Enhancing sequential pattern mining with time and reasoning

Thomas Guyet

ANNÉE 2020

Habilitation Thesis / UNIVERSITÉ DE RENNES 1

Mention: Computer Science
Ecole doctorale MATHSTIC
defended by

Thomas Guyet

UMR CNRS 6974 – IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires

Enhancing sequential pattern mining with time and reasoning

Defensed in Rennes
on February the 10^{th}, 2020

Jury compostion :

Carlo COMBI
Professor, University of Verona / reviewer

Panagiotis PAPAPETROU
Professor, Stockholm University / reviewer

Chritel VRAIN
Professor, Univ Orlans / reviewer

Lakhdar SAIS
Professor, University of Artois / examiner

Alexandre TERMIER
Professor, University of Rennes / examiner

Contents

Chapter 1

Introduction

The general context of this work is the analysis of a complex dynamical system. Dynamical systems are systems which evolve in time, often in response to external stimulation or forcing. It may be a technical, physical, living or digital systems. A system is said *complex* while simple causal reasoning is not sufficient to predict its trajectory. More especially, we assume that it can not be formalized completely by a numeric or a symbolic model. But, it is not a chaotic system. In some conditions, the system behaves with some regularities and we are interested in discovering these regularities.

The analysis of a dynamical system aims at understanding its functioning. The objective of understanding a dynamical system is to acquire new knowledge, but it also has practical purposes. One might wish to predict its trajectory, to control it, to prevent from having faulty behavior or to improve its efficiency. Understanding is both a scientific challenge for people studying a system – the domain experts – but also an engineering need for developing tools in interaction with the system.

In our digital era, a revolution is the massive and continuous acquisition of heterogeneous data. It turns the problem of system understanding into data analytic problem. The functioning of a dynamical system is captured in digital traces and collected to form massive *logs*. Understanding a system is making sense from logs of its trajectories.

Then, **our main challenge is to empower domain experts with tools to make sense from logs of a dynamical system, such that they can get insight about its functioning.**

Figure 1.1 illustrates interactions between a (dynamical) system, event logs (temporal data) and an expert. Event logs are collected from a sensed system. Event logs are analyzed by an expert. The analysis of *logs* is mediated by (data analytics) tools. Expert has partial, not necessarily formalized knowledge about the system.

Let us start with few examples of this general context for some concrete applications. We illustrate event logs using a first order logic syntax. Literal `event(`I`,`t`,`e`)` represents an occurrence of event e at time t for the individual I.

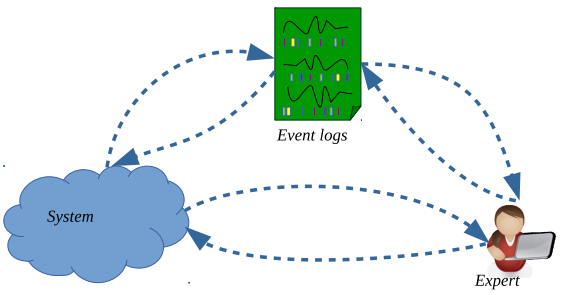


Figure 1.1: General framework: event logs are collected from a sensed system. Events logs are analyzed by an expert. Expert has knowledge about the system.

Example 1 (Understanding customer habits (C. Gautrais's PhD Thesis)). *In this context, the system is a customer which purchases articles in a supermarket. The event log is filled with purchases recorded by the loyalty card program. The listing below illustrates the event log of a client c_1. The customer came three times to the supermarket (at dates 0, 7 and 14). At date 7, he/she purchases three products (`"carrots"`, `"spinach"` and `"banana"` four times).*

```
event(c1, 0, product("banana", 1) ).
event(c1, 0, product("coffee", 3) ).
event(c1, 0, product("pasta", 2) ).
event(c1, 0, product("chocolate", 1) ).
event(c1, 7, product("spinach", 1) ).
event(c1, 7, product("carrots", 1) ).
event(c1, 7, product("banana", 4) ).
event(c1, 14, product("banana", 1) ).
event(c1, 14, product("chocolate", 1) ).
event(c1, 14, product("ice cream", 2) ).
```

The expert is a supermarket manager who aims at better understanding customer habits to recommend him/her products.

Example 2 (Pharmaco-epidemiology with medico-administrative databases (Y. Dauxais and J. Bakalara PhD Thesis)). *In this context, the system is a patient interacting with the healthcare system. The expert is an epidemiologist answering pharmaco-epidemiological questions. For instance, the epidemiologist studies the effect of anti-epileptic drug switches on epileptic seizure events. The final aim of the epidemiologist is to recommend guidelines for epileptic patient medication.*

The event log is made from medical events that are known for patients. The medical events may be collected for an ad hoc studies, but in our case we use medico-administrative databases (SNDS) which contain information about drug deliveries, hospital stays and medical procedure of patients. These events are timestamped and they give a longitudinal view on the patient.

The listing below illustrates the event log of a male patient p_1 who had four drugs deliveries and three medical procedures. At time 45, he received 2 drug boxes identified by the drug code 3723093 (Propranalol ratiopharm) then, at date 62, he had a medical procedure identified by the medical procedure code GAQE001 (nasal endoscopy).

```
sex(p1, 1).
birthYear(p1, 1979).
event(p1, 0, drugDelivery(3430126, 1) ).
event(p1, 3, procedure(ECGH011) ).
event(p1, 12, drugDelivery(4991401, 1) ).
event(p1, 23, drugDelivery(3588755, 2) ).
event(p1, 38, procedure(YYYY600) ).
event(p1, 45, drugDelivery(3723093, 2) ).
event(p1, 62, procedure(GAQE001) ).
```

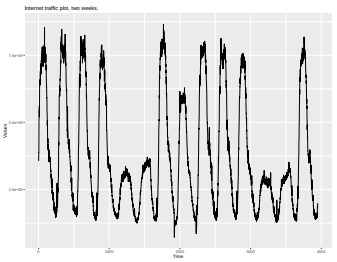
Compared to previous example, this data set contains different types of events in the event log. There are `drugDelivery` events which describe the delivery of drugs boxes, but other events such as `procedure` that indicate patient p_1 gets a medical procedure identified by a CCAM[1] code. In addition, medical procedures and drugs can be classified in standard taxonomies. These taxonomies are formalized background knowledge.

Example 3 (Capacity planning of servers (C. Leverger's PhD Thesis)). *In this applied context, the system is a server (or cluster of servers) which answers the requests from users. For instance, in the Orange Money service, the server answers request for money transfers. The expert is the manager that has to ensure the efficiency of the service. He/she has to decide whether the capacities of the server must be increased or decreased to fit the demand. Many "sensors" are available on technical systems: technical data (CPU usage, RAM memory usage, network usage, etc), functional data (number of applications, of threads, etc), operational data (number of requests per second, number of connected users, etc.).*

[1] CCAM: French Codification of Medical Acts.

Again, the collected data are timestamped. Compared to the previous examples, the values are sampled regularly. For each sensor, a piece of data with a given frequency. Such type of data are usually called time series.

```
event(m1, 0, CPU(1,56) ).
event(m1, 0, RAM(12) ).
event(m1, 1, CPU(1,52) ).
event(m1, 1, RAM(14) ).
event(m1, 2, CPU(1,58) ).
event(m1, 2, RAM(17) ).
event(m1, 3, CPU(1,65) ).
event(m1, 3, RAM(20) ).
```



The Figure on the right is a graphical representation of a multivariate time series. Each curve corresponds to one sensor.

Example 4 (Intrusion detection in web-server (Post-doc and N. Sourbier's PhD Thesis))**.** *In the context of intrusion detection of web servers, the system is a user interacting with a web server to protect. Understanding how intruders attack a server is the objective of information security.*

Intrusion detection system (IDS) monitor the local network activity through the analysis of massive connection logs. IDS implements complex system of rules to efficiently detect intrusion and prevent the infrastructure from attack. The difficulties is to feed the IDS with smart rules. And this is where data analytic can support experts. While attacks have been detected, logs must be analyzed (with a background knowledge about the infrastructure) to identify the intrusion signature that will witness future similar attacks.

Supporting sense making by a domain expert

According to De Bie and Spyropoulou [2013], today's data size and complexity call for extensive exploration stages by means of capable and intuitive exploratory data mining techniques, before predictive modeling or confirmatory analysis can realistically and usefully be applied.

Knowledge Discovery in Databases (KDD) is the research field that develops theories, techniques and systems for extracting and delivering useful knowledge out of large masses of data [Fayyad et al., 1996]. On the one side, knowledge discovery is a technology that blends data analysis methods (data mining, machine learning, algorithmic, statistics, visual analytic) that are dedicated to the automatic, efficient and almost agnostic extraction of patterns from the data. On the other side, knowledge delivery has the purpose to develop integrated tools that deliver the results of these algorithms in a way that supports the understanding of the data.

