is required to obtain useful and valuable information for the organization. For example, we can analyse a medical problem and extract information that can be relevant for a decision making process: studying manifestations and signs in the course of a disease, analyzing the causes of mortality of a group of patients with a particular problem, etc.

The essential step in KDD is the Data Mining (DM) phase, which incorporates very different techniques from the fields of machine learning, statistics, decision-making systems and artificial intelligence in general, as well as other ideas from computing and information management systems. DM consists of applying data analysis and discovery algorithms that produce a particular enumeration of structures over the data [Fayyad et al., 1996] where these structures can be patterns or models [Mannila, 2002]. DM is used in several tasks such as classification, clustering, prediction or discovery of interesting patterns (e.g. hidden patterns, trends or other relations present in the data).

In most of the techniques proposed in the DM literature, data analysis is carried out without taking into account the temporal component of the data. Nevertheless, there exist fields of application where the data to be analysed have temporal interdependencies, being the order between them fundamental to the analysis. Temporal Data Mining (TDM) emerges as an important branch of DM and it can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes [Bettini et al., 1996]. TDM has the capability of mining activity, inferring associations of contextual and temporal proximity, some of which may also indicate a cause-effect association. This important kind of knowledge can be overlooked when the temporal component is ignored or treated as a simple numeric attribute [Roddick and Spiliopoulou, 2002].

Inside of TDM, there is an important task called Temporal Pattern Mining (TPM) which searches for the different kind of patterns that can be obtained from a temporal database. These patterns can be very different depending on the context we are studying and the type of information that we are looking for. In this way, in [Moerchen, 2010] three different aspects are differentiated within a pattern:

Temporal Data Models. How is the temporal data representation? Mainly, we deal with time point representation or time interval representation.

Temporal Concepts. What semantic we want to express in each pattern? Normally this semantic is related to order or concurrency (or closeness) relation.

Temporal Operators. How are the data compared and combined by the different algorithms in the TPM task? For instance, an event A can be before another event B and close to another event C.

Regarding the temporal concepts and operators, there are mainly three different approaches which deal with different kind of patterns (in order of complexity):

Itemset Data Mining (IDM). In this approach [Agrawal et al., 1993] a list of transactions is considered as input database, where each transaction denotes an occurrence time and is composed by several items that occur at the same time. Therefore, this approach finds patterns with a temporal concept of synchronicity, i.e. the frequent sets of items which occur together at the same time in a minimum of database entries.

Sequence Data Mining (SDM). In this approach [Agrawal and Srikant, 1995] the key point is the use of temporal order. The input database is a list of sequences where each sequence can have several transactions (as in IDM). The patterns found are subsequences which appear in the database a minimum number of times.

Discovery of Frequent Episodes (DFE). In this case, the input database is a long and single sequence of different items occurring in an increasing temporal order [Mannila et al., 1997]. The searched patterns in this approach correspond to partial orders where both the temporal order and concurrency concepts are shown.

While IDM is the simplest case, SDM [Agrawal and Srikant, 1995] and DFE [Mannila et al., 1997] are much more complex. In particular, SDM has a bigger space search than

IDM since there are more operators and possibilities to create patterns, and hence, the basic operations of subsequence checking can easily take longer.

SDM has been widely studied [Agrawal and Srikant, 1995; Srikant and Agrawal, 1996; Zaki, 2001; Han et al., 2000; Pei et al., 2004; Ayres et al., 2002], with a number of algorithms and applications, such as the discovery of motifs in DNA sequences, analysis of customer purchase sequences, web click streams, and so forth.

The task of discovering the set of all frequent sequences in large databases is challenging as the search space is extremely large. Different strategies have been proposed so far, among which Vertical Database Format [Zaki, 2001] and Pattern Growth [Pei et al., 2004] strategies are the most popular ones. These strategies show good performances in databases containing short frequent sequences or when the sequences are very common in the database.

Most of the algorithms have been designed to find qualitative patterns, being mainly focused on the order between the points in patterns but regardless of the distance between them. However, in some contexts we need to know the distance between the different occurrences, being this distance either the exact temporal distance or a range of distances within a lowest and upper bounds. This kind of patterns is known as quantitative patterns and they make the task of mining frequent patterns different. In this case, the concept of frequent pattern is changed, leading us to a different amount of frequent patterns for the same supports as in in the qualitative case. The strategies for mining frequent quantitative patterns are exactly the same that in the qualitative version, but all the strategies need to bear in mind the issues typically found in the mining of quantitative patterns.

According to these different aspects in a pattern, the majority of the efforts in SDM so far have considered a point representation as data model. In this representation, patterns can have three different time operators to associate points (*before*, *equals* and *after* relations). It is possible to include also temporal distances if we are interested in quantitative patterns. Considering every pattern as a concatenation of points leads us to the discovery of very simple patterns, reduced to an ordered sequence of events. Unfortunately, sometimes this abstraction is not enough to express the complexity of temporal relationships between different events. In many real world scenarios such as medical, multimedia, meteorology or finance domains, their events tend to persist for periods of time instead of having an instantaneous occurrence, and the data is usually a sequence of events with both beginning and end time. For instance, in the medical field, a simple ordered sequence of events for a myocardial infarction could be something as "chest pain → increasing of cardiac enzymes" [Chen et al., 2011], while if we consider the duration of such events, we could find "chest pain contains increasing of cardiac enzymes". Therefore, the correlation between the symptoms and the diseases, or the influences between different diseases and others can be studied. For the latter example, we would rather use an interval-based representation than a point-based one, and for that representation, all the common problems in SDM are particularly relevant.

In the time interval representation there are different ways to relate intervals each other, where the most known ones are the Allen's interval algebra [Allen, 1983], the Freksa's semi-interval relations [Freksa, 1992], Roddick's Midpoint interval relations [Roddick and Mooney, 2005] and the Time Series Knowledge Representation [Mörchen, 2006], being the former the most widely used representation framework. We can perceive that time point-based data are a special case of the time interval-based data, where both beginning and end points occur at the same time (for each interval) and the relations between these points become simpler (*before*, *equals* and *after*). Therefore, while with points there are only three different relations [Vilain, 1982], in the Allen's interval algebra there are thirteen relations that configure a very expressive language, and make much more complicated the pattern representation and the tasks related to temporal reasoning. These complex relations are actually crucial bottlenecks when we are interested in developing an efficient and effective algorithm for mining complex patterns, since these relations may lead to generate a larger number of candidate sequences than when we execute the same example with a point based representation. To avoid those drawbacks, several pattern representations have been proposed [Kam and Fu, 2000; Höppner, 2001; Wu and Chen, 2007; Patel et al., 2008]. These representations provide the key features: to be compact, legible and non-ambiguous.

Furthermore, in several contexts, a more detailed representation is needed to show the exacted temporal distances between the different types of considered interval events or the duration of the intervals themselves, being necessary a more explicit representation. For example, in [Palma et al., 2006] a real clinical case is described for an acute myocardial infarction and the time distances between the different interval events become crucial.

2.2 Preliminaries and general notation

Let consider an item as a pair of a label and a duration $i = \langle e, d \rangle$, where i denotes the occurrence of an event e that is extended over the time with a duration d. This duration d has the value 0 if the item i denotes a point whereas, if an interval is represented, the duration value is greater than 0, $d > 0$. For an item, we define a boundary beginning point as the temporal point where the item starts, and we define a boundary end point as the temporal point where that item ends. We assume an item as anything that can occur in the context which we are dealing with, i.e. customer purchases in a shop, medical parameters that are monitored by physicians in a hospital, access to websites by an user, etc.

Let $\mathcal{I}$ be a set of items. A set $I = \{i_1, i_2, \ldots, i_k\} \subseteq \mathcal{I}$ is called an itemset or k-itemset if it contains k-items. For simplicity, from now on we denote an itemset I as a concatenation of items between brackets. For instance, the itemset $I_1 = (abcd)$ is a 4-itemsets composed of items a, b, c and d. Without loss of generality, we can establish an order between the events of every itemset I. We say that, for an itemset, an item is less than another item if the first one is less than the second one in a lexicographic order. More formally: $(i_m = \langle e_m, d_m \rangle < i_n = \langle e_n, d_n \rangle) \Leftrightarrow (e_m <_{lex} e_n)$.

Let us define a transaction as a duple (t, I), being t the timestamp when the itemset $I \in \mathcal{I}$ occurs. We call $|T|$ the number of items in the itemset of T. Besides, we say that a transaction T_m is less than a transaction T_n if its itemset time t_m is less than the itemset time t_n of T_n: $\forall T_m, T_n \in s, T_m = \langle t_m, I_m \rangle, T_n = \langle t_n, I_n \rangle$, $(T_m < T_n) \Leftrightarrow (t_m < t_n)$. Therefore, for a transaction $T = (t_k, I_k)$, each item i_k that is included in I_k starts at t_k time and it is extended so long as its duration d denotes, ending at time $t_k + d$.

We will use the concept of transaction in our algorithms. In addition, when we need to refer to the itemset of the transaction, we directly write it as a transaction in order to enhance readability. Therefore, in some occasions, we will use between transactions some operations that are defined for itemsets in transactions, such as the set membership. For instance, if we have transactions $T_1 = \langle t_1, I_1 \rangle$ and $T_2 = \langle t_2, I_2 \rangle$, and we want to mean that $I_1 \subseteq I_2$, we also use the expression $T_1 \subseteq T_2$, denoting exactly the same.

A sequence s is an ordered set $s = \{T_1, T_2, \ldots, T_n\}$, where each T_k, $\forall k : 1 \leq k \leq n$, is a transaction. All the transactions in a sequence are sorted in an increasing temporal order: $\forall T_m, T_n \in s, T_m = \langle t_m, I_m \rangle, T_n = \langle t_n, I_n \rangle$, $(T_m < T_n) \Leftrightarrow (t_m < t_n)$.

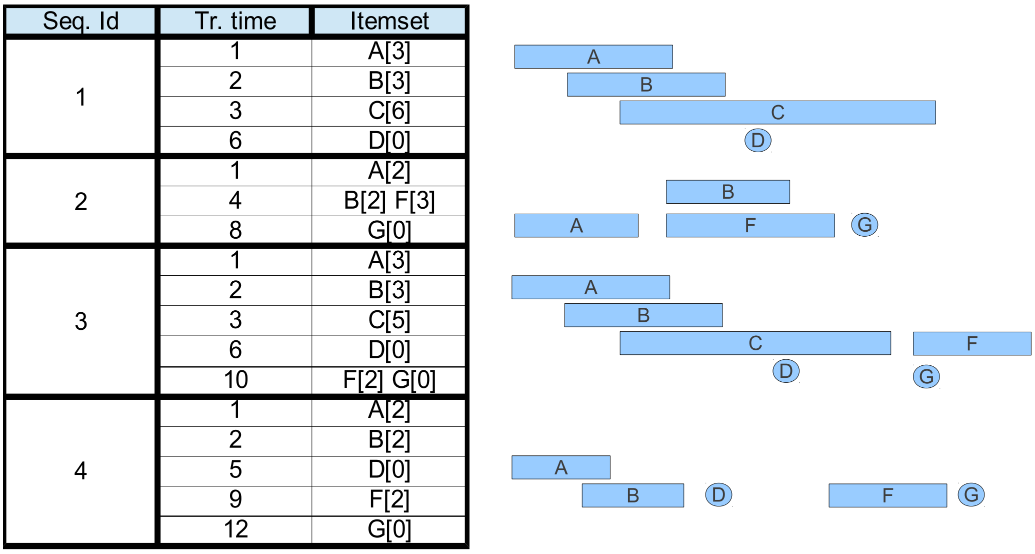
If we are not interested in the timestamp associated with each transaction, such as it occurs in the mining of qualitative patterns, we can simply define a sequence by an ordered set of Itemsets $s = \{I_1, I_2, \ldots, I_n\}$. With this alternative definition each itemset I_j is supposed to appear one or more time units later than I_i, $\forall i < j$. Thus, we can still use this definition to extract patterns whose items are ordered by time. In practice, we use both definitions based in transactions or itemsets, depending on whether we want to include the timestamp associated with the transaction or not, i.e. if we deal with quantitative or qualitative patterns, respectively.

We denote the size of a sequence $|s|$ as the number of transactions (i.e. itemsets) in that sequence. We define the length of a sequence ($l(s) = \sum_{k=1}^{n} |I_k|$) as the number of items in it, and every sequence with k items is called a k-sequence. For instance, the sequence $\alpha = \langle (\langle a, 2 \rangle \langle b, 3 \rangle)(\langle b, 2 \rangle \langle c, 5 \rangle) \rangle$ is a 4-sequence with a size of 2 itemsets.

In the rest of the work, we use the terms pattern and sequence interchangeably.

An input sequence is is a tuple $is = \langle id, s \rangle$ with $id \in \mathbb{N}$ and s is a sequence as we previously commented. We call id the identifier of the input sequence.

A sequence database D is a collection of input sequences $D = \langle s_1 s_2 \ \ldots \ s_n \rangle$, ordered by the identifier of the contained sequences. In Figure 2.1 we can see an input database composed of four sequences and, for instance, the first sequence contains three intervals (A, B and C) and a point (D).



Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	D[0]
2	1	A[2]
	4	B[2] F[3]
	8	G[0]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[0]
	10	F[2] G[0]
4	1	A[2]
	2	B[2]
	5	D[0]
	9	F[2]
	12	G[0]

Figure 2.1: Example of sequential database.

For instance, an input sequence can be a patient (sequence) in a clinical database. Each patient has a limited number of observations (different items appearing in the database), where several observations occurs at the same time, as a medical visit (i.e. itemset). Besides, these observations sets (medical visits) are repeated during the visits of the same patient (number of itemsets per sequence).

Definition 2.2.1. Let i_i and i_j be two events in the sequence α. We say α is a subsequence of another sequence β (or β is a supersequence of the sequence α), denoted as $\alpha \preceq \beta$, if there exists a bijective function f which preserves the order and maps events in α to events in β, in such a way that 1) $\forall i \in \alpha, \forall f(i) \in \beta, i \subseteq f(i)$ and 2) the relation between two events $i_i, i_j \in \alpha$ is maintained by $f(i_i), f(i_j) \in \beta$. We can also say that if $\alpha = \langle T_{a_1} T_{a_2} \dots T_{a_n} \rangle$ and $\beta = \langle T_{b_1} T_{b_2} \dots T_{b_m} \rangle$, there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $T_{a_1} \subseteq T_{b_{j_1}}, T_{a_2} \subseteq T_{b_{j_2}}, \dots, T_{a_n} \subseteq T_{b_{j_n}}$.

Let us define the concept of temporal distance that we will need to work with quantitative patterns. Given two transactions $T_j = (t_j, I_j)$ and $T_k = (t_k, I_k)$, we call temporal distance and denote as "$R[t_{jk}]$" to the difference of time between the beginning point of an item i_k appearing in T_k minus the ending point of an item i_j appearing in T_j. Note that, since i_j appears in T_j, its ending point occurs at time $t_j + i_j.d$, denoting with $i_j.d$ the duration of i_j. Regarding i_k, as it appears in T_k, its beginning point occurs at time t_k, having a $t_{jk} = t_k - (t_j + i_j.d)$ that is associated with a relation. Therefore, we can see that the end points of elements of I_j have different distances with respect to the elements of I_k, being all of them depending on the duration of I_j items. In addition to this definition, we also consider as temporal distances the item durations and we also have them into account in our algorithms.

Definition 2.2.2. The *support* (or *frequency*) of a sequence α, denoted as $\sigma(\alpha, D)$, is the total number of sequences in the input database D that contain α. A pattern or sequence is called *frequent* if it occurs at least a number of times specified by a given user specified threshold *min_sup*, called the minimum support. We define $\mathcal{FS}$ as the whole collection of frequent sequences. The problem of frequent sequence mining is now to find $\mathcal{FS}$ in a given input database, for a given minimum support threshold.

Given two sequences $\alpha = \langle T_1 T_2 \dots T_n \rangle$ and $\beta = \langle T_1' T_2' \dots T_m' \rangle$ with $m < n$, we say that β is a ***prefix*** with respect to α if $\forall 1 \leq i < m, T_i = T_i'$ and $T_m' \preceq T_m$, the events in T_m' are the first ones in T_m. We can also call *m-prefix* to β since is a prefix with a length of m events. In the prior definition, the sequence $\langle (T_m - T_m')\ T_{m+1} ... T_n \rangle$ is called the ***suffix*** of α with regards to prefix β.

For instance, the sequence $\alpha = \langle (\langle a, 2\rangle) < (\langle a, 3\rangle\langle b, 4\rangle\langle c, 5\rangle) < (\langle a, 2\rangle\langle c, 3\rangle) < (\langle b, 3\rangle) < (\langle c, 5\rangle\langle f, 0\rangle)\rangle$, has the suffix $\gamma = \langle (*\langle c, 3\rangle) < (\langle b, 3\rangle) < (\langle c, 5\rangle\langle f, 0\rangle)\rangle$ with regards to the prefix $\beta = (\langle a, 2\rangle) < (\langle a, 3\rangle\langle b, 4\rangle\langle c, 5\rangle) < (\langle a, 2\rangle)$, where the $*$ symbol in γ means that the first itemset in α was not completed by the prefix β.

Definition 2.2.3. A frequent sequence $s \in \mathcal{FS}$ is called *closed* if there not exists another frequent supersequence $s' \in \mathcal{FS}$ with the same support as it. The whole set of frequent closed sequences is denoted by $\mathcal{FCS}$.

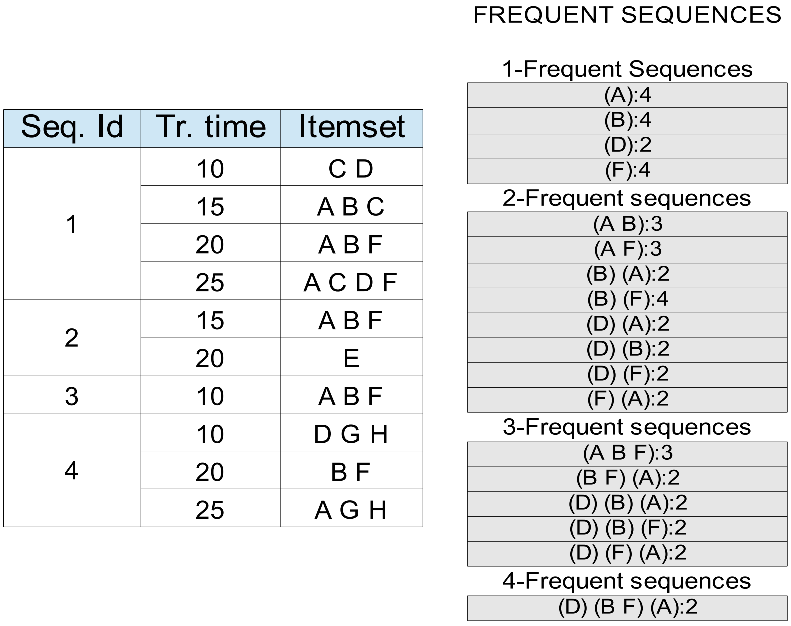
Definition 2.2.4. A frequent sequence $s \in \mathcal{FS}$ is called *maximal* if there not exists another frequent supersequence of it. The whole set of frequent maximal sequences is denoted by $\mathcal{FMS}$.

According to definitions 2.2.3 and 2.2.4, in SDM (as in IDM) we can deduce that $\mathcal{FS}$, $\mathcal{FCS}$ and $\mathcal{FMS}$ are related between them: $\mathcal{FMS} \subseteq \mathcal{FCS} \subseteq \mathcal{FS}$.

2.3 Time Point data

As we previously said, when we work in SDM with patterns where the temporal data model corresponds to time points, the concept of temporal order can be described by means of the qualitative relations $R_p = \{before, equals, after\}$ (usually denoted as $R_p = \{<, =, >\}$) defined by Vilain [Vilain, 1982]. According to the previous Section and due to the fact that each item has a duration, in the point-based case every item has a duration of zero time units. From later on, for the point data case, we use the terms point and event interchangeable. Furthermore, since the "$after$" operator is the inverse of the "$before$" relation, if we always consider a relation from the point which occurs the first, we do not need to use the "$after$" relation. For instance, if we have $A > B$ we will say instead $B < A$. So, with these two relations we can define patterns or sequences. Two patterns α and β are exactly equal if their points are exactly the same and the have exactly the same relations in the same positions, i.e. $\alpha \preceq \beta$ and $\beta \preceq \alpha$.

In Figure 2.2 we can see an example of a point-based sequential database, taken from [Zaki, 2001], which is composed of four sequences that will be used for introducing the algorithms in this Section.



Seq. Id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

FREQUENT SEQUENCES

1-Frequent Sequences

(A):4
(B):4
(D):2
(F):4

2-Frequent sequences

(A B):3
(A F):3
(B) (A):2
(B) (F):4
(D) (A):2
(D) (B):2
(D) (F):2
(F) (A):2

3-Frequent sequences

(A B F):3
(B F) (A):2
(D) (B) (A):2
(D) (B) (F):2
(D) (F) (A):2

4-Frequent sequences

(D) (B F) (A):2

Figure 2.2: Example of a point-based sequential database.

In addition to the previous explanations for point based patterns, when we are interested in quantitative patterns we have to express the temporal distance somehow. Thus, we still use the same two relations $<$ and $=$, but now we add a temporal distance between square brackets, having $\{< [t], =\}$ as the two relations used in quantitative patterns. The t value denotes any possible time units value, derived from the input database. Note that the *equals* relation does not need a temporal distance associated with it since the equality relation has an implicit distance of 0 time units between the two events that relates. With this new definition of relations, two relations are equal if they express the same relation and share the same temporal distance. Therefore, we say that two point based patterns are equal if and only if all their items, their relations, and their distances are exactly the same. For instance, pattern $\alpha = A < [2]B$ is different from the pattern $\beta = A < [3]B$ since although both patterns share the same relation ($<$), they have different temporal distances.

2.3.1 Algorithms for mining qualitative patterns

The SDM started to be studied in 1995 [Agrawal and Srikant, 1995] and since then we can see that is a challenging task. As it is shown in [Zaki, 2001], on the set of all sequences on the items (or events with duration zero) we can define a hyper-lattice with the subsequence relation $\preceq$. In that hyper-lattice the join operation is the set of minimal common supersequences, while the meet operation corresponds to the set of maximal common subsequences. Unlike in the IDM task [Goethals, 2003], where the search space is the power set of items, in SDM the search space is potentially infinite since we can compose infinite long sequences by means of the relation $before$. Fortunately, in all practical cases we can establish an upper-bound by the expression $2^{k-1}n^k$, where n is the number of different items and k is the length of the longest sequence in the input database [Zaki, 2001]. Therefore the complexity is $O(n^k)$.

Since we are only interested in the frequent sequences, in the expression above the n value can be referred only to the frequent items, while k is the length of the longest frequent subsequence contained by a min_sup of sequences in the in the input database. In fact, we can represent the set of all frequent sequences in a meet-semilattice, closed with the meet operation, i.e., if α and β are frequent sequences, then the maximal common subsequence is also frequent. Note that it is not the case of supersequences, hence we say that it is a meet-semilattice [Zaki, 2001]. In Figure 2.3 we can see the lattice induced by the frequent sequences of database from Figure 2.2.

From the above mentioned meet-semilattice, the closure with the meet operation leads to the main property introduced by Agrawal and Srikant [Agrawal and Srikant, 1995] called monotonicity property:

Proposition 1. *All subsequences of a frequent sequence s are also frequent. Besides, the support of every subsequence is greater or equal than the support of s.*

Proof. Let α be a subsequence of the frequent sequence s such that $\alpha \preceq s$. If s is a frequent sequence and D is the input database then $\sigma(s, D) \geq min\_sup$. Since α is subsequence of s, the support of α is at least the same as s, since α appears wherever s appears, and then α is also frequent. Besides, s can appear in any other input sequences, hence $\sigma(\alpha, D) \geq \sigma(s, D) \geq min\_sup$. □

Corollary 1. *All supersequences of an infrequent sequence s are also infrequent. Besides, the support of every supersequence is less or equal than the support of s.*

Proof. Let α be a frequent supersequence of the infrequent sequence s such that $s \preceq \alpha$. If α is a frequent sequence, since proposition 1 all its subsequences are frequent. Concretely,

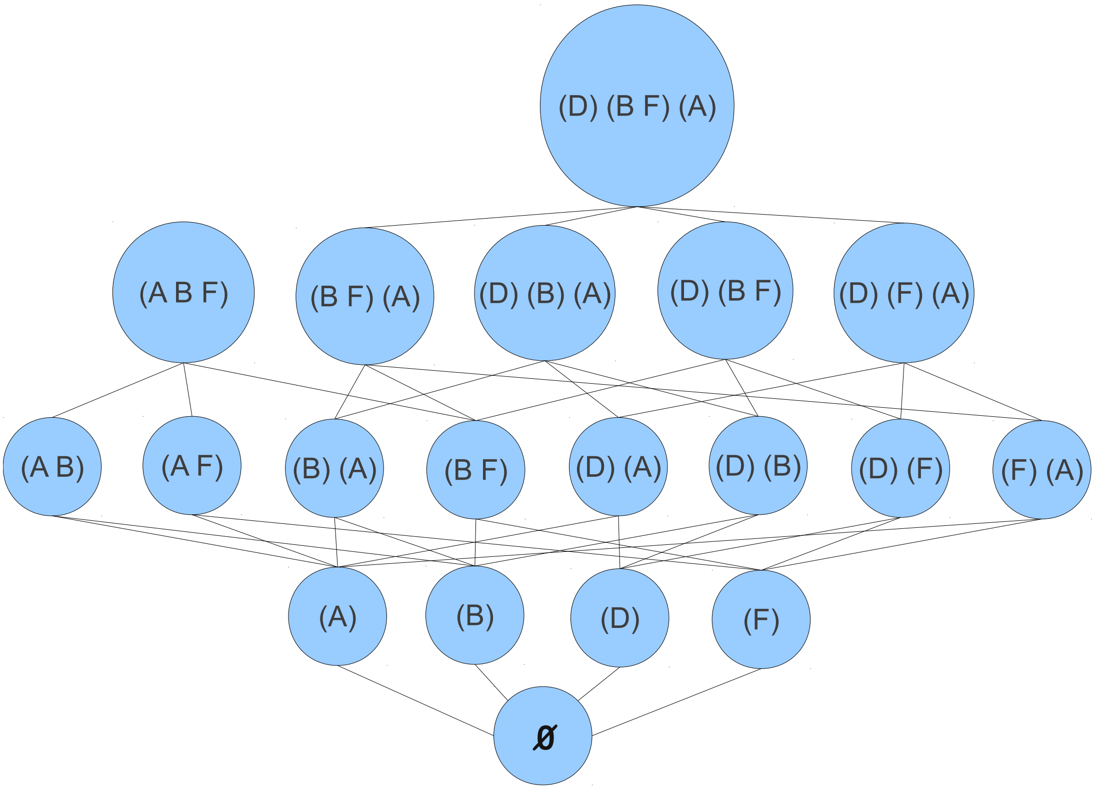


Figure 2.3: Lattice of the example database.

if s is a subsequence of α, s should be frequent, but s is not frequent and we have a contradiction. Therefore, all of the supersequences of s are infrequent and hence, their supports are less or equal than min_sup. □

The above property is the basis for the different strategies. Using this property, we can create candidate sequences as supersequences from frequent sequences, and somehow to determine if those new candidate sequences are frequent. There exist several ways to achieve this and most of them make a new candidate k-sequence from two frequent $(k-1)$-sequences.

So far, three different strategies have been mainly developed to obtain the whole set of frequent sequences ($\mathcal{FS}$). We enumerate and describe them in a briefly way, and in the following Subsections we describe them thoroughly.

Apriori-like [Agrawal and Srikant, 1995]. This strategy is based on a breadth-first search (BFS) for exploiting the monotonicity property. The algorithms consist of a continuous loop that does 1) a candidate generation followed by 2) a support checking phase for each generated candidate. We highlight especially the GSP algorithm [Srikant and Agrawal, 1996].

Vertical format database [Zaki, 2001]. This second strategy was created by Zaki and is composed by algorithms that keep in memory only the information required to calculate the support in a sequence by using a database in vertical format. The first and most representative algorithm that belongs to this strategy is SPADE [Zaki, 2001]. In this algorithm, both breadth-first and depth-first search (DFS) can be used.

Pattern growth approach [Han et al., 2000]. This last strategy was introduced by Han et al. and it is consist of algorithms that obtain the whole $\mathcal{FS}$ without making a candidate generation phase. They use a DFS and a technique called "database projection" based on a *divide and conquer* strategy to achieve the final results. The most representative algorithm in this group is PrefixSpan [Pei et al., 2004].

Apriori-like

This strategy was directly inherited from IDM [Agrawal et al., 1993], and concretely, the algorithms AprioriAll and Apriori-Some were the first algorithms to face the SDM problem [Agrawal and Srikant, 1995]. The same authors published in 1996 an improved algorithm called GSP (Generally Sequential Patterns) [Srikant and Agrawal, 1996], which also searched for patterns with other extra constraints as temporal windows and minimum and maximum gaps.

In this survey, we choose the GSP algorithm to summarize the phases executed by the Apriori strategy. GSP, as every Apriori-based algorithm, carries out a BFS. The algorithm starts obtaining the frequent 1-sequences ($\mathcal{F}_1$) while it reads the input database. From $\mathcal{F}_1$, the candidate 2-sequences ($\mathcal{C}_2$) are created by combining 1-sequences. Then, the algorithm counts the support of each candidate 2-sequence, and removes every infrequent candidate sequence. These steps are recursively repeated until there are no more candidates or none of the is frequent.

In more detail, for a k-step the two main phases in GSP are:

1. **Candidate generation.** This phase is mainly based on the monotonicity property and consists of two steps:

   (a) *Generation of the set of candidate k-sequences.* Given a set of frequent (k-1)-sequences $\mathcal{F}_{k-1}$, the next candidate level $\mathcal{C}_k$ is obtained by joining elements in $\mathcal{F}_{k-1}$. Two sequences α_1 and α_2 in $\mathcal{F}_{k-1}$ will generate a new element in $\mathcal{C}_k$ if dropping the first item i_{α_1} in α_1 and the last item i_{α_2} in α_2, the remaining parts α'_1 and α'_2 are exactly the same sequence ($\alpha'_1 \preceq \alpha'_2$ and $\alpha'_2 \preceq \alpha'_1$).

   (b) *Pruning phase.* After obtaining $\mathcal{C}_k$, the algorithm executes a pruning step to try to reduce the number of elements of $\mathcal{C}_k$. Considering the monotonicity property, for each frequent element β in $\mathcal{C}_k$ all its $k-1$ $(k-1)$-subsequence must also be frequent. Therefore, if we can find a $(k-1)$-subsequence which is not in $\mathcal{F}_{k-1}$ then it can be safely removed because it cannot be frequent.

2. **Counting the candidate support.** Once $\mathcal{C}_k$ has been generated, we need to check what patterns are really frequent. To this end, GSP goes through the entire input database. So, for each sequence s in the database GSP checks what candidates are subsequences of s and increases the candidate counter. In the end, we keep in $\mathcal{C}_k$ only the frequent elements and prune the infrequent ones, or in short, we get $\mathcal{F}_k$.

After GSP, other several Apriori-based algorithms [Masseglia et al., 1998; Antunes and Oliveira, 2004; Guil et al., 2004] were proposed to speed up the execution time. Most of them tried to get a smaller $\mathcal{C}_k$ in the candidate generation phase, or, especially, to accelerate the counting of the support by means of tree structures or making the database scan in a different way. Despite of these improvements, Apriori-based algorithms have several inherent drawbacks that are independent of any optimization techniques:

- It performs multiple scans of the database. An Apriori-based algorithm has to repeat both candidate generation and counting of the support steps exactly k times, where k is the length of the biggest frequent sequence. So, if the algorithm works with a low support or deals with databases where, on average, the sequences have a lot of different transactions, that k number will be higher.

- It needs to do a BFS. To find the smallest $\mathcal{C}_k$ and to avoid repeating database scans, the algorithm needs to keep in the memory the whole k-level of candidates. With some databases it is not possible to keep in the memory the whole $\mathcal{C}_k$.

- It generates many useless candidates. Even though the algorithm executes a pruning step in the candidate generation phase, the pruned $\mathcal{C}_k$ contains a lot of sequences which will not be frequent. Therefore, their generation is time-consuming (when the supports are calculated) and, at the end, they are removed from $\mathcal{C}_k$ to $\mathcal{F}_k$. Ideally, the algorithm should only generate the candidates which will be frequent.

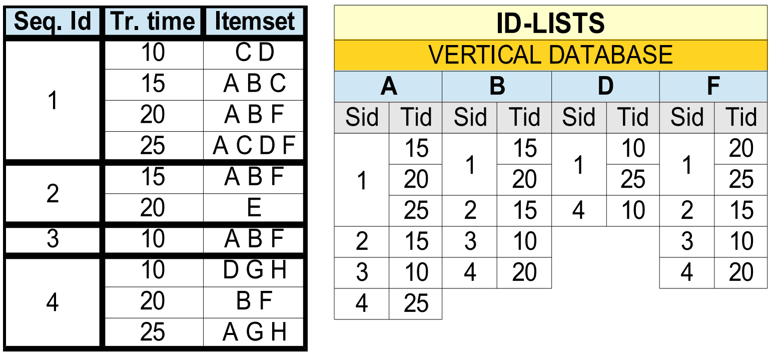
Vertical Database Format

This strategy emerged with the SPADE algorithm [Zaki, 2001] by M. Zaki. The main idea in this new strategy had been previously introduced by the Eclat algorithm [Zaki, 2000a] for IDM. In order to overcome the main drawbacks in the Apriori-based algorithms, Zaki proposed a strategy that uses a vertical input database, that could be executed both with a DFS and BFS, and that was capable of obtaining the whole $\mathcal{FS}$ without making several scans of the input database. Besides, the algorithm can partition the search space in different pieces and solve them separately. This algorithm was the seed for the Vertical Database Format strategy, and after it, some other algorithms were proposed [Leleu et al., 2003; Gouda and Hassaan, 2011; Ayres et al., 2002; Gouda et al., 2010].

Since that all of algorithms for this strategy have a similar behaviour, we focus in the SPADE algorithm. SPADE uses an input database with a vertical format, where each item is associated with the sequences and transactions where it appears (see Figure 2.4). From these items with that vertical format, SPADE creates supersequences, all of them also with vertical format. This new format has several advantages. On the one hand, it makes possible to easily count the support of each sequence, and on the other hand, it allows an efficient way to create a common supersequence from two candidate frequent sequences.

If we go deeper into the main points in SPADE we can highlight:

Decomposition of the original lattice. Zaki defined an equivalence relation between sequences based on the idea of the above mentioned meet-semilattice (see Section



Seq. Id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

ID-LISTS							
VERTICAL DATABASE							
A		B		D		F	
Sid	Tid	Sid	Tid	Sid	Tid	Sid	Tid
1	15	1	15	1	10	1	20
	20		20		25		25
	25	2	15	4	10	2	15
2	15	3	10			3	10
3	10	4	20			4	20
4	25						

Figure 2.4: IdLists for the frequent items in the example database.

2.2). These equivalence relations are based on sequence prefixes, so that, two sequences α and β belong to the same equivalence class $[\gamma]$ if and only if γ is a prefix of α and γ is also a prefix of β. With this definition, each equivalence class defines a sub-lattice that can be processed independently, and hence, if SPADE works with huge databases, it can partition the original search space into smaller pieces. According to the example, in Figure 2.5 we can see a decomposition of the original lattice in four smaller sub-lattices (induced by the equivalence classes $[A], [B], [C]$ and $[D]$). In Figure 2.6 the last equivalence class $[D]$ is also partitioned into three equivalence classes ($[(D)(A)]$, $[(D)(B)]$ and $[(D)(F)]$).

IdLists. Each pattern p is associated with a list (called *IdList* and denoted by $\mathcal{L}(p)$) where each entry is composed of a pair (Sid, Tid). The IdList contains all the input sequence (Sid) and transaction (Tid) identifiers where p appears. By means of $\mathcal{L}(p)$, the most important tasks in the algorithm can be easily performed:

1. Candidate generation. Also based in the monotonicity property, if two frequent k-sequences α and β share the same $(k-1)$-prefix γ (therefore they belong to the equivalence class $[\gamma]$), they generate candidate $(k+1)$-sequences by means of temporal joins in their IdLists. SPADE distinguishes between two types of temporal joins:

   (a) Itemset extension. It corresponds to the temporal join between two frequent k-sequences $\alpha = (\langle p \rangle i_i)$ and $\beta = (\langle p \rangle i_j)$ in $[p]$, where $\langle p \rangle$ is a k-sequence and i_i, i_j are items in I, with $i_i \neq i_j$, such that it generates a candidate which is the result of adding the last item in β, (i_j), to the last transaction of α. The resulting IdList $\mathcal{L}((\langle p \rangle i_i i_j))$ is composed by the common entries in $\mathcal{L}((\langle p \rangle i_i))$ and $\mathcal{L}((\langle p \rangle i_j))$.

   (b) Sequence extension. It occurs when two patterns $\alpha = (\langle p \rangle (i_i))$ and $\beta = (\langle p \rangle (i_j))$ ($\alpha, \beta \in [p]$) generate a pattern $\gamma = \langle p \rangle (i_i)(i_j)$ that is the result of including the last item of β (i_j) in a new transaction after those transactions in α. The resulting IdList $\mathcal{L}(\langle p \rangle (i_i)(i_j))$ consists of, for each Sid, the entries in β whose Tid is higher than any of those in α.

   Zaki describes three different possibilities to generate candidate $(k+1)$-sequences from two frequent k-sequence:

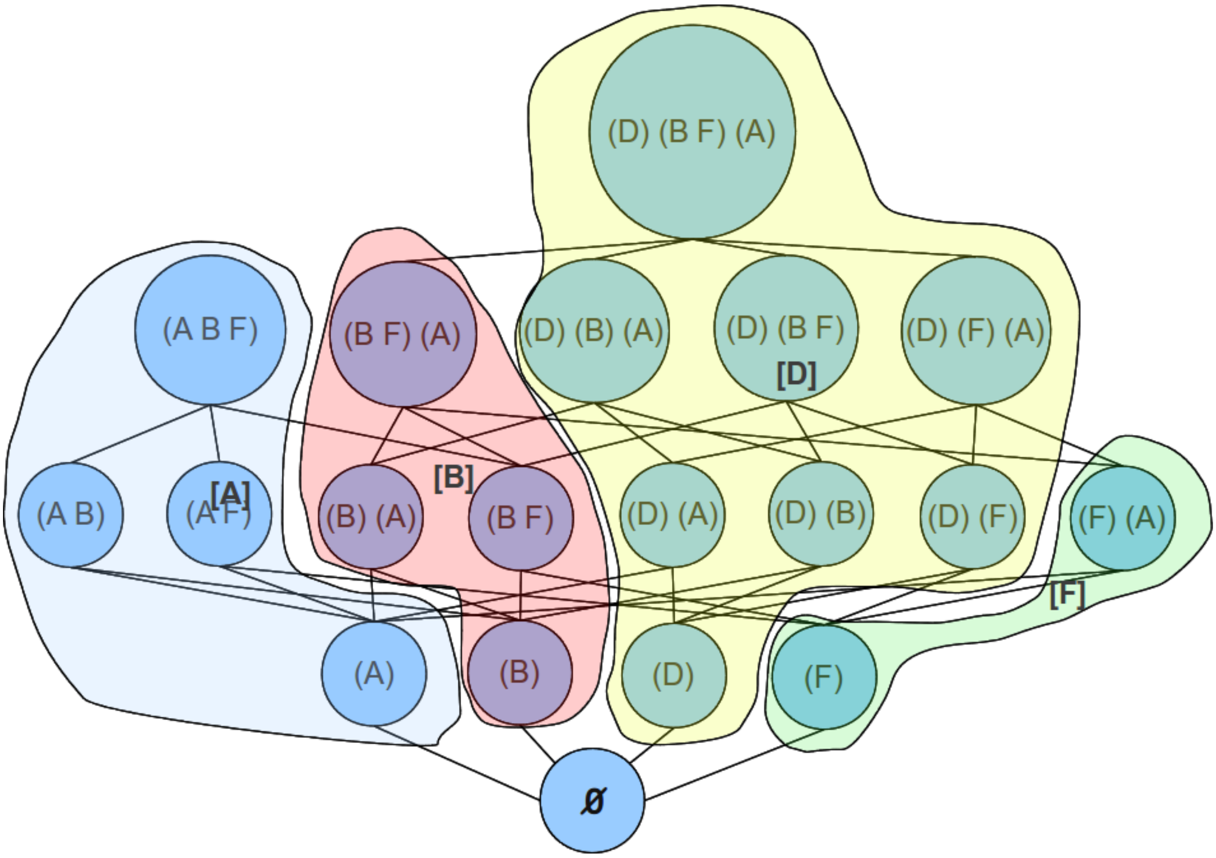


Figure 2.5: Division of example into four equivalence classes.

- Join of itemset extension with itemset extension: they generate another candidate itemset extension, e.g. (AB) and (AC) generate the candidate (ABC).
- Join of itemset extension with sequence extension (or vice versa): they generate a candidate sequence extension, e.g. $(D)(BF)$ and $(D)(B)(A)$ generate the candidate $(D)(BF)(A)$.
- Join of sequence extension with sequence extension: if both sequences are just the same, the result is another sequence extension, e.g. $(F)(A)$ with itself produces the candidate $(F)(A)(A)$. Otherwise both frequent k-sequences produce two sequence extension and one itemset extension, e.g. $(D)(A)$ and $(D)(B)$ generate the candidate sequence extensions $(D)(A)(B)$ and $(D)(B)(A)$ and the itemset extension $(D)(AB)$.

2. Support counting. For each $\mathcal{L}(p)$, its associated support is the count of different Sid values contained in it (see Figure 2.4).

Besides we can also do a pruning step to remove those candidates that contain infrequent subsequences to avoid temporal joins in candidate sequences potentially infrequent.

Different search strategies. Since the support counting can be done by means of IdLists and the original lattice can be divided into sub-lattices, it is possible to

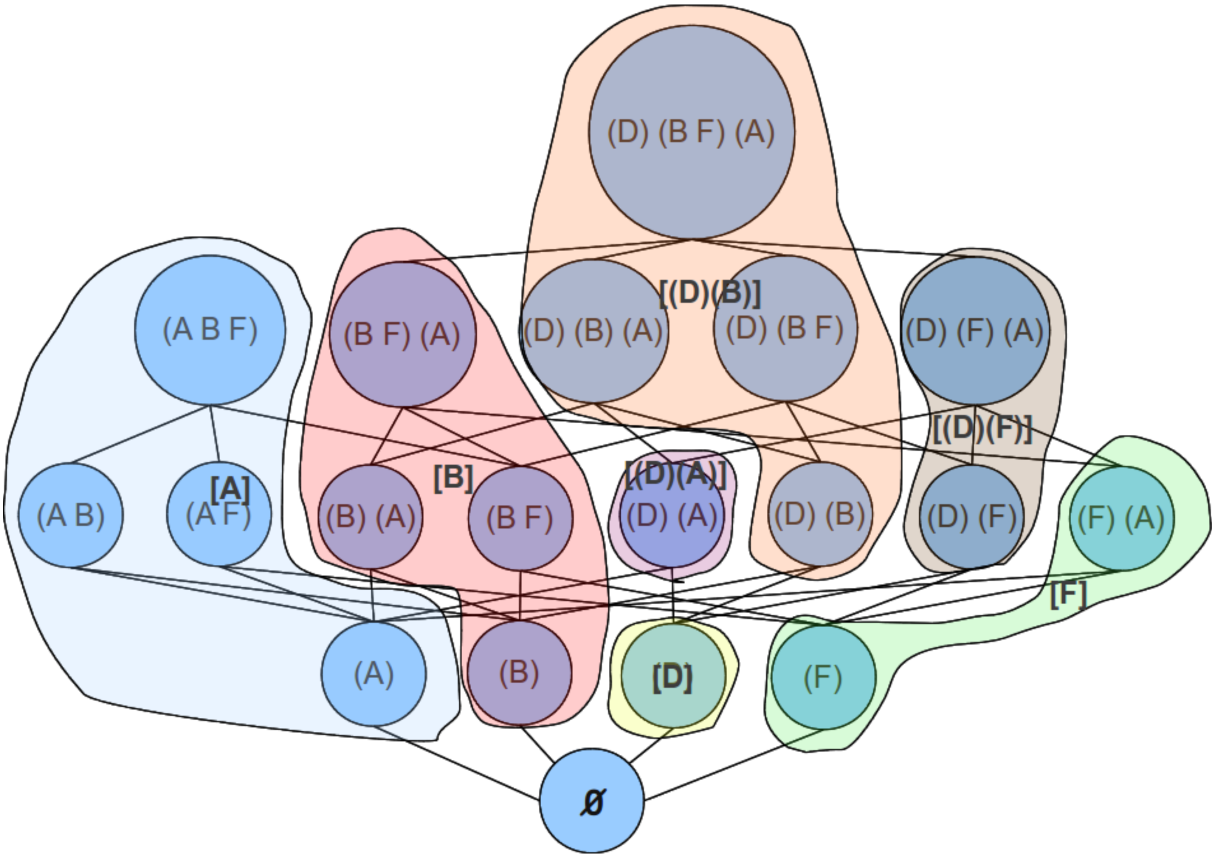


Figure 2.6: Division of example into four equivalence classes. Besides, the equivalence class [D] is also divided into three equivalence classes.

use both a BFS and DFS. The advantage of BFS over DFS is that it has more information available for pruning. On the other hand DFS requires less main-memory than BFS. While BFS needs to keep all of the equivalence classes in two consecutive levels, DFS needs to keep only the intermediate IdLists for two consecutive equivalence classes.

In [Zaki, 2001], Zaki recommends to run SPADE with DFS and without pruning. The main motivation is due to the simplicity of the temporal joins in the IdList, and because searching for an infrequent sequence in the $(k-1)$-subsequences in the pruning step takes longer than doing the temporal join directly.

Since most of the time of the algorithm is spent in temporal joins, several algorithms were proposed to speed up the temporal join operations. Among them, D-SPADE [Gouda and Hassaan, 2011], SPAM [Ayres et al., 2002] and Prism [Gouda et al., 2010], are the most important ones. Without going into detail with any of them, we summarize how these algorithms improve SPADE:

1. D-SPADE [Gouda and Hassaan, 2011]: is an algorithm especially useful with dense databases. Since SPADE IdLists have the time occurrence for every sequence and in a dense database every item appears in almost all transactions of each sequence, the IdList representation is highly inadequate since it takes a lot of memory. Therefore, in this kind of database it is better if we register the no occurrence time for

a sequence. In this way, the IdList representation is much smaller since the no occurrences are no frequent.

2. SPAM [Ayres et al., 2002]: this algorithm proposes a new IdList representation to speed up the temporal joins. To achieve this, the authors represent IdLists by means of bitmaps and the temporal joins operations are reduced to basic logic operations. The main drawback for this approach is that it requires more memory than SPADE to allow very fast temporal join operations.

3. Prism [Gouda et al., 2010]: this algorithm also tries to speed up the temporal joins by means of basic operations. In this case, they propose an innovative "primal block encoding" by using the first n prime numbers to represent the IdLists and doing the greatest common divisor operations successive times (whose results are kept in a lookup table). Unlike SPAM, Prism takes much less memory than SPADE and its execution is even faster.

The Vertical Database Format algorithms obtain a low execution time and good memory allocation. The only drawback that remains in each different implementation of this strategy is the candidate generation. While the counting of support is done quickly, they generate a large number of candidates that are finally discarded. With huge databases this candidate generation can be time-consuming.

Pattern Growth

In 2002 Han et al. published the first version of the PrefixSpan algorithm [Pei et al., 2004]. This algorithm was an evolution of FreeSpan [Han et al., 2000], which came from the idea of the FP-Growth algorithm [Han and Pei, 2000] in IDM. This strategy is based on dividing the database into pieces according to prefixes. Thus, applying the divide and conquer principle, they avoid generating candidates and the algorithm finds the frequent sequences directly in the successive prefix-based partitions (also known as projections) of the database.

In this survey we summarize how this strategy works through PrefixSpan algorithm that consists of three steps:

1. To find the frequent items or 1-sequences in the original database D.

2. To divide the search space into different projected databases according to the number of frequent items. Each partition D_α is the projection of the original database D with the prefix α, being each projection the suffix in sequence $s \in D$ that is associated with the prefix α.

3. Recursively with a DFS, the algorithm finds the whole set of frequent items in D_α. For each frequent item i in a projected database D_α, another projected database D_β is obtained, being the new prefix β the concatenation between the previous one α and the frequent item i, $\beta = \alpha \cdot i$.

The best point for the Pattern-growth strategy is that it obtains the whole set of frequent sequences without generating candidates. Furthermore, with a depth-first search and due to the fact that the projected databases keep shrinking, PrefixSpan has a low use of memory. However, since the major cost is the construction of projected databases, when databases have a high number of transactions by sequence the same item can be projected many times in the same sequence. This phenomena leads to a great number of projections, and therefore a big memory allocation, and a slower execution. In order to avoid this drawback, in the creation of database projections, instead of making physical projection in the memory, the authors proposed to use pointers to refer to the sequence projections. These projections are referred as *pseudo-projections*. Therefore, a projected database is a set of pointers to sequences and each pointer refers to an item of a sequence, denoting the offset of it in the corresponding projected sequence. In Figure 2.7 we can see an example of pseudo-projections for several projected database.

The main advantage for this strategy is the efficient management of the memory when it works with pseudo-projections. Another advantage is that this strategy is a good starting point for finding other possible patterns with special properties, as we will see later. After PrefixSpan, other algorithms such as MEMISP [Lin and Lee, 2005] or LAPIN [Yang et al., 2007] were proposed. While the first tried to make optimized pseudo-projections, the second algorithm used specific characteristics of the sequential databases to obtain a better runtime.

2.3.2 Algorithms for mining quantitative patterns

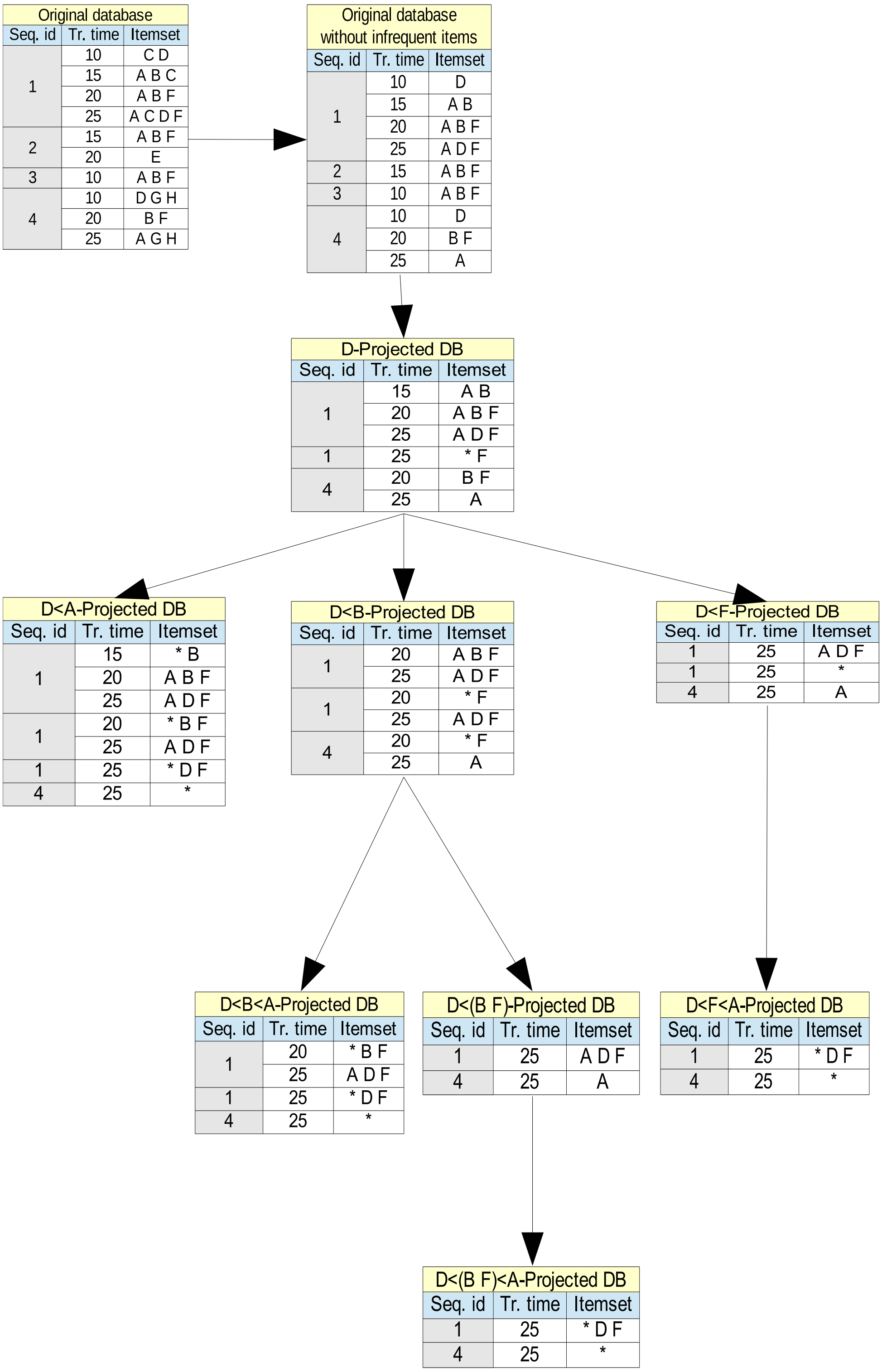
All the issues discussed in the previous subsection remain when we are interested in mining quantitative patterns. Unfortunately, since there can potentially be infinite relations as a quantitative relation has a temporal distance associated, and the domain of that distance is infinite. This fact makes both the hyperlattice and the meet-semilattice far more complex since we have many more relations. Fortunately, in practice the temporal distance associated with relations is bounded by the structure of the database and the longest distance between the first and the last transaction in a sequence.

However, all the main definitions and properties present in qualitative pattern mining also appear in quantitative mining. Therefore, the same strategies can be applied to this mining, but the subsequence checking and the candidate generation steps have to be modified. Furthermore, the mining of the frequent 2-patterns become now crucial in some strategies since we need to know all the different temporal distances that can extend the different frequent patterns in order to get largest frequent supersequences.

In regards to algorithms for mining quantitative patterns, there only a few that consider them. These algorithms extend sequential patterns to capture durations between elements in the sequences. Besides, most of them are based on Pattern Growth strategy and little effort has been made in developing algorithms based on Vertical Database Format strategy.

In this Subsection we give a brief summary of some of the previous works for mining quantitative point based patterns. These algorithms are:

[Yoshida et al., 2000]. Yoshida et al. propose a notion of temporal sequential called delta pattern, and integrate sequences with temporal constraints in the form of bounding intervals. Delta Patterns have the form $A \xrightarrow{[t_i,t_j]} B$, denoting a sequential pattern $A \rightarrow B$ that frequently occurs in the data set with transition times from A



Original database

Seq. id	Tr. time	Itemset
1	10	C D
	15	A B C
	20	A B F
	25	A C D F
2	15	A B F
	20	E
3	10	A B F
4	10	D G H
	20	B F
	25	A G H

Original database without infrequent items

Seq. id	Tr. time	Itemset
1	10	D
	15	A B
	20	A B F
	25	A D F
2	15	A B F
3	10	A B F
4	10	D
	20	B F
	25	A

D-Projected DB

Seq. id	Tr. time	Itemset
1	15	A B
	20	A B F
	25	A D F
1	25	* F
4	20	B F
	25	A

D<A-Projected DB

Seq. id	Tr. time	Itemset
1	15	* B
	20	A B F
	25	A D F
1	20	* B F
	25	A D F
1	25	* D F
4	25	*

D<B-Projected DB

Seq. id	Tr. time	Itemset
1	20	A B F
	25	A D F
1	20	* F
	25	A D F
4	20	* F
	25	A

D<F-Projected DB

Seq. id	Tr. time	Itemset
1	25	A D F
1	25	*
4	25	A

D<B<A-Projected DB

Seq. id	Tr. time	Itemset
1	20	* B F
	25	A D F
1	25	* D F
4	25	*

D<(B F)-Projected DB

Seq. id	Tr. time	Itemset
1	25	A D F
4	25	A

D<F<A-Projected DB

Seq. id	Tr. time	Itemset
1	25	* D F
4	25	*

D<(B F)<A-Projected DB

Seq. id	Tr. time	Itemset
1	25	* D F
4	25	*

Figure 2.7: Pseudo projections in PrefixSpan for the example database.

to B that are contained in $[t_i, t_j], \forall i \leq j$. All Delta patterns are found by means of an Apriori-like algorithm that makes a clustering step in every k level.

I-Apriori and I-PrefixSpan [Chen et al., 2003]. These algorithms are both the natural extensions from the native algorithms Apriori and PrefixSpan. Now, both algorithms change their ways of generating candidates. The problem of this work is that both algorithms do not process the temporal distances and give to the user the responsibility of providing those temporal distances.

MisTA [Giannotti et al., 2006]. The authors propose to extend the sequence mining paradigm to linear temporally-annotated sequences by making an annotation of a duration t, in each transition $A \rightarrow B$ of every sequence. The resulting patterns are found through a clustering step and are denoted by $A \xrightarrow{t} B$. They use the Pattern Growth strategy, and the algorithm uses a threshold in order to know if two sequences are closed enough in temporal distance when they execute a subsequence checking operation. In every projection made by the algorithm, a density based clustering is executed in order to get the proper temporal distances.

QPrefixSpan[Nakagaito et al., 2009]. Nakagaito et al. proposed an algorithm derived from the qualitative interval algorithm PrefixSpan, based on Pattern Growth Strategy. The main contribution is that, in every extension of a pattern, it executes a density-based numerical clustering in order to know the range of time in the quantitative relations. However, this is an important overhead that emphasizes the problems that normally appear in Pattern Growth algorithms.

As we can see, all the highlighted algorithms suffers from additional problems to those that typically appears in the different strategies, which lead us to more complex algorithms and slower executions.

2.3.3 Comparatives

When we compare the different strategies, in general terms, we can say both Vertical Database Format and Pattern Growth strategies outperform Apriori strategy. This is mainly due to the drawbacks appearing in that strategy, especially its necessity of making several database scans.

The time execution of both strategies varies depending on the properties of the database where the algorithms are applied. Pattern Growth shows good performance and scales well in memory, especially with sparse databases or when databases mainly consist of small itemsets, whereas when we deal with large dense databases that have large itemsets, the performance of Vertical Database Format algorithms is better. In order to show this last statement, we define the different properties that a sequential database can have and that influence the algorithms execution. These different properties are: The number of different items that exist in the whole database (N), the number of items per itemset (T), the number of itemsets per sequence (C), and the number of sequences in the database (D). We can see the correspondences between abbreviations and meanings at the Table 2.1. We define the database density as the quotient $\delta = \frac{T}{N}$. The higher δ value the more dense is the database.

We have used the well-known data generator provided by IBM to run SPADE, a Vertical Database Format algorithm, and PrefixSpan, a Pattern Growth one, with different configurations. In Figures 2.8 and 2.9 we can observe the behaviour of both SPADE and

Abbr.	Meaning
D	Number of sequences (in 000s)
C	Average itemset in a sequence
T	Average items in an itemset
N	Number of different items (in 000s)
S	Average itemsets in maximal sequences
I	Average items in maximal sequences

Table 2.1: Parameters for IBM Quest data generator.

PrefixSpan when we vary the density and the number of itemsets. In Figure 2.8 we show the running time of the algorithms with a different number of items (100, 500, 1000 and 2500 items) with a constant $T = 20$ value. Since for a database, the density grows either if the numerator increases or the denominator decreases, the Figures have been obtained by just varying the denominator. Besides, Figure 2.9 depicts the behaviour of the algorithms when the number of itemsets changes between values of $C \in \{10, 20, 40, 80\}$ while we keep the density constant ($\delta = \frac{20}{2500}$). We can see that PrefixSpan shows good results when both density and the number of itemsets are low, but when a database is denser and C parameter grows, we notice how SPADE outperforms PrefixSpan.

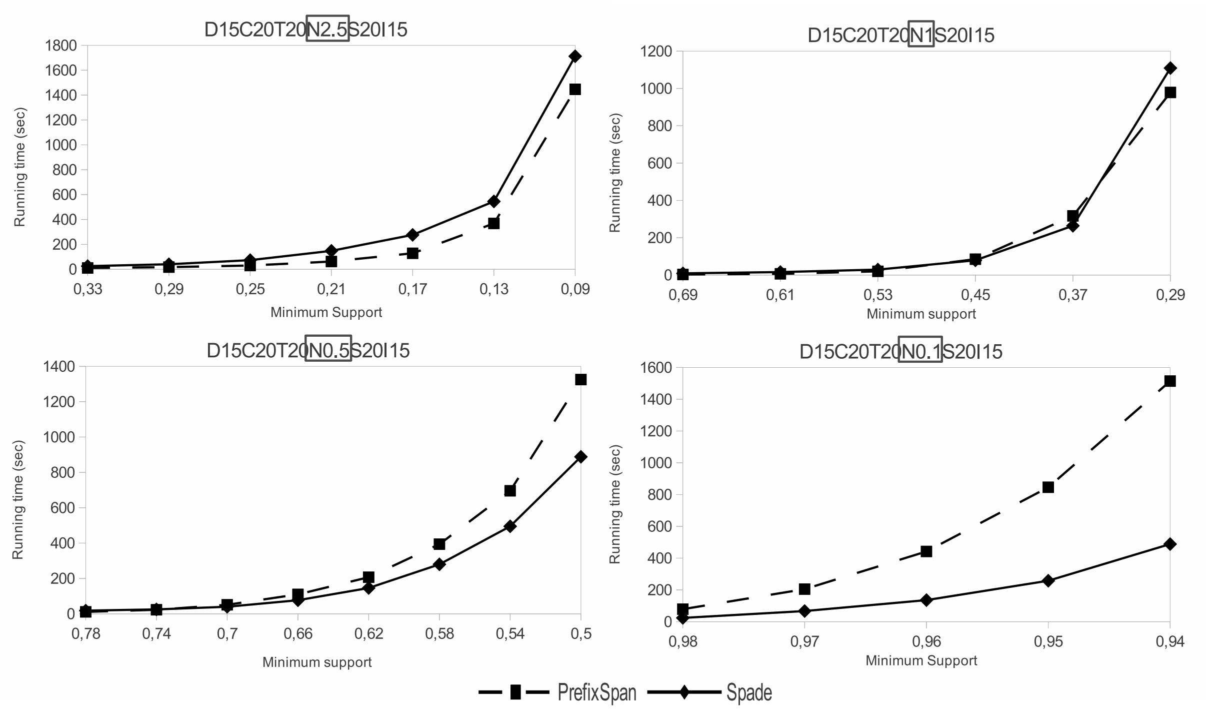


Figure 2.8: Scalability of the algorithms when the number of items increases.

The performed tests show that Vertical Database Format strategy behaves better for general purposes. However, if we deal with sparse database with short itemsets, we can use Pattern Growth algorithms instead.

The breadth-first search method of Apriori-like algorithms implies the necessity of keeping on memory the whole candidate level, and this makes Apriori-like algorithms the worst election, regarding the memory consumption. If instead, we consider Vertical

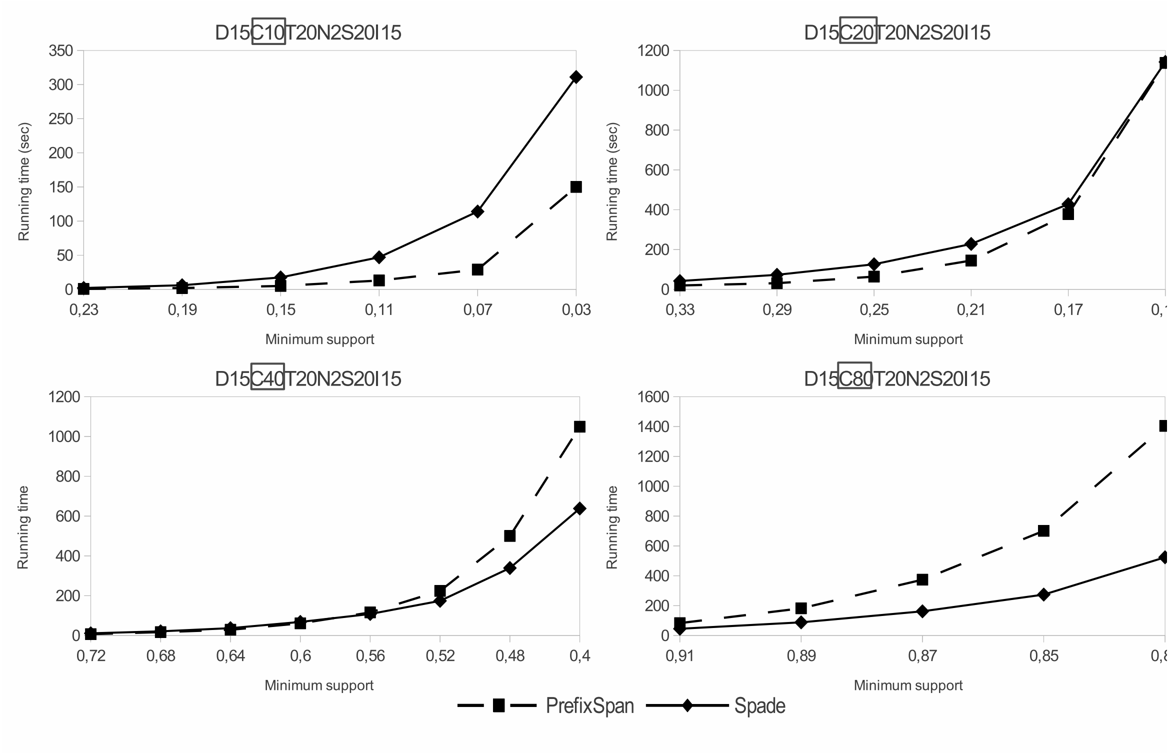


Figure 2.9: Scalability of the algorithms when we vary the C value.

Database Format algorithms executed in a depth-first way, the behaviour is much better but the algorithm still needs to keep on memory all the information associated with IdLists. However, Pattern Growth algorithms do not need to generate candidates (with the pseudo-projection optimization) and they do not need to maintain on memory any kind of information about patterns, what produces the better results in terms of memory consumption.

2.3.4 Reduction of patterns

A typical problem in SDM algorithms is that when the support is decreased the number of frequent patterns increases sharply. Especially when we work with big databases and with very low supports, this problem becomes extremely limiting. This phenomena is known as the *pattern explosion* problem and leads us to two main problems: problems with memory and a long time generating too many meaningless patterns. Regarding the first problem, when the algorithm is executed with a very low support the runtime is longer and much more memory is needed, and even sometimes it is impossible to complete the algorithm execution due to the memory overflow. Besides, in regards to the second problem, even when we can obtain results from the database with very low supports, it is too difficult to make sense of these frequent patterns. Furthermore, there exist too many meaningless and redundant patterns.

In order to solve these problems, several approaches have been considered. On the one hand, a possibility is to search for patterns with some constraints. For instance, in some contexts if an itemset appears too close or too far from another itemset, the resulting pattern may not make sense. To avoid the search for these patterns some algorithms, like GSP [Srikant and Agrawal, 1996] or SPADE (CSPADE) [Zaki, 2000b], allow the option of establishing a minimum and maximum gap in the patterns. Since there are more constraints, the algorithms generate shorter patterns and usually less patterns.

On the other hand, another approach consists of searching for patterns with additional properties. In IDM several approaches have been studied [Bayardo and Agrawal, 1999; Pasquier et al., 1999; Calders and Goethals, 2007; Boulicaut and Jeudy, 2001], being the closed and maximal patterns the most widely used (see definitions 2.2.3 and 2.2.4). The search for this kind of patterns has two benefits at the same time: a reduction of the number of candidates, and a more compact output.

In IDM several algorithms like CHARM [Zaki and Hsiao, 2002], CLOSET [Pei et al., 2000], DCI [Lucchese et al., 2004] have been proposed to find the closed itemsets, while other algorithms like GenMax [Gouda and Zaki, 2005] find the whole set of maximal itemsets. Despite the fact that $\mathcal{FMS}$ is smaller than both $\mathcal{FCS}$ and $\mathcal{FS}$ (as we saw in Section 2.2), some information about the support is lost when it is used instead of them. Nevertheless, $\mathcal{FCS}$ does keep that information and still represents the same patterns as $\mathcal{FS}$ without the redundant ones.

Unfortunately, in SDM the problem of finding the closed set of patterns is more complicated since an item can appear from few to many times in the same sequence depending on the number of transactions. While in [Pasquier et al., 1999] the author defined a Galois connection that allows to identify the unique closed pattern that summarizes a set of non-closed patterns, in the sequence domain this property is not accomplished, and Casas Garriga redefined it in [Casas-Garriga, 2005]. Nevertheless some algorithms capable of finding $\mathcal{FCS}$ have been developed, and even some techniques are proposed to summarize in a better way the sequences in $\mathcal{FCS}$ by means of partial orders of closed sequences. We summarize all these approaches in the next Subsection.

Algorithms

There are two approaches for obtaining $\mathcal{FCS}$: 1) to modify the algorithms in order to obtain only closed sequences, and 2) by means of a post-processing of the $\mathcal{FS}$ generated by any algorithm. If we try to adapt directly the CHARM [Zaki and Hsiao, 2002] or some another algorithm from IDM, we can see that many problems arise because of having multiple transactions per sequence. Although a post-processing phase from the $\mathcal{FS}$ (obtained by any algorithm) is possible, it would be very interesting to avoid generating many redundant patterns while the algorithm is executed. Han et al. proposed some algorithms [Yan et al., 2003; Wang et al., 2007; Tzvetkov et al., 2005] to find $\mathcal{FCS}$ among which we briefly describe CloSpan [Yan et al., 2003] and Bide [Wang et al., 2007], both based on the Pattern Growth strategy.

CloSpan [Yan et al., 2003]. It is based on combining the ideas of closed sequence and projected database: if two different prefixes α and β (with $\alpha \prec \beta$) produce the same projected database $D_\alpha = D_\beta$ then both prefixes have the same support, and hence, α is not a closed sequence and it can be safely pruned. The algorithm uses an enumeration tree structure and when a non-closed sequence is found, the algorithm, instead of generating that branch in the tree, links the subsequence with its supersequence. The algorithm considers two different situations: 1) backward subpattern pruning and 2) backward superpattern pruning, showed in Figure 2.10. The first pruning occurs when the algorithm finds a subpattern with the same projected database as a superpattern previously generated, while the second pruning is the opposite case, and it takes place when the algorithm finds a super-pattern of a frequent sequence previously found. Since it is very inefficient to save the projected database, the algorithm has a hash function indexed by the size of the projected database and when two sequences α and β have the same size value and either $\alpha \preceq \beta$ or $\beta \preceq \alpha$ some of backward pruning methods is applied.

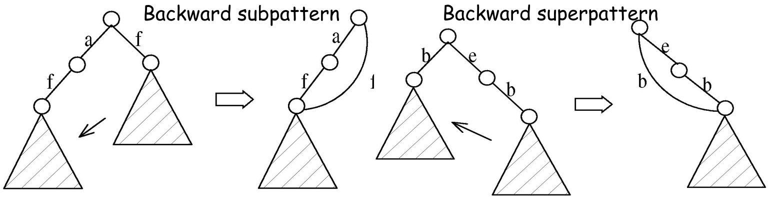


Figure 2.10: Pruning methods of CloSpan.

Bide [Wang et al., 2007]. The main idea of Bide is to avoid maintaining the set of sequences already mined. Most algorithms (both in IDM and SDM) have to keep in memory the sequences already found to be able to decide whether a new sequence is closed. Bide proposed three steps: 1) forward-extension event checking, 2) backward-extension event checking and 3) backscan pruning. While the first two steps are responsible for ensuring that the final set returned by the algorithm is only composed of closed sequences, the third one prunes the search space, so as to avoid mining many prefixes identified as non-closed.

Besides CloSpan and Bide, another algorithm called TSP has also been proposed [Tzvetkov et al., 2005]. This algorithm introduces another different way to mine closed patterns: instead of providing a minimum support, they provide a minimum number of closed sequences with a minimal length, the algorithm tries to search for the closed sequences that satisfy these constraints.

Casas-Garriga [Casas-Garriga, 2005] proposed a new way to consider the Galois connection in SDM and she defined a post-processing step to form partial orders with closed sequences obtained by an algorithm like CloSpan or Bide. These partial orders join several closed sequences, providing a more reduced output with more meaningful patterns. Casas-Garriga proposes three steps to obtain these partial orders :

1. To mine closed sequential patterns (e.g. by means of CloSpan or Bide).

2. To mine maximal conjunctive groups of non-redundant sequential patterns that are observed in the same transactions.

3. To convert every group to a partial order, where each closed sequential pattern detected in the first step is a path in the partial order.

To sum up, the approach of finding closed sequences is the most used in SDM to reduce $\mathcal{FS}$ but even this does not solve the problem of having a lot of similar patterns that are difficult to interpret by an expert. To improve this output closed partial order can be created, but, so far, there only exists the Casas-Garriga post-processing approach, which can have some limitations when it faces big outputs.

2.4 Time Interval data

The main motivation for introducing this kind of representations was due to the necessity of representing the persistence of an event over a period of time. As we said in Section 2.1, there exist several representation capable of expressing relations between intervals. The first representation was introduced by Allen [Allen, 1983]. Even though Allen's algebra was originally invented for studying temporal reasoning tasks, such as deriving time intervals associated with temporal facts or the consistency checking, it has also been widely used in temporal data mining.

As we could see in 2.2 Section, among the different approaches for representing interval relations, Allen's intervals algebra is the most widely studied and accepted one. The defined thirteen relations $R_I = \{before$, *meets*, *overlaps*, *starts*, *during*, *equals*, $finishes$, *contains*, *started by*, $finished$ *by*, *overlapped by*, *met by*, $after\}$ such that given a pair of intervals there exists only one possible interval relation between them. Figure 2.11 shows the thirteen Allen's relations. From now on we refer to each interval relation by means of its short name as shown in Table 2.2.

Interval relation name	short name
$before$	$<$
meets	m
overlaps	o
starts	s
during	d
equals	$=$
$finishes$	f
contains	c
started by	s^{-1}
$finished$ *by*	f^{-1}
overlapped by	o^{-1}
met by	m^{-1}
$after$	a

Table 2.2: Short names for interval Allen's relations.

An interval can be described by means of two points: the temporal point when the interval starts (beginning point) and the point when the interval finishes (end point). These two points must be in a distance longer than 0 time units, since in a database with interval data, each interval has a duration greater than zero. In the same way, we can define the relations between intervals through point relations, and hence we can establish a correspondence with the Allen's relations, as it is shown in Figure 2.11.

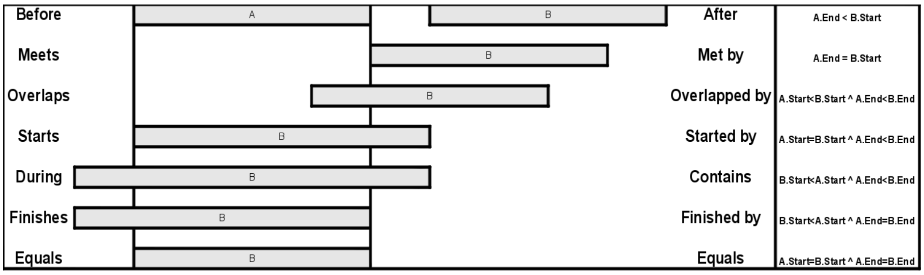
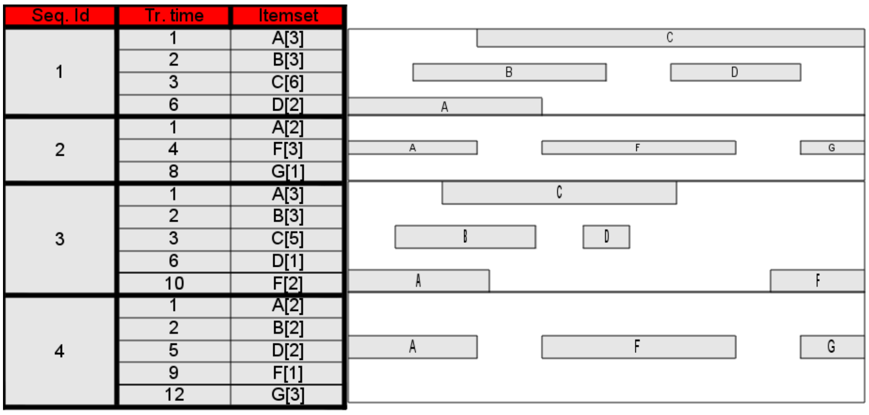

Before
A
B
After
A.End < B.Start
Meets
B
Met by
A.End = B.Start
Overlaps
B
Overlapped by
A.Start<B.Start ^ A.End<B.End
Starts
B
Started by
A.Start=B.Start ^ A.End<B.End
During
B
Contains
B.Start<A.Start ^ A.End<B.End
Finishes
B
Finished by
B.Start<A.Start ^ A.End=B.End
Equals
B
Equals
A.Start=B.Start ^ A.End=B.End

Figure 2.11: Allen's algebra relations.



Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	D[2]
2	1	A[2]
	4	F[3]
	8	G[1]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[1]
	10	F[2]
4	1	A[2]
	2	B[2]
	5	D[2]
	9	F[1]
	12	G[3]

Figure 2.12: Example of an Interval-based sequential database.

Some years later, Freksa [Freksa, 1992] proposed his semi-interval's relations. Based on disjunction of Allen's relations, Freksa defined eleven relations considering only a concrete point of the interval, either the beginning or the end.

Reich et al. [Reich, 1994] extended Allen's algebra to also include points. While Vilain [Vilain, 1982] had proposed the same issue, adding thirteen new relations, some years before, Reich only needed to add five new relations to consider each possible relation between points and intervals. Recently, Roddick [Roddick and Mooney, 2005] proposed to consider midpoints in Allen's relations in order to get more expressive patterns. With this change the thirteen original relations become forty nine relations.

Finally, Moerchen [Mörchen, 2006] introduced the TSKR representation, considering only the coincidences of different intervals and simplifying the representation.

Although we can establish relationships between a point and an interval as Meiri proposed in [Meiri, 1996] (note that a point is an interval where both beginning and end points occur at the same time), most algorithms are interested in mining only intervals. As an example, Figure 2.12 shows an interval database composed of four sequences.

Regarding quantitative representation, the standard point representation shows the temporal distance that exists between an event and the following one. This representation can be also used for quantitative interval representation taking into account the distance between the interval boundary points (either the distances of a point interval respect to another point of a different interval or the distance between the boundary points in a same interval). Of course, a combined point and interval representation is also possible because we do not distinguish between normal points or boundary points.

Since we will use Allen's relations, when we are interested in mining quantitative interval patterns we need to take into account the temporal distance in the Allen's relations. Figure 2.13 shows the Allen's relations with the temporal distances associated with them, when it is needed. We modify every relation such that we add the temporal distance between the temporal occurrence of the first interval end point with respect to the second interval beginning point. Besides the temporal distance, we need to consider the interval duration in order to express the different quantitative patterns. Note that, in the quantitative relations of Figure 2.13 we add the temporal distances just in some relations that need them. Thus, $\{<, o, c\}$ relations need that temporal distance since we need two know the concrete distance that there exists between the intervals immersed in those relations, whereas $\{m, s, f^{-1}, =\}$ do not need such temporal distance because we can exactly order the intervals that are related. For instance, if $b[2]\_ < [5]\_a[3]$ we know that b occurs five time units before a whereas if $b[2]\ s\ a[3]$ we know that the beginning of both a and b occur at the same time since that is exactly what *starts* relation expresses.

When we use quantitative relations instead of qualitative ones, we find a number of quantitative patterns represented by the same qualitative pattern. For example, in the database shown in Figure 2.1, we can see that the qualitative pattern $s = \langle(b)\rangle$ is frequent since it occurs at the sequences (1,2,3 and 4); however, if we are interested in quantitative sequences, we find two sequences $s_1 = \langle(b[2])\rangle$ and $s_2 = \langle(b[3])\rangle$, appearing s_1 at sequences 2 and 4, whereas s_2 occurs at sequence 1 and 3.

2.4.1 Allen's algebra and pattern representation

When using Allen's relations for data mining, it is usual to consider only the subset of *direct* relations and the equals relation, $R_{I_{reduced}} = \{<, m, o, s, f^{-1}, c, =\}$, ignoring the *inverse* relations $>, m^{-1}, o^{-1}, s^{-1}, f, d$. For example, if an event A *is_overlapped_by* another event B, $(A\ o^{-1}\ B)$, we can instead say that B *overlaps* A, $(B\ o\ A)$. While with point data we could establish an order in the "point sequence", with interval data this is not so simple because each interval has to be related with all of the rest of intervals, otherwise some ambiguity can exist. For instance, in the point approach we could infer that if A *before* B and B *before* C, then A *before* C $(A < B < C)$; while if in the interval approach we have that A *overlaps* B and B *overlaps* C, $(A\ o\ B\ o\ C)$, several relations can occur between A and C, as it is shown in Figure 2.14.

Just with the current representation we could start to deduce qualitative or quantitative patterns by means of Allen's relations, depending on which we are interested. For instance, in the second sequence of the shown database, $s_2 = \langle(1, \langle a, 2\rangle)(4, \langle b, 2\rangle\langle f, 3\rangle)(8, \langle g, 0\rangle)\rangle$ we can observe the Allen qualitative relations $a < b$, $a < f$, $a < g$, $b\ s\ f$, $b < g$ and $f < g$; or the quantitative relations $a[2]\ < [1]\ b[2]$, $a[2]\ < [1]\ f[3]$, $a[2]\ < [5]\ g[0]$, $b[2]\ s\ f[3]$, $b[2]\ < [2]\ g[0]$ and $f[3]\ < [1]\ g[0]$. From those relations, we could start to create longer patterns and, step by step, to obtain the whole pattern set. But, in order to have a simpler and more readable representation, we change this representation for other equivalent ones that make us easy the task of mining frequent patterns.

Another difficulty compared to the point based representation is the high cost of

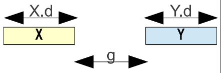
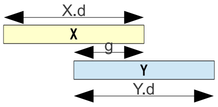
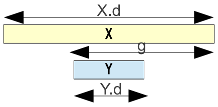
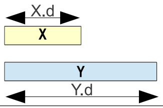
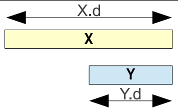
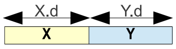
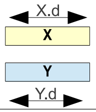
Allen's Relation	Allen's Relation (Inverse)	Graphical Example	Boundary point representation
X before[g] Y	Y after[g] X		(t=0,X+)(t=X.d,X-)(t=X.d+g,Y+)(t=X.d+g+Y.d,Y-)
X overlaps[g] Y	Y overlapped-by[g] X		(t=0,X+)(t=X.d-g,Y+)(t=X.d,X-)(t=X.d+Y.d-g,Y-)
X contains[g] Y	Y during[g] X		(t=0,X+)(t=X.d-g,Y+)(t=X.d+Y.d-g,Y-)(t=X.d,X-)
X starts Y	Y started-by X		(t=0,X+ Y+)(t=X.d,X-)(t=Y.d,Y-)
X finished-by Y	Y finishes X		(t=0,X+)(t=X.d-Y.d,Y+)(t=X.d,X- Y-)
X meets Y	Y met-by X		(t=0,X+)(t=X.d,X- Y+)(t=X.d+Y.d,Y-)
X equal Y	Y equal X		(t=0,X+ Y+)(t=X.d,X- Y-)

Figure 2.13: Conversion from Allen's intervals to boundary point sequence.

checking whether a sequence is a subsequence of another one. In order to overcome this problem some authors have proposed several representations of the relations that facilitate this operation:

1. **Hierarchical patterns** [Kam and Fu, 2000]. In this approach the authors make a representation similar to the point-based data approach, i.e. they establish a sequence of intervals instead of relating each interval with all of the others explicitly. They combine two intervals with a temporal relation to build a new interval. This resulting interval is also combined with another single interval and so on. At last, there is a long interval composed of a combination of intervals. The problem is that this representation is ambiguous since different patterns are represented with the same hierarchical pattern. In Figure 2.14 we show how these patterns are built and some of the problems that this representation can have.