Exploratory Data Analysis (EDA) is a discipline that aims at designing integrated tools. It suggests to have an exploratory strategy which consists in investigating multiple data analysis tools to extract patterns, to identify anomalies and regularities in data without initial precise goal. The discovered patterns foster hypothesis developments and refinements which in turn invite to further investigations. It allows human users to make sense of data with little or no known data model. In Guyet [2007], we advocate for grounding principles of exploratory data analysis on results from cognitive science. More especially, Enaction [Froese and Ziemke, 2009] is a cognitive science paradigm that resonates with our intuition of Exploratory Data Analysis: 1) Enaction considers action (interaction with the world) as the basis of cognition; EDA intuitively requires interaction with data. 2) Enaction considers sense-making as a (never-ending) process which tries to reconcile the meaning of the world with the user knowledge; EDA intuitively engages the user in a succession of interactions with the data until interesting patterns are identified. 3) Enaction considers the process of sense-making as individual, situated; Some work of EDA tries to conceive tools which self-adapt from user experience.

This principle of thinking data analysis is not the heart of this manuscript, but it infuses our choices to conceive methods or tools to analyze data:

- ▷ our approach aims at supporting human understanding. Our problem is not to create an artificial intelligence that would "*understand*" but to propose tools and methods that support domain experts while they are exploring logs of their system. A recent concern of data

science is to have *interpretable* machine learning. The objective is to provide enough information about a model to make the user more confident in automatic decisions taken by this model. The problem is then to understand the model while our problem is to understand the data. But, we face similar challenges, and our method choice is guided by this objective of empowering domain experts with data analytic tools.

- ▷ there is not meaningful analysis without knowledge about the system from which data were collected and about how they were collected. This means that the domain expert keep an eye on the real system, and has strong background knowledge about it, even if this background knowledge is not necessarily formalized.

Focus on temporal data analysis

The above examples shed light on the importance that is given to the temporal dimension of event logs: most of the pieces of information are timestamped. Indeed, we are interested in dynamical systems and the temporal dimension is essential to accurately represents the trajectory of such systems.

Our ambition is not only to represent the temporal information of logs. But also to conceive tools that can exploit this dimension. This motivates our interest in temporal data analysis.

Temporal data mining is a branch of data mining that deals with temporal data [Laxman and Sastry, 2006; Mitsa, 2010]. The most common type of temporal data is time series data, which consists of real values sampled at regular time intervals. Time series analysis has quite a long history. Techniques for statistical modeling of real or time series have been in use for more than fifty years. But, temporal data may be nominal-valued or symbolic (rather than being real-valued). Such temporal data are so called sequences of events due to the intuitive ordering of data elements (events) by its timestamps. Sequences are encountered in a wide range of applications and need for dedicated analysis technique to cluster, classify or extract knowledge from them. Indeed, sequence are different from time series in the nature of datasets and the manner in which data are collected. These differences render the classical statistical time series techniques inoperable.

To explore event logs, our work investigates *sequential pattern mining* techniques. Sequential pattern mining is a subclass of pattern mining dedicated to sequential or temporal data. Pattern mining [Termier, 2013] is a class of techniques which aims at exhibiting structural correlations or causal relationships between structured objects (itemsets, sequences or graphs). These correlations are caught by patterns, i.e. a local structure that makes a specific statement about few variables or events. For example, a timestamped list of items bought by customers reveals which sets of items tend to be frequently consumed together, or whether a customer has some particular abnormal consumption patterns, etc. The combinations of items is a pattern. While the combination of items takes into account the sequential order of the data, it is called a sequential pattern. Due to their readily interpretable structure, patterns play a particularly dominant role in data mining. The interpretability of sequential patterns fits our objective to propose methods to support domain expert in their sense making from data.

Then, we advocate for the use of **sequential pattern mining to support domain experts to make sense from event logs.**

Challenges of sequential pattern mining

Sequential pattern mining [Fournier-Viger et al., 2017] (see Section A of Chapter 2) is studied since the early stages of pattern mining, several decades ago. Today, the state of the art algorithms can extract very efficiently sequential patterns from very large logs.

Nonetheless, there are potential improvements of sequential pattern mining:

Enhance temporal information in sequential patterns. A sequential pattern characterizes only the order of the events in logs. This poor temporal information may not be sufficient to characterize the functioning of a dynamical system. Some information about the delay between events, or the event duration potentially may improve the expressiveness of patterns. Chapter 3 is dedicated to the mining of metric temporal patterns from sequence of timestamped events. The main objective of chapter is to introduce new pattern domains that

augment classical patterns with metric temporal information. The temporal information is part of the pattern and is discovered during the mining process.

Exploit the expert knowledge to extract less but meaningful patterns. In some domain, experts may have knowledge about the system that informs about the usefulness of patterns. Then, the difficulty is to have pattern mining algorithms that can deal with expert knowledge. Constraints is a way to formalize expert knowledge. For instance, one might assume that a drug delivered at time t has no more effect at time $t + 20$. Then, situations involving drugs deliveries more distant in time that 20 days have not to be taken into account. Some pattern mining algorithms can deal with such pattern constraints. But, there are as many sequential pattern mining algorithms as there are types of constraint. This means that it is practically difficult to use.

Chapter 4 presents an original approach concerned by implementing sequential pattern mining in Answer Set Programming (ASP). ASP is a declarative programming language. This chapter illustrates that declarative programming (ASP, SAT, Constraint Programming) can solve a wide range of sequential pattern mining tasks and open the possibility to integrate expert knowledge easily.

Ease data preparation to make sequential pattern mining speak. In practice, the use of sequential pattern mining techniques requires to prepare raw data. Indeed, algorithms require sequences of items as input. But, in our examples, logs contain complex events. For instance, different types of events, `drugDeliveries/3` vs `acts/2`; or events with complex descriptions. A drug delivery is represented by the active principle (a class in a taxonomy), a quantity and a status. A data wrangling step is essential to prepare the data. And it is of particular importance to extract meaningful patterns.

Chapter 5 presents tools to support the preparation of temporal databases to the use of temporal pattern mining algorithms.

Our contributions are illustrated using the analysis of care pathway as a running example. Section B of Chapter 2 introduces the context of the example and details the rational of study we investigate through care pathway analysis.

Chapter 2

Preliminaries

In this preliminary chapter, we introduce sequential pattern mining and a case study of care pathway analysis. The first section introduces the definitions, notations, properties and algorithms that are the basis of sequential pattern mining. The notation that are introduced are used all along the manuscript. The second section presents the rational of a case study that we use to illustrate the proposals presented in next chapters.

A Sequential pattern mining in a nutshell

In this section, we introduce the sequential pattern mining task. In short, the sequential pattern mining problem consists in retrieving from a sequence database $\mathcal{D}$ every frequent non empty sequence $\boldsymbol{s}$, so-called a sequential pattern. Sequences, either in $\mathcal{D}$ or sequential patterns $\boldsymbol{s}$, are sequences of itemsets over a fixed alphabet of symbols (also called items). A pattern is frequent if it is a subsequence of at least f_{min} sequences of $\mathcal{D}$, where f_{min} is an arbitrary given threshold.

A.1 Notations and backgrounds

A.1.1 Sequential pattern mining

We introduce here the basic definitions of a sequence of itemsets. In the sequel, $[n] = \{1, 2, \ldots, n\}$ denotes the set of the n first integers:

Definition 1 (Sequences)**.** *Let $\mathcal{I}$ be the set of **items** (alphabet). An **itemset** $A = (a_1, a_2, ..., a_n) \subseteq \mathcal{I}$ is a subset of items from $\mathcal{I}$. A **sequence** $\boldsymbol{s}$ is an ordered set of itemsets $\boldsymbol{s} = \langle s_1\ s_2\ \ldots\ s_n \rangle$ such that $s_i \subseteq \mathcal{I}$ and $s_i \neq \emptyset$ for all $i \in [n]$. The length of sequence $\boldsymbol{s}$, denoted $|\boldsymbol{s}|$, is equal to its number of itemsets. Two sequences $\boldsymbol{s} = \langle s_1\ s_2\ \ldots\ s_n \rangle$ and $\boldsymbol{t} = \langle t_1\ t_2\ \ldots\ t_m \rangle$ are equals iff $n = m$ and $s_i = t_i$ for all $i \in [n]$.*

The set of items represents the alphabet on which sequences are built. Without loss of generality, a total order on $\mathcal{I}$ is classically defined. Such total order is useful for algorithmic optimization but not formally mandatory.

Definition 2 (Sub-sequence and embedding)**.** *Let $\boldsymbol{t} = \langle t_1\ t_2\ \ldots\ t_m \rangle$ and $\boldsymbol{s} = \langle s_1\ s_2\ \ldots\ s_n \rangle$ be two sequences. $\boldsymbol{t}$ is a **sub-sequence** of $\boldsymbol{s}$, denoted $\boldsymbol{t} \preceq \boldsymbol{s}$, iff there exists a sequence of integers $1 \leq i_1 < i_2 < ... < i_m \leq n$ such that $\forall k \in [m]$, $t_k \subseteq s_{i_k}$. In other words, $(i_k)_{1 \leq k \leq m}$ defines a mapping from $[m]$, the set of indexes of $\boldsymbol{t}$, to $[n]$, the set of indexes of $\boldsymbol{s}$. We denote by $\boldsymbol{t} \prec \boldsymbol{s}$ the strict sub-sequence relation such that $\boldsymbol{t} \preceq \boldsymbol{s}$ and $\boldsymbol{t} \neq \boldsymbol{s}$.*

*The sequence of integers $(i_k)_{k \in [m]}$ is also called an **embedding**.*

It is worth noticing that the sub-sequence relation does not enforce the itemsets of a subsequence $\boldsymbol{t}$ to occur consecutively in sequence $\boldsymbol{s}$ nor that itemsets must be equals. There are possible gaps between the occurrences of the itemsets of $\boldsymbol{t}$ in $\boldsymbol{s}$ and an itemset occurs as soon as it is a subset of an itemset of $\boldsymbol{s}$.

Definition 3 (Prefix). $\boldsymbol{t} = \langle t_1\ t_2\ \dots\ t_m \rangle$ *is a* ***prefix*** *of* $\boldsymbol{s} = \langle s_1\ s_2\ \dots\ s_n \rangle$*, denoted* $\boldsymbol{t} \preceq_b \boldsymbol{s}$*, iff* $\forall i \in [m-1],\ t_i = s_i$ *and* $t_m \subseteq s_m$*.*

In the notation of the prefix relation, b stands for "backward". Indeed, the prefix relation can be seen as a specific containment relation for which only the last itemset of a sequence may change. The Lemma 1 states that this relation is more specific than the sub-sequence relation.

Lemma 1. $\boldsymbol{t} \preceq_b \boldsymbol{s} \implies \boldsymbol{t} \preceq \boldsymbol{s}$*.*

Definition 4 (Containment relation / occurrence / support). *A sequence* $\boldsymbol{t}$ ***is supported*** *by a sequence* $\boldsymbol{s}$ *if* $\boldsymbol{t}$ *is a sub-sequence of* $\boldsymbol{s}$*, i.e.* $\boldsymbol{t} \preceq \boldsymbol{s}$*.*

The relation "*is supported by*" is the classical notion of pattern mining to specify that a sequence is contained into another sequence. We also say that a sequence occurs in another sequence.

At this stage, the containment relation indicates that a sequence occurs in a sequence, but not how many times it occurs. Indeed, a sequence can occur several times in another sequence. For instance, we could say that $\langle a\ b \rangle$ occurs twice in the sequence $\langle a\ b\ b \rangle$. In the classical definition of sequential pattern mining, the multiple occurrences are not considered. But in some variant of the sequential pattern mining task, this may raise some specific challenges (see Section A.4).

Example 5 (Sequences, subsequences and prefixes, embeddings). *Let* $\mathcal{I} = \{a, b, c\}$ *and the sequence* $\boldsymbol{s} = \langle a\ (bc)\ (abc)\ c\ b \rangle$*. To simplify the notation, parentheses are omitted around itemsets containing only one item. The length of* $\boldsymbol{s}$ *is 5.* $\boldsymbol{t_1} = \langle a\ b\ b \rangle$*,* $\boldsymbol{t_2} = \langle (bc)\ (ac) \rangle$ *or* $\boldsymbol{t_3} = \langle a\ (bc)\ (abc)\ c\ b \rangle$ *are sub-sequences of* $\boldsymbol{s}$*.* $\langle a \rangle$*,* $\langle a\ (bc) \rangle$ *and* $\langle a\ (bc)\ a \rangle$ *are prefixes of* $\boldsymbol{s}$*.*

The embeddings of $\boldsymbol{t_1}$ *in* $\boldsymbol{s}$ *are* $\langle 1, 2, 3 \rangle$*,* $\langle 1, 3, 5 \rangle$ *or* $\langle 1, 2, 5 \rangle$*, and the unique embedding of* $\boldsymbol{t_2}$ *in* $\boldsymbol{s}$ *is* $\langle 2, 3 \rangle$*.*

Definition 5 (Support measure). *Let* $\mathcal{D} = \{\boldsymbol{s}_1, \boldsymbol{s}_2, \dots, \boldsymbol{s}_N\}$ *be a set of* N *sequences.* $\mathcal{D}$ *is called a* ***sequence database****. The* ***support of a sequence*** $\boldsymbol{s}$ *in* $\mathcal{D}$*, denoted by* $supp_{\mathcal{D}}(\boldsymbol{s})$*, is the number of sequences of* $\mathcal{D}$ *that support* $\boldsymbol{s}$*:*

$$supp_{\mathcal{D}}(\boldsymbol{s}) = |\{\boldsymbol{s}_i \in \mathcal{D} | \boldsymbol{s} \preceq \boldsymbol{s}_i\}|$$

Definition 6 (Frequent sequential pattern mining). *Let* σ *be a* ***minimal support threshold****. For any sequence* $\boldsymbol{s}$*, if* $supp_{\mathcal{D}}(\boldsymbol{s}) \geq \sigma$*, we say that* $\boldsymbol{s}$ *is a* ***frequent sequence*** *or a* ***(frequent) sequential pattern*** *of* $\mathcal{D}$*.*

Mining frequent sequential patterns *consists in extracting all and only frequent subsequences in a sequence database* $\mathcal{D}$*.*

σ *is a parameter of frequent sequential pattern mining defined by the analyst.*

From the algorithmic point of view, we are interested in complete and correct algorithms. This means that an algorithm has to extract all the sequential patterns that are frequent in the database (completeness) and that each outputted sequential pattern must actually be frequent (correctness).

Example 6 (Sequential pattern mining). *To illustrate the concepts introduced above, we consider the following sequence database* $\mathcal{D}$ *containing sequences built on items in* $\mathcal{I} = \{a, b, c, d\}$*.*

Id	*Sequence*
$\boldsymbol{s_1}$	$\langle a\ c \rangle$
$\boldsymbol{s_2}$	$\langle a\ b\ (bc)\ c \rangle$
$\boldsymbol{s_3}$	$\langle d\ b \rangle$
$\boldsymbol{s_4}$	$\langle a\ (bc)\ c \rangle$
$\boldsymbol{s_5}$	$\langle a\ b\ d \rangle$
$\boldsymbol{s_6}$	$\langle a\ c\ b\ c \rangle$
$\boldsymbol{s_7}$	$\langle a\ (bc)\ c \rangle$

Given a minimal support threshold $\sigma = 3$*, the frequent sequential patterns are:* $\langle a \rangle$*,* $\langle b \rangle$*,* $\langle c \rangle$*,* $\langle a\ c \rangle$*,* $\langle b\ c \rangle$*,* $\langle (bc)\ c \rangle$*,* $\langle a\ b \rangle$*,* $\langle a\ (bc) \rangle$*,* $\langle a\ b\ c \rangle$*, and* $\langle a\ (bc)\ c \rangle$*.*

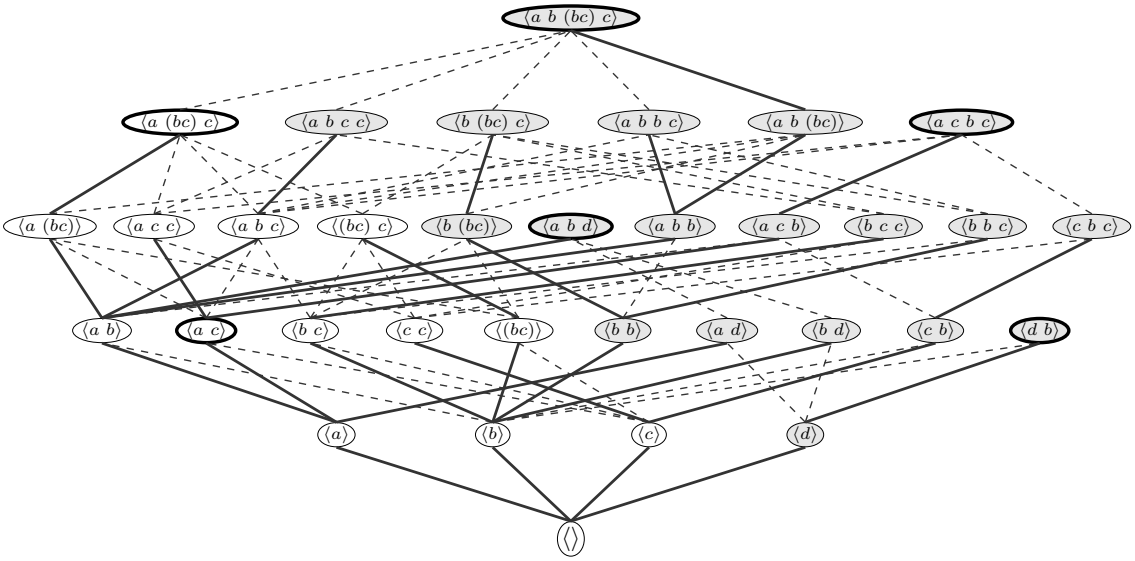


Figure 2.1: Hasse diagram of the lattice of sequential patterns occurring at least once in the data set. Lines between nodes represent the inclusion relation. Plain lines are prefix relations ($\preceq_b$), and dashed lines are non-prefix inclusion relations ($\preceq$).