2. **Nested representation with counters** [Patel et al., 2008]. The Patel et al.'s proposal consists of making only a simple sequence by linking an interval with those that occur after it. Actually, it is the same representation as the above mentioned Hierarchical patterns, and, in order to avoid the ambiguity, now five counters are used to show the number of *contains*, *finished_by*, *meets*, *overlaps* and *starts* relations that there exist in the pattern. Figure 2.15 shows an example. Now, to represent k intervals we need these k intervals and 5 counters between each pair of intervals,

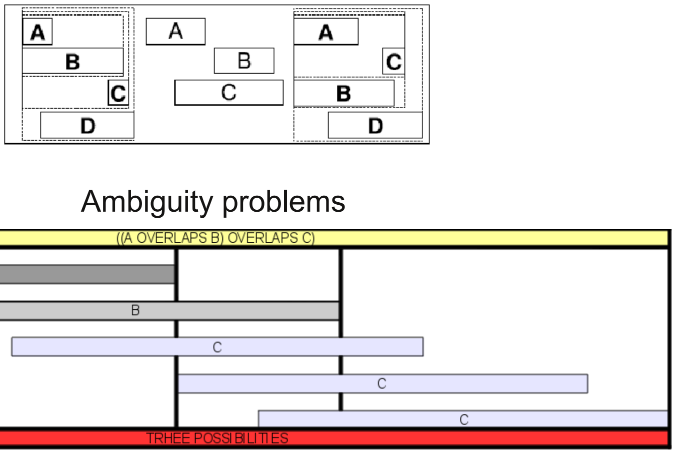
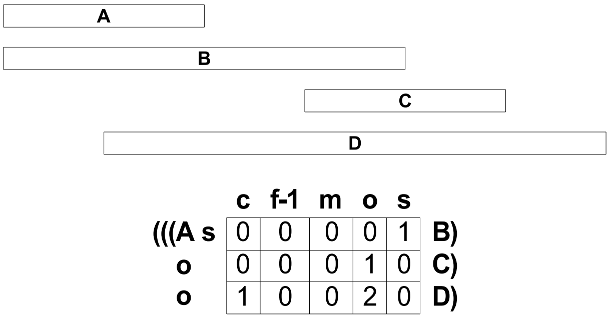


Figure 2.14: Hierarchical representation.

that is, $5(k-1)$ counters. The main problem here is that this representation is not very readable.



	c	f-1	m	o	s	
(((A s	0	0	0	0	1	B)
o	0	0	0	1	0	C)
o	1	0	0	2	0	D)

Figure 2.15: Nested representation.

3. **Triangular matrix representation** [Höppner, 2001]. Hoeppner proposed a matrix to represent every relation between all intervals as shown on the top picture of Table 2.3. An interval A placed in a row i has an interval relation with an event B placed in the column j; we call R_{AB} to that relation. Since, in the matrix, an event has an *equals* relation with itself and two events A and B have two relations between them, the direct relation from A to B and the direct relation from B to A (being the latter the inverse of the former relation), we can avoid the use of the relations highlighted in dark in the upper table of Table 2.3. Therefore, for k intervals there are $\frac{k(k-1)}{2}$ relations in the triangular matrix such as shown in the bottom table in Table 2.3. We call *simplified matrix* to this approach, and it is a good option to represent the pattern with all its relations in a compact and non-ambiguous way.

4. **Sequence of interval boundaries** [Wu and Chen, 2007]. Wu and Chen represent each interval with their boundary points, i.e. beginning point (A^+) and end point (A^-). So, a pattern with k intervals is represented by a sequence of $2k$ boundary points related between them with $2k-1$ point relations. Here, instead of having a pattern composed of intervals with Allen's relations, a pattern is built with points

	A	B	C	D	E
A	R_{AA}	R_{AB}	R_{AC}	R_{AD}	R_{AE}
B	R_{AB}^{-1}	R_{BB}	R_{BC}	R_{BD}	R_{BE}
C	R_{AC}^{-1}	R_{BC}^{-1}	R_{CC}	R_{CD}	R_{CE}
D	R_{AD}^{-1}	R_{BD}^{-1}	R_{CD}^{-1}	R_{DD}	R_{DE}
E	R_{AE}^{-1}	R_{BE}^{-1}	R_{CE}^{-1}	R_{DE}^{-1}	R_{EE}

	B	C	D	E
A	R_{AB}	R_{AC}	R_{AD}	R_{AE}
B		R_{BC}	R_{BD}	R_{BE}
C			R_{CD}	R_{CE}
D				R_{DE}

Table 2.3: Triangular Matrix structure.

and their relations. This kind of patterns has the constraint that a pattern with an incomplete interval is not correct. Thus, all the intervals have to be properly translated. This representation is more compact than the triangular matrix approach and it is also non-ambiguous. Figure 2.16 shows an example of this representation for a sequence of four interval events.

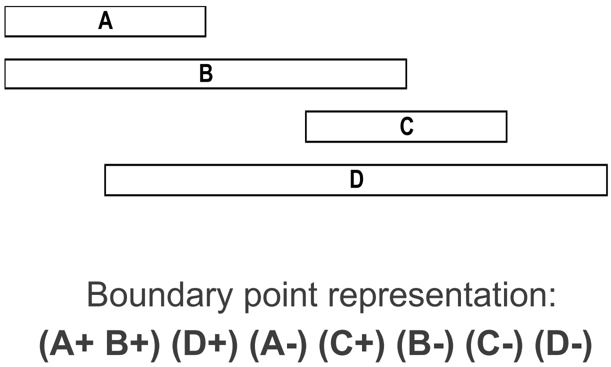

A
B
C
D
Boundary point representation:
(A+ B+) (D+) (A-) (C+) (B-) (C-) (D-)

Figure 2.16: Boundary points representation.

5. **SIPO patterns (Semi Intervals Partial order)** [Mörchen and Fradkin, 2010]. Moerchen and Fradkin proposed this representation as a sequence of interval boundaries without any constraint. Therefore, in these patterns, the intervals can be incomplete, appearing only its beginning or the end point. With this approach we can build patterns that represent partial orders with less constraints that in other approaches. Figure 2.17 shows both an example of Semi-interval pattern and a posterior partial order inferred from three Semi Interval patterns.

2.4.2 Algorithms for qualitative interval patterns

All the issues discussed in the point data algorithms (see Subsection 2.3.1) are still present when we want to mine interval patterns. But now, for this case, both the hyperlattice

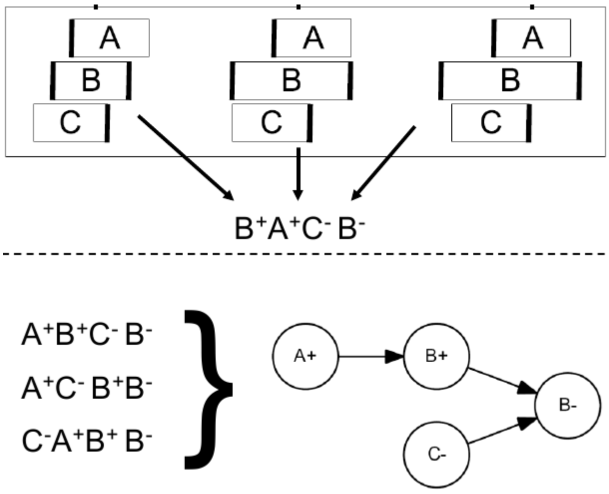
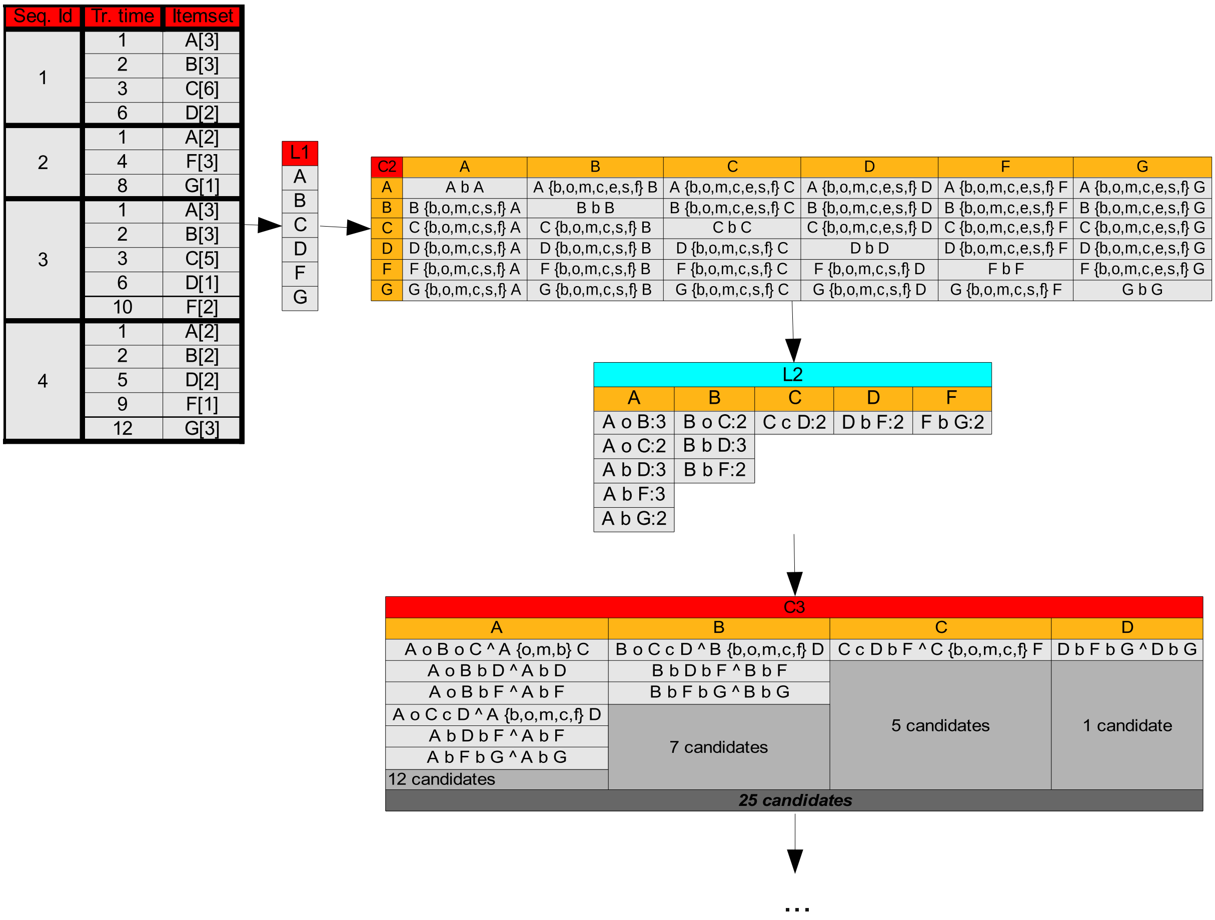


Figure 2.17: SIPO representation.

and the meet-semilattice are quite more complex since now we have seven direct relations ($\{<, m, o, s, f^{-1}, c, =\}$) instead of two direct relations ($\{<, =\}$). Anyway, all the same strategies can be applied to interval mining. However, only a few algorithms have been develop for mining interval temporal data, and they have to deal with two main difficulties originated by the complexity of the patterns: a higher amount of candidate patterns and a more complex subsequence checking process.

For instance, Figure 2.18 shows an outline about how an Apriori-based algorithm for the interval database of our example could work. In that figure we consider a min_support of 2 and we do not take into account the pattern representation used by the algorithm since the main goal here is to focus on the complexity for the algorithm in a general way. Note the high number of candidate sequences generated. For the set of 2-candidates $\mathcal{C}_2$ any relation is possible between two intervals, e.g. if A and B belong to the set of frequent intervals $\mathcal{L}_1$ we have thirteen candidates, $A < B$, $A\ m\ B$, $A\ o\ B$, $A\ c\ B$, $A\ s\ B$, $A\ f\ B$, $A = B$, $B < A$, $B\ m\ A$, $B\ o\ A$, $B\ s\ A$, $B\ f\ A$, $B\ c\ A$. Proceeding in the same way, 201 candidate 2-sequences are identified, and then the algorithm checks their support to get only the frequent ones. Once we have the set of frequent 2-sequences $\mathcal{L}_2$ we can combine candidates in a similar way as GSP does. The problem in this step is that with two frequent 2-sequences it is possible to create several candidates. For example, from the frequent 2-sequences $A\ o\ B$ and $B\ o\ C$ we can create the sequence $A\ o\ B\ o\ C$ but, since the Allen's algebra establish relations between all the intervals, we do not know what is the relation between intervals A and C. The simplest approach is to assume any relation between A and C and later discard those infrequent ones.

In [Campos et al., 2007] the authors proposed a better solution and this consists of applying temporal reasoning to generate only those candidate whose relation is temporally consistent. With this assumption, from $A\ o\ B$ and $B\ o\ C$ there exist only three different possibilities: $A\ o\ B\ o\ C \wedge A\ o\ C$, $A\ o\ B\ o\ C \wedge A\ m\ C$ and $A\ o\ B\ o\ C \wedge A\ b\ C$. Using this option, only 25 candidate 3-sequences are found. Afterwards the algorithm executes a pruning step and checks the support of all the candidates of $\mathcal{C}_3$, and it repeats the Apriori process until the end. Note that an efficient pattern representation becomes crucial for the support counting and for the pruning step. On the one hand, the operations have to be fast enough to be worthwhile doing temporal reasoning rather than generating all possible combinations of candidates. On the other hand, in the support counting and in the pruning steps, we continuously do the subsequence checking and this operation must be as fast as possible, or otherwise the algorithm becomes very slow and the pruning phase it is not worthwhile.



Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	D[2]
2	1	A[2]
	4	F[3]
	8	G[1]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[1]
	10	F[2]
4	1	A[2]
	2	B[2]
	5	D[2]
	9	F[1]
	12	G[3]

L1
A
B
C
D
F
G

C2	A	B	C	D	F	G
A	A b A	A {b,o,m,c,e,s,f} B	A {b,o,m,c,e,s,f} C	A {b,o,m,c,e,s,f} D	A {b,o,m,c,e,s,f} F	A {b,o,m,c,e,s,f} G
B	B {b,o,m,c,s,f} A	B b B	B {b,o,m,c,e,s,f} C	B {b,o,m,c,e,s,f} D	B {b,o,m,c,e,s,f} F	B {b,o,m,c,e,s,f} G
C	C {b,o,m,c,s,f} A	C {b,o,m,c,s,f} B	C b C	C {b,o,m,c,e,s,f} D	C {b,o,m,c,e,s,f} F	C {b,o,m,c,e,s,f} G
D	D {b,o,m,c,s,f} A	D {b,o,m,c,s,f} B	D {b,o,m,c,s,f} C	D b D	D {b,o,m,c,e,s,f} F	D {b,o,m,c,e,s,f} G
F	F {b,o,m,c,s,f} A	F {b,o,m,c,s,f} B	F {b,o,m,c,s,f} C	F {b,o,m,c,s,f} D	F b F	F {b,o,m,c,e,s,f} G
G	G {b,o,m,c,s,f} A	G {b,o,m,c,s,f} B	G {b,o,m,c,s,f} C	G {b,o,m,c,s,f} D	G {b,o,m,c,s,f} F	G b G

L2				
A	B	C	D	F
A o B:3	B o C:2	C c D:2	D b F:2	F b G:2
A o C:2	B b D:3			
A b D:3	B b F:2			
A b F:3				
A b G:2				

C3			
A	B	C	D
A o B o C ^ A {o,m,b} C	B o C c D ^ B {b,o,m,c,f} D	C c D b F ^ C {b,o,m,c,f} F	D b F b G ^ D b G
A o B b D ^ A b D	B b D b F ^ B b F	5 candidates	1 candidate
A o B b F ^ A b F	B b F b G ^ B b G		
A o C c D ^ A {b,o,m,c,f} D	7 candidates		
A b D b F ^ A b F			
A b F b G ^ A b G			
12 candidates			
25 candidates			

...

Figure 2.18: An Apriori-like algorithm with interval data.

Some algorithms have been proposed to face to SDM with interval data [Kam and Fu, 2000; Höppner, 2001; Papapetrou et al., 2005; Winarko and Roddick, 2007; Campos et al., 2007; Wu and Chen, 2007; Patel et al., 2008; Moskovitch and Shahar, 2009; Wu and Chen, 2009; Chen, 2010; Chen et al., 2011]. In this survey we want to highlight the more interesting among all of them and we classify them into one of the different strategies mentioned above in Section 2.3. We will overview the Apriori-based algorithms [Campos et al., 2007] and IEMiner [Patel et al., 2008], the H-DFS [Papapetrou et al., 2005], Karmalego [Moskovitch and Shahar, 2009] and HTPM [Wu and Chen, 2009] algorithms based on the Vertical Database Format, and Armada [Winarko and Roddick, 2007], TPrefixSpan [Wu and Chen, 2007], CTMiner [Chen, 2010] and CEMiner [Chen et al., 2011] algorithms which follow the Pattern Growth strategy. Among these algorithms, those based on Pattern Growth strategy [Chen, 2010; Chen et al., 2011] seem to obtain better results.

A common characteristic in all the algorithms is that the use of more expressive patterns and a larger number of relations than in the standard point based sequential algorithms converts the scalability into a key issue. This scalability will be reflected in the search strategies, pattern representation, the candidate generation, the counting of support and the pruning methods used in the algorithms.

Apriori-like algorithm

Campos et al. [Campos et al., 2007] proposed an Apriori-like approach based on a Triangular Matrix representation (see Section 2.4.1). The main points exposed in the algorithms are:

- They mine constraint networks using temporal relations.
- The method used like pruning consist of making a consistency checking.

The main advantage of this algorithm is the good expressiveness for their patterns and its pruning power. However, since the pruning method is quite inefficient, the algorithm becomes quite slow.

The IEMiner [Patel et al., 2008] algorithm proposed by Patel et al. introduced the nested representation with counters previously mentioned (see Section 2.4.1) that allows them to speed up the subsequence checking operation, improving the pruning and support counting phases. The main contributions of the algorithm are:

- In the candidate generation they define the concept of dominant interval, i.e. the interval whose end occurs in the latest time. The algorithm combines a frequent $k-1$-sequence α with a frequent 2-sequence β to build a candidate k-sequence γ, where the first interval in β has to be the dominant interval in α. What is more, they maintain the set of frequent 2-pattern up-to-date, i.e. when they create the candidate k-sequences $\mathcal{C}_{k+1}$ the 2-pattern chosen must occur in at least $k-1$ frequent k-patterns.
- In the support counting phase, the algorithm reads every input sequence only once and maintains a list of possible candidates that can be covered. To that end, the algorithm uses two lists:
    1. One list to control active intervals. An interval is active if it is not ended in the specific time considered by the support checking. An active interval can have any relation with a new interval except for the "before" relation.
    2. Another list to control passive intervals. An interval is passive if it is already ended in the specific time considered by the support checking. A passive interval can only have a "before" relation with the newer intervals found.

The main advantage of this algorithm is that it has a good pruning power, while as drawbacks the algorithm has the typical problems of Apriori-based methods, where the most limiting one is the necessity of reading several times the original database to count the support of every $\mathcal{C}_k$.

Vertical Database Format

Papapetrou et al. [Papapetrou et al., 2005] proposed three algorithms based on the Vertical Database Format strategy that were essentially the same except for the way they explore the space search. The authors made BFS, DFS and Hybrid searches (called H-DFS), being H-DFS the fastest one. This search consists of exploring the first two levels in a BFS and from there the algorithm continues with a DFS.

H-DFS uses the same representation as IEMiner and it only considers five Allen's relation (they do not take into account the *starts* and *finished by* relations). The algorithm also uses an enumeration tree in order to factorize the patterns discovered. Two main changes are done with respect to SPADE:

- In the candidate generation the algorithm joins a frequent $k-1$-sequence with a frequent interval (1-sequence). Besides, the algorithm does not use temporal reasoning. Therefore all of possible combinations are developed, creating a very big candidate set $\mathcal{C}_k$. In order to reduce this big $\mathcal{C}_k$, a pruning step is executed, checking if there exists any non-frequent $k-1$-subsequence in other branches previously generated.

- The IdLists now have to include information about intervals. So, for each IdList, besides the sequence and transaction identifiers (note that the Tid is the start time for the interval), they add the end time for the interval so as to check whether each possible temporal relation is present.

Later in 2009, Moskovitch et al. proposed the Karmalego algorithm [Moskovitch and Shahar, 2009]. The main idea of this algorithm is almost the same as in H-DFS algorithm, but Moskovitch takes the idea of [Campos et al., 2007] and adds temporal reasoning to the candidate generation phase in order to reduce the candidate sets. Karmalego works very similar to H-DFS: it uses an enumeration tree and the same type of IdLists, it generates the first two levels of frequent patterns with BFS and from there it goes on with DFS. The main changes in Karmalego with respect to H-DFS are:

1. It uses eight Allen's relations (they also use the *started by* relation, besides the seven relations direct relations).

2. It uses a triangular matrix to represent the patterns (see Triangular matrix representation in Section 2.4.1).

3. It generates k-candidates from a frequent $(k-1)$-pattern and a frequent 1-pattern. The algorithm works in a depth-first search way and its main contribution is that they use the transitivity of temporal relations proposed by Freksa [Freksa, 1992] to obtain the suitable candidates They use temporal reasoning to generate the lowest possible number of candidates.

The advantages and drawbacks in this strategy are just the same as in the point data case, but it this case the candidate generation and temporal join operations are more complex. However, the algorithm generates a huge amount of candidates and it does not use any prune method to reduce the number of candidates, leading the algorithm to have a quite slow counting of support phase.

Finally, HTPM [Wu and Chen, 2009] was developed for mining hybrid temporal patterns from event sequences, which contain both point-based and interval-based events. The main points of this algorithm are:

- The algorithm mines both points and intervals at the same time.
- Despite most of the Vertical Database Format algorithms are executed in a DFS way, HTPM executes a BFS in order to implement a pruning method
- Both steps, candidate generation and counting of support, are merged in a same method in the algorithm.

As an overall assessment of the algorithm, although it does not scan several times the database, the breath-first search way that the algorithm executes can lead it to memory overflow.

Pattern Growth like algorithms

TPrefixSpan[Wu and Chen, 2007], Armada[Winarko and Roddick, 2007], CTMiner [Chen, 2010] and CEMiner [Chen et al., 2011] are the four most interesting algorithms for temporal interval data which belong to the Pattern Growth strategy. While TPrefixSpan, CTMiner and CEMiner are based on the PrefixSpan [Pei et al., 2004] algorithm, Armada is based on the Memisp algorithm [Lin and Lee, 2005]. We summarize the main changes added to these interval algorithms with respect to a normal algorithm that follow the Pattern Growth strategy:

Armada [Winarko and Roddick, 2007]. This algorithm uses a triangular matrix to represent patterns. Armada maintains the occurrence time of the sequence of intervals for the prefix used in each pseudo-projection (called index set in Memisp [Lin and Lee, 2005]). After that, the algorithm searches for the frequent intervals in the projected database, taking into account the relation of each new interval with every component in the prefix.

TPrefixSpan [Wu and Chen, 2007]. This algorithm uses a sequence of interval boundary points as pattern representation (see Subsection 2.4.1). The definitions of prefix, suffix and database projection are more complex than in the original algorithm. The main overhead takes place when the algorithm finds a frequent interval within a projected database and there is not only one single combination between the prefix for the projected database and the frequent item just found.

CTMiner [Chen, 2010]. This algorithm for mining temporal patterns is described by a representation based in the interval coincidences inspired in the TSKR representation. The algorithm uses some pruning methods to reduce the search space and mines the different interval coincidences and it composes the different frequent patterns. CTMiner is also capable of mining both point and interval events through a single algorithm.

CEMiner [Chen et al., 2011]. This algorithm, capable of mining both points and intervals, also uses a sequence of interval boundary points as pattern representation and translates every interval to points. Then, it mines the frequent patterns directly in points, discarding those pairs of points that do not belong to the same interval. Actually, CEMiner simulates a Bide algorithm, thus the outcome corresponds to the closed frequent patterns.

The drawbacks of these algorithms are the same as in the point data case, but the algorithms for the interval approach are more complex due to the overhead added by the number of Allen's relations.

2.4.3 Algorithms for quantitative interval patterns

As in the point data case, all the issues discussed in qualitative interval algorithms remain when we mine quantitative interval patterns. As before, since there can potentially be infinite relations as a quantitative relation has a temporal distance associated, and the domain of that distance is infinite. Therefore, both the hyperlattice and the meet-semilattice are far more complex because we have many more relations. Fortunately, in practice the temporal distance associated with relations is bounded by the structure of the database and the longest distance between the interval appearing in the first and the last transactions in a sequence.

Since the problem of mining the frequent quantitative interval mining is quite similar to the qualitative version, the same strategies can be applied. However, both the subsequence checking and the candidate generation steps have to be modified. Furthermore, the mining of the frequent patterns of length 2 become now crucial in some strategies since we need to know all the different temporal distances that can extend the different frequent patterns in order to get largest frequent supersequences.

Regarding to the different algorithms that have been developed for mining quantitative interval patterns, only few works have been done. In particular, we highlight four algorithms:

1. In **QTempIntMiner** [Guyet and Quiniou, 2008], the authors, from an Apriori algorithm, use a density based clustering in order to find the quantitative patterns. At every k level, the use a very expensive density based clustering that makes the algorithm time-consuming.

2. In **QTiPrefixSpan** [Guyet and Quiniou, 2011], the same authors make a similar algorithm but this time based on a Pattern Growth algorithm, concretely PrefixSpan. As before, the use a very expensive density based clustering makes the algorithm quite slower.

3. In **QTPrefixSpan** [Nakagaito et al., 2009], the authors propose an algorithm derived from the qualitative interval algorithm TPrefixSpan, based on Pattern Growth Strategy. QTPrefixSpan is capable of discovering quantitative patterns and its main contribution is that, in every extension of a pattern, it executes a density-based numerical clustering in order to know the range of time in the quantitative relations.

As before, this derived in an overhead in the original TPrefixSpan, and therefore the algorithm is very slow.

4. Regarding the discovery of temporal relations, a better work was done in **ASTP-miner** algorithm [Álvarez et al., 2013], where the authors define an Apriori algorithm to search for quantitative patterns through a clustering technique to obtain patterns of length 2. Then frequent patterns of length k are combined to build candidate patterns of length k+1, and all the later candidates are generated from the frequent 2-patterns. However, the algorithm is based in Apriori strategy and has to make several database scans in order to count the support.

2.4.4 Comparatives

Regarding the experiments which compare the algorithms for temporal interval data, there are not a lot of comparative studies. Anyway, the behaviours described in Subsection 2.3.3 are still maintained, but now there is an important overhead in each of the different Algorithms.

In CEMiner work, this algorithm is compared to T-PrefixSpan, H-DFS, IEMiner and CTMiner, obtaining better execution times than all of them. However, while several efforts have been made implementing tuned algorithms in Pattern Growth strategy, little effort has been made in developing algorithms based on Vertical Database Format Strategy for mining interval patterns.

Summarizing, the behaviour of these algorithms must be similar to the correspondent algorithms in Section 2.3 but these algorithms can be more or less overloaded depending on the pattern representation and on the candidate generation.

2.4.5 Reduction of patterns

The reduction of interval patterns has not been widely studied so far. In principle, the concepts of maximal and closed patterns could also be applied. The CEMiner algorithm [Chen et al., 2011] depicted above, is based on Bide algorithm [Wang et al., 2007] and it searches the closed patterns set. Besides this, Mörchen et al. [Mörchen and Fradkin, 2010] proposed the creation of intervals and semi-intervals partial orders based on the idea of their previous work of Time Series Knowledge Representation (TSKR) [Mörchen, 2006]. The process can discover both points and intervals and is divided in three steps:

1. The authors translate the original database into boundary points (as in TPrefixSpan).

2. They extract the closed patterns (temporal points data) using either CloSpan [Yan et al., 2003] or Bide [Wang et al., 2007]. They are interested in extracting those frequent patterns composed by points, semi-intervals or intervals (the rest of algorithms find only complete intervals). Even though they use that algorithm, they do not explain in detail how the algorithm works, and they simply use it as a tool to obtain results.

3. Finally, since all the closed patterns are composed by points (boundary points), they apply the process proposed by Casas-Garriga [Casas-Garriga, 2005] to obtain partial orders with closed sequences. The result is a set of partial orders composed by boundary points, such that each partial order can have only points, semi-intervals or intervals.

In the experiment sections of [Mörchen and Fradkin, 2010] the authors also obtain the closed set of interval patterns applying a brute force algorithm and they do a study about what patterns are more meaningful. According to the authors, in the studied databases, the partial order representation with semi-intervals is more meaningful than the closed interval patterns.

2.5 Conclusions

In this survey, we have presented a deep analysis of the different kind of algorithms, for both qualitative and quantitative patterns, developed so far to discover the frequent sequences in SDM.

To the best of our knowledge, there not exist any previous work that addressed a strategy comparison depending on the database properties such as we have done in this survey.

We have shown, in terms of comparison of the different strategies, that Vertical Database Format is the fastest approach. However, for databases where, on average, there are few transactions per sequence, the Pattern Growth strategy can be better.

Furthermore, we have shown the problems that emerge when we deal with quantitative patterns. Besides, we have described the additional changes that we have to do in the strategies in order to properly find quantitative patterns.

In the same way, we have also seen the problems that carry an algorithm implementation for interval mining. We have also exposed the typical problems that appear when we search for these patterns and we have explained that the pruning step, normally used in the Apriori-based and Vertical Database Format algorithms, gains much importance in this case.

Finally, we have also seen the problem of the pattern explosion, and we have shown some known solutions in the state-of-the-art to solve that problem in both the point and interval cases.

Chapter 3

PaGAPIS and FaSPIP: Two New Fast Algorithms for Mining Points and Intervals Qualitative Patterns

In Chapter 2 we saw that Vertical Database Format algorithms show a better behaviour when they deal with dense databases with long itemsets. Since most of the algorithms for mining qualitative patterns with points and intervals events are based on PrefixSpan algorithm, and its behaviour is derived from it, that idea motivates us design for an algorithm based on the Vertical Database Format strategy.

To this end, we describe here the algorithms PaGAPIS and FaSPIP to address the mining of qualitative patterns with both point and interval events. Both algorithms use a boundary point representation. For these algorithms all the definitions given in Chapter 2 are used. However, each algorithm needs to add some specific definitions or descriptions.

The chapter is organized as follows. In Section 3.1 we define this representation used for sequences and the basic operation between items. Sections 3.2 and 3.3 describe the new algorithm for points and intervals based on Pattern Growth strategy and Vertical Database Format strategy, respectively. An experimental and performance study is presented in Section 3.4, while a wide discussion about the behavioural differences of the algorithms is given in Section 3.5. Finally, we provide our conclusions in Section 3.6.

3.1 Additional definitions for problem setting

Given an input sequence $\alpha = \langle (t_1, I_1), (t_2, I_2), \ldots, (t_m, I_m) \rangle$, where the itemsets may contain both points and intervals, we are going to transform it into a sequence whose itemsets only contain points. That is, an interval will be represented by its beginning and end point. If a transaction contains an interval e with duration d and occurs at time x, $T = \langle (t = x, e) \rangle$, we introduce in the new sequence two transactions, $T' = \langle (t = x, e^+), (t = x+d, e^-) \rangle$; if a transaction contains a point e at time x, we introduce the same transaction in the new sequence. This new sequence will also be ordered by time. Besides, if an interval label appears more than once within a boundary sequence, we attach an occurrence number to distinguish its multiple occurrences. For instance, the sequence $\beta = \langle (t = 1, \langle A, 3 \rangle)(t = 2, \langle B, 3 \rangle)(t = 3, \langle C, 5 \rangle)(t = 6, \langle D, 0 \rangle)(t = 10, \langle A, 2 \rangle \langle B, 3 \rangle) \rangle$, with A, B and C being intervals and D being a point, is identified with the boundary sequence $\beta_e = \langle(t=1, {A_1}^{+})(t=2, {B_1}^{+})(t=3, C^{+})(t=4, {A_1}^{-})(t=5, {B_1}^{-})(t=6, D)(t=8, C^{-})(t=10, {A_2}^{+}{B_2}^{+})(t=12, {A_2}^{-})(t=13, {B_2}^{-})\rangle$. Since a boundary point is an item with a duration of zero and we use its event identifier instead, hereinafter, we will use event and item interchangeably. What is more, when we refer to any sequence as a frequent sequence instead of a concrete sequence in a database, we can ignore the itemset time information since we are only interested in the qualitative relations "$<$" and "$=$" that there exist between all the boundary point in the sequence. Thus, if the previous sequence were a frequent pattern found in the database, we would denote it as $\beta_e = \langle({A_1}^{+})({B_1}^{+})(C^{+})\ ({A_1}^{-})({B_1}^{-})(D)(C^{-})({A_2}^{+}{B_2}^{+})({A_2}^{-})({B_2}^{-})\rangle$. Henceforth, unless we need to make an explicit reference to a boundary input sequence, we will refer to all the sequences without temporal information as their itemsets.

Boundary point representation has several benefits, being the simplification of interval relations the most important one. The conversion from the Allen's thirteen relations into their corresponding boundary point representation is depicted in Figure 3.1. By means of this boundary point representation, we obtain a good representation that is: 1) scalable, since we only need a maximum of $2k$ points for describing k-items; 2) non-ambiguous, since relations appearing in a boundary point sequence correspond to only one interval relation, as we saw in Section 2; and 3) simple, since an interval sequence needs thirteen Allen's relations and a boundary point sequence only need the standard point relations $\{<, =, >\}$.

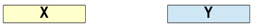
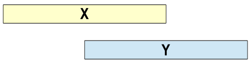
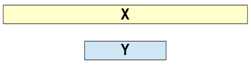
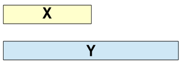
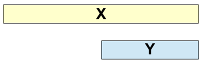
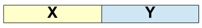
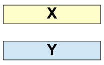
Allen's Relation	Allen's Relation (Inverse)	Graphical Example	Boundary point representation
X before Y	Y after X		(A+)(A-)(B+)(B-)
X overlaps Y	Y overlapped-by X		(A+)(B+)(A-)(B-)
X contains Y	Y during X		(A+)(B+)(B-)(A-)
X starts Y	Y started-by X		(A+ B+)(A-)(B-)
X finished-by Y	Y finishes X		(A+)(B+)(A- B-)
X meets Y	Y met-by X		(A+)(A- B+)(B-)
X equal Y	Y equal X		(A+ B+)(A- B-)

Figure 3.1: Conversion from Allen's intervals to boundary point sequence.

Given a sequence database, we can transform every original sequence to their equivalents boundary sequences. For example, in Table 3.1, we show our database example transformed into its four corresponding boundary sequences. From now on, we suppose that every temporal database is given in its boundary point representation.

Boundary Sequence database												
SID \ tid	1	2	3	4	5	6	7	8	9	10	11	12
1	(a^+)	(b^+)	(c^+)	(a^-)	(b^-)	(d)			(c^-)			
2	(a^+)		(a^-)	(b^+f^+)		(b^-)	(f^-)	(g)				
3	(a^+)	(b^+)	(c^+)	(a^-)	(b^-)	(d)		(c^-)		(f^+g)		(f^-)
4	(a^+)	(b^+)	(a^-)	(b^-)	(d)						(f^-)	(g)

Table 3.1: Example database converted to boundary point sequences.

Considering this representation, the implementation of the subsequence checking, sequence extensions or the operation with prefix and suffix for sequences of boundary points is straightforward. We say that the boundary point sequence α is a subsequence of another boundary point sequence β (or β is a supersequence of the α), denoted as $\alpha \preceq \beta$, if there exists a bijective function f which preserves the order and maps the boundary points in α to boundary points in β, in such a way that 1) $\forall e_i \in \alpha, \forall f(e_i) \in \beta, e_i \subseteq f(e_i)$ and 2) if $e_i < e_j \Rightarrow f(e_i) < f(e_j)$, $\forall e_i, e_j \in \alpha, \forall f(e_i), f(e_j) \in \beta$. We can also say that for sequences $\alpha = \langle I_{\alpha_1} I_{\alpha_2} \ldots I_{\alpha_n} \rangle$ and $\beta = \langle I_{\beta_1} I_{\beta_2} \ldots I_{\beta_m} \rangle$, there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $I_{\alpha_1} \subseteq I_{\beta_{j_1}}, I_{\alpha_2} \subseteq I_{\beta_{j_2}}, \ldots, I_{\alpha_n} \subseteq I_{\beta_{jn}}$. Besides, every subsequence α must have a correspondence between all the boundary points in β, and for those boundary points chosen in β that represent the beginning and the end points of a same interval event (${e_i}^+$ and ${e_i}^-$), the appearance of any other boundary point that represent the same interval event e_i between them is not allowed. For instance, the boundary point sequence $\alpha = \langle (a^+)(b^+)(c)(a^-)(b^-) \rangle$ is a subsequence of $\beta = \langle (a^+)({d_1}^+)(b^+)({d_1}^-)(c_1)(a^-)({d_2}^+)(c_2)({d_2}^-)(b^-) \rangle$ but not of $\gamma = \langle (a^+)({b_1}^+)({b_1}^-)(c)(a^-)({b_2}^+)({b_2}^-) \rangle$ because the event b_1 is closed before c.

Let $\alpha = \langle I_{\alpha_1} I_{\alpha_2} \ldots I_{\alpha_n} \rangle$ and $\beta = \langle I_{\beta_1} I_{\beta_2} \ldots I_{\beta_m} \rangle$ be two sequences with $m < n$. We say that β is a prefix with respect to α if $\forall i, 1 \leq i < m,\ I_{\alpha_i} = I_{\beta_i}$ and $I_{\beta_m} \preceq I_{\alpha_m}$, where the events in I_{β_m} are the first ones in I_{α_m}. Besides, the sequence $\langle (I_{\alpha_m} - I_{\beta_m}) I_{\alpha_{m+1}} \ldots I_{\alpha_n} \rangle$ is called the suffix of α with regards to prefix β.

We define some operations in order to extend a sequence with an item, creating a new sequence. We will need these operations in the candidate generation of our algorithms. Let $\alpha = \langle I_1 I_2 \ldots I_n \rangle$ be a sequence and let $b_i \in \{e_i^+, e_i^-, e_i\}$ be a point. We define the *S-extension* α' being $I'_n = I_n \cup b_i$. That is, the point b_i is added to I_n. For instance, given the sequence $\alpha = \langle (a)(b) \rangle$ and a point $c \in \mathcal{I}$, the sequence $\beta = \langle (a)(b)(c) \rangle$ is a S-extension and $\gamma = \langle (a)(bc) \rangle$ is an I-extension. In order to sort two extension sequences, given two sequences β and γ such that both are S-extensions (or I-extensions) of a common prefix α, with items e_i and e_j respectively, we say that β precedes γ, $\beta < \gamma$, if $e_i <_{lex} e_j$ in a lexicographic order. If, on the contrary, one of them is a S-extension and the other one is an I-extension, the S-extension always precedes the I-extension.

Definition 3.1.1. We say that a sequence s has a **left-open interval** if there is at least one beginning boundary point e^+ in s which does not have its corresponding end boundary point after it. If s contains only the end boundary points of an interval, without a corresponding beginning boundary point before it, we say that s has a **right-open interval**. For example, in sequence $s_1 = \langle (a^+)(b^+)(a^-) \rangle$ b, is a left-open interval and sequence $s_2 = \langle (a^+)(b^+)(a^-)(c^+) \rangle$ has two left-open intervals (intervals b and c), while $s_3 = \langle (a^+)(b^-)(a^-) \rangle$ has a right-open interval (interval b) and $s_3 = \langle (b^+)(a^-) \rangle$ has both a left-open and right-open intervals (being b the left-open interval and a the right-open one). If a sequence s has an open interval (either left-open or right-open), we call it **improper sequence** and, conversely, if all of the intervals are not open, we say that s is a **proper sequence**.

In Figure 3.2, we show the final frequent set composed of proper sequences when we mine the example database with a minimum support of 2. We can see that there are different types of frequent sequences: those composed of both simple points and intervals (boundary points), for instance the 5-sequence $\langle (b^+)(c^+)(b^-)\ (d)(c^-) \rangle$; those formed only by intervals (boundary points), like the 4-sequence $\langle (a^+)(b^+)(a^-)(b^-) \rangle$; and those with only points, i.e. the 2-sequence $\langle (d)(g) \rangle$. In total, the final frequent sequence set has 33 frequent boundary point sequences, being two 7-sequences the largest ones.

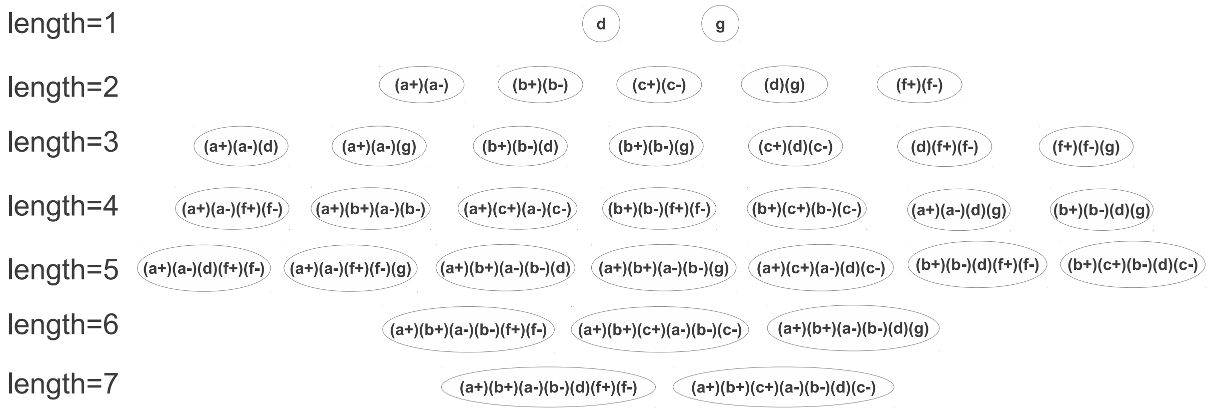


Figure 3.2: Frequent sequence set of example database.

Note that all the databases considered for the algorithms introduced in this chapter, PaGAPIS and FaSPIP, have previously been translated into databases with boundary point sequences.

3.2 PaGAPIS. New algorithm for points and intervals based on Pattern-Growth strategy

In this Section, we formulate and explain every step of our PaGAPIS algorithm. The algorithm consists in a continuous loop where for every frequent point: 1) that point is taken as a prefix, and a database projection is done with respect to it, and 2) a recursive call is done in order to find the pattern set derived from that prefix. Algorithm 1 shows the loop corresponding to the first step of the algorithm, while Algorithm 2 shows the following steps in a more detailed way. As we can see, this algorithm follows the same structure as PrefixSpan, in nature, but with some changes to be able to deal with our patterns. In our case, we need to add a new method to check the correspondence between boundary points, i.e. 1) we have to be sure that an end point that we choose in the algorithm has a corresponding left-open interval whose beginning point was previously chosen; and 2) we cannot insert right-open intervals in a pattern. In Algorithm 1, a projected database is obtained (line 4) for every frequent item e_i and the method PaGAPISLoop is called (Line 5). This last method searches, for each projection, the corresponding frequent patterns that belong to the branch of the search space that starts with e_i.

The method PaGAPISLoop is shown in Algorithm 2. In the first place (Lines 1-5), the frequent pattern set is initialized to just the pattern p, which is given as a parameter, only if p has all its intervals properly completed. Line 6 finds the set of frequent points $\mathcal{A}$ that can be found in the projected database $\mathcal{D}_p$. If $\mathcal{A}$ is not empty, for every point e in $\mathcal{A}$ (line 8) we get its corresponding projected database (line 10) and we execute a recursive call to method PaGAPISLoop (line 11), being its output saved (line 12). Finally, in Line 15, the method finishes returning the frequent pattern set found in the corresponding branch.

Algorithm 1 PaGAPIS

```
1: $\mathcal{F}_1$ = {frequent 1-sequences}
2: $\mathcal{FS} = \emptyset$
3: for all  i $\in \mathcal{F}_1$  do
4:    $\mathcal{D}_i$ = ProjectDatabase(i, $\mathcal{D}$)
5:    $\mathcal{F}_i$=PaGAPISLoop(i,$\mathcal{D}_i$)
6:    $\mathcal{FS} = \mathcal{FS} \cup \mathcal{F}_i$
7: end for
Ensure: The final frequent pattern set $\mathcal{FS}$
```

Algorithm 3 shows how the frequent points are found in each PaGAPISLoop execution. As in the original PrefixSpan algorithm, in a $\mathcal{D}_p$ we are interested in those points that occur at the same time as the last point of p and are greater than this last point in a lexicographic order, or in those points that appear after the last point of p (lines 2-4). Unfortunately, that works fine with point events but not with intervals. Furthermore, when we work with intervals, if the prefix p contains some left-open intervals ${e_i}^+$, we could take into account a point (either simple point or boundary point) that appears after the time where we should find the expected ${e_i}^-$ of those open intervals. Besides, we could also include a right-open interval. Both aspects are considered in Lines 5-6, that is, these lines avoid the generation of improper intervals. Finally, the final frequent point set is returned in line 8. Therefore, there can be two types of points: 1) those that occur after pattern p, and 2) those that appear as an extension of the last itemset of the pattern p. For instance, if the pattern $p = \langle (a^+)(b^+) \rangle$ has the point c^+ after it, and the point d^+ takes place at the same time as b^+, the two points c^+ and $*d^+$ are two candidates to be returned in the final point set.

Finally, Algorithm 4 shows how to do a database projection. The method is essentially the same as in the original PrefixSpan algorithm, but now, we avoid the generation of projections that start in a point that appears after an expected end point corresponding to an open interval. If we would consider those projections that start after an expected end point, all of the points that, lately, were found by the Algorithm *FindFrequentItems*

Algorithm 2 PaGAPISLoop($p, \mathcal{D}_p$)

```
Require: the current frequent pattern $p = (I_1, I_2, \ldots, I_n)$, the p-projected database $\mathcal{D}_p$
if (in $p$ it appears any interval, and each interval is complete, appearing both $e^+$ and $e^-$ ) then
    $\mathcal{F}_p = \{p\}$
else
    $\mathcal{F}_p = \emptyset$
end if
$\mathcal{A}$ = FindFrequentEvents($p$,$\mathcal{D}_p$)
if ($\mathcal{A} \neq \emptyset$) then
    for all valid $e \in \mathcal{A}$ do
        $p' = p \cup \{e\}$
        $\mathcal{D}_{p'}$ = ProjectDatabase($\alpha, \mathcal{D}_p$)
        $\mathcal{F}_{p'}$=PaGAPISLoop($p'$,$\mathcal{D}_{p'}$)
        $\mathcal{F}_p = \mathcal{F}_p \cup \mathcal{F}_{p'}$
    end for
end if
return $\mathcal{F}_p$
Ensure: The frequent pattern set $\mathcal{F}_p$
```

would have been discarded since they would not accomplish the conditions in lines 5-6 of Algorithm 3. Therefore, we can save some time if we prune these wrong projections.

Algorithm 3 FindFrequentItems($p, \mathcal{D}_p$)

```
Require: the current frequent pattern $p = (I_1, I_2, \ldots, I_n)$, the p-projected database $\mathcal{D}_p$
1: $\mathcal{A} = \emptyset$
2: scan $\mathcal{D}_p$ once, find every frequent event $e$ such that
3: (a) $p$ can be extended by $e$ to $p' = (I_1, I_2, \ldots, I_n \cup \{e\})$
4: (b) $p$ can be extended by $e$ to $p' = (I_1, I_2, \ldots, I_n, \{e\})$
5: AND $e$ appears before, or at the same time but is lexicographically less, than all the expected end points of the open intervals in $p$
6: AND if $e$ is an end point $e^-$, there exists an open interval $e^+$ in $p$
7: add every $\alpha$ to $\mathcal{A}$
8: return $\mathcal{A}$
Ensure: The frequent items set $\mathcal{A}$ found after $p$ in $\mathcal{D}_p$
```

In order to know if a point appears before, at the same time or after an expected end point, we need to add some extra information to the data structures. In our case, for every projected sequence, we maintain a queue with events sorted by end time. In this way, when project in a sequence and we take a beginning point of an open interval, we add its corresponding end time to the queue. If the queue already had other values, we would put the new value in its correct position. When then, we project a sequence $s = \langle I_1 I_2 \ldots I_n \rangle$ with an event e that occurs at time t, we find several possibilities:

1. t occurs before the first value of the queue: The point cannot be an end point since

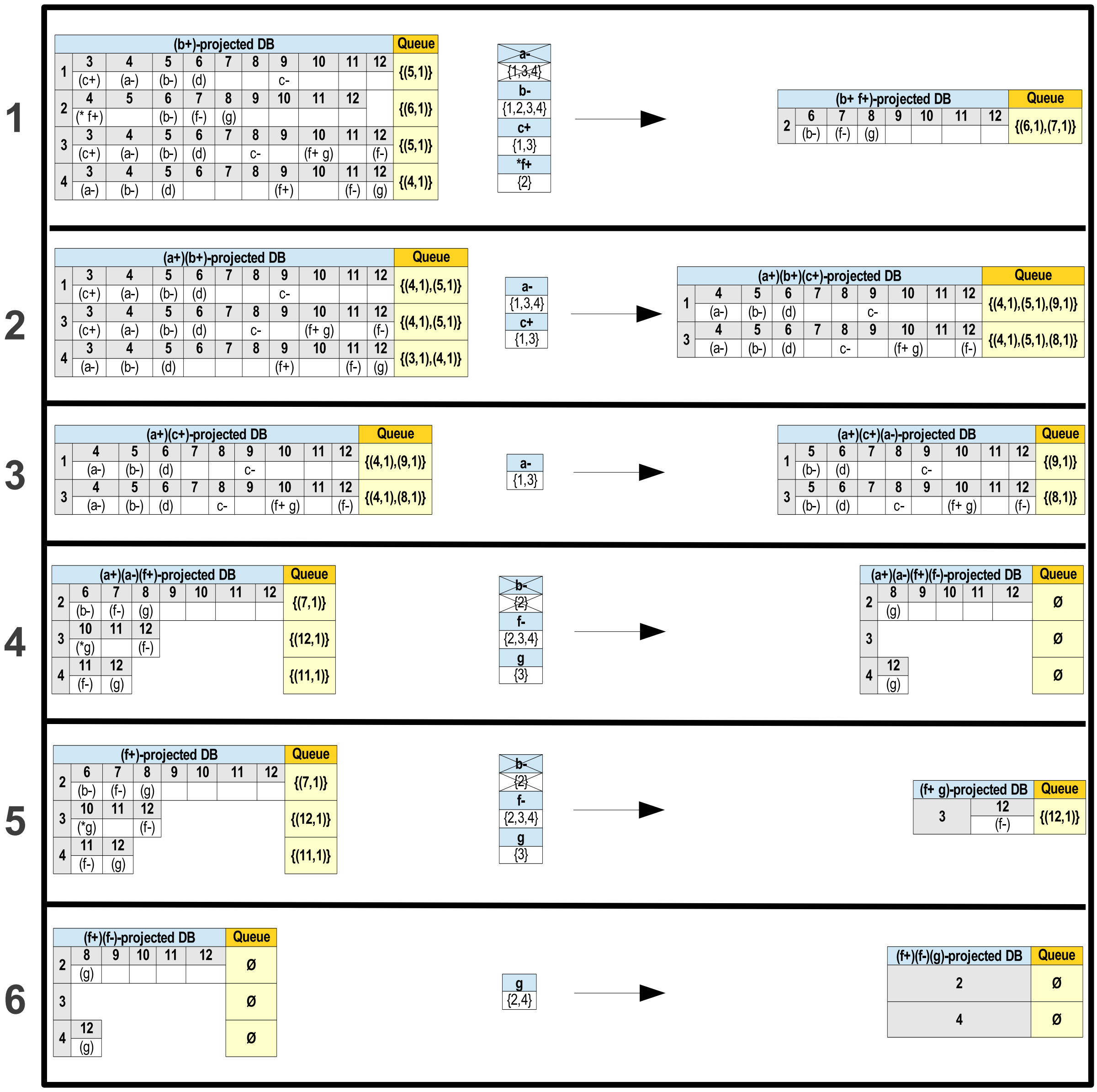
```
Algorithm 4 ProjectDatabase($\alpha, \mathcal{D}_p$)
Require: the current frequent boundary point $\alpha$, the p-projected database $\mathcal{D}_p$
$\mathcal{D}'_p = \emptyset$
for all $S_p \in \mathcal{D}_p$ do
    $S'_p = \emptyset$
    for all projection $s_p \in S_p$ do
        for all appearance of $\alpha$ in $s_p$ that appears before than any expected end point
        that corresponds to an open interval in $s_p$ do
            $s'_p$ = the subsequence that begins after item $\alpha$
            $S'_p = S'_p \cup s'_p$
        end for
    end for
    add $S'_p$ to $\mathcal{D}'_p$
end for
return $\mathcal{D}'_p$
Ensure: The p' projected database $\mathcal{D}'_p$
```

the first expected end point appears just at the time signalled by the first value of the queue. If e is a beginning point e^+ we add its corresponding end point time to the queue.

2. t occurs exactly at the same time as the first value of the queue: If e is a beginning point, and e is less, in a lexicographic order, than the corresponding event label denoted by the first value of the queue, then its corresponding expected end point time is stored in the queue. Otherwise, if e is greater than the event denoted by the first value of the queue, then e is discarded. If, on the contrary, e is an end point, e has to be just the corresponding end point that completes the open interval whose end time is the first value of the queue.

3. t occurs after the first value of the queue: e is always discarded since if we allow it, we would build improper patterns.

In Figure 3.3 we show six examples of different projections from several projected-databases (derived from the original database in Figure 3.1). In each row we see three tables. The one on the left is the database projection with a prefix (shown in the header of the table). In that table, we see the sequences (in the first projection of the example) with their events, the time of occurrence and the queue for each one. In the table in the middle, we show the frequent boundary items that appear before the first expected item, which is the first element of the queue (shown on the right part of the projected database). Besides, if any frequent boundary item, added to the pattern corresponding to the projected database, makes an improper sequence, we discard it. The discarded items are shown in the Figure by means of strike-through items. Finally, on the rightmost part, we show a new database projection derived from the previous database projection and a frequent item among those elements of the set previously depicted. Thus, in the Figure, we see how the algorithm works when different boundary points are found. For instance, in rows 1 and 2, we show what the algorithm does when we add a new boundary point corresponding to the beginning of an interval. In these situations, the queues grow with new expected elements. Rows 3 and 4 show the converse situation, when we extend a pattern of a projected database with a boundary point corresponding to the end of an interval. In these other cases, we check that these are the expected points in both queues, and we remove them from the queue. Finally, rows 5 and 6 show the behaviour when the algorithm deals with simple points. In these cases, the queues are not modified.



1

(b+)-projected DB											Queue
1	3	4	5	6	7	8	9	10	11	12	{(5,1)}
	(c+)	(a-)	(b-)	(d)			c-				
2	4	5	6	7	8	9	10	11	12		{(6,1)}
	(* f+)		(b-)	(f-)	(g)						
3	3	4	5	6	7	8	9	10	11	12	{(5,1)}
	(c+)	(a-)	(b-)	(d)		c-		(f+ g)		(f-)	
4	3	4	5	6	7	8	9	10	11	12	{(4,1)}
	(a-)	(b-)	(d)				(f+)		(f-)	(g)	

~~a-~~
~~{1,3,4}~~
b-
{1,2,3,4}
c+
{1,3}
*f+
{2}

(b+ f+)-projected DB								Queue
2	6	7	8	9	10	11	12	{(6,1),(7,1)}
	(b-)	(f-)	(g)					

2

(a+)(b+)-projected DB											Queue
1	3	4	5	6	7	8	9	10	11	12	{(4,1),(5,1)}
	(c+)	(a-)	(b-)	(d)			c-				
3	3	4	5	6	7	8	9	10	11	12	{(4,1),(5,1)}
	(c+)	(a-)	(b-)	(d)		c-		(f+ g)		(f-)	
4	3	4	5	6	7	8	9	10	11	12	{(3,1),(4,1)}
	(a-)	(b-)	(d)				(f+)		(f-)	(g)	

a-
{1,3,4}
c+
{1,3}

(a+)(b+)(c+)-projected DB										Queue
1	4	5	6	7	8	9	10	11	12	{(4,1),(5,1),(9,1)}
	(a-)	(b-)	(d)			c-				
3	4	5	6	7	8	9	10	11	12	{(4,1),(5,1),(8,1)}
	(a-)	(b-)	(d)		c-		(f+ g)		(f-)	

3

(a+)(c+)-projected DB										Queue
1	4	5	6	7	8	9	10	11	12	{(4,1),(9,1)}
	(a-)	(b-)	(d)			c-				
3	4	5	6	7	8	9	10	11	12	{(4,1),(8,1)}
	(a-)	(b-)	(d)		c-		(f+ g)		(f-)	

a-
{1,3}

(a+)(c+)(a-)-projected DB									Queue
1	5	6	7	8	9	10	11	12	{(9,1)}
	(b-)	(d)			c-				
3	5	6	7	8	9	10	11	12	{(8,1)}
	(b-)	(d)		c-		(f+ g)		(f-)	

4

(a+)(a-)(f+)-projected DB								Queue
2	6	7	8	9	10	11	12	{(7,1)}
	(b-)	(f-)	(g)					
3	10	11	12					{(12,1)}
	(*g)		(f-)					
4	11	12						{(11,1)}
	(f-)	(g)						

~~b-~~
~~{2}~~
f-
{2,3,4}
g
{3}

(a+)(a-)(f+)(f-)-projected DB						Queue
2	8	9	10	11	12	Ø
	(g)					
3						Ø
4	12					Ø
	(g)					

5

(f+)-projected DB								Queue
2	6	7	8	9	10	11	12	{(7,1)}
	(b-)	(f-)	(g)					
3	10	11	12					{(12,1)}
	(*g)		(f-)					
4	11	12						{(11,1)}
	(f-)	(g)						

~~b-~~
~~{2}~~
f-
{2,3,4}
g
{3}

(f+ g)-projected DB		Queue
3	12	{(12,1)}
	(f-)	

6

(f+)(f-)-projected DB						Queue
2	8	9	10	11	12	Ø
	(g)					
3						Ø
4	12					Ø
	(g)					

g
{2,4}

(f+)(f-)(g)-projected DB	Queue
2	Ø
4	Ø

Figure 3.3: Examples of several projection derived from projected databases.

3.3 FaSPIP. New algorithm for point and intervals based on Vertical Database format

In this Section, we change from the Pattern Growth strategy to the Vertical-Database Format strategy and introduce and explain every step of our FaSPIP algorithm. Algorithm 5 shows the pseudocode corresponding to the main lines of the algorithm. Firstly, FaSPIP keeps every frequent 1-sequence (Line 1), and secondly, for every frequent 1-sequence, the method *DFS-Explore* explores recursively the corresponding subtree in a depth-first search. The union of every resulting set $\mathcal{FS}_i$ corresponding to each frequent 1-sequence, besides the simple points that appear in $\mathcal{F}_1$, provide the final frequent pattern set $\mathcal{FS}$ composed of the proper boundary sequences that have at least *min_sup* appearances.

Algorithm 5 FaSPIP

```
1: $\mathcal{F}_1$ = {frequent 1-sequences}
2: $\mathcal{FS}$ = all the simple points in 1-sequences
3: **for all** i $\in \mathcal{F}_1$ **do**
4:     $\mathcal{F}_{ie}$ = {frequent 1-sequences greater than i}
5:     $\mathcal{FS}_i$=DFS-Explore(i,$\mathcal{F}_1$,$\mathcal{F}_{ie}$)
6:     $\mathcal{FS} = \mathcal{FS} \cup \mathcal{FS}_i$
7: **end for**
**Ensure:** The final frequent pattern set $\mathcal{FS}$
```

The method *DFS-Explore*, in Algorithm 6, executes recursively both the candidate generation (by means of I-extensions and S-extensions, explained in Section 3.1) and the support checking, returning a part of $\mathcal{FS}$ relative to the pattern p taken as parameter. The method takes as parameters two sets with the candidate items to do S-extensions and I-extensions respectively ($\mathcal{X}_S$ and $\mathcal{X}_I$ sets). The same process is done twice in the algorithm, executed for $\mathcal{X}_S$ in Lines 3-17 and for $\mathcal{X}_I$ in Lines 18-32. Firstly, in line 5, the correspondent extension is checked in order to know whether the new p' extension is already bad formed or not (this method is explained later). Second, the algorithm checks the frequency of p' extension (lines 6-8). If the extension p' is frequent, we include it in a $\mathcal{C}_s$ set (line 9) composed by the valid frequent patterns, and if all the intervals of p' are completed, we add p' to the final set of patterns, $\mathcal{L}_s$, (corresponding to the extension of pattern p, in line 11). Next, in line 17, the method *ExploreChildren* (Algorithm 7) is called, and there, *DFS-Explore* is executed for each new frequent S-extension. Lines 18-32 perform the same steps but with I-extensions. Finally, in line 33, the complete frequent pattern set (with a prefix p) is returned.

There are two main different changes added in FaSPIP with respect to SPADE: 1) the step to check if the subtree of a pattern can be skipped (lines 5 and 20 of Algorithm 6), and 2) the prune method that removes the non-valid occurrences of the patterns that appear in the IdList associated with a pattern p' (lines 8 and 23 of Algorithm 6).

The method *IsAPossiblePattern* prunes the improper extensions p' of a pattern p. There are two different cases to allow the prune: 1) if the last item of p' corresponds to a beginning interval boundary point with the same interval identifier of a previous open interval within the pattern p; and 2) if the last item of p' corresponds to an end interval

Algorithm 6 DFS-Explore(p, $\mathcal{X}_s$, $\mathcal{X}_I$)

```
Require: the current frequent pattern $p = (I_1, I_2, \ldots, I_n)$, set of items for S-extension $\mathcal{X}_S$, set of items for I-extension $\mathcal{X}_I$
 1: $\mathcal{X}_{\mathcal{S}temp} = \emptyset$, $\mathcal{X}_{\mathcal{I}temp} = \emptyset$
 2: $\mathcal{F}_b = \emptyset$,$\mathcal{C}_s = \emptyset$,$\mathcal{C}_i = \emptyset$, $\mathcal{L}_s = \emptyset$, $\mathcal{L}_i = \emptyset$
 3: for all e $\in \mathcal{X}_{\mathcal{S}}$ do
 4:     $p' = (I_1, I_2, \ldots, I_n, \{e\})$
 5:     if isAPossiblePattern($p'$) then
 6:         if ($p'$ is frequent) then
 7:             $\mathcal{X}_{\mathcal{S}temp} = \mathcal{X}_{\mathcal{S}temp} \cup \{i\}$
 8:             if ($p'$ appears at least $min\_sup$ times without exceeding any expected end point corresponding to a beginning
                point that appears in $p'$ that does not have its end point in $p'$) then
 9:                 $\mathcal{C}_s = \mathcal{C}_s \cup \{p'\}$
10:                 if (all of intervals appearing in $p'$ are completed) then
11:                     $\mathcal{L}_s = \mathcal{L}_s \cup \{p'\}$
12:                 end if
13:             end if
14:         end if
15:     end if
16: end for
17: $\mathcal{F}_b = \mathcal{F}_b \cup \mathcal{L}_s \cup$ ExploreChildren($\mathcal{C}_s$,$\mathcal{X}_{\mathcal{S}temp}$,$\mathcal{X}_{\mathcal{S}temp}$)
18: for all e $\in \mathcal{X}_I$ do
19:     $p' = (I_1, I_2, \ldots, I_n \cup \{i\})$
20:     if isAPossiblePattern($p'$) then
21:         if ($p'$ is frequent) then
22:             $\mathcal{X}_{\mathcal{I}temp} = \mathcal{X}_{\mathcal{I}temp} \cup \{i\}$
23:             if ($p'$ appears at least $min\_sup$ times without exceeding any expected end point corresponding to a beginning
                point that appears in $p'$ that does not have its end point in $p'$) then
24:                 $\mathcal{C}_i = \mathcal{C}_i \cup \{p'\}$
25:                 if (all of intervals appearing in $p'$ are completed) then
26:                     $\mathcal{L}_i = \mathcal{L}_i \cup \{p'\}$
27:                 end if
28:             end if
29:         end if
30:     end if
31: end for
32: $\mathcal{F}_b = \mathcal{F}_b \cup \mathcal{L}_i \cup$ ExploreChildren($\mathcal{C}_i$,$\mathcal{X}_{\mathcal{S}temp}$,$\mathcal{X}_{\mathcal{I}temp}$)
33: return $\mathcal{F}_b$
Ensure: The frequent pattern set $\mathcal{F}_b$ of this node and its children
```

Algorithm 7 ExploreChildren($\mathcal{P}$, $\mathcal{X}_s$, $\mathcal{X}_i$)

```
Require: The pattern set $\mathcal{P}$ which contains all the patterns whose children are going to be explored, the set of valid items
    $\mathcal{X}_s$ that generates the $\mathcal{P}$ set by means of S-extensions, the set of valid items $\mathcal{X}_i$ that generates the $\mathcal{P}$ set by means of
    I-extensions
 1: $\mathcal{F}_s = \emptyset$
 2: for all p $\in \mathcal{P}$ do
 3:     $\mathcal{I}$ = All of elements in $\mathcal{X}_i$ greater than the last item $e_i$ in $p$
 4:     $\mathcal{F}_s = \mathcal{F}_s \cup$ DFS-Explore($p$,$\mathcal{X}_s$,$\mathcal{I}$)
 5: end for
 6: return $\mathcal{F}_s$
Ensure: The frequent pattern set $\mathcal{F}_s$ for all of the patterns' children
```

boundary point with the same interval identifier of a previous completed interval within the pattern p. For instance, the pattern $p_1 = \langle (a^+)(a^+) \rangle$ corresponds to the first case above since there exist two beginning points for interval a without having an end point of a between them; and the pattern $p_2 = \langle (a^+)(b^-) \rangle$ corresponds to the second case since the end point b^- appears and p_2 does not have any beginning point b^+ before it.

In the step to prune those patterns whose support is under the given threshold, we need to consider some issues. We know that by means of an usual counting of support step (lines 6 and 21), all the frequent points that take place after (for S-extensions), or at the same time as the last item in a pattern p (for I-extensions), are considered as possible future extensions and used as $\mathcal{X}_S$ and $\mathcal{X}_I$ sets. The problem that we find with this counting is that, for some occurrences in the database, we are counting some items that occur after some expected items that can be a left-open interval in the current pattern. Thus, we need a new phase where those non-valid occurrences are not taken into account for the actual support counting. In order to add this new control step, we use a queue associated with the IdList of every pattern (we will detail the implementation of the IdLists later). Let us remember that these IdLists are the temporal structure that keep the Vertical Database Format representation, and they are also used to represent the pattern occurrences in the database and, besides, by means of the join operation between two different IdList we can simultaneously extend a pattern and count its support.

Finally, the last pruning method is implemented by means of the information already kept in the IdLists. In our case, we include all the necessary fields to keep a trace of what intervals are open and their temporal information (in order to check if the sequences are proper). The structure of the IdList is shown in Figure 3.4, where the different fields are:

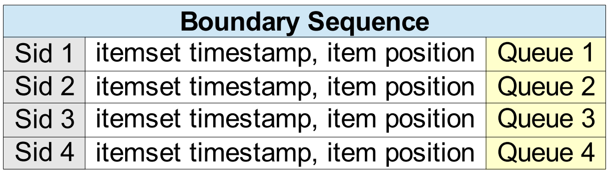
1. Sid corresponds to the sequence identifier in the database.

2. Itemset timestamp is the temporal occurrence of the last itemset in the sequence.

3. Item position refers to the relative position of the last item in the last itemset of the sequence.

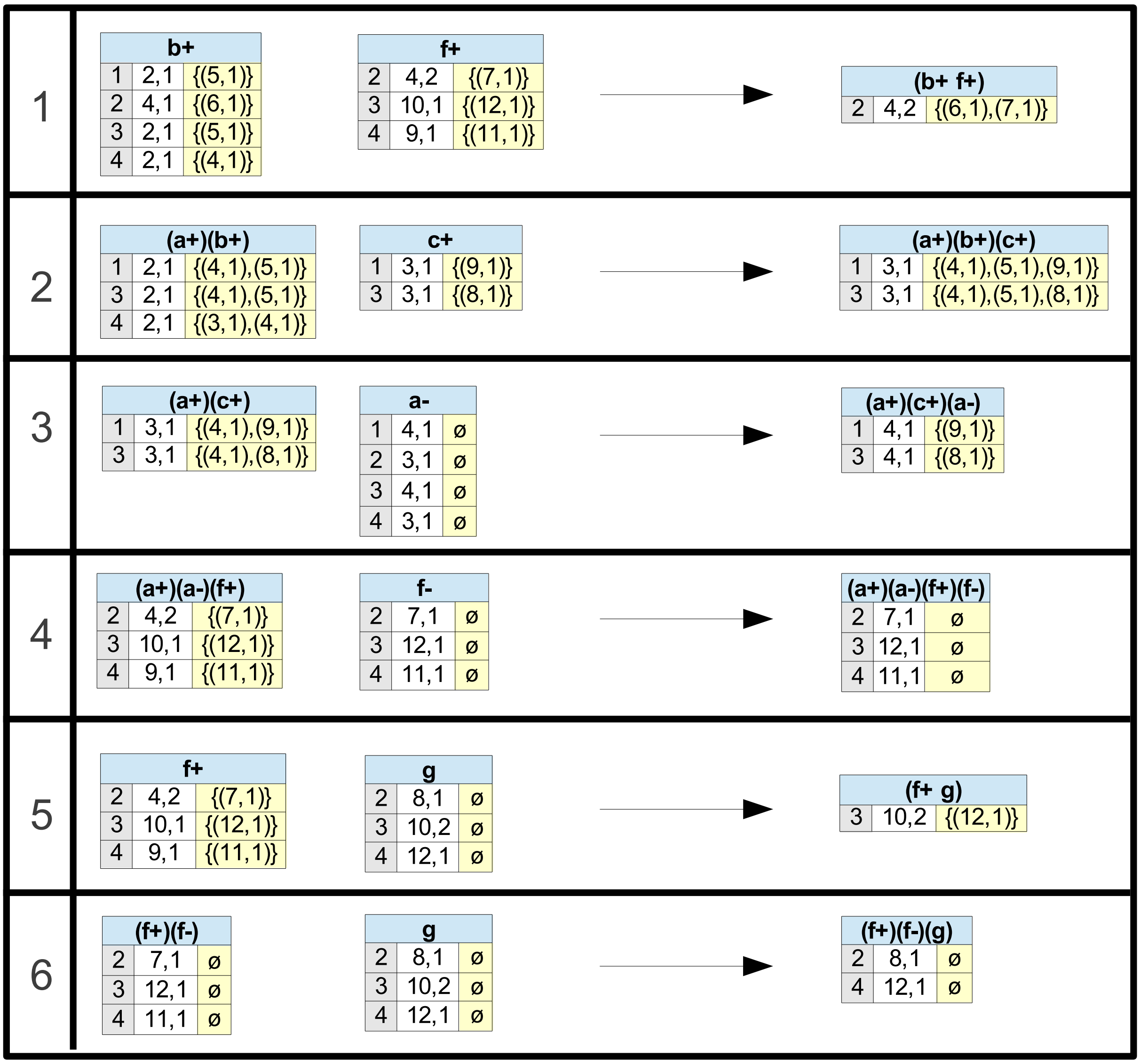
4. Queue is a sorted list that contains the time for the expected end points that refer to all the open intervals in the sequence. Each time position in the queue is a pair with the timestamp of the itemset and the item position in that itemset.

Regarding the join operation, as in SPADE, there are two kinds of join: the first one finds the occurrences of the second IdList that are after the first pattern; while the second one finds the occurrences in both first and second IdLists that occur at the same time. In our algorithm, the first join operation corresponds with S-extensions ($\mathcal{X}_S$) and the second one is identified with I-extensions ($\mathcal{X}_I$). In our case, besides checking both temporal relations, $<$ and $=$, from the first pattern (sequence) with respect the second pattern (item), we need to check whether the pair (itemset timestamp, position item) occurs before the first element in the queue of the checked sequence. For instance, in Figure 3.5, in the first block we can see the I-extension of the sequence $\langle(b^+)\rangle$ with the item f^+. They occur at the same time only in the second database sequence, occurring $\langle(b^+)\rangle$ in $(4,1)$ and f^+ in $(4,2)$. Since $(4,2)$ is less than the first element in the queue for the second sequence ($(4,2) < (6,1)$), we can make the I-extension to obtain the new sequence $\langle(b^+f^+)\rangle$. If, conversely, we make the S-extension for the same previous sequence and item, $\langle(b^+)\rangle$ and f^+, we see that there is no sequence where the condition of finding f^+ before the first element of the queue is accomplished (neither in the third sequence ($(10,1) < (5,1)$) nor in the fourth sequence ($(9,1) < (4,1)$)), and, therefore, the resulting S-extension cannot be done. In Figure 3.5 the examples already shown in Section 3.2 are considered, and, as before, several situations are introduced. Thus, rows 1 and 2 show the different IdList when we extend a boundary sequence with a boundary point corresponding to an interval beginning point; rows 3 and 4 show the extension of a boundary sequence with a boundary point corresponding to an interval end point of an open interval that was started in the previous boundary sequence; and, finally, rows 5 and 6 show the IdList when a boundary sequence is extended with a simple boundary point.



Boundary Sequence		
Sid 1	itemset timestamp, item position	Queue 1
Sid 2	itemset timestamp, item position	Queue 2
Sid 3	itemset timestamp, item position	Queue 3
Sid 4	itemset timestamp, item position	Queue 4

Figure 3.4: New structure for the IdLists used in the new way of count the support in order to find proper boundary sequences.



1

b+		
1	2,1	{(5,1)}
2	4,1	{(6,1)}
3	2,1	{(5,1)}
4	2,1	{(4,1)}

f+		
2	4,2	{(7,1)}
3	10,1	{(12,1)}
4	9,1	{(11,1)}

(b+ f+)		
2	4,2	{(6,1),(7,1)}

2

(a+)(b+)		
1	2,1	{(4,1),(5,1)}
3	2,1	{(4,1),(5,1)}
4	2,1	{(3,1),(4,1)}

c+		
1	3,1	{(9,1)}
3	3,1	{(8,1)}

(a+)(b+)(c+)		
1	3,1	{(4,1),(5,1),(9,1)}
3	3,1	{(4,1),(5,1),(8,1)}

3

(a+)(c+)		
1	3,1	{(4,1),(9,1)}
3	3,1	{(4,1),(8,1)}

a-		
1	4,1	ø
2	3,1	ø
3	4,1	ø
4	3,1	ø

(a+)(c+)(a-)		
1	4,1	{(9,1)}
3	4,1	{(8,1)}

4

(a+)(a-)(f+)		
2	4,2	{(7,1)}
3	10,1	{(12,1)}
4	9,1	{(11,1)}

f-		
2	7,1	ø
3	12,1	ø
4	11,1	ø

(a+)(a-)(f+)(f-)		
2	7,1	ø
3	12,1	ø
4	11,1	ø

5

f+		
2	4,2	{(7,1)}
3	10,1	{(12,1)}
4	9,1	{(11,1)}

g		
2	8,1	ø
3	10,2	ø
4	12,1	ø

(f+ g)		
3	10,2	{(12,1)}

6

(f+)(f-)		
2	7,1	ø
3	12,1	ø
4	11,1	ø

g		
2	8,1	ø
3	10,2	ø
4	12,1	ø

(f+)(f-)(g)		
2	8,1	ø
4	12,1	ø

Figure 3.5: Examples of several extensions of different boundary sequences.

3.4 Experimental results

A synthetic database generator with which to create datasets has been used in all the comparisons in order to control the experimental parameters. Several parameters have been set in this generator, whose abbreviations are shown in Table 3.2. Firstly, the generator generates the number of patterns previously set (with an average pattern length, all of which are totally consistent) and these patterns are then introduced to appear with a frequency of at least *min_sup* in the sequence database, *min_sup* being a previously established parameter. Finally all the sequences are completed in order to be adjusted to the initial configuration settings. Let us redefine the quotient of density of the database in function of these new parameters. Let us redefine the quotient of density (shown in Section 2.3) in function of these new parameters. Now, such density is the quotient between the average transaction length (ptl) and the number of different items that appear in the database (n), average density is now defined as $\delta = \frac{ptl}{n}$, whereas maximum density is $\delta = \frac{mtl}{n}$. We will consider this quotient in all our experiments, for comparing the performance of the two algorithms: PaGAPIS and FaSPIP.

Abbr.	Meaning
s	Number of sequences
psl	Average number of transactions in a sequence
msl	Maximum number of transactions in a sequence
ptl	Average items per transaction
mtl	Maximum number of items per transaction
ppl	Average length for the created frequent patterns
mpl	Maximum length for the created frequent patterns
n	Number of different items in the whole database
sup	Minimum support (minimal number of sequence appearances)

Table 3.2: Parameters for our synthetic database generator.

All the experiments were carried using 4-cores of 2.4GHZ Intel Xeon, running the Linux Ubuntu 12.10 Desktop edition. All the algorithms are implemented in Java 7 SE with a Java Virtual Machine of 12GB of main memory.

In Figure 3.6 we show the behaviour for moderated density values. In the 3.6A plot we can see the running time for the both algorithms PaGAPIS and FaSPIP when we mine a dataset with a density of 0.2 and with very few transactions per sequence (ptl). We can see that, for low support, PaGAPIS outperforms FaSPIP with no much difference. Also plots 3.6B and 3.6C show the behaviour for both algorithms when we increase a little the transaction average length parameter (ptl). Both plots show the behaviour with two settings for the number of different items (n) in the database (for values of 50 and 100). 3.6B plot shows a similar execution time for low supports when we have 50 items for the database. However, if we have a larger number of items, 100 in the given plot, we can see that FaSPIP is slower than PaGAPIS, especially for low supports, even though the density decreases. For all the three plots, the time executions are low and if we increase the *n* value, PaGAPIS improves its behaviour with respect to FaSPIP.

In Figure 3.7, we can see what happens when we increase the number of transactions per sequence and decrease the transaction average length. In summary, we can say that we have similar results to those obtained in Figure 3.6 even though we deal with sparser database (since the parameter *ptl* is less than previous Figure). In the 3.7A plot the execution times are very similar because the transaction average length has been reduce to 2 items per transaction and the drawbacks of PaGAPIS do not arise as important factor. Plots 3.7B and 3.7C depict the behaviour of both algorithms when we increase the transaction average length to 5 items. In that case, FaSPIP clearly outperforms PaGAPIS either when we consider 50 or 100 different items for the dataset. Besides,

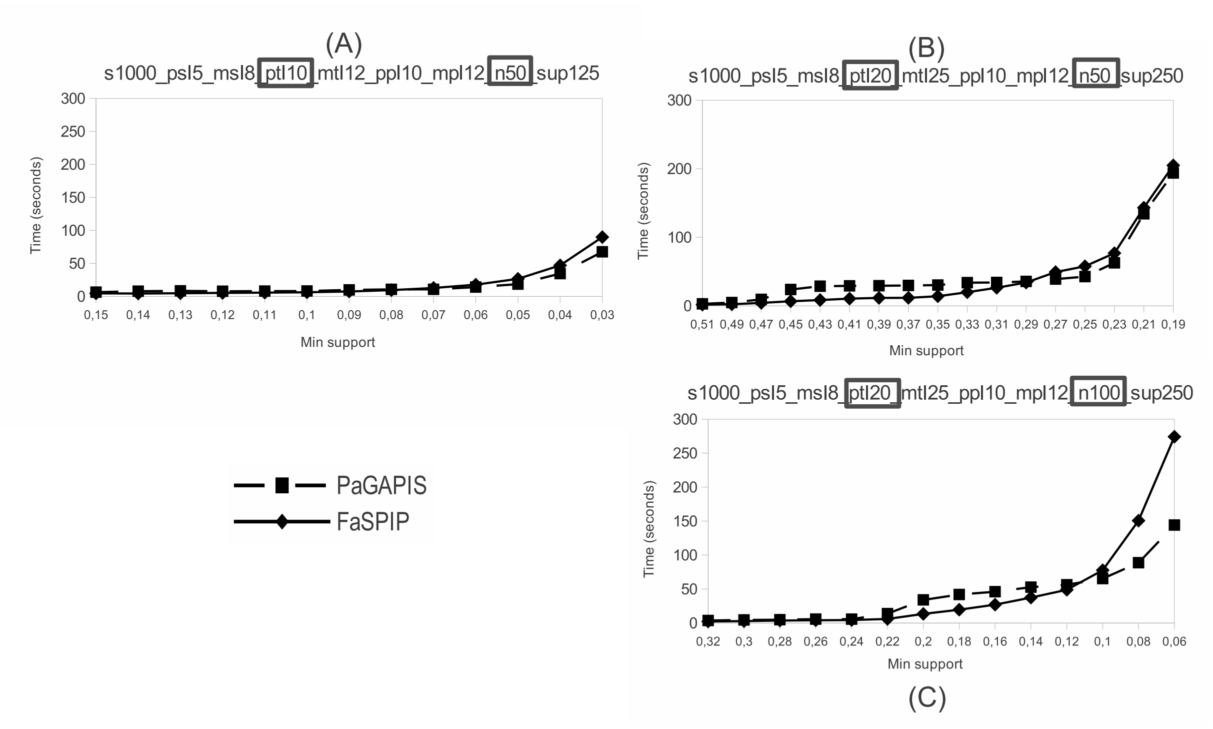


Figure 3.6: Varying support for datasets $s1000\_psl5\_msl8\_ptl10$–20_mtl12–25_ppl10 $\_mpl12$ $\_n50$–100.

we can see that the PaGAPIS behaviour for n=100 is different from that shown in plot Figure 3.6C. This phenomenon is mainly due to the fact of having a bigger number of transactions per sequence, what exposes the drawbacks of PaGAPIS. The 3.7D plot on the bottom right hand side shows that the difference between FaSPIP and PaGAPIS is even bigger when we increase more the transaction average length (10 items), considering 100 different items for the whole database. In this plot, the drawbacks of Pattern Growth strategy are present, but now, their relevance is higher than in plots on left hand side. In general, in this Figure, the main differences in time between the two algorithms appear before, with higher supports, unlike we showed in Figure 3.6.