A.1.2 Structure on sequences and anti-monotonicity of the support

Let us now denote $\mathcal{L}$ the set of sequences (sequential patterns) on the vocabulary $\mathcal{I}$. Without sequence length boundary $\mathcal{L}$ is an infinite set. In practice, it can be bound by the length of the longest sequence of the dataset.

Proposition 1 (Partial orders)**.** *$(\mathcal{L}, \preceq)$ and $(\mathcal{L}, \preceq_b)$ are posets (partially ordered sets).*

Proposition 2 (Infinite semi-lattice of sequences)**.** *$(\mathcal{L}, \preceq)$ and $(\mathcal{L}, \preceq_b)$ are infinite semi-lattices.*

In the following, we denote $(\mathcal{L}, \lhd)$ the set of patterns structures with the partial order $\lhd \in \{\preceq, \preceq_b\}$. The symbol $\lhd$ is to be read as "is less specific than (or as specific as)".

Being a semi-lattice means that any subset of $\mathcal{L}$ has a least general element. Next section will show that the lattice structure is at the origin of the sequential pattern mining algorithms.

Example 7 (Lattice of sequential patterns)**.** *In this example, the Figure 2.1 illustrates the lattice of sequential patterns in the dataset of sequences of Example 6. The lattice illustrates only the sequences that occur at least once in the data set. Surrounded nodes are elements of the dataset. Lines gives comparable sequences with respect to the $\preceq$ binary relation. The grey nodes are infrequent sequential pattern ($\sigma = 3$).*

One can notice that this semi-lattice has the empty sequence as bottom, but does not has a unique top: $\langle d\ b \rangle$, $\langle a\ b\ d \rangle$, $\langle a\ b\ (bc)\ c \rangle$ and $\langle a\ c\ b\ c \rangle$.

Proposition 3 (Anti-monotonicity of the support)**.** *$supp_{\mathcal{D}}(\cdot)$ is an anti-monotonic measure on the set of sequences structured by $\preceq$ or $\preceq_b$.*

This proposition implies that for all pairs of sequences $(\boldsymbol{p}, \boldsymbol{q})$:

$$\boldsymbol{p} \preceq \boldsymbol{q} \implies supp_{\mathcal{D}}(\boldsymbol{p}) \geq supp_{\mathcal{D}}(\boldsymbol{q}), \text{ and}$$

$$\boldsymbol{p} \preceq_b \boldsymbol{q} \implies supp_{\mathcal{D}}(\boldsymbol{p}) \geq supp_{\mathcal{D}}(\boldsymbol{q}).$$

A.2 Efficient sequential patterns mining algorithms

Now we defined sequential pattern mining, we look at how to extract the frequent sequential patterns. Algorithm 1 illustrates the naive approach of sequential pattern mining. This algorithm

iterates over the set of sequential patterns and evaluates the support of each pattern. It is complete and correct, but not time efficient. Indeed, considering only sequence of items, the time complexity of the algorithm is $\mathcal{O}\left(|\mathcal{D}| \times \left(2^{|\mathcal{I}|}\right)^{l}\right)$ where l represents the maximum length of sequential patterns to consider (maximum length of sequences in $\mathcal{D}$). Thus, there are $\left(2^{|\mathcal{I}|}\right)^{l}$ potential sequential patterns[1] and for each candidate, the algorithm evaluates its support to know whether it is frequent of not.

Algorithm 1: Naive algorithm for sequential pattern mining.

input : $\mathcal{D}$: set of sequences, σ: minimum support threshold
output: $\mathcal{FP}$: set of frequent sequential patterns

```
1 FP ← ∅;
2 for p ∈ L do
3     sup ← supp_D(p);
4     if sup ≥ σ then
5         FP ← FP ∪ {p};
```

Example 8 (Ex.7 (cont.))**.** *In the case of Example 7, the maximal sequence length is 5, the maximal itemset size is 2 and* $|\mathcal{I}| = 4$*. Then, the number candidate patterns is* $5 \times (4 \times 3) = 60$*. For each pattern, the database will be browsed to evaluate the support.*

A.2.1 Pattern mining trick

The success of the field of pattern mining comes from an anti-monotonicity property that has been introduced by Agrawal and Srikant [1995]. In principle, **a pattern occurs more than its more specific patterns**. Based on this intuitive idea, the pattern mining trick consists in pruning the search space (candidate patterns) using infrequent patterns. As soon as a sequential pattern is infrequent, any of its more specific pattern may be frequent.

To apply the pattern mining trick, two properties are required:

- ▷ the search space, i.e. the set of sequential patterns, must be structured with a partial order, denoted $(\mathcal{L}, \lhd)$ in the following.
- ▷ the support measure must effectively be anti-monotonic on the $(\mathcal{L}, \lhd)$, i.e. for all pattern $\boldsymbol{p}$ and $\boldsymbol{q}$:

$$\boldsymbol{p} \lhd \boldsymbol{q} \implies supp_{\mathcal{D}}(\boldsymbol{p}) \geq supp_{\mathcal{D}}(\boldsymbol{q})$$

This strategy is illustrated by Algorithm 2. Compared to the previous algorithm, this algorithm evaluates (line 4) a candidate sequential pattern $\boldsymbol{p}$ with respect to the set of infrequent patterns. If a less specific sequential pattern $\boldsymbol{q}$ is known to be infrequent, then $\boldsymbol{p}$ is a priori infrequent and it prevents from evaluating the support. The larger the dataset, the more time-efficient. Nonetheless, the existential test of line 4 may be costly because of the size of $\mathcal{IP}$ (infrequent patterns). In case $\mathcal{IP}$ exceeds the size of the dataset, the computation cost of evaluating the support is lower than the cost of checking subsequence with infrequent patterns. In addition, the efficiency of the approach is strongly dependent on the order in which the set of candidate is traversed.

In practice, the principle of a pattern mining algorithm is to explore the search space of candidate sequential patterns from the less specific to the more specific, i.e. from the shortest sequences to the largest. Identifying a small infrequent sequence prunes a large number of patterns.

Example 9 (Ex.7 (cont.))**.** *Let us continue the Example 7 for which we know that* $\preceq_b$ *is a partial order on the set of sequential patterns (plain lines in Figure 2.1).*

Let us assume that the exploration order of patterns by algorithm is $\boldsymbol{p_1} = \langle c \rangle$, $\boldsymbol{p_2} = \langle c\ b \rangle$, $\boldsymbol{p_3} = \langle c\ c\ b \rangle$, $\boldsymbol{p_4} = \langle a\ c\ b \rangle$ *... Initially,* $\mathcal{FP} = \emptyset$ *and* $\mathcal{IP} = \emptyset$*. While* $\boldsymbol{p_1}$ *is evaluated (line 7), the pattern is frequent (white node in Figure 2.1) and then* $\mathcal{FP} = \{\boldsymbol{p_1}\}$*. Then,* $\boldsymbol{p_2}$ *is processed,* $\mathcal{IP} = \emptyset$ *so the support of* $\boldsymbol{p_2}$ *is evaluated line 7. It is infrequent and thus* $\mathcal{IP} = \{\boldsymbol{p_2}\}$*. Then,* $\boldsymbol{p_3}$ *is*

[1] $2^{|\mathcal{I}|}$ is the number of itemsets.

Algorithm 2: Sequential pattern mining algorithm based on anti-monotonicity property.

input : $\mathcal{D}$: set of sequences, σ: minimum support threshold
output: $\mathcal{FP}$: set of frequent sequential patterns

```
1  FP ← ∅;
2  IP ← ∅ //Infrequent patterns
3  for p ∈ L do
4      if ∃q ∈ IP, q ◁ p then
5          IP ← IP ∪ {p};
6          continue
7      sup ← supp_D(p);
8      if sup ≥ σ then
9          FP ← FP ∪ {p};
10     else
11         IP ← IP ∪ {p};
```

processed, and we have that $\boldsymbol{p_2} \preceq_b \boldsymbol{p_3}$*. Thus, we know a priori (Proposition 3) that* $\boldsymbol{p_3}$ *is infrequent without evaluating its support and* $\mathcal{IP} = \{\boldsymbol{p_2}, \boldsymbol{p_3}\}$*. Then,* $\boldsymbol{p_4}$ *is processed, but this time* $\boldsymbol{p_2} \preceq_b \boldsymbol{p_4}$ *nor* $\boldsymbol{p_3} \preceq_b \boldsymbol{p_4}$ *and the pattern* $\boldsymbol{p_4}$ *is evaluated.*

We can notice that in the case where $(\mathcal{L}, \lhd)$ is a poset, it seems difficult to improve the algorithm while keeping its completeness. Possible improvements can be proposed in case each pattern is *accessible* from the bottom.

A.2.2 Accessibility of patterns

If for any pattern $\boldsymbol{p}$ there is at least one path in $(\mathcal{L}, \lhd)$ from it to pattern $\bot = \langle\rangle$ (i.e. the most general pattern), then algorithms can be improved without loosing completeness.