In Figure 3.8 we can see the behaviour when we deal with larger sequences (40 transactions on average), with different transaction average length (5, 10 and 20) and big changes in the number of different items (50, 100, 500 and 1000), having sparser density values compared to those given in Figures 3.6 and 3.7. In plots 3.8A and 3.8B, FaSPIP clearly outperforms PaGAPIS. Of course, that difference is smaller for the sparser case (100 items). Plots 3.8C and 3.8D show the result with a dataset where the patterns inserted are quite longer than the cases above and there are 500 and 1000 different items, respectively. Therefore, we have dealt with sparser databases and the time execution has increased (770 and 701 seconds, respectively), and even in this case, due to the other setting parameters, FaSPIP still outperforms PaGAPIS. In addition, we can see a hard and sudden increase in the time execution in both plots, appearing at supports 0.45-0.43 in plot 3.8C and supports 0.38-0.32 in plot 3.8D. This is due to a big increase in the number of patterns for that support. Finally, plot 3.8E shows a sparse database (500 different items) with medium-high transaction length (20 items on average). In this case, we see

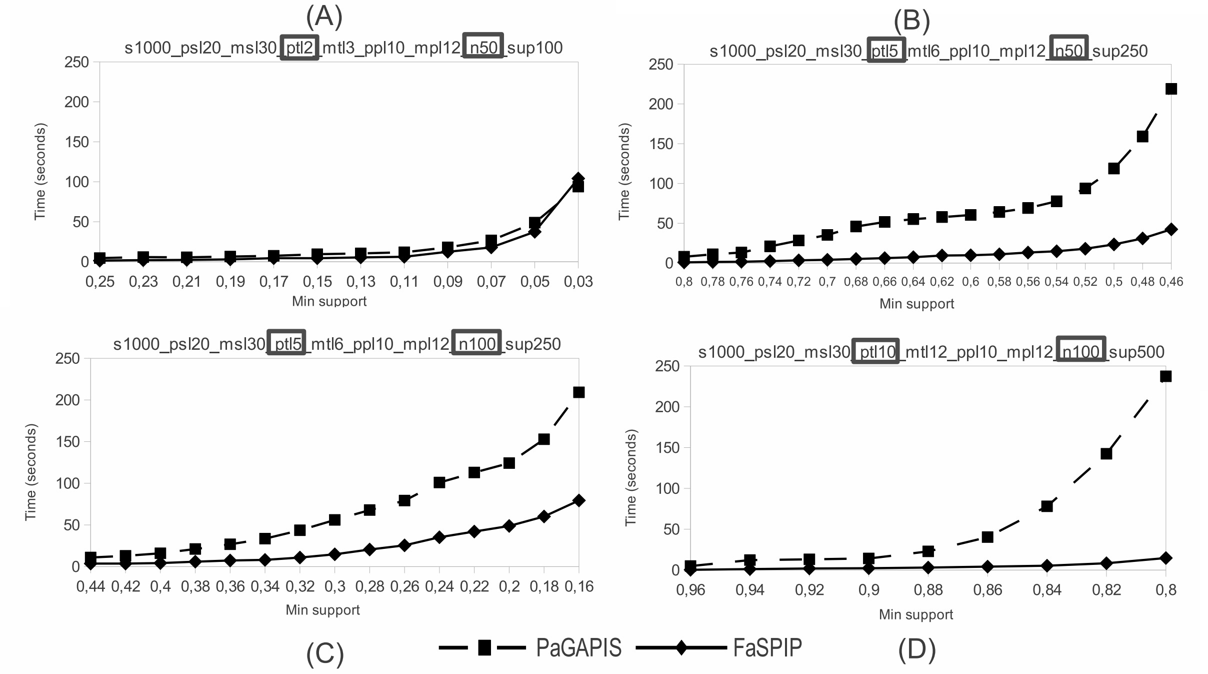


Figure 3.7: Varying support for datasets $s1000\_psl20\_msl30\_ptl2–5–10\_mtl3–6–12\_ppl$ 10 $\_mpl12\_n50–100$.

that the differences between the two algorithms appear before, at higher supports, due to a greater length in the patterns. Again, the lower the support the bigger the difference is, being always FaSPIP faster than PaGAPIS. Even though these differences increase before, the plot shows a similar behaviour to those shown in plots 3.8C and 3.8D.

Figure 3.9 show what happens with both algorithms in larger databases (10000 sequences) and with high values of different items (n = 500 and 1000). Plots 3.9A and 3.9B have medium sequence length (20 transactions on average), medium-high transaction length (30 items on average) and medium pattern length (10 items on average) and 500 and 1000 different items for the database. The dataset that correspond to the plots 3.9C and 3.9D have larger sequences (40 transactions on average) and smaller transactions (10 items on average). In all the plots, the domain of FaSPIP over PaGAPIS is evident. We can see that, having a configuration similar to that shown in Figure 3.7, we also have a similar behaviour. Besides, as we commented on previous Figure 3.8, all these plots show sudden change in their behaviour, having a big increase at concrete supports (support 0.7-0.69 in plot 3.9A, supports 0.67-0.65 in plot 3.9B, supports 0.59-0.57 in plot 3.9C and supports 0.67-0.65 in plot 3.9D), and, as before, this is also due to the big increment in the number of patterns that implies a higher processing time.

As we did in Section 2.3, we finally show the behaviour when we change those crucial parameters that especially affect to the algorithms execution. As before, we show the executions when we progressively change the "transactions per sequence" parameter, and one of those parameters that affect to the density, the number of items in this case. In Figure 3.10 we see what happens when we have a database with moderated parameters (1000 sequences, 20 transaction per sequence on average, and 12 items per transaction,

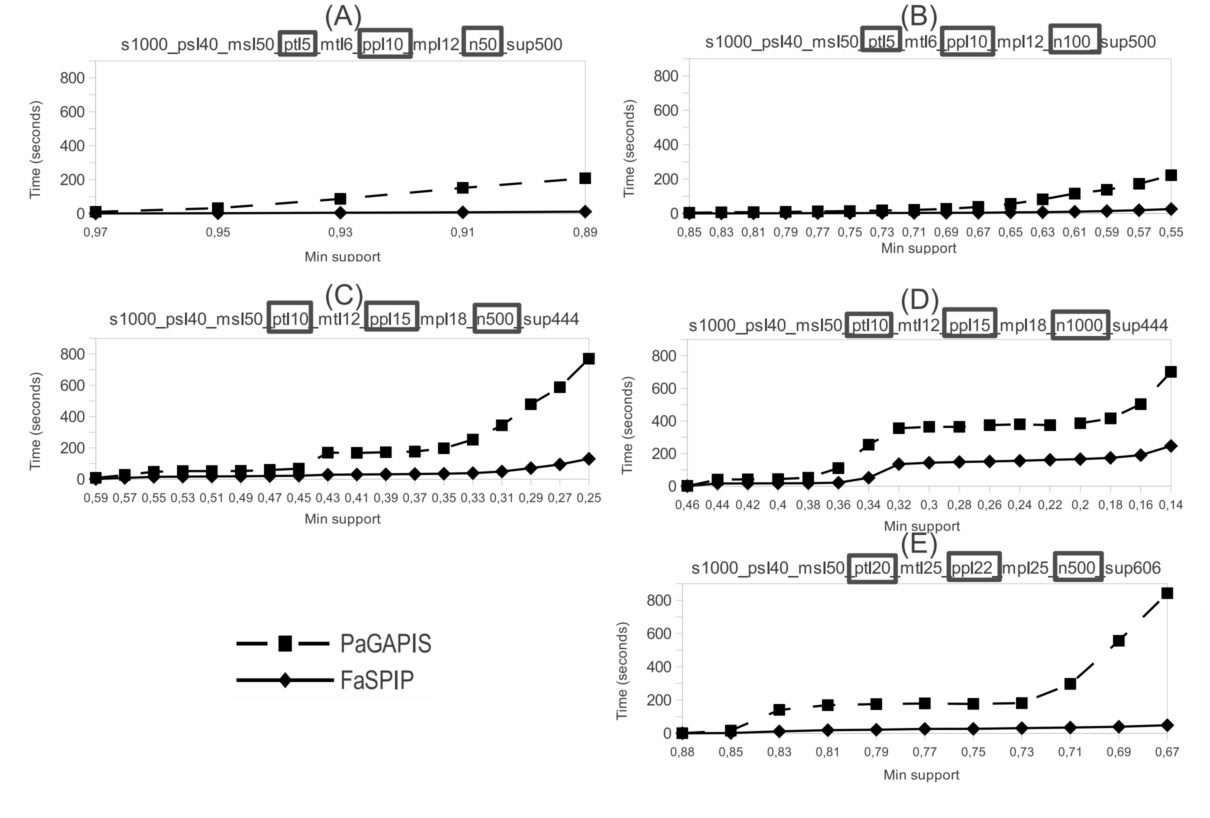


Figure 3.8: Varying support for datasets $s1000\_psl40\_msl50\_ptl5$–10–20$_mtl6$–12–25 $\_ppl10$– 15–22$_mpl12$–18–25$_n50$–100–500–1000.

being the length of the patterns of the database of 12 items on average) and we vary the number of different items (100 in plot 3.10A, 200 in plot 3.10B, 500 in plot 3.10C and 1000 in plot 3.10D). As before, the denser database, the better FaSPIP execution is and, the sparser database, the better PaGAPIS execution is. In the same way, Figure 3.11 shows the behaviour when we consider 100 different items per database, and we vary the "transactions per sequence" parameter (10 in plot 3.11A, 20 in plot 3.11B and 40 in plot 3.11C). Now, there is also a similar behaviour to that obtained in Section 2.3, when we worked with point-based database. However, now is so much difficult for PaGAPIS to obtain good results than for the point-based case due to the nature of the algorithm when we mine interval items. These new problems are widely explained in the next Section 3.5, comparing the benefits and drawbacks for the both new algorithms PaGAPIS and FaSPIP.

3.5 Discussion. Comparing both algorithms

In this Section we discuss the main advantages of FaSPIP with regard to PaGAPIS. In the first place, let us recall the main differences between the original versions of the Vertical Database Format and Pattern Growth algorithms on which FaSPIP and PaGAPIS are respectively based. We shall later show the behaviour of these algorithms once all the

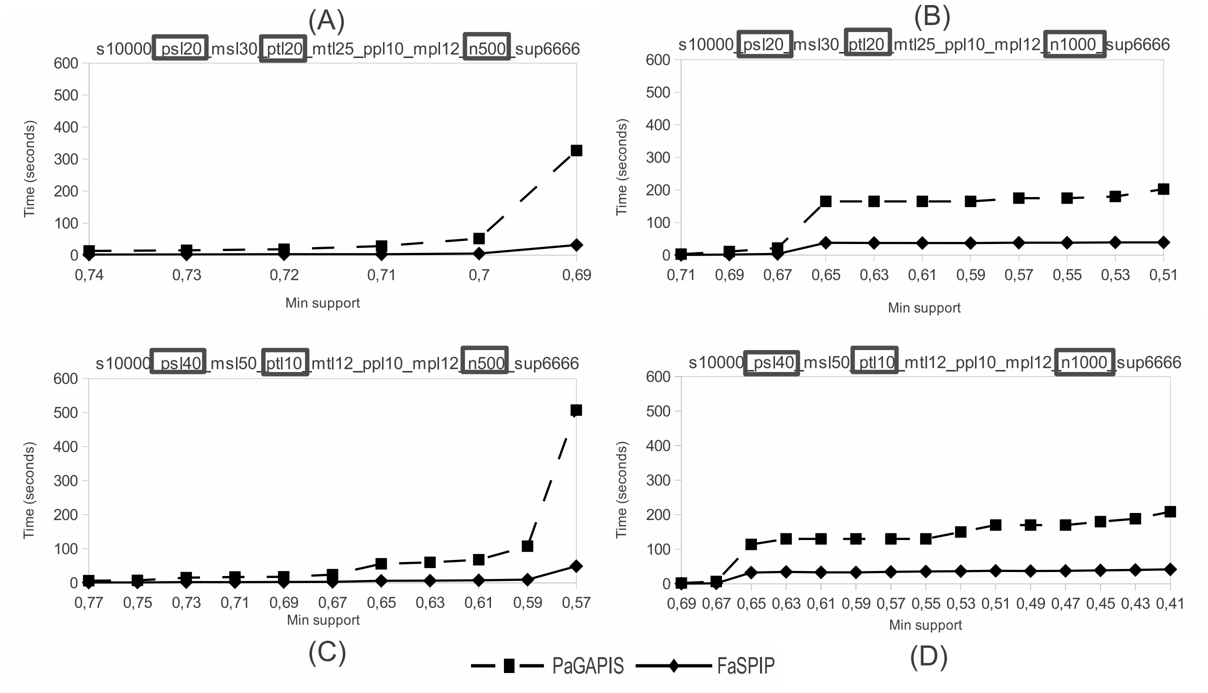


Figure 3.9: Varying support for datasets $s10000\_psl40\_msl50\_ptl10$–20_mtl12–25_ppl10 $\_mpl12\_n500$–1000.

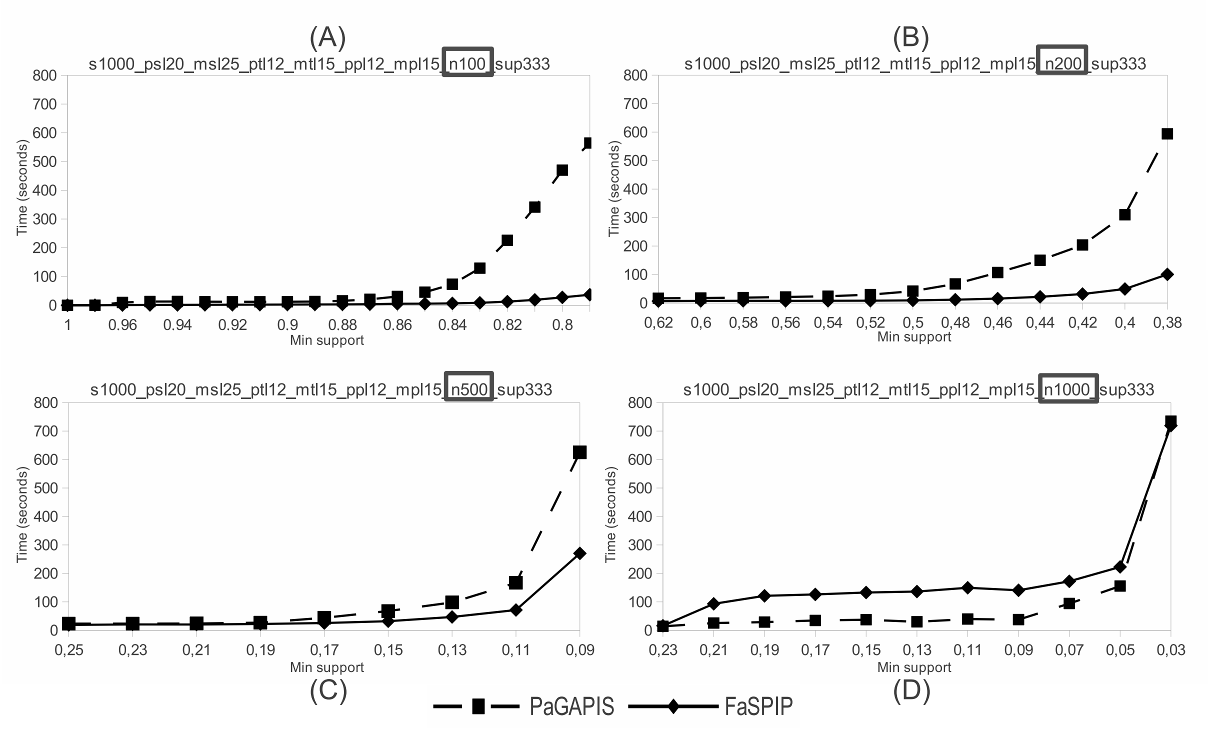


Figure 3.10: Varying support for a same configuration of datasets where we change the number of items (100, 200, 500 and 1000).

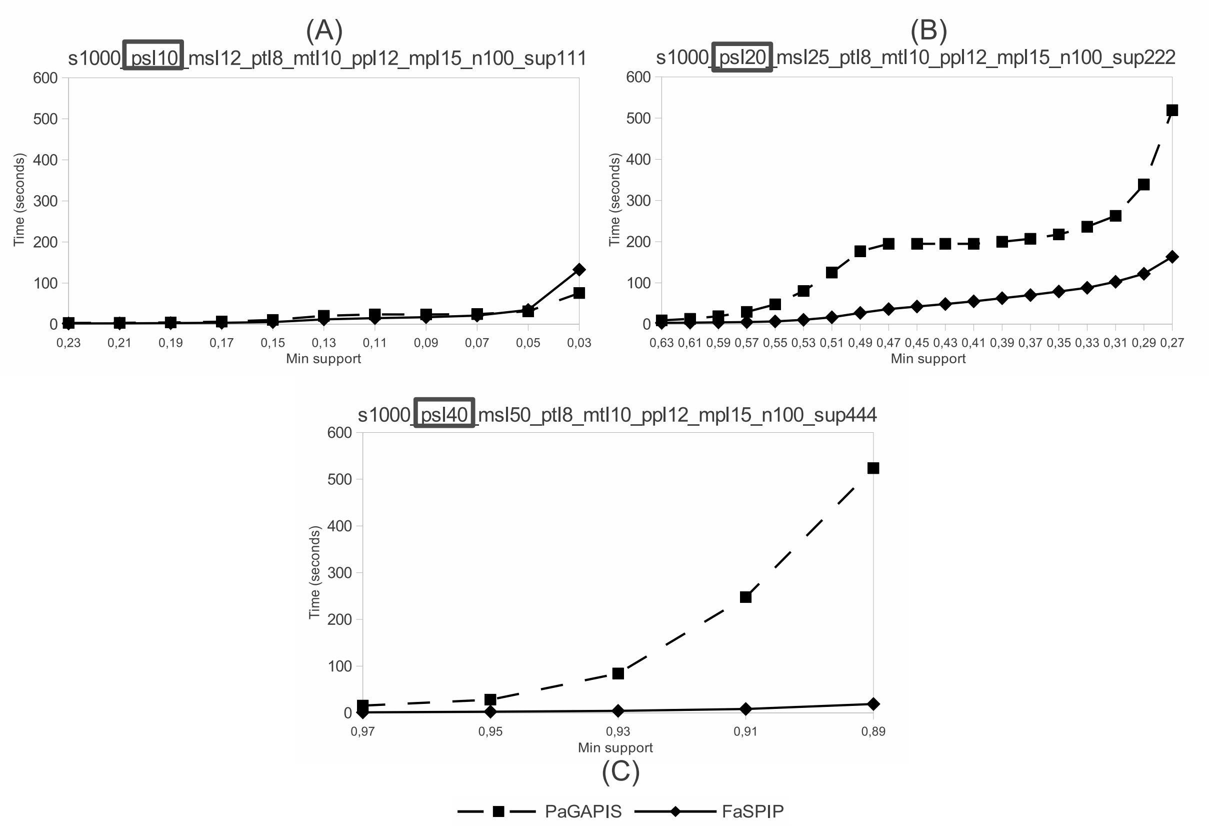
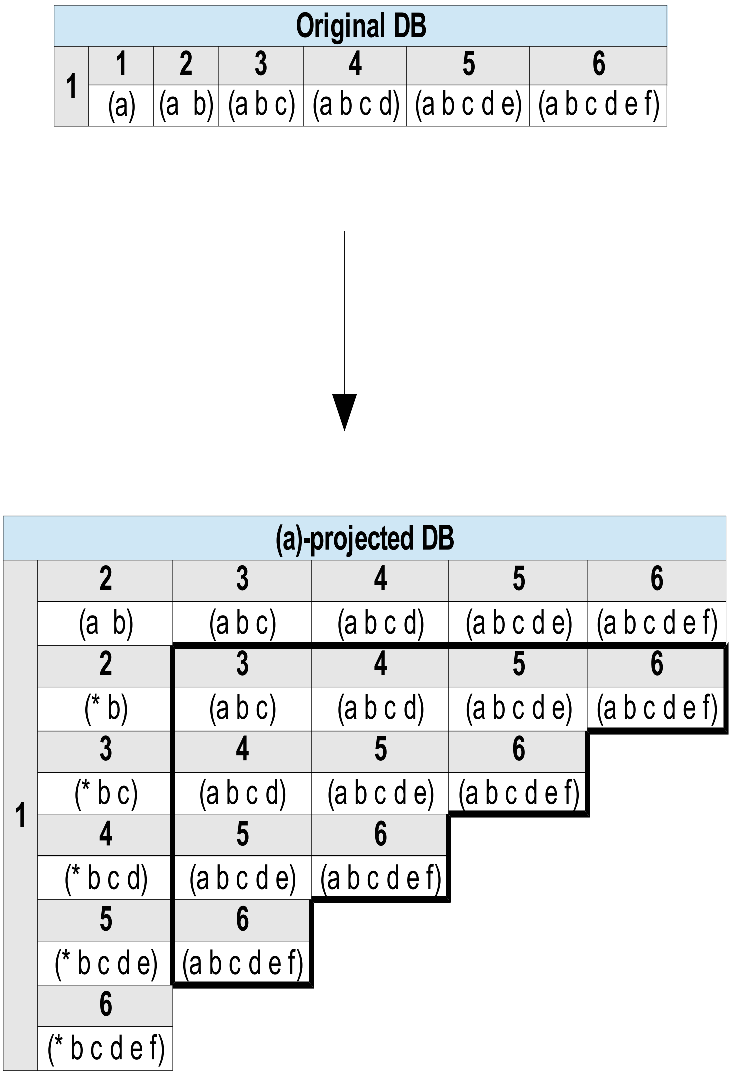


Figure 3.11: Varying support for a same configuration of datasets where we change the number of items per itemset (10, 20 and 40).

necessary information with which to simultaneously manage points and intervals and the pruning methods has been included, in order to verify that all the patterns contain proper intervals.

In some algorithms such as PrefixSpan, one of the situations overlooked is the presence of several appearances of the same item or event type in a sequence. On the one hand, in PrefixSpan it is necessary to make all the projections of an item that appears several times in a sequence in order to guarantee that all the I-extensions associated with that item will be discovered. If, conversely, it was not necessary to find the I-extensions, or if there were only one item per itemset, or only one appearance of each item per sequence existed, there would be sufficient information with a single projection. On the other hand, in Vertical Database Format algorithms, such as SPADE, it is not necessary to take into account the problem of PrefixSpan, and all the S-extension and I-extensions explore the whole search space. It could thus be said that the latter is less influenced by the database structure.

Clearly, this fact has an important impact in the efficiency of both algorithms for some database configuration. For example, let us view the behaviour of both PrefixSpan and SPADE in the case of the database composed of the single sequence $s = \langle (a)_1 (ab)_2 (abc)_3 (abcd)_4\ (abcde)_5 (abcdef)_6 \rangle$ and a $min\_sup = 1$. In this case, there are six itemsets, occurring between times 1 and 6. If we consider sequence $\langle (a) \rangle$, the six different projections associated with this sequence are shown in Figure 3.12. Note that if only the first projection is considered in our example, then items $*b$, $*c$, $*d$, $*e$ and $*f$ will not be considered to be frequent items, and in order to count these items it is necessary to take into account the first itemset of every subsequent projection. PrefixSpan must necessarily scan all the projections but, in the case of those projections after the first one, PrefixSpan must take into account only the first itemset and can ignore the remaining itemsets. Figure 3.12 highlights all the itemsets that can be ignored by PrefixSpan.



Original DB

	1	2	3	4	5	6
1	(a)	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)

(a)-projected DB

1	2	3	4	5	6
	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	2	3	4	5	6
	(* b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	3	4	5	6	
	(* b c)	(a b c d)	(a b c d e)	(a b c d e f)	
	4	5	6		
	(* b c d)	(a b c d e)	(a b c d e f)		
	5	6			
	(* b c d e)	(a b c d e f)			
	6				
	(* b c d e f)				

Figure 3.12: Projected database for the brief example with the standard PrefixSpan algorithm.

With regard to SPADE, and using the same example as above, this algorithm builds the associated IdLists (shown in Figure 3.13) without any special consideration. This point is one of the great advantages of the Vertical Database Format strategy over the Pattern Growth strategy for certain database configurations.

With regard to FaSPIP and PaGAPIS, the above characteristics also define the principal behaviour of both algorithms. However, as was shown in Sections 3.2 and 3.3, both algorithms have new changes with respect to the original SPADE and PrefixSpan algorithms in order to guarantee that all the boundary sequences are well-formed. Concretely,

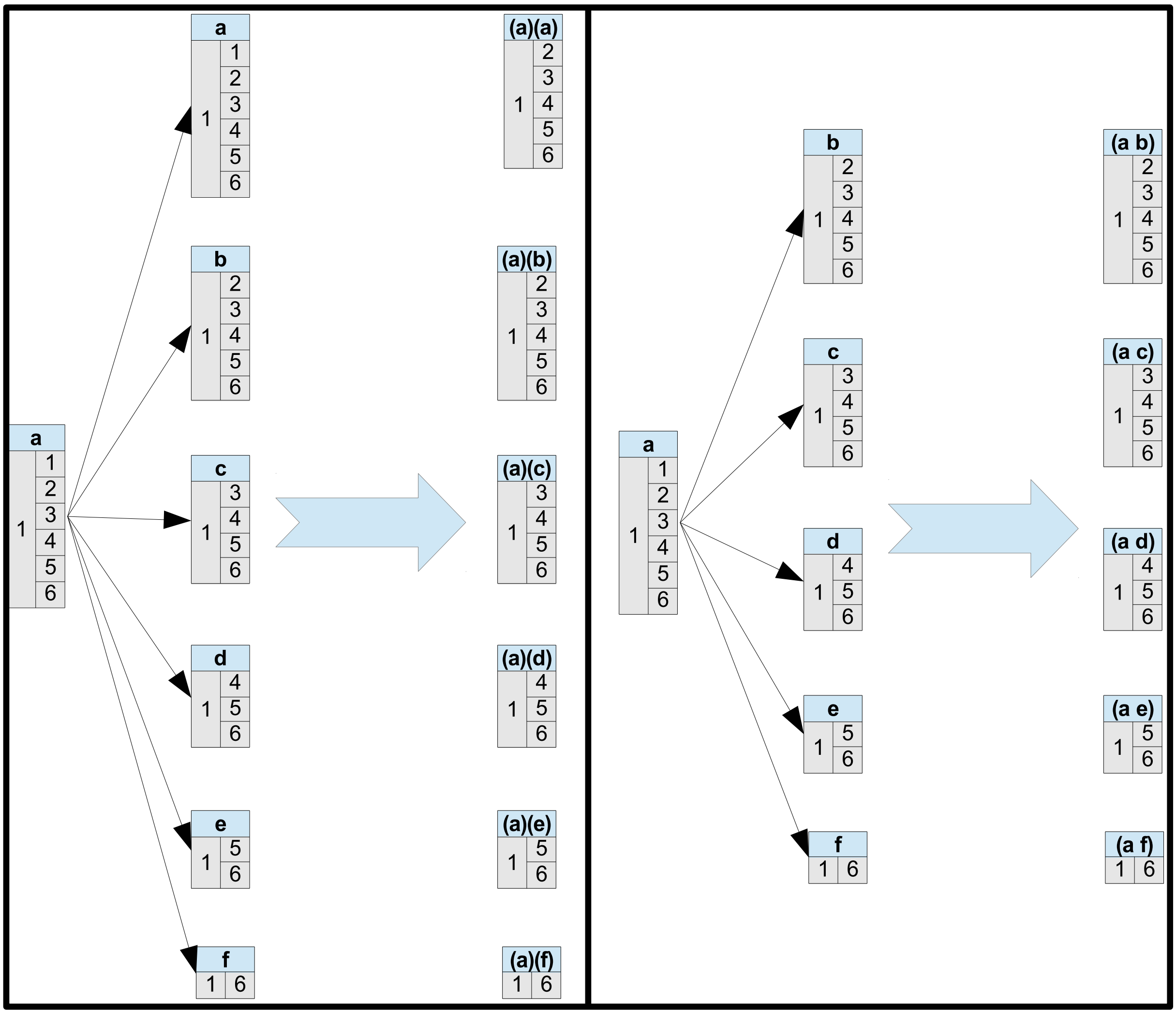
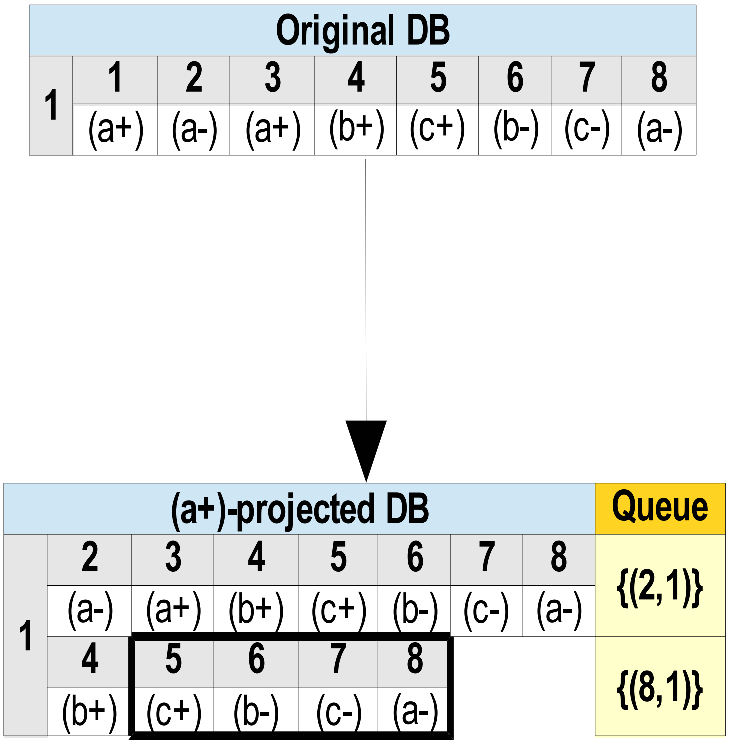


Figure 3.13: SPADE IdList for the brief example.

even when FaSPIP uses two new pruning mechanisms, the principal behaviour remains. However, the PaGAPIS algorithm has other new drawbacks, and while in the simple version of the original PrefixSpan it is possible to avoid exploring the part highlighted in Figure 3.12, since these relations are already taken into account in the first projection (relation before), PaGAPIS needs to explore them in case the event can be repeated in a sequence. When we discover intervals, it is not now possible to ignore that highlighted part because this would signify overlooking certain temporal relations, such as “meets”, “overlaps”, “contains”, “starts”, “is finished by” or “equals”.

For instance, let us suppose that the database is formed only of the sequence $\langle(a^+)(a^-)(a^+)(b^+)(c^+)(b^-)(c^-)(a^-)\rangle$ and $min\_supp = 1$. If we project this using the 1-sequence (a^+), in Figure 3.14 we can find the two projections associated with (a^+). If this example were to be processed, as in the PrefixSpan algorithm, we would take into account all the itemsets in the first projection and only the first itemset in the second projection. We would not therefore find the patterns $\langle(a^+)(b^+)(b^-)(a^-)\rangle$, $\langle(a^+)(c^+)(c^-)(a^-)\rangle$ and $\langle(a^+)(b^+)(c^+)(b^-)(c^-)(a^-)\rangle$, since in the first projection the elements are discarded after the itemset at time $t = 2$, which is precisely the moment at which the interval a has finished. In order to find the complete set of frequent boundary sequences, it is thus necessary to carry out a complete analysis of every itemset of every projection. This necessity is a new drawback for PaGAPIS as regards the original PrefixSpan that has to be introduced in order to mine proper intervals.



Original DB								
1	1	2	3	4	5	6	7	8
	(a+)	(a-)	(a+)	(b+)	(c+)	(b-)	(c-)	(a-)

(a+)-projected DB								Queue
1	2	3	4	5	6	7	8	{(2,1)}
	(a-)	(a+)	(b+)	(c+)	(b-)	(c-)	(a-)	
	4	5	6	7	8			{(8,1)}
	(b+)	(c+)	(b-)	(c-)	(a-)			

Figure 3.14: Projected database for the brief example with PaGAPIS algorithm.

Finally, after running the experiment in Section 3.5 we can say that, in general, FaSPIP is quite faster than PaGAPIS. The algorithms show similar results when we deal with sparse datasets with few itemsets per sequence but, nevertheless, the results are very favourable to FaSPIP when we address dense databases with several itemsets per sequence.

3.6 Conclusions

In this Chapter we have introduced two algorithms that deal with these issues for mining qualitative interval patterns. Our main contributions are as follow:

- We simplify the processing of relations among intervals by getting all the information of all boundary points in a sequence. The main advantage of this representation, compared with the representations commonly used, is that the relations among boundary points are reduced to *before*, *equals* and *after* instead of using the thirteen Allen's relations. We refer to this pattern representation as *sequence of interval boundaries*, and it expresses a pattern or sequence without any ambiguity, avoiding the common problems that occur in various existing representations. Thus, only by means of points we can express the relations among both the intervals and/or points that can appear in a pattern.

- We describe in detail the algorithm PaGAPIS (**Pa**ttern **G**rowth **A**lgorithm for **P**oint and **I**nterval **S**equences), an implementation based on Pattern Growth strategy, that uses a boundary point representation and is capable of finding the whole set of frequent patterns containing relations between points, intervals or points and intervals. A similar algorithm has been used in works [Mörchen and Fradkin, 2010; Chen et al., 2011]. Besides, we show the main changes that we have to add to original algorithm to enable the discovery of patterns with proper intervals.

- We introduce a novel algorithm, called FaSPIP, which stands for **Fa**st **S**trategy for **P**oints and **I**ntervals **P**atterns, capable of discovering the same final set of frequent patterns as PaGAPIS does. FaSPIP is based on the Vertical Database Strategy and, to the best of our knowledge, it is the first algorithm for points and intervals that it is based on that strategy. Besides, FaSPIP uses efficient methods to avoid the generation of useless candidates and to check the frequency of the candidates.

- We use in both FaSPIP and PaGAPIS algorithms a novel candidate generation based on boundary points instead of intervals. Besides, we describe in detail the different pruning mechanisms that we use to discard those sequences that are not correct.

- We make an exhaustive comparison between PaGAPIS and FaSPIP. We show that FaSPIP is more efficient and scalable and outperforms PaGAPIS in the most of the test performed.

- In general terms, the drawbacks that normally appear in Pattern Growth algorithms are highlighted in PaGAPIS even with a higher intensity. Furthermore, a pruning mechanism, as it is used in FaSPIP, is very convenient in order to reduce the search tree. Therefore, the previous two reasons coupled with a faster execution time of FaSPIP with respect to PaGAPIS, lead us to conclude that Vertical Database Format strategy is more appropriate for mining qualitative interval databases.

Chapter 4

PaGAPIMS and FaSPIMP: Two New Fast Algorithms for Mining Points and Intervals Quantitative patterns

In Chapter 2 we saw that Vertical Database Format algorithms show a better behaviour than Pattern Growth algorithms when they deal with dense databases with long itemsets. That reason motivates us to compare both strategies when we implement two efficient algorithms capable of finding frequent quantitative patterns with both points and intervals and to check if the domain of Vertical Database Format strategy still remains. Therefore, here we extend the algorithms exposed in Chapter 3 to mine quantitative patterns composed of both points and intervals. We call to these new algorithms PaGAPIMS and FaSPIMP and they follow the Pattern Growth and Vertical Database Format strategies, respectively. We maintain the boundary point representation but the expressiveness is increased by using quantitative relations.

In this work we define the quantitative relations as a quantification over the standard qualitative relations. We assume that the quantitative relations domain is discrete since all the temporal distances in the database are finite. For example, if we consider a database of behaviour of people, a customer goes shopping from one day to another concrete day or a patient is admitted a certain day in a hospital and he is discharged another certain day. Anyway, if we had a continuous domain we would have to discretize that domain before starting the mining process.

This Chapter is organized as follows. We include in Section 3.1 some definitions in order to take into account the temporal distances between boundary points. Sections 4.2 and 4.3 describe the new algorithms for points and intervals based on Pattern Growth strategy and Vertical Database Format strategy, respectively. In Section 4.4 we propose some optimizations in order to improve the performance of the algorithms. An experimental and performance study is presented in Section 4.5, while as a wide discussion about the behavioural differences of the algorithms is given in Section 4.6. Finally, we provide our conclusions in Section 4.7.

4.1 Additional definition and description for problem setting

Here we extend some previous definitions to include the temporal distances.

We call item-pair to the pair (t, i) that is associated with transaction T, where t is the T itemset time, and i is an item that appears in the itemset I.

In order to properly express the temporal distances in boundary point sequences that we need in quantitative patterns, we maintain the transaction time in our notations and we convert the absolute time to relative time with respect to the first itemset. This can simply achieved if we subtract to all the time itemset the value of the first itemset time. Thus, if we have a sequence like $\beta = \langle(t = 1, {A_1}^+)(t = 2, {B_1}^+)(t = 3, C^+)(t = 4, {A_1}^-)(t = 5, {B_1}^-)(t = 6, D)(t = 8, C^-)(t = 10, {A_2}^+{B_2}^+)(t = 12, {A_2}^-)(t = 13, {B_2}^-)\rangle$, we would convert it as $\beta = \langle(t = 0, {A_1}^+)(t = 1, {B_1}^+)(t = 2, C^+)$ $(t = 3, {A_1}^-)(t = 4, {B_1}^-)(t = 5, D)(t = 7, C^-)(t = 9, {A_2}^+{B_2}^+)(t = 11, {A_2}^-)(t = 12, {B_2}^-)\rangle$.

With this representation above, note that the temporal distances between events are implicit. Therefore, if we want to know what distance exists between two events or items, we have to derive it from the times associated with their respective itemsets. Besides, as their relative transaction times are present in the boundary point representation, two sequences completely equal in items and with the same itemset structure will be different if their itemset time are not the same. For instance $s_1 = \langle(t = 0, A^+)(t = 1, B^+)(t = 2, A^-)(t = 3, B^-)\rangle$ is different from the sequence $s_2 = \langle(t = 0, A^+)(t = 8, B^+)(t = 13, A^-)(t = 16, B^-)\rangle$. Note also that, since we use boundary points, the above definition of item-pair as a duple (t, i) now becomes (t, e) because all the boundary points have a duration of zero.

Consequently, considering this representation, the implementation of the subsequence checking, sequence extensions or the operation with prefix and suffix of sequence now has to change, but it is still straightforward. We say that the boundary point sequence α is a subsequence of another boundary point sequence β (or β is a supersequence of the α), denoted as $\alpha \preceq \beta$, if there exists a bijective function f that preserves the temporal distance and maps the boundary points in α to boundary points in β, in such a way that 1) $\forall e_i \in \alpha, \forall f(e_i) \in \beta, e_i \subseteq f(e_i)$ and 2) if $e_i < [t]e_j \Rightarrow f(e_i) < [t]f(e_j)$, $\forall e_i, e_j \in \alpha, \forall f(e_i), f(e_j) \in \beta$. Note that all the temporal distances between events in α must also hold in sequence β.

Regarding the prefix and suffix concepts, both definitions given in Section 3.1 are still valid but bearing in mind the new subsequence checking operation. Besides, we change the I-extension and S-extension definitions. In those concepts, while the first one occurs when an extension is done in the same last itemset of a sequence, S-extension add that new item in a new itemset. Nevertheless, in the quantitative approach both extensions are the same since we extend by distances, and the same meaning has extending by a distance of 0 or any other greater distance.

Therefore, we define an operation in order to extend a sequence with an item. We will need these operations in the candidate generation of our algorithms. Let $\alpha = \langle T_1T_2 \ldots T_n\rangle$ be a sequence and let $b_i \in \{e_i^+, e_i^-, e_i\}$ be a point. We call *extension* α' to the supersequence of α, extended with a new transaction containing a single point e_i that has a temporal distance t_e respect to the last transaction T_n, $\alpha' = \langle T_1T_2 \ldots T_nT_{n+1}\rangle$, $T_{n+1} =$

(t, b_i), where t is the associated time with T_n plus the temporal distance t_e that separates T_{n+1} from T_n. If the temporal distance that separates the last transaction of α is exactly 0, we add the item to the last transaction T_n of α. For instance, given the sequence $\alpha = \langle(t=0,a)(t=5,b)\rangle$ and a point $c \in \mathcal{I}$, the sequence $\beta = \langle(t=0,a)(t=5,b)(t=7,c)\rangle$ is an extension of time 2 and $\gamma = \langle(t=0,a)(t=5,bc)\rangle$ is extension of time 0.

In Figure 4.1, we show the final frequent set composed of proper sequences when we mine the example database with a minimum support of 2. We can see that there are different types of frequent sequences: those composed of both simple points and intervals (boundary points), for instance the 5-sequence $\langle(t=0,a^+)(t=1,b^+)(t=3,a^-)(t=4,b^-)(t=5,d)\rangle$; those formed by only intervals (boundary points), like the 4-sequence $\langle(t=0,a^+)(t=1,b^+)(t=3,a^-)(t=4,b^-)\rangle$; and those with only points, i.e. the 1-sequence $\langle(t=0,d)\rangle$. In total, the final frequent sequence set has 12 frequent boundary point sequences, being a 5-sequence the largest one.

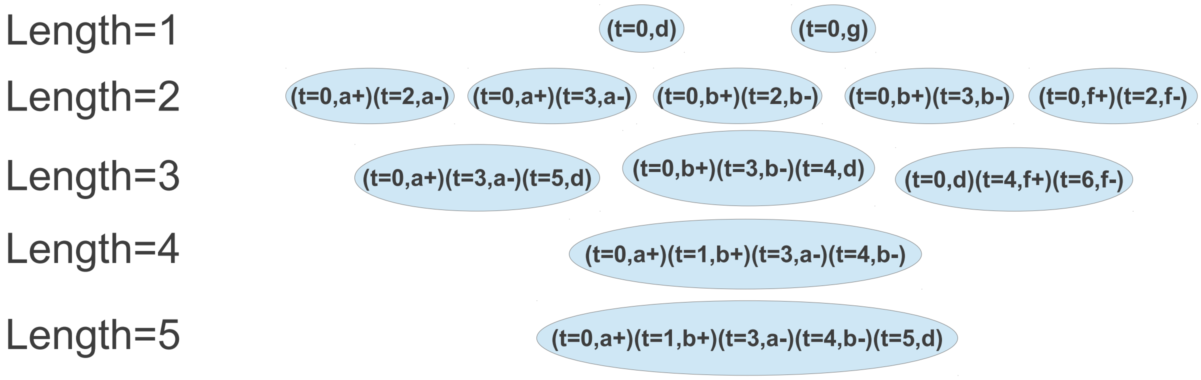


Figure 4.1: Frequent sequence set of example database.

4.2 PaGAPIMS. New algorithm for points and intervals based on Pattern-Growth format

In this Section, we formulate and explain every step of our PaGAPIMS algorithm. The algorithm consists in a continuous loop where for every frequent point: 1) that point is taken as a prefix, and a database projection is done with respect to it, and 2) a recursive call is done in order to find the pattern set derived from that prefix. Algorithm 8 shows the loop corresponding to the first step of the algorithm, while Algorithm 9 shows the following steps in a more detailed way. As we can see, this algorithm follows the same structure as PrefixSpan, in nature, but with some changes to be able to deal with our patterns. In our case, we need to add a new method to check the correspondence between boundary points, i.e. 1) we have to check that the point that we want to add occurs with exactly the temporal distance that we expect to find, 2) we have to be sure that an end point that we choose in the algorithm has a corresponding left-open interval whose beginning point was previously chosen; and 3) we cannot insert right-open intervals in a pattern. In Algorithm 8, a projected database is obtained (line 4) for every frequent item e_i and the method PaGAPIMSLoop is called (Line 5). This last method searches, for each projection, the corresponding frequent patterns that belong to the branch of the search space that start with e_i.

The method PaGAPIMSLoop is shown in Algorithm 9. In first the place (Lines 1-5), the frequent pattern set is initialized to just the pattern p, which is considered as parameter, only if p has all its intervals properly completed. Line 6 finds the set of frequent item-pairs $\mathcal{A}$ that can be found in the projected database $\mathcal{D}_p$. If $\mathcal{A}$ is not empty, for every item-point (t_e, e) in $\mathcal{A}$ (Line 8), we get their corresponding projected database (Line 14) and we execute a recursive call to the method PaGAPIMSLoop (Line 15), saving its returning frequent pattern set (Line 16). Finally, in Line 18, the method finishes returning the frequent pattern set found in the corresponding branch.

Algorithm 8 PaGAPIMS

```
$\mathcal{F}_1 = \{$frequent 1-sequences, being all of them as $(0, e)\}$
$\mathcal{FS} = \emptyset$
for all  i $\in \mathcal{F}_1$  do
    $\mathcal{D}_i$ = ProjectDatabase(i=$(0, e)$, $\mathcal{D}$)
    $\mathcal{F}_i$=PaGAPIMSLoop(i,$\mathcal{D}_i$)
    $\mathcal{FS} = \mathcal{FS} \cup \mathcal{F}_i$
end for
Ensure: The final frequent pattern set $\mathcal{FS}$
```

Algorithm 10 shows how the frequent item-pairs are found in the PaGAPIMSLoop. As in the original PrefixSpan algorithm, in a $\mathcal{D}_p$ we are interested in those points that occurs at a temporal distance of zero (those that appear at the same time as the last point of p) and are greater than the last item in a lexicographic order, or in those points that are at a distance greater than zero (they appear in transactions after the last point of p) (lines 2-4). Unfortunately, that works fine with point events but not with intervals. Furthermore, when we work with intervals, if our prefix p contains some left-open intervals ${e_i}^+$, for a specific projected sequence, if we would act as in the point case, we could take into account a point (either simple point or boundary point) that appears after the time where we found the expected ${e_i}^-$ that matches with those beginning points in p without their corresponding ${e_i}^-$. Besides, we could also include a right-open interval. Lines 5-6 avoid the generation of improper intervals and, finally, the final frequent item-pair set is returned in line 8. Therefore, there can be item-pairs with a time value of zero (those that appear as an extension of the last itemset of the pattern p), or with time values greater than zero (those that occur after pattern p). For instance, if the pattern $p = \langle(0, a^+)(2, b^+)\rangle$ has the point c^+ appearing at a distance of 5 after it, and the point d^+ occurs at the same time as b^+, the two item-pairs $(5, c^+)$ and $(0, d^+)$ are two candidates to be returned.

Finally, Algorithm 11 shows how to do a database projection. The method is essentially the same as in the original PrefixSpan algorithm, but now, we avoid the generation of projections that start in a point that appears after an expected end point corresponding to an open interval. If we would consider those projections that start after an expected end point, all of the item-pairs that, lately, were found by the Algorithm *FindFrequentItems* would have been discarded since they would not accomplish the conditions in lines 5-6 of Algorithm 10. Therefore, we can save some time if we prune these wrong projections.

Algorithm 9 PaGAPIMSLoop($p, \mathcal{D}_p$)

```
Require: the current frequent pattern $p = (T_1, T_2, \ldots, T_n)$, the p-projected database $\mathcal{D}_p$
if (in $p$ it appears any interval, and each interval is complete, appearing both $e^+$ and $e^-$ ) then
    $\mathcal{F}_p = \{p\}$
else
    $\mathcal{F}_p = \emptyset$
end if
$\mathcal{A}$ = FindFrequentEvents($p$,$\mathcal{D}_p$)
if ($\mathcal{A} \neq \emptyset$) then
    for all valid item-pair $\alpha = (t_e, e) \in \mathcal{A}$ do
        if $t_e = 0$ then {$e$ is added in the last transaction of $p$}
            $p' = (T_1, T_2, \ldots, T'_n = T_n \cup e)$
        else {$e$ is added in a new transaction $T_{n+1}$ that has a distance of $t_e$ with respect to $T_n$}
            $p' = (T_1, T_2, \ldots, T_n, T_{n+1} = (t_{T_n} + t_e, e))$
        end if
        $\mathcal{D}_{p'}$ = ProjectDatabase($\alpha = (t_e, e), \mathcal{D}_p$)
        $\mathcal{F}_{p'}$=PaGAPIMSLoop($p'$,$\mathcal{D}_{p'}$)
        $\mathcal{F}_p = \mathcal{F}_p \cup \mathcal{F}_{p'}$
    end for
end if
return $\mathcal{F}_p$
Ensure: The frequent pattern set $\mathcal{F}_p$
```

Besides, since we are searching for quantitative sequences, we are only interested in those points that have a distance of t_e with respect the beginning of the projected sequence, being the t_e value the temporal distance associated with the item-pair that we have as parameter.

Algorithm 10 FindFrequentItems($p, \mathcal{D}_p$)

```
Require: the current frequent pattern $p = (T_1, T_2, \ldots, T_n)$, the p-projected database $\mathcal{D}_p$
$\mathcal{A} = \emptyset$
scan $\mathcal{D}_p$ once, find every frequent event $e$ such that
(a) $p$ can be extended, at a distance of zero, by $e$ to $p' = (T_1, T_2, \ldots, T_n \cup \{e\})$
(b) $p$ can be extended, at a distance $t_e > 0$, by $e$ to $p' = (T_1, T_2, \ldots, T_n, \{(t_{T_n} + t_e, e)\})$
AND $e$ appears before, or at the same time but is lexicographically less, than all the expected end points of the open intervals in $p$
AND if $e$ is an end point $e^-$, there exists an open interval $e^+$ in $p$
add every $\alpha$ to $\mathcal{A}$
return $\mathcal{A}$
Ensure: The frequent items set $\mathcal{A}$ found after $p$ in $\mathcal{D}_p$
```

In order to know if a point appears before, at the same time or after an expected end point, we need to add some information to the data structures. In our case, for

Algorithm 11 ProjectDatabase($\alpha, \mathcal{D}_p$)

```
Require: the current frequent item-pair $\alpha = (t_e, e)$, the p-projected database $\mathcal{D}_p$
$\mathcal{D}'_p = \emptyset$
for all $S_p \in \mathcal{D}_p$ do
    $S'_p = \emptyset$
    for all projection $s_p \in S_p$ do
        for all appearance of $e$ in $s_p$ that appears before than any expected end point
        that corresponds to an open interval in $s_p$, and it appears with a distance of $t_e$
        with respect to the beginning of the projected sequence $s_p$ do
            $s'_p$ = the subsequence that begins after item $\alpha$
            $S'_p = S'_p \cup s'_p$
        end for
    end for
    add $S'_p$ to $\mathcal{D}'_p$
end for
return $\mathcal{D}'_p$
Ensure: The p' projected database $\mathcal{D}'_p$
```

every projected sequence, we maintain a queue with events sorted by end time. In this way, when we make a projection in a sequence and we take a beginning point of an open interval, we add its corresponding end time to the queue according to the time value. If previously the queue already had other values, we would put the new value in its correct position. When then, we make a projection of a sequence $s = \langle T_1 T_2 \ldots T_n \rangle$ with an event e that occurs at time t, we find several possibilities:

1. t occurs before the first value of the queue: The point cannot be an end point since the first expected end point appears just at the time signalled by the first value of the queue. If e is a beginning point e^+ we add its corresponding end point time to the queue.

2. t occurs exactly at the same time as the first value of the queue: If e is a beginning point, e and e is less in a lexicographic order than the corresponding event label denoted by the first value of the queue, then its corresponding expected end point time is stored in the queue. Otherwise, if e is greater than the event denoted by the first value of the queue, then e is discarded. If, on the contrary, e is an end point, e has to be just the corresponding end point that completes the open interval whose end time is the first value of the queue.

3. t occurs after the first value of the queue: e is always discarded since if we had allowed it, we would have built improper patterns.

In Figure 4.2 we show six examples of different projections from several projected-databases (derived from the original database in Figure 2.13). In each row we see three tables. The one on the left is the database projection with a prefix (shown in the header of the table). In that table, we see the sequences (in the first projection of the example) with their events, the time of occurrence and the queue for each one. In the table in the middle, we show the frequent boundary items-pairs that appear before the first expected item, which is the first element of the queue (shown on the right part of the projected database). Besides, if any frequent boundary item, added to the pattern corresponding to the projected database, makes an improper sequence, we discard it. The discarded items are shown by means of strike-through items. Finally, on the rightmost part, we show a new database projection derived from the previous database projection and a frequent item among those elements of the set previously depicted. Thus, in the Figure, we see how the algorithm works when different boundary points are found. For instance, in rows 1 and 2, we show what the algorithm does when we add a new boundary point corresponding to the beginning of an interval. In these rows, the queues grow with new expected elements. Rows 3 and 4 show the converse situation, when we extend a pattern of a projected database with a boundary point corresponding to the end of an interval. In these other cases, we check that these are the expected points in both queues, and we remove them from the queue. Finally, rows 5 and 6 show the behaviour when the algorithm deals with simple points. In these cases, the queues are not modified.

4.3 FaSPIMP algorithm. New algorithm for point and intervals based on Vertical Database format

In this Section, we change from the Pattern Growth strategy to the Vertical-Database Format strategy and introduce and explain every step of FaSPIMP algorithm. Algorithm 12 shows the pseudocode corresponding to the main lines of the algorithm. Firstly, FaSPIMP keeps every frequent 1-sequence (line 1) and gets all the possible item-pairs related to each frequent 1-sequence by means of the method *getNeighbours*; and secondly, for every frequent 1-sequence, the method *DFS-Explore* explores recursively the corresponding subtree in a depth-first search. The union of every resulting set $\mathcal{FS}_i$ corresponding to each frequent 1-sequence, besides the simple points that appear in $\mathcal{F}_1$, provide the final frequent pattern set $\mathcal{FS}$ composed of the proper boundary sequences that have at least *min_sup* appearances.

Algorithm *getNeighbours* is responsible for finding all the possible item-pairs for the different items in the original database that have any temporal distance with respect to each frequent 1-sequence. The algorithm finds all the possible candidates in the different sequences by measuring the temporal distance to the frequent 1-sequence that we consider.

The method *DFS-Explore*, in Algorithm 14, executes recursively both the candidate generation (by means of extensions, explained in Section 4.1) and the support checking, returning a part of $\mathcal{FS}$ relative to the pattern p taken as parameter. The method takes as parameters a set with the candidate items to do extensions ($\mathcal{X}$ set), and is executed in lines 3-20. Firstly, in line 9, the correspondent extension is checked in order to know whether the new p' extension is already bad formed or not (this method is explained later). Second, the algorithm checks the frequency of p' extension (lines 10-12). If the extension p' is frequent, we include it in a $\mathcal{C}$ set (line 13) composed by the valid frequent patterns, and if all the intervals of p' are completed, we add p' to the final set of patterns, $\mathcal{L}$, (corresponding to the extension of pattern p, in line 15). Next, in line 21, the method *ExploreChildren* (Algorithm 15) is called, and there, *DFS-Explore* is executed for each

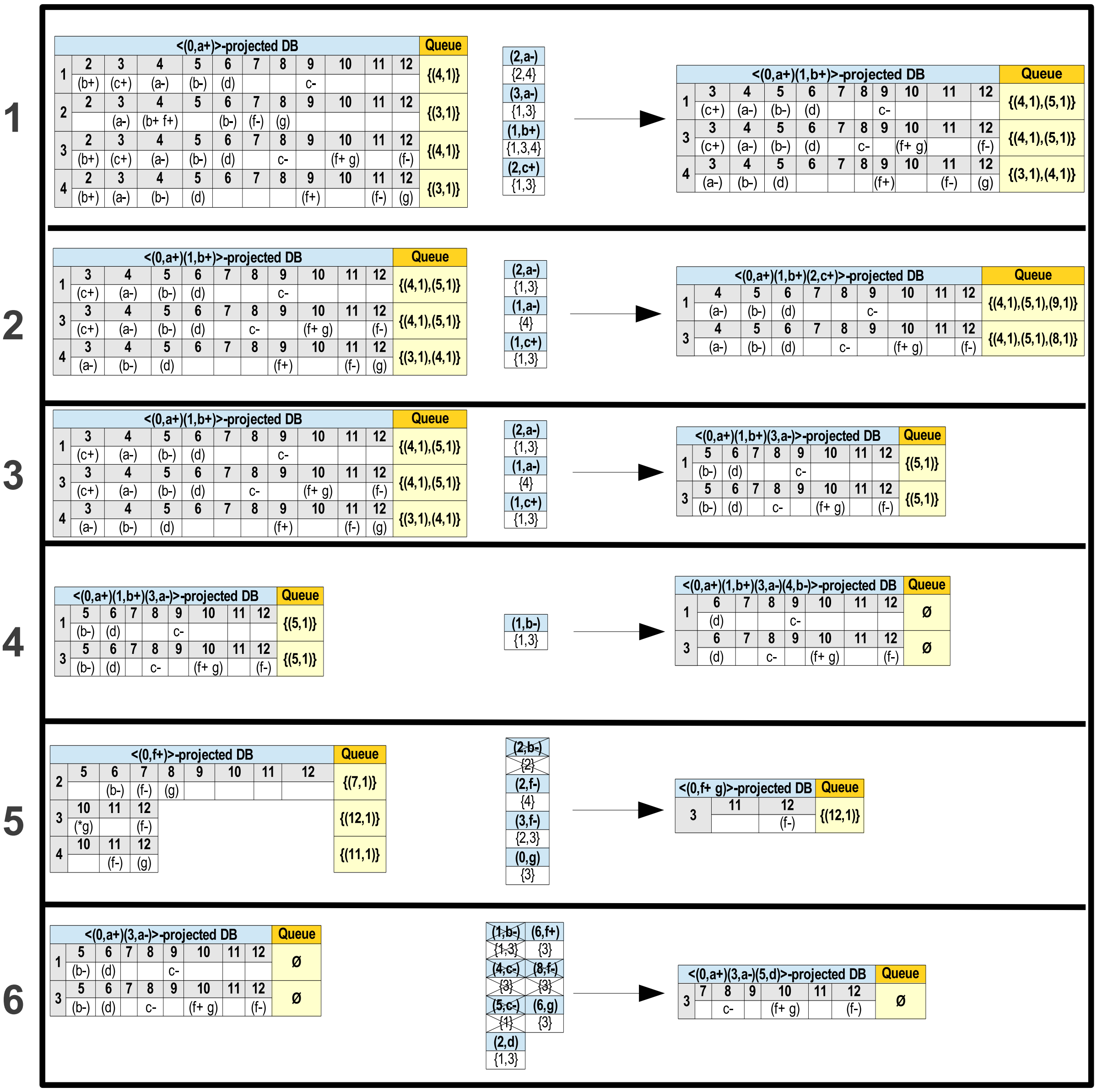


Figure 4.2: Examples of several projection derived from projected databases.

new frequent extension. Finally, in line 22, the complete frequent pattern set (with a prefix p) is returned.

There are two main different changes added in FaSPIMP with respect to SPADE: 1) the step to check if the subtree of a pattern can be skipped (line 9 of Algorithm 14), and 2) the prune method that removes the non-valid occurrences of the patterns that appear in the IdList associated with a pattern p' (line 12 of Algorithm 14).

The method *IsAPossiblePattern* prunes the improper extensions p' of a pattern p. There are two different cases that allow the prune: 1) if the last item of p' corresponds to a beginning interval boundary point with the same interval identifier of a previous open interval within the pattern p; and 2) if the last item of p' corresponds to an end interval boundary point with the same interval identifier of a previous completed interval within the pattern p. For instance, the pattern $p_1 = \langle(0, a^+)(12, a^+)\rangle$ corresponds to the first

Algorithm 12 FaSPIMP

```
$\mathcal{F}_1 = \{$frequent 1-sequences$\}$
$\mathcal{FS} = \emptyset$
$\mathcal{FN}$ = getNeighbours($\mathcal{F}_1$,$\mathcal{F}_2$)
**for all** $p_i = (0, e) \in \mathcal{F}_1$ **do**
    **if** $p_i$ is a simple point **then**
        $\mathcal{FS} = \mathcal{FS} \cup p_i$
    **end if**
    $\mathcal{FN}_i$= the ith-element of $\mathcal{FN}$
    $\mathcal{FS}_i$=DFS-Explore($p_i$,$\mathcal{FN}_i$)
    $\mathcal{FS} = \mathcal{FS} \cup \mathcal{FS}_i$
**end for**
**Ensure:** The final frequent pattern set $\mathcal{FS}$
```

Algorithm 13 getNeighbours($\mathcal{F}_1$)

```
**Require:** The frequent 1-sequence set $\mathcal{F}_1$
$\mathcal{FN}$=$\emptyset$
**for all** $p_i = (0, e_1) \in \mathcal{F}_1$ **do**
    $\mathcal{N}_1$= all the items that have a temporal distance with respect to $p_i$
    $\mathcal{FN}$=$\mathcal{FN} \cup \mathcal{N}_1$
**end for**
**return** $\mathcal{FN}$
**Ensure:** The final set of event sets $\mathcal{FN}$ that have different distances with respect the
    each frequent 1-sequence
```

Algorithm 14 DFS-Explore(p, $\mathcal{X}$)

```
**Require:** the current frequent pattern $p = (T_1, T_2, \ldots, T_n)$, set of item-pairs for extension $\mathcal{X}$
$\mathcal{X}_{temp} = \emptyset$
$\mathcal{F}_b = \emptyset$,$\mathcal{C} = \emptyset$, $\mathcal{L} = \emptyset$
**for all** $(t_e, e) \in \mathcal{X}$ **do**
    **if** $t_e = 0$ **then** $\{e$ is added in the last transaction of $p\}$
        $p' = (T_1, T_2, \ldots, T'_n = T_n \cup e)$
    **else** $\{e$ is added in a new transaction $T_{n+1}$ that has a distance of $t_e$ with respect to $T_n\}$
        $p' = (T_1, T_2, \ldots, T_n, T_{n+1} = (t_{T_n} + t_e, e))$
    **end if**
    **if** isAPossiblePattern($p'$) **then**
        **if** ($p'$ is frequent) **then**
            $\mathcal{X}_{temp} = \mathcal{X}_{temp} \cup \{(t_e, e)\}$
            **if** ($p'$ appears at least $min\_sup$ times without exceeding any expected end point corresponding to a beginning
            points, that appears in $p'$ that does not have its end point in $p'$) **then**
                $\mathcal{C} = \mathcal{C} \cup \{p'\}$
                **if** (all of intervals appearing in $p'$ are completed) **then**
                    $\mathcal{L} = \mathcal{L} \cup \{p'\}$
                **end if**
            **end if**
        **end if**
    **end if**
**end for**
$\mathcal{F}_b = \mathcal{F}_b \cup \mathcal{L} \cup$ ExploreChildren($\mathcal{C}$,$\mathcal{X}_{temp}$)
**return** $\mathcal{F}_b$
**Ensure:** The frequent pattern set $\mathcal{F}_b$ of this node and its children
```

Algorithm 15 ExploreChildren($\mathcal{P}$, $\mathcal{X}$)

```
Require: The pattern set $\mathcal{P}$ which contains all the patterns whose children are going to be explored, the set of valid item-pairs $\mathcal{X}$ that generates the $\mathcal{P}$ set by means of extensions
$\mathcal{F}_s = \emptyset$
for all p $\in \mathcal{P}$ do
    $\mathcal{F}_s = \mathcal{F}_s \cup$ DFS-Explore($p$,$\mathcal{X}$)
end for
return $\mathcal{F}_s$
Ensure: The frequent pattern set $\mathcal{F}_s$ for all of the patterns' children
```

case above since there exist two beginning point for interval a without having an end point of a between them; and the pattern $p_2 = \langle(0, a^+)(5, b^-)\rangle$ corresponds to the second case since the end point b^- appears and p_2 does not have any beginning point b^+ before it.

In the step to prune those patterns whose support is under the given threshold, we need to consider some issues. We know that by means of an usual counting of support step (line 10), all the frequent points that take place in a temporal distance with respect to the last event of p, are considered as possible future extensions. The problem that we have with this counting is that, for some occurrences in the database, we are counting some items that occur after some expected items that can be a left-open interval in the current pattern. Thus, we need a new phase where those non-valid occurrences are not taken into account for the actual support counting. In order to add this new control step, we use a queue associated with the IdList of every pattern (we will detail the implementation of the IdLists later).

Finally, the last pruning method is implemented by means of the information already kept in the IdLists. These IdLists are the temporal structure that keep the Vertical Database Format representation, and they are also used to represent the pattern occurrences in the database and, besides, by means of the join operation between two different IdLists we can simultaneously extend a pattern and count its support. In our case, we include all the necessary fields to keep a trace of what intervals are open and their temporal information (in order to check if the sequences are proper). The structure of the IdList is shown in Figure 4.3, where the different fields are:

1. Sid corresponds to the sequence identifier in the database.

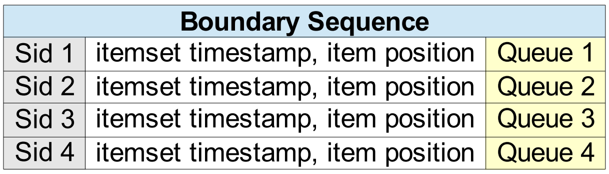
2. Itemset timestamp is the actual temporal occurrence of the last itemset in the sequence whose information is shown in the IdList.

3. Item position refers to the relative position of the last item in the last itemset.

4. Queue is a sorted list that contains the time for the expected end points that refer to all the open intervals that are in the sequence. Each time position in the queue is a pair with the timestamp of the itemset and the item position in that itemset.

Regarding the join operation, unlike in SPADE, there is only a kind of join that finds the occurrences of the second IdList (that belongs to an item-pair) which are in a concrete distance with respect to the first pattern. In our case, besides checking that the temporal distance is respected, we need to check whether the pair (transaction timestamp, position item) occurs before the first element in the queue of the checked sequence. For instance, in Figure 4.4, in the first block we can see the extension of the sequence $\langle(t = 0, a^+)\rangle$ with the item-pair $(1, b^+)$. The temporal distance between them is 1, only in the sequences 1, 3 and 4, occurring $\langle(a^+)\rangle$ in $(1, 1)$ and b^+ in $(2, 1)$ in the three sequences. Since $(2, 1)$ is less than the first element in the queue for sequences 1, 3 and 4 ($(2, 1) < (4, 1)$, $(2, 1) < (4, 1)$ and $(2, 1) < (3, 1)$, respectively), we can make the extension to obtain the new sequence $\langle(0, a^+)(1, b^+)\rangle$. Notice that, in the second sequence, the temporal distance between sequence $\langle(t = 0, a^+)\rangle$ and item b^+ is 3 and is different from the expected distance of 1. In another example, if we want to make an extension for the same previous sequence $\langle(a^+)\rangle$ with the item-pair $(3, b^+)$, we can see that there is no sequence where the condition of occurring before the first element of the queue is accomplished (the second sequence $((4, 1) > (3, 1))$). Therefore, the resulting extension has a support of 0.

In Figure 4.4 the examples already shown in Section 4.2 are considered, and, as before, several situations are introduced. Thus, rows 1 and 2 show the different IdList when we extend a boundary sequence with a boundary item corresponding to an interval beginning point; rows 3 and 4 show the extension of a boundary sequence with a boundary point corresponding to an interval end point of an open interval that was started in the previous boundary sequence; and, finally, rows 5 and 6 show the IdList when a boundary sequence is extended with a simple boundary point.



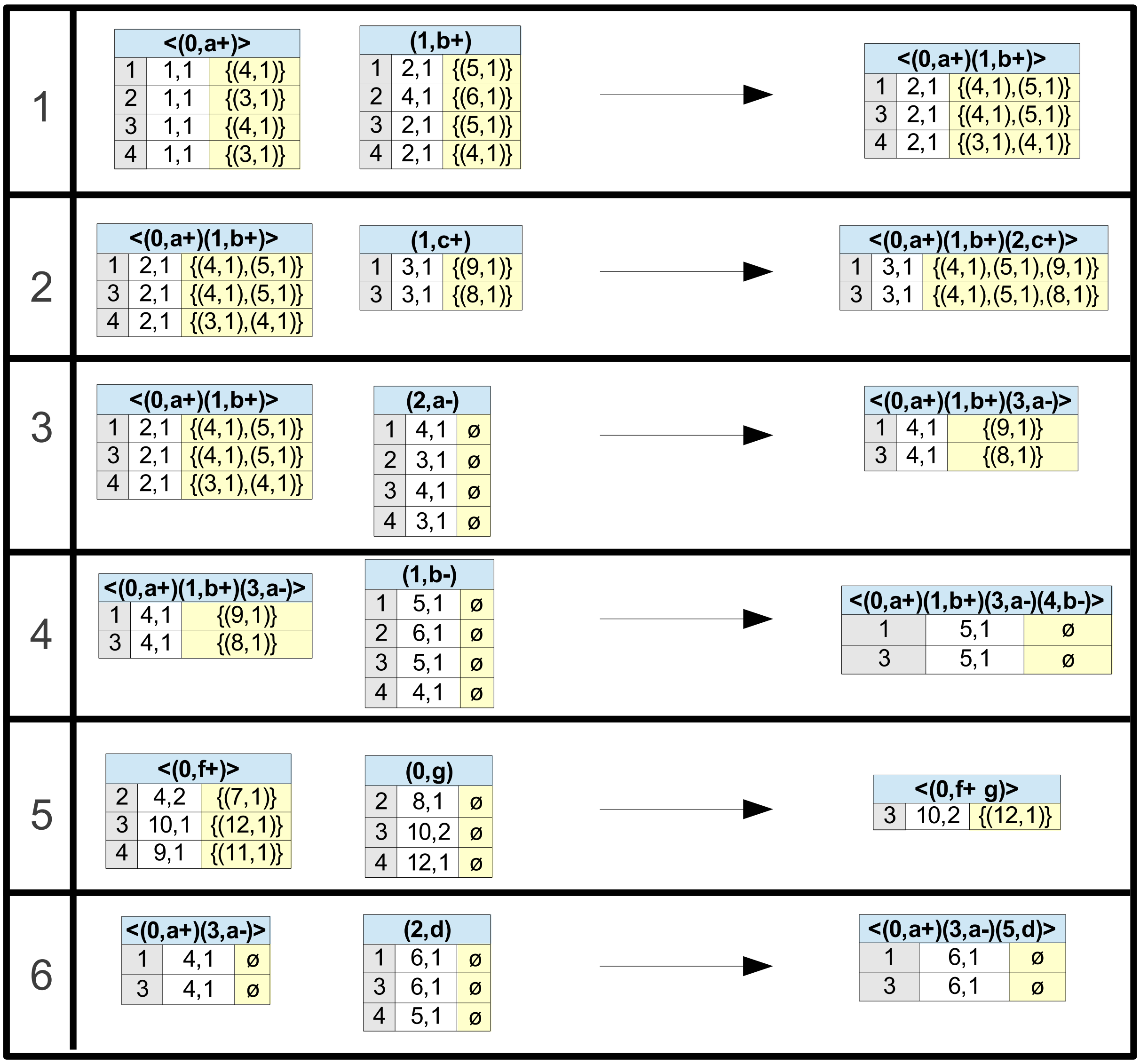
Boundary Sequence		
Sid 1	itemset timestamp, item position	Queue 1
Sid 2	itemset timestamp, item position	Queue 2
Sid 3	itemset timestamp, item position	Queue 3
Sid 4	itemset timestamp, item position	Queue 4

Figure 4.3: New structure for the IdLists used in the new way of count the support in order to find proper boundary sequences.

4.4 Optimizations

In this section we expose some optimizations in order to improve both PaGAPIMS and FaSPIMP algorithms. The first of these optimizations refers to simplify the original database from which we obtain the frequent quantitative patterns whereas the second optimization is only related to FaSPIMP and its generation of frequent 2-patterns.

Regarding the first optimization, in both algorithms we mine all the events using a representation of boundary points and we use some pruning methods to assure the correspondence between the beginning and end of intervals, such as we mentioned in the previous sections. In the algorithms, we firstly remove all those infrequent boundary points that appear in the initial database size is reduced. However, since we are mining quantitative patterns and we are interested in the duration of the items we can firstly remove those infrequent $\langle event, duration\rangle$ pairs from the initial database and to start in that moment the mining of that reduced database. For instance, in Figure 4.5 we see a database composed of two sequences from which we want to extract the frequent quantitative patterns. The upper table of Table 4.1 shows the result of the translation to boundary points of the database shown in Figure 4.5. We can see that all the boundary



1

<(0,a+)>		
1	1,1	{(4,1)}
2	1,1	{(3,1)}
3	1,1	{(4,1)}
4	1,1	{(3,1)}

(1,b+)		
1	2,1	{(5,1)}
2	4,1	{(6,1)}
3	2,1	{(5,1)}
4	2,1	{(4,1)}

<(0,a+)(1,b+)>		
1	2,1	{(4,1),(5,1)}
3	2,1	{(4,1),(5,1)}
4	2,1	{(3,1),(4,1)}

2

<(0,a+)(1,b+)>		
1	2,1	{(4,1),(5,1)}
3	2,1	{(4,1),(5,1)}
4	2,1	{(3,1),(4,1)}

(1,c+)		
1	3,1	{(9,1)}
3	3,1	{(8,1)}

<(0,a+)(1,b+)(2,c+)>		
1	3,1	{(4,1),(5,1),(9,1)}
3	3,1	{(4,1),(5,1),(8,1)}

3

<(0,a+)(1,b+)>		
1	2,1	{(4,1),(5,1)}
3	2,1	{(4,1),(5,1)}
4	2,1	{(3,1),(4,1)}

(2,a-)		
1	4,1	ø
2	3,1	ø
3	4,1	ø
4	3,1	ø

<(0,a+)(1,b+)(3,a-)>		
1	4,1	{(9,1)}
3	4,1	{(8,1)}

4

<(0,a+)(1,b+)(3,a-)>		
1	4,1	{(9,1)}
3	4,1	{(8,1)}

(1,b-)		
1	5,1	ø
2	6,1	ø
3	5,1	ø
4	4,1	ø

<(0,a+)(1,b+)(3,a-)(4,b-)>		
1	5,1	ø
3	5,1	ø

5

<(0,f+)>		
2	4,2	{(7,1)}
3	10,1	{(12,1)}
4	9,1	{(11,1)}

(0,g)		
2	8,1	ø
3	10,2	ø
4	12,1	ø

<(0,f+ g)>		
3	10,2	{(12,1)}

6

<(0,a+)(3,a-)>		
1	4,1	ø
3	4,1	ø

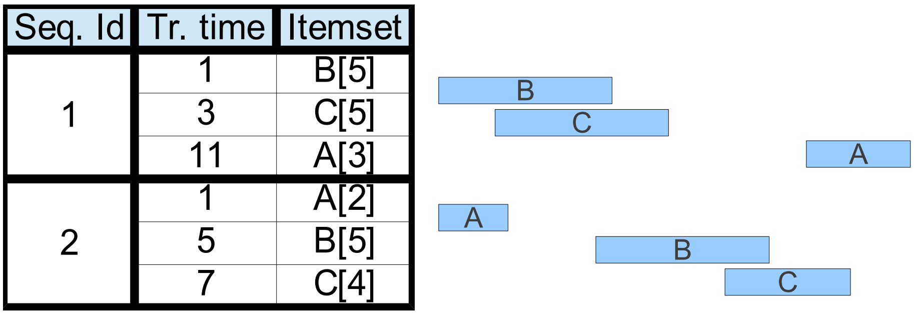
(2,d)		
1	6,1	ø
3	6,1	ø
4	5,1	ø

<(0,a+)(3,a-)(5,d)>		
1	6,1	ø
3	6,1	ø

Figure 4.4: Examples of several extensions of different boundary sequences.

points are considered frequent and we would directly execute the algorithms on it. On the contrary, if before translating to boundary points we remove the infrequent items ($\langle event, duration\rangle$ pairs) we can see that the original database is highly reduced to only two boundary points per sequence, such as is shown in the bottom table of Table 4.1. This optimization can be done in both PaGAPIMS and FaSPIMP and it greatly improves the execution of both algorithms, particularly with big databases, since there normally exist a lot of infrequent items ($\langle event, duration\rangle$ pairs) that can be removed. Note that, as we said before, the support considered is lower than in the qualitative case.

As for the generation of the frequent 2-patterns let us see the behaviour of both algorithms. While PaGAPIMS calculates the temporal distances of the frequent boundary points in each database projection, FaSPIMP needs to find all the possible temporal distances that can exists in a previous step before starting the execution. This calculation, in FaSPIMP, is exactly done in the step when the frequent 2-patterns are built. For this



Seq. Id	Tr. time	Itemset
1	1	B[5]
	3	C[5]
	11	A[3]
2	1	A[2]
	5	B[5]
	7	C[4]

Figure 4.5: Small Interval database for showing the effect of the first optimization.

Boundary Sequence database									
SID/tid	1	3	5	6	7	8	10	11	14
1	(b^+)	(c^+)		(b^-)		(c^-)		(a^+)	(a^-)
2	(a^+)	(a^-)	(b^+)		(c^+)		(b^-)	(c^-)	

Boundary Sequence database				
SID/tid	1	5	6	10
1	(b^+)		(b^-)	
2		(b^+)		(b^-)

Table 4.1: Examples of the effect of removing the infrequent events from the database shown in Figure 4.5. In the upper figure every boundary point is maintained since all of them are frequent while in the bottom figure only the item $\langle b, 5\rangle$ as that item is the only frequent.

step, we necessarily have to make a complete scan of the database and to find all the possible 2-patterns, selecting only the frequent ones. Furthermore, these frequent 2-patterns do not have to be proper since we need the distances between all the possible frequent combinations of events. Thus, if we are executing FaSPIMP on the example database shown in Figure 2.1, after removing the infrequent items ($\langle event, duration\rangle$ pairs) we mine all the possible frequent 2-length that can be found with the different combinations of boundary points. Figure 4.6 shows all the frequent 2-patterns extracted over the example database.

Some problems are associated with the mining of the frequent 2-patterns. These problems normally arise with big databases, from low supports, and because we take into account all the possible combinations of patterns, without discarding those non-proper ones. What is more, if we have very different temporal distances between two boundary points we need to count them as different 2-patterns. Therefore, sometimes the algorithm cannot complete the search for frequent 2-pattern if the support is very low.

In order to fix this important problem we propose to make a change in the search of frequent 2-patterns. This change consists of executing this level as a Pattern Growth algorithm, making partitions for each group of 2-patterns that are created from a frequent boundary point. Later, we continue executing FaSPIMP as usual, having all the item pairs that we need for its execution. Through this search method for frequent 2-patterns, the

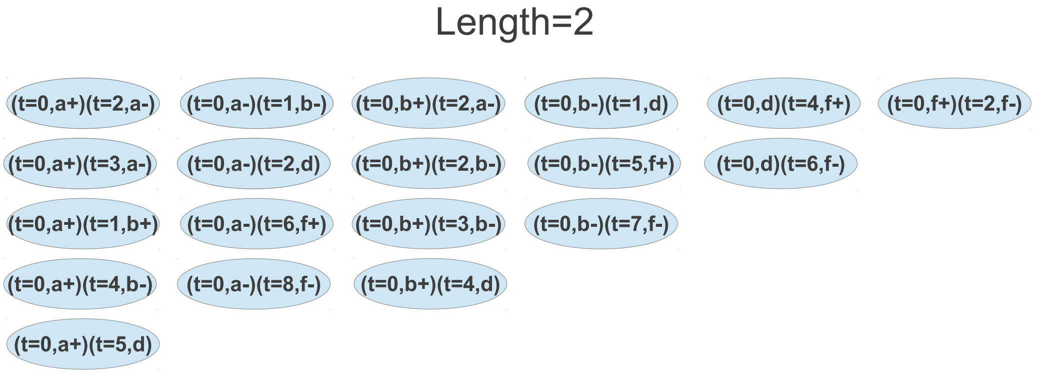


Figure 4.6: Frequent 2-patterns needed by FaSPIMP in order to a proper execution.

algorithm do not suffers from the prior problems and we can execute FaSPIMP in almost any database, as long as the frequent 2-pattern set fits in main memory. For now on, we will refer to this algorithm, FaSPIMP with a Pattern Growth algorithm for mining the frequent 2-pattern set, as ***Prefix_FaSPIMP***.

4.5 Experimental results

The setting of the experiments is the same to the explained in Section 3.4.

All the experiments, except for the last one (Figure 4.13), we use for FaSPIMP the version that uses a Pattern Growth method for mining the set of frequent 2-patterns (See Section 4.4). Besides, all the items considered in all the databases can extend from a duration of 1 to 10 units of time.

In Figure 4.7 we show the behaviour for moderated density values (n= $\{50, 100\}$). In the 4.7A plot we can see the running time for the both algorithms PaGAPIMS and FaSPIMP when we mine a dataset with a density of 0.2 and with very few transactions per sequence (ptl=5). We can observe that the time execution is very similar for both algorithms for this configuration. Plot 4.7B shows the behaviour for both algorithms when we consider a denser database ($\delta = 0.4$) by reducing the number of different items (n=50). We can see that FaSPIMP is slower than PaGAPIMS, despite having a denser database. This phenomenon occurs due to the time spent by FaSPIMP in the mining of the set of frequent 2-patterns.

In Figure 4.8, we can see what happens when we increase the number of transactions per sequence and decrease the transaction average length for different number of items (n=50,100). In summary, we can say that we obtain similar results to those in Figure 4.7 even though we deal with sparser database (parameter *ptl* is lower). Both plots 4.8A and 4.8B have the same behaviour: the execution times of PaGAPIMS and FaSPIMP are very similar, however, for low supports, FaSPIMP shows worse results than PaGAPIMS. As in plot 4.7B, the sudden cost in time spent by FaSPIMP is because of the number of frequent 2-patterns sharply grows. Furthermore, in both plots, the number of frequent patterns has a great change for the last support values (in 0.06-0.04 in 4.8A and in 0.04-0.02 in 4.8B).

In Figure 4.9 we can see the behaviour when we deal with larger sequences (40 trans-

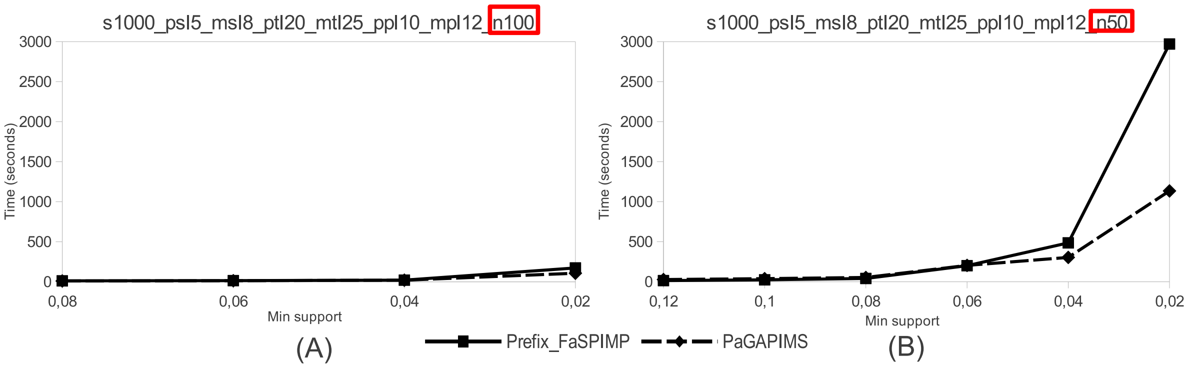


Figure 4.7: Varying support for datasets $s1000\_psl5\_msl8\_ptl20\_mtl25\_ppl10\ \_mpl12\ \_n50$–100.

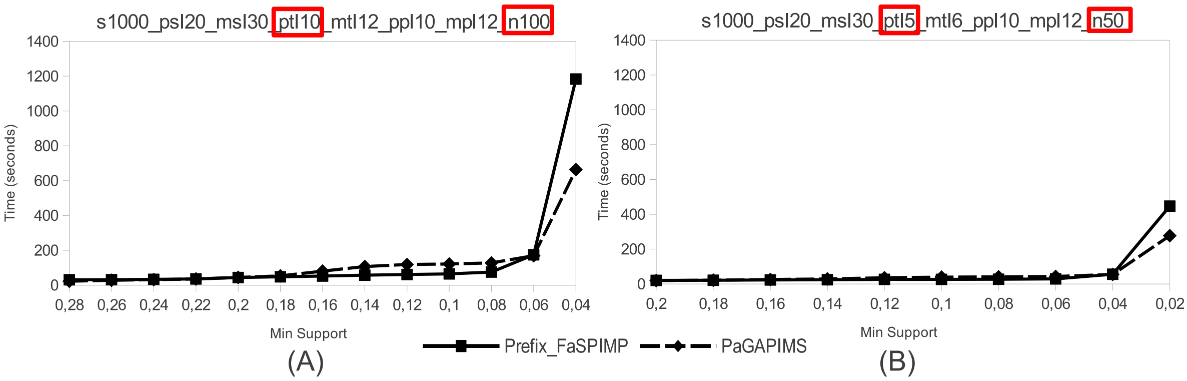


Figure 4.8: Varying support for datasets $s1000\_psl20\_msl30\_ptl5$–10_mtl6–$12\_ppl10\ \_mpl12\ \_n50$–100.

actions on average), with different transaction average length (5, 10 and 20) and big changes in the number of different items (50, 100, 500 and 1000), having sparser density values compared to those given in Figures 4.7 and 4.8. In plots 4.9A and 4.9B, PaGAPIMS outperforms FaSPIMP. Plots 4.9C and 4.9D show the results with a dataset where the patterns inserted are quite longer than the cases above and there are 1000 and 500 different items, respectively. Therefore, we have dealt with sparser databases and the time execution has increased (720 and 547 seconds, respectively). For plot 4.9C, now FaSPIMP outperforms PaGAPIMS, having a wide difference in execution time for low supports. Plot 4.9D shows a better result for PaGAPIMS in the lowest supports whereas plot 4.9E, with a sparse database (500 different items) with medium-high transaction length (20 items on average), shows a better behaviour of FaSPIMP against PaGAPIMS. In general terms we see two kinds of results: 1) PaGAPIMS is better than FaSPIMP because of sparse databases with short transaction per sequence and/or a costly mining of frequent 2-patterns for FaSPIMP, and 2) FaSPIMP shows a better time execution than PaGAPIMS in denser databases when the mining of frequent 2-patterns is not very costly and the typical drawbacks of PaGAPIMS are particularly remarkable. In addition, we can see a hard and sudden change in plots 4.9B and 4.9C for supports 0.04-0.2 in plot 4.9B and 0.08-0.04 in plot 4.9C.

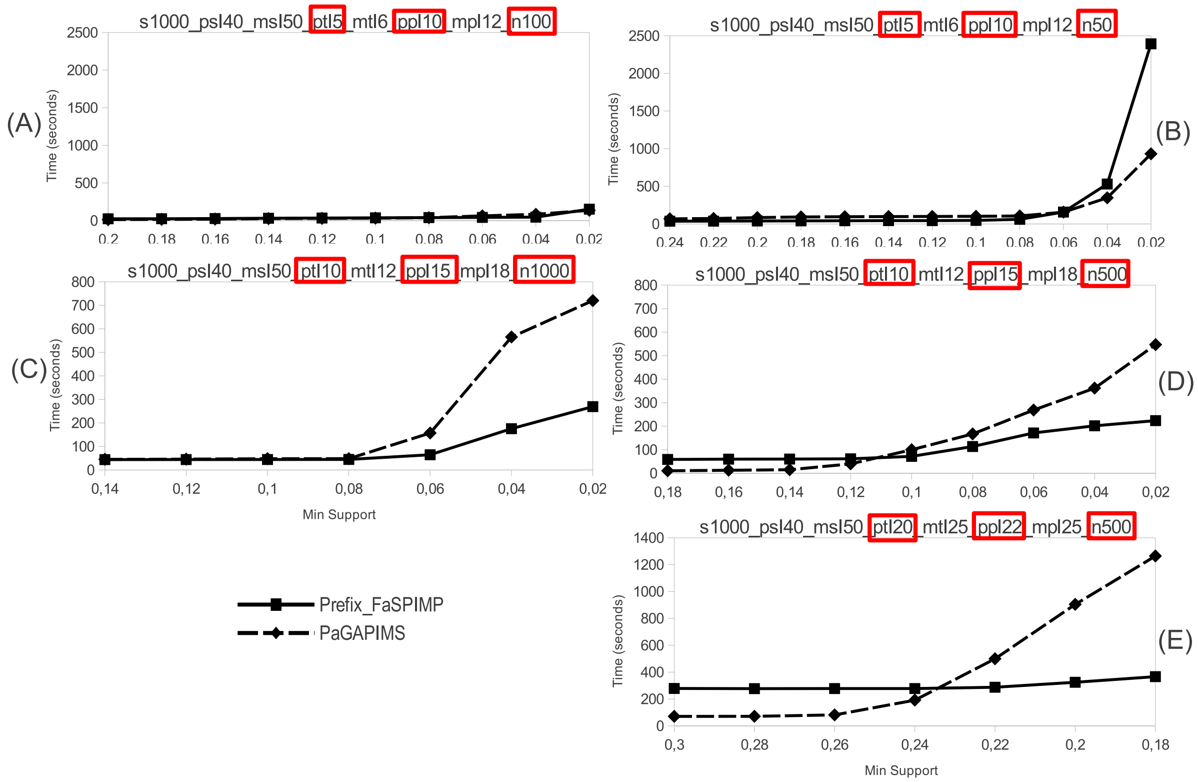


Figure 4.9: Varying support for datasets $s1000\_psl40\_msl50\_ptl5–10–20\_mtl6–12–25\ \_ppl$ $10–15–22\_mpl12–18–25\_n50–100–500–1000$.

Figure 4.10 shows what it happens with both algorithms when we deal with larger databases (10000 sequences) and with high values of different items (n = 500 and 1000). Plots 4.10A and 4.10B have medium sequence length (20 transactions on average), medium-high transaction length (20 items on average) and medium pattern length (10 items on average) and 500 and 1000 different items for the database. The dataset that correspond to the plots 4.10C and 4.10D have larger sequences (40 transactions on average) and smaller transactions (10 items on average). In all the plots, FaSPIMP has a little domain over PaGAPIMS, drawing a similar curve in all the cases. We can see that, having a configuration similar to that shown in Figure 4.8, we have a similar behaviour, but now, with a faster execution of FaSPIMP since its advantages are remarkable over PaGAPIMS. Besides, as we commented on previous Figure 4.9, all these plots show a sudden change in their behaviour, having a big increase at concrete supports (support 0.08-0.06 in plot 4.10A, supports 0.04-0.02 in plot 4.10B, supports 0.08-0.04 in plot 4.10C and supports 0.12-0.08 in plot 4.10D), and, as before, this is also due to the big increment in the number of patterns when we mine the dataset with those supports.

As we did in Section 2.3, we finally show the behaviour when we change those crucial parameters that especially affect to the algorithms execution. As before, we show the executions when we progressively change the “transactions per sequence” parameter, and one of those parameters that affect to the density, the number of items in our case. In Figure 4.11 we see what happens when we have a database with moderated parameters (1000 sequences, 20 transaction per sequence on average, and 12 items per transaction,

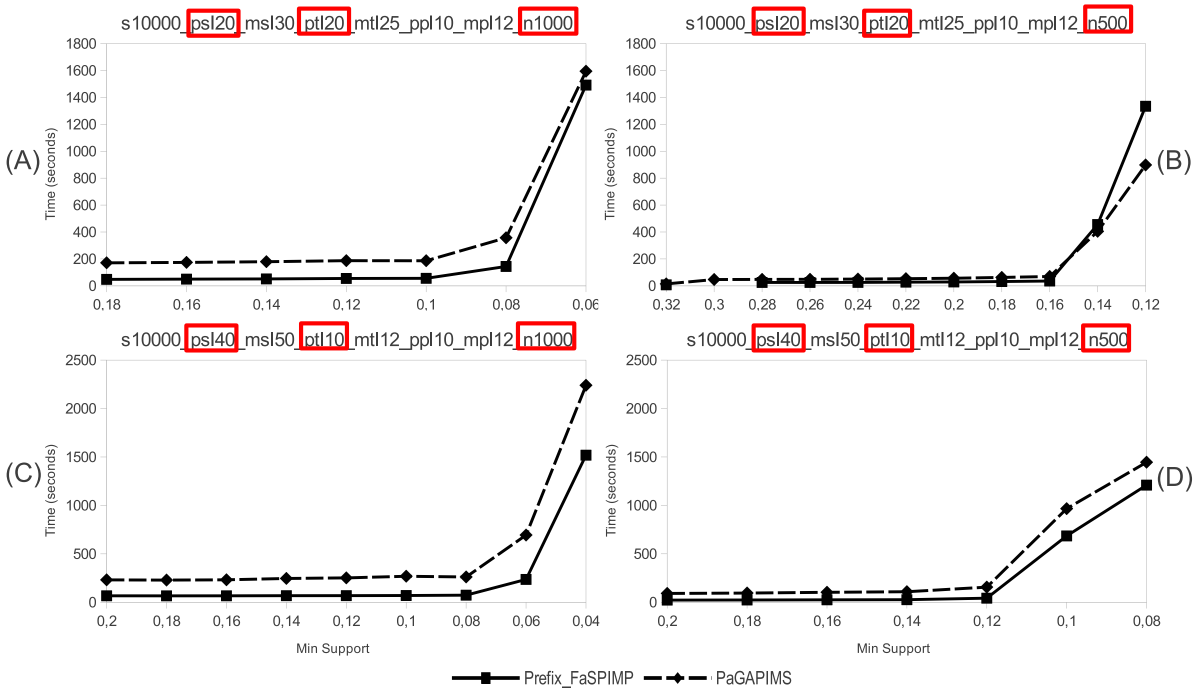


Figure 4.10: Varying support for datasets *s*10000_*psl*40_*msl*50_*ptl*10–20_*mtl*12–25_*ppl*10 _*mpl*12_*n*500–1000.

being the length of the patterns of the database of 12 items on average) and we vary the number of different items (50 in plot 4.11A, 100 in plot 4.11B and 200 in plot 4.11C). In general, there is not a clear domain of any algorithm. Since a Vertical Database Format algorithm normally is faster than a Pattern Growth strategy in dense databases, we think that for these examples the drawbacks derived from the spent time in the mining of the set of frequent 2-patterns implies a longer execution time. In fact, plots 4.11A and 4.11B show a big change from FaSPIMP to PaGAPIMS and this should be related with that issue. In the same way, Figure 4.12 shows the behaviour considering 100 different items per database, and different “transactions per sequence” (10 in plot 4.12A, 20 in plot 4.12B and 40 in plot 4.12C). Now, there is also a similar behaviour to that obtained in Section 2.3, when we worked with point-based database and qualitative relations. However, now is much more difficult for PaGAPIMS to obtain good results than for the point-based case due to the introduction of interval items. These new problems are widely explained in the next Section 4.6, comparing the benefits and drawbacks for the both new algorithms PaGAPIMS and FaSPIMP.

Finally, we show different plots comparing two different FaSPIMP versions: 1) that one that considers a brute force search algorithm for the mining of the frequent 2-patterns, and 2) that one that executes a Pattern Growth algorithm for the search of the frequent 2-patterns. Figure 4.13 shows three plots where both algorithms are compared. We refer as FaSPIMP to the first version and Prefix_FaSPIMP to the second one. Two cases are shown in the figure. The first case occurs in plots 4.13A and 4.13B and shows that the version with a Pattern Growth search is faster than the other one. This is because the standard search count all the possible 2-patterns first, and afterward it discards those

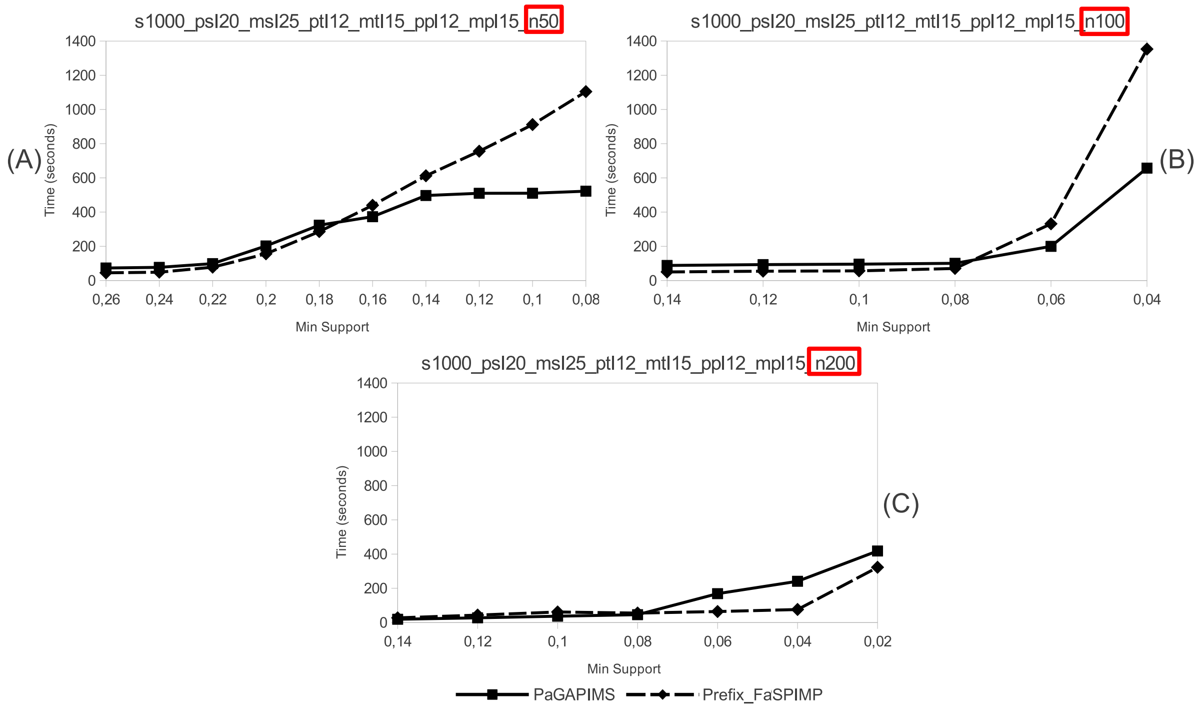


Figure 4.11: Varying support for a same configuration of datasets where we change the number of items (50, 100 and 200).

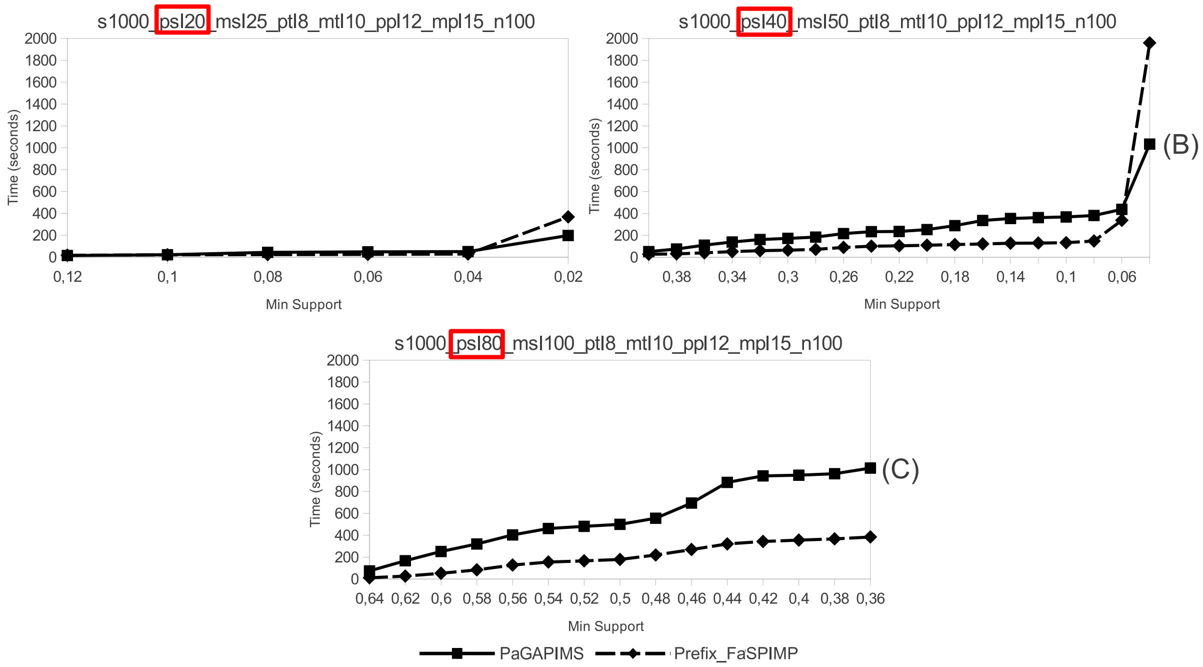


Figure 4.12: Varying support for a same configuration of datasets where we change the number of items per itemset (20, 40 and 80).

infrequent whereas the version with a Pattern Growth does this in an incremental way. The second case is the most important one and it is shown in plot 4.13C. In this case we have a longer database with longer sequences (psl = 80) and the standard method cannot be completed. That is precisely what occurs for supports 0.08 and 0.06, where only the version with Pattern Growth search can finish. Note that, when we increase the number of transactions per sequence (parameter psl), the execution time is higher and the algorithms have more problems to reach low supports.

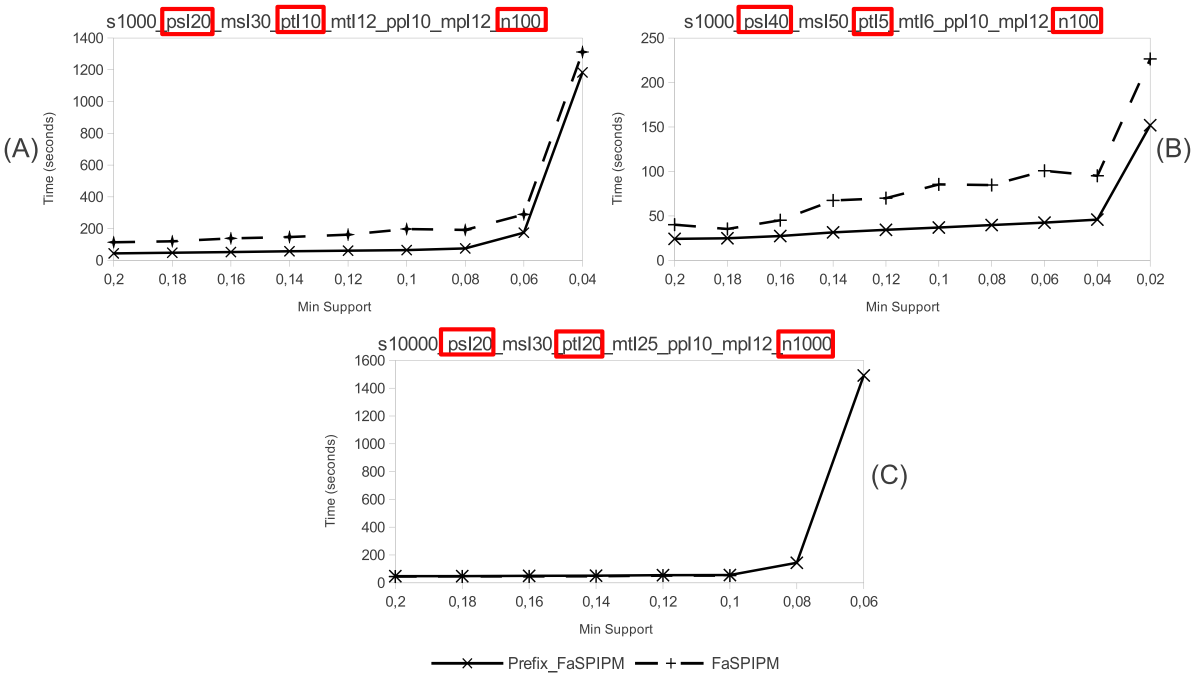


Figure 4.13: Comparison between FaSPIMP and Prefix_FaSPIMP.

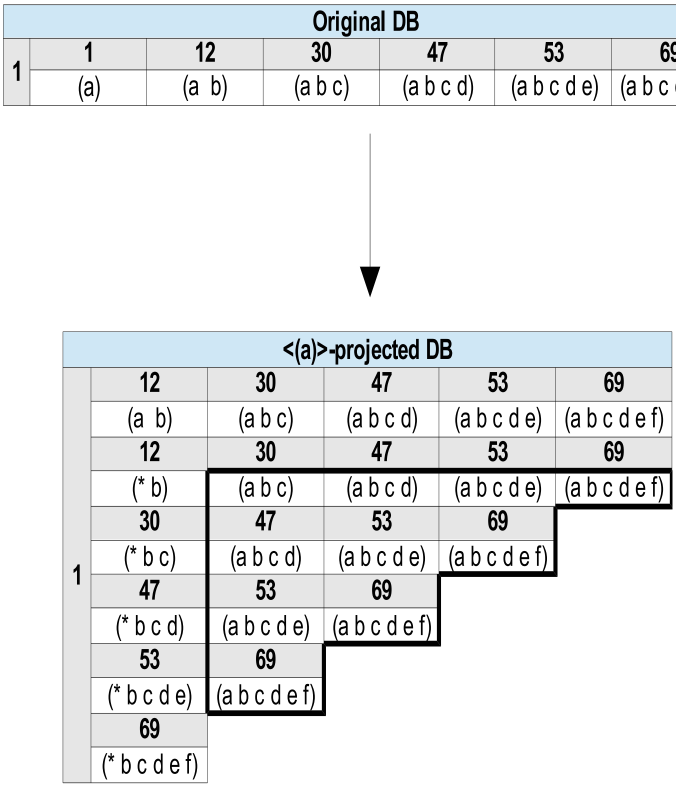
In general, we can see that all the executions achieve lower supports that for the qualitative algorithms and, in some points, they suddenly increase. This is because of the pattern explosion phenomenon that is explained in the next section.

4.6 Discussion. Comparing both algorithms

In this Section we discuss the main advantages of FaSPIMP with regard to PaGAPIMS. In the first place, let us recall the main differences between the original versions of the Vertical Database Format and Pattern Growth algorithms on which FaSPIMP and PaGAPIMS are respectively based. Later, we will show the behaviour of these algorithms when we mine quantitative patterns, and, especially once all the necessary information with which to simultaneously manage points and intervals and the pruning methods has been included, in order to verify that all the patterns contain proper quantitative intervals. Finally we explain some drawbacks that are inherent to the problem of mining quantitative patterns.

One of the situations overlooked in some algorithms such as PrefixSpan, is the presence of several appearances of the same item or event type in a sequence. On the one hand, in PrefixSpan it is necessary to make all the projections of an item that appears several times in a sequence in order to guarantee that all the I-extensions associated with that item will be discovered. If, conversely, it was not necessary to find the I-extensions, or if there were only one item per itemset, or only one appearance of each item per sequence existed, there would be sufficient information with a single projection. On the other hand, in Vertical Database Format algorithms, such as SPADE, it is not necessary to take into account the problem of PrefixSpan, and all the S-extension and I-extensions explore the whole search space. It could thus be said that the latter is less influenced by the database structure.

Clearly, this fact has an important impact in the efficiency of both algorithms, depending on the original database. For example, let us view the behaviour of both PrefixSpan and SPADE in the case of the database composed of the single sequence $s = \langle(1, a)(12, ab)(30, abc)\ (47, abcd)(53, abcde)(69, abcdef)\rangle$ and a $min\_sup = 1$. In this case, there are six itemsets, occurring between times 1 and 69. If we consider sequence $\langle(a)\rangle$, the six different projections associated with this sequence are shown in Figure 4.14. Note that if the first projection is considered in our example, then items b, c, d, e and f that has a distance of zero with respect to $\langle(a)\rangle$ will not be considered to be frequent items, and in order to count these items it is necessary to take into account the first itemset of every subsequent projection. Figure 4.14 highlights all the itemsets that can be ignored by PrefixSpan.



Original DB

	1	12	30	47	53	69
1	(a)	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)

<(a)>-projected DB

1	12	30	47	53	69
	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	12	30	47	53	69
	(* b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	30	47	53	69	
	(* b c)	(a b c d)	(a b c d e)	(a b c d e f)	
	47	53	69		
	(* b c d)	(a b c d e)	(a b c d e f)		
	53	69			
	(* b c d e)	(a b c d e f)			
	69				
	(* b c d e f)				

Figure 4.14: Projected database for the brief example with the standard PrefixSpan algorithm.

With regard to SPADE, and using the same example as above, this algorithm builds the associated IdLists (shown in Figure 4.15) without any special consideration. This point is one of the great advantages of the Vertical Database Format strategy over the

Pattern Growth strategy for certain database configurations.

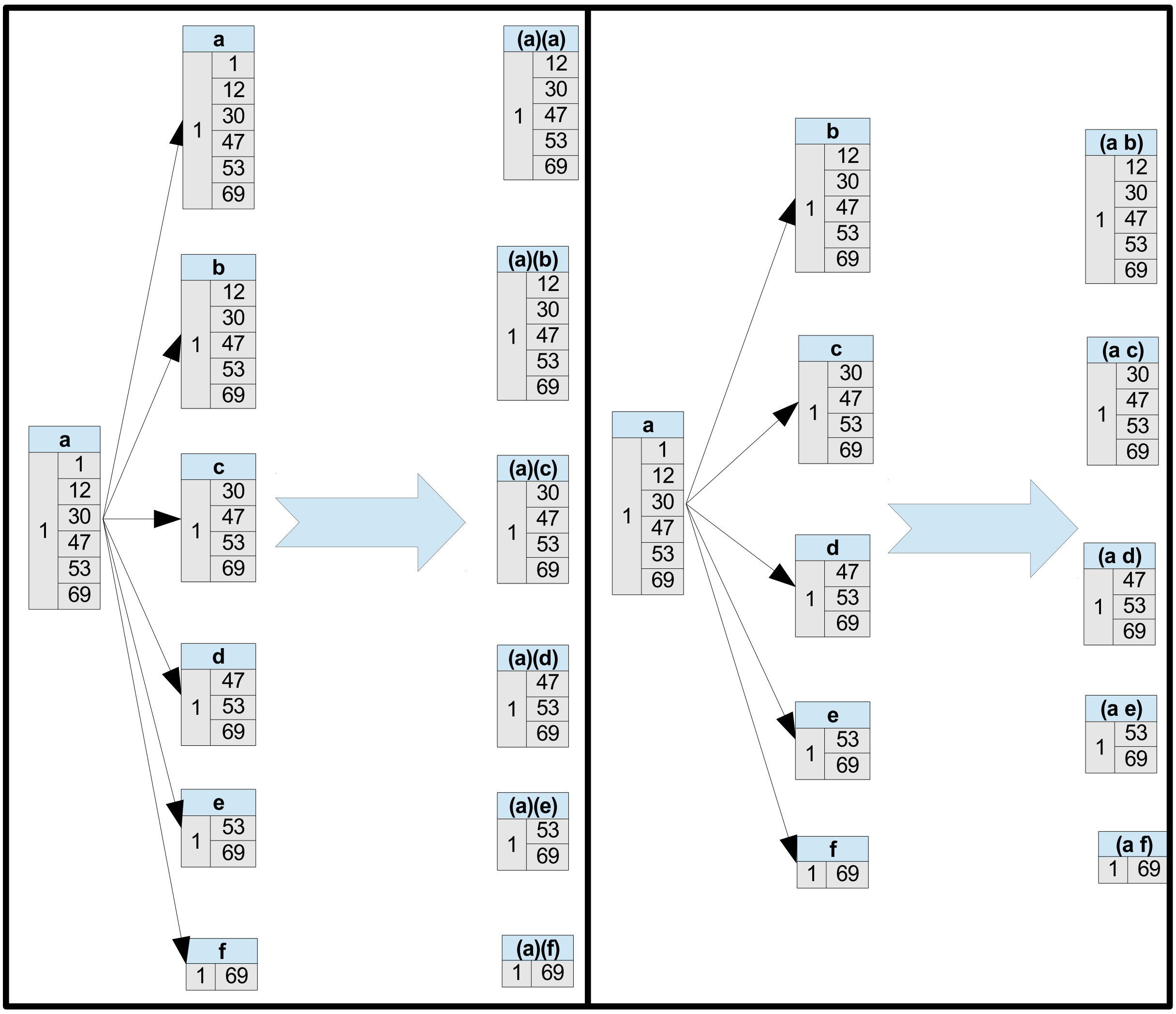


Figure 4.15: SPADE IdList for the brief example.

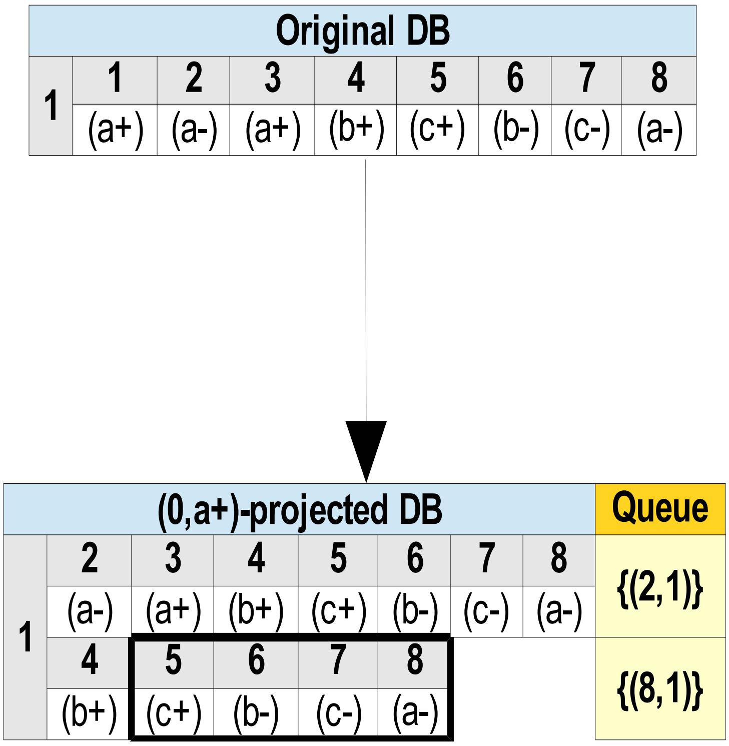
In regards to the quantitative version of SPADE and PrefixSpan, the above characteristics also define the main behaviour of both algorithms. Concretely, whereas for SPADE we have to be sure that the itemset time information that appear in IdLists has exactly the temporal distance indicated by the item-pair that we are using to do the extension, PrefixSpan cannot avoid the exploration of the shadowed part shown in Figure 4.15. In the original PrefixSpan, used to mine point-based patterns, we can avoid the exploration of the shadowed part in Figure 4.14, since those relations are already taken into account in the first projection (relation before). Now, for a quantitative version of PrefixSpan, it is not possible to ignore that shadowed part because we would not find all the item-pairs needed. For instance, regarding the Figure 4.14, if we use PrefixSpan in the usual way, we will ignore some item-pairs like $(18, a)$ (found in the second projection) or $(17, a)$ (found in the third projection).

With regard to FaSPIMP and PaGAPIMS, as we saw in Sections 4.2 and 4.3, both algorithms have new changes with respect to the original SPADE and PrefixSpan algorithms in order to guarantee that all the boundary sequences are well-formed. Concretely, even when FaSPIMP uses two new pruning mechanisms, the principal behaviour remains. Besides, in PaGAPIMS, the exploration of the shadowed part for all the projections of a projected sequence is still mandatory. Now, besides the above explanation, PaGAPIMS has other new drawbacks since we could overlook some interval relation like *meets*, *overlaps*, *contains*, *starts*, *finished by* or *equals*.

For instance, let us consider the database formed only of the sequence $\langle(1, a^+)(2, a^-)(3, a^+)(4, b^+)(5, c^+)(6, b^-)(7, c^-)(8, a^-)\rangle$ and $min\_sup = 1$. If we project it by the 1-sequence $(0, a^+)$, in Figure 4.16 we can find the two projections associated with the $(0, a^+)$. There, if we process it as in the original PrefixSpan algorithm, we will take into account all the eight itemsets in the first projection and only the first itemset in the second projection. Therefore, we would not find the patterns $\langle(0, a^+)(1, b^+)(2, b^-)(3, a^-)\rangle$, $\langle(0, a^+)(1, c^+)(2, c^-)(3, a^-)\rangle$ and $\langle(0, a^+)(1, b^+)(2, c^+)(3, b^-)(4, c^-)(5, a^-)\rangle$, since in the first projection we discard the elements after the itemset at time $t = 2$, that is precisely the moment when the interval a is finished. Thus, in order to find the complete set of frequent boundary sequences, we need to completely analyse every itemset of every projection, being this necessity a new drawback for PaGAPIMS in regards to original PrefixSpan.

Furthermore, as we saw in Section 4.3, FaSPIMP needs to mine the set of frequent 2-patterns and a good way to do it is through a Pattern Growth method. This last fact makes FaSPIMP slower than a normal Vertical Database Format algorithm and, for supports where the algorithms mostly find frequent 1-patterns and 2-patterns, PaGAPIMS is clearly better than FaSPIMP. Even though both of them can execute a Pattern Growth Strategy up to 2-patterns, PaGAPIMS discards the improper interval and FaSPIMP does not do that at first in order to catch all the temporal distances. However, after mining level 2, FaSPIMP normally starts to be faster than PaGAPIMS since the nature of Vertical Database Format algorithms is faster than Pattern Growth algorithm in dense databases with several transactions per sequence.

Finally, we discuss about the drawbacks that appear in every algorithm for mining quantitative patterns. One drawback is the sharp change in the execution time curve which normally appears when we run any algorithm for quantitative patterns. This phenomenon belongs to the nature of the quantitative search. In this case, there are a lot of occurrences of patterns whose items share the event identifiers but have different durations, and with a lot of relations with different temporal distances between them. Thus, it typically occurs that, for high and moderated supports, the number of frequent patterns is quite low, whereas for very low supports, the number of patterns sharply increases. Another drawback appears in Vertical Database Format strategy, fact that directly affects to FaSPIMP algorithm. For this strategy, since we need all the different temporal distances of the relations in order to create larger candidates and we ignore the distances that there appear in the original database, a second scan of the database is needed for a proper execution of the algorithms. As we previously explained in Section 4.4, that search can have several problems when we execute a brute force mining for the frequent 2-patterns, and a good alternative is to use a Pattern Growth algorithm just to find the set of frequent 2-patterns. Anyway, even though we execute this last method, the search of frequent 2-patterns take a longer time than if we could already have the temporal dis-



Original DB

1	1	2	3	4	5	6	7	8
	(a+)	(a-)	(a+)	(b+)	(c+)	(b-)	(c-)	(a-)

(0,a+)-projected DB

								Queue
1	2	3	4	5	6	7	8	{(2,1)}
	(a-)	(a+)	(b+)	(c+)	(b-)	(c-)	(a-)	
	4	5	6	7	8			{(8,1)}
	(b+)	(c+)	(b-)	(c-)	(a-)			

Figure 4.16: Projected database for the brief example with PaGAPIMS algorithm.

tances between relations and directly create candidates of length 2 and, thus, this search means a time overhead in Vertical Database Format algorithms for quantitative patterns.

4.7 Conclusions

In this Chapter we have introduced two algorithms that deal with these issues for mining quantitative interval patterns. Our main contributions are as follow:

- We simplify the processing of relations among quantitative intervals by getting all the information of all quantitative boundary points in a sequence. The main advantage of this representation, compared with the representations commonly used, is that the relations among quantitative boundary points are reduced to *before*, *equals* and *after*, followed by a temporal distance, instead of using directly quantitative Allen's relations. We refer to this pattern representation as *sequence of quantitative interval boundaries*, and it expresses a pattern or sequence without any ambiguity. Thus, only by means of points and their distances we can express the relations among both the intervals and/or points that can appear in a pattern.

- We describe in detail the algorithm PaGAPIMS (**Pat**tern **G**rowth **A**lgorithm for **P**oint and **I**nterval **M**etric **S**equences), an implementation based on Pattern Growth strategy, that uses the same pattern representation and capable of finding the whole set of frequent quantitative patterns containing relations between points, intervals or points and intervals. A similar algorithm has been used in works [Mörchen and

Fradkin, 2010; Chen et al., 2011] but without quantitative relations. Besides, we show the main changes that we have to add to original algorithm to enable the discovery of patterns with proper intervals.

- We introduce a novel algorithm, called FaSPIMP, which stands for **Fa**st **S**trategy for **P**oints and **I**ntervals **M**etric **P**atterns, capable of discovering the same set of frequent quantitative patterns found by PaGAPIMS. FaSPIMP is based on the Vertical Database Strategy and, to the best of our knowledge, it is the first algorithm for quantitative point relations between points and intervals that it is based on that strategy. Besides, FaSPIMP uses efficient methods to avoid the generation of useless candidates and to check the frequency of the candidates.

- We use, in both FaSPIMP and PaGAPIMS algorithms, a novel candidate generation based on boundary points instead of quantitative intervals. Besides, we describe in detail the different pruning mechanisms that we use to discard those sequences that are not correct.

- Regarding the drawbacks of the algorithms, in general terms, the drawbacks that normally appear in Pattern Growth algorithms are also present in PaGAPIMS even with a higher intensity. Furthermore, a pruning mechanism, as it is used in FaSPIMP, is very convenient in order to reduce the search tree. Besides, when we face the problem of mining quantitative patterns a boundary representation initially finds some problems since we only remove the infrequent event without taking into account the duration associated with them. What is more, when a Vertical Database Format algorithm is used, a problem emerges because we need to store all the possible temporal distances associated with relations that appear between any two points. In this Chapter we introduce two valid optimizations that can effectively solve the two previous problems.

- We make an exhaustive comparison between PaGAPIMS and FaSPIMP. We show that there not exist a remarkable difference between both execution but we recommend the use of FaSPIMP when large patterns can be found and the algorithm spends a lot of time generating candidates with a length bigger than 2.

- Finally, all the performed tests do not offer a clear advantage for any algorithm, nevertheless, when we mine databases where a large number of patterns with length bigger than 2 is found, the advantages of FaSPIMP can be seen more clearly. Therefore, this reason lead us to conclude that Vertical Database Format strategy is more appropriate for mining quantitative interval databases.

Chapter 5

BreadthPIS and DepthPIS: Two New Fast Algorithms for Mining Points and Intervals Qualitative Patterns

In this Chapter we introduce two algorithms, BreadthPIS and DepthPIS, being both of them based on Vertical Database Format strategy. In these two algorithms we use another data representation called Triangular Matrix representation. One of the key aspects of these algorithms is that the generation of candidates is based on temporal reasoning.

This Chapter is organized as follows. In Section 5.1 we add some aspects related to the representation used for the sequences and the basic operation between items. Section 5.2 describes the new breath-first search algorithm BreadthPIS, whereas in Section 5.3 we describe algorithm DepthPIS. An experimental and performance study is shown in Section 5.5, while a wide discussion about the behavioural differences of the algorithms is given in Section 5.6. Finally, we provide our conclusions in Section 5.7.

5.1 Additional definition and description for problem setting

Let us now define the method through which we convert a sequence database in another based on Triangular Matrix representation. Let $\alpha = \langle (t_1, I_1), (t_2, I_2), \ldots, (t_m, I_m) \rangle$ be a sequence where the itemsets may contain both points and intervals. In an iterative way, we can convert α into another representation with explicit qualitative relations between the events. So, for the first itemset, for the first item, we check the relation with all the rest of items appearing in both, the same itemset and in later itemsets, inferring those relations as is shown on the right hand side in Figure 5.1. Thus, we have $(l(\alpha) - 1)$ relations, that are exactly the relations between the first item and all the rest of α. Afterwards, we repeat the same process for the second item, obtaining $(l(\alpha) - 2)$ relations (between the second item and all the rest appearing forward). If we continue, we will end with the item $i_{m,n-1}$ and $i_{m,n}$ and the only relation that appears between them. We represent that in a triangular matrix, where both the rows and columns are the events, and the cells contain the relation between them (R_{ij} is the relation between the i event and the j event). From now on, we refer to this triangular matrix with the name of *TriMax*.

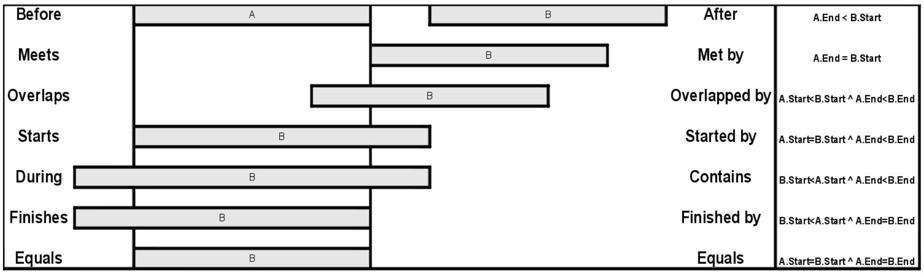

Before
A
B
After
A.End < B.Start
Meets
B
Met by
A.End = B.Start
Overlaps
B
Overlapped by
A.Start<B.Start ^ A.End<B.End
Starts
B
Started by
A.Start=B.Start ^ A.End<B.End
During
B
Contains
B.Start<A.Start ^ A.End<B.End
Finishes
B
Finished by
B.Start<A.Start ^ A.End=B.End
Equals
B
Equals
A.Start=B.Start ^ A.End=B.End

Figure 5.1: Allen's algebra relations.

So, the relations of a triangular matrix can refer to relations between points, interval or points and intervals.

Let us see an example of the given process above. Let $\alpha = \langle (t = 1, \langle A, 3\rangle)(t = 2, \langle B, 3\rangle)(t = 3, \langle C, 6\rangle)(t = 6, \langle D, 0\rangle)\rangle$ be a sequence, such as shown in the first sequence of the example database shown in Figure 2.1. The associated triangular matrix with this sequence is shown in Table 5.1.

	B	C	D
A	o	o	$<$
B		o	$<$
C			c

Table 5.1: Triangular matrix for example sequence.

Triangular matrix representation has several benefits, being the ability to infer relations from other relations the most important one (see Section 2.4). By means of this triangular matrix representation, we obtain a good representation that is: 1) non-ambiguous, since all the relations between all the events are shown in the half-matrix and 2) simple, since a complex sequence can be easily read and all the relations are explicit.

Considering this representation, the implementation of the subsequence checking or the operation with prefix and suffix of sequence is straightforward. We say that the sequence α is a subsequence of another sequence β (or β is a supersequence of the α), denoted as $\alpha \preceq \beta$, if there exists a bijective function f that preserves the order and maps the items in α to items in β, in such a way that 1) $\forall i \in \alpha, \forall f(i) \in \beta, i \subseteq f(i)$ and 2) if $\forall i, j \in \alpha, \forall f(i), f(j) \in \beta$, i has a relation R_{ij} with j, $i\ R_{ij}\ j$, then that relation is maintained by $f(i)$ and $f(j)$ $i\ R_{ij}\ j \Rightarrow f(i)\ R_{ij}\ f(j)$. If we analyse this in terms of triangular matrices, we have to check 1) if the events of α are included in the events of β and 2) the relations of the triangular matrix between the events of α are the same as those that appear in their corresponding events in the triangular matrix of sequence β. For instance, the sequence $\alpha = \langle (t = 1, \langle A, 3\rangle)(t = 2, \langle B, 3\rangle)(t = 3, \langle C, 0\rangle)\rangle$ is a subsequence of $\beta = \langle (t = 1, \langle A, 3\rangle)(t = 2, \langle B, 3\rangle)(t = 3, \langle C, 0\rangle)(t = 7, \langle D, 5\rangle)\rangle$ but not of

$\gamma = \langle(t = 1, \langle A, 2\rangle)(t = 2, \langle B, 3\rangle)(t = 3, \langle C, 0\rangle)\rangle$, since the relation between A and C in α is *contains* whereas in β is *meets*.

We say that two sequences α and β are related by an equivalence relation if they share a common k-prefix. In that case with $[P]$ we denote the *equivalence class* where α and β belong to, being P the k-prefix that relates both α and β. In addition, we call *members* of $[P]$ to the set of patterns that are in the same equivalence class $[P]$. If the prefix P of an equivalence class $[P]$ has a length of k events, $l(P) = k$, we say that $[P]$ is a k*-equivalence class*. We say that a pattern p always belongs to the equivalence class $[P]$.

Definition 5.1.1. We call transition function to the function $f : \mathcal{R} \times \mathcal{R} \rightarrow 2^{\mathcal{R}}$, being $\mathcal{R}$ the set composed of the thirteen Allen's relations and $2^{\mathcal{R}}$ its power set.

Through this transition function we can infer a set of relations from two given single Allen relations. In later Sections, we will define by enumeration the different transition functions that we will use.

In Figure 5.2, we show the final frequent sequence set when we mine the example database under a minimum support of 2. Even though we do not make any distinction between the different patterns, there are different types of frequent sequences: those composed of both simple points and intervals, for instance all the 4-sequences where all the patterns have event points d or g; those formed by only intervals, like all of the patterns that do not have the events d or g in them; and those with only points, composed of those sequences with only the event points d, g or d and g. In total, the final frequent sequence set has 33 frequent sequences.

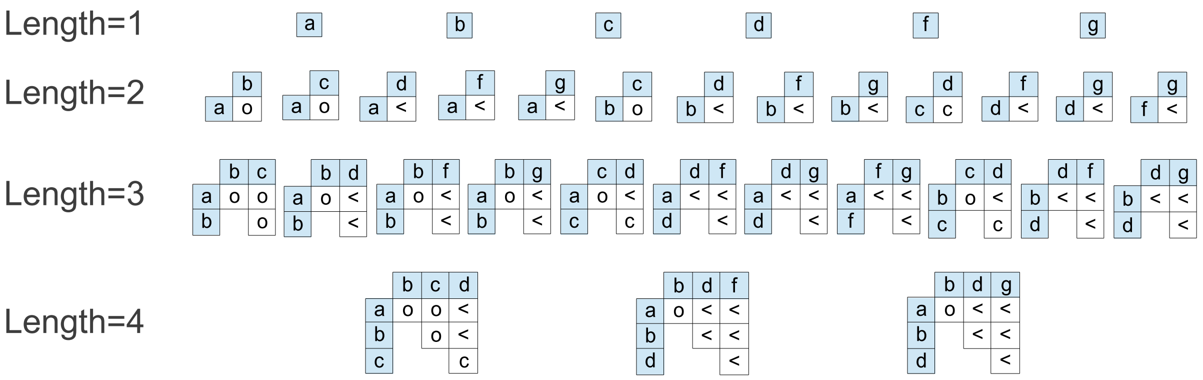
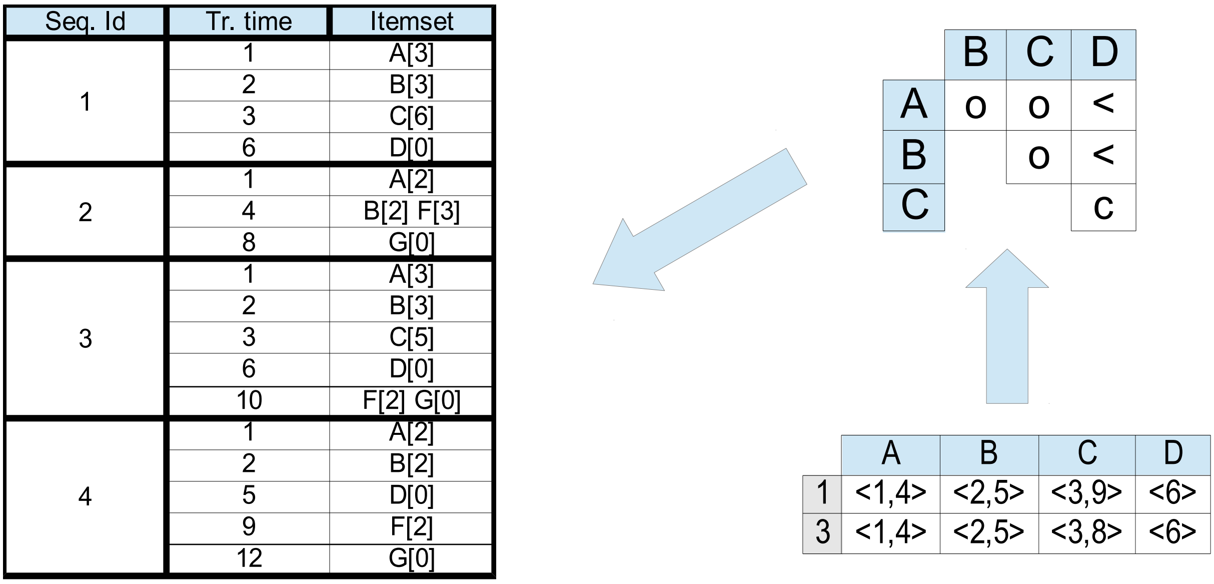


Figure 5.2: Frequent sequence set of example database.

5.2 BreadthPIS: an algorithm for mining point and intervals temporal events based on breadth-first search

In this Section, we formulate and explain every step of our BreadthPIS algorithm. Before starting with the description of main methods, we have to give some keynotes relatives to BreadthPIS algorithm. Since we represent a pattern with a TriMax, and each pattern has a number of appearances in the database, stored in an IdList, whenever we refer to a pattern p, we refer to the couple $\langle TriMax, IdList \rangle$ (both of p). Besides, we will denote the terms $p.TriMax$ and $p.IdList$ to refer to these parts of a pattern p.

The above mentioned IdList is a data structure where we store, for each sequence, a list with all the appearances of a concrete pattern in that sequence. Each appearance is a list of events (point or intervals) and it also includes its timestamp for the beginning and its end when it is an interval. In Figure 5.3, we see the IdList of the example database for the pattern given in Table 5.1.



Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	D[0]
2	1	A[2]
	4	B[2] F[3]
	8	G[0]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[0]
	10	F[2] G[0]
4	1	A[2]
	2	B[2]
	5	D[0]
	9	F[2]
	12	G[0]

	B	C	D
A	o	o	<
B		o	<
C			c

	A	B	C	D
1	<1,4>	<2,5>	<3,9>	<6>
3	<1,4>	<2,5>	<3,8>	<6>

Figure 5.3: IdList for the pattern in the example database.

The BreadthPIS algorithm, shown in Algorithm 16, consists in a main continuous loop that executes a breadth-first search. As first steps, all the frequent 1-patterns and 2-patterns have to be found (we try all the possible direct relations between frequent 1-patterns to obtain the frequent 2-patterns). The mentioned loop is composed of two steps: 1) All the possible (k+1)-superpatterns are generated from the current frequent (k)-patterns, and 2) we select only the frequent patterns among all the possible generated candidates. If this frequent $(k+1)$-pattern set is not empty, we repeat the loop again in order to find longer frequent patterns. Finally, the algorithm ends returning the final frequent pattern set. The steps of candidate generation and counting of support both are shown in Algorithms 17 and 19 respectively.

Algorithm 17 shows the candidate generation algorithm. We base this method on that introduced in GSP algorithm [Srikant and Agrawal, 1996]. The main idea is merging two k-patterns p_1 and p_2 where the $(k-1)$-suffix of p_1 is equal to the $(k-1)$-prefix of p_2. When we find two patterns that accomplish this condition, we can create a new $(k+1)$-pattern built by concatenating to p_1 the last event of p_2. In order to find the patterns that fulfil those conditions, Lines 4-9 create two hash table, one for prefixes and other for suffixes, where we link each pattern to all the patterns to which they are related. Then, in Lines 10-20 we iterate all the entries in the prefix table and, an entry is merged with a pattern in the suffix Table if they are equal. Finally, we return all the generated candidates in

Algorithm 16 BreadthPIS(*IsPruneActivated*)

```
 1: $\mathcal{F}_1 = \{$frequent items$\}$
 2: $\mathcal{F} = \mathcal{F}_1$
 3: $\mathcal{F}_2$ =getFrequent2Sequences($\mathcal{F}_1$)
 4: $\mathcal{F}_k = \mathcal{F}_2$
 5: while $\mathcal{F}_k \neq \emptyset$ do
 6:    $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$
 7:    $C_k$ =GenerateCandidates($\mathcal{F}_k$,$IsPruneActivated$)
 8:    $\mathcal{F}_k$ =CountingOfSupport($C_k$)
 9: end while
10: return $\mathcal{F}$
Ensure: The frequent pattern set $\mathcal{F}$
```

Line 21.

Algorithm 17 GenerateCandidates($\mathcal{F}_k$,*IsPruneActivated*)

```
Require: the (k-1) frequent pattern set $\mathcal{F}_k$, a boolean indicating if the pruning method
    has to be executed $IsPruneActivated$
 1: $\mathcal{H}_{pre} = \emptyset$ { Hash table with duples $\langle pre, preSet \rangle$, being $pre$ a pattern and $preSet$ set
    of patterns }
 2: $\mathcal{H}_{post} = \emptyset$ { Hash table with duples $\langle post, postSet \rangle$, being $post$ a pattern and $postSet$
    set of patterns }
 3: $\mathcal{C}_k$
 4: for all $p \in \mathcal{F}_k$ do
 5:    $p_{pre} = (k-1)$ prefix of $p$
 6:    $p_{post} = (k-1)$ suffix of $p$
 7:    insert $p$ to the associated set of $\mathcal{H}_{pre}(p_{pre})$
 8:    insert $p$ to the associated set of $\mathcal{H}_{post}(p_{post})$
 9: end for
10: for all $\langle pre, preSet \rangle \in \mathcal{H}_{pre}$ do
11:    $\mathcal{S}$ =the associated set $postSet$ of $\mathcal{H}_{post}(pre)$
12:    if $\mathcal{S} \neq \emptyset$ then
13:       for all $p_{pre} \in preSet$ do
14:          for all $p_{post} \in \mathcal{S}$ do
15:             $\mathcal{MS}$ =MergePatterns($p_{post}$,$p_{pre}$,$IsPruneActivated$)
16:             $\mathcal{C}_k = \mathcal{C}_k \cup \mathcal{MS}$
17:          end for
18:       end for
19:    end if
20: end for
21: return $\mathcal{C}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```

The method *MergePatterns*, shown in Algorithm 18, introduces the main changes added by this algorithm with respect to the original point-based ones. These changes are those derived from the use of intervals relations and also from the check of frequency. The method receives two patterns p_{post} and p_{pre} as parameters, and a flag value that indicates if a pruning method has to be executed as soon as we create new candidates. The parameters p_{post} and p_{pre} are k-sequences with a common (k-1)-subsequence (suffix of p_{post} and prefix of p_{pre}) and, therefore, their associated TriMax share some cells as shown in Figure 5.4. We can see that the new candidate is formed by adding the last element and the last column of p_{pre} to p_{post}.

Once we have merged both TriMax, we still have to infer the relation between the first event of p_{post} and the last of p_{pre} (highlighted by an ellipsis in Figure 5.4). We have considered two options:

- to consider the Allen's thirteen relations in that cell and, thus, having thirteen different candidates whose frequency will have to be checked later.
- to infer the relations that connect the event E_1 and the event E_{k+1} by means of all the possible paths E_j, $\forall 1 < j < k+1$, which relate them through an intermediate event. That is, using the "path consistency" as defined by Allen [Allen, 1983]. Such inference is defined by a transition function (see Definition 5.1.1) that returns a set of Allen's relations for a given pair of relations and that set contains the different candidates that can be derived from p_{post} and p_{pre}. In Table 5.2 we show the definition of the transition function $f_{BFS}(R_r, R_c)$ that we use in BreadthPIS algorithm. In such table, the seven relations appearing in the rows correspond to the first argument R_r of f_{BFS}, whereas the same seven relations in the columns are those corresponding to the second argument R_c. Summarizing, f_{BFS} has 49 different combinations from its two arguments, being the smallest resulting sets composed of only one relation and the largest ones of five relations.

	$<$	m	o	f^{-1}	c	$=$	s
$<$	$<$	$<$	$<$	$<$	$<$	$<$	$<$
m	$<$	$<$	$<$	$<$	$<$	m	m
o	$<$	$<$	$\{<, m, o\}$	$\{<, m, o\}$	$\{<, m, o, f^{-1}, c\}$	o	o
f^{-1}	$<$	m	o	f^{-1}	c	f^{-1}	o
c	$\{<, m, o, f^{-1}, c\}$	$\{o, f^{-1}, c\}$	$\{o, f^{-1}, c\}$	c	c	c	$\{o, f^{-1}, c\}$
$=$	$<$	m	o	f^{-1}	c	$=$	s
s	$<$	$<$	$\{o, m, <\}$	$\{o, m, <\}$	$\{<, m, o, f^{-1}, c\}$	s	s

Table 5.2: Transition table for all the different intervals relations.

In this algorithm we take the second alternative given above and we use the transition function defined in Table 5.2 to generate the different candidates. The Algorithm 18, *MergePatterns*, obtains the inference associated with the TriMax of p_{post} and p_{pre}, set of relations $\mathcal{RS}$ (Line 1). Notice that, in order to infer the new relation, we need to know what relation exists in the paths from E_1 to E_{k+1} through all intermediate events E_j. The relations R_{E_1,E_j} between the events E_1 and E_j and $R_{E_j,E_{k+1}}$ between E_j and E_{k+1} are the arguments R_r and R_c of the transition function f_{BFS}. In order to simplify our algorithm, we search for that E_j that generates the minimum number of different relations.

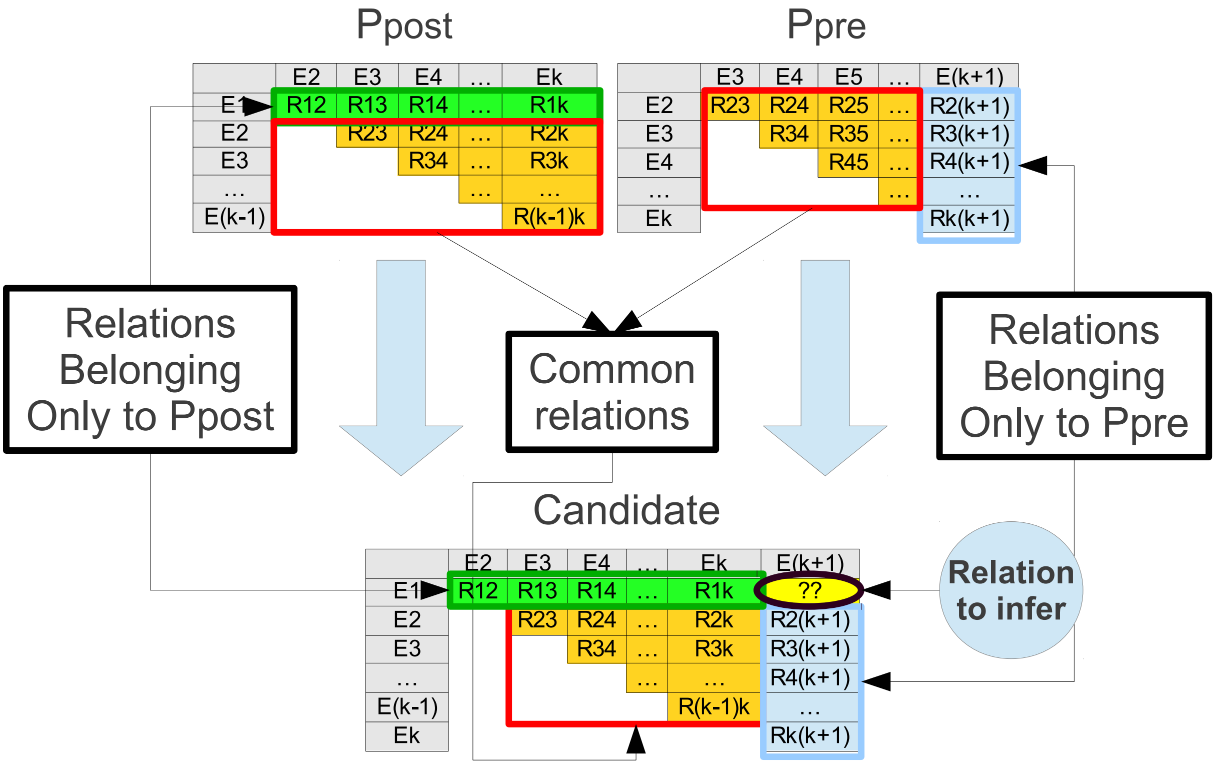


Figure 5.4: Generation of a candidate from two frequent patterns.

In Figure 5.5 we show that method for an example. Since we have to find the possible relations that exist between the event A (E_1) and the event E (E_{k+1}), all the three events B, C and D are a possible intermediate event (E_j). So, at first, we start with the B event, and as $R_{AB} =$ "c" and $R_{BE} =$ "$<$", we have $f_{BFS}(R_{AB}, R_{BE}) = \{<, m, o, f^{-1}, c\}$ as resulting set. Then, we continue exploring the paths to see if less candidates are generated. We see that using C, we find $f_{BFS}(R_{AC}, R_{CE}) = \{<, m, o\}$ and if we use D, we find $f_{BFS}(R_{AD}, R_{DE}) = \{<\}$. In this moment we can stop the search without analysing more paths since $f_{BFS}(R_{AD}, R_{DE})$ only contains a single relation, and therefore the candidate set cannot be reduced. In the same way, if we were lucky and in our first choice of E_j we had a result with a single relation, we would not need to continue with the process since we cannot find a set with less relations. Conversely, it can occur that we do not find any result with a single relation and, in that case, our result is the smallest set that we could find.

Once we infer the smallest relation set $\mathcal{RS}$, for each relation of $\mathcal{RS}$, we repeat the same process: we create its new TriMax as we previously saw in Figure 5.4 (Line 3 in Algorithm 18); we check if the candidate can be pruned (Line 6), whenever the parameter *IsPruneActivated* is true; and, finally, if the candidate is not pruned, we obtain the IdList associated with the 1-pattern form by the last event e of p_{pre} pattern (Line 9), and we create the IdList corresponding to the new candidate from the IdLists of p_{post} and e (Lines 10 and 11). The algorithm ends returning all the candidates derived from p_{post} and p_{pre} in Line 15.

We would like to go into detail about the call to the method CreateIdList (Line 10)

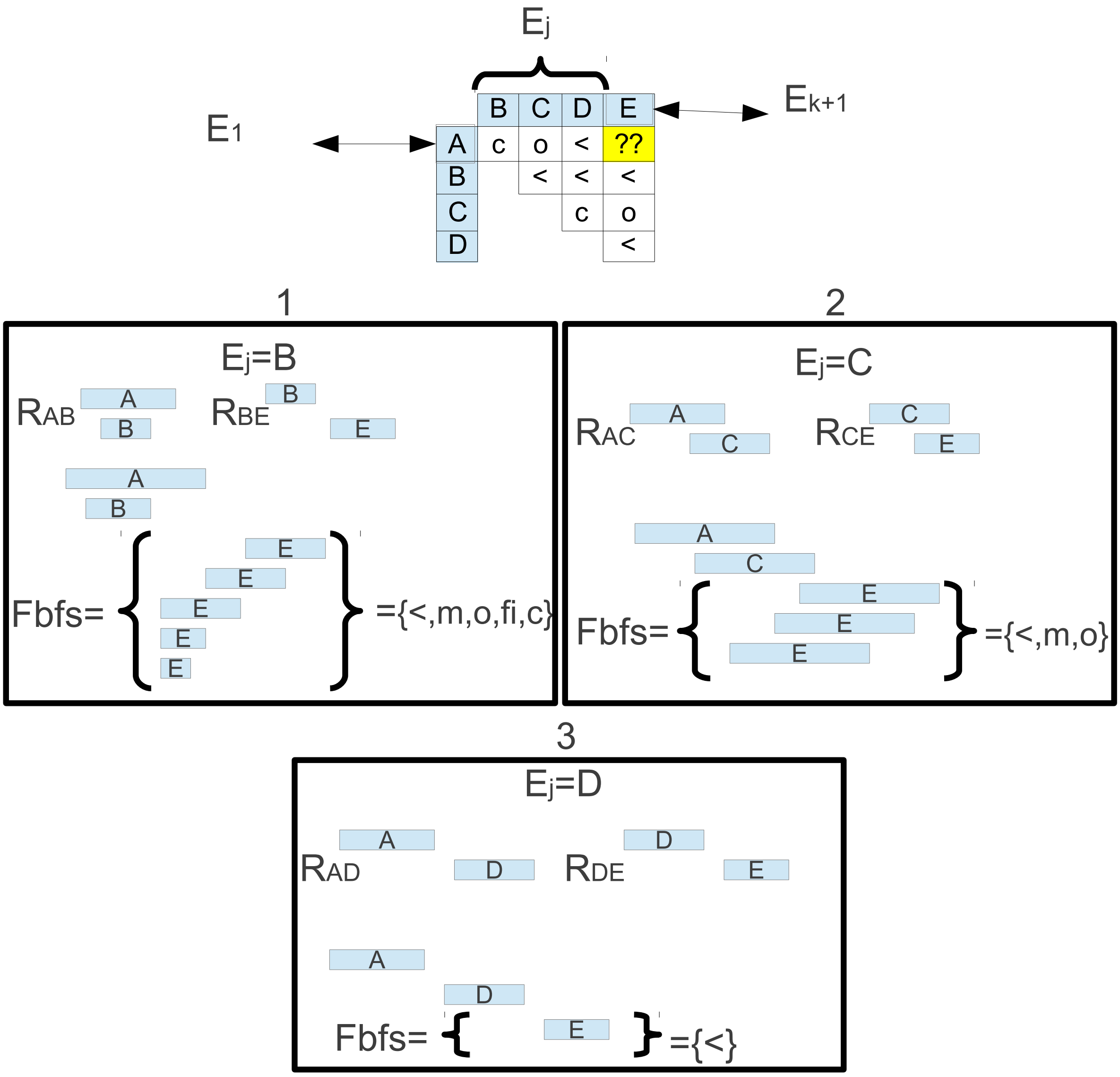


Figure 5.5: Process of choosing the smallest candidate relation set for BreadthPIS.

and explain how a new IdList is created from the IdList of p_{post} and p_{pre}. To obtain such IdList, we check that all the appearances of the last event of p_{pre} are present in the IdList of the new candidate previously generated (from p_{post} and p_{pre}). Note that we have to check necessarily the relations of all the appearances because if we only check the inferred relation, some of the previous events could not be related with the new event with the relation that we expect. Figure 5.6 shows the steps for merging two IdList of two patterns. The new IdList contains the IdList of C_1 reduced to those sequences that also contain one of the new relations inferred in Algorithm *MergePatterns* with respect the event E. In Figures 5.7 and 5.8 we show the calculation for the entries of the new IdList. In particular, we expose two different cases: one where the new IdList entry is empty and one where the IdList entry is not empty. Figure 5.7 shows the new empty IdList entry found when we try to merge the first C_1 entry ($\langle < 1, 10 >, < 3, 15 >, < 12, 40 >, < 20, 25 > \rangle$) and the first E_1 entry ($\langle < 30, 35 > \rangle$) of the first sequence. As we said, the interval that represents

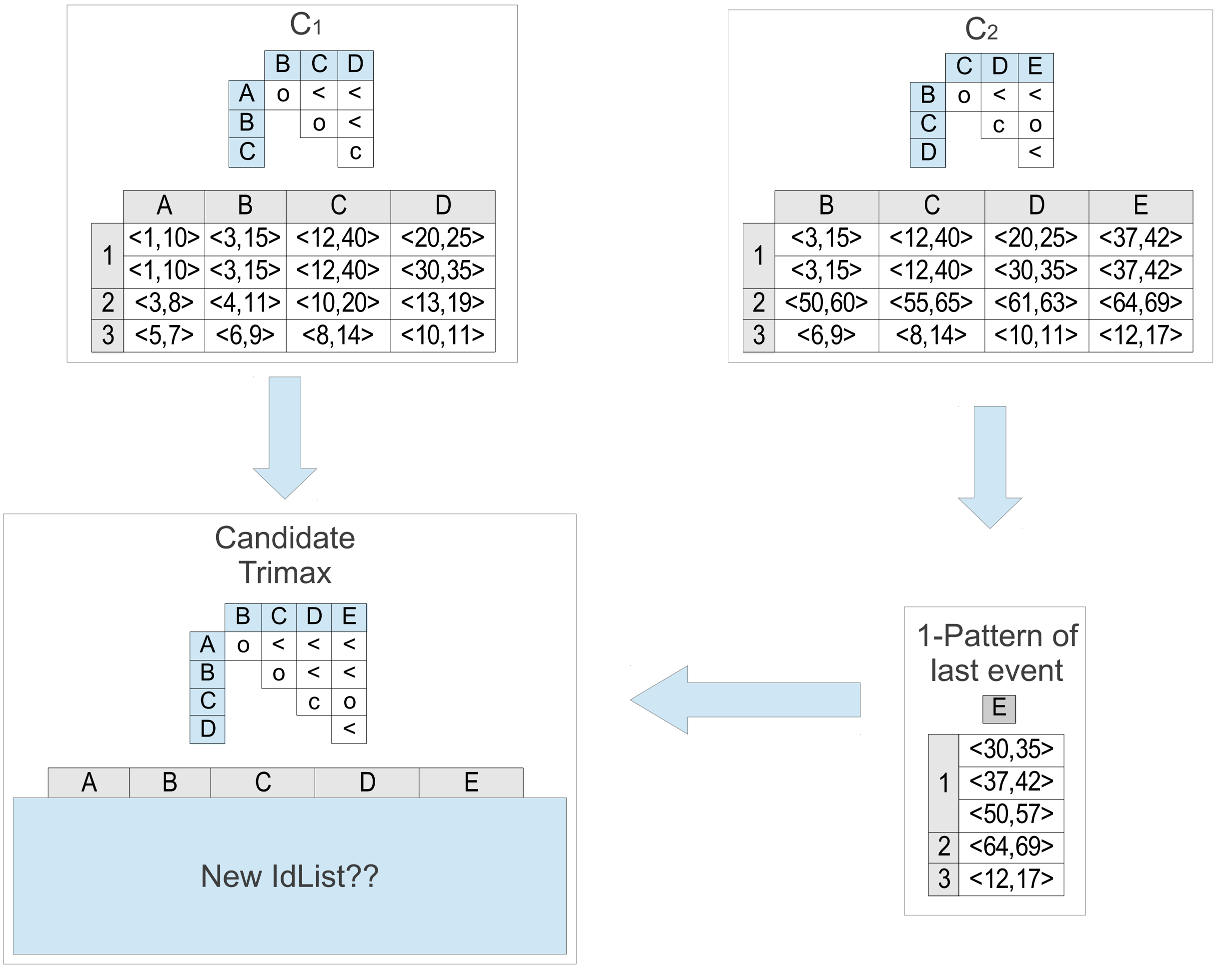
Algorithm 18 MergePatterns(p_{pre},p_{post},$IsPruneActivated$)

```
Require: The two pattern to merge $p_{post}$ and $p_{pre}$, a boolean indicating if the pruning
    method has to be executed $IsPruneActivated$
$\mathcal{RS}$ = TemporalReasoning($p_{post}.TriMax$,$p_{pre}.TriMax$)
for all $r \in \mathcal{RS}$ do
    $t$ = CreateTriMax($p_{post}.TriMax$,$p_{pre}.TriMax$,$r$)
    $pruned$ = false
    if $IsPruneActivated$ then
        $pruned$ = CheckPrune($t$)
    end if
    if not $pruned$ then
        $e = LastEventOf(p_{pre})$
        $NewIdList$ =CreateIdList($p_{post}.IdList$,$e.IdList$,$p_{post}.TriMax$,$t$)
        $c$ = pattern $\langle t, NewIdList \rangle$
        $\mathcal{C}_k = \mathcal{C}_k \cup \{c\}$
    end if
end for
return $\mathcal{C}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```

E has to accomplish all the relations denoted by the last column of the new candidate TriMax. We do that in decreasing order, starting from the last relation of the column $R_{k,k+1}$ up to the first relation of the column $R_{1,k+1}$. In our example we start checking if the last but one event of the new TriMax D has a “ $<$ ” relation with E, i.e. for the current entry, we check that the interval associated with D ($\langle < 20, 25 > \rangle$) *before* E ($\langle < 30, 35 > \rangle$) exists. Then, we continue doing the same process with the previous relation of the last column, which contains a “o'' relation between C and E. Unfortunately, we check that the interval associated with C in the current entry $\langle < 12, 40 > \rangle$ has not a “o'' relation with E ($\langle < 30, 35 > \rangle$), leading us to abort the check of that new possible entry.

Conversely, Figure 5.8 shows a successful case that makes a perfect matching between the first entry of C_1 IdList and the second entry of E IdList, both also associated with the first sequence. In this case, we have D ($\langle < 20, 25 > \rangle$) “ $<''$ E ($\langle < 37, 42 > \rangle$), C ($\langle < 12, 40 > \rangle$) “$o''$ E ($\langle < 37, 42 > \rangle$), B ($\langle < 3, 15 > \rangle$) “ $<''$ E ($\langle < 37, 42 > \rangle$), and A ($\langle < 1, 10 > \rangle$) “ $<''$ E ($\langle < 37, 42 > \rangle$), and thus all the relations in the last column of the new TriMax are accomplished. After processing the rest of entries, we finally have the resulting IdList, shown in Figure 5.9, where besides the second entry of E for the first sequence a new entry appears for the third sequence.

Finally, Algorithm 19 selects only those generated candidates whose support is at least *min_sup*. This method is very simple since the creation of the IdList have been already done through the method *GenerateCandidates*.



C1

	B	C	D
A	o	<	<
B		o	<
C			c

	A	B	C	D
1	<1,10>	<3,15>	<12,40>	<20,25>
	<1,10>	<3,15>	<12,40>	<30,35>
2	<3,8>	<4,11>	<10,20>	<13,19>
3	<5,7>	<6,9>	<8,14>	<10,11>

C2

	C	D	E
B	o	<	<
C		c	o
D			<

	B	C	D	E
1	<3,15>	<12,40>	<20,25>	<37,42>
	<3,15>	<12,40>	<30,35>	<37,42>
2	<50,60>	<55,65>	<61,63>	<64,69>
3	<6,9>	<8,14>	<10,11>	<12,17>

Candidate Trimax

	B	C	D	E
A	o	<	<	<
B		o	<	<
C			c	o
D				<

A	B	C	D	E

New IdList??

1-Pattern of last event

	E
1	<30,35>
	<37,42>
	<50,57>
2	<64,69>
3	<12,17>

Figure 5.6: Example of merging of two IdLists ($p_{post} = C_1$ and $p_{pre} = C_2$).