In Example 7, we can see that for any node, there are lines from it to the bottom node. In the case of the $\preceq$, there are multiple paths using plain and dashed lines. But while considering $\preceq_b$ (plain lines) there is a unique path from any node to the bottom.

Definition 7 (Accessibility relation)**.** *An accessibility relation* $\rightsquigarrow$ *on the set of patterns* $(\mathcal{L}, \lhd)$ *is such that*

1. $\boldsymbol{p} \rightsquigarrow \boldsymbol{q} \implies \boldsymbol{p} \lhd \boldsymbol{q}$ *, for all* $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{L}$
2. $\langle\rangle \rightsquigarrow^* \boldsymbol{p}$*, for all* $\boldsymbol{p} \in \mathcal{L}$

where $\rightsquigarrow^*$ *denotes the transitive accessibility between patterns:* $\boldsymbol{q} \rightsquigarrow^* \boldsymbol{p}$ *iff there is a sequence of patterns* $(\boldsymbol{q_i})_{i \in [m]}$ *such that* $\boldsymbol{q_i} \rightsquigarrow \boldsymbol{q_{i+1}}$ *for all* $i \in [m-1]$ *and* $\boldsymbol{q_1} = q$ *and* $\boldsymbol{q_m} = \boldsymbol{p}$*.*

Lemma 2 (Decreasing paths)**.** *The support measure is decreasing along accessibility path.*

Such accessibility relations are interesting to propose a strategy to traverse the search space. The algorithm 3 illustrates it. The algorithm uses a set of candidate patterns, denoted $\mathcal{C}$. While $\mathcal{C}$ is not empty, each of its element $\boldsymbol{p}$ is evaluated (line 5). If $\boldsymbol{p}$ is frequent then it is added to the frequent patterns and all its accessible patterns are added to the set of new candidate patterns to explore. If it is not frequent, then any of its accessible patterns are a priori frequent (Lemma 2). Thus, they are not added to the new candidates.

This algorithm is complete. More especially, while patterns are discarded from the new candidate patterns, $\mathcal{N}$, we know a priori that these discarded patterns are not frequent. In addition, by definition of the accessibility relation, there is a path to access any pattern form $\mathcal{L}$. The correctness of the algorithm is ensured by Line 7 which enforces any patterns of $\mathcal{FP}$, appended Line 8, to be frequent.

Let us do some more remarks about this algorithm and the accessibility relation.

- ▷ Using $\lhd$ as accessibility relation is equivalent to the naive algorithm (Algorithm 1). Indeed, since $\langle\rangle = \bot \lhd \boldsymbol{q}$ for all sequence $\boldsymbol{p}$ then $\{\boldsymbol{q} \in \mathcal{L} \mid \langle\rangle \lhd \boldsymbol{q}\} = \mathcal{L}$. As the algorithm starts to evaluate the pattern $\langle\rangle$, then $\mathcal{N} = \mathcal{L}$ after the first while loop.

Algorithm 3: Sequential pattern mining algorithm using accessibility relation.

input : $\mathcal{D}$: set of sequences, σ: minimum support threshold
output: $\mathcal{FP}$: set of frequent sequential patterns

```
1  FP ← ∅;
2  C ← {⟨⟩} //Candidate patterns
3  while C ≠ ∅ do
4      for p ∈ C do
5          sup ← supp_D(p);
6          N ← ∅ //New candidate patterns
7          if sup ≥ σ then
8              FP ← FP ∪ {p};
9              N ← N ∪ {q ∈ L | p ⇝ q};
10         else
11             N ← N \ {q ∈ L | p ⇝ q};
12     C ← N;
```

- ▷ Compute the set of accessible patterns may be difficult in the general case ($\{\boldsymbol{q} \in \mathcal{L} \mid \boldsymbol{p} \rightsquigarrow \boldsymbol{q}\}$). Accessibility relations that enable to evaluate easily this set are preferred.
- ▷ Lines 10-11 are not mandatory but enable to discard from the candidate patterns, $\mathcal{N}$, the pattern $\boldsymbol{q}$ generated from a frequent pattern $\boldsymbol{p}$ for which there is another pattern $\boldsymbol{p}'$ that directly accesses $\boldsymbol{q}$ and that is not frequent. For instance, the pattern $\langle a\ b\ b\rangle$ in Figure 2.1 is added to the new candidates by $\langle a\ b\rangle$, but is discarded by the infrequent less specific pattern $\langle b\ b\rangle$.
  The processing order of the patterns has impact of the capability of this algorithm to discard some patterns.

Then, the idea of the accessibility relation is to propose a relation with a minimal change from one pattern to the other. In the case of the sequential patterns, a minimal change is a difference of one item between the two patterns.

We use each partial order ($\preceq$ and $\preceq_b$) to induce an accessibility relation.

Definition 8 (Direct inclusion relation)**.** *Let $\boldsymbol{p} = \langle p_1\ \ldots\ p_n\rangle$ and $\boldsymbol{q} = \langle q_1\ \ldots\ q_m\rangle$ be two sequential patterns then $\boldsymbol{p} \rightsquigarrow_i \boldsymbol{q}$ iff:*

- ▷ *(Combination) $m = n$ and there exists $k \in [m]$ such that $p_i = q_i$ for all $i \in [m] \setminus \{k\}$, $p_k \subset q_k$ and $|p_k| = |q_k| - 1$*
- ▷ *(Extension) $m = n + 1$, $p_i = q_i$ for all $i \in [n]$ and $|q_m| = 1$*

Definition 9 (Direct backward relation)**.** *Let $\boldsymbol{p} = \langle p_1\ \ldots\ p_n\rangle$ and $\boldsymbol{q} = \langle q_1\ \ldots\ q_m\rangle$ be two sequential patterns then $\boldsymbol{p} \rightsquigarrow_b \boldsymbol{q}$ iff:*

- ▷ *(Combination) $m = n$, $p_i = q_i$ for all $i \in [m-1]$, $p_m \subset q_m$ and $|p_k| = |q_k| - 1$*
- ▷ *(Extension) $m = n + 1$, $p_i = q_i$ for all $i \in [n]$ and $|q_m| = 1$*

In the two proposed relations, the combination relation consists in considering that a sequence $\boldsymbol{q}$ differs from $\boldsymbol{p}$ on itemset k, for which $\boldsymbol{q}$ contains one more item. For instance, $\langle a\ b\ c\rangle \rightsquigarrow_i \langle a\ (bd)\ c\rangle$. In the case of the direct backward relation, the combination is only possible with the last itemset.[2] The previous example does not satisfy the $\rightsquigarrow_b$ relation, but $\langle a\ b\ c\rangle \rightsquigarrow_b \langle a\ b\ (cd)\rangle$.

The extension is the case in which a sequence $\boldsymbol{p}$ is a prefix of $\boldsymbol{q}$ with only one more itemset for $\boldsymbol{q}$. The last itemset containing only one item. For instance, $\langle a\ b\ (cd)\rangle \rightsquigarrow_i \langle a\ b\ (cd)\ b\rangle$, and $\langle a\ b\ (cd)\rangle \rightsquigarrow_b \langle a\ b\ (cd)\ b\rangle$.

In Figure 2.1, edges represent the direct relations.

Lemma 3 (Accessibility relations)**.** *$\rightsquigarrow_i$ and $\rightsquigarrow_b$ are accessibility relations.*

[2] In addition, it is usual to define a total order on $\mathcal{I}$. Then, the combination enforces the last item to be greater than the greatest element of the itemset. For sake of simplicity, we do not add this additional complexity here.

The main advantage of these two accessibility relations is that it is easy to compute the set of directly accessible sequential patterns ($\{\boldsymbol{q} \in \mathcal{L} \mid \boldsymbol{p} \rightsquigarrow \boldsymbol{q}\}$). The two relations can then be used to design a frequent sequential pattern mining algorithm based on Algorithm 3. The algorithmic complexity (in the worse case) would unfortunately be the same as the naive algorithm[3] but, in practice it is much more efficient.

The differences between the two relations are:

- ▷ the set of directly accessible sequential patterns generated by $\rightsquigarrow_b$ is smaller than the one generated by $\rightsquigarrow_i$. This means that there will be potentially less candidate patterns to evaluate with $\rightsquigarrow_b$. For instance, in Figure 2.1, we can see that there are each much more $\rightsquigarrow_i$ (dashed lines) than $\rightsquigarrow_b$ (plain lines) relations between patterns. As a consequence, patterns like $\langle a\ c\ b\ c\rangle$ would a priori not be evaluated with $\rightsquigarrow_b$ as its only backward ancestor is infrequent, but it will be evaluated while considering $\rightsquigarrow_i$.
- ▷ the $\rightsquigarrow_i$ relation may potentially discard patterns more quickly. For instance, let us consider the pattern $\langle a\ b\ d\rangle$ in Figure 2.1: the support of this pattern will be evaluated while using $\rightsquigarrow_b$ because a new candidate generated by the pattern $\langle a\ b\rangle$. But, while using $\rightsquigarrow_i$ and because one of its ancestor ($\langle a\ d\rangle$) is not frequent, we can a priori discard $\langle a\ b\ d\rangle$ from the candidates using lines 10-11 of Algorithm 3. More especially, these lines are useful in the case of the backward relation as there is a unique direct backward accessible pattern.