Algorithm 19 CountingOfSupport(C_k)

```
Require: the k-candidate set $C_k$ $\mathcal{F}_k = \emptyset$
for all $c_k \in C_k$ do
    if $c_k.support() > min\_sup$ then
        $\mathcal{F}_k = \mathcal{F}_k \cup c_k$
    end if
end for
return $\mathcal{F}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```

5.3 DepthPIS: an algorithm for mining point and intervals temporal events based on depth-first search

Let us now describe our DepthPIS algorithm. As in Algorithm BreadthPIS (Section 5.3), the structure of the patterns uses a TriMax and an IdList. One of the key points of this

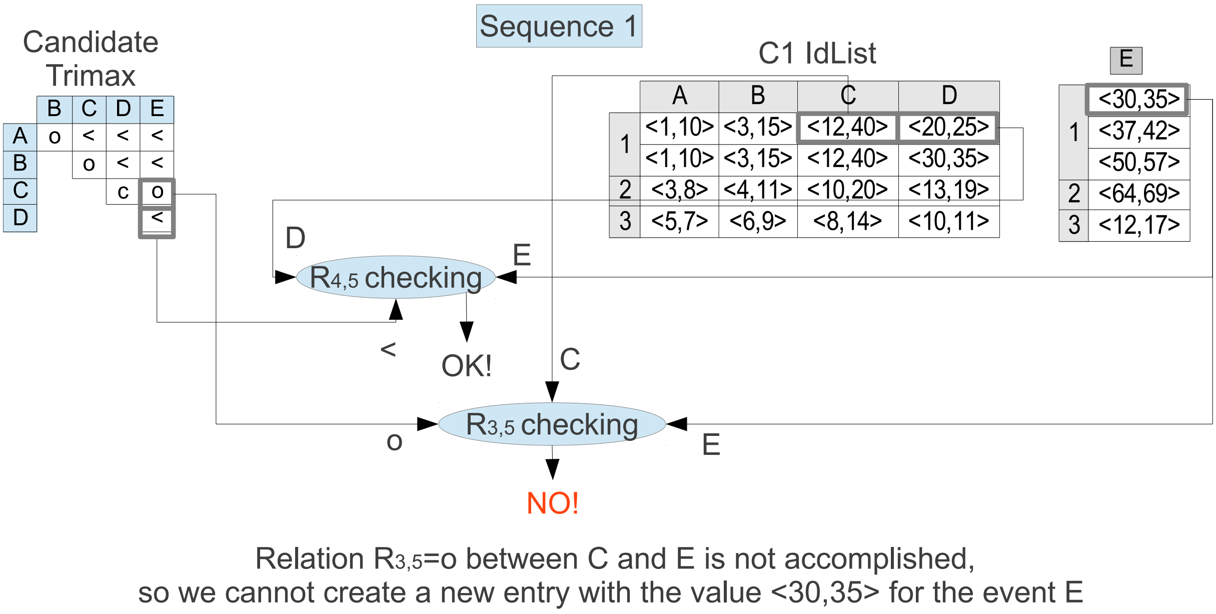

Sequence 1
Candidate Trimax
C1 IdList
E
<30,35>
R4,5 checking
R3,5 checking
D
E
<
OK!
C
o
E
NO!
Relation R3,5=o between C and E is not accomplished, so we cannot create a new entry with the value <30,35> for the event E

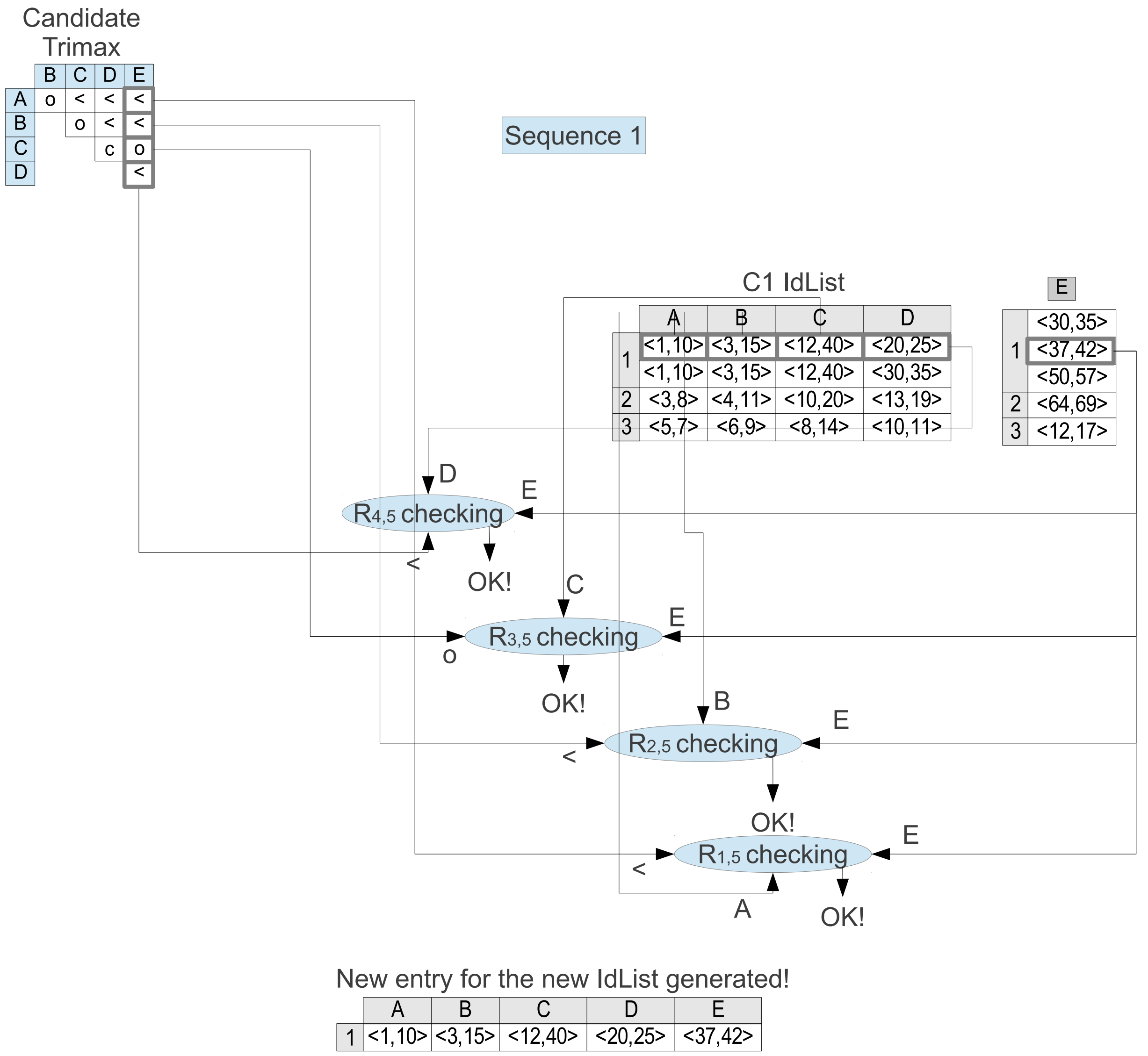
Figure 5.7: Example of a failed case in the calculation of an entry of the new IdList.

algorithms is the use of the concept of equivalence class given at the end of Section 5.1.

DepthPIS, shown in Algorithm 20, consists in a depth-first search that explores all the equivalence classes derived from all the frequent 1-patterns. In order to make the algorithms more efficient, frequent 1-patterns and 2-pattern have been previously found and all the frequent 2-patterns are inserted in the equivalence classes given by their 1-prefixes (Lines 9-11). Then, all the equivalence classes are explored and the algorithm ends returning the final set of frequent patterns $\mathcal{F}$.

Algorithm 21 shows the method *EnumerateFrequentSequences*, which is based on the method introduced in SPADE algorithm [Zaki, 2001]. The main idea consists of merging two k-equivalence classes $[Xy]$ and $[Xz]$, being both belonging to the k-class $[X]$, and having both the same common (k)-prefix X. In order to merge the patterns, we go over each member of the k-equivalence class $[X]$ ($k+1$-superpatterns of X) and merge it with all the $k+1$-patterns that are after it in $[X]$ (Lines 2-4). Then, the method *GenerateCandidates* is called and we obtain a set of superpatterns candidates for both Xy and Xz patterns. In this point, for each candidate, we have to create its IdList (Line 7), and the new equivalence class $[c]$ (Line 8), as well as check its support (Line 9). When we find a frequent candidate, we can add it to the final frequent pattern set (that corresponds to the equivalence class $[X]$) and, depending on its k-prefix, we insert it into the equivalence class ($[Xy]$ or $[Xz]$). Finally, the Algorithm 21 ends returning the frequent pattern set of the equivalence class $[X]$.

The method *GenerateCandidates*, shown in Algorithm 22, introduces the main changes added by this algorithm with respect to the original point-based Vertical Database Format ones. As the Algorithm previously shown in Section 5.5 does, the method receives two equivalence classes $[Xy]$ and $[Xz]$ as parameters and infers the candidates. Let see how a new candidate is inferred from $[Xy]$ and $[Xz]$ (schema shown in Figure 5.10). Since both $[Xy]$ and $[Xz]$ belong to the equivalence class $[X]$, their patterns have a common



Candidate Trimax

	B	C	D	E
A	o	<	<	<
B		o	<	<
C			c	o
D				<

C1 IdList

	A	B	C	D
1	<1,10>	<3,15>	<12,40>	<20,25>
	<1,10>	<3,15>	<12,40>	<30,35>
2	<3,8>	<4,11>	<10,20>	<13,19>
3	<5,7>	<6,9>	<8,14>	<10,11>

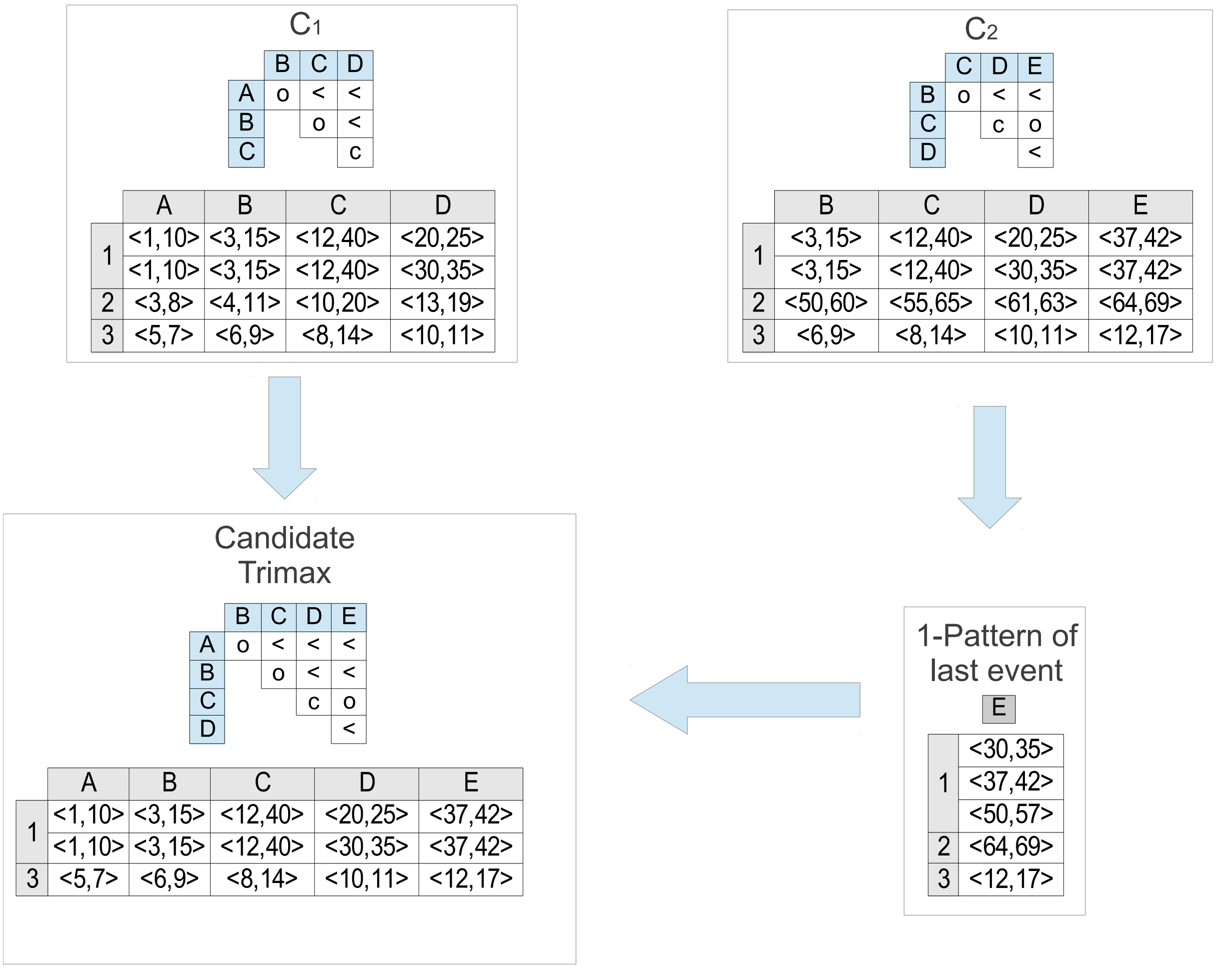
	E
1	<30,35>
	<37,42>
	<50,57>
2	<64,69>
3	<12,17>

New entry for the new IdList generated!

	A	B	C	D	E
1	<1,10>	<3,15>	<12,40>	<20,25>	<37,42>

Figure 5.8: Example of a successful case in the calculation of an entry of the new IdList.

$(k-1)$-prefix, and, therefore, their associated TriMax share several cells. Concretely, Xy and Xz share all the relations except their last row, both highlighted and joined by arrows in Figure 5.10. Thus, a new candidate is formed by adding the last column of the TriMax of Xz to the TriMax of Xy. In this point, as in Section 5.5, we see that we still need to infer the relation between the last event of Xy and Xz. Using a new transition function $f_{DFS}(R_r, R_c)$, defined by the transition Table (5.3). This new transition function is different to that one used in BreadthPIS algorithm, and it has been explicitly created for this DepthPIS algorithm. The arguments of this function are the relation that connect an event E_j, $\forall 1 \leq j < k+1$, to the last but one event E_k and the relation that connects the same event E_j to the last event E_{k+1}. In this case $E_k = y$, $E_{k+1} = z$, and E_j takes as values all the events in X. In that table the seven direct relations that we use as rows correspond to the argument R_r, whereas the columns correspond to the second argument R_c. As in BreadthPIS algorithm, we also have 49 different resulting sets with a minimum



C1

	B	C	D
A	o	<	<
B		o	<
C			c

	A	B	C	D
1	<1,10>	<3,15>	<12,40>	<20,25>
	<1,10>	<3,15>	<12,40>	<30,35>
2	<3,8>	<4,11>	<10,20>	<13,19>
3	<5,7>	<6,9>	<8,14>	<10,11>

C2

	C	D	E
B	o	<	<
C		c	o
D			<

	B	C	D	E
1	<3,15>	<12,40>	<20,25>	<37,42>
	<3,15>	<12,40>	<30,35>	<37,42>
2	<50,60>	<55,65>	<61,63>	<64,69>
3	<6,9>	<8,14>	<10,11>	<12,17>

1-Pattern of last event

	E
1	<30,35>
	<37,42>
	<50,57>
2	<64,69>
3	<12,17>

Candidate Trimax

	B	C	D	E
A	o	<	<	<
B		o	<	<
C			c	o
D				<

	A	B	C	D	E
1	<1,10>	<3,15>	<12,40>	<20,25>	<37,42>
	<1,10>	<3,15>	<12,40>	<30,35>	<37,42>
3	<5,7>	<6,9>	<8,14>	<10,11>	<12,17>

Figure 5.9: Example of merging of two IdLists. Final result.

cardinality of one and a maximum of thirteen. In Figure 5.11 we can see the process in DepthPIS for the TriMax given. In that example, a set of five relations is the minimum cardinality found.

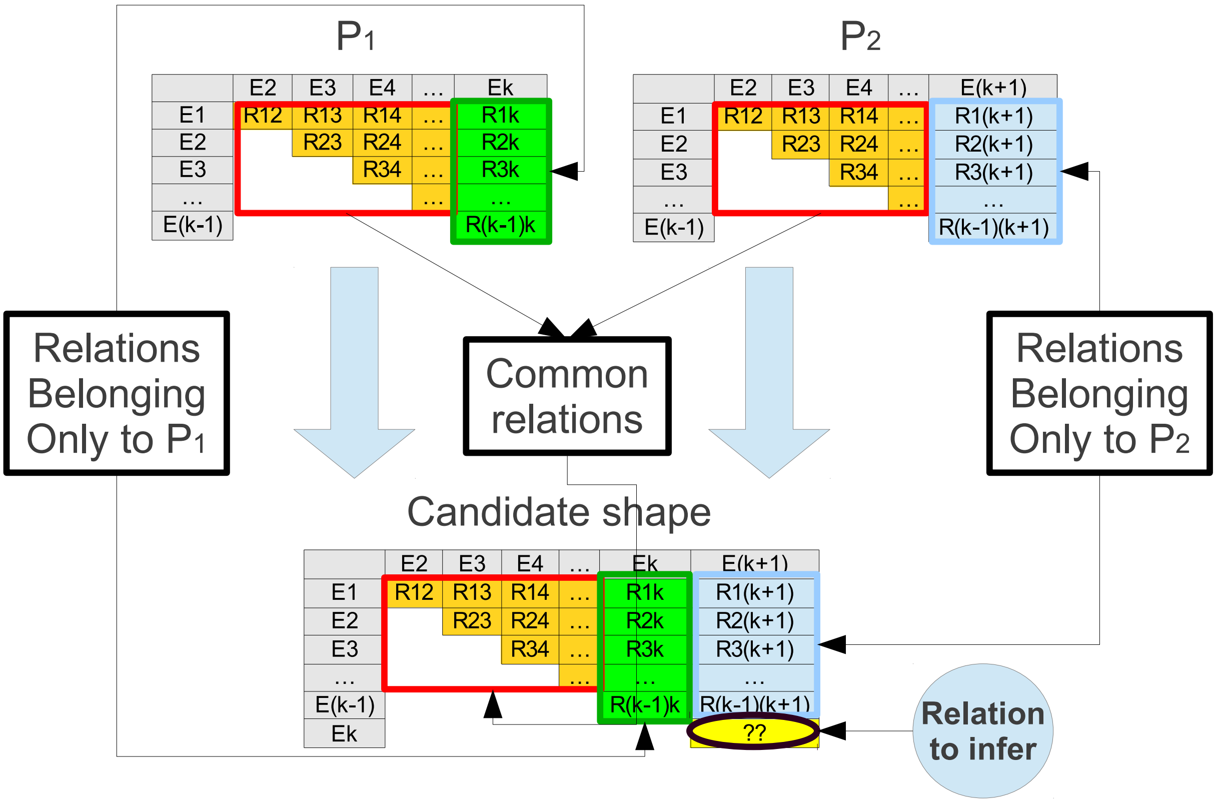
Algorithm 22, GenerateCandidates, infers the TriMax of $[Xy]$ and $[Xz]$, obtaining a set of relations $\mathcal{RS}$ (Line 1). Then, for each relation of $\mathcal{RS}$, we repeat the same process of creating its new TriMax as we previously saw in Figure 5.4 (Lines 7 and 9). The algorithm ends returning all the candidates derived from Xy and Xz in Line 14. The aim of lines 5-7 is exactly the same as line 9. The motivation of this lines comes from the existence of inverse relations in some resulting relation sets from the transition table (Table 5.3).

For instance, the thirteen relations are possible when two *before* relations are given as parameters. In order to avoid the use of inverse relations ($>$, d, si, oi, mi and f) we have to swap the order of the events as shown in Figure 5.12.

Algorithm *CreateIdList* is exactly the same as that used in BreadthPIS algorithm (see Section 5.2). Figure 5.13 shows the final result of merging two IdList for the two patterns given.

Algorithm 20 DepthPIS()

```
$\mathcal{F}_1 = \{$frequent items$\}$
for all $X \in \mathcal{F}_1$ do
    add $X$ to equivalence class $[X]$
    $\mathcal{EC} = \mathcal{EC} \cup [X]$
end for
$\mathcal{F} = \mathcal{F}_1$
$\mathcal{F}_2 =$getFrequent2Sequences$(\mathcal{F}_1)$
for all $Y \in \mathcal{F}_2$ do
    add $Y$ to equivalence class $[Y]$
    $p =$ 1-prefix of $Y$
    add $Y$ to equivalence class $[p]$
end for
$\mathcal{F} = \mathcal{F} \cup \mathcal{F}_2$
for all $[X] \in \mathcal{EC}$ do
    $\mathcal{F}_k =$EnumerateFrequentSequences$([X])$
    $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$
end for
return $\mathcal{F}$
Ensure: The frequent pattern set $\mathcal{F}$
```


P1
P2
Relations Belonging Only to P1
Common relations
Relations Belonging Only to P2
Candidate shape
Relation to infer

	E2	E3	E4	…	Ek
E1	R12	R13	R14	…	R1k
E2		R23	R24	…	R2k
E3			R34	…	R3k
…				…	…
E(k-1)					R(k-1)k

	E2	E3	E4	…	E(k+1)
E1	R12	R13	R14	…	R1(k+1)
E2		R23	R24	…	R2(k+1)
E3			R34	…	R3(k+1)
…				…	…
E(k-1)					R(k-1)(k+1)

	E2	E3	E4	…	Ek	E(k+1)
E1	R12	R13	R14	…	R1k	R1(k+1)
E2		R23	R24	…	R2k	R2(k+1)
E3			R34	…	R3k	R3(k+1)
…				…	…	…
E(k-1)					R(k-1)k	R(k-1)(k+1)
Ek						??

Figure 5.10: Example of merging of two TriMax.

Algorithm 21 EnumerateFrequentSequences($[X]$)

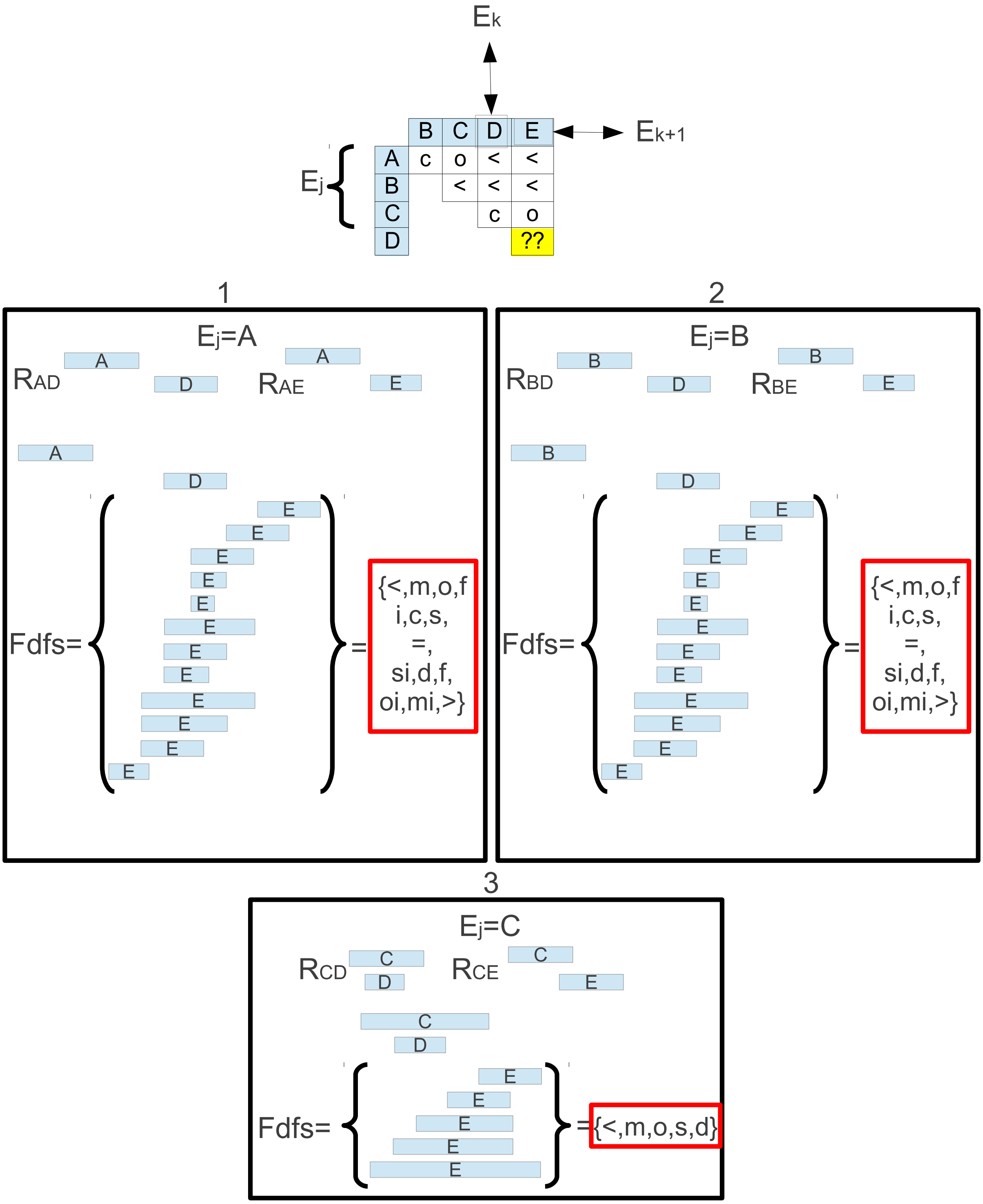
```
Require: the current equivalence class $[X]$ on which we search for the frequent patterns in a DFS way
$\mathcal{F}_k = \emptyset$
$components$ =members of $[X]$ extended with only one more event than $X$
for all $[Xy] \in components$ do
  for all $[Xz] \in components$ that appear after $[Xy]$ do
    $C_k$ =GenerateCandidates($[Xy]$,$[Xz]$)
    for all $c \in C_k$ do
      newIdList = CreateIdList($c$,$[Xy]$,$[Xz]$)
      Create a new pattern $c$ with $newIdList$, and insert $c$ in the equivalence class $[c]$
      if $c.support() \geq min\_sup$ then
        $\mathcal{F}_k = \mathcal{F}_k \cup \{c\}$
        if the pattern $Xy$ is a prefix of $c$ then
          add $[c]$ to $[Xy]$
        else
          add $[c]$ to $[Xz]$
        end if
      end if
    end for
  end for
  if we have just found any frequent pattern then
    $\mathcal{F}_k = \mathcal{F}_k \cup$ EnumerateFrequentSequences($[Xy]$)
  end if
end for
return $\mathcal{F}_k$
Ensure: The frequent pattern set $\mathcal{F}_k$
```

5.4 Mining points and intervals

In the description of the previous two algorithms we have given a transition table for each one. This transition table is an enumeration for all the possible arguments that we can have in their corresponding transition function but it has been only defined for interval items. In those tables, we suppose two qualitative interval relations as arguments, finding another qualitative interval argument as result. Since in the calculation made by the transition function we have three different items i_i, i_j and i_k (being i_i and i_j related by R_r; and i_j and i_k by R_c), and those items can be either a point or an interval, we have $2^3 = 8$ different combinations.

In order to deal with all the combinations we have two alternatives: 1) to use an only transition table and taking into account if we are referring to a point or an interval; or 2) to use eight transition tables (seven additional tables to that proposed in Section 5.2 or Section 5.3) and to deal directly there with the whole range of possibilities.

Regarding the first alternative, we need to redefine the translation into points for those relations that are not in the relations between point and intervals, or interval and



	B	C	D	E
A	c	o	<	<
B		<	<	<
C			c	o
D				??

Figure 5.11: Process of choosing the smallest candidate relation set for DepthPIS.

points, which Meiri proposed [Meiri, 1996]. These two relations that not exist in the Meiri's relations are $\{m, o\}$. If we redefine those relations, we can simply use the table previously defined and whenever we take into account point, intervals or both, we create the candidates with the relation set given by the transition function.

Algorithm 22 GenerateCandidates([Xy],[Xy])

Require: The two equivalence classes that, from their patterns, we use for finding the different possible candidates.

```
$\mathcal{RS}$ = TemporalReasoning([$Xy$].TriMax,[$Xz$].TriMax)
$C_k = \emptyset$
if $\mathcal{RS} \neq \emptyset$ then
    for all $r \in \mathcal{RS}$ do
        if IsAnInverseRelation($r$) then
            $ri$ =getDirectRelation($r$)
            $c$ =createTriMaxWithInverseRelations($Xy$.TriMax,$Xz$.TriMax,$ri$)
        else
            $c$ =createTriMax($Xy$.TriMax,$Xz$.TriMax,$r$)
        end if
        $C_k = c$
    end for
end if
return $C_k$
```

Ensure: The candidate set C_k derived from [Xy] and [Xz]

Nevertheless, this first alternative carries two problems. Firstly we have to make more complicated to check whether a relation is maintained and, secondly, we have to deal with a lot of candidates that will be infrequent when we deal with points. For instance, if we are using the transition function given in DepthPIS algorithm and we have three points A, B and C such that $A < B$ and $B < C$, our transition function give us a candidate set of thirteen relations, whereas only two different relations are possible between points and, therefore, we will surely have 11 infrequent candidates.

In regards to the second alternative, we define seven additional transition tables to complete the different combinations between three items, i_i, i_j and i_k, which are immersed when we create candidates. The seven tables are built from those define in BreadthPIS and DepthPIS and the creation method of each table is the following:

1. If the two first items are points, we remove all the rows different from relations $<$ and $=$. If, on the contrary, they are a point followed by an interval, we remove all the rows different from relations $<$ and s. If we have an interval followed by a point, we remove all the rows that do not correspond to the relations $<$, c and f^{-1}.

2. If the second and third items are points, we remove all the columns different from relations $<$ and $=$. If, on the contrary, they are a point followed by an interval, we remove all the columns different from relations $<$ and s. If we have an interval followed by a point, we remove all the columns that do not correspond to the relations $<$, c and f^{-1}.

3. Finally, we remove from the relation sets those relations that are not associated with the relation from the first item to the third one. Thus, if the first and third items are points we remove from each relation set all the relations different from $<$ and $=$. If, on the contrary, they are a point and an interval, we remove from each

	$<$	m	o	f^{-1}	c	$=$	s
$<$	$\{<, m, o, f^{-1}, c, s, =, si, d, f, oi, mi, >\}$	$\{d, f, oi, mi, >\}$	$\{d, f, oi, mi, >\}$	$>$	$>$	$>$	$\{d, f, oi, mi, >\}$
m	$\{<, m, o, f^{-1}, c\}$	$\{s, =, si\}$	$\{d, f, oi\}$	mi	$>$	mi	$\{d, f, oi\}$
o	$\{c, f^{-1}, o, m, <\}$	$\{c, f^{-1}, o\}$	$\{o, f^{-1}, c, si, =, s, d, f, oi\}$	$\{c, si, oi\}$	$\{c, si, oi, mi, >\}$	oi	$\{d, f, oi\}$
f^{-1}	$<$	m	$\{o, s, d\}$	$\{f^{-1}, =, f\}$	$\{c, si, oi, mi, >\}$	f	d
c	$<$	$<$	$\{<, m, o, s, d\}$	$\{<, m, o, s, d\}$	$\{<, m, o, f^{-1}, c, s, =, si, d, f, oi, mi, >\}$	d	d
$=$	$<$	m	o	f^{-1}	c	$=$	s
s	$\{<, m, o, f^{-1}, c\}$	$\{o, f^{-1}, c\}$	$\{o, f^{-1}, c\}$	c	c	si	$\{s, =, si\}$

Table 5.3: Transition table for all the different intervals relations.

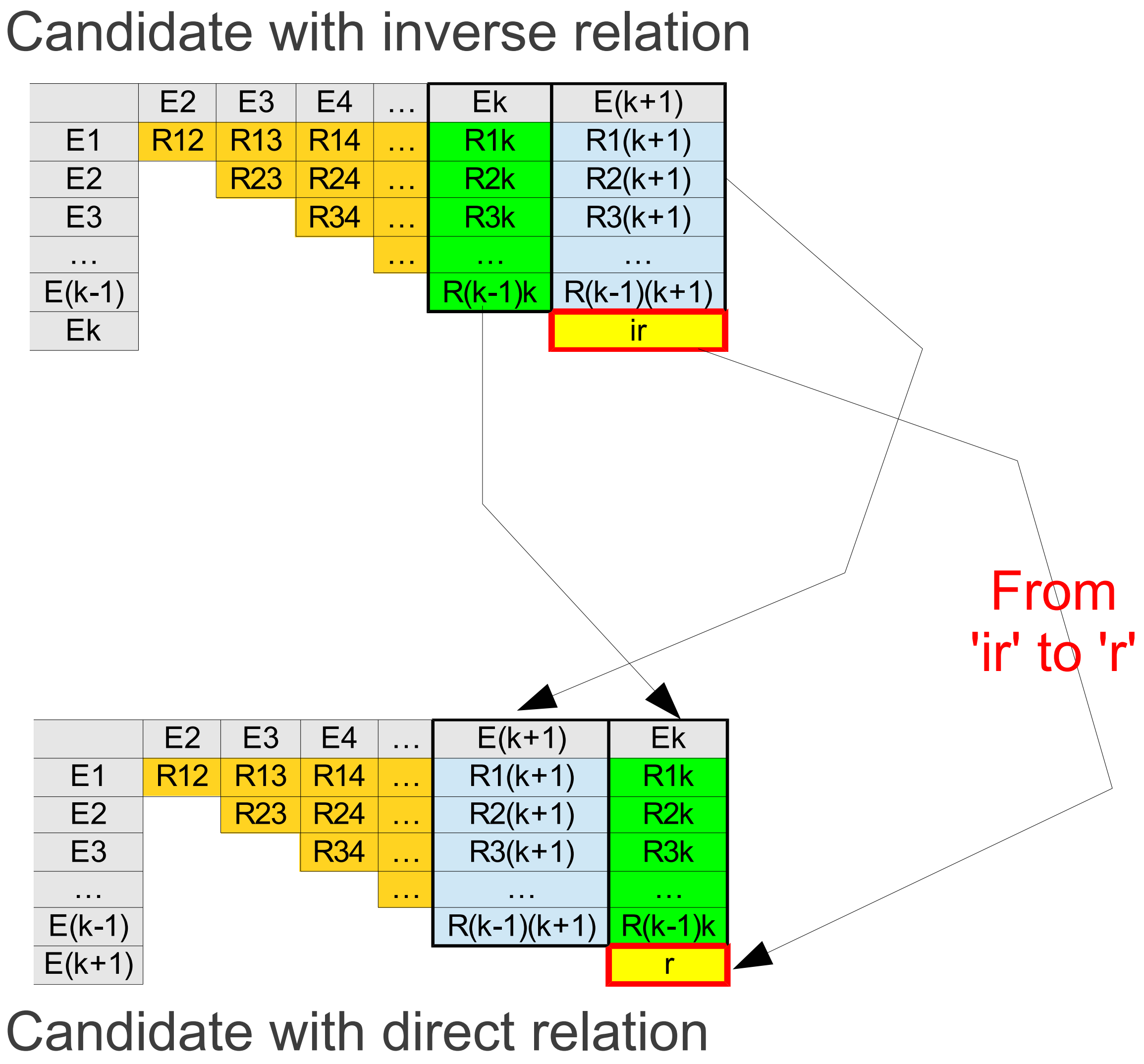
relation sets all the relations apart from relations $<$ and s. If we have that the first item is an interval and the second one is a point, we remove all the columns that do not correspond to the relations $<$, c and f^{-1}.

In our algorithms, BreadthPIS and DepthPIS, we use this second alternative and, when we generate all these tables first, we only look for the relation sets in the corresponding table depending on the type of the events. Therefore, thanks to this last alternative we can easily manage the candidate generation of both points and interval events.

5.5 Experimental results

The setting of the experiments is the same to the explained in Section 3.4.

In all the comparative studies made we find two different situations: 1) executions where both algorithms find a similar number of candidates and 2) executions where the


Candidate with inverse relation

	E2	E3	E4	…	Ek	E(k+1)
E1	R12	R13	R14	…	R1k	R1(k+1)
E2		R23	R24	…	R2k	R2(k+1)
E3			R34	…	R3k	R3(k+1)
…				…	…	…
E(k-1)					R(k-1)k	R(k-1)(k+1)
Ek						ir

From 'ir' to 'r'

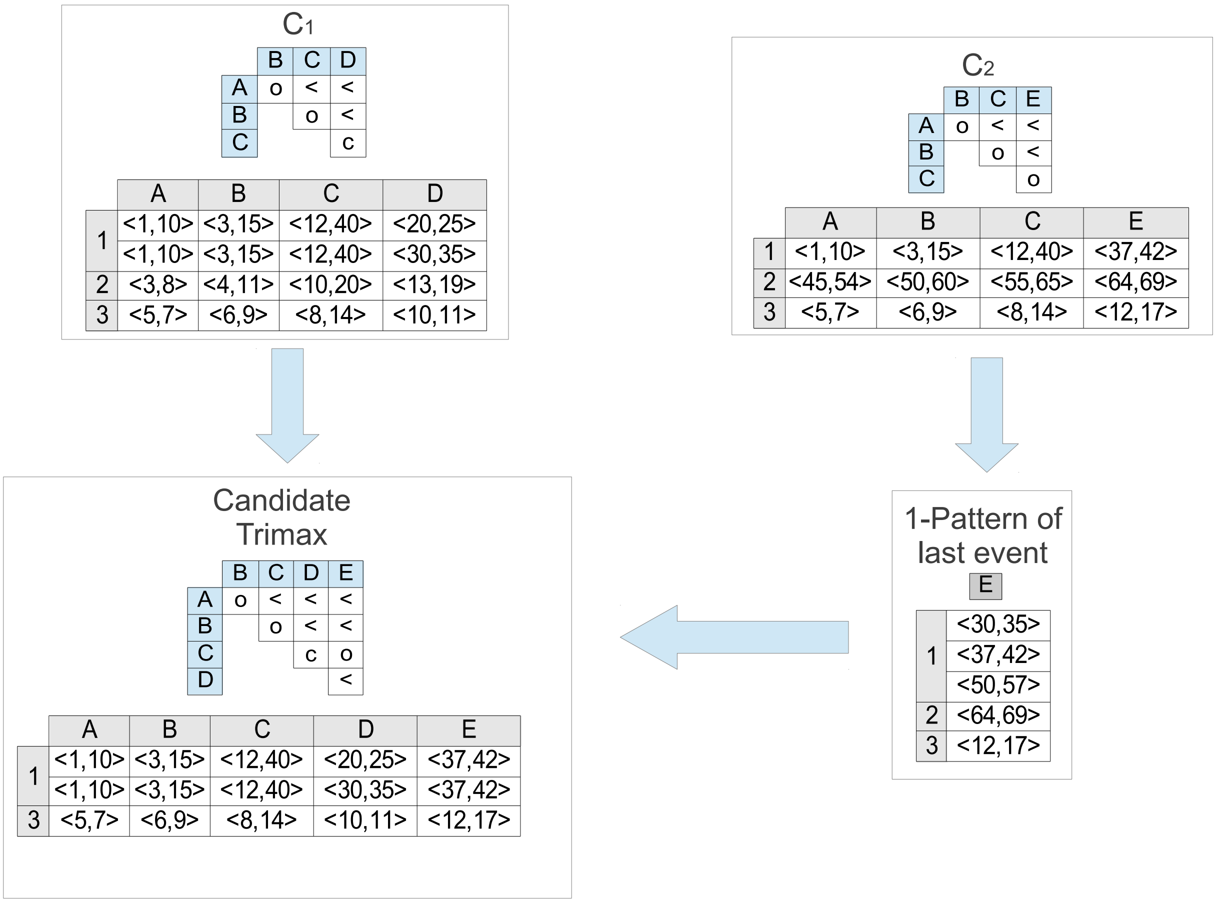
	E2	E3	E4	…	E(k+1)	Ek
E1	R12	R13	R14	…	R1(k+1)	R1k
E2		R23	R24	…	R2(k+1)	R2k
E3			R34	…	R3(k+1)	R3k
…				…	…	…
E(k-1)					R(k-1)(k+1)	R(k-1)k
E(k+1)						r

Candidate with direct relation

Figure 5.12: Conversion of a TriMax with an inverse relations.

number of candidates found by BreadthPIS is clearly less than the amount generated by DepthPIS.

As for the first situation, Figure 5.14 show plots where the running times for BreadthPIS and DepthPIS are very similar. Figure 5.14 shows what happens with both algorithms when we deal with large databases (10000 sequences). Plots 5.14A and 5.14B have medium sequence length (20 transactions on average), medium-high transaction length (20 items on average), medium pattern length (10 items on average), and 500 and 1000 different items for the database. The datasets that correspond to the plots 5.14C and 5.14D have larger sequences (40 transactions on average) and smaller transactions (10 items on average). In both rows of plots, the same phenomenon is repeated, having a little domain of DepthPIS over BreadthPIS for the denser database ($n = 500$), and very similar execution when we use a greater n value. The same phenomenon occurs in plots 5.14C and 5.14D, and even, in 5.14D plot, BreadthPIS cannot get the final low supports due to its memory overflow.



	B	C	D
A	o	<	<
B		o	<
C			c

	A	B	C	D
1	<1,10>	<3,15>	<12,40>	<20,25>
	<1,10>	<3,15>	<12,40>	<30,35>
2	<3,8>	<4,11>	<10,20>	<13,19>
3	<5,7>	<6,9>	<8,14>	<10,11>

	B	C	E
A	o	<	<
B		o	<
C			o

	A	B	C	E
1	<1,10>	<3,15>	<12,40>	<37,42>
2	<45,54>	<50,60>	<55,65>	<64,69>
3	<5,7>	<6,9>	<8,14>	<12,17>

	E
1	<30,35>
	<37,42>
	<50,57>
2	<64,69>
3	<12,17>

	B	C	D	E
A	o	<	<	<
B		o	<	<
C			c	o
D				<

	A	B	C	D	E
1	<1,10>	<3,15>	<12,40>	<20,25>	<37,42>
	<1,10>	<3,15>	<12,40>	<30,35>	<37,42>
3	<5,7>	<6,9>	<8,14>	<10,11>	<12,17>

Figure 5.13: Example of merging of two IdLists.

All in all, Figure 5.14 refer to execution where the candidates generated for both transition functions f_{BFS} and f_{DFS} are very similar in results. Figure 5.15 shows the number of candidates generates for plots 5.14A and 5.14C (plots 5.15B and 5.15C, respectively). We can see that the shapes are consistent with execution time. In addition, we see that the rough increments that we have in some plots (for instance the remarkable change in supports 0.64-0.6 in plot 5.14A or 0.5-0.48 in plot 5.14C are corresponded by the same increment in the number of candidates in their respective plots of Figure 5.15.

As far as the second situation is concerned, where the number of candidates created by DepthPIS exceeds by far the number created by BreadthPIS, Figures 5.16 and 5.17 show different comparatives. In this case, the running time taken by DepthPIS is much longer than the time used by BreadthPIS. In Figure 5.16, we can see the execution time when we have a moderated number of transactions per sequence (psl=20) with low transaction average length (ptl from 2 up to 10). We see similar curves for both BreadthPIS and DepthPIS, but BreadthPIS always outperforms DepthPIS. In the 5.16A plot the execution time of BreadthPIS is faster than DepthPIS with a factor of 2. With fewer items per transaction, Plots 5.16B and 5.16C, BreadthPIS still outperforms DepthPIS but, for low supports (after 0.43 in 5.16B plot and after 0.11 in 5.16C plot) we can see that BreadthPIS have problems and even cannot complete the execution because of memory overflow. The 5.16D plot shows that the difference between DepthPIS and BreadthPIS is increased when

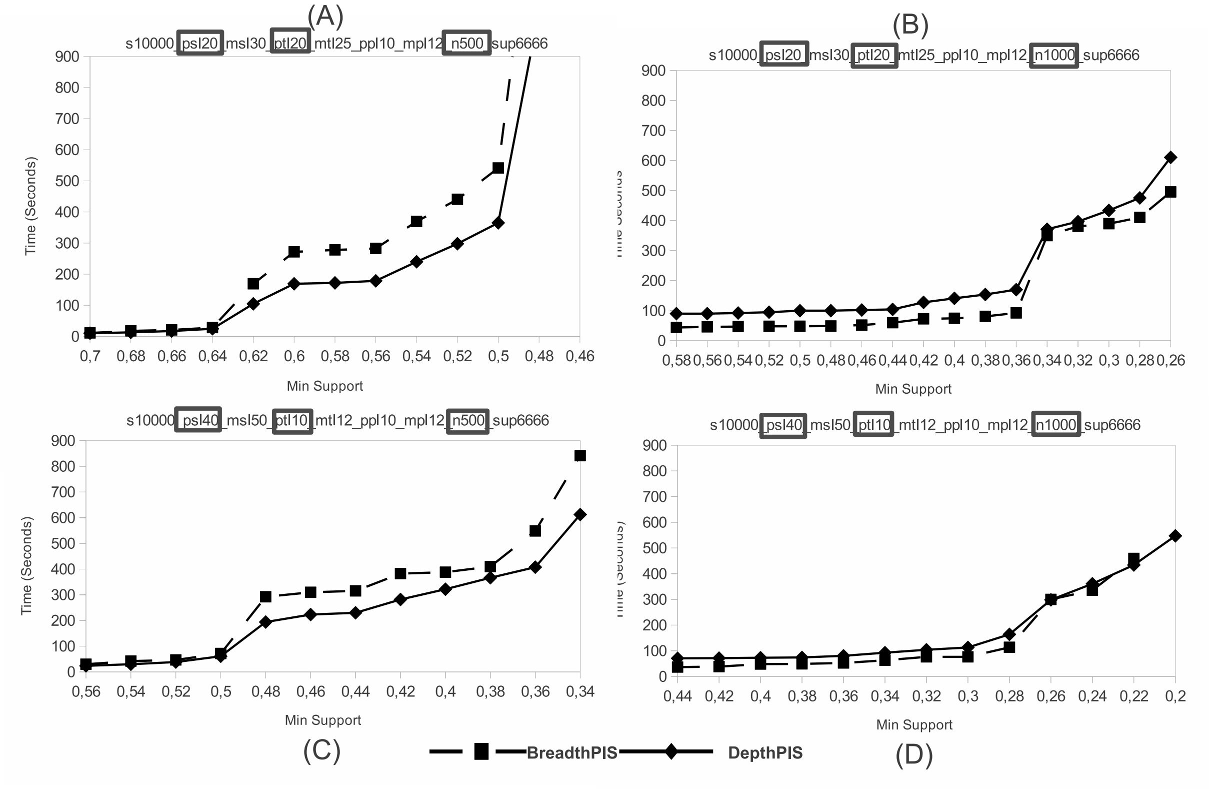


Figure 5.14: Varying support for datasets where the number the candidates is very closed for both algorithms BreadthPIS and DepthPIS. $s1000\_psl20–40\_msl30–50\_ptl10–20\_mtl12–25\_ppl10\_mpl12\_n500–1000$.

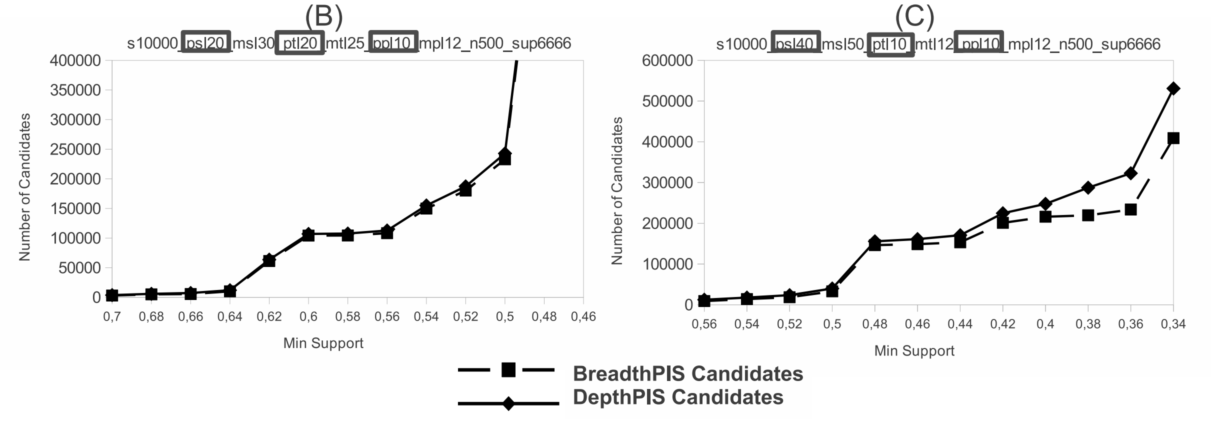


Figure 5.15: Candidate number for BreadthPIS and DepthPIS in plots 5.14A and 5.14C.

we change the transaction average length (10 items), considering 100 different items for the whole database. As before, BreadthPIS cannot reach the lowest support and even, in the last time point it exists an anomaly due to the memory overflow problems. All the databases exposed by Figure 5.16 specially have patterns with "<" relations since the distances between different transactions are quite long and, as we will see in the next Section, f_{BFS} is much more efficient than f_{DFS} for some particular inferences.

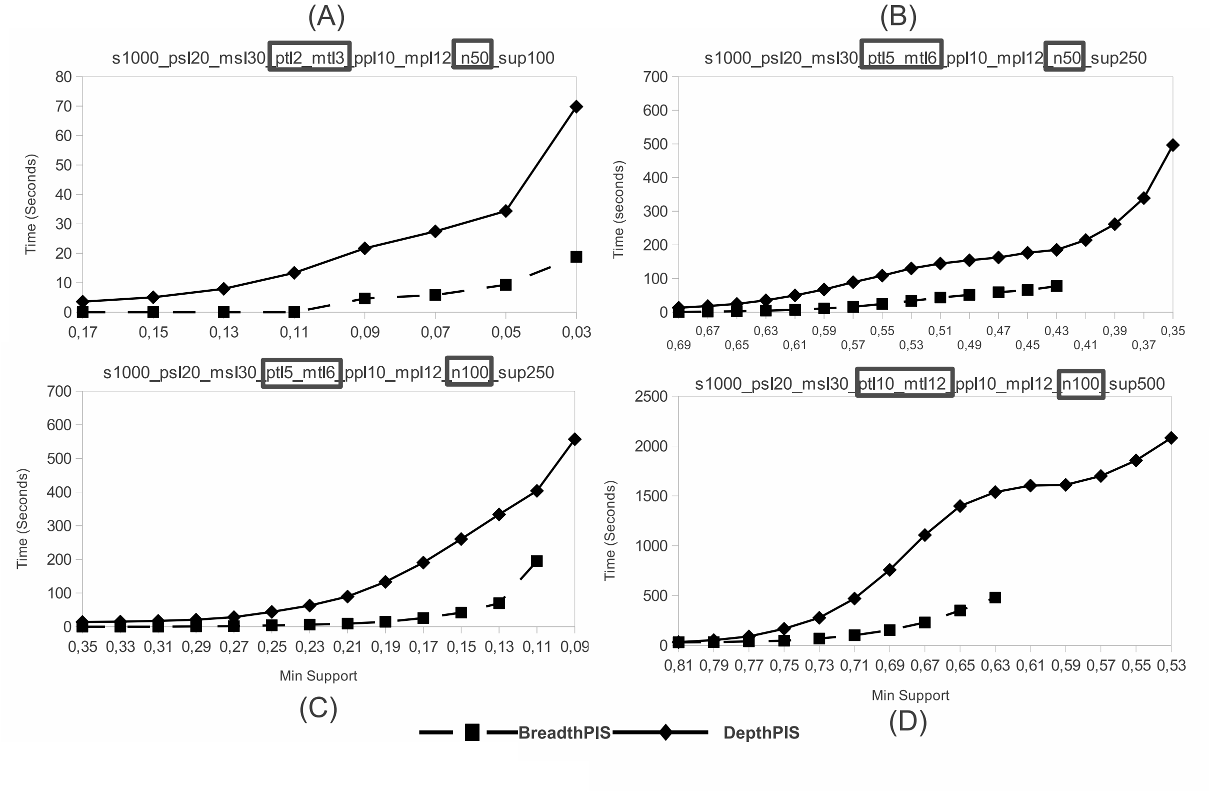


Figure 5.16: Varying support for datasets where the number the candidates generated is less for BreadthPIS than for DepthPIS. $s1000\_psl20\_msl30\_ptl2$–5–10 $\_mtl3$–6–12$_ppl10$ $\_mpl12\_n50$–100.

Figure 5.17 includes several plots where its databases have transactions very close in time and, therefore, for medium and low supports, very different Allen's relations are present in their patterns. The previous situation still remains: BreadthPIS outperforms DepthPIS and, for low supports, DepthPIS can be executed and BreadthPIS either does not finish or obtains an extremely high time non-correlated with the number of candidates generated. These experiments use databases with 1000 sequences that have moderated values as sequence and transaction lengths (psl=10 ptl=10 and 20), with an average of 8 events for pattern and a number of different events of 50 and 100 (n). In particular, plots 5.17A and 5.17B have 10 events as transaction length whereas plots 5.17C and 5.17D have 20 events on average. Conversely plots 5.17A and 5.17C deal with a number of different events of 50, while 5.17B and 5.17D have a number of 100. In general, all the experiments show the same behaviour except for plot 5.17D. In that plot, we can see that the difference between BreadthPIS and DepthPIS curves is wider than those that appear in plots 5.17A, 5.17B and 5.17C. This phenomenon is due to the larger number of candidates created in that concrete database with respect to the other ones, as it can be seen in Figure 5.18, where all the candidates generated in the plots of Figure 5.17 are shown.

To sum up, in this second situation BreadthPIS is always faster than DepthPIS but cannot get all the results for low supports due to memory overflow problems. So, we will need to establish a trade-off to decide what is better in the database that we want to mine: speed versus lower support results. As we expose above, the problems related to

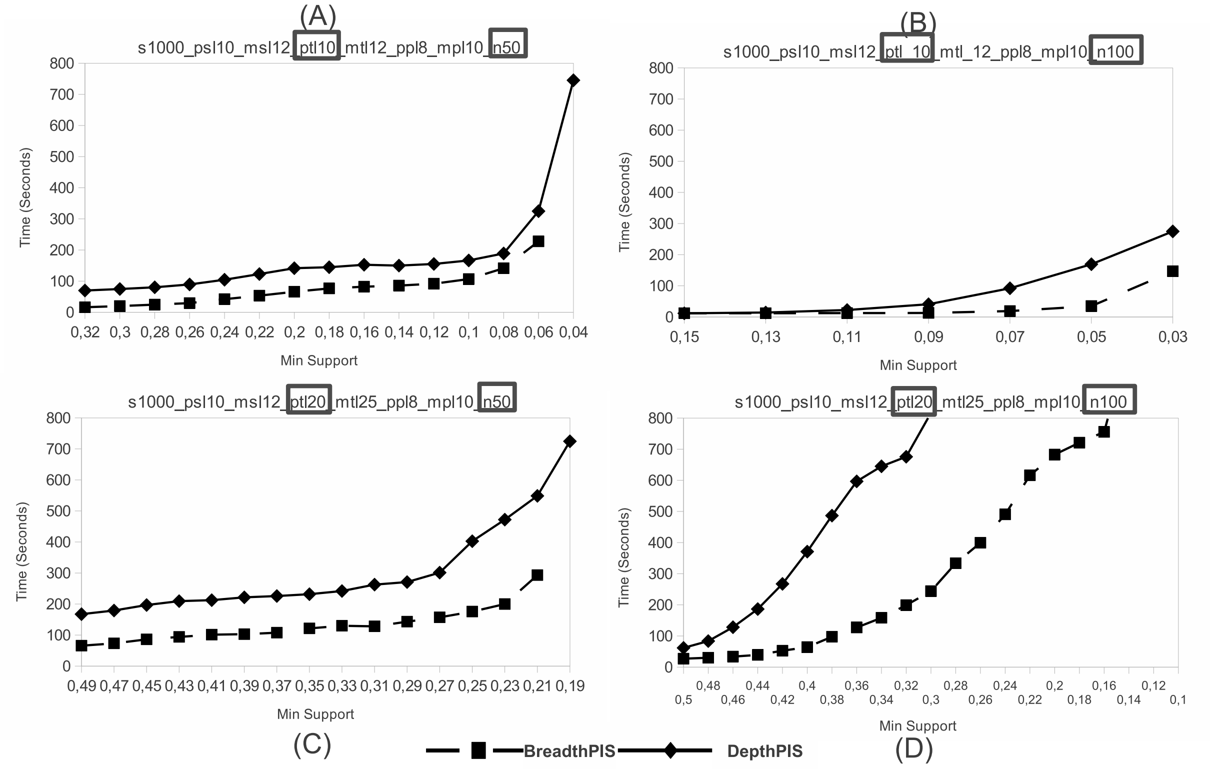


Figure 5.17: Varying support for datasets where the number the candidates generated is less for BreadthPIS than for DepthPIS. $s1000\_psl10\_msl12\_ptl10$–20_mtl12–$25\_ppl8\_mpl10\_n50$–100.

BreadthPIS come from its breadth-first search, whereas the difference in the time execution comes from the number of candidates that each algorithm generates. So as to show this latter phenomenon, Figure 5.19 shows a ratio that comes from dividing the number of DepthPIS candidates generated by the number of BreadthPIS candidates for experiments 5.16D (plot 5.19A) and 5.17D (plot 5.19B). We can see that the number of candidates of DepthPIS is always a factor bigger than the number of candidates of BreadthPIS and, therefore, this provokes the difference between the running times associated with their corresponding algorithms. Besides, in general terms, the ratio usually grows at first but it is reduced when we get lower support. This is due to the different candidate generation strategies.

5.6 Discussion. Comparing both algorithms

In this Section we discuss what are the main advantages of DepthPIS with regard to BreadthPIS. Firstly, we expose the main differences between the associated transition tables of both algorithms. Secondly we compare the method of search of both algorithms, breadth-first search for BreadthPIS and depth-first search for DepthPIS. Finally we show an important advantage that have the DepthPIS algorithm that is the possibility of resolving portions of search tree independently and in a parallel way.

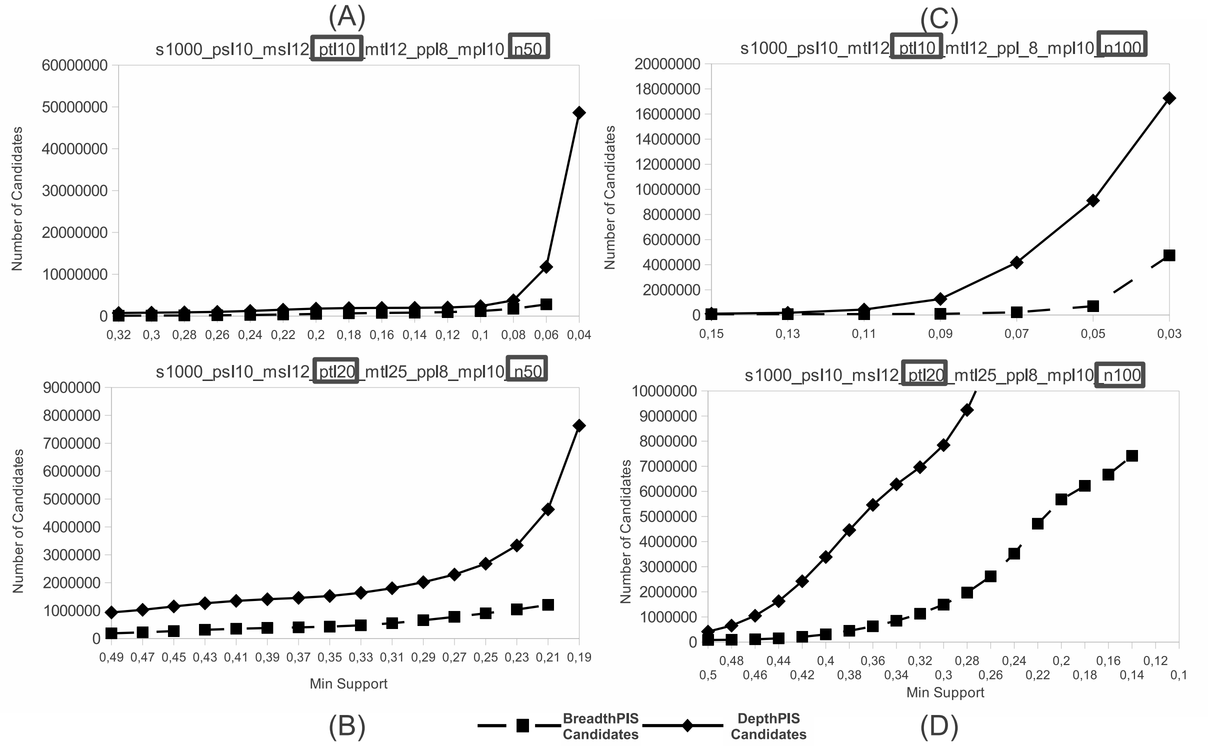


Figure 5.18: Candidate number for BreadthPIS and DepthPIS in plots of Figure 5.17.

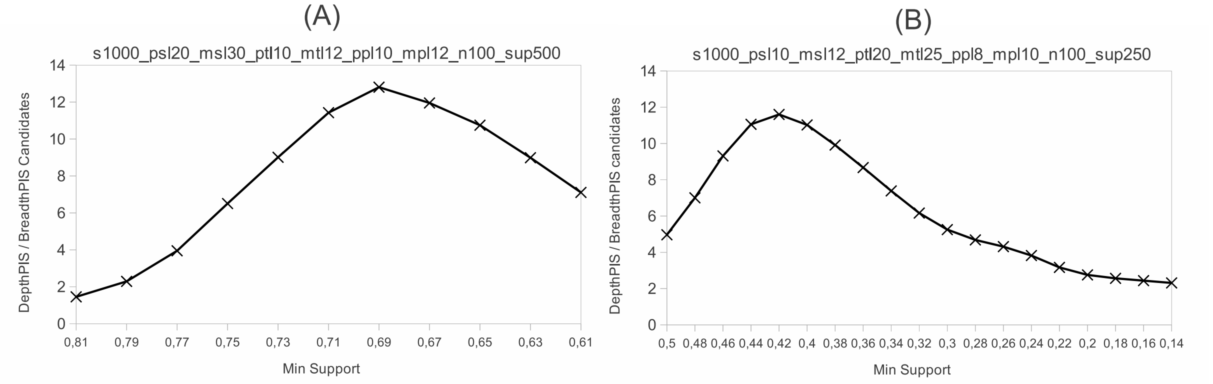


Figure 5.19: Ratio between the number of candidates generated by DepthPIS and the number of candidates generated by BreadthPIS for plots 5.16D and 5.17D.

In Sections 5.2 and 5.3 we saw the corresponding transition tables (Table 5.2 and Table 5.3, respectively), associated with the transition functions of BreadthPIS and DepthPIS, f_{BFS} and f_{DFS}. If both tables are viewed, it will be noted that, in general, the sets in $f_{\mathcal{DFS}}$ have a higher cardinality than those in $f_{\mathcal{BFS}}$.

This is owing to the nature of the candidate generation method. If we have three events A, B and C and we know R_{AB} and R_{BC}, BreadthPIS obtains the possible relations R_{AC} by means of $f_{\mathcal{BFS}}$. Conversely, in DepthPIS we know the relations R_{AB} and R_{AC}, and $f_{\mathcal{DFS}}$ obtains all the R_{BC} relations that can exist. This means of inferring candidates and how the sequences are arranged are the keys to understand the difference between their transition functions. We shall explain the functioning of both methods by means of Figures 5.20 and 5.21. Figure 5.20 shows the candidate generation for the BreadthPIS algorithm. It will be observed that, from the three events (A, B and C), the number of relations formed by candidate generation is bounded since the beginning of A occurs at the same time or before the beginning of C. On the contrary, Figure 5.21 shows that when DepthPIS algorithm infers the relation R_{BC}, any relation is possible between events B and C.

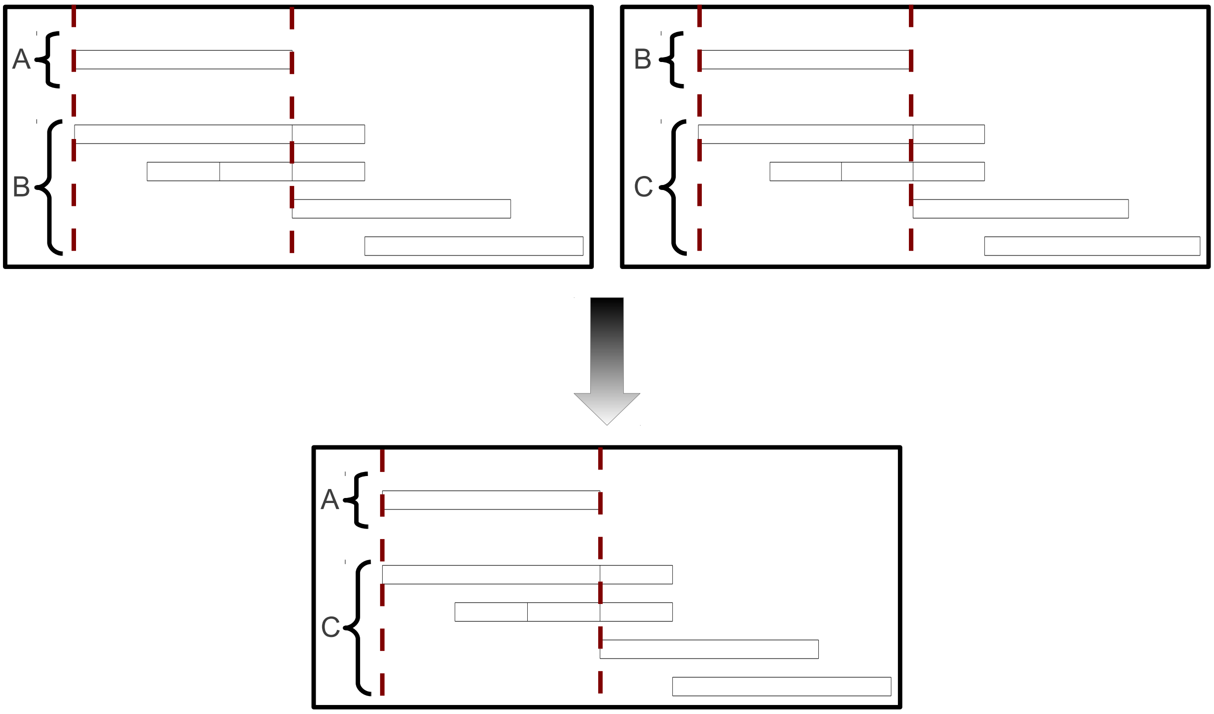


Figure 5.20: Study of candidate generation for BreadthPIS.

An analysis of all the set of relations resulting from both transition functions $f_{\mathcal{BFS}}$ and $f_{\mathcal{DFS}}$ leads to certain ideas. For example, if we add all the possible relations obtained for the 49 combinations of the transition function we obtain 75 relations for $f_{\mathcal{BFS}}$ and 143 for Table $f_{\mathcal{DFS}}$. These values provide an average cardinality of 1.53 relations for $f_{\mathcal{BFS}}$ and 2.91 for $f_{\mathcal{DFS}}$. These two numbers allow us to see that, on average, the number of candidates generated by DepthPIS is twice as large as the number of candidates produced by BreadthPIS. Moreover, if we study the average cardinality of the set of relations when we know one of the two relation arguments R_r or R_c of the transition functions $f_{\mathcal{BFS}}(R_r, R_c)$ and $f_{\mathcal{DFS}}(R_r, R_c)$, the result provides some interesting information. The top of Table 5.4 shows these average values when we know the first argument R_r, while the bottom part shows the average when we know the second argument R_c. On the one hand, in the first table it will be noted that except for the *equal* relation, $f_{\mathcal{DFS}}$ obtains bigger sets than $f_{\mathcal{BFS}}$, and there are remarkable differences between relations $<$, o and c, in which there is a factor from almost two up to more than four from $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$. On the other hand, the second table also shows significant differences between the number of relations provided by $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$ (except for the *equal* relation). In particular, the most significant differences appear in relations $<$, o, c and s, which have

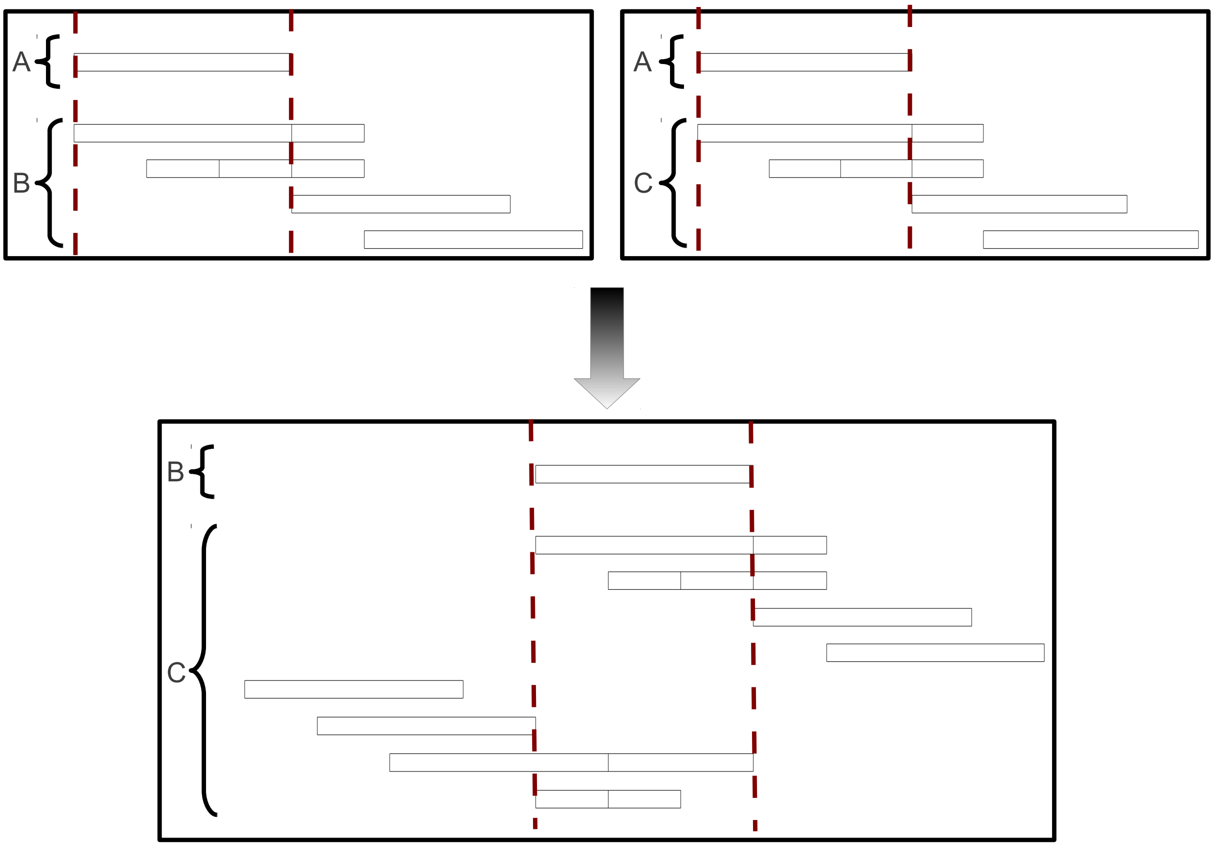


Figure 5.21: Study of candidate generation for DepthPIS.

a factor from almost two up to almost three relations.

BreadthPIS		DepthPIS	
$<$	1	$<$	4.43
m	1	m	2.43
o	2.14	o	4.14
f^{-1}	1	f^{-1}	2.14
c	2.43	c	3.86
$=$	1	$=$	1
s	2.14	s	2.43

	$<$	m	o	f^{-1}	c	$=$	s
BreadthPIS	1.57	1.28	1.86	1.57	2.14	1	1.28
	$<$	m	o	f^{-1}	c	$=$	s
DepthPIS	4.43	2.43	4.14	2.14	3.86	1	2.43

Table 5.4: Number of relations on average when we know the first or the second argument of the transition function.

Note that "m", "s", "f^{-1}" and " $=$ " are the least frequent relations that usually appear in patterns because the events that fulfil these relations need to have at least an equal relation between their boundary points. The main difference provided by $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$ therefore implies that the candidate generation of BreadthPIS is quite a lot more efficient than that provided by DepthPIS, since the minor differences in their averages are in the least frequent relations. What is more, the large sets of relations provided by $f_{\mathcal{DFS}}$ lead to the generation of several candidates in DepthPIS that will not eventually be frequent.

Another relevant difference between both algorithms is the search method used. As stated previously, BreadthPIS follows a breadth-first search strategy whereas DepthPIS carries out a depth-first search. The main problem caused by a breadth-first search is the need to maintain all the frequent patterns discovered in each level in the memory in order to be able to generate the set of candidate patterns. Although a breadth-first search normally makes the use of a pruning method possible, as was pointed in the methods of BreadthPIS in Section 5.2, with the tests that were executed it was not worth the effort of enabling such a prune. This is mainly because it is possible to quickly compute the support of a pattern. Instead, DepthPIS does not need to maintain all the k-patterns discovered in the memory, since it only needs the members of an equivalence class that are extended by only one event.

Finally, thanks to the way in which DepthPIS is executed and owing to its depth-first search, it is possible to split the search for frequent patterns up into different separate parts. To do this it is necessary to explore each equivalence class independently and maintain the set of frequent patterns associated with it. It is eventually only necessary to join all the intermediate sets of frequent patterns in the final frequent pattern set. This therefore makes it possible to parallelize the search of all the different classes in order to obtain faster results. Figure 5.22 shows a graphical schema of the separation of the different equivalence classes for the example database. In the figure, each equivalence class surrounds all the patterns that are contained in it.

5.7 Conclusions

In this Chapter we have introduced two algorithms that deal with these issues for mining interval patterns. Our main contributions of this Chapter are as follow:

- We simplify the processing of complex relations among intervals by getting all the information in a *triangular matrix of relations*. The main advantage of this representation is that the thirteen Allen's relations are perfectly summarized in a straightforward structure. We refer to this pattern representation as *TriMax*, and it expresses a pattern or sequence without any ambiguity, avoiding the common problems that occur in various existing representations. Thus, only by means of a structure we can express the relations among both the intervals and/or points that can appear in a pattern.

- We introduce a novel algorithm, called BreadthPIS, which stands for **Breadth**-first search algorithm for **P**oint and **I**nterval **S**equences, capable of discovering the whole set of frequent patterns containing relations between points, intervals or points and intervals. BreadthPIS is based on the Vertical Database Format Strategy and, by

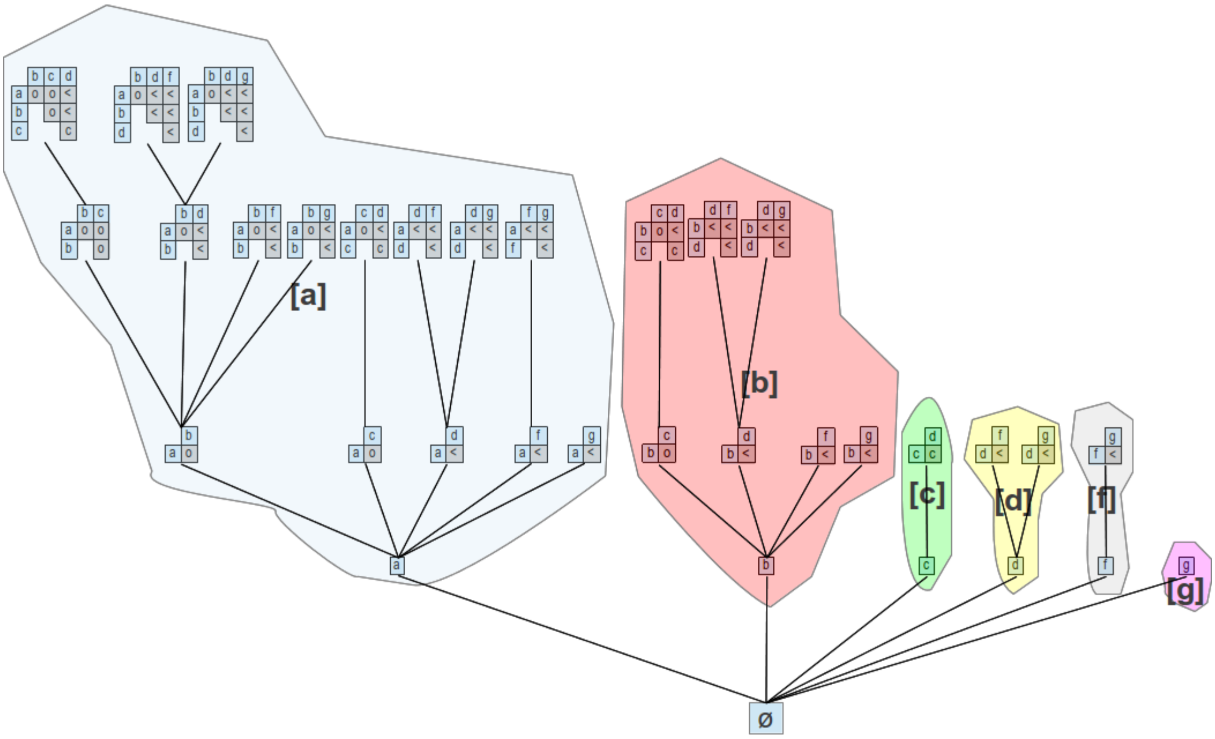


Figure 5.22: Division of the different equivalence class in independent problems.

means of temporal reasoning principles, it uses efficient methods to generate only the right candidates that can be derived from shorter patterns.

- We describe in detail the algorithm DepthPIS (**Depth**-first search algorithm for **P**oint and **I**nterval **S**equences), an implementation also based on Vertical Database Format strategy, that uses the same pattern representation and is capable of finding the same final pattern set as BreadthPIS. The main differences of this algorithm with respect to the previous one are: 1) the temporal reasoning used, i.e. the transition functions to infer new relations and generate candidates are different; and 2) the search method: while BreadthPIS executes a breadth-first search, DepthPIS uses a depth-first search.

- We do an exhaustive comparison between BreadthPIS and DepthPIS, analysing the advantages and drawbacks that come from the execution of their searches. We show that, in general, BreadthPIS is more efficient and scalable and outperforms DepthPIS in most of the performed test since the transition function of BreadthPIS give a more reduced candidate set and is the most optimized. However, DepthPIS reach lower support since does not suffer from the problems of memory overflow associated with BreadthPIS and its breadth-first search candidate generations and its use is more convenient when we face big databases.

Chapter 6

BreadthPIMS and DepthPIMS: Two New Fast Algorithms for Mining Points and Intervals Quantitative Patterns

In this Chapter we extend the algorithms and data structures exposed in Chapter 5 to develop two new algorithms BreadthPIMS and DepthPIMS, which are able to mine quantitative patterns composed of both points and intervals. All the definitions given in Section 5 are also valid for these new quantitative algorithm versions but we have to add some changes in order to take into account the temporal distances between items. Besides, as we did in Chapter 4, the time is assumed to have a discrete domain.

This Chapter is organized as follows. All the new definitions with respect to Chapter 5 are introduced in Section 5.1. Section 6.2 describes the new breath-first search Algorithm BreadthPIMS, whereas in Section 6.3 Algorithm DepthPIMS is described. In Section 6.4 we give all the changes that we have to include in both algorithms in order to mine both points and interval events while in Section 6.5 we propose some optimizations in order to improve the performance of the algorithms. An experimental and performance study is shown in Section 6.6, while as a wide discussion about the behavioural differences of the algorithms is given in Section 6.7. Finally, we provide our conclusions in Section 6.8.

6.1 Additional definition and description for problem setting

Unlike with the previous Chapter, where we were interested in the qualitative relations between items but not in their durations, when we deal with quantitative relations we need both the relation between items, and the items themselves, and we do keep the duration of events in the triangular matrix. Therefore, for quantitative patterns, all the TriMax have events with the associated duration with them. As an example, let $\alpha = \langle(t = 1, \langle A, 3\rangle)(t = 2, \langle B, 3\rangle)(t = 3, \langle C, 6\rangle)(t = 6, \langle D, 0\rangle)\rangle$ be a sequence, such as shown in the first sequence of the example database shown in Figure 2.1. The associated triangular matrix with this sequence is shown in Table 6.1.

	$B[3]$	$C[6]$	$D[0]$
$A[3]$	$o[-2]$	$o[-1]$	$<[2]$
$B[3]$		$o[-2]$	$<[1]$
$C[6]$			$c[-3]$

Table 6.1: Triangular matrix for example sequence.

In the same way, we have to redefine the concept of subsequence checking and, now, we have to consider both the durations of events and the temporal distances associated with relations in order to make a proper checking. For instance, the sequence $\alpha = \langle (t=1, \langle A,3\rangle)(t=2, \langle B,3\rangle)(t=3, \langle C,0\rangle)\rangle$ is a subsequence of $\beta = \langle (t=1, \langle A,3\rangle)(t=2, \langle B,3\rangle)(t=3, \langle C,0\rangle)(t=7, \langle D,5\rangle)\rangle$ but not of $\gamma = \langle (t=1, \langle A,3\rangle)(t=2, \langle B,4\rangle)(t=3, \langle C,0\rangle)\rangle$, since the event B has a duration of 3 time units in α whereas in β that duration is 4 time units.

Accordingly to the change of the subsequence checking operations, all the definitions such as prefix or equivalence class are changed to accomplish those operations. Furthermore, the transition function now refers to quantitative Allen's relations (see Section 2.4) instead of qualitative ones. Those transition function will be defined by BreadthPIMS and DepthPIMS in their respective sections (Section 6.2 and Section 6.3).

Finally, in Figure 6.1, we show the final frequent sequence set when we mine the example database with the quantitative version of TriMax, with a minimum support of 2. Nevertheless, we will talk about items in the patterns, regardless if they are points or intervals. So, different types of sequences are distinguished: those composed of both simple points and intervals, for instance the 3-sequence where the pattern have the event points d; those formed by only intervals, like the first 2-pattern that do not have the events d or g in them; and those with only points, composed of those sequences with only the event points d, g. In total, the final frequent sequence set has 12 frequent sequences.

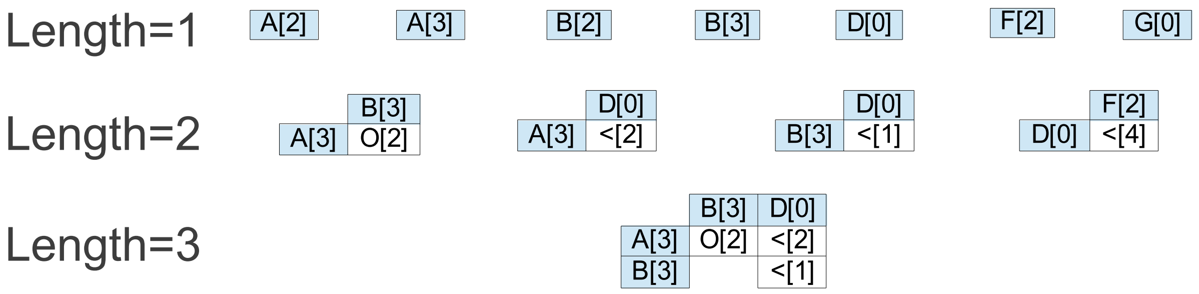
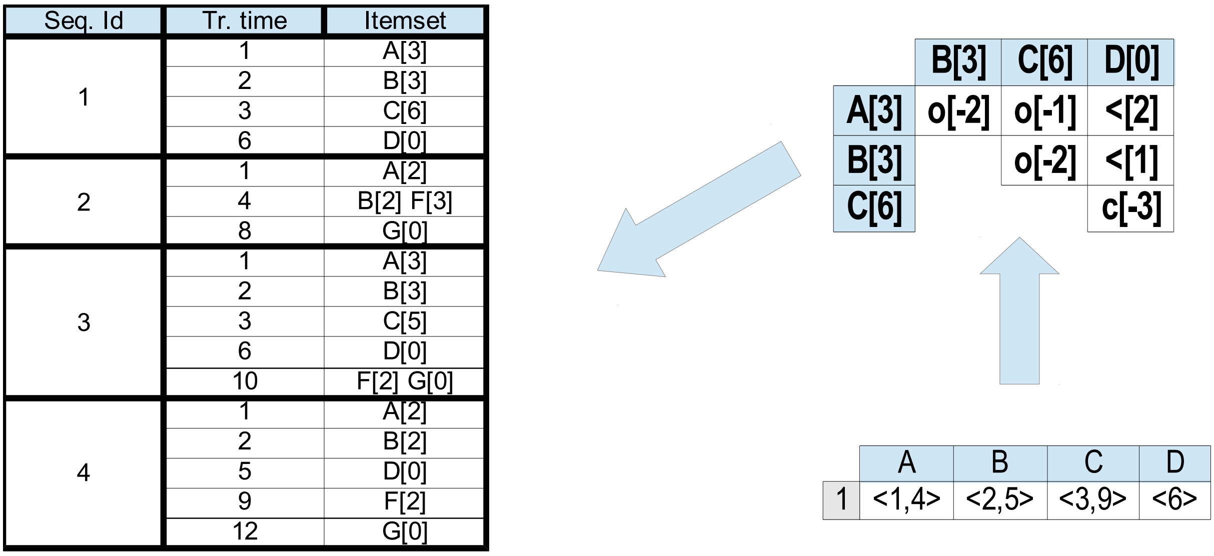


Figure 6.1: Frequent sequence set of example database.

6.2 BreadthPIMS: an algorithm for mining point and intervals temporal events based on breadth-first search

In this Section, we formulate and explain every step of our BreadthPIMS algorithm. Before starting with the description of main methods, we have to give some keynotes relatives to BreadthPIMS algorithm. Since we represent a pattern with a TriMax, and each pattern has a number of appearances in the database, stored in an IdList, whenever we refer to a pattern p, we refer to the couple $\langle TriMax, IdList\rangle$ (both of p). Besides, we will denote $p.TriMax$ and $p.IdList$ to refer to these parts of pattern p.

The above mentioned IdList is a data structure where we store, for each sequence, a list with all the appearances of a concrete pattern in that sequence. Each appearance is a list of events (point or intervals) and it also includes its timestamp for the beginning and its end when it is an interval. In Figure 6.2, we see the IdList of the example database for the pattern given in Table 6.1.



Seq. Id	Tr. time	Itemset
1	1	A[3]
	2	B[3]
	3	C[6]
	6	D[0]
2	1	A[2]
	4	B[2] F[3]
	8	G[0]
3	1	A[3]
	2	B[3]
	3	C[5]
	6	D[0]
	10	F[2] G[0]
4	1	A[2]
	2	B[2]
	5	D[0]
	9	F[2]
	12	G[0]

	B[3]	C[6]	D[0]
A[3]	o[-2]	o[-1]	<[2]
B[3]		o[-2]	<[1]
C[6]			c[-3]

	A	B	C	D
1	<1,4>	<2,5>	<3,9>	<6>

Figure 6.2: IdList for the pattern in the example database.

The BreadthPIMS algorithm, shown in Algorithm 23, consists in a main continuous loop that executes a breadth-first search. As first steps, all the frequent 1-patterns and 2-patterns have to be found (we try all the possible direct relations between frequent 1-patterns to obtain the frequent 2-patterns). The mentioned loop is composed of two steps: 1) All the possible (k+1)-superpatterns are generated from the current frequent (k)-patterns, and 2) we select only the frequent patterns among all the possible generated candidates. If this frequent $(k + 1)$-pattern set is not empty, we repeat the loop again in order to find longer frequent patterns. Finally, the algorithm ends returning the final frequent pattern set. The steps of candidate generation and counting of support both are shown in Algorithms 24 and 26 respectively.

Algorithm 24 shows the candidate generation algorithm. We base this method on that introduced in GSP algorithm [Srikant and Agrawal, 1996]. The main idea is merging two

Algorithm 23 BreadthPIMS($IsPruneActivated$)

```
$\mathcal{F}_1 = \{$frequent items$\}$
$\mathcal{F} = \mathcal{F}_1$
$\mathcal{F}_2 =$getFrequent2Sequences($\mathcal{F}_1$)
$\mathcal{F}_k = \mathcal{F}_2$
while $\mathcal{F}_k \neq \emptyset$ do
    $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$
    $C_k =$GenerateCandidates($\mathcal{F}_k$,$IsPruneActivated$)
    $\mathcal{F}_k =$CountingOfSupport($C_k$)
end while
return $\mathcal{F}$
Ensure: The frequent pattern set $\mathcal{F}$
```

k-patterns p_1 and p_2 where the $(k-1)$-suffix of p_1 is equal to the $(k-1)$-prefix of p_2. When we find two patterns that accomplish this condition, we can create a new $(k+1)$-pattern built by concatenating to p_1 the last event of p_2. In order to find the patterns that fulfil those conditions, Lines 4-9 create two hash table, one for prefixes and other for suffixes, where we link each pattern to all the patterns to which they are related. Then, in Lines 10-20 we iterate all the entries in the prefix table and, an entry is merged with a pattern in the suffix table if they are equal (Line 15). Finally, we return all the generated candidates in Line 21.

The method *MergePatterns*, shown in Algorithm 25, introduces the main changes added by this algorithm with respect to the original point-based ones. These changes are those derived from the use of intervals relations and also from the check of frequency. The method receives two patterns p_{post} and p_{pre} as parameters, and a flag value that indicates if a pruning method has to be executed as soon as we create candidates. The parameters p_{post} and p_{pre} are k-sequences with a common (k-1)-subsequence (suffix of p_{post} and prefix of p_{pre}) and, therefore, their associated TriMax share some cells as shown in Figure 6.3. We can see that the new candidate is formed by adding the last element and the last column of p_{pre} to p_{post}.

Once we have merged both TriMax, we still have to infer the relation between the first event of p_{post} and the last of p_{pre} (highlighted by an ellipsis in Figure 6.3). We have considered two options:

- to consider the Allen's thirteen relations in that cell and, thus, having thirteen different candidates whose frequency will have to be checked later.
- to infer the relations that connect the event E_1 and the event E_{k+1} by means of all the possible paths E_j, $\forall 1 < j < k+1$, which relate them through an intermediate event. That is, using the "path consistency" as defined by Allen [Allen, 1983]. Such inference is defined by a transition function (Definition 5.1.1 applied to quantitative relations) that returns a set of Allen's relations for a given pair of relations. That set contains the different candidates that can be derived from p_{post} and p_{pre}. In Table 6.4 we show the definition of the transition function $f_{BFS}(R_r, R_c)$ that we use in BreadthPIMS algorithm. In such table, the seven relations appearing in the rows

Algorithm 24 GenerateCandidates($\mathcal{F}_k$,$IsPruneActivated$)

```
Require: the (k-1) frequent pattern set $\mathcal{F}_k$, a boolean indicating if the pruning method
    has to be executed $IsPruneActivated$
$\mathcal{H}_{pre} = \emptyset$ {Hash table with duples $\langle pre, preList\rangle$, being $pre$ a pattern and $preSet$ set
    of patterns}
$\mathcal{H}_{post} = \emptyset$ {Hash table with duples $\langle post, postList\rangle$, being $post$ a pattern and $postSet$
    set of patterns}
$\mathcal{C}_k$
for all $p \in \mathcal{F}_k$ do
    $p_{pre} = (k-1)$ prefix of $p$
    $p_{post} = (k-1)$ suffix of $p$
    insert $p$ to the associated set of $\mathcal{H}_{pre}(p_{pre})$
    insert $p$ to the associated set of $\mathcal{H}_{post}(p_{post})$
end for
for all $\langle pre, preSet\rangle \in \mathcal{H}_{pre}$ do
    $\mathcal{S}$ =the associated set $postSet$ of $\mathcal{H}_{post}(pre)$
    if $\mathcal{S} \neq \emptyset$ then
        for all $p_{pre} \in preSet$ do
            for all $p_{post} \in \mathcal{S}$ do
                $\mathcal{MS}$ =MergePatterns($p_{post}$,$p_{pre}$,$IsPruneActivated$)
                $\mathcal{C}_k = \mathcal{C}_k \cup \mathcal{MS}$
            end for
        end for
    end if
end for
return $\mathcal{C}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```

correspond to the first argument R_r of f_{BFS}, whereas the same seven relations in the columns are those corresponding to the second argument R_c. Summarizing, f_{BFS} has 49 different combinations from its two arguments, being the smallest resulting sets composed of only one relation and the largest ones of five relations.