In conclusion, accessibility relations comparing lot of elements (e.g. $\rightsquigarrow_i$) tend to explore patterns that are never accessed by accessibility relations comparing few elements (e.g. $\rightsquigarrow_b$), but because it exploits more inclusion relations, the former relations tend to discard a priori patterns that the latter relations have to evaluate. Both strategies are complete and correct. Then, the sets of extracted patterns are exactly the same. The way the search space is traversed impacts the efficiency of the algorithm. But there is no ideal strategy. It also depends on the characteristics of the dataset.

To sum up, the principle of pattern mining algorithms, illustrated in this case with sequential pattern mining, is based on three important aspects of the frequent pattern mining problem:

- ▷ the pattern domain $(\mathcal{L}, \lhd)$ is a poset.
- ▷ an accessibility relation ($\rightsquigarrow$) defines the strategy to traverse the search space and ensures the completeness of the algorithm. This accessibility relation is induced by $\lhd$.
- ▷ the measure of interestingness – the support measure in this case – is anti-monotonic on $(\mathcal{L}, \lhd)$ to ensure the correctness of the pruning strategy.

While a pattern domain and a measure on interestingness holds these properties, the state of the art pattern mining algorithms can be directly adapted to propose new correct, complete and *efficient* pattern mining algorithms. The next section reviews the most used sequential pattern mining algorithms. For alternative pattern domain, the real efficiency of the algorithms depends on the search space itself (and on the support measure). The more complex pattern domains lead to huge search spaces that, despite the efficient traversal strategy, would require to output a lot of patterns.[4]

Finding new expressive pattern domains is at the cross-road between:

- ▷ the analyst demand: a new pattern domain is interesting if it answers analyst needs that classical pattern domains can not.
- ▷ the requirement of theoretical properties: the pattern domains must have properties that fit the requirements of the frequent pattern mining framework presented above
- ▷ the need for practical algorithms: in some cases, these properties are not strong enough or does not hold. In such case, practical solutions are required.

In this section, the measure of interestingness of a pattern is the number of occurrences of a pattern in a dataset of sequences. This is the most encountered setting. But it is possible to propose alternative measure of interestingness. In Section A.3.3, we present a measure of support in a stream of itemsets.

Remark 1 (Semi-lattice)**.** *It is worth noticing that* $(\mathcal{L}, \lhd)$ *does not need to be a semi-lattice. This*

[3] The worse case is when the dataset contains n identical sequences of itemset equals to $\mathcal{I}$, $n \leq \sigma$.

[4] The best pattern mining algorithm, LCM for itemset mining Uno et al. [2005], achieves output polynomial complexity, meaning that the delay between two successive outputted patterns is polynomial.

property is necessary while dealing with formal concepts analysis framework, but not for the anti-monotonicity property and breadth first search of depth first search algorithms. Note that a pattern domain that enjoys being a semi-lattice has the accessibility property.

A.2.3 Sequential pattern mining algorithms

There are several proposed methods for sequential pattern mining [Agrawal and Srikant, 1995; Srikant and Agrawal, 1996a; Han et al., 2000; Pei et al., 2001; Zaki, 2001]. One of the first algorithms ever proposed for solving this particular type of problems is the **GSP algorithm**, proposed by Srikant and Agrawal [1996b] (GSP: Generalized Sequential Patterns). This algorithm was proposed by the same authors that proposed the A-Priori algorithm [Agrawal and Srikant, 1994] for itemsets mining, so the GSP algorithm is in fact a continuation of their work. GSP has the same A-Priori strategy for attacking the problem of sequential pattern mining.

However, the GSP algorithm has some limitations. Han et al. [2000] and Pei et al. [2001] explain that there are three main problems with the APriori-like sequential pattern mining algorithms. First, the candidate generation strategy: if we have a large sequence database, there is a risk of generating a huge set of candidate sequences. Second, the algorithm scans the database too many times. Third, when we have long sequential patterns, they grow from a very large number of smaller sequences, and the number of such candidate sequences is exponential with the length of the sequential patterns to be mined. Because of these limitations, numerous alternative approaches were proposed. Among the most popular ones are SPADE and PrefixSpan.

The **SPADE** (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) algorithm was proposed by Zaki [2001]. One of the most important features in SPADE is the database layout, which is transformed from horizontal into vertical id-list database format in this algorithm. Another important principle is the *equivalent classes*. With them, the search space is decomposed into smaller pieces. Then, these pieces can be processed in the memory independently. Within each equivalence class is used depth-first search for enumerating the frequent sequences. According to the experiments presented by Zaki [2001], SPADE outperforms GSP by a factor of two.

Han et al. propose a new method for sequential pattern mining, which they call **FreeSpan** (**Fre**quent pattern-projected **S**equential **pa**ttern mining) in [Han et al., 2000]. Instead of the APriori-like candidate-generation strategy, the authors propose a completely different way of tackling the problem. As they explain, *its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database* [Han et al., 2000]. It corresponds to a depth-first search strategy instead of a breath-first search strategy used by GSP. According to the results [Han et al., 2000], FreeSpan is considerably faster than GSP. After this proposal, part of the same group of authors, proposes **PrefixSpan** (**Prefix**-projected **S**equential **pa**ttern mining) in [Pei et al., 2001], which, bottom-line, is an improved version of FreeSpan. As the authors explain in [Pei et al., 2001] *its general idea is to examine only the prefix subsequences and project only their corresponding postifx subsequences into projected databases*, while in FreeSpan the projection is done on the whole sequence, thus making it less efficient. According to a comprehensive performance study presented in [Pei et al., 2004], the authors point out that PrefixSpan, in most cases, outperforms GSP, FreeSpan and SPADE.

A.3 Reducing the number of patterns

With two decades of research on sequential pattern mining and the increase of computation power, sequential pattern mining algorithms are now able to process huge databases of sequences with low minimum frequency thresholds. The problem turns from the deluge of data ... to the deluge of patterns. And recent studies have been more focused on the mining of less but meaningful patterns.

In this section, we present in this section several classical approaches that have been proposed to reduce the number of sequential patterns: condensed representation of patterns, user-specified constraints on the patterns and sampling techniques. These approaches are smarter than just a post-processing of extracted patterns. They prevent early to generate patterns that are useless according to their specific definitions.

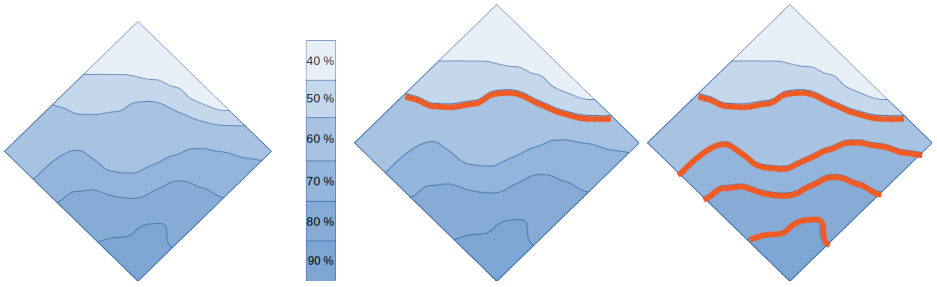


Figure 2.2: From left to right: Illustration of the lattice of patterns $((\mathcal{L}, \lhd))$ with decreasing pattern support from bottom to top (cf Prop 3). Illustration of maximal patterns (bold-orange line) extracted with minimal frequency of 60%. Illustration of closed patterns (bold-orange line) extracted with minimal frequency of 60%.

A.3.1 Condensed sequential pattern mining

Condensed patterns are subsets of the frequent patterns that are considered as representative of the whole content of the frequent pattern. Two types of condensed patterns have been defined: the closed and the maximal patterns. A closed pattern is such that none of its frequent super-patterns has the same support. A maximal pattern is such that none of its super-patterns is frequent.

Definition 10 (Maximal pattern)**.** *A frequent pattern $\boldsymbol{s}$ is **maximal** (resp. **backward-maximal**) with respect to the relation $\preceq$ (resp. $\preceq_b$) iff there is no other frequent pattern $\boldsymbol{s}'$ such that $\boldsymbol{s} \preceq \boldsymbol{s}'$ (resp. $\boldsymbol{s} \preceq_b \boldsymbol{s}'$).*

Definition 11 (Closed patterns)**.** *A frequent pattern $\boldsymbol{s}$ is **closed** (resp. **backward-closed**) with respect to the relation $\preceq$ (resp. $\preceq_b$) iff there is no proper superpattern $\boldsymbol{s}'$ such that $\boldsymbol{s} \preceq \boldsymbol{s}'$ (resp. $\boldsymbol{s} \preceq_b \boldsymbol{s}'$) and $supp(\boldsymbol{s}) = supp(\boldsymbol{s}')$.*

Mining closed patterns significantly reduces the number of patterns without loss of information for the analyst. Having closed patterns and their support, the support of any frequent pattern can be computed. This is not the case for maximal patterns and it explains why closed patterns are of particular interest.