In this algorithm we take the second alternative given above and we use the transition function defined in Table 6.4 to generate the different candidates. The Algorithm 25, *MergePatterns*, obtains the inference associated with the TriMax of p_{post} and p_{pre}, set of relations $\mathcal{RS}$ (Line 1). Notice that, in order to infer the new relation we need to know what relation exists in the paths from E_1 to E_{k+1} through all intermediate events E_j. The relations R_{E_1,E_j} between the events E_1 and E_j and $R_{E_j,E_{k+1}}$ between E_j and E_{k+1} are the arguments R_r and R_c of the transition function f_{BFS}. In order to simplify our algorithm, we search for that E_j that generates the minimum number of different relations. In Figure 6.5 we show that method for an example. Since we have to find the possible relations that exist between the event $A[9]$ (E_1) and the event $E[6]$ (E_{k+1}), all the three events $B[4]$, $C[9]$ and $D[2]$ are a possible intermediate event (E_j). So, at first, we start with the $B[4]$ event, and as R_{AB} = "c$[-7]$" and R_{BE} = " $<$ [9]", we

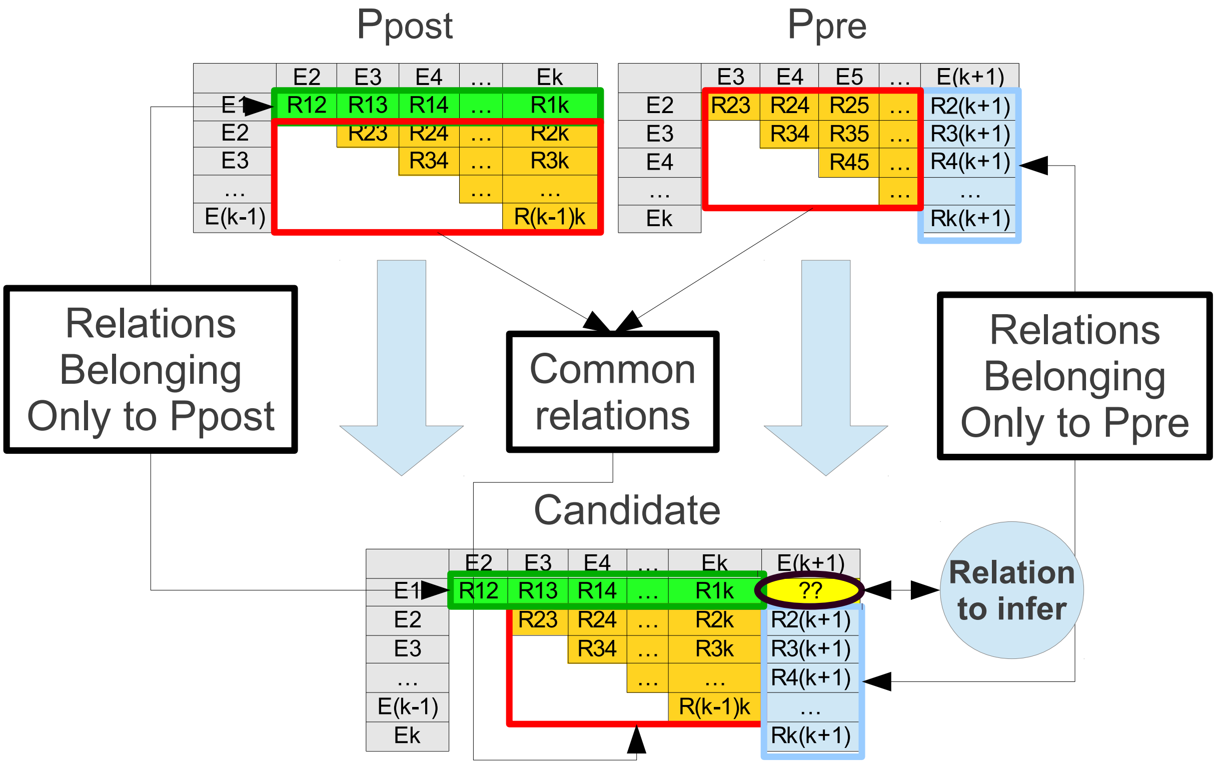
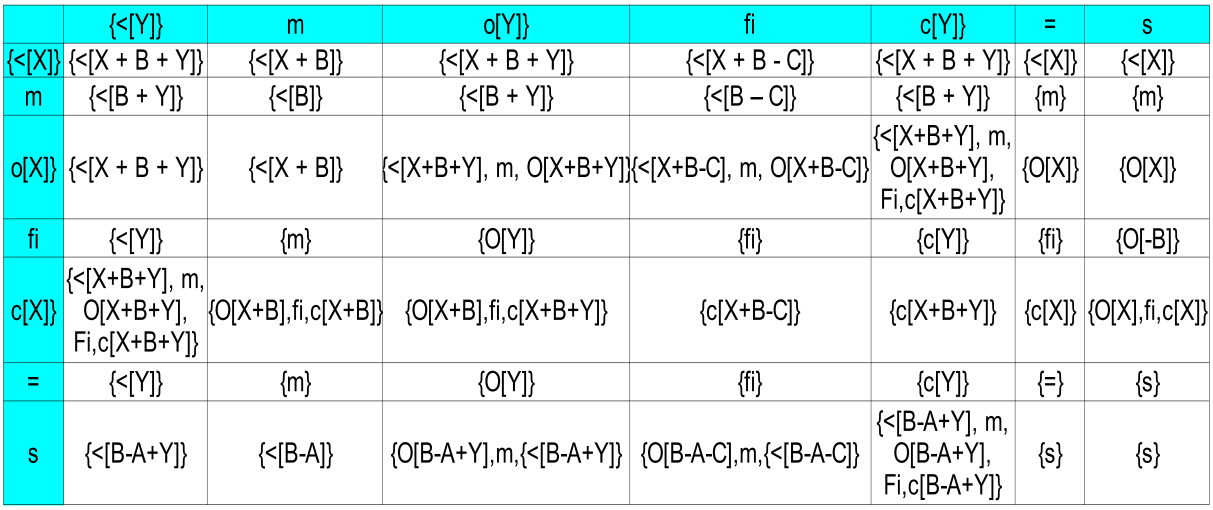


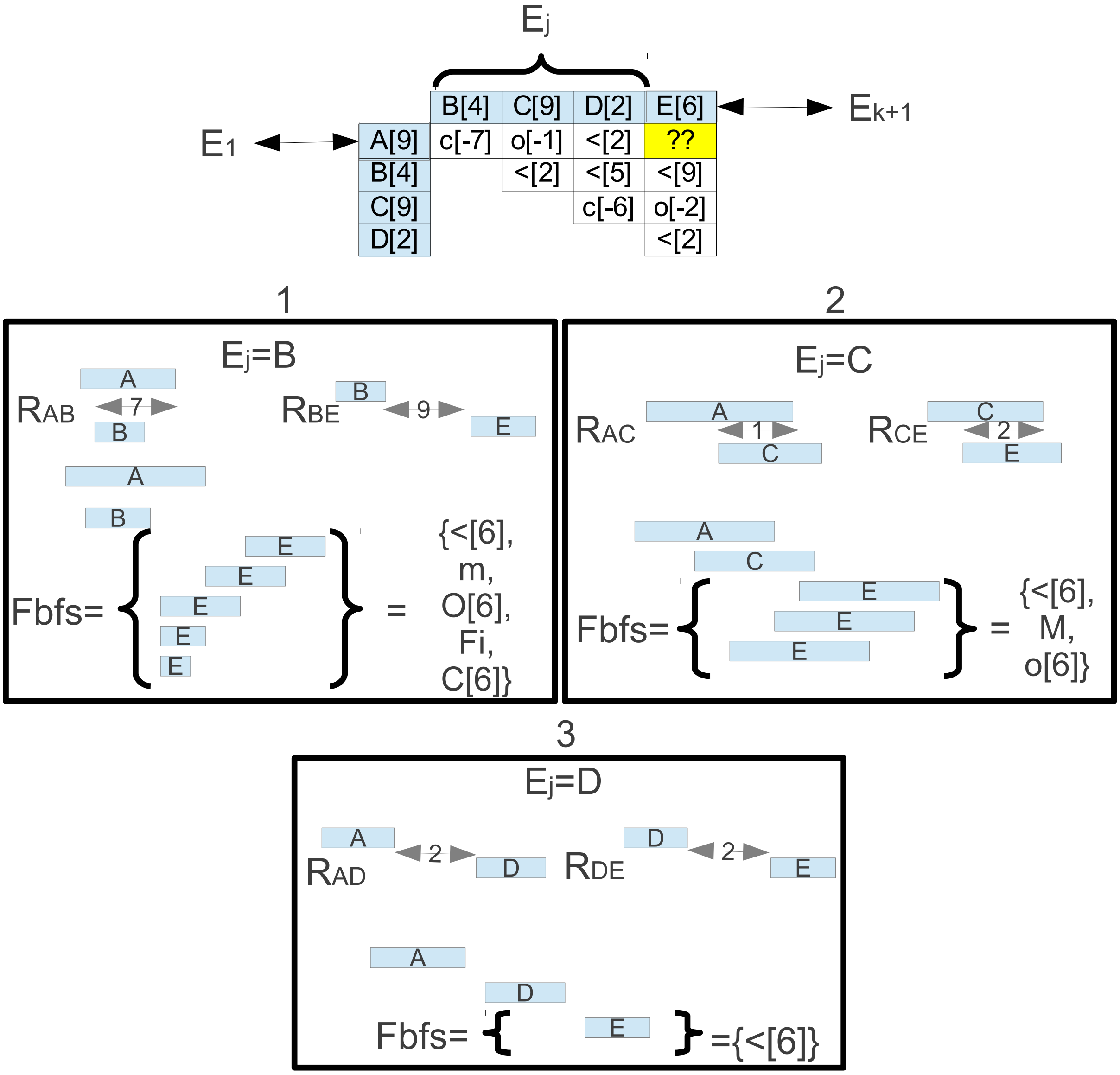
Figure 6.3: Generation of a candidate from two frequent patterns.



	{<[Y]}	m	o[Y]}	fi	c[Y]}	=	s
{<[X]}	{<[X + B + Y]}	{<[X + B]}	{<[X + B + Y]}	{<[X + B - C]}	{<[X + B + Y]}	{<[X]}	{<[X]}
m	{<[B + Y]}	{<[B]}	{<[B + Y]}	{<[B – C]}	{<[B + Y]}	{m}	{m}
o[X]}	{<[X + B + Y]}	{<[X + B]}	{<[X+B+Y], m, O[X+B+Y]}	{<[X+B-C], m, O[X+B-C]}	{<[X+B+Y], m, O[X+B+Y], Fi,c[X+B+Y]}	{O[X]}	{O[X]}
fi	{<[Y]}	{m}	{O[Y]}	{fi}	{c[Y]}	{fi}	{O[-B]}
c[X]}	{<[X+B+Y], m, O[X+B+Y], Fi,c[X+B+Y]}	{O[X+B],fi,c[X+B]}	{O[X+B],fi,c[X+B+Y]}	{c[X+B-C]}	{c[X+B+Y]}	{c[X]}	{O[X],fi,c[X]}
=	{<[Y]}	{m}	{O[Y]}	{fi}	{c[Y]}	{=}	{s}
s	{<[B-A+Y]}	{<[B-A]}	{O[B-A+Y],m,{<[B-A+Y]}	{O[B-A-C],m,{<[B-A-C]}	{<[B-A+Y], m, O[B-A+Y], Fi,c[B-A+Y]}	{s}	{s}

Figure 6.4: Transition table for all the different intervals relations.

have $f_{BFS}(R_{AB}, R_{BE}) = \{< [6], m, o[6], f^{-1}, c[6]\}$ as resulting set. Then, we continue exploring the paths to see if less candidates are generated. We see that using $C[9]$, we find $f_{BFS}(R_{AC}, R_{CE}) = \{< [6], m, o[6]\}$ and if we use $D[2]$, we find $f_{BFS}(R_{AD}, R_{DE}) = \{< [6]\}$. In this moment we can stop the search without analysing more paths since $f_{BFS}(R_{AD}, R_{DE})$ only contains a single relation, and therefore the candidate set cannot be reduced. In the same way, if we were lucky and in our first choice of E_j we had a result with a single relation, we would not need to continue with the process since we cannot find a set with less relations. Conversely, it can occur that we would not find any result with a single relation and, in that case, our result is the smallest set that we could find.



	B[4]	C[9]	D[2]	E[6]
A[9]	c[-7]	o[-1]	<[2]	??
B[4]		<[2]	<[5]	<[9]
C[9]			c[-6]	o[-2]
D[2]				<[2]

Figure 6.5: Process of choosing the smallest candidate relation set for BreadthPIMS.

We would like to go into detail about the call to the method CreateIdList (Line 10) and explain how a new IdList is created from the IdList of p_{post} and p_{pre}. To obtain such IdList, we check that all the appearances of the last event of p_{pre} are present in the IdList of the new candidate previously generated (from p_{post} and p_{pre}). Note that we have to check necessarily the relations of all the appearances because if we only check the inferred relation, some of the previous events could not be related with the new event with the relation that we expect. Figure 6.6 shows the steps for merging two IdList of two patterns.

The new IdList contains the IdList of C_1 reduced to those sequences that also contain one of the new relations inferred in Algorithm *MergePatterns* with respect the event E. In Figures 6.7 and 6.8 we show the calculation for the entries of the new IdList. In particular,

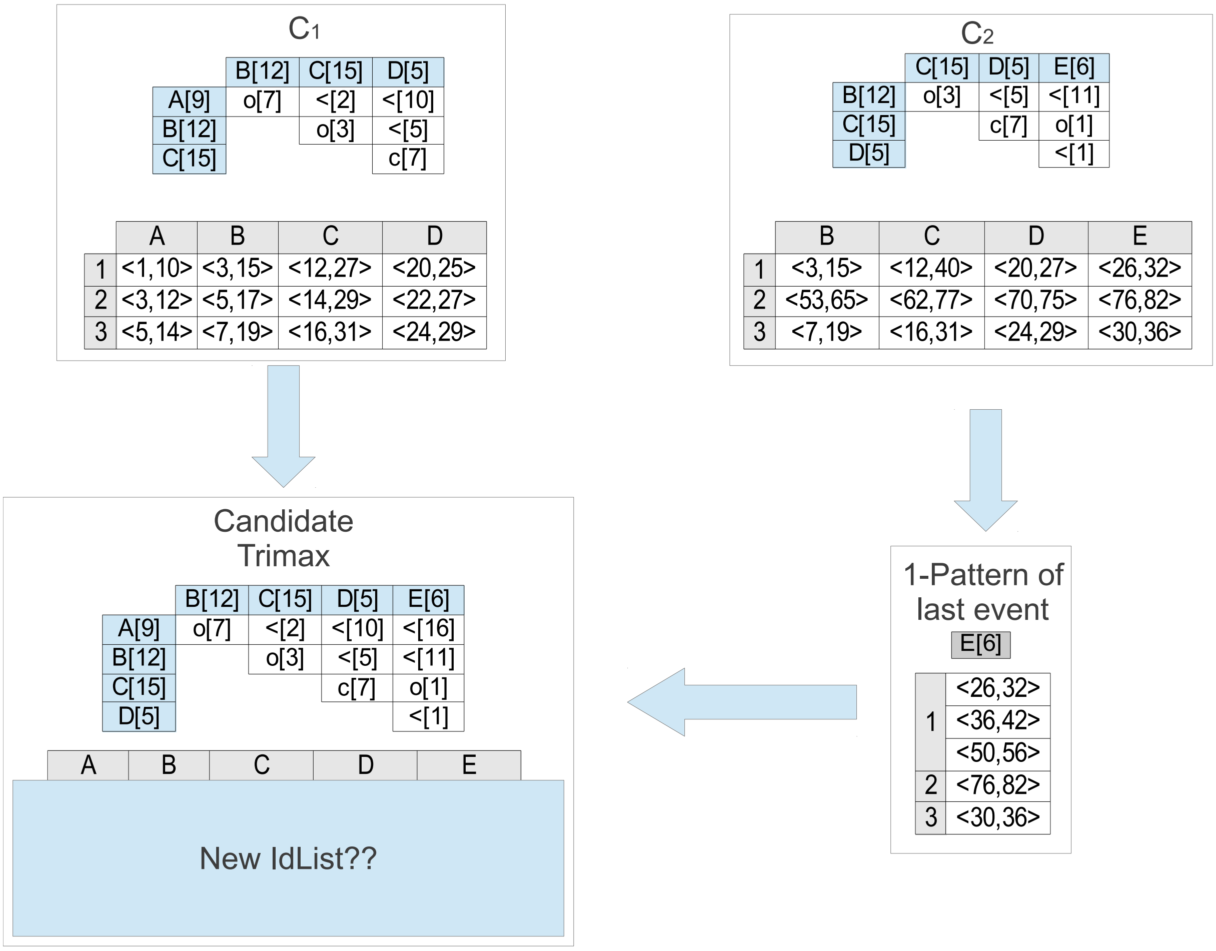
Algorithm 25 MergePatterns(p_{pre},p_{post},$IsPruneActivated$)

```
Require: The two pattern to merge $p_{post}$ and $p_{pre}$, a boolean indicating if the pruning method has to be executed $IsPruneActivated$
$\mathcal{RS}$ = TemporalReasoning($p_{post}.TriMax$,$p_{pre}.TriMax$)
for all $r \in \mathcal{RS}$ do
    $t$ = CreateTriMax($p_{post}.TriMax$,$p_{pre}.TriMax$,$r$)
    $pruned$ = false
    if $IsPruneActivated$ then
        $pruned$ = CheckPrune($t$)
    end if
    if not $pruned$ then
        $e = LastEventOf(p_{pre})$
        $NewIdList$ =CreateIdList($p_{post}.IdList$,$e.IdList$,$p_{post}.TriMax$,$t$)
        $c$ = pattern $\langle t, NewIdList \rangle$
        $\mathcal{C}_k = \mathcal{C}_k \cup \{c\}$
    end if
end for
return $\mathcal{C}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```

we expose two different cases: one where the new IdList entry is not empty and one where the IdList entry is empty. Figure 6.7 shows a case where there is a perfect matching between the first entry of C_1 IdList ($\langle < 1, 10 >, < 3, 15 >, < 12, 27 >, < 20, 25 > \rangle$) and the first entry of E IdList($\langle < 26, 32 > \rangle$), both also associated with the first sequence. As we said, the interval that represents $E[6]$ has to accomplish all the relations denoted by the last column of the new candidate TriMax. We do that in decreasing order, starting from the last relation of the column $R_{k,k+1}$ up to the first relation of the column $R_{1,k+1}$. In our example we start checking if the last but one event of the new TriMax $D[5]$ ($\langle < 20, 25 > \rangle$) has a " $<$ [1]" relation with $E[6]$ ($\langle < 26, 32 > \rangle$). Then, we continue doing the same process with the previous relation of the last column, which contains a "o[1]" relation with $C[15]$ ($\langle < 12, 27 > \rangle$) and with $E[6]$. We keep doing the same process with the rest of elements of the column having $B[12]$ ($\langle < 3, 15 > \rangle$) a relation of " $<$ [11]" with $E[6]$, and $A[9]$ ($\langle < 1, 10 > \rangle$) has a " $<$ [16]" relation with $E[6]$, and thus all the relations in the last column of the new TriMax are accomplished.

Conversely, Figure 6.8 shows the new empty IdList entry found when we try to merge the first C_1 entry and the second E_1 entry ($\langle < 36, 42 > \rangle$) of the first sequence. In this other example we start checking if the last but one event of the new TriMax $D[5]$($\langle < 20, 25 > \rangle$) has a " $<$ [1]" relation with $E[6]$, i.e. for the current entry, we check that the interval associated with $D[5]$ *before*, with a temporal distance of 1, to E ($\langle < 36, 42 > \rangle$) exists. Unfortunately, we check that the interval associated with $D[5]$ in the current entry $\langle < 20, 25 > \rangle$ has a " $<$ [11]" relation with E ($\langle < 36, 42 > \rangle$) instead of " $<$ [1]", leading us to abort the check of that new possible entry. After processing the rest of entries, we finally have the resulting IdList, shown in Figure 6.9, where besides the second entry of E for the first sequence a new entry appears for the third sequence.

Finally, Algorithm 26 selects only those generated candidates whose support is at least



	B[12]	C[15]	D[5]
A[9]	o[7]	<[2]	<[10]
B[12]		o[3]	<[5]
C[15]			c[7]

	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<20,25>
2	<3,12>	<5,17>	<14,29>	<22,27>
3	<5,14>	<7,19>	<16,31>	<24,29>

	C[15]	D[5]	E[6]
B[12]	o[3]	<[5]	<[11]
C[15]		c[7]	o[1]
D[5]			<[1]

	B	C	D	E
1	<3,15>	<12,40>	<20,27>	<26,32>
2	<53,65>	<62,77>	<70,75>	<76,82>
3	<7,19>	<16,31>	<24,29>	<30,36>

	B[12]	C[15]	D[5]	E[6]
A[9]	o[7]	<[2]	<[10]	<[16]
B[12]		o[3]	<[5]	<[11]
C[15]			c[7]	o[1]
D[5]				<[1]

A	B	C	D	E

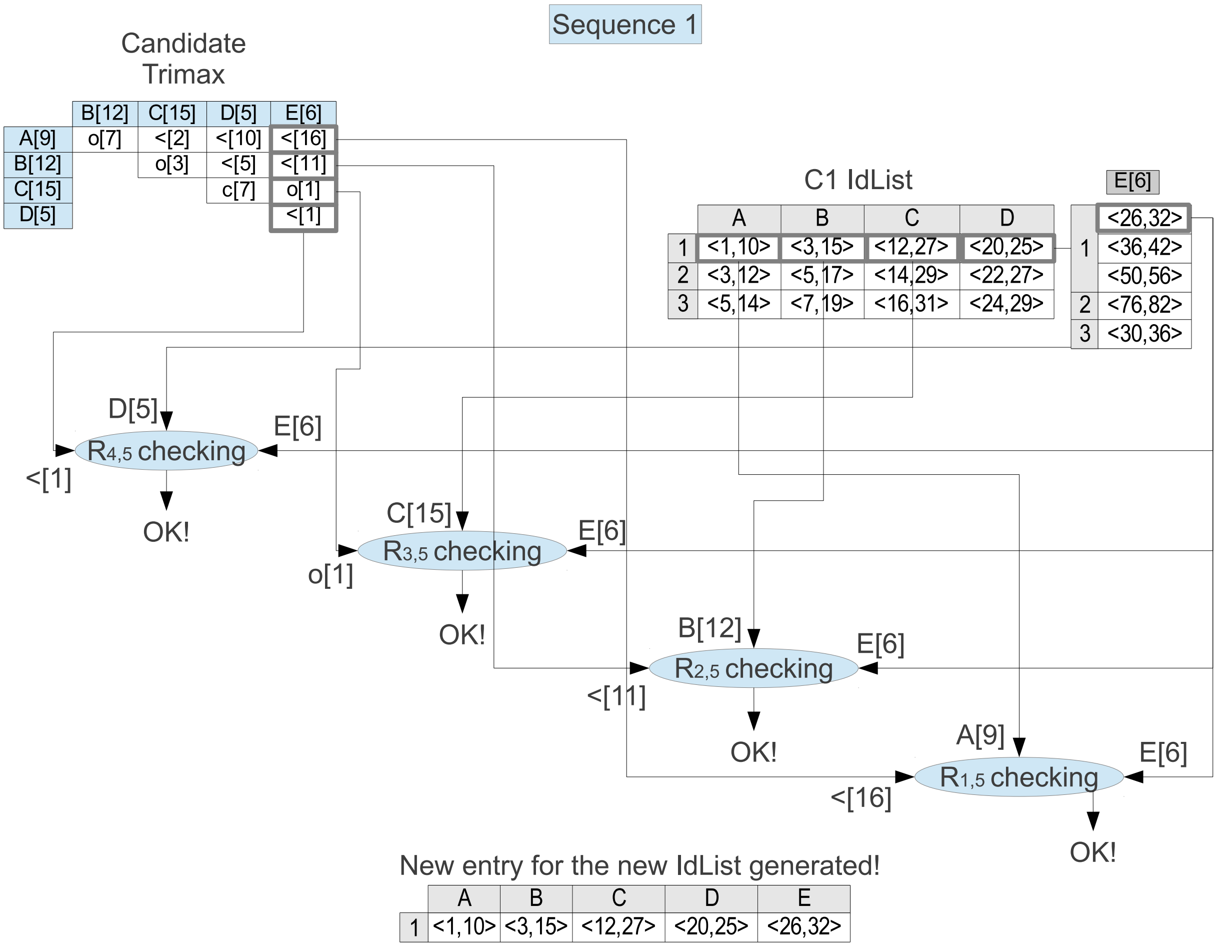
1	<26,32> <36,42> <50,56>
2	<76,82>
3	<30,36>

Figure 6.6: Example of merging of two IdLists ($p_{post} = C_1$ and $p_{pre} = C_2$).

min_sup. This method is very simple since the creation of the IdList have been already done through the method *MergePatterns*.

Algorithm 26 CountingOfSupport(C_k)

```
Require: the k-candidate set $C_k$ $\mathcal{F}_k = \emptyset$
for all $c_k \in C_k$ do
    if $c_k.support() > min\_sup$ then
        $\mathcal{F}_k = \mathcal{F}_k \cup c_k$
    end if
end for
return $\mathcal{F}_k$
Ensure: The frequent pattern set $\mathcal{F}$
```



	B[12]	C[15]	D[5]	E[6]
A[9]	o[7]	<[2]	<[10]	<[16]
B[12]		o[3]	<[5]	<[11]
C[15]			c[7]	o[1]
D[5]				<[1]

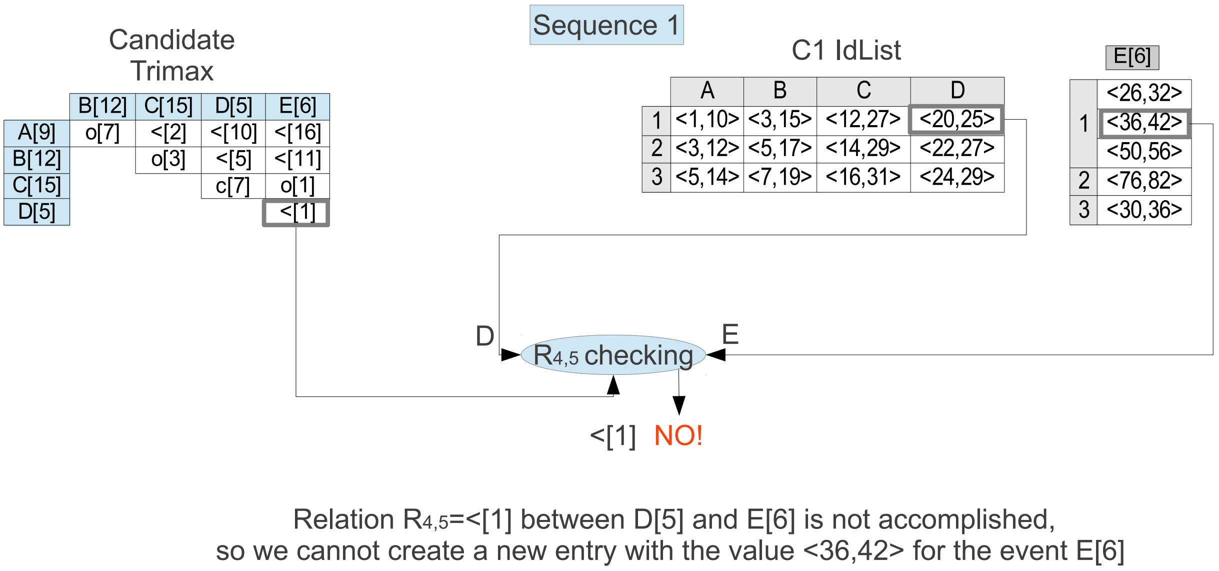
	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<20,25>
2	<3,12>	<5,17>	<14,29>	<22,27>
3	<5,14>	<7,19>	<16,31>	<24,29>

	E[6]
1	<26,32>
	<36,42>
	<50,56>
2	<76,82>
3	<30,36>

New entry for the new IdList generated!

	A	B	C	D	E
1	<1,10>	<3,15>	<12,27>	<20,25>	<26,32>

Figure 6.7: Example of a successful case in the calculation of an entry of the new IdList.



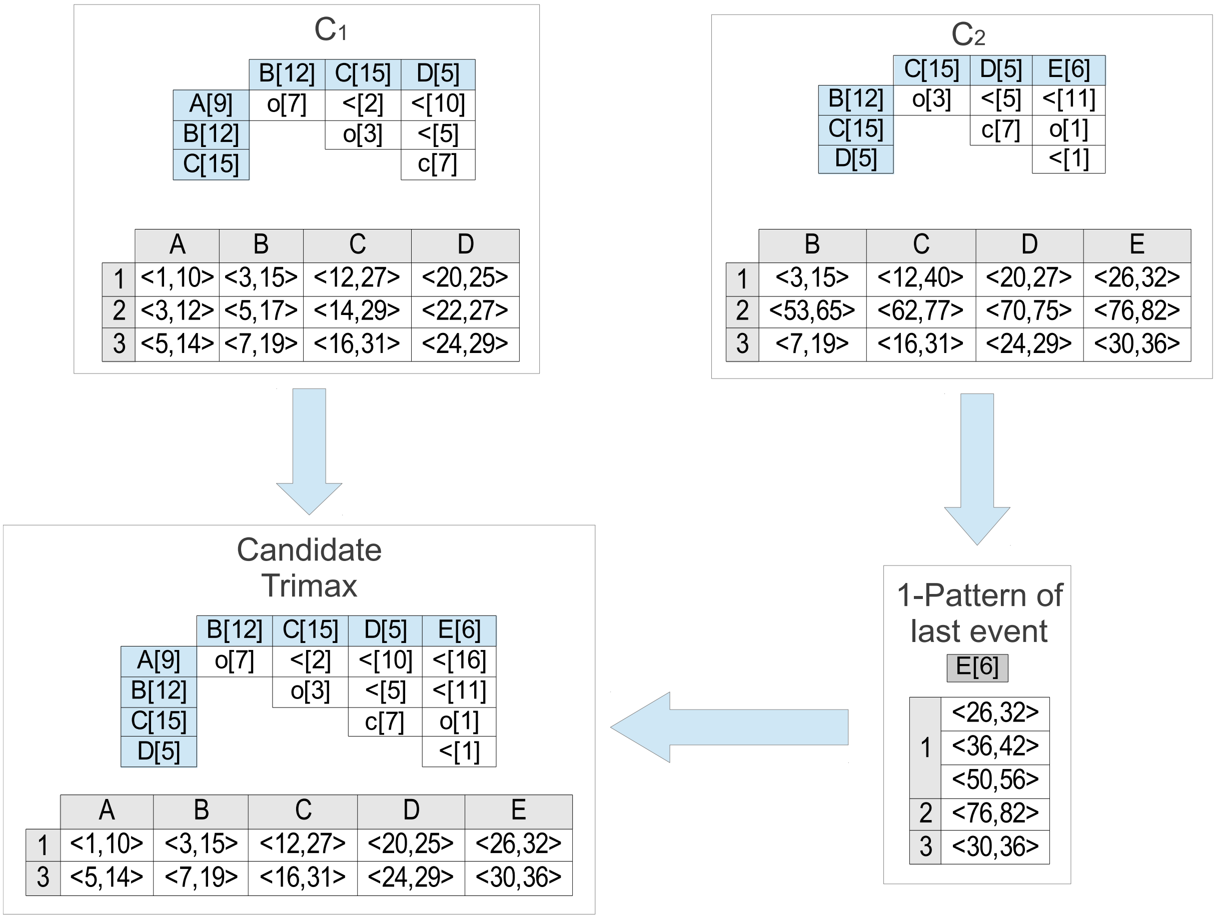
	B[12]	C[15]	D[5]	E[6]
A[9]	o[7]	<[2]	<[10]	<[16]
B[12]		o[3]	<[5]	<[11]
C[15]			c[7]	o[1]
D[5]				<[1]

	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<20,25>
2	<3,12>	<5,17>	<14,29>	<22,27>
3	<5,14>	<7,19>	<16,31>	<24,29>

	E[6]
	<26,32>
1	<36,42>
	<50,56>
2	<76,82>
3	<30,36>

Relation $R_{4,5}$=<[1] between D[5] and E[6] is not accomplished,
so we cannot create a new entry with the value <36,42> for the event E[6]

Figure 6.8: Example of a failed case in the calculation of an entry of the new IdList.



C_1

	B[12]	C[15]	D[5]
A[9]	o[7]	<[2]	<[10]
B[12]		o[3]	<[5]
C[15]			c[7]

	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<20,25>
2	<3,12>	<5,17>	<14,29>	<22,27>
3	<5,14>	<7,19>	<16,31>	<24,29>

C_2

	C[15]	D[5]	E[6]
B[12]	o[3]	<[5]	<[11]
C[15]		c[7]	o[1]
D[5]			<[1]

	B	C	D	E
1	<3,15>	<12,40>	<20,27>	<26,32>
2	<53,65>	<62,77>	<70,75>	<76,82>
3	<7,19>	<16,31>	<24,29>	<30,36>

1-Pattern of last event

E[6]

1	<26,32> <36,42> <50,56>
2	<76,82>
3	<30,36>

Candidate Trimax

	B[12]	C[15]	D[5]	E[6]
A[9]	o[7]	<[2]	<[10]	<[16]
B[12]		o[3]	<[5]	<[11]
C[15]			c[7]	o[1]
D[5]				<[1]

	A	B	C	D	E
1	<1,10>	<3,15>	<12,27>	<20,25>	<26,32>
3	<5,14>	<7,19>	<16,31>	<24,29>	<30,36>

Figure 6.9: Example of merging of two IdLists. Final result.

6.3 DepthPIMS: an algorithm for mining point and intervals temporal events based on depth-first search

Let us now describe our DepthPIMS algorithm. As in Algorithm BreadthPIMS (Section 6.3), the structure of the patterns uses a TriMax and an IdList. One of the key points of this algorithms is the use of the concept of equivalence class given at the end of Section 6.1.

DepthPIMS, shown in Algorithm 27, consists in a depth-first search that explores all the equivalence classes derived from all the frequent 1-patterns. In order to make the algorithms more efficient, frequent 1-patterns and 2-pattern have been previously found and all the frequent 2-patterns are inserted in the equivalence classes given by their 1-prefixes (Lines 9-11). Then, all the equivalence classes are explored and the algorithm ends returning the final set of frequent patterns $\mathcal{F}$.

Algorithm 28 shows the method *EnumerateFrequentSequences*, which is based on the method introduced in SPADE algorithm [Zaki, 2001]. The main idea consists of merging two k-equivalence classes $[Xy]$ and $[Xz]$, being both belonging to the k-class $[X]$, and having both the same common (k)-prefix X. In order to merge the patterns,

Algorithm 27 DepthPIMS()

```
$\mathcal{F}_1 = \{$frequent items$\}$
for all $X \in \mathcal{F}_1$ do
    add $X$ to equivalence class $[X]$
    $\mathcal{EC} = \mathcal{EC} \cup [X]$
end for
$\mathcal{F} = \mathcal{F}_1$
$\mathcal{F}_2$ =getFrequent2Sequences($\mathcal{F}_1$)
for all $Y \in \mathcal{F}_2$ do
    add $Y$ to equivalence class $[Y]$
    $p =$ 1-prefix of $Y$
    add $Y$ to equivalence class $[p]$
end for
$\mathcal{F} = \mathcal{F} \cup \mathcal{F}_2$
for all $[X] \in \mathcal{EC}$ do
    $\mathcal{F}_k$ =EnumerateFrequentSequences($[X]$)
    $\mathcal{F} = \mathcal{F} \cup \mathcal{F}_k$
end for
return $\mathcal{F}$
Ensure: The frequent pattern set $\mathcal{F}$
```

we go over each member of the k-equivalence class $[X]$ ($k+1$-superpatterns of X) and merge it with all the $k+1$-patterns that are after it in $[X]$ (Lines 2-4). Then, the method *GenerateCandidates* is called and we obtain a set of superpatterns candidates for both Xy and Xz patterns. In this point, for each candidate, we have to create its IdList (Line 7), and the new equivalence class $[c]$ (Line 8), as well as check its support (Line 9). When we find a frequent candidate, we can add it to the final frequent pattern set (that corresponds to the equivalence class $[X]$) and, depending on its k-prefix, we insert it into the equivalence class ($[Xy]$ or $[Xz]$). Finally, the Algorithm 28 ends returning the frequent pattern set of the equivalence class $[X]$.

The method *GenerateCandidates*, shown in Algorithm 29, introduces the main changes added by this algorithm with respect to the original point-based Vertical Database Format ones. As the Algorithm previously shown in Section 6.2 does, the method receives two equivalence classes $[Xy]$ and $[Xz]$ as parameters and infers the candidates. Let see how a new candidate is inferred from $[Xy]$ and $[Xz]$ (schema shown in Figure 6.10). Since both $[Xy]$ and $[Xz]$ belong to the equivalence class $[X]$, their patterns have a common $(k-1)$-prefix, and, therefore, their associated TriMax share several cells. Concretely, Xy and Xz share all the relations except their last row, both highlighted and joined by arrows in Figure 6.10. Thus, a new candidate is formed by adding the last column of the TriMax of Xz to the TriMax of Xy. In this point, as in Section 6.2, we see that we still need to infer the relation between the last event of Xy and Xz using a new transition function $f_{DFS}(R_r, R_c)$, defined by the transition table (6.12). This new transition function is different to that one used in BreadthPIMS algorithm, and it has been explicitly created for this DepthPIMS algorithm. The arguments of this function are the relation that connect an event E_j, $\forall 1 \leq j < k+1$, to the last but one event E_k and the relation that

Algorithm 28 EnumerateFrequentSequences($[X]$)

```
Require: the current equivalence class $[X]$ on which we search for the frequent patterns in a DFS way
$\mathcal{F}_k = \emptyset$
components =members of $[X]$ extended with only one more event than $X$
for all $[Xy] \in$ components do
    for all $[Xz] \in$ components that appear after $[Xy]$ do
        $C_k$ =GenerateCandidates($[Xy]$,$[Xz]$)
        for all $c \in C_k$ do
            newIdList = CreateIdList($c$,$[Xy]$,$[Xz]$)
            Create a new pattern $c$ with newIdList, and insert $c$ in the equivalence class $[c]$
            if $c.support() \geq min\_sup$ then
                $\mathcal{F}_k = \mathcal{F}_k \cup \{c\}$
                if the pattern $Xy$ is a prefix of $c$ then
                    add $[c]$ to $[Xy]$
                else
                    add $[c]$ to $[Xz]$
                end if
            end if
        end for
    end for
    if we have just found any frequent pattern then
        $\mathcal{F}_k = \mathcal{F}_k \cup$ EnumerateFrequentSequences($[Xy]$)
    end if
end for
return $\mathcal{F}_k$
Ensure: The frequent pattern set $\mathcal{F}_k$
```

connects the same event E_j to the last event E_{k+1}. In this case $E_k = y$, $E_{k+1} = z$, and E_j takes as values all the events in X. In that table the seven direct relations that we use as rows correspond to the argument R_r, whereas the columns correspond to the second argument R_c. As in BreadthPIMS algorithm, we also have 49 different resulting sets, with a minimum cardinality of one and a maximum of thirteen. In Figure 6.11 we can see the process in DepthPIMS for the TriMax given. In that example, a set of five relations is the minimum cardinality found.

Algorithm 29, GenerateCandidates, infers the TriMax of $[Xy]$ and $[Xz]$, obtaining a set of relations $\mathcal{RS}$ (Line 1). Then, for each relation of $\mathcal{RS}$, we repeat the same process of creating its new TriMax as we previously saw in Figure 6.3 (Lines 7 and 9). The algorithm ends returning all the candidates derived from Xy and Xz in Line 14. The aim of Lines 5-7 is exactly the same as Line 9. The motivation of this lines comes from the existence of inverse relations in some resulting relation sets from the transition table (Table 6.12).

For instance, the thirteen relations are possible when two *before* relations are given as parameters. In order to avoid the use of inverse relations ($>$, d, si, oi, mi and f) we have to swap the order of the events as shown in Figure 6.13.

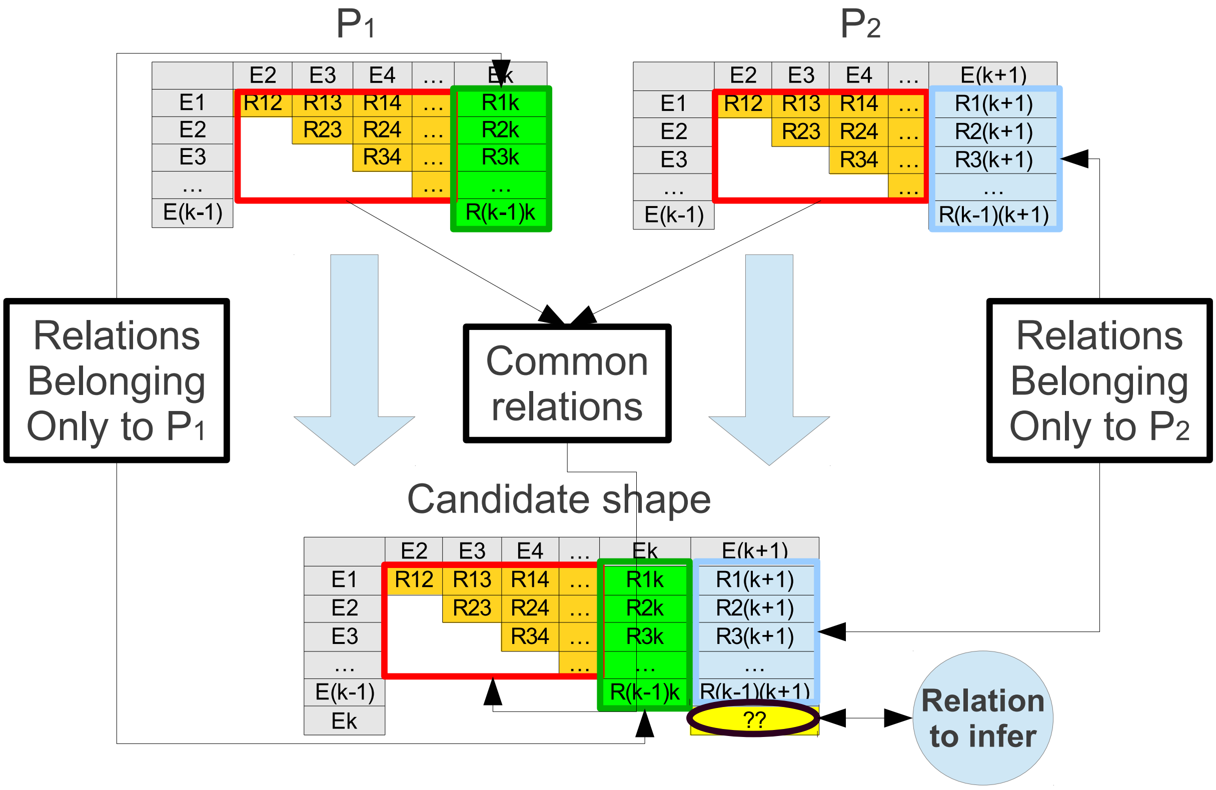


Figure 6.10: Example of merging of two TriMax.

```
Algorithm 29 GenerateCandidates([$Xy$],[$Xy$])
Require: The two equivalence classes that, from their patterns, we use for finding the
    different possible candidates.
$\mathcal{RS}$ = TemporalReasoning([$Xy$].TriMax,[$Xz$].TriMax)
$C_k = \emptyset$
if $\mathcal{RS} \neq \emptyset$ then
    for all $r \in \mathcal{RS}$ do
        if IsAnInverseRelation($r$) then
            $ri$ =getDirectRelation($r$)
            $c$ =createTriMaxWithInverseRelations($Xy$.TriMax,$Xz$.TriMax,$ri$)
        else
            $c$ =createTriMax($Xy$.TriMax,$Xz$.TriMax,$r$)
        end if
        $C_k = c$
    end for
end if
return $C_k$
Ensure: The candidate set $C_k$ derived from [$Xy$] and [$Xz$]
```

The algorithm CreateIdList is exactly the same as that used in BreadthPIMS algorithm (see Section 6.2). Figure 6.14 shows the final result of merging two IdList for the two

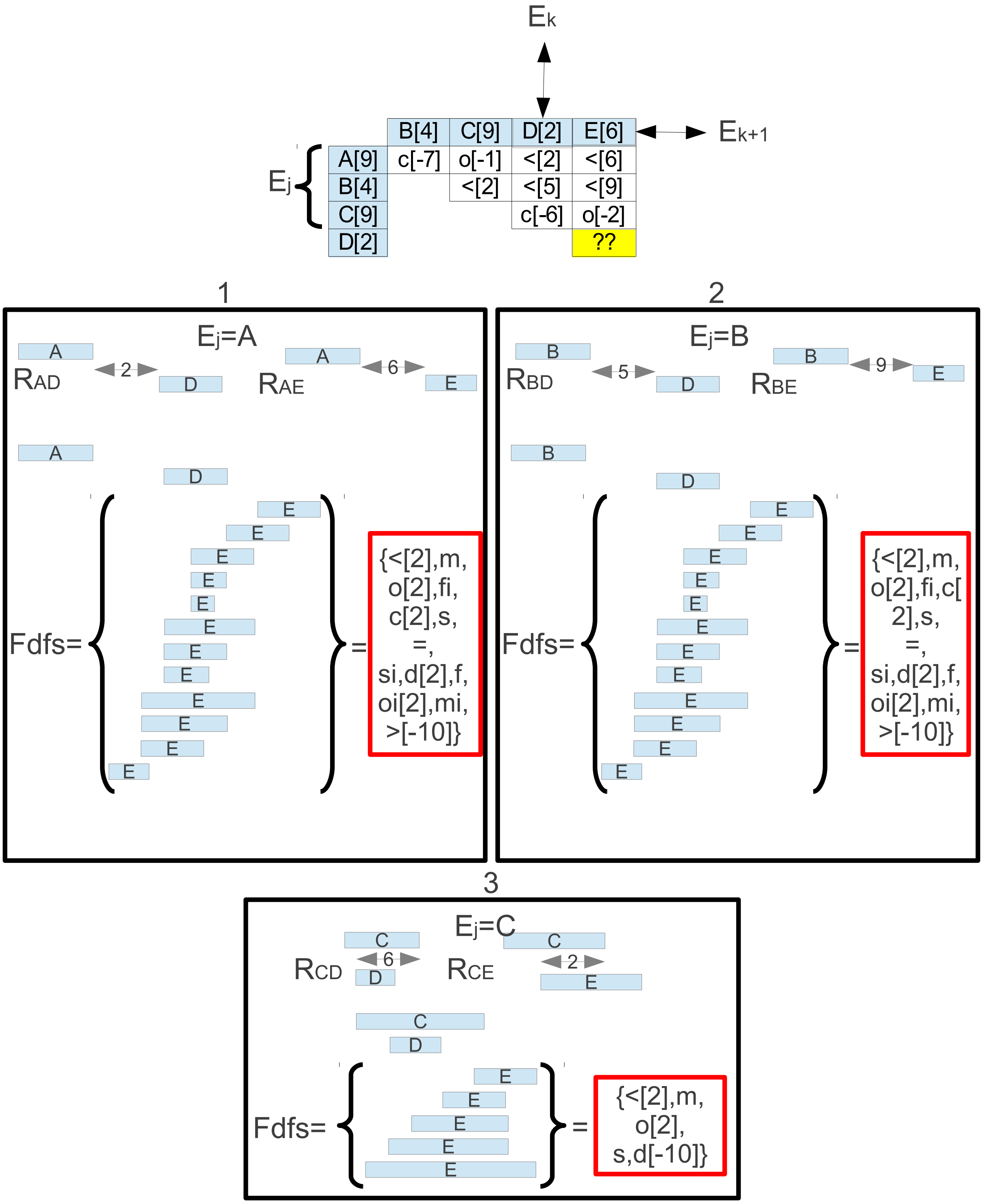
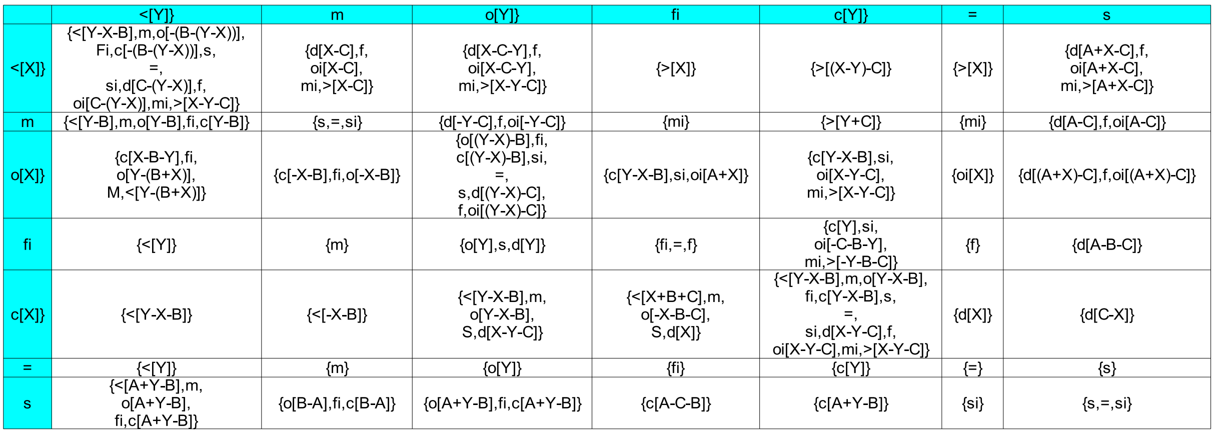


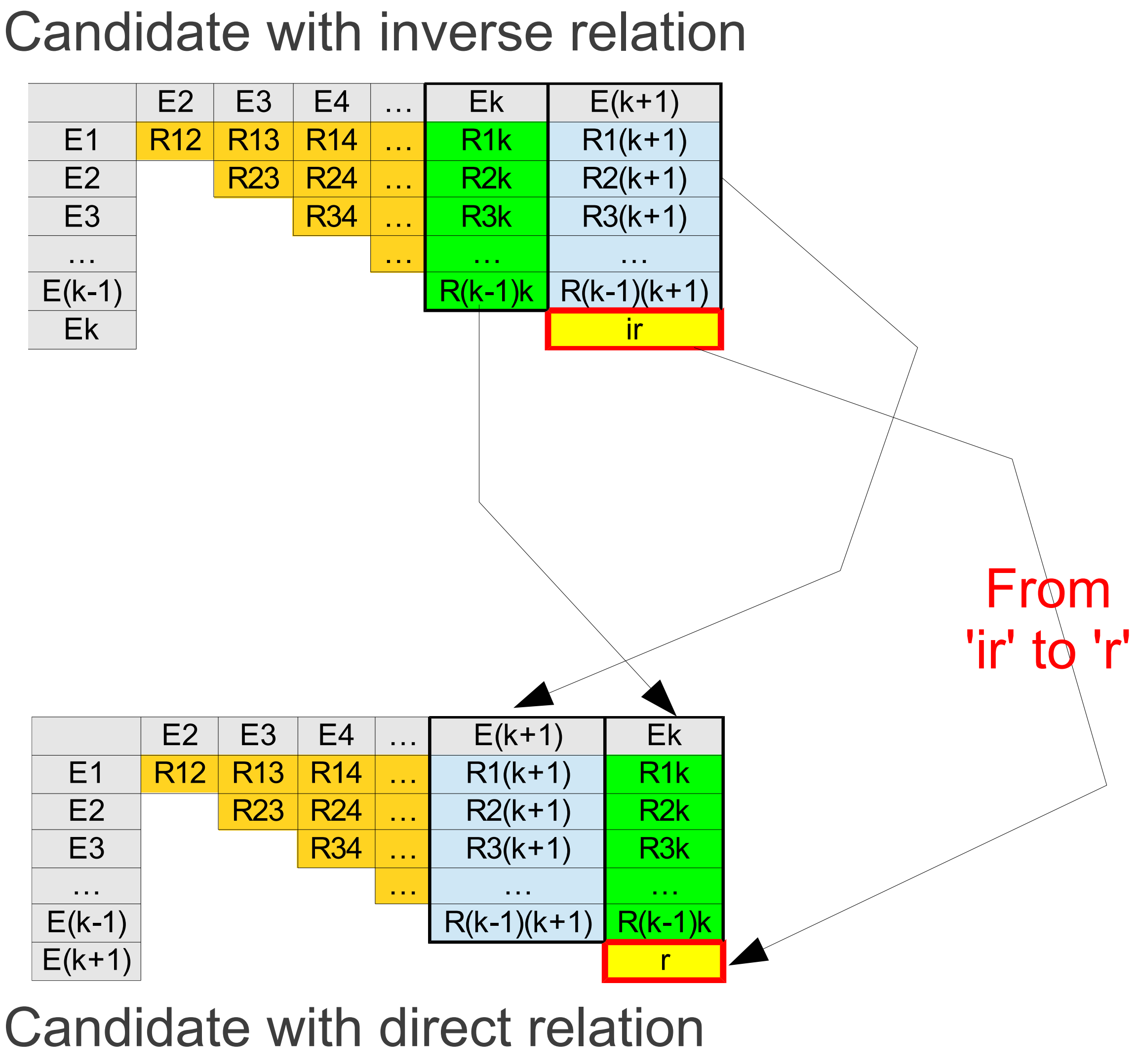
Figure 6.11: Process of choosing the smallest candidate relation set for DepthPIMS.

patterns given.



	<[Y]}	m	o[Y]}	fi	c[Y]}	=	s
<[X]}	{<[Y-X-B],m,o[-(B-(Y-X))], Fi,c[-(B-(Y-X))],s, =, si,d[C-(Y-X)],f, oi[C-(Y-X)],mi,>[X-Y-C]}	{d[X-C],f, oi[X-C], mi,>[X-C]}	{d[X-C-Y],f, oi[X-C-Y], mi,>[X-Y-C]}	{>[X]}	{>[(X-Y)-C]}	{>[X]}	{d[A+X-C],f, oi[A+X-C], mi,>[A+X-C]}
m	{<[Y-B],m,o[Y-B],fi,c[Y-B]}	{s,=,si}	{d[-Y-C],f,oi[-Y-C]}	{mi}	{>[Y+C]}	{mi}	{d[A-C],f,oi[A-C]}
o[X]}	{c[X-B-Y],fi, o[Y-(B+X)], M,<[Y-(B+X)]}	{c[-X-B],fi,o[-X-B]}	{o[(Y-X)-B],fi, c[(Y-X)-B],si, =, s,d[(Y-X)-C], f,oi[(Y-X)-C]}	{c[Y-X-B],si,oi[A+X]}	{c[Y-X-B],si, oi[X-Y-C], mi,>[X-Y-C]}	{oi[X]}	{d[(A+X)-C],f,oi[(A+X)-C]}
fi	{<[Y]}	{m}	{o[Y],s,d[Y]}	{fi,=,f}	{c[Y],si, oi[-C-B-Y], mi,>[-Y-B-C]}	{f}	{d[A-B-C]}
c[X]}	{<[Y-X-B]}	{<[-X-B]}	{<[Y-X-B],m, o[Y-X-B], S,d[X-Y-C]}	{<[X+B+C],m, o[-X-B-C], S,d[X]}	{<[Y-X-B],m,o[Y-X-B], fi,c[Y-X-B],s, =, si,d[X-Y-C],f, oi[X-Y-C],mi,>[X-Y-C]}	{d[X]}	{d[C-X]}
=	{<[Y]}	{m}	{o[Y]}	{fi}	{c[Y]}	{=}	{s}
s	{<[A+Y-B],m, o[A+Y-B], fi,c[A+Y-B]}	{o[B-A],fi,c[B-A]}	{o[A+Y-B],fi,c[A+Y-B]}	{c[A-C-B]}	{c[A+Y-B]}	{si}	{s,=,si}

Figure 6.12: Transition table for all the different intervals relations.



Candidate with inverse relation

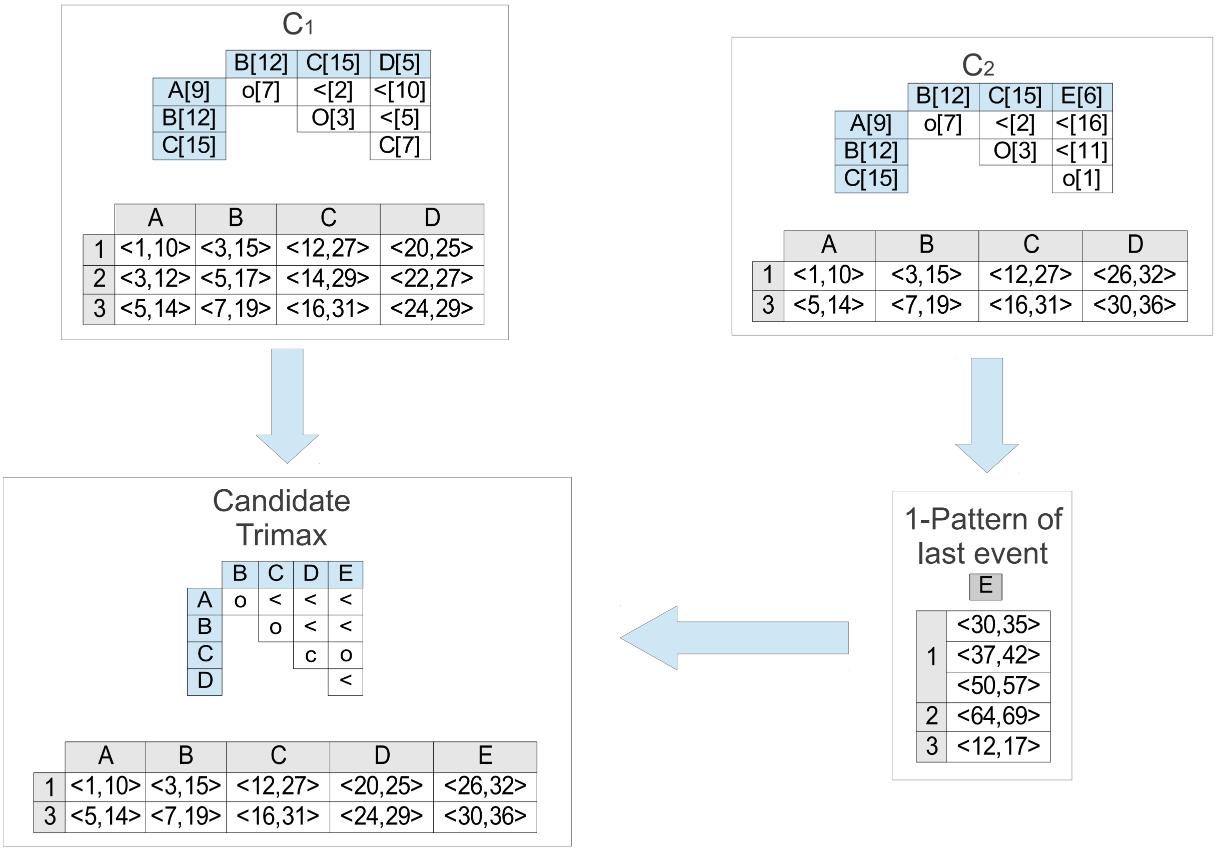
	E2	E3	E4	…	Ek	E(k+1)
E1	R12	R13	R14	…	R1k	R1(k+1)
E2		R23	R24	…	R2k	R2(k+1)
E3			R34	…	R3k	R3(k+1)
…				…	…	…
E(k-1)					R(k-1)k	R(k-1)(k+1)
Ek						ir

From 'ir' to 'r'

	E2	E3	E4	…	E(k+1)	Ek
E1	R12	R13	R14	…	R1(k+1)	R1k
E2		R23	R24	…	R2(k+1)	R2k
E3			R34	…	R3(k+1)	R3k
…				…	…	…
E(k-1)					R(k-1)(k+1)	R(k-1)k
E(k+1)						r

Candidate with direct relation

Figure 6.13: Conversion of a TriMax with an inverse relations.



C1

	B[12]	C[15]	D[5]
A[9]	o[7]	<[2]	<[10]
B[12]		O[3]	<[5]
C[15]			C[7]

	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<20,25>
2	<3,12>	<5,17>	<14,29>	<22,27>
3	<5,14>	<7,19>	<16,31>	<24,29>

C2

	B[12]	C[15]	E[6]
A[9]	o[7]	<[2]	<[16]
B[12]		O[3]	<[11]
C[15]			o[1]

	A	B	C	D
1	<1,10>	<3,15>	<12,27>	<26,32>
3	<5,14>	<7,19>	<16,31>	<30,36>

1-Pattern of last event

E

1	<30,35> <37,42> <50,57>
2	<64,69>
3	<12,17>

Candidate Trimax

	B	C	D	E
A	o	<	<	<
B		o	<	<
C			c	o
D				<

	A	B	C	D	E
1	<1,10>	<3,15>	<12,27>	<20,25>	<26,32>
3	<5,14>	<7,19>	<16,31>	<24,29>	<30,36>

Figure 6.14: Example of merging of two IdLists.

6.4 Mining points and intervals

For these two algorithms we do exactly the same that we did in Section 5.4. However, in this case, the eight transition tables include both relations and the temporal distances associated with them, such as we showed in the transition tables exposed in Section 6.2 and 6.3.

6.5 Optimizations

As we did in Section 4.4, two optimizations are made in order to improve the execution of both BreadthPIMS and DepthPIMS algorithms. Both optimizations are the same: 1) a simplification of the original database from which we obtain the frequent quantitative patterns; and 2) the generation of frequent 2-patterns by a Pattern Growth method. We refer to that section for a better understanding of both optimizations bearing in mind that now we deal with a different representation. This representation eases the first optimization since the durations are directly associated with items and with boundary points the durations were considered like temporal distances between points.

6.6 Experimental results

As in previous chapters, we use the setting of the experiments is the same to the explained in Section 3.4. Besides, the considered versions for the algorithms are those that mines the set of frequent 2-patterns through a Pattern Growth method since is more efficient than a brute force search (see Section 6.5 for more details). Furthermore, all the items considered in all the databases can extend from a duration of 1 to 10 units of time.

In all the comparative studies made we find that both algorithms have the same behaviour: executions where both algorithms find a similar number of candidates and, thus, a similar time execution where, in some occasions, BreadthPIMS cannot complete its execution due to its Breadth-first search.

Figure 6.15 shows plots where the execution times for BreadthPIMS and DepthPIMS are almost equal for medium datasets (1000 sequences) with medium sequence length (20 transactions on average), medium-high transaction length (20 items on average), medium pattern length (8 items on average), and 100 and 500 different items for the database. In both plots we can see that BreadthPIMS cannot finish its execution for the lowest supports (0.02 in plot 6.15A and 0.08 in plot 6.15B).

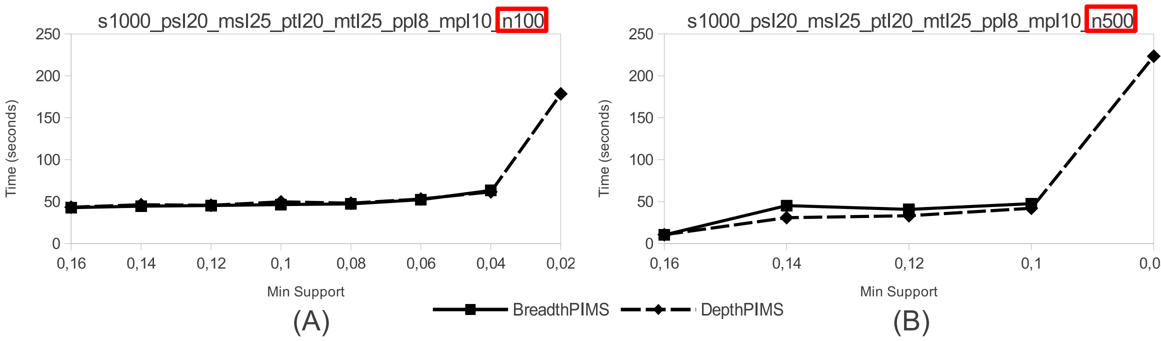


Figure 6.15: Varying support for datasets $s1000\_psl20\_msl25\_ptl20\_mtl25\_ppl8\_mpl10\_n100$–500.

Figure 6.16 shows two plots for databases with bigger sequence length (40 transactions on average) keeping all the rest of properties as in Figure 6.15. Now, we see that the behaviour of both plots are very similar to those appearing in Figure 6.15. However, the time which Figure 6.16 achieve for the same supports, is four times bigger than that showed in Figure 6.15 because its number of itemsets per sequence is increased (psl= 20 in Figure 6.15 and psl= 40 in Figure 6.16). As before, for the lowest supports BreadthPIMS cannot complete its execution due to its Breadth-first search method for generating candidates.

In Figure 6.17 we see two plots for databases with a less transaction length (10 items on average) than those appearing in 6.16, and the patterns that are set in the databases are longer than before (15 items on average). We study two cases, one with a sparser database ($n = 1000$) in plot 6.17A, and a denser database ($n = 500$) in plot 6.17B. As in the previous cases, both curves are very similar, however, we see a little domain of BreadthPIMS over DepthPIMS for the lowest domains. Furthermore, since the databases are not very big (1000 sequences) and the transaction length are moderated (10 items on average), the execution time reach time values not very high (161 and 101 seconds

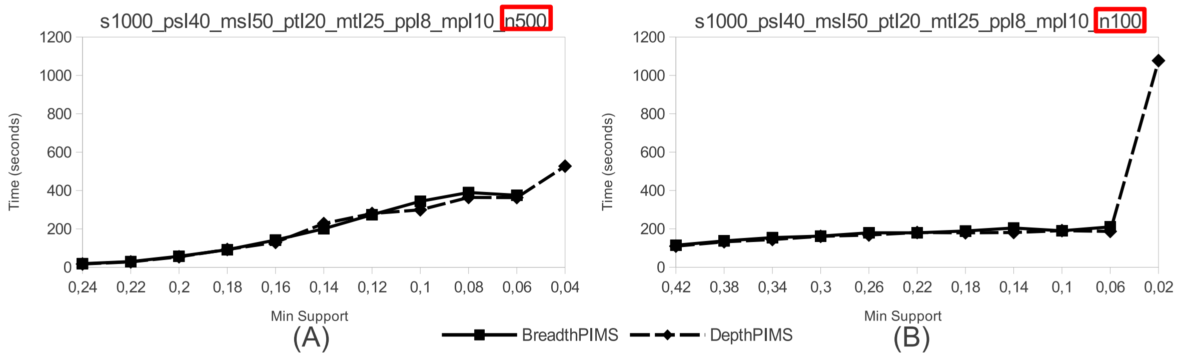


Figure 6.16: Varying support for datasets $s1000\_psl40\_msl50\_ptl20\_mtl25\_ppl8\_mpl10\_n100$–500.

respectively).

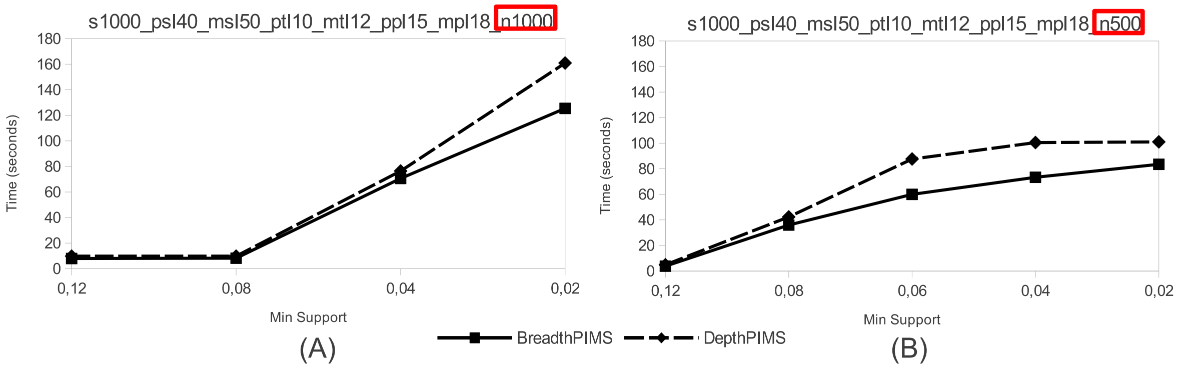


Figure 6.17: Varying support for datasets $s1000\_psl40\_msl50\_ptl10\_mtl12\_ppl15\_mpl18\_n500$–1000.

Finally, 6.18 shows four plots for bigger databases, with 10000 sequences each one. The plots show different sequence lengths (20 transactions on average for plots 6.18A and 6.18B and 40 transactions on average for plots 6.18C and 6.18D), different transaction length (20 items on average for plots 6.18A and 6.18B and 10 items on average for plots 6.18C and 6.18D) and different number of different items (1000 items in plots 6.18A and 6.18C and 500 items in plots 6.18B and 6.18D). All the plots show almost equivalent behaviours for both BreadthPIMS and DepthPIMS and, for plots 6.18C and 6.18D, BreadthPIMS cannot finish due to its BFS candidate generation.

In general, we can see very similar behaviour for both BreadthPIMS and DepthPIMS algorithms and all the executions get lower supports that for the qualitative algorithms and, in some points, they suddenly increase. This is because of the pattern explosion phenomenon that is explained in the next section.

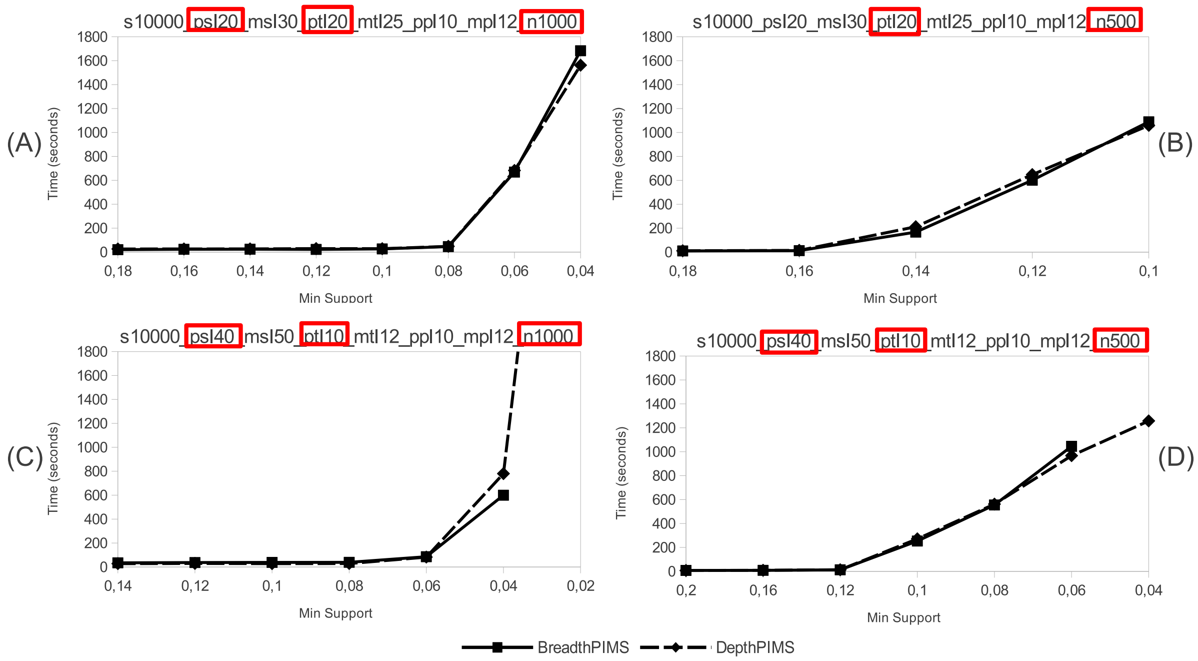


Figure 6.18: Varying support for datasets $s10000\_psl40\_msl50\_ptl10$–20_mtl12–25_ppl10 $\_mpl12\_n500$–1000.

6.7 Discussion. Comparing both algorithms

All the issues addressed in Section 5.6 are also valid for this discussion but now applied in quantitative algorithms.

Firstly, the generation of candidates of BreadthPIMS is better than that used in DepthPIMS since its transition table produces less candidates. However, in the performed tests we do not observe a significant difference between both algorithm executions when we mine databases with different databases properties. Nevertheless, what is important is that BreadthPIMS cannot complete its execution for low supports due to its breadth-first search method, whereas DepthPIMS can successfully finish since it uses a depth-first search method. For further details we refer to Section 5.6.

In the same way, the drawbacks about mining quantitative patterns that we dealt in Section 4.5 also occur for BreadthPIMS and DepthPIMS. Besides, since both algorithms are based on Vertical Database Format, both of them need to have the frequent 2-patterns and if we execute a brute force method we can find a lot of problems because there exist a lot of patterns. A solution to this issue is to apply a Pattern Growth method to mine the frequent 2-patterns (see Section 4.5 for more details).

6.8 Conclusions

In this Chapter we have introduced two algorithms that deal with these issues for mining quantitative interval patterns. Our main contributions of this Chapter are as follow:

- We extend the triangular matrix representation to include also quantitative information. The main advantage of this representation is that the thirteen quantitative version of Allen's relations are perfectly summarized in a straightforward structure. Note that not all the relations need quantitative information.

- We introduce a novel algorithm, called BreadthPIMS, which stands for **Breadth**-first search algorithm for **P**oint and **I**nterval **M**etric **S**equences, capable of discovering the whole set of frequent quantitative patterns containing relations between points, intervals or points and intervals. BreadthPIMS is based on the Vertical Database Format Strategy and, by means of temporal reasoning principles, it uses efficient methods to generate only the right candidates that can be derived from shorter quantitative patterns.

- We describe in detail the algorithm DepthPIMS(**Depth**-first search algorithm for **P**oint and **I**nterval **M**etric **S**equences), an implementation also based on Vertical Database Format Strategy, that uses the same pattern representation and capable of finding the same final quantitative pattern set as BreadthPIMS. The main differences of this algorithm with respect to the previous one are: 1) the temporal reasoning used, i.e. the relations taken into account to infer new relations are different in both algorithms; and 2) the search method is also different in both algorithms: while BreadthPIMS executes a breadth-first search, DepthPIMS uses a depth-first search. To the best of our knowledge, these algorithms are the first one based on Vertical Database Format strategy for mining quantitative patterns.

- We show the problem that Vertical Database Algorithms face since the have to find all the possible temporal distances that appear in the frequent 2-patterns and we give a valid solution so as to obtain a good execution of these kind of algorithms.

- We do an exhaustive comparison between BreadthPIMS and DepthPIMS, analysing the advantages and drawbacks that comes from the execution of their searches. Although both transition functions are efficient, the transition function of BreadthPIS give a more reduced candidate set and is the most optimized. Anyway, we show that, in general, there not exist a significant difference between both algorithms but, nevertheless, DepthPIMS reach lower support since it does not suffer from the problems of memory overflow associated with a breadth-first search. That reason lead us to choose DepthPIMS as a better general algorithm.

Chapter 7

ClaSP: An Efficient Algorithm for Mining Frequent Closed Sequences

Finally, we introduce an algorithm for mining closed patterns with point-based events. Since this chapter has been already published in the proceedings on Pacific-Asia Conference on Knowledge Discovery and Data Mining, we introduce this Chapter as the paper appeared there.

7.1 Introduction

Sequence Data Mining (SDM) is a well-extended field of research in Temporal Data Mining that consist of looking for a set of patterns frequently enough occurring across time among a large number of objects in a given input database. The threshold to decide if a pattern is meaningful is called *minimum support*. SDM has been widely studied [Srikant and Agrawal, 1996; Zaki, 2001; Han et al., 2000; Pei et al., 2004; Ayres et al., 2002], with broad applications, such as the discovery of motifs in DNA sequences, analysis of customer purchase sequences, web click streams, and so forth.

The task of discovering the set of all frequent sequences in large databases is challenging as the search space is extremely large. Different strategies have been proposed so far, among which *SPADE*, exploiting a Vertical Database Format [Zaki, 2001], and PrefixSpan, based on projected Pattern Growth [Pei et al., 2004] are the most popular ones. These strategies show good performances in databases containing short frequent sequences or when the support threshold is not very low. Unfortunately, when long sequences are mined, or when a very low support threshold is used, the performance of such algorithms decreases dramatically and the number of frequent patterns increases sharply, resulting in too many meaningless and redundant patterns. Even worse, sometimes it is impossible to complete the algorithm execution due to a memory overflow.

One of the most interesting proposals to solve both problems are so called *closed sequences* [Yan et al., 2003], based on the same notion for regular frequent closed itemsets, as introduced by Pasquier et al. [Pasquier et al., 1999]. A frequent sequence is said to be closed, if there no exists a supersequence with the same support in the database. The final collection of closed sequences provides a much more simplified output, still keeping all the information about the frequency of each of the sequences. Some algorithms have been developed to find the complete set of closed sequences, where most of them are based on the Pattern Growth strategy [Yan et al., 2003; Wang et al., 2007].

In this Chapter, we propose a new algorithm, called ClaSP (Closed Sequential Patterns algorithm) which exploits several efficient search space pruning methods. Depending on the properties of the database, we argue about the desirability of using the Vertical Database Format as compared to Pattern Growth techniques. We also show the suitability of the Vertical Database Format in obtaining the frequent closed sequence set, and how, with some database configurations, a standard Vertical Database Format algorithm can already be faster than Pattern Growth algorithms for closed sequences, by only adding a simple post-processing step. Experiments on both synthetic and real datasets show that ClaSP generates the same complete closed sequences as CloSpan [Yan et al., 2003] but has much better performance Figures.

The remaining of the Chapter is organized as follows. Section 7.2 introduces the preliminary concepts of frequent closed sequential pattern mining and the notation used in the Chapter. In Section 7.3, we present the most relevant related work. In Section 7.4, the pruning methods and ClaSP algorithm are presented. The performance study is presented in Section 7.5 and, finally, we state our conclusions in Section 7.6.

7.2 Problem setting

Let $\mathcal{I}$ be a set of items. A set $X = \{e_1, e_2, \ldots, e_k\} \subseteq \mathcal{I}$ is called an itemset or k-itemsets if it contains k items. For simplicity, from now on we denote an itemset I as a concatenation of items between brackets. So, $I_1 = (ab)$ and $I_2 = (bc)$ are both two 2-itemsets. Also, without loss of generality, we assume the items in every itemset are represented in a lexicographic order.

A sequence s is a tuple $s = \langle I_1 I_2 \, \ldots \, I_n \rangle$ with $I_i \in \mathcal{I}$, and $\forall i : 1 \leq i \leq n$. We denote the size of a sequence $|s|$ as the number of itemsets in that sequence. We denote the length of a sequence ($l = \sum_{i=1}^{n} |I_i|$) as the number of items in it, and every sequence with k items is called a k-sequence. For instance, the sequence $\alpha = \langle (ab)(bc) \rangle$ is a 4-sequence with a size of 2 itemsets.

We say $\alpha = \langle I_{a_1} I_{a_2} \ldots I_{a_n} \rangle$ is a subsequence of another sequence $\beta = \langle I_{b_1} I_{b_2} \, \ldots I_{b_m} \rangle$ (or β is a supersequence of α), denoted as $\alpha \preceq \beta$, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $I_{a_1} \subseteq I_{b_{j_1}}, I_{a_2} \subseteq I_{b_{j_2}}, \ldots, I_{a_n} \subseteq I_{b_{j_n}}$. For instance, $\langle (b)(c) \rangle$ is a subsequence of $\langle (ab)(bc) \rangle$, since $(b) \subseteq (ab)$ and $(c) \subseteq (bc)$ and the order in the itemsets is preserved. Furthermore, the sequence $\langle (b)(c) \rangle$ is not a subsequence of $\langle (abc) \rangle$.

In the rest of the work, we use the terms pattern and sequence interchangeably.

An input sequence is is a tuple $is = \langle id, s \rangle$ with $id \in \mathbb{N}$ and s is a sequence. We call id the identifier of the input sequence. We say that an input sequence is contains another sequence α, if $\alpha \preceq s$.

A sequence database D is collection of input sequences $D = \langle s_1 s_2 \ldots s_n \rangle$, incrementally ordered by the identifier of the contained sequences. In Table 7.1 we show a sample input database D with four input sequences.

Definition 7.2.1. The *support* (or *frequency*) of a sequence, denoted as $\sigma(\alpha, D)$, is the total number of input sequences in the input database D that contain α. A pattern or

Sequence Id.	Sequence
1	$\langle(a)(ab)(bc)\rangle$
2	$\langle(a)(abc)\rangle$
3	$\langle(d)(a)(ab)(bc)\rangle$
4	$\langle(d)(ad)\rangle$

Table 7.1: A sample sequence database.

sequence is called *frequent* if it occurs at least a given user specified threshold *min_sup*, called the minimum support. $\mathcal{FS}$ is the whole collection of frequent sequences. The problem of frequent sequence mining is now to find $\mathcal{FS}$ in a given input database, for a given minimum support threshold.

Given a sequence $\alpha = \langle I_1 I_2 \ldots I_n\rangle$ and an item e_i, we define the *s-extension* α' as the super-sequence of α, extending it with a new itemset containing a single item e_i, $\alpha' = \langle I_1 I_2 \ldots I_n I_{n+1}\rangle$, $I_{n+1} = (e_i)$. We define the *i-extension* of α if the last itemset I'_n of $\alpha' = \langle I_1 I_2 \ldots I'_n\rangle$ satisfies $(I'_n = I_n \cup e_i)$. That is, the item e_i is added to I_n. For instance, given the sequence $\alpha = \langle(a)(b)\rangle$ and an item $c \in \mathcal{I}$, the sequence $\beta = \langle(a)(b)(c)\rangle$ is an s-extension and $\gamma = \langle(a)(bc)\rangle$ is an i-extension.

Given two sequences β and γ such that both are s-extensions (or i-extensions) of a common prefix α, with items e_i and e_j respectively, we say β precedes γ, $\beta < \gamma$, if $e_i <_{lex} e_j$ in a lexicographic order. If, on the contrary, one of them is an s-extension and the other one is i-extension, the s-extension always precedes the i-extension.

Definition 7.2.2. If a frequent sequence α does not have another supersequence with the same support, we say that α is a *closed* sequence. Otherwise, if a frequent sequence β has a super-sequence γ with exactly the same support, we say that β is a non-closed sequence and γ *absorbs* β. The whole set of frequent closed sequences is denoted by $\mathcal{FCS}$. More formally, $\alpha \in \mathcal{FCS}$ if $\forall \beta \in \mathcal{FS}, \alpha \preceq \beta, \sigma(\alpha, D) \neq \sigma(\beta, D)$. The problem of closed sequence mining is now to find $\mathcal{FCS}$ in a given input database, for a given minimum support threshold.

Clearly, the collection of frequent closed sequences is smaller than the collection of all frequent sequences.

Example 1. *In our sample database, shown in Table 7.1, for a support min_sup* $= 2$*, we find* $|\mathcal{FCS}| = 5$ *frequent closed sequences,* $\mathcal{FCS} = \{\langle(a)\rangle, \langle(d)(a)\rangle, \langle(a)(ab)\rangle, \langle(a)(bc)\rangle, \langle(a)(ab)(bc)\rangle\}$*, while the corresponding* $\mathcal{FS}$ *has* 27 *frequent sequences.*

7.3 Related work

Looking for frequent sequences in sequence databases was first proposed by Agrawal and Srikant [Agrawal and Srikant, 1995; Srikant and Agrawal, 1996]. Their algorithms (Apriori-based) consist of executing a continuous loop of a candidate generation phase followed by a support checking phase. Two main drawbacks appear in those algorithms:

1) they need to do several scans of the database to check the support of the candidates; and 2) a breath-first search is needed for the candidate generation, leading to high memory consumption.

Later, two other strategies were proposed: 1) depth-first search based on a Vertical Database Format [Zaki, 2001] and 2) projected Pattern Growth [Han et al., 2000; Pei et al., 2004]. The Vertical Database Format strategy was created by Zaki in the SPADE algorithm [Zaki, 2001] which is capable of obtaining the frequent sequences without making several scans of the input database. His algorithm allows the decomposing of the original search tree in independent problems that can be solved in parallel in a depth-first search (DFS), thus enabling the processing of big databases.

The Pattern growth strategy was introduced by Han et al. [Han et al., 2000] and it consists in algorithms that obtain the whole frequent sequence set by mean of techniques based on the so called projected Pattern Growth. The most representative algorithm in this strategy is PrefixSpan [Pei et al., 2004]. PrefixSpan defines a projected database as the set of suffixes with respect to a given prefix sequence. After projecting by a sequence, new frequent items are identified. This process is applied in a recursive manner by means of DFS, identifying the new frequent sequences as the concatenation of the prefix sequence with the frequent items that are found.

PrefixSpan shows good performance and scales well in memory, especially with sparse databases or when databases mainly consist of small itemsets. However, when we deal with large dense databases that have large itemsets, the performance of PrefixSpan is worse than that of SPADE. In order to show this issue, we have conducted several tests. In a sequential database, several important properties have an influence on the algorithms execution, some of which are shown in Table 7.2.

Abbr.	Meaning
D	Number of sequences (in 000s)
C	Average itemset in a sequence
T	Average items in a itemset
N	Number of different items (in 000s)
S	Average itemsets in maximal sequences
I	Average items in maximal sequences

Table 7.2: Parameters for IBM Quest data generator.

We define the database density as the quotient $\delta = \frac{T}{N}$. We have used the well-known data generator provided by IBM to run SPADE and PrefixSpan with different configurations. In Figures 7.1 and 7.2 we can observe the behaviour of both SPADE and PrefixSpan when we vary the density and the number of itemsets. In Figure 7.1 we show the running time of the algorithms with a different number of items ($N \in \{100, 500, 1000, 2500\}$ items) with a constant $T = 20$ value. Since for a database, the density grows either if the numerator increases or the denominator decreases, the Figures have been obtained just varying the denominator. Besides, Figure 7.2 depicts the behaviour of the algorithms when the number of itemsets is changed between values of $C \in \{10, 20, 40, 80\}$ while we keep the density constant ($\delta = \frac{20}{2500}$). The higher δ the more dense the database. We can see that PrefixSpan shows good results when both density and the number of itemsets are low, but when a database is denser and parameter C grows, we notice how SPADE outperforms PrefixSpan.

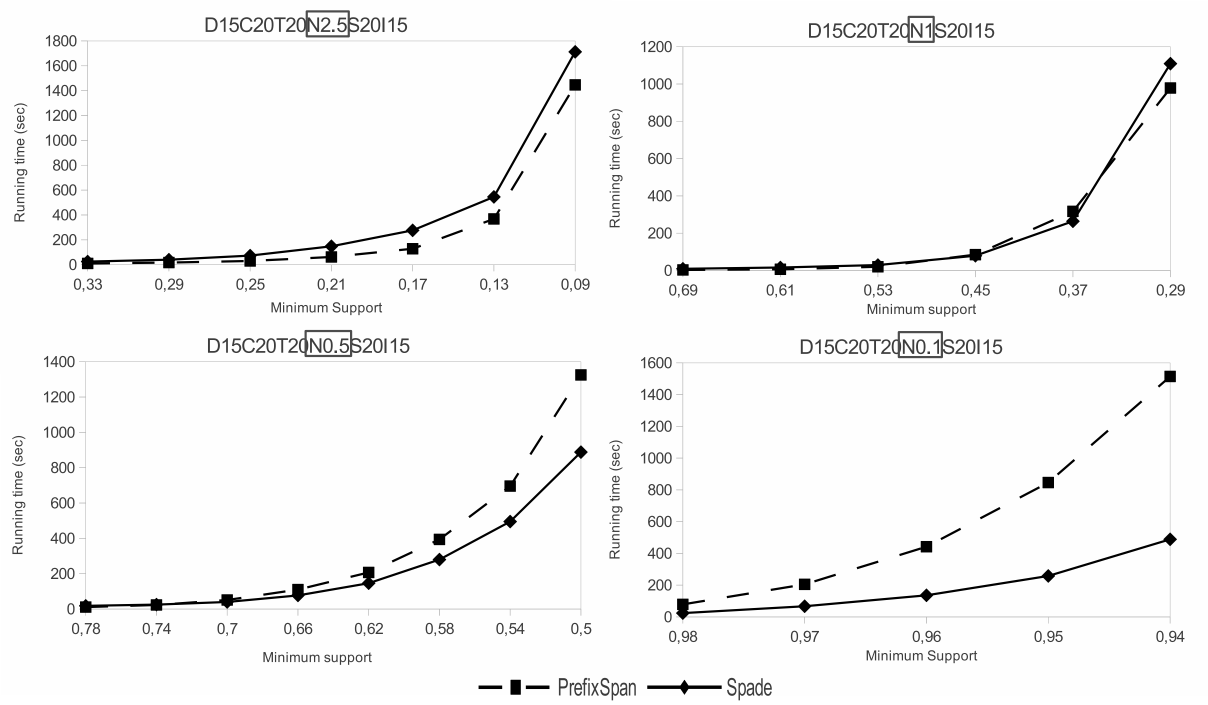


Figure 7.1: Behaviour of SPADE and PrefixSpan when density changes (in the number of items).

For mining closed sequences, there exist two approaches: 1) run any algorithm for mining all frequent sequences and execute a post-processing step to filter out the set of closed sequences, or 2) obtain the set of closed sequences by gradually discarding the non-closed ones. Some algorithms have been developed to find the complete set of closed sequences. The most important algorithms developed so far, are CloSpan [Yan et al., 2003] and Bide [Wang et al., 2007], both derived from PrefixSpan. While CloSpan uses a prefix tree to store the sequences and uses two methods to prune non-frequent sequences, Bide executes some checking steps in the original database that allows it to avoid maintaining the sequence tree in memory. However, to the best of our knowledge, there exist no algorithms for closed sequence mining based on the Vertical Database Format as is presented here.

7.4 ClaSP: algorithm and implementation

In this Section, we formulate and explain every step of our ClaSP algorithm. ClaSP has two main phases: The first one generates a subset of $\mathcal{FS}$ (and superset of $\mathcal{FCS}$) called Frequent Closed Candidates ($\mathcal{FCC}$), that is kept in main memory; and the second step executes a post-pruning phase to eliminate from $\mathcal{FCC}$ all non-closed sequences to finally obtain exactly $\mathcal{FCS}$.

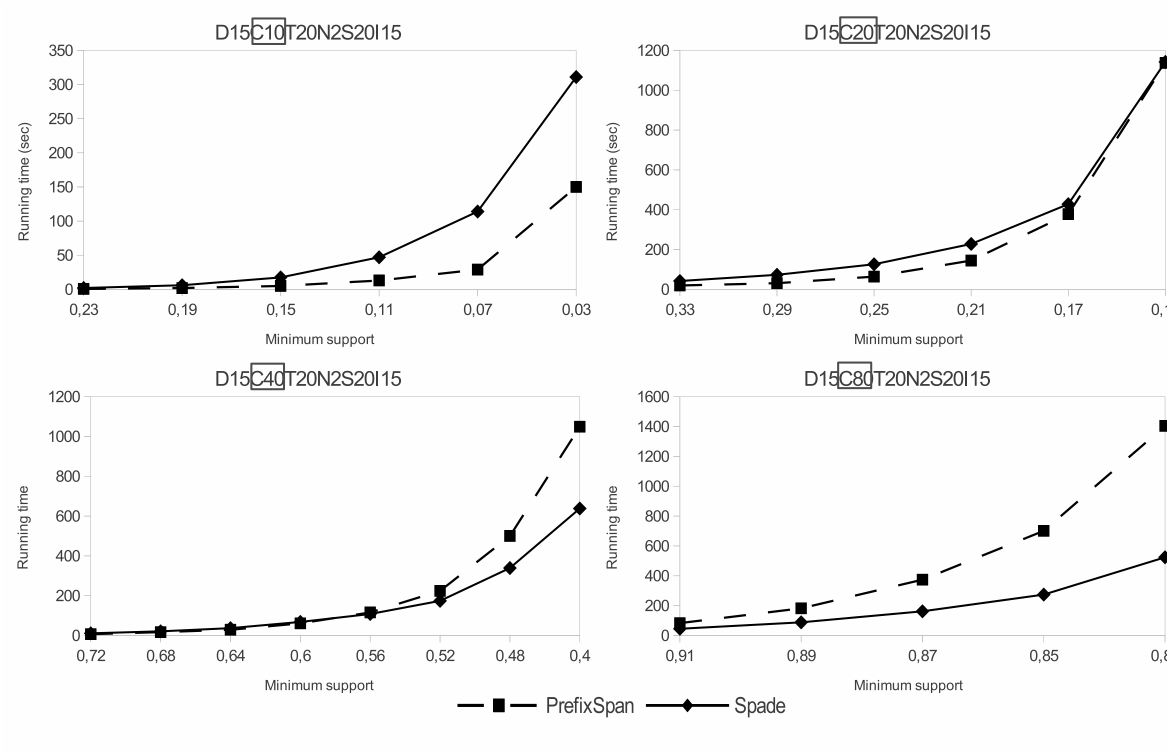


Figure 7.2: Behaviour of SPADE and PrefixSpan when the number of itemsets changes.

Algorithm 30 ClaSP

```
$\mathcal{F}_1$ = {frequent 1-sequences}
$\mathcal{FCC} = \emptyset$, $\mathcal{FCS} = \emptyset$
for all i $\in \mathcal{F}_1$ do
    $\mathcal{F}_{ie}$ = {frequent 1-sequences greater than i}
    $\mathcal{FCC}_i$=DFS-PRUNING(i,$\mathcal{F}_1$,$\mathcal{F}_{ie}$)
    $\mathcal{FCC} = \mathcal{FCC} \cup \mathcal{FCC}_i$
end for
$\mathcal{FCS}$ = N-ClosedStep($\mathcal{FCC}$)
Ensure: The final closed frequent pattern set $\mathcal{FCS}$
```

Algorithm 30, *ClaSP*, shows the pseudocode corresponding to the two main steps. It first finds every frequent 1-sequence, and after that, for all of frequent 1-sequences, the method *DFS-Pruning* is called recursively to explore the corresponding subtree (by doing a depth-first search). $\mathcal{FCC}$ is obtained when this process is done for all of the frequent 1-sequences and, finally, the algorithm ends removing the non-closed sequences that appear in $\mathcal{FCC}$.

Algorithm 31, *DFS-Pruning*, executes recursively both the candidate generation (by means of i-extensions and s-extensions) and the support checking, returning a part of $\mathcal{FCC}$ relative to the pattern p taken as parameter. The method takes as parameters two sets with the candidate items to do s-extensions and i-extensions respectively ($\mathcal{S}_n$ and $\mathcal{I}_n$ sets). The algorithm first checks if the current pattern p can be discarded, by using the method *checkAvoidable* (this algorithm is explained later in algorithm 34). Lines 4-9 perform all the s-extensions for the pattern p and keep in $\mathcal{S}_{temp}$ the items which make frequent extensions. In line 10, the method *ExpChildren* (algorithm 33) is called, and there, *DFS-Pruning* is executed for each new frequent s-extensions. Lines 11-16 and 17 perform the same steps, with i-extensions. Finally, in line 19, the complete frequent patterns set (with a prefix p) is returned.

To store the patterns in memory, we use a lexicographic sequence tree. The elements in the tree are sorted by a lexicographic order according to extension comparisons (see Section 7.2). In Figure 7.3 we show the associated sequence tree for $\mathcal{FS}$ in our example and we denote an s-extension with a line, and an i-extension with a dotted line. This tree is traversed by algorithms 30, 31 and 33, using a depth-first traversal.

Algorithm 31 DFS-Pruning(p, $\mathcal{S}_n$, $\mathcal{I}_n$)

```
Require: Current frequent pattern $p = (s_1, s_2, \ldots, s_n)$, set of items for s-extension $\mathcal{S}_n$, set of items for i-extension $\mathcal{I}_n$
 1: $\mathcal{S}_{temp} = \emptyset$, $\mathcal{I}_{temp} = \emptyset$
 2: $\mathcal{F}_i = \emptyset$, $\mathcal{P}_s = \emptyset$, $\mathcal{P}_i = \emptyset$
 3: if (not checkAvoidable( $p$, $\mathcal{I}(\mathcal{D}_p)$ )) then
 4:     for all  i $\in \mathcal{S}_n$  do
 5:         if ($p' = (s_1, s_2, \ldots, s_n, \{i\})$ is frequent) then
 6:             $\mathcal{S}_{temp} = \mathcal{S}_{temp} \cup \{i\}$
 7:             $\mathcal{P}_s = \mathcal{P}_s \cup \{p'\}$
 8:         end if
 9:     end for
10:     $\mathcal{F}_i = \mathcal{F}_i \cup \mathcal{P}_s \cup$ ExpChildren($\mathcal{P}_s$,$\mathcal{S}_{temp}$,$\mathcal{S}_{temp}$)
11:     for all  i $\in \mathcal{I}_n$  do
12:         if ($p' = (s_1, s_2, \ldots, s_n \cup \{i\})$ is frequent) then
13:             $\mathcal{I}_{temp} = \mathcal{I}_{temp} \cup \{i\}$
14:             $\mathcal{P}_i = \mathcal{P}_i \cup \{p'\}$
15:         end if
16:     end for
17:     $\mathcal{F}_i = \mathcal{F}_i \cup \mathcal{P}_i \cup$ ExpChildren($\mathcal{P}_s$,$\mathcal{S}_{temp}$,$\mathcal{I}_{temp}$)
18: end if
19: return  $\mathcal{F}_i$
Ensure: Frequent pattern set $\mathcal{F}_i$ of this node and its children
```

There are two main different changes added in ClaSP with respect to SPADE: (1) the step to check if the subtree of a pattern can be skipped (line 3 of algorithm 31), and (2) the step where the remaining non-closed patterns are eliminated (line 6 of algorithm 30).

To prune the space search, ClaSP used the method *CheckAvoidable* that is inspired on the pruning methods used in CloSpan. This method tries to find those patterns $p = \langle \alpha\, e_j \rangle$ and $p' = \langle \alpha\, e_i\, e_j \rangle$, such that, all of the appearances of p are in those of p', i.e., if every time we find a sequence α followed by an item e_j, there exists an item e_i between them, then we can safely avoid the exploration of the subtree associated to the pattern p. In order to find this kind of patterns, we define two numbers: 1) $l(s,p)$, is the size of all the suffixes with respect to p in sequence s, and 2) $\mathcal{I}(\mathcal{D}_p) = \sum_{i=1}^{n} l(s_i, p)$, the total number of remaining items with respect to p for the database $\mathcal{D}$, i.e. the addition of all of $l(s,p)$ for every sequence in the database. Using $\mathcal{I}(\mathcal{D})$ and the subsequence checking operation, in algorithm 34, ClaSP checks the equivalence between the $\mathcal{I}(\mathcal{D})$ values for two patterns: Given two sequences, s and s', such that $s \preceq s'$, if $\mathcal{I}(\mathcal{D}_s) = \mathcal{I}(\mathcal{D}_{s'})$, we can deduce that the support for all of their descendants is just the same.

In algorithm 34, the pruning phase is implemented by two methods: 1) Backward sub-pattern checking and 2) Backward super-pattern checking. The first one (lines 8-10) occurs when we find a pattern which is a subsequence of a pattern previously found with the same $\mathcal{I}(\mathcal{D})$ value. In that case, we can avoid exploring this new branch in the tree for

Algorithm 32 N-ClosedStep($\mathcal{FCC}$)

```
Require: A frequent closed candidates set $\mathcal{FCC}$
$\mathcal{FCS} = \emptyset$
A hash table $H$ is created
for all $p \in \mathcal{FCC}$ do
    Add a new entry $\langle \mathcal{T}(\mathcal{D}_p), p \rangle$ in $H$
end for
for all entry $e \in H$ do
    for all $p_i \in e$ do
        for all $p_j \in e$, $j > i$ do
            if ($p_i.support() = p_j.support()$) then
                if ($p_i \preceq p_j$) then
                    Remove $p_i$ from $e$
                else
                    if ($p_j \preceq p_i$) then
                        Remove $p_j$ from $e$
                    end if
                end if
            end if
        end for
    end for
    $\mathcal{FC}_e$= all patterns $p \in e$
    $\mathcal{FCS} = \mathcal{FCS} \cup \mathcal{FC}_e$
end for
return $\mathcal{FCS}$
Ensure: The final closed frequent pattern set $\mathcal{FCS}$
```

Algorithm 33 ExpChildren($\mathcal{P}$, $\mathcal{S}_s$, $\mathcal{S}_i$)

```
Require: The pattern set $\mathcal{P}$ which contains all the patterns whose children are going to be explore, the set of valid items $\mathcal{S}_s$ which generate the $\mathcal{P}$ set by means of s-extensions, the set of valid items $\mathcal{S}_i$ which generate the $\mathcal{P}$ set by means of i-extensions
$\mathcal{F}_s = \emptyset$
for all p $\in \mathcal{P}$ do
    $\mathcal{I}$ = Elements in $\mathcal{S}_i$ greater than the last item $e_i$ in $p$
    $\mathcal{F}_s = \mathcal{F}_s \cup$ DFS-Pruning($p$,$\mathcal{S}_s$,$\mathcal{I}$)
end for
return $\mathcal{F}_s$
Ensure: Frequent pattern set $\mathcal{F}_s$ for all of the patterns' children
```

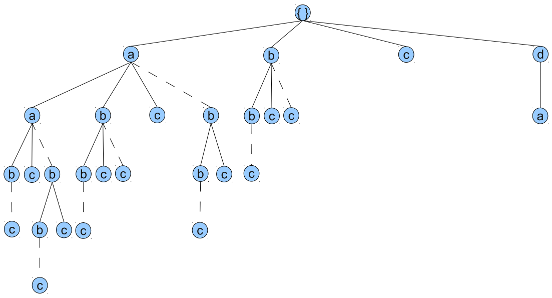


Figure 7.3: Whole lexicographic sequence tree for our thorough example.

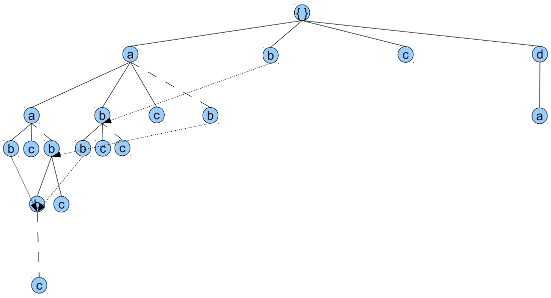


Figure 7.4: Whole lexicographic sequence tree after processing ClaSP algorithm.

this new pattern. The second method (lines 12-16 and 20-24) is the opposite situation and it occurs when we find a pattern that is a super-sequence of another pattern previously found with the same $\mathcal{I}(\mathcal{D})$ value. In this case we can transplant the descendants of the previous pattern to the node of this new pattern.

In Figure 7.4 we show the ClaSP search tree w.r.t. our example without all pruned nodes. In our implementation, to store the relevant branches, we define a hash function

Algorithm 34 CheckAvoidable(p, k)