Figure 2.2 illustrates the maximal and closed patterns. Classically, the search space (a lattice) is represented by a diamond (cf. Figure 2.1 for a concrete lattice of sequential patterns). And the anti-monotonicity of the support ensures that the support decreases from bottom ($\bot$ pattern) to top (largest patterns). The problem of frequent pattern mining is to identify the shape of the boundary between frequent and infrequent patterns. While extracting the complete set of frequent pattern, the problem is to identify the patterns that are below the boundary. While extracting the maximal patterns, the problem consists simply to identify patterns on the boundary. And the problem of closed frequent patterns is to identify the patterns that are on the boundaries of patterns regions with same support.

Example 10 (Maximal and closed-sequential pattern mining)**.** *Considering the data set of Example 6, among the frequent patterns with $f_{min} = 3$, the only maximal pattern is $\langle a\ b\ c\rangle$. The set of backward-maximal is $\{\langle c\rangle, \langle b\ c\rangle, \langle a\ c\rangle, \langle a\ b\ c\rangle\}$.*

The set of closed patterns is $\{\langle a\rangle, \langle b\rangle, \langle a\ b\rangle, \langle a\ c\rangle, \langle a\ b\ c\rangle\}$. $\langle b\ c\rangle$ is not closed because in any sequence it occurs, it is preceded by an a. Thus $supp(\langle b\ c\rangle) = supp(\langle a\ b\ c\rangle) = 4$.

The set of backward-closed patterns is $\{\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle b\ c\rangle, \langle a\ c\rangle, \langle a\ b\ c\rangle\}$. $\langle b\ c\rangle$ is backward-closed because not any pattern $\langle b\ c\ ?\rangle$ is frequent.

The mining of maximal patterns are simply based on the classical sequential pattern mining algorithms adapted to output only the most maximal ones. More research efforts have been done on closed patterns that are more challenging to extract, but that are known to represent the exact same information as the total set of frequent patterns (the frequency of all frequent patterns can be computed from the set of (backward-)closed patterns). On the other hand, the LCM algorithm

[Uno et al., 2005] proved the efficiency of mining strategy based on closed itemset patterns. Several algorithms have been proposed to extract efficiently closed sequential patterns. The most known approach are CloSpan [Yan et al., 2003] and BIDE [Wang and Han, 2004]. The second adopts a novel sequence closure checking scheme based on a bi-directional extension of the patterns. Finally, a sequential version of LCM is also available [Uno, 2004].

A.3.2 Constrained sequential pattern mining

The principle of constrained sequential pattern mining is to specify additional constraints on the expected patterns [Zaki, 2000]. The more constrained are the patterns, the less they are. To be efficiently processed by pattern mining algorithms, such patterns have to be anti-monotonic[5].

Pei et al. [2007] defined seven types of constraints on patterns and embeddings:

- ▷ **Constraint 1 – item constraint.** An item constraint specifies what are the particular individual or groups of items that should or should not be present in the patterns. For instance, the constraint "patterns must contain item 1 but not item 2 nor item 3".
- ▷ **Constraint 2 – length constraint.** A length constraint specifies a prerequisite on pattern length. The maximal length constraint is anti-monotonic while the minimal length is not anti-monotonic.
- ▷ **Constraint 3 – super-pattern constraint.** A super-pattern constraint enforces the extraction of patterns that contain one or more given sub-patterns.
- ▷ **Constraint 4 – aggregate constraint.** An aggregate constraint is a constraint on an aggregation of items in a pattern, where the aggregate function can be *sum*, *avg*, *max*, *min*, *standard deviation*, etc.
- ▷ **Constraint 5 – Regular expression.** Such a constraint is satisfied if the pattern is an accepted regular expression as stated by the user. SPIRIT [Garofalakis et al., 1999] is one of the rare algorithms that considers complex pattern constraints expressed as regular expressions.
- ▷ **Constraint 6 – Duration constraints.** The duration (or span) of some pattern is the difference between its last item timestamp and its first item timestamp. A duration constraint requires that the pattern duration should be longer or shorter than a given time period.
- ▷ **Constraint 7 – Gap constraints.** A gap constraint specifies the maximal/minimal number of positions (or timestamp difference) between two successive itemsets in an embedding. The maximal gap constraint is anti-monotonic while the minimal gap is not anti-monotonic.

We organize them in four main categories, the three categories defined by Negrevergne and Guns [2015] (pattern constraints, embedding constraints and pattern set constraints) to which we add the transaction set constraints:

1. **pattern constraints**: they specify constraints on the pattern $\boldsymbol{p}$ itself. They bias directly the search space. The maximal length of the pattern is an example of popular pattern constraint. Among the Pei's constraint presented above, Constraints 1, 2, 3 and 5 are pattern constraints. Pattern constraints are directly related to the definition of the search space. This means that adding pattern constraint is a smart solution to reduce computing time.
2. **embedding constraints**: they specify constraints on the pattern embeddings. Gap and duration constraints are the most popular embedding constraints for sequential patterns. Among the Pei's constraint presented above, Constraints 4, 6 and 7 are embedding constraints.
   Contrary to pattern constraints, such constraints do not necessarily reduce computing time. The time required to evaluate such constraints may not be compensate by the reduction of occurrences.
3. **transaction set constraints**: they specify constraints on the set of supported transactions. Frequency constraint is the most popular example of such kind of constraints.
4. **pattern set constraints**: they specify constraints involving several patterns. Closure or maximality constraints or skypatterns [Ugarte et al., 2015] are examples of such kind of constraints.

More than just impacting computing times, any constraint contributes to the semantic of outputted patterns. A simple example is the case of a minimal support constraint, having such a

[5] meaning that if the constraint holds for pattern X it must hold for all subpatterns of X.

constraint tells that extracted patterns are the "frequent" ones. This means that mining constraints have to be taken into account while analyzing outputted patterns.

The ConQuest approach objective was to propose a high-level query language for specifying constraints on patterns [Bonchi et al., 2006]. The challenges for such approach are, on the one hand, to propose an intuitive query language to express complex constraints on patterns. On the other hand, combining different constraints with different properties may be difficult [Boulicaut and Jeudy, 2005]. Designing an algorithm that fully take advantage of specified constraints to remain efficient is not easy. Different approaches have been proposed. ConQuest is based on the notion of soft-constraints [Bistarelli and Bonchi, 2007]. SkyPatterns [Ugarte et al., 2015] enable to mix several pattern set constraints. Dominance programming [Negrevergne et al., 2013] is an abstract paradigm that combines constraints for specifying a mining task. Finally, declarative pattern mining approaches have been recently studied to model pattern mining by constraint programming. We will detailed this approach in Chapter 4.

A.3.3 Mining subsets of frequent patterns mining

A solution to reduce the number of frequent pattern is simply to not extract the complete set of patterns. In this case, the problem is to extract a diverse set of patterns that are representative of the complete set of desired patterns.

The CSSampling approach [Diop et al., 2018] extracts sequential pattern mining using on pattern sampling. Pattern sampling is a non-exhaustive method for efficiently discovering a set of relevant patterns with strong statistical guarantees. Instead of exploring patterns by traversing the search space, patterns are drawn randomly. CSSampling is a sequential pattern sampling under constraint: it adds some constraints on the length of the pattern (norm). The method randomly draws sequential patterns according to frequency with an interval constraint on the norm.

Another technique to explore the search space with an incomplete heuristic consists in having a random exploration of the search space. The approach of Bosc et al. [2018] uses Monte-Carlo Tree Search to explore randomly stepwise the search space and to select the most promising next steps according to a selection criteria which favor diversity and completeness. SeqScout [Mathonat et al., 2019] explores the same principle with multi-armed bandit to address the problem of subgroup discovery in labeled sequences.

A third class of techniques consists is turning the problem of pattern mining into a machine learning problem, i.e. in an optimization problem. MDL-based pattern mining approach consists in encoding sequences with patterns. The encoding has two parts: the description of the model (patterns) and the description of the sequence encoding. The problem is thus to minimize the whole encoding. The minimum MDL encoding will discard redundant patterns from the model and select only the more informative once (those that rebuilt well the sequences of the dataset). Tatti and Vreeken [2012] employ the MDL principle to identify the set of sequential patterns that summarises the data best. They formalize how to encode sequential data using sets of sequential patterns, and use the encoded length as a quality score. In [Lam et al., 2014], the authors propose GoKrimp, a MDL-based algorithm for mining non-redundant sets of sequential patterns from a sequence database. Finally, the MiSeRe algorithm [Egho et al., 2015] combines a sampling technique with an MDL criteria. It mines sequential rules for classification purpose. The algorithm generates randomly a set of subsequence from the sequences of the dataset and it evaluates their interest in the light of the level criteria.
One advantage of these approaches is that they are parameter free. Nonetheless, parameter free does not mean that it extracts always the best patterns. The encoding is the essential part that biases the notion of interesting patterns in a very complex way.