```
Require: a frequent pattern $p$, its hash-key $k$, hash table $H$
 1: $\mathcal{M}_k$ = Entries in the hash table with key $k$
 2: if ($\mathcal{M}_k = \emptyset$) then
 3:     Insert a new entry $\langle k,p \rangle$ in $H$
 4: else
 5:     for all pair $m \in \mathcal{M}_k$ do
 6:         $p' = m.value()$
 7:         if ($p$.support()=$p'.support()$) then
 8:             //Backward sub-pattern
 9:             if ($p \preceq p'$) then
10:                 $p$ has the same descendants as $p'$, so $p$ points to $p'$ descendants
11:                 return true
12:             else
13:                 //Backward super-pattern
14:                 if ($p' \preceq p$) then
15:                     $p$ has the same descendants as $p'$, so $p$ points to $p'$ descendants
16:                     Remove the current entry $\langle k, p' \rangle$ from $H$
17:                 end if
18:             end if
19:         end if
20:     end for
21:     //Backward super-pattern executed?
22:     if (Pruning method 2 has been accomplished) then
23:         Add a new entry $\langle k, p \rangle$ in $H$
24:         return true
25:     end if
26: end if
27: //k does not exist in the hash table or it does exist but the present patterns are not related with p
28: Add a new entry $\langle k, p \rangle$ in $H$
29: return false
Ensure: It answers if the generation of $p$ can be avoided
```

with $\mathcal{I}(\mathcal{D})$ value as key and the pattern (i.e. the node in the tree for that pattern) as value ($\langle \mathcal{I}(\mathcal{D}_p), p \rangle$). We use a global hash table and, when we find a sequence p, if the backward sub-pattern condition is accomplished, we do not put the pair $\langle \mathcal{I}(\mathcal{D}_p), p \rangle$, whereas, if the backward super-pattern condition is true, we replace all the previous pairs $\langle \mathcal{I}(\mathcal{D}_{p'}), p' \rangle$ (s.t. $p' \preceq p$) with the new one $\langle \mathcal{I}(\mathcal{D}_p), p \rangle$. If instead we do not find any pattern with the same $\mathcal{I}(\mathcal{D})$ value of the pattern p, or those patterns with the same value are not related with p by means of the subsequence operation, we put the pair $\langle \mathcal{I}(\mathcal{D}_p), p \rangle$ to the global hash table (line 28). Note that when one of the two pruning conditions is true, we also need to check if the support for s and s' is the same since two $\mathcal{I}(\mathcal{D}_p)$ and $\mathcal{I}(\mathcal{D}_{p'})$ values can be equal but they do not necessarily have the same support.

We also need to consider all of the $\mathcal{I}(\mathcal{D})_s$ for every appearance in a sequence. For instance, in our example shown in Table 7.1, regarding the three first sequences, if we consider just the first $\mathcal{I}(\mathcal{D}_s)$ for the first appearance, if we have the pattern $\langle (a)(b) \rangle$ in our example, we deduce that $\mathcal{I}(\mathcal{D}_{\langle (a)(ab) \rangle}) = \mathcal{I}(\mathcal{D}_{\langle (a)(b) \rangle})$ (both with value $\mathcal{I}(\mathcal{D}) = 5$), so we can avoid generating the descendants of $\langle (a)(b) \rangle$ because they are the same as in $\langle (a)(ab) \rangle$. However, we can check that $\langle (a)(bc) \rangle$ is frequent (with support 3), whereas $\langle (a)(abc) \rangle$ is not (support 1). This forces us to count in $\mathcal{I}(\mathcal{D}_s)$, all the number of items after every appearance.

Finally, regarding the non-closed pattern elimination (algorithm 32), the process consists of using a hash function with the support of a pattern as key and the pattern itself as value. If two patterns have the same support we check if one contains the other, and if this condition is satisfied, we remove the shorter pattern. Since the support value as key provoke a high number of collisions in the hash table (implemented with closed addressing), we use a $\mathcal{T}(\mathcal{D}_p) = \sum_{i=1}^{n} id(s_i)$ value, defined as the sum of all sequence ids where a pattern p appears. However, as the equivalence of $\mathcal{T}(\mathcal{D}_p)$ does not imply the equivalence of support, after checking that two patterns have the same $\mathcal{T}(\mathcal{D}_p)$ value, those patterns have to have the same support to remove one of them.

7.5 Performance study

We exhaustively experimented on both synthetic and real world datasets. To generate the synthetic data, we have used the IBM data generator mentioned above (see Section 7.3). In all our experiments we compare the performance of three algorithms: CloSpan, ClaSP and SPADE. For the last algorithm we add the same non-closed candidate elimination phase which is used in ClaSP to obtain $\mathcal{FCS}$.

All experiments are done on a 4-cores of 2.4GHZ Intel Xeon, running Linux Ubuntu 10.04 Server edition. All the three algorithms are implemented in Java 6 SE with a Java Virtual Machine of 16GB of main memory.

Figure 7.5 shows the number of patterns and performance for the dataset D5C10T5N5 S6I4 (-rept 1 -seq.npats 2000 -lit.npats 5000). Figure 7.5(a) shows the number of frequent patterns, the patterns processed by ClaSP, and the number of closed patterns. We can see how there is approximately an order of difference between these numbers, i.e. for every 100 frequent patterns we process around 10 patterns by ClaSP, and approximately only 1 of these 10 patterns is closed. Figure 7.5(b) shows the running time. ClaSP clearly outperforms both SPADE and Clospan. For very low support (below 0.013), we have problems with the execution of SPADE due to the space in memory taken for the algorithm.

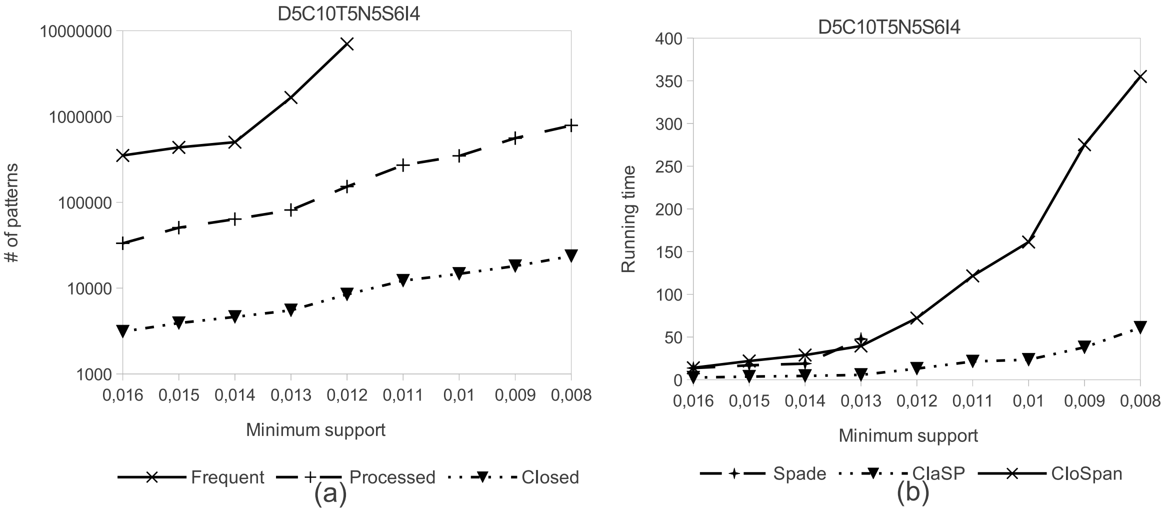


Figure 7.5: Varying support for dataset D5C10T5N5S6I4(-seq.npats 2000 -lit.npats 5000).

Figure 7.6 shows a dataset with larger parameters of C, T and a lower N. This database is denser than the database above and in Figure 7.6(a) we can observe that the difference between frequent patterns and processed patterns is not so big. Therefore, the pruning method checkAvoidable is not so effective and ClaSP and CloSpan are closer to the normal behaviour of SPADE and PrefixSpan, as is shown in Figure 7.6(b). The results show that both ClaSP and SPADE are much faster than Clospan.

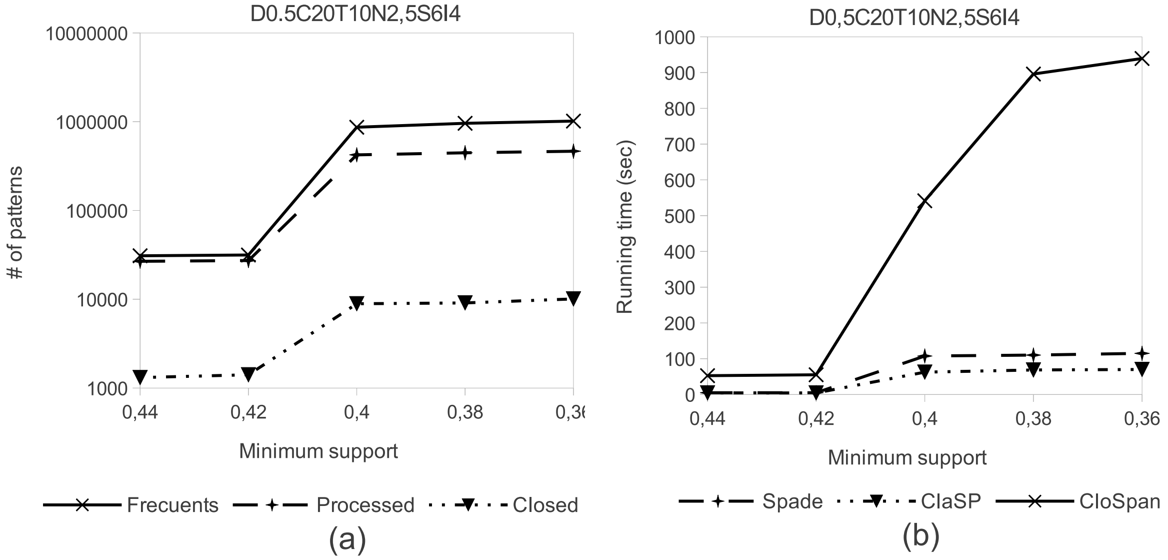


Figure 7.6: Varying support for dataset D0.5C20T10N2.5S6I4(-seq.npats 2000 -lit.npats 5000).

Finally we test our algorithm with the gazelle dataset. This dataset comes from click-stream data from gazelle.com, which no longer exists. The dataset was once used in KDDCup-2000 competition and, basically, it includes a set of page views (each page contains a specific product information) in a legwear and legcare website. Each session contains page views done by a customer over a short period. Product pages viewed in one session are considered as an itemset, and different sessions for one user is considered as a sequence. The database contains 1423 different products and assortments which are viewed by 29369 different users. There are 29369 sequences, 35722 sessions (itemsets), and 87546 page views (items). The average number of sessions in a sequence is around 1. The average number of pageviews in a session is 2. The largest session contains 342 views, the longest sequence has 140 sessions, and the largest sequence contains 651 page views. Figure 7.7 shows the runtime with several support (from 0.03% to 0.015%). We compare the runtime behaviour for both ClaSP and CloSpan and we can see how ClaSP outperforms CloSpan.

All the experiments show that ClaSP outperforms CloSpan, even if databases are sparse. This is because of, in very sparse databases, a high number of patterns are found only with extremely low support. Therefore, the lower the support is, the more patterns are found and the more items are chosen to create patterns. In this point, CloSpan, as PrefixSpan, is penalized since the algorithm has to projects several times the same item in the same sequence, having a worse time with respect to ClaSP. Besides, in those algorithms where SPADE is better than PrefixSpan, ClaSP is also faster than CloSpan.

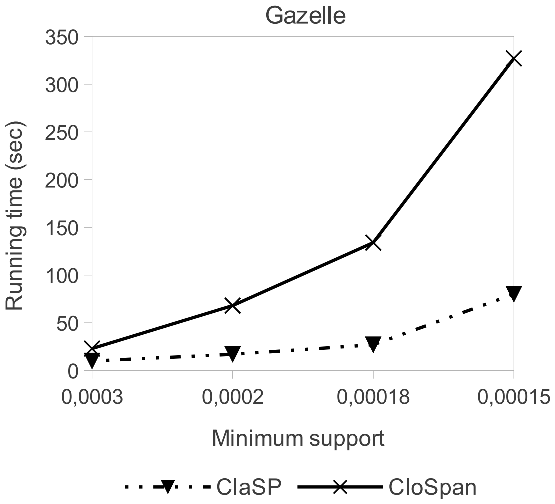


Figure 7.7: Varying support for dataset Gazelle click stream.

7.6 Conclusions

In this Chapter, we study the principles for mining closed frequent sequential patterns and we compare the two main existing strategies to find frequent patterns. We show the benefits of using the Vertical Database Format strategy against Pattern Growth algorithms, especially when facing dense datasets. Then, we introduced a new algorithm called ClaSP to mine frequent closed sequences. This algorithm is inspired on the SPADE algorithm using a Vertical Database Format strategy and uses a heuristic to prune non-closed sequences inspired by the CloSpan algorithm. To the best of our knowledge, this is the first work based on the Vertical Database Format strategy to solve the closed sequential pattern mining problem. In all our tests ClaSP outperforms CloSpan, and even, with certain datasets configuration, SPADE also outperforms CloSpan when the non-closed elimination phase is executed after it.

Chapter 8

Experimental results

In the previous chapters we have analised the algorithms individually. This Chapter presents some comparisons between all the algorithms developed to mine patterns with points and intervals. In Section 8.1, we compare the four algorithms: PaGAPIS, FaSPIP, BreadthPIS and DepthPIS, whose purpose it is to mine qualitative patterns, while in Section 8.2, the same comparisons are made for the quantitative algorithm versions: PaGAPIMS, FaSPIMP, BreadthPIMS and DepthPIMS.

For these experiments we have used the same setting explained in Section 3.4.

8.1 Comparatives of qualitative algorithms

Figure 8.1 shows what happens when there are an intermedium number of transactions per sequence (20 itemsets on average), an average low transaction length (from 2 to 10 items on average) and a number of different items varying between the values of 50 and 100. In summary, it can be stated that the three algorithms that implement a Vertical Database Format strategy (FaSPIP, BreadthPIS and DepthPIS) obtain better results than that which is based on the Pattern Growth strategy (PaGAPIS). However, there are some differences depending on the plots.

Plot 8.1A shows the case of a database with very short itemsets and in which the execution times are quite low (the maximum value that can be seen in the plot is of 100 seconds). BreadthPIS obtains the best result followed by DepthPIS, which is affected by a larger number of infrequent candidates with regard to BreadthPIS. FaSPIP starts with good behaviour but worsens for low supports, being surpassed by PaGAPIS which, although it initially obtains the worst times is, eventually better than FaSPIP.

Plot 8.1B shows the execution of the four algorithms in the same database but considering a slightly larger itemset size (an average of 5 items). For this plot, the best two algorithms are BreadthPIS and FaSPIP. However, BreadthPIS cannot attain the lowest support owing to its breadth-first search. With regard to the two other algorithms, DepthPIS takes longer for high support but after support 0.49 PaGAPIS has a worse execution. As before, DepthPIS takes longer than BreadthPIS because its candidate generation is not so efficient.

Plot 8.1C depicts the results with a sparser database (n=100), and all the algorithms behave in a similar manner. In this case, BreadthPIS is the fastest, but is closely followed by FaSPIP. However, BreadthPIS cannot be executed with support lower than 0.11 since it has memory overflow problems. Moreover, for the lowest support value (0.09) the executions of FaSPIP and DepthPIS converge and take a similar time, and PaGAPIS is clearly the slowest.

Finally Plot 8.1D increments the number of items per itemsets (10 items), and the algorithms have higher executions times and do not attain such low supports as in the previous cases. Again the behaviour is maintained and, for this plot, FaSPIP is the fastest, followed by BreadthPIS which cannot be executed after support 0.63. Furthermore, with regard to DepthPIS and PaGAPIS, both take longer than the other two algorithms, and three different stages can be observed: up to support 0.71 DepthPIS is faster than PaGAPIS; from 0.69 to 0.59 PaGAPIS obtains a better execution time than DepthPIS; and, finally, from 0.59 to the end DepthPIS obtains better execution times. This behaviour in DepthPIS results from the high number of candidates created by its transition function. In general, in this Figure, FaSPIP and BreadthPIS quickly obtain better times than DepthPIS and PaGAPIS, and PaGAPIS has the slowest execution of all.

Figure 8.1 shows, in general, that when the number of items per itemsets (*ptl*) is increased, and thus the density, the algorithms find more patterns at higher supports. We can see that execution time is increased from plot 8.1A to plot 8.1D.

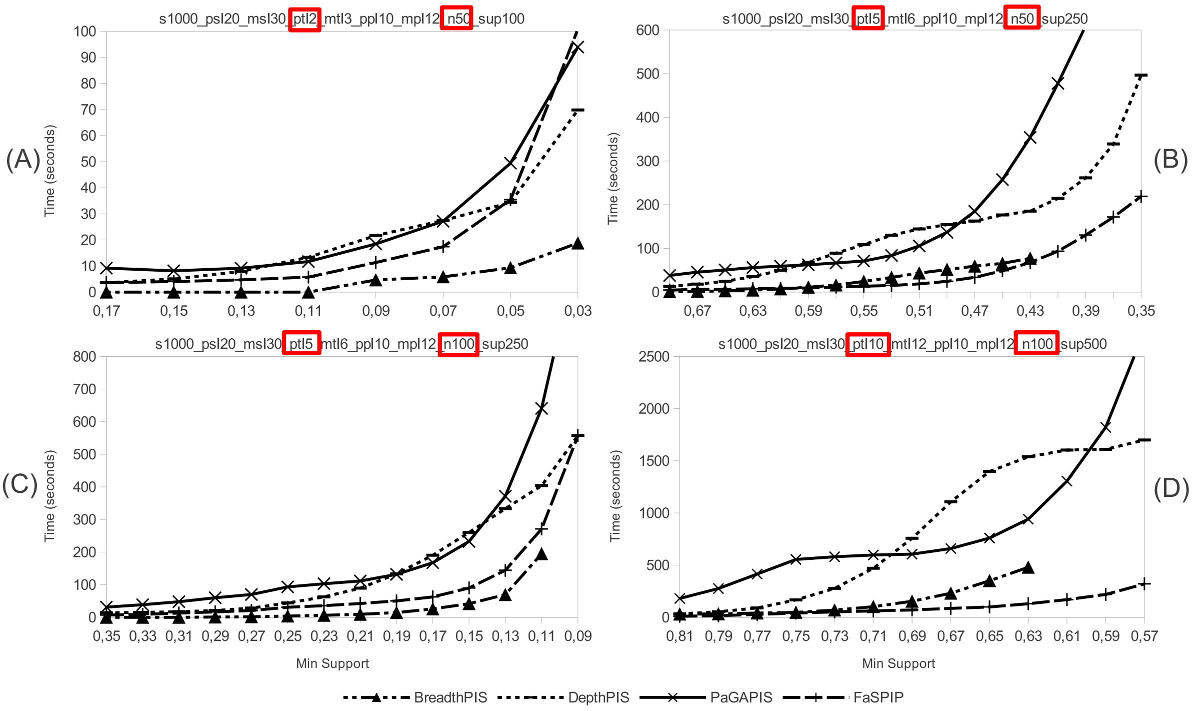


Figure 8.1: Varying support for datasets $s1000\_psl20\_msl30\_ptl2\text{–}5\text{–}10\_mtl3\text{–}6\text{–}12\_ppl10\_mpl12\_n50\text{–}100$.

Figure 8.2 shows what happens to the algorithms in the case of larger databases (10000 sequences) with high values of different items (n = 500 and 1000). Plots 8.2A and 8.2B have a medium sequence length (an average of 20 transactions), a medium-high transaction length (an average of 30 items) and a medium pattern length (an average of 10 items), and 500 and 1000 different items for the database. The datasets corresponding to plots 8.2C and 8.2D have larger sequences (an average of 40 transactions) and smaller transactions (an average of 10 items). All the plots show the same time window (from 0 to 1200 seconds) and it is possible to observe the superiority of all the algorithms based on the Vertical Database Format (FaSPIP, BreadthPIS and DepthPIS) over the algorithm based on the Pattern Growth strategy (PaGAPIS), which are significantly different in terms of execution times in all cases. The behaviour shown by BreadthPIS and DepthPIS is very similar, since the candidates generated by their transition functions are very close. With regard to the behaviour of FaSPIP, it will be observed that it is slightly faster for the denser plots (Plots 8.2A and 8.2C) and slower in the sparser cases (Plots 8.2B and 8.2D). This effect, which also occurs in PaGAPIS, is owing to the fact that in the case of the latter databases, the candidates generated by FaSPIP are largely increased from supports 0.44-0.42, in Plot 8.2B, and 0.36-0.32 in 8.2D. This is related to the nature of its representation (boundary points), in which more candidate boundary points are taken into account, and more time is necessary for checking if those candidate are valid.

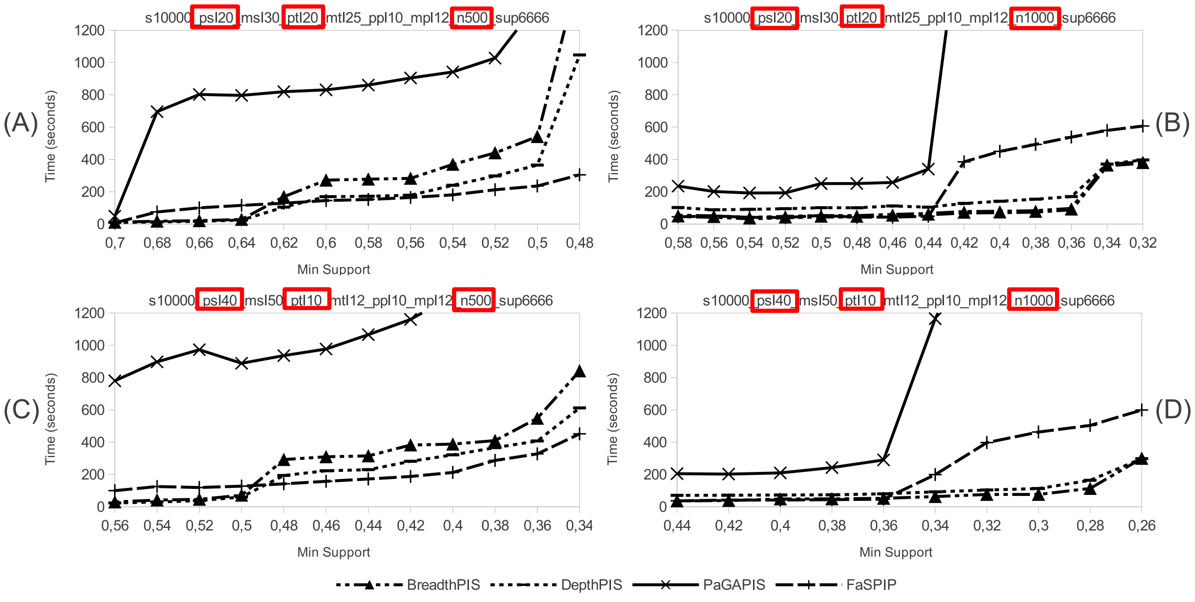


Figure 8.2: Varying support for datasets $s10000\_psl40\_msl50\_ptl10$–20_mtl12–25_ppl10 $\_mpl12\_n500$–1000.

As occurred in Section 2.3, we now show the behaviour when those crucial properties that normally affect the algorithms' execution are changed. We show the executions when we progressively change one of the parameters that affect the density - the number of items - and also when we change the "transactions per sequence" parameter. Figure 8.3 shows what happens when we have a database with moderated parameters (1000 sequences, an average of 20 transaction per sequence, and 12 items per transaction, the length of the patterns in the database being 12 items on average) and we vary the number of different items (100, 200, 500 and 1000 in plots 8.3A, 8.3B, 8.3C and 8.3D). Note that in the four plots the execution time stay in the same units, but the support achieved decreases in each one.

It will be noted that there are two groups in all the plots. On the one hand, both

PaGAPIS and FaSPIP appear and, as in Section 2.3, the denser the database, the better the FaSPIP execution, while the sparser the database, the better the PaGAPIS execution. On the other hand, the executions times for BreadthPIS and DepthPIS are almost identical, and both consequently find a similar number of candidates in their executions, both being faster than PaGAPIS and FaSPIP. Furthermore, in view of the results it will be noted that BreadthPIS and DepthPIS are not so affected by the change of the n value. This difference in the behaviour of the algorithms is therefore owing to the inner representation, and those algorithms that implement a boundary point representation have worse results.

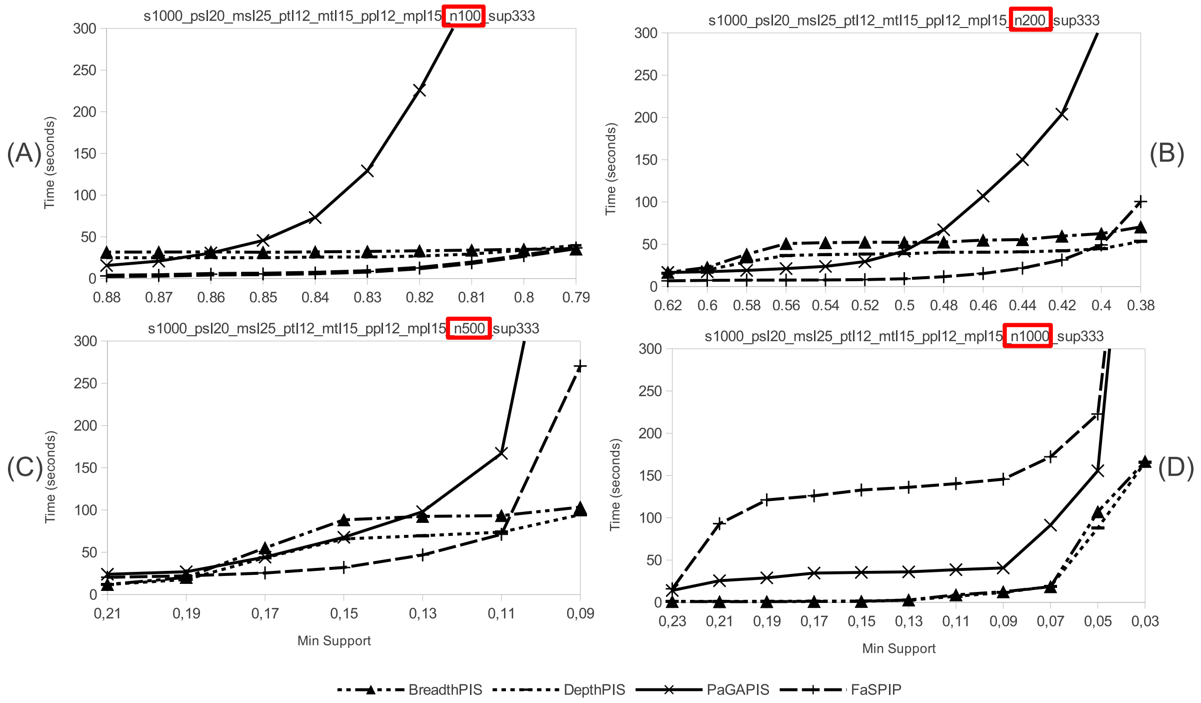


Figure 8.3: Varying support for a same configuration of datasets where we change the number of items (100, 200, 500 and 1000).

Figure 8.4 similarly shows the behaviour with the same database, but when considering 100 different items per database and varying the "transactions per sequence" value (40, 80 and 160 for plots 8.4A, 8.4B and 8.4C). Note that in this case the executions time and support achieved are very different. All the algorithms based on the Vertical Database Format strategy now have very different results to those obtained in Figure 8.3.

Plot 8.4A depicts the results when the database has less transactions per sequence (psl=10), and BreadthPIS is the fastest algorithm followed by PaGAPIS, which is this time faster than FaSPIP, since the Pattern Growth strategy deals better with databases with few transactions per sequence. Lastly, DepthPIS is the slowest since it is greatly affected by the large number of candidates created by its transition function. This wide difference that there is between BreadthPIS and DepthPIS execution times is because BreadthPIS generates much fewer candidates through its transition function. Secondly, in Plot 8.4B, the fastest algorithms are FaSPIP and BreadthPIS, which behave in a similar manner. However, BreadthPIS cannot complete all the support values after support 0.26.

DepthPIS and PaGAPIS are far slower than the other two algorithms, PaGAPIS being the slowest despite having better execution times than DepthPIS from 0.38 to 0.26 supports values. In this Plot it is possible to observe that PaGAPIS starts to be affected by a larger number of transactions per sequence (psl=20) and that DepthPIS still creates a lot of infrequent candidates.

Finally, Plot 8.4C, shows a database with an average of 40 transactions per sequence. Here it will be noted that FaSPIP is the fastest algorithm by far. After FaSPIP is BreadthPIS, closely followed by PaGAPIS for the lowest supports. Lastly, DepthPIS has the highest execution time, despite being under the PaGAPIS curve before the 0.83 support value. However, now it is much more difficult for PaGAPIS to obtain good results than in the point-based case owing to the nature of the algorithm when the interval items are mined. These new problems are explained in greater detail in the following Section, which compares the benefits and drawbacks of both new algorithms, PaGAPIS and FaSPIP. This Figure shows that, as the number of transactions per sequences is increased, there is a sharp deterioration in the execution of both BreadthPIS and DepthPIS. This problem is owing to the Triangular Matrix Representation, as will be discussed in Chapter 9.

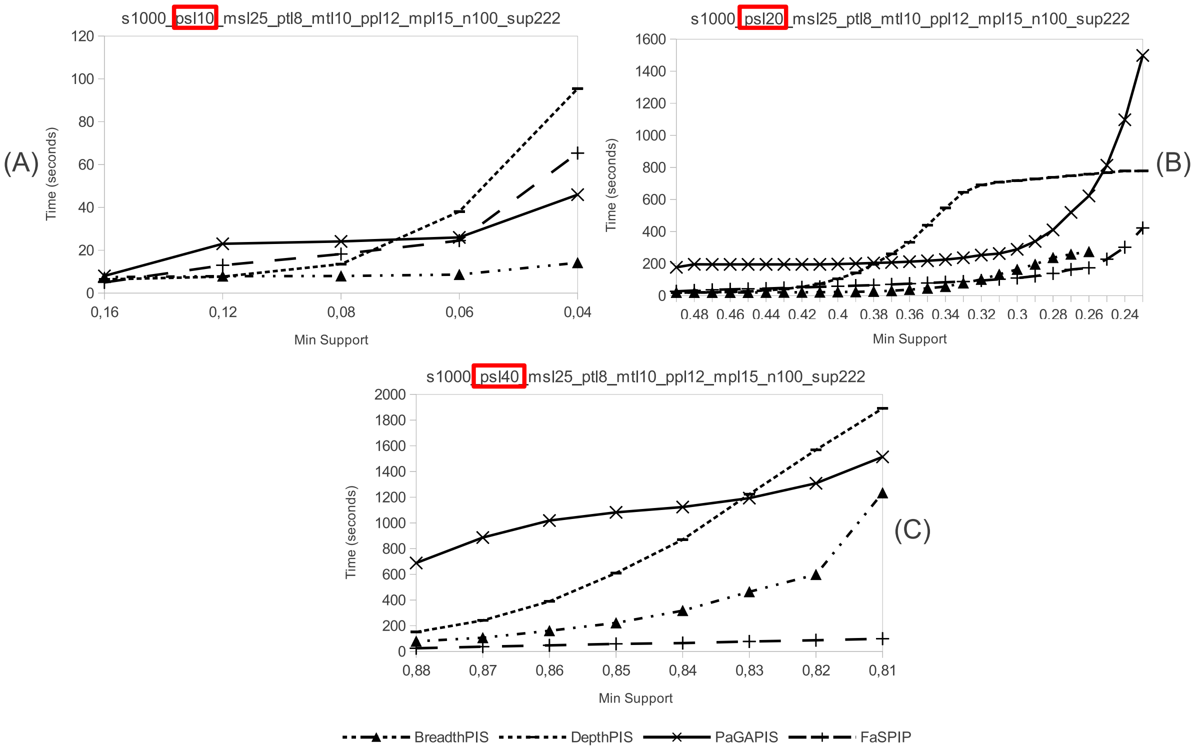


Figure 8.4: Varying support for a same configuration of datasets where we change the number of items per itemset (10, 20 and 40).

8.2 Comparatives of quantitative algorithms

This Section considers those versions that mine the set of frequent 2-patterns for the FaSPIMP, BreadthPIMS and DepthPIMS algorithms using a Pattern Growth method, since this is more efficient than a brute force search (see Section 4.4 and Section 6.5 for more details). What is more, all the optimisations proposed in those sections are considered in all the algorithms, and all the items considered in all the databases can extend from a duration of 1 to 10 time units.

Figure 8.5 shows the behaviour when dealing with databases with 1000 large sequences (an average of 40 transactions), with an average of 10 items per transactione (ptl = 10) and with different values in the number of items (500 and 1000). In both plots 8.5A and 8.5B, the algorithms based on the Vertical Database Format strategy (FaSPIMP, BreadthPIMS and DepthPIMS) are faster than that based on the Pattern Growth strategy (PaGAPIMS). The overall behaviour is maintained in both plots: BreadthPIMS and DepthPIMS are the fastest, both having a similar execution time, and therefore a similar number of candidates created by their transition function. PaGAPIMS is the slowest by far with regard to the other three algorithms. In Plot 8.5B, FaSPIMP is slower than PaGAPIMS up to support 0.12. This is due to the huge number of 2-patterns that the algorithm processes. Later, after support 0.12, FaSPIP starts to improve, since the advantages of the Vertical Database Format as opposed to the Pattern Growth algorithms have a greater presence in these supports.

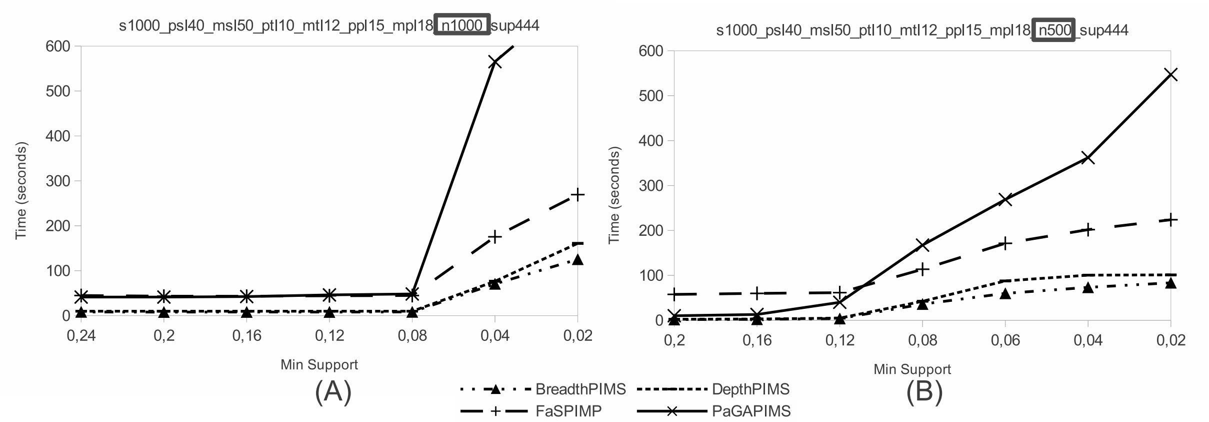


Figure 8.5: Varying support for datasets $s10000\_psl40\_msl50\_ptl10\_mtl12\_ppl15\_mpl18\_n500$–1000.

Figure 8.6 shows the behaviour when dealing with larger databases (10000 sequences) and with high different values of items (n = 500 and 1000). Plots 8.6A and 8.6B have a medium sequence length (an average of 20 transactions), a medium transaction length (an average of 20 items) and a medium pattern length (an average of 10 items). The datasets corresponding to plots 8.6C and 8.6D have larger sequences (an average of 40 transactions) and smaller transactions (an average of 10 items). The two sparser databases are those related to Plots 8.6A and 8.6C whereas the denser ones are shown in Plots 8.6B and 8.6D. For all the plots, BreadthPIMS and DepthPIMS have the fastest executions, followed by FaSPIMP and PaGAPIMS. Nevertheless, it will be noted that in the two sparser databases (Plots 8.6A and 8.6B), the FaSPIMP curve is very close to those of BreadthPIMS and DepthPIMS up to the lowest supports. This problem originates from the fact that more time is spent mining the huge number of frequent 2-patterns and from an increase in the number of frequent patterns. Please recall that FaSPIMP must know all the possible distances that can exist between all the frequent boundary points, whereas BreadthPIMS and DepthPIMS deal with the interval items as a whole, and only spend time finding the temporal distances associated with the temporal relations and not to the size of the intervals.

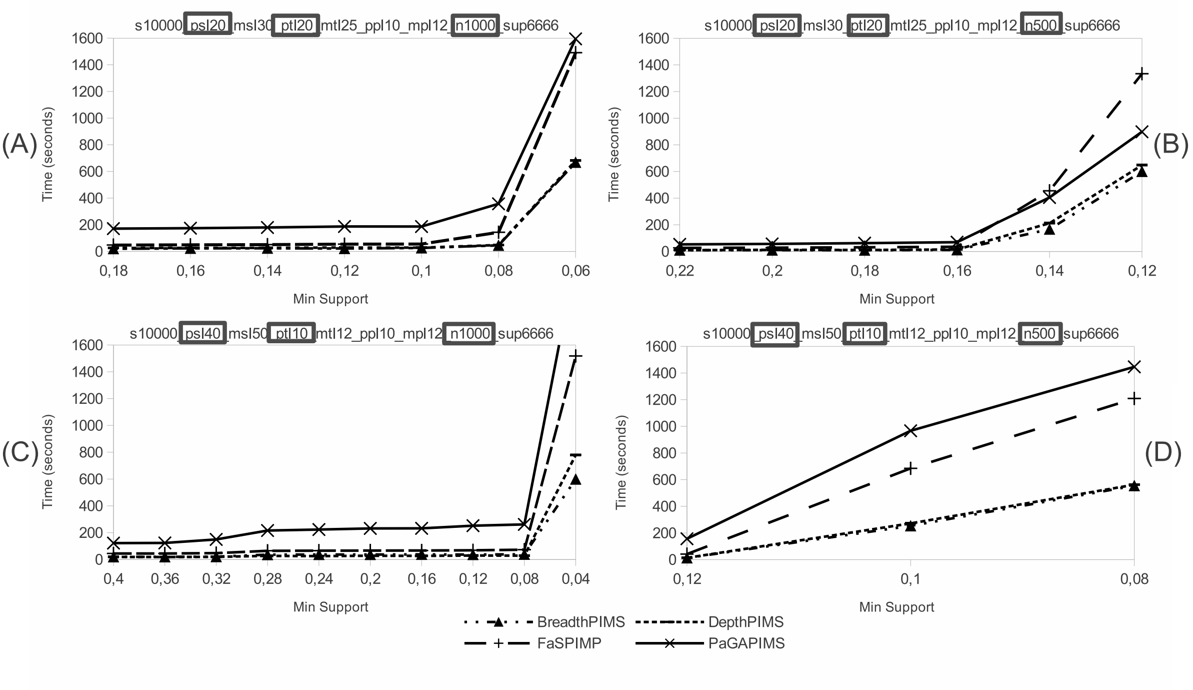


Figure 8.6: Varying support for datasets $s10000\_psl40\_msl50\_ptl10$–20_mtl12–25_ppl10 $\_mpl12\_n500$–1000.

Finally, as occurred in the previous Section, we show the behaviour when we change those properties that particularly affect the algorithms' execution. As before, we show the behaviour when we progressively change one of the parameters that affect the density, the number of items in our case, and also what occurs when we change the "transactions per sequence" parameter. Figure 8.7 shows what happens when we have a database with moderated parameters (1000 sequences, an average of 20 transaction per sequence, and 12 items per transaction, the length of the patterns in the database being an average of 12 items) and we vary the number of different items (50, 100 and 200 in plots 8.7A, 8.7B and 8.7C). All the plots show that BreadthPIMS and DepthPIMS are clearly superior to the other two algorithms (FaSPIMP and PaGAPIMS). The curves of BreadthPIMS and DepthPIMS are almost equal, signifying that they deal with more or less the same number of candidates. However, the behaviour of PaGAPIMS and FaSPIMP changes between the different plots. As stated in the previous section, boundary point representation is that least able to deal with the change of density in the databases.

Figure 8.8 similarly shows the behaviour with a similar database, but when considering

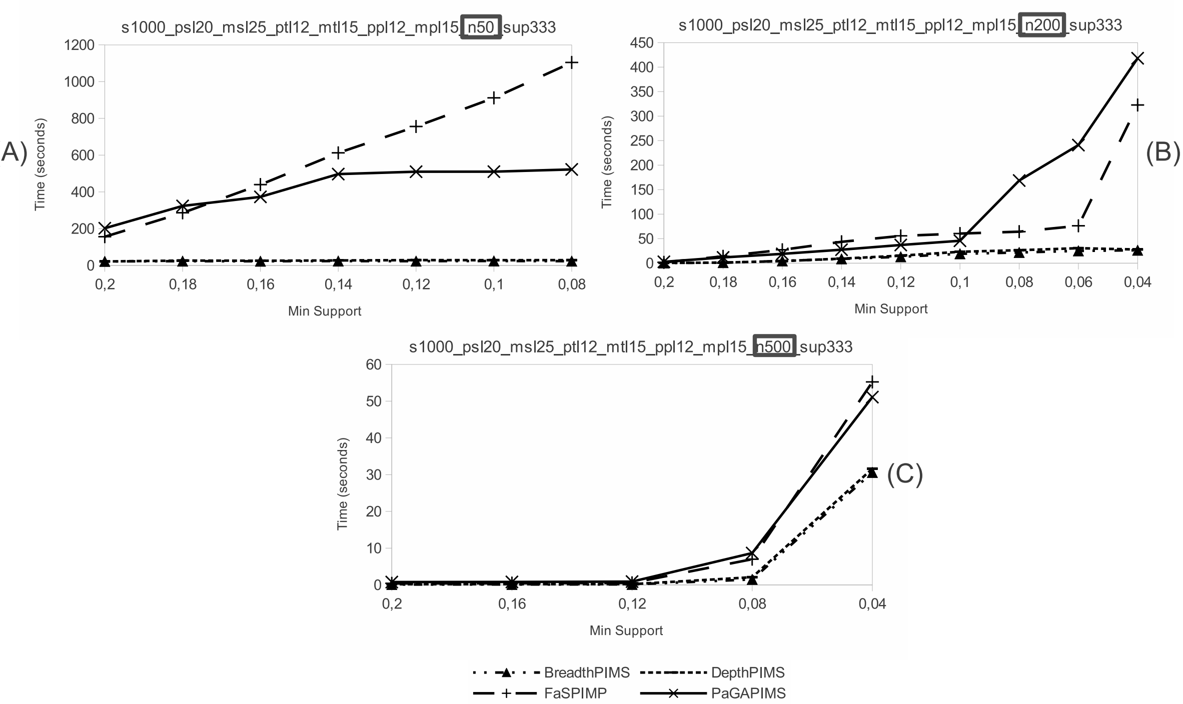


Figure 8.7: Varying support for a same configuration of datasets where we change the number of items (50, 200 and 500).

100 different items per database and varying the "transactions per sequence" value (40, 80 and 160 in plots 8.8A, 8.8B and 8.8C, respectively). As in Figure 8.7, all the plots show that both BreadthPIMS and DepthPIMS are the fastest, with an almost identical execution time. As before, the behaviour of PaGAPIMS and FaSPIMP changes throughout the different databases. In Plot 8.8A FaSPIMP, has a good execution but there is suddenly a great increase in its execution for support 0.04. It will be noted that this change also occurs with PaGAPIMS, but to a lesser extent. This great change is mainly owing to the boundary point representation and it is particularly severe in FaSPIMP because of the step in which the frequent 2-patterns are mined. Nevertheless, in Plot 8.8B FaSPIMP's behaviour is quite a lot better than that of PaGAPIMS, whereas in Plot 8.8C both execution times are very similar and much higher than the BreadthPIMS and DepthPIMS execution times.

In general terms, it will be observed that, when dealing with quantitative patterns, it is easy to attain the lower supports than in the qualitative case because of the difficulty of finding a pattern with identical temporal distances and item durations. Furthermore, when we mine quantitative patterns, we very often find low execution times that suddenly change to very high times. This phenomenon is owing to the explosion of quantitative patterns, which will be extensively studied in Chapter 9.

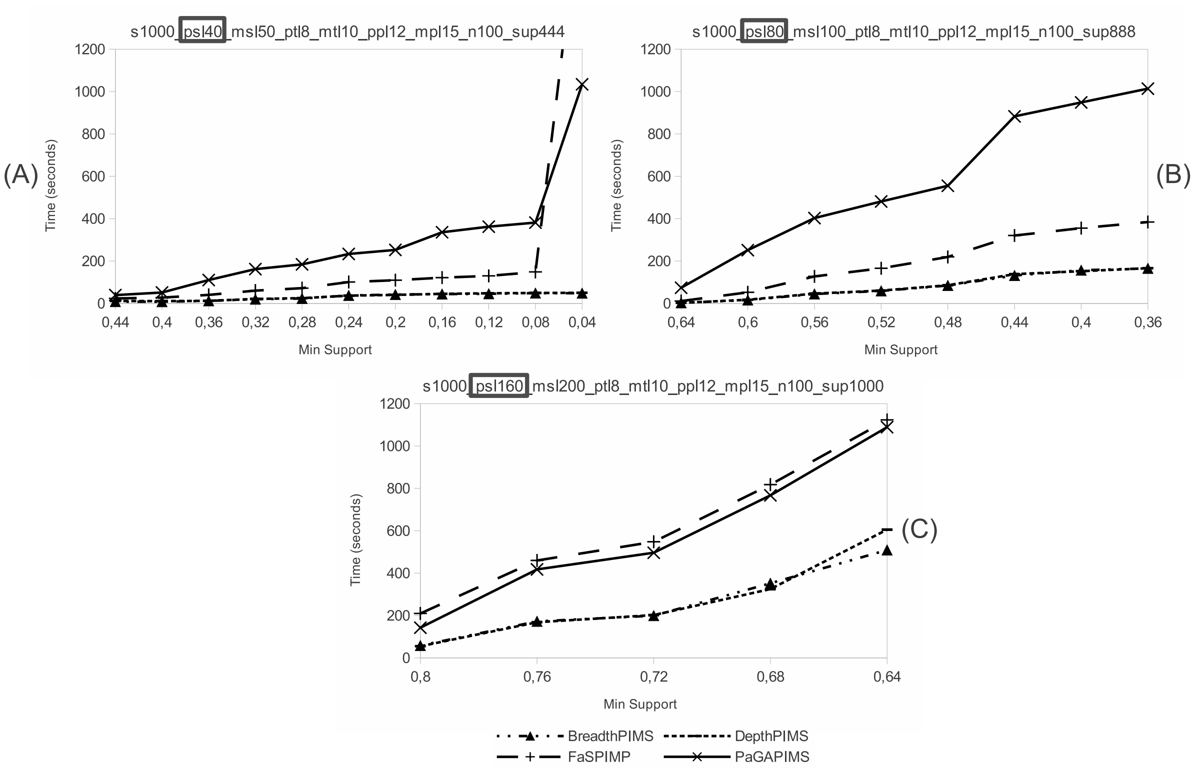


Figure 8.8: Varying support for a same configuration of datasets where we change the number of items per itemset (40, 80 and 160).

Chapter 9

Discussion

This Chapter provides a general discussion on the advantages and drawbacks of the various alternatives for the design of an algorithm for SDM. We first show the benefits and disadvantages of FaSPIP and PaGAPIS when compared to BreadthPIS and DepthPIS. Then we discuss the convenience of using the boundary point representation, used in PaGAPIS and FaSPIP (and PaGAPIMS and FaSPIMP), or the triangular matrix (Trimax), as it is used in BreadthPIS and DepthPIS (and their quantitative versions BreadthPIMS and DepthPIS). We subsequently show an in-depth exploration regarding the difference in the nature of the algorithms that discover qualitative patterns and quantitative patterns, explaining the main problems that commonly appear. We next show the advantages and drawbacks of the different search methods and provide some criteria as to their use in an efficient algorithm. Finally, we discuss the advantages of using a Vertical Database Format algorithm in order to find the closed frequent set, as occurs with a ClaSP algorithm.

9.1 Intra-comparisons. FaSPIP vs PaGAPIS. Breadth vs DepthPIS

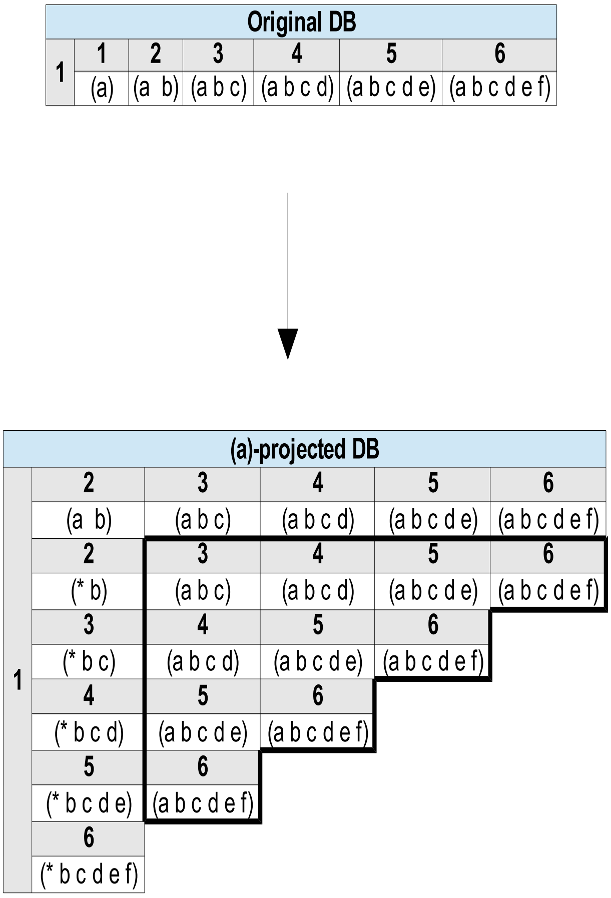
9.1.1 FaSPIP vs PaGAPIS

In this Subsection we discuss the main advantages of FaSPIP with regard to PaGAPIS. In the first place, let us recall the main differences between the original versions of the Vertical Database Format and Pattern Growth algorithms on which FaSPIP and PaGAPIS are respectively based. We shall later show the behaviour of these algorithms once all the necessary information with which to simultaneously manage points and intervals and the pruning methods has been included, in order to verify that all the patterns contain proper intervals.

As explained in Chapter 3, one of the situations overlooked in some algorithms such as PrefixSpan, is the presence of several appearances of the same item or event type in a sequence. On the one hand, in PrefixSpan it is necessary to make all the projections of an item that appears several times in a sequence in order to guarantee that all the I-extensions associated with that item will be discovered. If, conversely, it was not necessary to find the I-extensions, or if there were only one item per itemset, or only one appearance of each item per sequence existed, there would be sufficient information with a single projection.

On the other hand, in Vertical Database Format algorithms it is not necessary to take into account the problem of PrefixSpan, and all the S-extension and I-extensions explore the whole search space. It could thus be said that the latter is less influenced by the database structure.

For example, let us view the behaviour of both PrefixSpan and SPADE in the case of the database composed of the single sequence $s = \langle (a)_1 (ab)_2 (abc)_3 (abcd)_4 (abcde)_5 (abcdef)_6 \rangle$ and a $min\_\ supp = 1$. In this case, there are six itemsets, occurring between times 1 and 6. If we consider sequence $\langle (a) \rangle$, the six different projections associated with this sequence are shown in Figure 9.1. Note that if only the first projection is considered in our example, then items $*b$, $*c$, $*d$, $*e$ and $*f$ will not be considered to be frequent items, and in order to count these items it is necessary to take into account the first itemset of every subsequent projection. PrefixSpan must necessarily scan all the projections but, in the case of those projections after the first one, PrefixSpan must take into account only the first itemset and can ignore the remaining itemsets. Figure 9.1 highlights all the itemsets that can be ignored by PrefixSpan.



Original DB

1	1	2	3	4	5	6
	(a)	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)

(a)-projected DB

1	2	3	4	5	6
	(a b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	2	3	4	5	6
	(* b)	(a b c)	(a b c d)	(a b c d e)	(a b c d e f)
	3	4	5	6	
	(* b c)	(a b c d)	(a b c d e)	(a b c d e f)	
	4	5	6		
	(* b c d)	(a b c d e)	(a b c d e f)		
	5	6			
	(* b c d e)	(a b c d e f)			
	6				
	(* b c d e f)				

Figure 9.1: Projected database for the brief example with the standard PrefixSpan algorithm.

With regard to SPADE, and using the same example as above, this algorithm builds the associated IdLists (shown in Figure 9.2) without any special consideration. This point is one of the great advantages of the Vertical Database Format strategy over the Pattern

Growth strategy for certain database configurations.

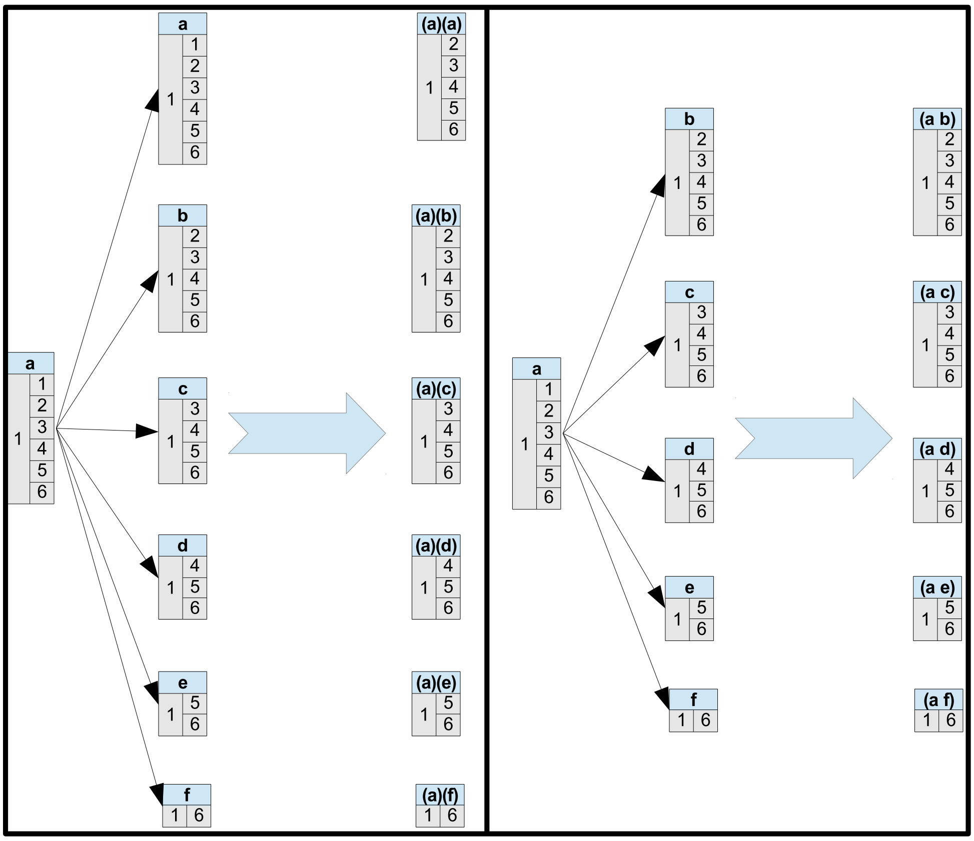


Figure 9.2: SPADE IdList for the brief example.

With regard to FaSPIP and PaGAPIS, the above characteristics also define the principal behaviour of both algorithms. However, as was shown in Chapter 3, both algorithms have new changes with respect to the original SPADE and PrefixSpan algorithms in order to guarantee that all the boundary sequences are well-formed. Concretely, even when FaSPIP uses two new pruning mechanisms, the principal behaviour remains. However, the PaGAPIS algorithm has other new drawbacks, and while in the simple version of the original PrefixSpan it is possible to avoid exploring the part highlighted in Figure 9.1, since these relations are already taken into account in the first projection (relation before), PaGAPIS needs to explore them in case the event can be repeated in a sequence. When we discover intervals, it is not now possible to ignore that highlighted part because this would signify overlooking certain temporal relations, such as "meets", "overlaps", "contains", "starts", "is finished by" or "equals".

For instance, let us suppose that the database is formed only of the sequence $\langle(a^+)(a^-)(a^+)(b^+)(c^+)(b^-)(c^-)(a^-)\rangle$ and $min\_supp = 1$. If we project this using the 1-sequence (a^+), in Figure 9.3 we can find the two projections associated with (a^+). If this example were to be processed, as in the PrefixSpan algorithm, we would take into account all the itemsets in the first projection and only the first itemset in the second projection. We would not therefore find the patterns $\langle(a^+)(b^+)(b^-)(a^-)\rangle$, $\langle(a^+)(c^+)(c^-)(a^-)\rangle$ and $\langle(a^+)(b^+)(c^+)(b^-)(c^-)(a^-)\rangle$, since in the first projection the elements are discarded after the itemset at time $t = 2$, which is precisely the moment at which the interval a has finished. In order to find the complete set of frequent boundary sequences, it is thus necessary to carry out a complete analysis of every itemset of every projection. This necessity is a new drawback for PaGAPIS as regards the original PrefixSpan that has to be introduced in order to mine proper intervals.

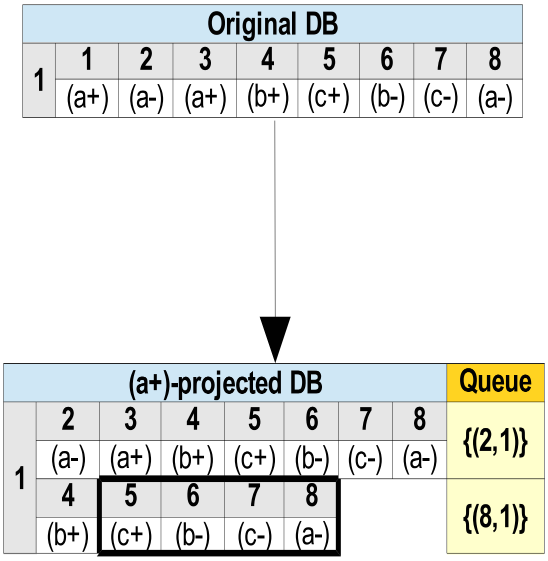


Figure 9.3: Projected database for the brief example with the PaGAPIS algorithm.

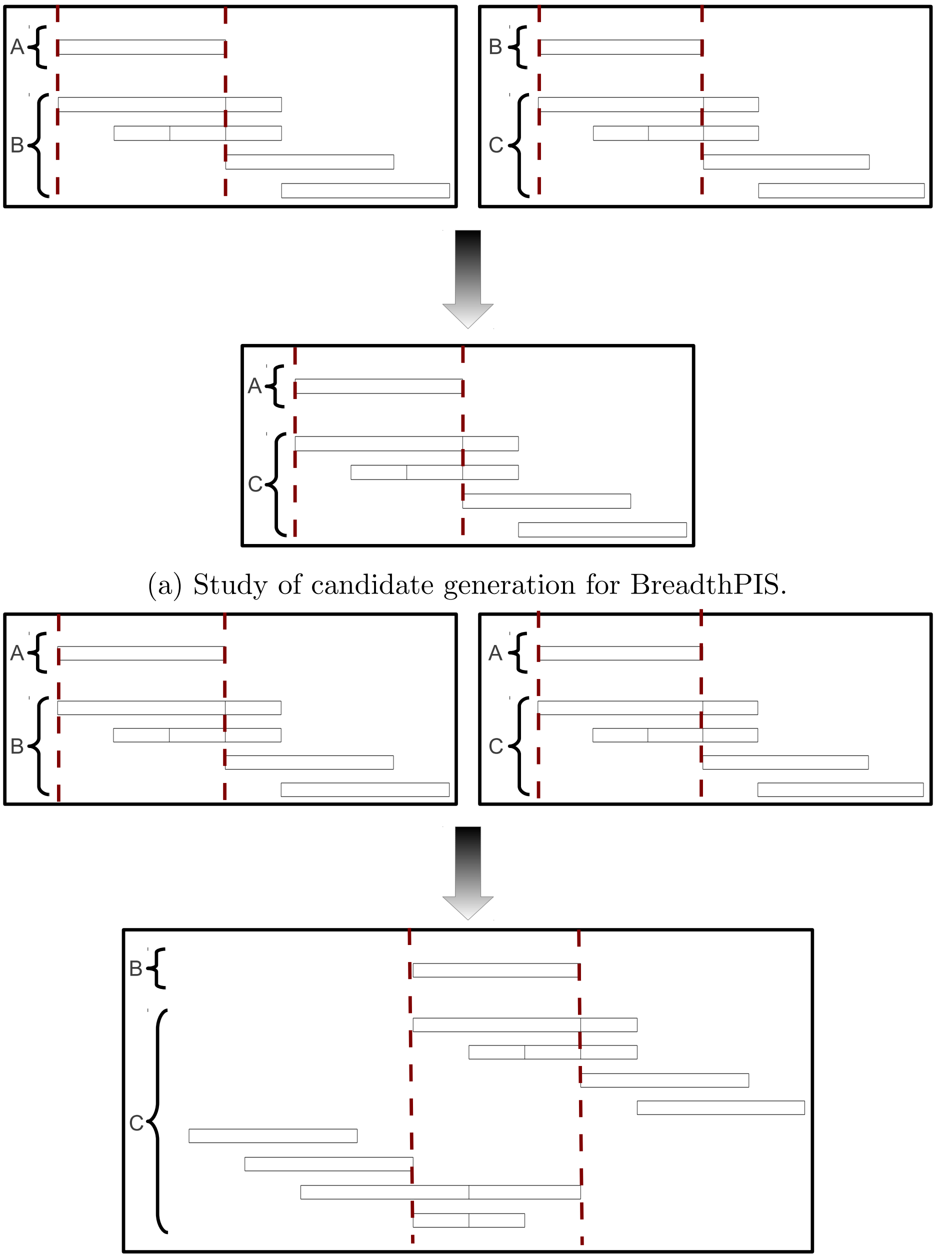
All the characteristics mentioned here are also present in their quantitative versions, the FaSPIMP and PaGAPIMS algorithms (for further details see Section 4.6).

9.1.2 BreadthPIS vs DepthPIS

In this Subsection we discuss the main advantages of DepthPIS with regard to BreadthPIS. Please recall that both algorithms use temporal reasoning to infer the temporal relations in the patterns. They differ in the structure of the transition function and the search method use. We shall now compare these two points.

In Sections 5.2 and 5.3 we saw the corresponding transition tables (Table 5.2 and Table 5.3, respectively), associated with the transition functions of BreadthPIS and DepthPIS, $f_{\mathcal{BFS}}$ and $f_{\mathcal{DFS}}$. If both tables are viewed, it will be noted that, in general, the sets in $f_{\mathcal{DFS}}$ have a higher cardinality than those in $f_{\mathcal{BFS}}$.

This is owing to the nature of the candidate generation method. If we have three events A, B and C and we know R_{AB} and R_{BC}, BreadthPIS obtains the possible relations R_{AC} by means of $f_{\mathcal{BFS}}$. Conversely, in DepthPIS we know the relations R_{AB} and R_{AC}, and $f_{\mathcal{DFS}}$ obtains all the R_{BC} relations that can exist. This means of inferring candidates and how the sequences are arranged are the keys to understand the difference between their transition functions. We shall explain the functioning of both methods by means of Figures 9.4a and 9.4b. Figure 9.4a shows the candidate generation for the BreadthPIS algorithm. It will be observed that, from the three events (A, B and C), the number of relations formed by candidate generation is bounded since the beginning of A occurs at the same time or before the beginning of C. On the contrary, Figure 9.4b shows that when DepthPIS algorithm infers the relation R_{BC}, any relation is possible between events B and C.



(a) Study of candidate generation for BreadthPIS.

(b) Study of candidate generation for DepthPIS.

Figure 9.4: Study of candidate generation.

An analysis of all the set of relations resulting from both transition functions $f_{\mathcal{BFS}}$ and $f_{\mathcal{DFS}}$ leads to certain ideas. For example, if we add all the possible relations obtained for the 49 combinations of the transition function we obtain 75 relations for $f_{\mathcal{BFS}}$ and 143 for Table $f_{\mathcal{DFS}}$. These values provide an average cardinality of 1.53 relations for $f_{\mathcal{BFS}}$ and 2.91 for $f_{\mathcal{DFS}}$. These two numbers allow us to see that, on average, the number of candidates generated by DepthPIS is twice as large as the number of candidates produced by BreadthPIS. Moreover, if we study the average cardinality of the set of relations when we know one of the two relation arguments R_r or R_c of the transition functions $f_{\mathcal{BFS}}(R_r, R_c)$ and $f_{\mathcal{DFS}}(R_r, R_c)$, the result provides some interesting information. The top of Table 9.1 shows these average values when we know the first argument R_r, while the bottom part shows the average when we know the second argument R_c. On the one hand, in the first table it will be noted that except for the *equal* relation, $f_{\mathcal{DFS}}$ obtains bigger sets than $f_{\mathcal{BFS}}$, and there are remarkable differences between relations $<$, o and c, in which there is a factor from almost two up to more than four from $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$. On the other hand, the second table also shows significant differences between the number of relations provided by $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$ (except for the *equal* relation). In particular, the most significant differences appear in relations $<$, o, c and s, which have a factor from almost two up to almost three relations.

BreadthPIS	
$<$	1
m	1
o	2.14
f^{-1}	1
c	2.43
$=$	1
s	2.14

DepthPIS	
$<$	4.43
m	2.43
o	4.14
f^{-1}	2.14
c	3.86
$=$	1
s	2.43

	$<$	m	o	f^{-1}	c	$=$	s
BreadthPIS	1.57	1.28	1.86	1.57	2.14	1	1.28
	$<$	m	o	f^{-1}	c	$=$	s
DepthPIS	4.43	2.43	4.14	2.14	3.86	1	2.43

Table 9.1: Average number of relations when we know the first (top table) or the second (bottom table) argument of the transition function.

Note that "m", "s", "f^{-1}" and " $=$ " are the least frequent relations that usually appear in patterns because the events that fulfil these relations need to have at least an equal relation between their boundary points. The main difference provided by $f_{\mathcal{DFS}}$ with regard to $f_{\mathcal{BFS}}$ therefore implies that the candidate generation of BreadthPIS is quite a lot more efficient than that provided by DepthPIS, since the minor differences in their averages are in the least frequent relations. What is more, the large sets of relations provided by $f_{\mathcal{DFS}}$ lead to the generation of several candidates in DepthPIS that will not eventually be frequent.

Another relevant difference between both algorithms is the search method used. As stated previously, BreadthPIS follows a breadth-first search strategy whereas DepthPIS carries out a depth-first search. The main problem caused by a breadth-first search is the need to maintain all the frequent patterns discovered in each level in the memory in order to be able to generate the set of candidate patterns. Although a breadth-first search normally makes the use of a pruning method possible, as was pointed in the methods of BreadthPIS in Section 5.2, with the tests that were executed it was not worth the effort of enabling such a prune. This is mainly because it is possible to quickly compute the support of a pattern. Instead, DepthPIS does not need to maintain all the k-patterns discovered in the memory, since it only needs the members of an equivalence class that are extended by only one event.

Finally, thanks to the way in which DepthPIS is executed and owing to its depth-first search, it is possible to split the search for frequent patterns up into different separate parts. To do this it is necessary to explore each equivalence class independently and maintain the set of frequent patterns associated with it. It is eventually only necessary to join all the intermediate sets of frequent patterns in the final frequent pattern set. This therefore makes it possible to parallelize the search of all the different classes in order to obtain faster results. Figure 9.5 shows a graphical schema of the separation of the different equivalence classes for the example database. In the figure, each equivalence class surrounds all the patterns that are contained in it.

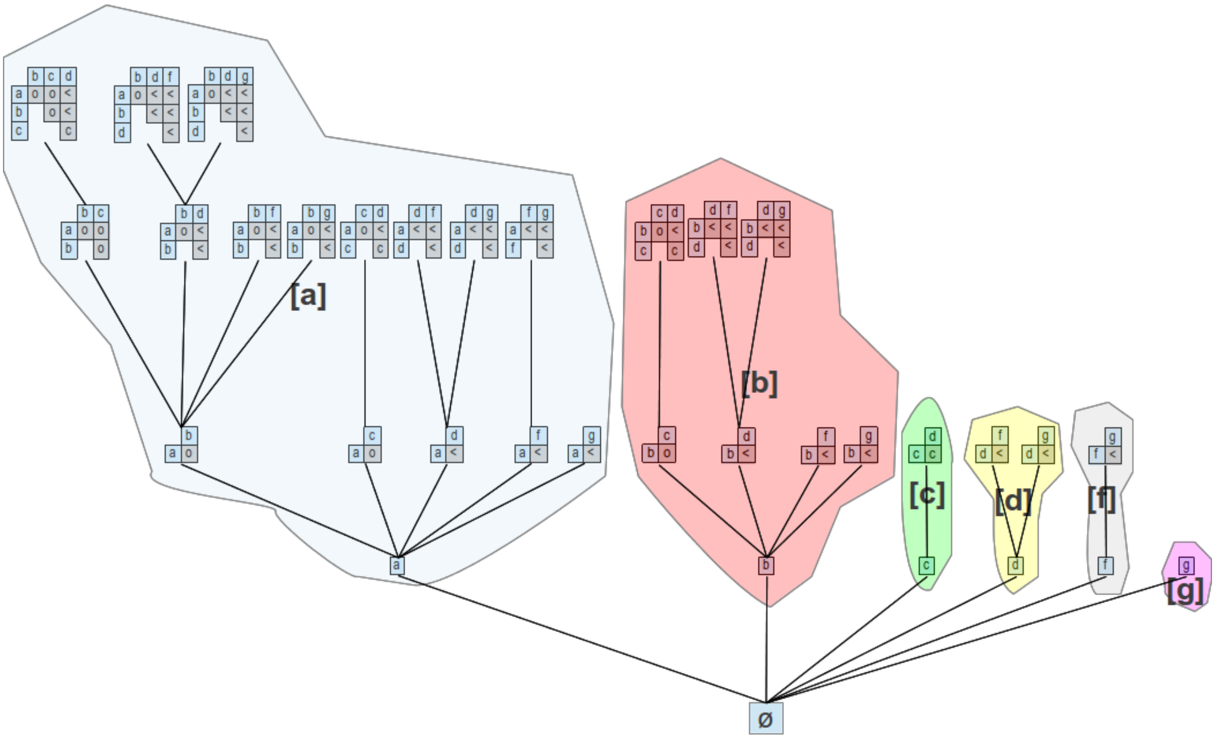


Figure 9.5: Division of the different equivalence class in independent problems.

9.2 Inter-comparisons. Boundary points vs triangular matrices representation

In the PaGAPIS and FaSPIP algorithms (and in those of PaGAPIMS and FaSPIMP) we have used a representation of patterns based on boundary points, while in BreadthPIS and DepthPIS (and BreadthPIMS and DepthPIMS) we have used a Triangular Matrix representation. These two representations are compared in this subsection.

On the one hand, we have identified the following advantages as regards boundary points:

- It is a more compact representation than any other, since we can code a sequence of k intervals with $2k$ boundary points, and thus only need $2k-1$ relations between the boundary points to obtain a non-ambiguous representation. Moreover, the relations between points are only "<" and "=". All the other attempts that have been developed to date, usually have a more complicated structure and, concretely, when we represent an interval sequence with a TriMax, we need $\frac{k(k-1)}{2}$ relations.

- It is a very simple, intuitive and readable representation since it is possible to see all the boundary points for intervals and the temporal order between them at a glance.

- This representation makes it possible to use the standard point-based algorithms directly for SDM.

On the other hand, we have identified the following drawbacks:

- Although direct point-based algorithms can be used, they have to be adapted in order to mine proper intervals. This adaptation is not always very simple and it has a high cost in terms of constraint checking. More specifically, it is necessary to add some new methods to check whether an end boundary point is ending an open interval properly. This new method makes the PaGAPIS and FaSPIP (and hence PaGAPIMS and FaSPIMP) algorithms slower.

With regard to the TriMax representation, we have identified the following benefits:

- Since we use $\frac{k(k-1)}{2}$ relations between the k intervals, it is very easy to extract a general view of the global placement and relations for all the intervals.

- As we have all the explicit relations between the intervals, we have a very powerful tool with which to create a temporal reasoning from two triangular matrices that correspond to two patterns with a common subpattern. This enables a very suitable candidate to be generated from the k-patterns, rather than extending each pattern with a new 1-pattern.

- Since an item is considered as a whole, we can include its duration in it. This is particularly interesting when dealing with algorithms for the mining of quantitative patterns as we avoid several calculations.

Conversely, we have found the following drawbacks:

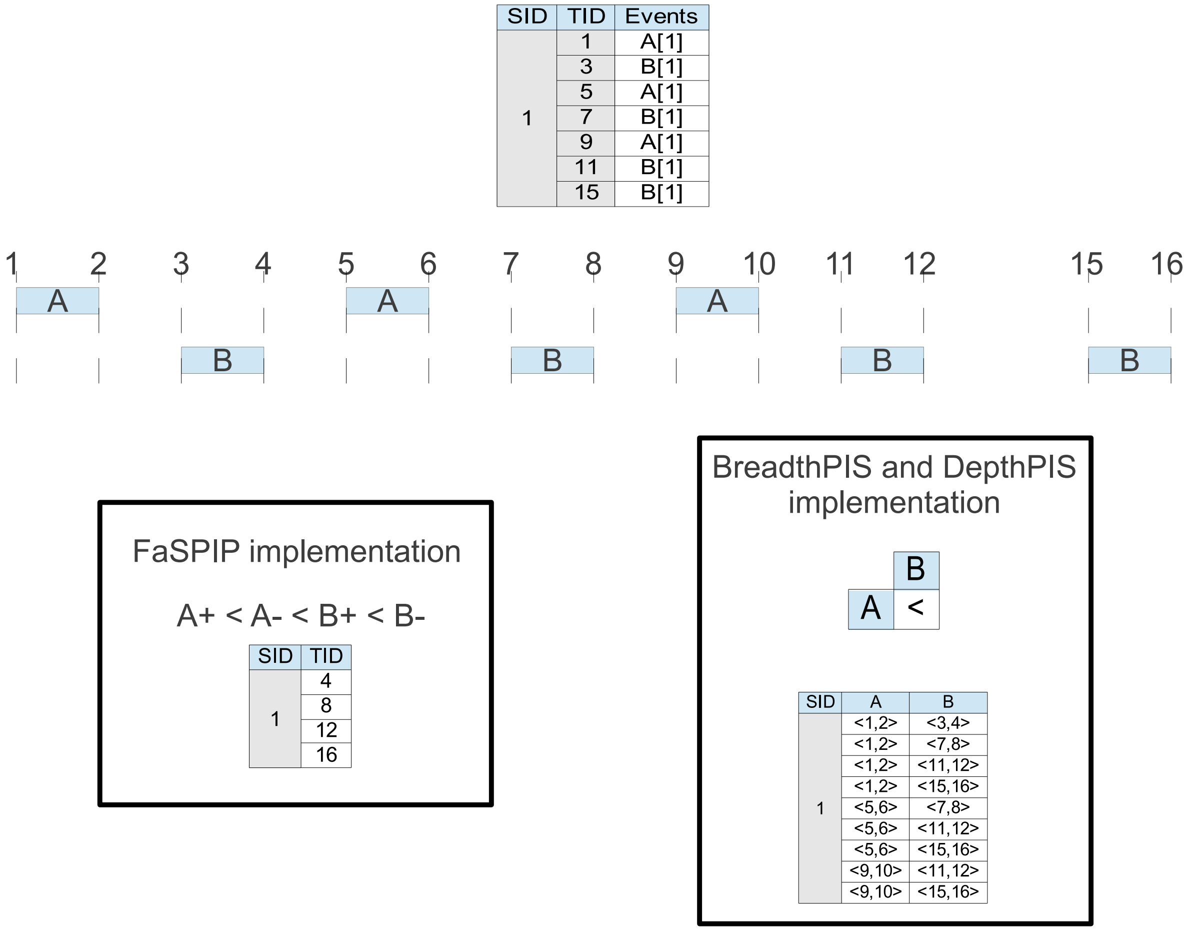
- The representation is expensive as regards space, since we need $\frac{k(k-1)}{2}$ relations to represent a pattern of k intervals.

- Since all the relations are maintained, it is also necessary to maintain all the appearances of every interval rather than registering only the last one (as occurred in the original point-based algorithm). This provokes some redundancy when the same pattern is found several times in the same sequences, and the scalability of the algorithm is thus affected.

Most of the algorithms in SDM normally have two phases: candidate generation and support counting. With regard to the candidate generation, algorithms based on boundary points progressively create new candidates by simply adding points. Therefore, apart from the normal structures associated with their basic algorithms, they need a second structure to be able to check the correctness of the candidates. This signifies that whenever a pattern with an interval that is badly formed appears it is necessary to rule it out in order to maintain only those patterns that have a correct structure. It is also necessary to carry out an initial counting of the appearances of the chosen candidate in order to create the correct candidate extension set for future candidates. Of course, this process has a cost in execution time, whereas in the algorithms that we have developed on the basis of TriMax representation, the candidate generation phase hardly has any cost. In this second alternative, since we consider an event as a whole and we do not make any distinction between point and intervals, all the possible candidates are created directly from the corresponding transition function with an insignificant cost in terms of execution time.

Additionally, when we focus on the Vertical Database Format algorithms, we also find some differences between FaSPIP (and FaSPIMP) and both BreadthPIS and DepthPIS (and their quantitative versions BreadthPIMS and DepthPIMS). While the former algorithm needs to have a queue to make certain that an occurrence of the candidate pattern that is considered will be found, the TriMax structure makes it possible to carry out a join operation without any additional structure. However, a noticeable advantage that FaSPIP (and FaSPIMP) has in comparison to BreadthPIS or DepthPIS (and their quantitative versions BreadthPIMS and DepthPIMS), is that, while FaSPIP only has to maintain the occurrence of the last event of the patterns in its IdList, i.e., the last boundary point, BreadthPIS and DepthPIS need to maintain all the occurrences of all the events that appear in the patterns. This difference is owing to the fact that in FaSPIP the join operation, through a queue, checks whether a concrete pattern appears in a sequence, and only two temporal relations, before and equals, are involved. Furthermore, it is possible to retain the occurrence of all the elements of the patterns with only the last element, whereas BreadthPIS and DepthPIS need to maintain each appearance of all the events of a pattern because we are working with thirteen interval relations and it is necessary to check whether the relation that a new event has with the remaining events that appear before in the pattern is that which was expected. This issue leads us to have, on average, more entries in the IdList and, therefore, a slower IdList computation. Figure 9.6 shows one difference between both ways of representing the IdList. The Figure depicts the representation of the pattern $A < B$ for only one sequence with the boundary points and TriMax representation. It will be observed that in the boundary point case, four entries are needed to code all the appearances of the pattern in the sequence, all of these being appearances of the timestamps where the end point of B occurs. However, in the case of BreadthPIS and DepthPIS nine entries are needed to code the same appearances. This is owing to the need to maintain all the occurrences of both events A and B. To sum up, in FaSPIP we have only the last event, and several entries are summarised by an appearance, whereas BreadthPIS and DepthPIS need to create all the different entries because all the event appearances are indicated in the IdList. This last effect may be an important problem when working with dense databases with large itemsets in which the same relation can be found between several itemsets.

In general terms, we can say that when we confront qualitative databases, a boundary representation is more appropriated and, in particular, FaSPIP shows a better behaviour than PaGAPIS, having a faster execution time.



SID	TID	Events
1	1	A[1]
	3	B[1]
	5	A[1]
	7	B[1]
	9	A[1]
	11	B[1]
	15	B[1]

SID	TID
1	4
	8
	12
	16

SID	A	B
1	<1,2>	<3,4>
	<1,2>	<7,8>
	<1,2>	<11,12>
	<1,2>	<15,16>
	<5,6>	<7,8>
	<5,6>	<11,12>
	<5,6>	<15,16>
	<9,10>	<11,12>
	<9,10>	<15,16>

Figure 9.6: Comparison between the different IdList implementation for boundary point representation and TriMax algorithms.

With regard to the memory consumption, the representation of both patterns and the IdList is more costly for BreadthPIS and DepthPIS than for FaSPIP and PaGAPIS since a TriMax representation and an IdList for the two former algorithms necessitate the storage of more information. However, when the DepthPIS algorithm is used, this difference is insignificant since it uses a depth-first search, and it is not therefore necessary to maintain the patterns previously found in the memory.

Finally, the DepthPIS algorithm enables a parallelization of the mining task since it is possible to resolve several equivalence classes independently, as is shown in Subsection 9.1.2. This possibility is particularly useful when working with large databases that cannot fit in the memory, and which can thus be split into different pieces.

9.3 Problems in the mining of quantitative patterns

All the points previously discussed for qualitative mining are still valid in the quantitative approach. Nevertheless, when searching for quantitative patterns there are additional problems that must be taken into account. These problems are intrinsic to the nature of the task of mining quantitative patterns and cannot be avoided by any algorithm. We particularly highlight two main problems: 1) the difficulty of finding pattern occurrences since both items with durations and relations with temporal distances must be exactly equal; and 2) the explosion of quantitative patterns that appear when we mine with a very low support.

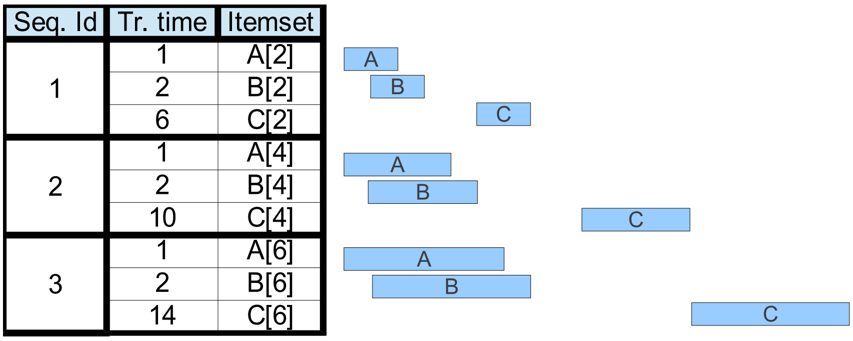
With regard to the first problem, as stated in Chapter 8, this occurs because in qualitative pattern mining the focus is solely on items and relations, and the item durations and the temporal distances associated with relations are ignored, whereas when we mine quantitative patterns we are interested in everything. This signifies that if it is necessary to search for pattern occurrences with high support in which events, item durations, relations and their associated temporal distances must be equal, then this task becomes much more complex.

Furthermore, the aforementioned reason is also linked to the second problem. Since it is very hard to find several occurrences for a pattern, in most cases this occurrence number is less than the *min_sup* value, and it is not therefore possible to find a high number of patterns. However, as very low supports are attained, a large amount of quantitative patterns, which were previously infrequent, become frequent.

Let us see an example in which both problems can be observed. Figure 9.7 shows an input database with three sequences, and three items: A, B and C. What is more, the same item changes its duration in every sequence and the transaction times vary throughout the different sequences. The first problem appears if, for example, we consider a *min_sup* value of 3 sequences, i.e., the patterns that appear in 100% of sequences. In this case we find 7 qualitative patterns, whereas there is no quantitative pattern. If we continue with the same example database from Figure 9.7, we can observe the explosion of quantitative patterns if we consider a *min_sup* value of 1 sequence, i.e., the patterns that appear in 33% of sequences. All the frequent patterns, both qualitative and quantitative, can be seen in Figure 9.8. We can therefore verify that for the qualitative approach there are still 7 frequent patterns, whereas for the quantitative approach there are now 21 frequent patterns.

Finally, we have yet to comment on two more problems associated with the quantitative approach that are dependent of the algorithm implementation. These two problems are linked to the boundary point representation and the Vertical Database Format.

With regard to boundary point representation, the problem that emerges is related to the item durations and, since in this representation we only take points into account, a temporal distance between two points can be associated with either an item duration or with a temporal distance. This forces us to always check the different distances in their join operation that is related to their IdList, regardless of whether it is a temporal distance or an item duration. On the contrary, when we use a pattern representation that considers an item duration as a part of the item, we do not need any special operation to check the equality of item durations. Therefore, in our four algorithms for the mining of quantitative patterns in point and interval databases, those that use a Triangular Matrix



Seq. Id	Tr. time	Itemset
1	1	A[2]
	2	B[2]
	6	C[2]
2	1	A[4]
	2	B[4]
	10	C[4]
3	1	A[6]
	2	B[6]
	14	C[6]

Figure 9.7: Example database for showing the problems that appear with quantitative pattern mining.

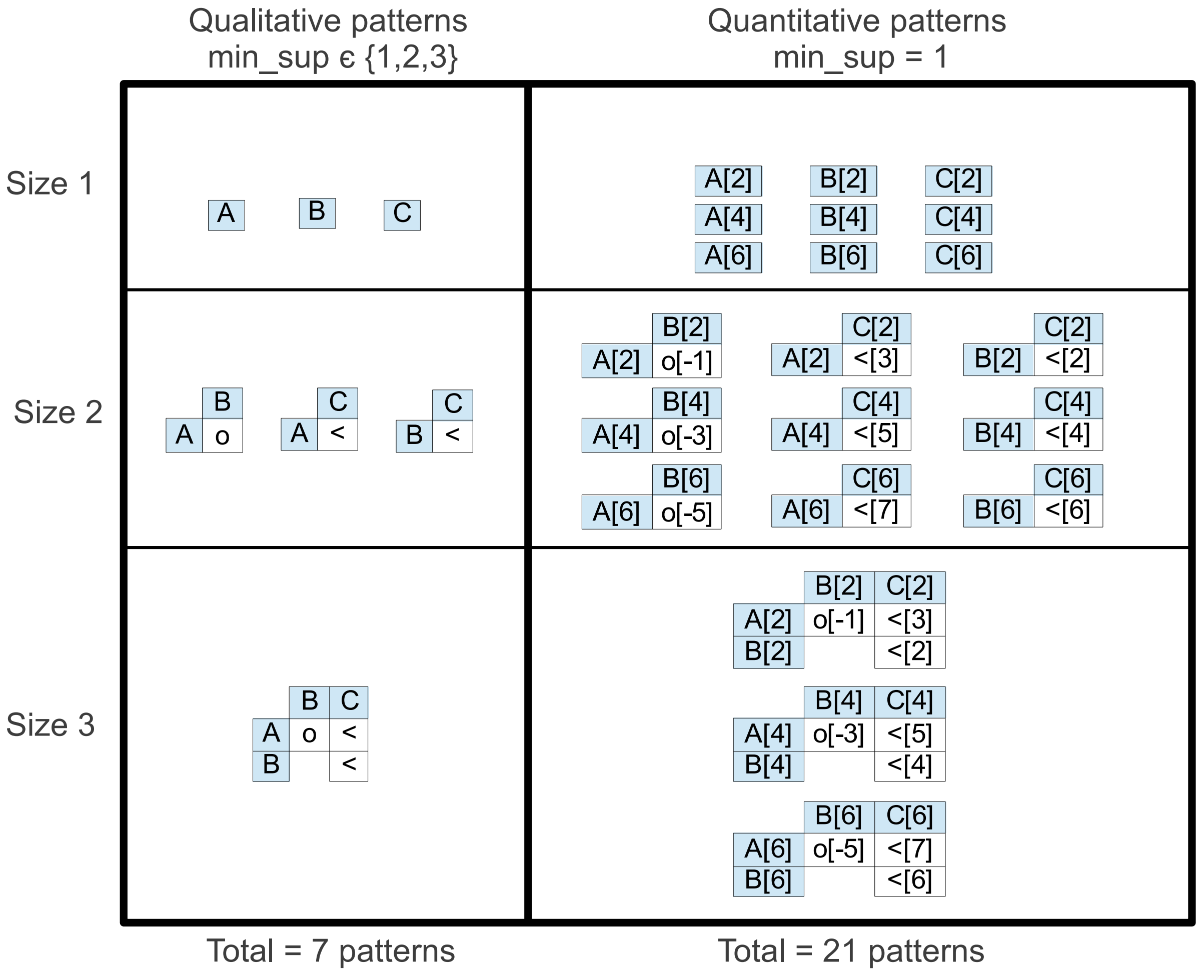


Figure 9.8: Frequent qualitative and quantitative patterns for the example database shown in Figure 9.7.

representation have a great advantage over those that use a boundary point representation.

With regard to Vertical Database Format algorithms, since these algorithms build candidates from frequent patterns, it is necessary to obtain all the temporal distances that are associated with the relations during the early stages of the algorithms. It is therefore necessary to extract all the frequent 2-patterns in order to obtain all the temporal distances, and larger patterns are then built from all the possible temporal relations. This problem is also associated with the Apriori-like strategy, since it also creates candidates. It does not, however, appear with the Pattern Growth strategy since its algorithms obtain both the item and the relation with the temporal distance when they scan each database projection. In order to alleviate this last problem, we have, in Chapters 4 and 6 proposed the execution of a Pattern Growth algorithm to find the frequent quantitative 2-patterns, and this set of patterns is then used to build larger patterns like a Vertical Database Format algorithm usually does.

After describing these two problems which are dependent on the algorithms' implementation, note that FaSPIMP has both drawbacks as it is a Vertical Database Format algorithm and it is based on a boundary point representation.

9.4 Search strategies: depth-first vs breadth-first vs mix (equivalence classes)

In this Subsection we analyse the conveniences and inconveniences of using the different search strategies. In normal terms, a breadth-first search is associated with an Apriori-style strategy, and sometimes with the Vertical Database Format strategy, whereas a depth-first search is associated with a Pattern Growth strategy and also with a Vertical Database Format. A third alternative that is also possible is to use a mixture between depth and breadth search. The breadth-first search is used in BreadthPIS (and BreadthPIMS), the depth-first search is used in PaGAPIS and FaSPIP (and PaGAPIMS and FaSPIMP), and a mixture is used in DepthPIS (and DepthPIMS).

It is possible to state that, in general, a breadth-first search works better in the candidate generation since all the patterns that have the same length are stored in the memory and the temporal reasoning is effective. Conversely, the problem that arises is that, for large databases, we cannot store all the patterns in the same level, signifying that the BFS algorithms cannot always achieve the low supports that a DFS algorithm can. On the other hand, a depth-first search has a more limited candidate generation since we do not have sufficient information to be able to carry out an effective prune as in BFS, but we obtain very efficient algorithms. Finally, a combined search is a very good option, such as that used in DepthPIS (and DepthPIMS). This search consists of BFS but only in the equivalence class that is being processed at a particular moment, signifying that it is very efficient in terms of memory management.

9.5 Mining of frequent closed patterns with Vertical Database Format strategy.

To date, the algorithms that have most frequently been used to find closed patterns have been based on the Pattern Growth strategy and, more specifically, on the PrefixSpan algorithm [Pei et al., 2004]. In this work we have proposed the mining of the frequent closed set with Vertical Database Format strategy algorithms. Our motivation originates from the convenience of these latter kinds of algorithms when we have certain database configurations: dense database with large itemsets (see Chapter 7 for more details).

The important drawback of the Pattern Growth strategy, which is pointed out in Section 9.1.1, is that it also affects those algorithms which are used to mine closed patterns. It is therefore at this point that an important gap is observed into which we can introduce algorithms based on the Vertical Database Format strategy, thus avoiding the typical drawbacks of the Pattern Growth algorithms mentioned above, and profiting from the pruning methods. ClaSP therefore improves the results of the SPADE or SPAM algorithms, avoiding the generation of several parts of its search tree and providing a new possibility in closed mining algorithms.

Chapter 10

Conclusions

Finally, in this Chapter we conclude by showing our conclusions, the contributions that we have provided, and some ideas that may be developed in future work.

10.1 Conclusions

In this Thesis we have defined a general framework for SDM. This general framework has been used to address various algorithms that are capable of mining patterns with different types of representation (points, intervals or points and intervals) and expressiveness (qualitative or quantitative patterns).

We have also introduced a Survey in which we have shown all the strategies that are typically used to mine patters. This Survey has enabled us to organise all the previous works in this field, which has subsequently helped us to define the basis and the formalities for the SDM task.

We have studied this problem from several viewpoints. In the first place, we have carried out a study in order to discover which strategy, of those most frequently used (Vertical Database Format and Pattern Growth), is most appropriate for application in the mining of different levels of representation and expressiveness. In the second place, we have compared two different pattern representations, one based on boundary points and the other on Triangular Matrices in order to see which has the best benefits for our purposes.

In order to conduct this study, we have firstly developed a synthetic databases generator. Through this generator we have been able to establish the most desired parameters, such as number of different items, items per itemset, itemsets per sequences, number of sequences, pattern length or support, which have allowed us to make different experiments in details. We did not use real datasets, such as those cited in [Mörchen and Fradkin, 2010], since the algorithms' behaviour in these databases must also be analised in terms of the former parameters.

To this end, we have developed four algorithms with which to mine patterns with points and intervals or both at the same time. These algorithms are also capable of discovering qualitative and quantitative patterns. The algorithms for qualitative mining are called PaGAPIS, FaSPIP, BreadthPIS and DepthPIS, the first of which is based on the Pattern Growth strategy and the other three of which are based on the Vertical Database

Format Strategy. Their quantitative versions are PaGAPIMS, FaSPIMP, BreadthPIMS and DepthPIMS. Both PaGAPIS and FaSPIP (and therefore PaGAPIMS and FaSPIMP) implement a boundary point representation, whereas the representation of BreadthPIS and DepthPIS (and BreadthPIMS and Depth-PIMS) is based on Triangular Matrices. To the best of our knowledge, FaSPIP is the first Vertical Database Format algorithm to implement a boundary point representation, and DepthPIS is the first algorithm for the mining of intervals to be based on equivalence classes. Moreover, to date no works have applied a Vertical Database Format strategy to the mining of quantitative patterns, such as occurs in this thesis by means of FaSPIMP, BreadthPIMS and DepthPIMS. All of these algorithms bridge some of the gaps that were present in the state-of-the-art, as can be seen in Table 10.1, thus extending it and allowing us to solve these problems in an efficient manner.

We have performed a comprehensive set of tests in which the various properties that are associated with the database configurations were varied. The tests performed show that, in most cases, the Vertical Database Format strategy is the most appropriate as regards mining point and interval databases, particularly when dealing with dense databases with several Itemsets per sequence. A database is dense when the number of items per itemsets is close to the number of different items that there exist in the database. This situation has never been faced in previous comparatives or algorithms.

We have additionally discussed all the advantages and drawbacks of using a boundary point or a Triangular Matrix representation, the former being the best choice for qualitative patterns and the latter being the most appropriate for quantitative mining. Therefore, in general terms, the fastest algorithms that we have found are FaSPIP for qualitative patterns and BreadthPIMS and DepthPIMS for quantitative patterns.

We have also studied the difficulties of mining quantitative patterns and shown the intrinsic problems found. Moreover, we have explained the additional problems that arise if a boundary point representation or a Vertical Database Format algorithm are used. We have also provided a good solution in order to alleviate the disadvantage of mining the length 2 frequent quantitative patterns associated with Vertical Database Format algorithms. This study has allowed us to understand why FaSPIMP does not achieve good results for the tests concerning the mining of quantitative patterns and why BreadthPIMS and DepthPIMS are the best algorithms for use in discovering quantitative patterns.

We have used, through BreadthPIS and DepthPIS (and their quantitative versions BreadthPIMS and DepthPIMS), temporal reasoning in order to generate candidates. This temporal reasoning is highly efficient since only the candidates that are needed are generated in a quickly manner. Two different reasonings have been proposed, one for BreadthPIS and another one for DepthPIS, being the temporal reasoning of BreadthPIS the most optimized one since, on average, it generates a more reduced candidate set. To the best of our knowledge, few works in SDM have faced detailed temporal reasoning such as we have done in BreadthPIS and DepthPIS.

With regard to the search method used, we have studied which is best for mining point and interval databases. Although a breadth-first search method (used in BreadthPIS and BreadthPIMS) enables a pruning method to be used, along with enabling the design of a very effective transition table with which to generate candidates, this search method leads to memory overflow problems when confronted with large databases. The use of a Depth-first search method (used in DepthPIS and DepthPIMS) is therefore more suitable since it does not have that problem, although its temporal reasoning, and thus its candidate generation step, is not so effective as in BreadthPIS (and BreadthPIMS). What is more, the pruning methods that were added to BreadthPIS showed that there is no gain. Thus, in general, it is preferable to use FaSPIP for qualitative patterns and DepthPIMS for quantitative patterns, rather than BreadthPIS or BreadthPIMS.

Finally, we have studied the principles used to mine closed frequent sequential patterns, and we have also applied a Vertical Database Format algorithm to the mining of frequent closed patterns with points. The ClaSP algorithm uses a heuristic mechanism to prune non-closed patterns, and was inspired by the CloSpan algorithm. To the best of our knowledge, this is the first work based on the Vertical Database Format strategy to solve the proposed problem and it bridge some of the gaps that were present in the state-of-the-art, as can be seen in Table 10.2. In all our tests, ClaSP outperforms the state-of-the-art algorithms. We conclude that the Vertical Database Format strategy is highly suitable for the discovery of closed patterns.

10.2 Contributions

This Thesis makes the following contributions:

1. We have provided a comprehensive definition of Sequential Data mining, showing the different types of patterns that may be relevant for SDM.

2. We have provided a clear organisation of all the different interval pattern representations, the search methods typically used and the different strategies that have been introduced for Sequential Data Mining with this organisation. We reveal some open issues in SDM that need further work.

3. We have bridged some of these gaps by developing four algorithms for the mining of qualitative and quantitative patterns in point and interval databases. Each algorithm is capable of finding the first four types of pattern shown in Table 1.1 with a single implementation, and they work in any type of database, formed of either points, intervals or points with intervals.

4. We have developed an algorithm with which to mine qualitative closed point-based patterns based on the Vertical Database Format strategy. We have measured the impact of the pruning method in the reduction of both the output and the execution time.

5. We have provided a synthetic sequential database generator that is capable of creating databases with very different configurations. We have used these databases to evaluate our algorithm and to guide the discussions about the dimensions of the SDM problem.

6. We have already submitted our work related to Chapter 7. That work was published in the proceedings on Pacific-Asia Conference on Knowledge Discovery and Data Mining, held in Gold Coast, Australia in 2013.

Database	Representation	Distances	Strategies		
			Apriori	Pattern Growth	Vertical Database Format
Points		qualitative	Apriori-All, GSP, PSP, TSET	FreeSpan, PrefixSpan, Memisp	SPADE, SPAM, Prism
		quantitative	I-Apriori, Yoshida	MisTA, QprefixSpan, i-PrefixSpan	
Intervals	Points	quantitative		QTPrefixSpan	
		qualitative		T-PrefixSpan	
	Intervals	quantitative	QTempIntMiner	QTiPrefixSpan	
		qualitative	IEMiner, Karmalego	Armada	H-DFS
Points and Intervals	Points	quantitative	ASTPminer	**PaGAPIMS**	**FaSPIMP**
		qualitative	HTPM	CTMiner, CEMiner, **PaGAPIS**	**FaSPIP**
	Intervals	quantitative	**BreadthPIMS**		**DepthPIMS**
		qualitative	**BreadthPIS**		**DepthPIS**

Table 10.1: Classification of the different algorithms already developed for SDM with our proposed algorithms in bold.

Apriori	Pattern Growth	Vertical Database Format
	CloSpan, Bide	**ClaSP**

Table 10.2: Classification of the different point-based algorithms for closed patterns already developed for SDM with our proposed algorithm in bold.

10.3 Future work

With regard to possible future lines in our work, our intentions are the following:

- In the field of Vertical Database Format algorithms, we shall attempt to find a better representation with which to implement the IdLists of Vertical Database Format algorithms in order to spend less time on the join operations. We shall also seek some possible pruning methods that are sufficiently efficient to reduce the number of candidates generated.
- We shall address the use of an epsilon value in order to facilitate the mining of the final set of frequent patterns. A huge amount of patterns are traditionally almost the same, and only differ in a few time units which convert them into different patterns. The use of an epsilon value in this point would alleviate this problem.
- Since, whenever we execute an algorithm with low support values, we tend to find a huge number of patterns, we would like to develop an efficient post-processing step in order to search for those interesting patterns that can summarise the others.
- We intend to deal with databases in continuous domains by using quantitative mining. We shall tackle the discretization of such domains in order to carry out an efficient and effective quantitative mining.
- We shall study the creation of various post-processing steps in order to obtain useful patterns useful for classifications. Various clustering methods from this approach could be studied.
- We shall address the fifth type of patterns that appears in Table 1.1. These patterns include both the item durations and the temporal distances associated with relations which are expressed with ranges bounded by a lower bound and an upper bound. This kind of patterns could be very useful in several domains and need very complicated algorithm implementations if they are to be executed correctly.
- With regard to the field of closed sequential patterns, we shall study the possibility of implementing a Vertical Database Format algorithm in the mining of closed patterns composed of both point and interval events.

Bibliography

Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, volume 22, pages 207–216, New York, NY, USA. ACM.

Agrawal, R. and Srikant, R. (1995). Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA. IEEE Computer Society.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Álvarez, M. R., Félix, P., and Cariñena, P. (2013). Discovering metric temporal constraint networks on temporal databases. *Artif. Intell. Med.*, 58(3):139–154.

Antunes, C. and Oliveira, A. (2004). Sequential pattern mining algorithms: Trade-offs between speed and memory. In *2nd Workshop on Mining Graphs, Trees and Seq.*

Ayres, J., Flannick, J., Gehrke, J., and Yiu, T. (2002). Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435. ACM.

Bayardo, R. J. and Agrawal, R. (1999). Mining the Most Interesting Rules. In Fayyad, U. M., Chaudhuri, S., and Madigan, D., editors, *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154, San Diego, CA, USA. ACM.

Bettini, C., Wang, X. S., and Jajodia, S. (1996). Testing complex temporal relationships involving multiple granularities and its application to data mining (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 68–78, New York, NY, USA. ACM.

Boulicaut, J. and Jeudy, B. (2001). Mining free itemsets under constraints. In *Database Engineering & Applications, 2001 International Symposium on.*, number July, pages 322–329. IEEE.

Calders, T. and Goethals, B. (2007). Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206.

Campos, M., Palma, J., and Marin, R. (2007). Temporal Data Mining with Temporal Constraints. In *Proceedings of the 11th conference on Artificial Intelligence in Medicine*, AIME'07, pages 67–76. Springer-Verlag.

Casas-Garriga, G. (2005). Summarizing sequential data with closed partial orders. In *Proceedings of SDM 2005*, pages 380–391, California, USA.

Chen, J. (2010). An UpDown Directed Acyclic Graph Approach for Sequential Pattern Mining. *IEEE Transactions on Knowledge and Data Engineering*, 22:913–928.

Chen, Y., Chiang, M., and Ko, M. (2003). Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, 25(3):343–354.

Chen, Y., Peng, W., and Lee, S. (2011). CEMiner – An Efficient Algorithm for Mining Closed Patterns from Time Interval-Based Data. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 121–130, Washington, DC, USA. IEEE Computer Society.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1-2):199–227.

Giannotti, F., Nanni, M., Pedreschi, D., and Pinelli, F. (2006). Mining sequences with temporal annotations. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC '06, pages 593–597, New York, NY, USA. ACM.

Goethals, B. (2003). Survey on Frequent Pattern Mining. Manuscript.

Gouda, K. and Hassaan, M. (2011). Mining Sequential Patterns in Dense Databases. *Journal of Database Management*, 3(1):179–194.

Gouda, K., Hassaan, M., and Zaki, M. (2010). Prism: An effective approach for frequent sequence mining via prime-block encoding. *Journal of Computer and System Sciences*, 76(1):88–102.

Gouda, K. and Zaki, M. (2005). Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Mining and Knowledge Discovery*, 11(3):223–242.

Guil, F., Bosch, A., and Marin, R. (2004). TSET: An algorithm for mining frequent temporal patterns. In *Proceedings of the First Int. Workshop on Knowledge Discovery in Data Streams, in conjunction with ECML/PKDD 2004*, ECML/PKDD'04, pages 65–74, Pisa, Italy. Springer-Verlag.

Guyet, T. and Quiniou, R. (2008). Mining Temporal Patterns with Quantitative Intervals. In *Proceedings of the 2008 IEEE International Conference on Data Mining Workshops*, ICDM '08 Workshop, pages 218–227, Washington, DC, USA. IEEE Computer Society.

Guyet, T. and Quiniou, R. (2011). Extracting Temporal Patterns from Interval-Based Sequences. In *IJCAI*, pages 1306–1311, Barcelona, Catalonia, Spain. IJCAI/AAAI.

Han, J. and Pei, J. (2000). Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, pages 1–20.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M. (2000). FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 355–359, New York, NY, USA. ACM.

Höppner, F. (2001). Discovery of Temporal Patterns. Learning Rules about the Qualitative Behaviour of Time Series. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '01, pages 192–203, London, UK. Springer-Verlag.

Kam, P. and Fu, A. (2000). Discovering Temporal Patterns for Interval-Based Events. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*, DaWaK 2000, pages 317–326, London, UK. Springer-Verlag.

Leleu, M., Rigotti, C., Boulicaut, J., and Euvrard, G. (2003). Go-spade: Mining sequential patterns over datasets with consecutive repetitions. In *Proceedings of the 3rd international conference on Machine learning and data mining in pattern recognition*, MLDM'03, pages 293–306, Berlin, Heidelberg. Springer-Verlag.

Lin, M. and Lee, S. (2005). Fast discovery of sequential patterns through memory indexing and database partitioning. *Journal of information science and engineering*, 21(1):109–128.

Lucchese, C., Orlando, S., and Perego, R. (2004). DCI Closed: A Fast and Memory Efficient Algorithm to Mine Frequent Closed Itemsets. In *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Mannila, H. (2002). Local and Global Methods in Data Mining: Basic Techniques and Open Problems. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 57–68, London, UK. Springer-Verlag.

Mannila, H., Toivonen, H., and Inkeri Verkamo, A. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289.

Masseglia, F., Cathala, F., and Poncelet, P. (1998). The PSP Approach for Mining Sequential Patterns. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD '98, pages 176–184, London, UK, UK. Springer-Verlag.

Meiri, I. (1996). Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1–2):343–385.

Moerchen, F. (2010). Temporal pattern mining in symbolic time point and time interval data. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 2:1–2:1, New York, NY, USA. ACM.

Mörchen, F. (2006). *Time series knowledge mining.* PhD thesis, University of Marburg.

Mörchen, F. and Fradkin, D. (2010). Robust mining of time intervals with semi-interval partial order patterns. In *Proceedings of the 10th SIAM International Conference on Data Mining*, pages 315–326. SIAM.

Moskovitch, R. and Shahar, Y. (2009). Karmalego: Fast Time Intervals Mining. Technical report, Technical Report 23, ISE-TECHREP Ben Gurion University.

Nakagaito, F., Ozaki, T., and Ohkawa, T. (2009). Discovery of Quantitative Sequential Patterns from Event Sequences. In *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ICDMW '09, pages 31–36, Washington, DC, USA. IEEE Computer Society.

Palma, J., Juarez, J., Campos, M., and Marin, R. (2006). Fuzzy theory approach for temporal model-based diagnosis: An application to medical domains. *Artificial Intelligence in Medicine*, 38(2):197–218.

Papapetrou, P., Kollios, G., Sclaroff, S., and Gunopulos, D. (2005). Discovering Frequent Arrangements of Temporal Intervals. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 354–361, Washington, DC, USA. IEEE Computer Society.

Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *ICDT'99: Proceedings of the 7th International Conference on Database Theory*, pages 398–416. Springer.

Patel, D., Hsu, W., and Lee, M. L. (2008). Mining relationships among interval-based events for classification. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 393–404, New York, NY, USA. ACM.

Pei, J., Han, J., and Mao, R. (2000). CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *Proceedings of ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2004). Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440.

Piatetsky-Shapiro, G. and Frawley, W. J., editors (1991). *Knowledge Discovery in Databases*, volume 13. AAAI/MIT Press.

Reich, A. J. (1994). Intervals, Points, and Branching Time. In *Proceedings of the TIME-94 International Workshop on Temporal Reasoning.*

Roddick, J. and Mooney, C. (2005). Linear Temporal Sequences and Their Interpretation Using Midpoint Relationships. *IEEE Transactions on Knowledge and Data Engineering*, 17:133–135.

Roddick, J. and Spiliopoulou, M. (2002). A Survey of Temporal Knowledge Discovery Paradigms and Methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767.

Srikant, R. and Agrawal, R. (1996). Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pages 3–17, London, UK. Springer-Verlag.

Tzvetkov, P., Yan, X., and Han, J. (2005). TSP: Mining top-k closed sequential patterns. *Knowledge and Information Systems*, 7(4):438–457.

Vilain, M. (1982). A system for reasoning about time. In *Proceedings of AAAI*, volume 82, pages 197–201.

Wang, J., Han, J., and Li, C. (2007). Frequent closed sequence mining without candidate maintenance. *Knowledge and Data Engineering, IEEE Transactions on*, 19(8):1042–1056.

Winarko, E. and Roddick, J. (2007). ARMADA-An algorithm for discovering richer relative temporal association rules from interval-based data. *Data & Knowledge Engineering*, 63(1):76–90.

Wu, S. and Chen, Y. (2007). Mining non-ambiguous temporal patterns for interval-based events. *Knowledge and Data Engineering, IEEE*, 19(6):742–758.

Wu, S. and Chen, Y. (2009). Discovering hybrid temporal patterns from sequences consisting of point- and interval-based events. *Data Knowl. Eng.*, 68(11):1309–1330.

Yan, X., Han, J., and Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of SIAM International Conference on Data Mining*, pages 166–177.

Yang, Z., Wang, Y., and Kitsuregawa, M. (2007). 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. Springer-Verlag.

Yoshida, M., Iizuka, T., S., H., and Ishiguro, M. (2000). Mining sequential patterns including time intervals. In *Proceedings of SPIE*, volume 4057, pages 213–220.

Zaki, M. (2000a). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390.

Zaki, M. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60.

Zaki, M. J. (2000b). Sequence mining in categorical domains: incorporating constraints. In *Proceedings of the 9th International Conference on Information and Knowledge Management*, CIKM '00, pages 422–429, New York, NY, USA. ACM.

Zaki, M. J. and Hsiao, C. (2002). CHARM: An Efficient Algorithm for Closed Itemset Mining. In *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. SIAM.

UNIVERSIDAD DE MURCIA

Departamento de Ingeniería de la Información y las Comunicaciones

Resumen Extendido de Tesis Doctoral

Técnicas para el Descubrimiento de Patrones Temporales

Autor

Antonio Gomariz Peñalver

Directores

Roque Marín Morales

Manuel Campos Martínez

Bart Goethals

Diciembre 2013

1. Introducción

1.1. Contexto y Motivación

Esta tesis se ha desarrollado dentro del contexto del proyecto de investigación de AI-SENIOR. Dicho proyecto, se marca como objetivo principal el establecimiento de pautas de comportamiento de sujetos que están continuamente monitorizados, los cuales se corresponden con personas de edad avanzada, padeciendo, muchos de ellos, enfermedades que determinan su calidad de vida. AI-SENIOR se centra en tres diferentes grupos de personas de edad avanzada, así como en el desarrollo y la validación de los resultados. Estos tres grupos están compuestos por: 1) poblaciones de personas que tienen algún tipo de enfermedad cardiovascular, 2) personas que padecen algún tipo de enfermedad pulmonar crónica, y 3) un último grupo formado por personas que pertenecen a la tercera edad y que, a pesar de que no sufren ningún tipo de enfermedad, viven completamente solos y requieren de la supervisión diaria de sus actividades con el fin de reconocer las diferentes situaciones que se producen en ellas.

Los tres grupos de personas, descritos anteriormente, presentan ciertos riesgos, ya sea debido a sus vidas en soledad o a las enfermedades que padecen, de forma que es necesaria una asistencia para mejorar su calidad de vida. A esto se añade que todos estos grupos representan una amplia proporción de la población, y el coste de proporcionar una asistencia a ésta es un problema particularmente importante para las administraciones públicas.

De este modo, un equipo de dispositivos fue instalado en cada domicilio, estando cada uno de dichos equipos integrado por varios sensores, dependiendo del grupo al que se aplicara. Este equipo de sensorización es heterogéneo, tanto desde el punto de vista del tipo de sensor, como de la salida proporcionada. En primer lugar, los sensores podían ser bien "wearables", o bien no invasivos. Además, la información dada por estos sensores puede ser de varios tipos: binaria, tales como aquellos que detectan la presencia; numérica, tales como aquellos que detectan la temperatura; y otros de los cuales son multi-variables, tales como aquellos que detectan la actividad (que proporciona una aceleración en cada eje). Adicionalmente, algunos de esos sensores fueron configurados con alta frecuencia (los que corresponden a los dos primeros grupos), mientras que otros tenían frecuencias más bajas (los que pertenecen al tercer grupo de estudio). Una última característica que diferencia a los sensores es el modo de activación, habiendo sensores que proporcionaban una serie temporal con una frecuencia fija, mientras que otros sólo proporcionaban una señal cuando una actividad era detectada.

Al encontrar tanta diversidad, es necesario crear un marco donde se integren estas fuentes de datos y que permite representar toda la información, y darle una interpretación adecuada. Por ejemplo, la cantidad de tiempo que una persona permanece en una habitación puede tener diferentes significados, y por lo tanto se hace necesario considerar los eventos como intervalos.

Uno de los sub-objetivos del proyecto corresponde a realizar una tarea de descubrimiento de conocimiento que nos permitirá: (i) determinar el nivel de supervisión que una persona debe tener en su domicilio, (ii) ajustar el sistema de alarmas con la máxima precisión posible. Nosotros creemos que los "patrones secuenciales" son una herramienta muy útil en la tarea de modelar el comportamiento de los usuarios. Es por ello que necesitamos definir estos patrones en los diferentes niveles de complejidad para así obtener una visión más detallada siempre que se necesite. Consecuentemente, podemos comenzar mediante la obtención de patrones simples, tal y como cuando una persona, durante la noche, se levanta, bebe agua, va al baño, para acabar volviendo a la cama; o por el contrario, considerar patrones mucho más complejos que incluyan eventos intervalos con distancias temporales entre ellos, como por ejemplo, una persona duerme durante 3 horas, después pasa 3 minutos en la cocina y, después de ver la televisión durante 40 minutos, regresa a dormir 2 horas más. Hay una amplia variedad de patrones entre los anteriores dos extremos.

El objetivo de esta tesis es proponer un marco general que sea capaz de hacer frente a las distintas bases de datos que podamos encontrar dentro del campo de la inteligencia ambiental, además de encontrar algoritmos interesantes que puedan ser de gran valor cuando nos enfrentamos a estos problemas.

1.2. Temporal Data Mining

Uno de los problemas que recientemente ha emergido asociado con las tecnologías de la información es analizar la enorme cantidad de datos que se origina durante las actividades que ocurren diariamente en las organizaciones. El proceso general a través del cual se lleva a cabo este análisis se llama “descubrimiento del conocimiento en bases de datos” (KDD por sus siglas en inglés), y se define como ”la extracción no trivial de información implícita, previamente desconocida, y potencialmente útil que proviene de los datos”[Piatetsky-Shapiro and Frawley, 1991]. Dicho proceso se necesita para obtener información útil y valiosa para las organizaciones, y nos permite, por ejemplo, analizar un problema médico y extraer la información que puede ser relevante para un proceso de toma de decisiones, tal y como: estudiar manifestaciones y síntomas durante el curso de una enfermedad, analizar las causas de mortalidad de un grupo de pacientes que tienen un problema concreto, etc.

El paso esencial es KDD es la denominada fase de Minería de Datos (MD), que incorpora técnicas muy diferentes provenientes de los campos de aprendizaje computacional, estadística, sistemas de ayuda a la decisión y la inteligencia artificial en general, junto con otras ideas procedentes de la computación y de los sistemas de gestión de la información. La MD consiste en aplicar análisis de datos y algoritmos de descubrimiento que producen una enumeración particular de estructuras sobre los datos [Fayyad et al., 1996], pudiendo ser estas estructuras patrones o modelos [Mannila, 2002]. La MD se utiliza en distintas tareas, tales como la clasificación, clustering, predicción o el descubrimiento de patrones interesantes (ej: patrones ocultos, tendencias u otras relaciones presentes en los datos).

En la mayor parte de las técnicas propuestas en la literatura asociada con la MD, el análisis de datos se lleva a cabo sin tener en cuenta la componente temporal de los datos. Sin embargo, existen campos de aplicación en los que los datos que se analizan tienen interdependencias temporales. En otras palabras, el orden entre ellos es fundamental para el análisis. La Minería de Datos Temporal (MDT) se ha convertido en una importante rama de la MD y puede ser definida como la búsqueda de asociaciones interesantes o patrones en grandes conjuntos de datos temporales acumulados para otros propósitos [Bettini et al., 1996]. La MDT puede descubrir actividades, inferir asociaciones de proximidad contextual y temporal, algunas de las cuales también puede indicar una relación causa-efecto. Esta importante clase de conocimiento se puede pasar por alto cuando la componente temporal es ignorada o tratada como un simple atributo numérico [Roddick and Spiliopoulou, 2002].

Dentro de TDM, existe una tarea importante que se llama Minería de Patrones Temporales (MPT) que busca los patrones que aparecen en al menos un número mínimo de entradas, llamado soporte, a partir de una base de datos temporal. Estos patrones pueden ser muy diferentes dependiendo del contexto de estudio y el tipo de información que se busca. Existen dos enfoques principales dentro de esta área, y se denominan minería de itemsets (IDM) y minería de datos secuenciales (SDM). Mientras IDM representa el concepto temporal de la sincronicidad y extrae un conjunto de itemsets o transacciones (ítems que ocurren simultáneamente) desde las bases de datos de entrada (formada por itemsets). En cambio, SDM representa el concepto temporal de orden y utiliza una base de datos de entrada formada por secuencias, en la que cada secuencia es una lista de itemsets. Mientras IDM es el caso más simple, SDM [Agrawal and Srikant, 1995] es mucho más complejo, teniendo un espacio de búsqueda mucho más grande que en IDM, ya que además de la búsqueda de los itemsets frecuentes, también es necesario tener en cuenta la relación temporal entre estos itemsets. La investigación en SDM ha generado varios algoritmos que pueden ser utilizados en diferentes aplicaciones, tales como el descubrimiento de motivos en secuencias de ADN, el análisis de secuencias de compras de clientes, etc.

La MPT puede ser analizada desde diferentes perspectivas: representación de patrones y expresividad de los patrones. Con respecto a la representación de los patrones se parte de considerar los eventos como puntos o como intervalos. En el primer caso, para un mismo evento, cada aparición se considera como un punto temporal. Por ejemplo, si estamos trabajando con una granularidad de horas en el contexto de un hospital y consideramos un evento que denota si un determinado paciente tiene fiebre, podemos considerar que tenemos puntos de ese evento cada hora en la que la fiebre está presente.

Desafortunadamente, este tipo de abstracción tan simple no siempre es suficiente para expresar cada uno de los eventos al completo. En muchos casos del mundo real, tales como contextos médicos, multimedia, meteorológicos o financieros, los eventos tienden a persistir durante un periodo de tiempo en vez de ocurrir en un instante puntual. En este caso, los datos generalmente son secuencias de eventos que tienen un instante comienzo y un instante final. Por tanto, cada evento intervalo tiene una duración asociada durante la cual éste está activado en un patrón. Continuando con el ejemplo anterior, con una representación de intervalos, la aparición de un evento es considerada durante todo el período de tiempo en el que tal evento está activado, es decir, si, por ejemplo, se detecta fiebre en un paciente durante cinco horas consecutivas, consideramos las cinco horas como una sola aparición de fiebre en el patrón.

En lo que se refiere a la expresividad de las relaciones que existen entre los eventos de los patrones, existen varias opciones, pero la mayoría de los algoritmos se centran generalmente en relaciones cualitativas o relaciones cuantitativas. Una relación cualitativa sólo representa la relación de orden que aparece entre dos eventos. Por ejemplo, si tenemos en cuenta dos eventos puntuales A y B que aparecen los tiempos 1 y 2, respectivamente, existe un relación "before" entre ellos, y de igual manera, si A y B, por el contrario, aparecen en tiempos 1 y 100, la relación entre ellos será exactamente la misma (A $before$ B), siendo imposible hacer ninguna distinción entre ambos patrones. Por otro lado, cuando necesitamos expresar la relación exacta que existe entre dos eventos entonces incluimos una distancia temporal además de la relación temporal. Por ejemplo, considerando el mismo ejemplo de antes, descubrimos dos patrones diferentes: uno en el que A aparece antes que B, con una unidad de tiempo de distancia temporal (A $before$[1] B), y un segundo en el que A aparece antes que B, con 99 unidades de tiempo de diferencia (A $before$[99] B).

La mayoría de los esfuerzos realizados hasta la fecha han determinado el uso de un tipo de representación y expresividad para extraer una base de datos de entrada bajo tales restricciones. Por tanto, la mayoría de los trabajos se diferencian entre las bases de datos basadas en puntos o las basadas en intervalos, teniendo en cuenta bien relaciones cualitativas o relaciones cuantitativas. No obstante, hay algunos entornos en los que el uso de casos mixtos sería bastante conveniente. Por ejemplo, podrían citarse algunos casos en los que ciertos atributos deben ser registrados como intervalos, mientras que otros podrían ocurrir ocasionalmente como puntos. En estos casos es necesario considerar al mismo tiempo puntos e intervalos, tanto en la base de datos como en los patrones que encontramos. Por ejemplo, en el caso de una base de datos médica puede haber diagnósticos y tratamientos de eventos en los que un diagnóstico se establece en un momento, pero un tratamiento tiene una duración continuada en el tiempo.

En SDM tradicionalmente se han considerado representaciones de puntos, teniendo relaciones cualitativas como modelo de datos. Esta representación basada en puntos genera patrones con tres diferentes relaciones temporales ("before" , "equals" y "after"). Sin embargo, si se utiliza una representación basada en intervalos, aumenta la complejidad de todos los aspectos comunes en SDM. Por ejemplo, mientras que en las representaciones de puntos sólo se utilizan las relaciones entre los mismos, para expresar el orden temporal dentro de un patrón, en las representaciones de intervalos temporales hay diferentes maneras con las que relacionar tales intervalos, como es el Álgebra de Allen [Allen, 1983] o las relaciones entre semi-intervalos de Freksa [Freksa, 1992]. De entre estas representaciones, la más habitual es el Álgebra de Allen, el cual tiene trece relaciones cualitativas que configuran un lenguaje muy completo y expresivo. La representación de patrones y las tareas que se relacionan con el razonamiento temporal se hacen mucho más complicadas en esta álgebra de intervalos que en la de puntos. Estas relaciones entre eventos de un patrón son importantes cuellos de botella que aparecen cuando nos interesamos en el desarrollo de un algoritmo eficiente con el que extraer patrones complejos, ya que estas relaciones pueden dar lugar a la generación de un número mucho mayor de secuencias candidatas que cuando tan sólo consideramos puntos. Para evitar todos los inconvenientes que resultan de las relaciones de intervalos, varias representaciones de patrones han sido propuestas, las cuales proporcionan las características de compactibilidad, la legibilidad y la no-ambigüedad .

Todas las representaciones cualitativas pueden ser convertidas en representaciones cuantitativas mostrando la distancia temporal entre los diferentes eventos, lo que significa que las relaciones se hacen más complicadas y su tratamiento es más costoso. Esto conduce a una mayor explosión de patrones ya que, para un soporte dado, cuando tenemos en cuenta esta relación cuantitativa nos encontramos con menos apariciones de un mayor número de patrones que cuándo sólo consideramos relaciones cualitativas. Por lo tanto, es necesario hacer hincapié en la escalabilidad de los algoritmos utilizados para extraer tales patrones. Esta escalabilidad se reflejará en las estrategias de búsqueda, la representación de los patrones, la generación de candidatos, el conteo de soporte y los métodos de poda.

Varios trabajos se han llevado a cabo para SDM, entre los cuales tres principales estrategias de búsqueda pueden ser claramente identificadas: 1) estrategia Apriori [Srikant and Agrawal, 1996], 2) estrategia basada en bases de datos en formato vertical [Zaki, 2001], y 3) estrategia basada en crecimiento de patrones [Pei et al., 2004]. Mientras los algoritmos Apriori consisten en la ejecución de un bucle que lleva a cabo una generación candidatos seguida por una fase de chequeo de frecuencia para cada candidato generado, las estrategias basadas en bases de datos en formato vertical y de crecimiento de patrones son métodos más directos que, en general, obtienen mejores resultados en lo que respecta tiempo de ejecución y consumo de memoria.

La mayoría de los algoritmos para minería de patrones secuenciales desarrollados hasta la fecha han sido diseñados para configuraciones de bases de datos específicas presentes en una serie de bases de datos reales, por ejemplo, bases de datos que tienen secuencias cortas, o con un número limitado de eventos, o con secuencias en las que no se da repetición de eventos, etc. En general, cuando estas condiciones cambian, el rendimiento de estos algoritmos disminuye drásticamente. Incluso, hay casos donde los algoritmos consiguen completar sus ejecuciones, pero el número de patrones frecuentes que se encuentran es tan grande que aparecen una gran cantidad de patrones que no aportan ninguna información significativa.

Un enfoque interesante utilizado en IDM, con el fin de resolver el problema mencionado en el párrafo anterior, consiste en buscar patrones que tienen ciertas propiedades concretas, tales como conjuntos de patrones “Closed” [Pasquier et al., 1999]. Buscar itemsets Closed nos proporciona dos beneficios al mismo tiempo: una reducción en el número de candidatos, y la obtención de una salida más compacta mientras se mantiene la máxima cantidad de información proporcionada por la ejecución del algoritmo. Dado que buscar secuencias Closed es parecido a buscar itemsets Closed, una estrategia de descubrimiento de patrones Closed también puede ser usada en SDM.

Esta tesis propone la realización de un amplio estudio y una amplia comparación de todas las alternativas anteriormente mencionadas. Además, también se proponen varios algoritmos, todos los cuales son capaces de trabajar bajo diferentes tipos de representación, expresividad y configuración de la base de datos.

1.3. Objetivo y sub-objetivos

El objetivo principal de esta tesis es definir un marco general para métodos de minería de datos secuencial (SDM), estrategias, estructuras de datos, y expresividad de los patrones, y para desarrollar nuevos algoritmos en todos los niveles posibles de SDM. Para lograr este objetivo, hemos definido los siguientes sub-objetivos:

- Mostramos, a través un capítulo del estado del arte, los problemas derivados de SDM, haciendo una comparación en profundidad entre todas las estrategias que se aplican a SDM.

- Hemos estructurado y organizado el conocimiento acerca de SDM a través de las siguientes dimensiones: representación de patrones, expresividad de los patrones y estrategias de búsqueda. Las Tablas 1, 2 y 3, son una guía para los objetivos de esta tesis que se muestran a continuación.

- Mostramos y comparamos todos los problemas derivados de los diferentes niveles de la representación de patrones (puntos, intervalos y puntos e intervalos).

- Visualizamos y contrastamos las cuestiones relacionadas con los diferentes niveles de expresividad (patrones cualitativos y patrones cuantitativos). Tanto para la representación como para la expresividad completamos algunas de las lagunas que no habían sido tratadas y que se muestran en la Tabla 1. Concretamente, nos centramos en los patrones mostrados en la primera, segunda, cuarta y quinta fila.

- Definimos cuatro nuevos algoritmos que son capaces de extraer patrones de bases de datos basadas en puntos, intervalos y puntos e intervalos. Cada algoritmo implementa una estrategia concreta y puede encontrar tanto relaciones cualitativas como relaciones cuantitativas. Los nombres de estos algoritmos son PaGAPIS, FaSPIP, BreadthPIS y DepthPIS para patrones cualitativos, y PaGAPIMS, FaSPIMP, BreadthPIMS y DepthPIMS para patrones cuantitativos. Además, estudiamos los métodos de poda que pueden ser aplicados, y la conveniencia de su uso. Estos algoritmos se utilizan para cubrir algunas de las lagunas que aparecen en la Tabla 2. Esto es posible gracias a que creamos nuevas combinaciones en términos de representación, expresividad y la representación de datos, que aún no han sido cubiertos. Téngase en cuenta que mientras que PaGAPIS y PaGAPIMS se basan en la estrategia basada en crecimiento de patrones, el resto de algoritmos que proponemos se basan en la estrategia basada en bases de datos en formato vertical. El motivo de hacer implementaciones de ambas estrategias es para hacer una comparación justa entre las estrategias, de manera integral.

- Se discute la conveniencia de utilizar cada estrategia de búsqueda para una representación y expresividad sobre una base de datos de entrada dada. También analizamos qué estrategia es la más adecuada para su aplicación cuando tratamos con características específicas en la base de datos de entrada. Concretamente, estudiamos la conveniencia de utilizar los diferentes algoritmos propuestos en esta tesis, haciendo una amplia comparación.

- Se propone un nuevo algoritmo con el que gestionar los problemas de explosión de patrones por medio de la minería de secuencias Closed y completar algunas de las lagunas que aparecen en la Tabla 3. Este nuevo algoritmo es el primero basado en la estrategia bases de datos en formato vertical que, por medio de varios métodos de poda, evita la generación de secuencias no cerradas.

Actualmente no tenemos constancia de que algún trabajo haya abordado un marco global para la minería patrones secuenciales, considerando una complejidad progresiva de los patrones aplicados a un solo dominio. Existen distintos trabajos que utilizan algoritmos que se ocupan de un único problema para un único tipo de patrones objetivo, que muestran sus ventajas sobre otros enfoques al susodicho problema objetivo abordado. Nos gustaría señalar que las bases de datos con las que nosotros experimentamos, cubren todo el espectro de posibilidades de patrones que nosotros necesitamos.

Con el fin de facilitar la lectura de esta tesis aportamos dos tablas que muestran las dimensiones analizadas con los objetivos previamente mencionados. La Tabla 1 muestra dos grupos: los patrones basados en puntos, que aparecen en las tres primeras filas, y los patrones basados en intervalos, en las tres últimas filas. Cada grupo puede tener dos tipos de relaciones: cualitativa o cuantitativa. Así mismo, existen dos tipos de patrones cuantitativos: aquellos con distancias temporales exactas y aquellos en los que la distancia está limitada por un rango que va desde una cota inferior, hasta una cota superior.

Tipo de patrones	Representación
Cualitativos basados en puntos	$A < B$
Cuantitativos exactos basados en puntos	$A < [5]B$
Cuantitativos con rangos basados en puntos	$A < [5,7]B$
Cualitativos basados en intervalos	$A\ o\ B$, $A < C$, $B\ o\ C$
Cuantitativos exactos basados en intervalos	$A[4]\ o[2]\ B[7]$, $A[4] < [3]C[8]$, $B[7]\ o[2]\ C[8]$
Cuantiativos con rangos basados en intervalos	$A[4,6]\ o[2,4]\ B[7,10]$, $A[4,6] < [1,3]C[8,12]$, $B[7,10]\ o[2,5]\ C[8,12]$

Cuadro 1: Clasificación de los diferentes tipos de patrones.

La Tabla 2 muestra una clasificación de los diferentes algoritmos desarrollados para SDM hasta la fecha (los detalles de estos algoritmos se proporcionan en la sección del estado del arte). En esta tabla, las dimensiones son el tipo de base de datos de entrada (bases de datos basadas en puntos, intervalos o puntos e intervalos), la expresividad (relaciones cualitativas o cuantitativas), y la estrategia de la minería elegida (Apriori, basada en bases de datos en formato vertical y basada en crecimiento de patrones). Como se puede observar, todavía hay algunas lagunas que requieren más estudio. Además, la Tabla 3 muestra los algoritmos de minería de patrones Closed, basados en eventos punto, que existen en SDM. Como podemos observar, no hay ningún algoritmo que siga una estrategia basada en bases de datos en formato vertical.

2. Conclusiones

En este capítulo terminamos este trabajo mostrando nuestras conclusiones, aportaciones que brindamos, así como algunas ideas que pueden ser desarrolladas en trabajos futuros.

2.1. Conclusiones

En esta tesis hemos definido un marco general para la minería de datos secuencial (SDM). Este marco general se ha utilizado para tratar diversos algoritmos que son capaces

Base de Datos	Representación	Distancias	Estrategias		
			Apriori	Crecimiento de Patrones	Bases de datos en formato Vertical
Puntos		Cualitativos	Apriori-All, GSP, PSP, TSET	FreeSpan, PrefixSpan, Memisp	SPADE, SPAM, Prism
		Cuantitativos	I-Apriori, Yoshida	MisTA, QprefixSpan, i-PrefixSpan	
Intervalos	Puntos	Cuantitativos		QTPrefixSpan	
		Cualitativos		T-PrefixSpan	
	Intervalos	Cuantitativos	QTempIntMiner	QTiPrefixSpan	
		Cualitativos	IEMiner, Karmalego	Armada	H-DFS
Puntos e Intervalos	Puntos	Cuantitativos	ASTPminer		
		Cualitativos	HTPM	CTMiner, CEMiner	
	Intervalos	Cuantitativos			
		Cualitativos			

Cuadro 2: Clasificación de los diferentes algoritmos ya existentes para SDM.

Apriori	Crecimiento de Patrones	Bases de datos en formato Vertical
	CloSpan, Bide	

Cuadro 3: Clasificación de los diferentes algoritmos para extraer patrones Closed con eventos punto, ya existentes en SDM.

de extraer patrones con diferentes tipos de representación (puntos, intervalos o puntos e intervalos) y expresividad (patrones cualitativos o cuantitativos).

Además hemos presentado un estado del arte en el que hemos mostrado todas las estrategias que se utilizan normalmente en la minería de patrones. Este estado del arte permite organizar todos los trabajos previos en este campo, lo que ha contribuido a definir la base y las formalidades para la SDM.

Hemos estudiado el problema de la SDM desde varios puntos de vista. En primer lugar, se ha llevado a cabo un estudio con el fin de descubrir qué tipo de estrategia, de aquellas que se utilizan con más frecuencia (basadas en bases de datos en formato vertical o en crecimiento de patrones), es la más apropiada para ser usada en la extracción de los diferentes niveles de representación y expresividad. En segundo lugar, hemos comparado dos representaciones de patrones, una basada en "extremos de intervalos" y otra basada en matrices triangulares, con el objetivo de estudiar cuál de ellas nos aporta un mayor beneficio.

Para llevar a cabo este estudio, hemos desarrollado un generador de bases de datos sintéticas. A través de este generador hemos podido establecer los parámetros más influyentes, tal y como son el número de diferentes ítems, el número de ítems por itemset, el número de itemsets por secuencia, el número de secuencias, la longitud patrón o el soporte, que nos han permitido conducir diferentes experimentos con los más minuciosos detalles. No hemos usado bases de datos reales, como por ejemplo las indicadas en [Mörchen and Fradkin, 2010], ya que el comportamiento de los algoritmos en estas bases de datos también está definido por los parámetros antes mencionados y, necesariamente, estas bases de datos también deben ser analizadas en función de tales parámetros.

Con este fin, hemos desarrollado cuatro algoritmos con los que extraer patrones con puntos, intervalos o ambos a la vez, que también son capaces de descubrir patrones cualitativos y cuantitativos. Los algoritmos que extraen patrones cualitativos se llaman PaGAPIS, FaSPIP, BreadthPIS y DepthPIS. El primero de ellos se basa en la estrategia basada en crecimiento de patrones, mientras que los otros tres se basan en la estrategia basada en bases de datos en formato vertical. En lo que se refiere a las versiones cuantitativas de dichos algoritmos, éstos se denominan PaGAPIMS, FaSPIMP, BreadthPIMS y DepthPIMS. Tanto PaGAPIS y FaSPIP (y por lo tanto PaGAPIMS y FaSPIMP) implementan una representación de "extremos de intervalos", mientras que la representación que implementan BreadthPIS y DepthPIS (y BreadthPIMS y DepthPIMS) se basa en matrices triangulares.

Hasta donde sabemos, FaSPIP es el primer algoritmo que implementa una estrategia basada en bases de datos en formato vertical con una representación "extremos de intervalos", mientras que DepthPIS es el primer algoritmo realizado para extraer tanto puntos como intervalos que se utiliza clases de equivalencia en la búsqueda de patrones. Por otra parte, hasta la fecha no existe ningún trabajo que haya aplicado una estrategia basada en bases de datos en formato vertical para la extracción de patrones cuantitativos, tal y como se presenta en esta tesis, por medio de los algoritmos FaSPIMP, BreadthPIMS y DepthPIMS. Todos estos algoritmos completan algunas de las lagunas que fueron descubiertos en el estado del arte, tal y como puede verse en la Tabla 4, lo que nos permite extender y resolver estos problemas de una manera eficiente.

Hemos llevado a cabo una amplia serie de pruebas en las que se han variado las distintas propiedades que se asocian con las configuraciones de bases de datos. Las pruebas realizadas muestran que, en la mayoría de los casos, la estrategia basada en bases de datos en formato vertical es la más adecuada para buscar en bases de datos de puntos e intervalos, especialmente cuando tratamos con bases de datos densas con varios itemsets por secuencia. Una base de datos es considerada como densa cuando el tamaño promedio de los itemsets se acerca al número de diferentes ítems que hay en toda la base de datos. Esta situación nunca antes ha sido estudiada en algoritmos o comparativas anteriores.

Adicionalmente, hemos contrastado todas las ventajas e inconvenientes que encontramos al utilizar una representación basada en “extremos de intervalos” o basada en matrices triangulares, encontrando que la primera representación es la mejor opción para extraer patrones cualitativos mientras que la segunda es la más adecuada para la extraer patrones cuantitativos. En definitiva y en términos generales, concluimos que FaSPIP es el algoritmo más rápido para extraer patrones cualitativos, mientras que los algoritmos BreadthPIMS y DepthPIMS son los más eficientes para extraer patrones cuantitativos.

Del mismo modo, hemos estudiado las dificultades y problemas intrínsecos que encontramos al extraer patrones cuantitativos, así como cuando utilizamos una representación de “extremos de intervalos” o cuando el algoritmo sigue una estrategia basada en bases de datos en formato vertical. Además, hemos proporcionado una buena solución para paliar el inconveniente que surge al extraer los patrones cuantitativos frecuentes de longitud 2 que ocurre en algoritmos que siguen una estrategia basada en bases de datos en formato vertical. Gracias a este estudio comprendemos por qué FaSPIMP no logra tan buenos resultados cuando extrae patrones cuantitativos y por qué BreadthPIMS y DepthPIMS muestran los mejores comportamientos, demostrando ser los mejores algoritmos para la extracción de patrones cuantitativos.

Un aspecto novedoso, es que hemos usado, a través de BreadthPIS y DepthPIS (y sus versiones cuantitativas BreadthPIMS y DepthPIMS), razonamiento temporal para generar candidatos. El uso de este razonamiento temporal es altamente eficiente ya que solamente son generados, de una forma rápida, aquellos candidatos que se necesitan. Hemos propuesto dos diferentes razonamientos, uno para BreadthPIS y otro para DepthPIS, siendo el razonamiento de BreadthPIS el más optimizado porque, en general, genera un conjunto de candidatos más reducido. Hasta donde sabemos, pocos trabajos en SDM han confrontado el uso de razonamiento temporal tan detallado como nosotros hemos realizado en BreadthPIS y DepthPIS.

En lo que se refiere a los métodos de búsqueda utilizados, hemos estudiado cuál es el mejor cuando extraemos patrones de bases de datos con puntos e intervalos. Los algoritmos que ejecutan un método de búsqueda en anchura (utilizado en BreadthPIS y BreadthPIMS) pueden ejecutar un método de poda y diseñar una tabla de transición muy eficaz con la que generar candidatos. Sin embargo, la exploración en anchura conduce a problemas de desbordamiento de memoria cuando tratan con enormes bases de datos. Por otra parte, si usamos un método de búsqueda en profundidad (utilizado en DepthPIS y DepthPIMS), encontramos que éste es más adecuado al no contar con ese problema, aunque el razonamiento temporal que se deriva de él, y por lo tanto la fase de generación de candidatos, no es tan eficaz como lo es en BreadthPIS (y BreadthPIMS). Es interesante ver que, los métodos de poda que fueron añadidos a BreadthPIS demostraron que no aportaban ninguna ganancia al tiempo de ejecución. Por lo tanto, en general, es preferible utilizar FaSPIP para los patrones cualitativos y DepthPIMS para patrones cuantitativos, en lugar de BreadthPIS o BreadthPIMS.

Finalmente, hemos estudiado los principios utilizados para extraer patrones secuenciales frecuentes Closed, y también hemos diseñado un algoritmo que sigue una estrategia basada en bases de datos en formato vertical capaz de extraer los patrones frecuentes Closed con eventos punto. El algoritmo, llamado ClaSP, utiliza métodos heurísticos para podar los patrones que no son Closed, y ha sido inspirado por el algoritmo CloSpan. Hasta donde sabemos, este es el primer trabajo que sigue una estrategia basada en bases de datos en formato vertical que se ha utilizado para resolver el problema propuesto y completa una laguna que existía en el estado del arte, tal y como puede verse en la Tabla 5. En todas las pruebas realizadas, ClaSP supera los algoritmos de estado del arte, y llegamos a la conclusión de que la estrategia basada en bases de datos en formato vertical es altamente adecuada para el descubrimiento de patrones Closed.

2.2. Aportaciones

Esta tesis hace las siguientes aportaciones:

1. Hemos proporcionado un amplio marco de definiciones de la minería de datos secuencial, que muestra los diferentes tipos de patrones que pueden ser relevantes para la SDM.

2. Ofrecemos una clara organización de las diferentes representaciones de patrones con intervalos, de los métodos de búsqueda que normalmente se utilizan y de las diferentes estrategias que han sido utilizadas para la SDM. También revelamos algunas cuestiones abiertas en SDM que deben ser afrontadas.

3. Hemos completado algunas de estas lagunas mediante el desarrollo de cuatro algoritmos capaces de extraer patrones cualitativos y cuantitativos en bases de datos con eventos puntos e intervalos. Cada algoritmo es capaz de encontrar los primeros cuatro tipos de patrón que son mostrados en la Tabla 1 con una sola implementación, y que funciona en cualquier tipo de base de datos, ya sean integradas por puntos, intervalos o puntos e intervalos.

4. Hemos desarrollado un algoritmo a través del cual extraer patrones frecuentes cualitativos Closed, integrados únicamente por puntos, que sigue una estrategia basada en bases de datos en formato vertical. Hemos medido el impacto del método de poda en la reducción de tanto el resultado final como del tiempo de ejecución.

5. Hemos proporcionado un generador sintético de base de datos secuenciales que es capaz de crear bases de datos con configuraciones muy diferentes. Hemos utilizado estas bases de datos para evaluar nuestros algoritmos y para guiar las discusiones sobre las dimensiones del problema de la SDM.

6. Hemos publicado el trabajo relativo al Capítulo 7 en un congreso internacional. Dicho trabajo ha sido publicado en las actas del congreso Pacific-Asia Conference on Knowledge Discovery and Data Mining, celebrado en Gold Coast, Australia en 2013.

Base de Datos	Representación	Distancias	Estrategias		
			Apriori	Crecimiento de Patrones	Bases de datos en formato Vertical
Puntos		Cualitativos	Apriori-All, GSP, PSP, TSET	FreeSpan, PrefixSpan, Memisp	SPADE, SPAM, Prism
		Cuantitativos	I-Apriori, Yoshida	MisTA, QprefixSpan, i-PrefixSpan	
Intervalos	Puntos	Cuantitativos		QTPrefixSpan	
		Cualitativos		T-PrefixSpan	
	Intervalos	Cuantitativos	QTempIntMiner	QTiPrefixSpan	
		Cualitativos	IEMiner, Karmalego	Armada	H-DFS
Puntos e Intervalos	Puntos	Cuantitativos	ASTPminer	**PaGAPIMS**	**FaSPIMP**
		Cualitativos	HTPM	CTMiner, CEMiner, **PaGAPIS**	**FaSPIP**
	Intervalos	Cuantitativos	**BreadthPIMS**		**DepthPIMS**
		Cualitativos	**BreadthPIS**		**DepthPIS**

Cuadro 4: Clasificación de los diferentes algoritmos, ya existentes para SDM, con los algoritmos desarrollados en esta Tesis señalados en negrita.

Apriori	Crecimiento de Patrones	Bases de datos en formato Vertical
	CloSpan, Bide	**ClaSP**

Cuadro 5: Clasificación de los diferentes algoritmos para extraer patrones Closed con eventos punto, ya existentes en SDM, con el algoritmo desarrollado en esta Tesis señalado en negrita.

2.3. Trabajo Futuro

Con respecto a las posibles líneas futuras de nuestro trabajo, nuestras intenciones son las siguientes:

- En el campo de los algoritmos que implementan una estrategia basada en bases de datos en formato vertical, vamos a tratar de encontrar una mejor representación con la que implementar las IdLists con el fin de mejorar la eficiencia en las operaciones de intersección. También nos proponemos buscar algunos posibles métodos de poda que sean lo suficientemente eficientes para reducir el número de candidatos generados.

- Trataremos el uso de un valor umbral épsilon para facilitar la extracción de los patrones frecuentes. Hay una gran cantidad de patrones que son casi iguales, y sólo se diferencian en unas pocas unidades de tiempo que los convierten en diferentes patrones. El uso de un valor umbral épsilon podría aliviar este problema.

- Dado que, cada vez que se ejecuta un algoritmo con bajos valores de soporte, tendemos a encontrar un gran número de patrones, nos gustaría desarrollar un post-procesado eficiente con el fin de buscar patrones interesantes que puedan resumir muchos otros.

- Tenemos la intención de hacer frente a las bases de datos de dominios continuos mediante el uso de la minería de patrones cuantitativos. Para ello, nos planteamos hacer una discretización de estos dominios con el fin de llevar a cabo una extracción cuantitativa eficiente y eficaz.

- Vamos a estudiar la creación de varias etapas de post-procesamiento con el fin de obtener una clasificación de patrones. Varios métodos de agrupamiento pueden ser estudiados desde este enfoque.

- Trataremos el quinto tipo de patrones que aparece en la Tabla 1. Estos patrones incluyen tanto las duraciones de los ítems como las distancias temporales asociadas con las relaciones, las cuales son expresadas a través de un rango con una cota inferior y una cota superior. Este tipo de patrones podría ser muy útil en diversos dominios y necesitan de implementaciones de algoritmos muy complejas para que puedan ser ejecutados correctamente y eficientemente.

- En lo que respecta al ámbito de los patrones secuenciales Closed, vamos a estudiar la posibilidad de implementar un algoritmo que siga una estrategia basada en bases de datos con formato vertical y que extraiga patrones Closed compuestos tanto por eventos punto y eventos intervalo.

Referencias

Agrawal, R. and Srikant, R. (1995). Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA. IEEE Computer Society.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Bettini, C., Wang, X. S., and Jajodia, S. (1996). Testing complex temporal relationships involving multiple granularities and its application to data mining (extended abstract). In *PODS '96: Proceedings of the fifteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 68–78, New York, NY, USA. ACM.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1-2):199–227.

Mannila, H. (2002). Local and Global Methods in Data Mining: Basic Techniques and Open Problems. In *ICALP '02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming*, pages 57–68, London, UK. Springer-Verlag.

Mörchen, F. and Fradkin, D. (2010). Robust mining of time intervals with semi-interval partial order patterns. In *Proceedings of the 10th SIAM International Conference on Data Mining*, pages 315–326. SIAM.

Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *ICDT'99: Proceedings of the 7th International Conference on Database Theory*, pages 398–416. Springer.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2004). Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440.

Piatetsky-Shapiro, G. and Frawley, W. J., editors (1991). *Knowledge Discovery in Databases*, volume 13. AAAI/MIT Press.

Roddick, J. and Spiliopoulou, M. (2002). A Survey of Temporal Knowledge Discovery Paradigms and Methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767.

Srikant, R. and Agrawal, R. (1996). Mining Sequential Patterns: Generalizations and Performance Improvements. In *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, pages 3–17, London, UK. Springer-Verlag.

Zaki, M. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60.