A.4 Mining frequent sequential pattern mining in alternative datasets

Sequential pattern mining is also classically defined for two alternative types of data:

- ▷ the temporal sequences,
- ▷ the streams (or event logs).

Sequential pattern mining in these contexts remains very similar the problem presented above. The pattern domain is identical, $(\mathcal{L}, \lhd)$, but the difference lays in the way the support of a pattern is evaluated.

Next sections introduce some additional definitions for temporal sequences, logs and stream; and they redefine the notion of support of a sequential pattern which fit our framework (anti-monotonicity).

A.4.1 Frequent sequential patterns in temporal sequences

In most of real applications, sequences that are collected are sequences of timestamped events. For instance, in a sequence of supermarket purchases, each purchase has a date. A temporal sequence is a sequence of timestamped items, so called events.

Definition 12 (Event and Temporal sequence)**.** *An **event** is a couple (e, t) where $e \in \mathcal{I}$ is the event type and $t \in \mathbb{R}$ is a timestamped. (e, t) is strictly before an event (e', t'), denoted $(e, t) \lessdot (e', t')$, iff $t < t'$ or $t = t' \wedge e <_{\mathbb{E}} e'$.*

*A **temporal sequence** $\boldsymbol{s}$ is an set of events $\boldsymbol{s} = \langle (e_1, t_1)\ (e_2, t_2)\ \dots\ (e_n, t_n) \rangle$ such that $(e_i, t_i) \lessdot (e_{i+1}, t_{i+1})$ for all $i \in [n-1]$. The length of sequence $\boldsymbol{s}$, denoted $|\boldsymbol{s}|$, is equal to its number of events. The duration of sequence is $t_n - t_1$.*

In addition, we assume that a temporal sequence does not contain two identical events (same event type at same date).

It is worth noticing that an event is associated to one item, but not to an itemset. Nonetheless, it is possible to represent event types occurring at the same time by two events with the same timestamp.

Then, it is easy to convert a temporal sequence to a sequence and conversely.

Example 11 (Sequences and Temporal sequences)**.** *For instance, in the example of a sequence of purchase $\langle (Chocolate, 34)\ (Milk, 35)\ (Bread, 35)\ (Apple, 42) \rangle$, the corresponding sequence is $\langle Chocolate\ (Milk, Bread)\ Apple \rangle$.*

On the opposite, the sequence $\langle Chocolate\ (Milk, Bread)\ Apple \rangle$ can be converted into a temporal sequence $\langle (Chocolate, 1)\ (Milk, 2)\ (Bread, 2)\ (Apple, 3) \rangle$.

One can notice that the conversion from sequence to temporal sequence (and conversely) preserves the order of the items, but not the timestamps.

Remark 2. *Please note that wherever we write (E, n) for some event type E and timestamp n (as in Example 11), the reduced fontsize of the timestamps has no meaning as it serves only readability purposes.*

Definition 13 (Occurrence and embedding of a sequential pattern)**.** *Let $\boldsymbol{s} = \langle (e_1, t_1)\ (e_2, t_2)\ \dots\ (e_n, t_n) \rangle$ be a temporal sequence and $\boldsymbol{p} = \langle p_1\ p_2\ \dots\ p_m \rangle$ be a sequential pattern. $\boldsymbol{p}$ occurs in $\boldsymbol{s}$, iff there exists a sequence of integers $1 \leq i_1 \leq i_2 \leq \dots \leq i_m \leq n$ such that $e_{i_k} \in p_k$ for all $k \in [m]$, $k < l \implies t_{i_k} = t_{i_l}$ and $k < l \implies t_{i_k} < t_{i_l}$. In other words, $(i_k)_{1 \leq k \leq m}$ defines a mapping from $[m]$, the set of indexes of $\boldsymbol{t}$, to $[n]$, the set of indexes of $\boldsymbol{s}$. We denote by $\boldsymbol{t} \prec \boldsymbol{s}$ the strict sub-sequence relation such that $\boldsymbol{t} \preceq \boldsymbol{s}$ and $\boldsymbol{t} \neq \boldsymbol{s}$.*

*$(i_k)_{1 \leq k \leq m}$ is called an **embedding**.*

A.4.2 Frequent pattern in a stream

The mining of a unique long sequence is usually associated to the name *serial episode mining*[6]. This task consists in extracting frequent sub-sequences from a unique sequence. It can typically correspond to the mining of recurrent behaviors from a long functioning trace from a single system.

On the one hand, this type of dataset can easily be transformed into a set of smalled sequences by windowing the sequence. It can be regularly spaced windows in time or a sliding windows as proposed in the WinEPI algorithm [Mannila et al., 1997]; or it can be windows preceding a given

[6] In episode mining introduced by Mannila et al. [1997] "serial" have to be opposed to concurrent episode, i.e. episode that consider concurrency between item occurrences. Sequential pattern mining are very similar to serial episodes.

item c as we did in our analysis of ECG datasets. The way the windows are built will have a lot of consequences on the results. In the last case, the extracted patterns correspond to patterns that frequently occur before occurrences of item c. Some correlations with c may be induced by the expert. This is not the case with other strategies.

One the other hand, the pattern mining algorithms of a such long sequence may be adapted considering alternative enumerations of pattern occurrences. Achar et al. [2010] reviewed several ways of enumerating occurrences and evaluated their respective complexity. In fact, in this setting, enumerating occurrences of a pattern in a sequence is a combinatorial task.

Some works was interested in mining sequence from a stream of items. The notion of stream brings the idea of incremental modifications of the dataset. In sequence mining, various kind of modifications have been proposed: some algorithms consider transactional incrementality and add new sequences in a database (CISpan [Yuan et al., 2008], PL4UP, [Ezeife. and Monwar, 2007]), some algorithms consider temporal incrementality and add new items at the end of the dataset transactions (IncSpam [Ho et al., 2006], IncSpan [Cheng et al., 2004], [Huang et al., 2006]) and we explored the mining of a unique stream of itemsets and adapt serial episode mining to the streaming case [Guyet and Quiniou, 2012].

B Case study: reproducing a pharmaco-epidemiological analysis

In this section, we introduce the case study we use as running example in the different chapters of this manuscript. This case study has been investigated within a collaborative project with REPERES, a joint laboratory between Rennes University Hospital and EHESP. REPERES studies medico-administrative databases to answer pharmaco-epidemiological questions.

Pharmaco-epidemiology is the study of uses of health products on population in real life. It applies the methodologies developed in general epidemiology [Public Policy Committee *et al.*, 2016] to answer questions about the uses of health products, drugs [Polard et al., 2015; Nowak et al., 2015] or medical devices [Colas et al., 2015], on population. In pharmaco-epidemiology studies, people who share common characteristics are recruited and data about them are collected to build a dataset. Then, meaningful data (drug exposures, events or outcomes, health determinants, etc.) are collected from the sampled population within a defined period of time. Finally, a statistical analysis highlights the links (or the lack of link) between drug exposures and outcomes (e.g. adverse effects). The main drawback of prospective cohort studies is the time required to collect the data and to integrate it. Indeed, in some cases of health product safety, health authorities have to answer quickly public health issues.

A recent approach consists in using administrative databases to perform such studies on care pathways, i.e. an individual sequence of drugs exposures, medical procedures and hospitalizations. Using medico-administrative databases (MADB) has been proved to be useful in pharmaco-epidemiological issues[7], since data is readily available and covering a large population. The french national health insurance database (SNDS/SNIIRAM) has these characteristics and recent national initiatives foster research effort on using it to improve cares.[8] However, it has been conceived for administrative purposes to ensure health reimbursements. Their use in epidemiology is uneasy. It requires to develop specific skills to treat data and to interpret results. They record, with some level of details, all drug deliveries and all medical procedures, for all insured. Such database gives an abstract view on the longitudinal patient care pathways. Medico-administrative databases are often used for risk assessment purposes, and more especially for detection of adverse drug reactions [Zhao et al., 2015]. While pharmaco-vigilance is related to detection, assessment and prevention of risk related to health products, pharmaco-epidemiology could also be viewed as discovering or explaining consequences of health products consumption. The later is a knowledge discovery process

[7] The interest of using this medico-administrative database has been demonstrated by the withdrawal of benfluorex [Weill et al., 2010].

[8] In the Villani report on AI, which founded the French National Plan on AI, the SNDS databases is mentioned as one of the data source to tackle by AI to improve the health care system.

from a large data warehouse of patient care pathways.

B.1 SNDS data warehouse

The french health insurance database, called SNDS (previously SNIIRAM), is organized since 2003 into a huge digital data warehouse which covers most of the french population (65 million inhabitants) with a sliding period of 3 past years. The main characteristics of this data warehouse have been described by Martin-Latry and Bégaud [2010]. Its initial objective was to contribute to a better management of the Health Insurance System. The main part of the data warehouse is made of outpatient reimbursed health expenditures including drugs deliveries. Since 2009, the data warehouse has been enriched with dates of death and with information of public and private hospitals. It does not contain all the inhospital collected data, but only the main stay information (dates of start and end of stays, the primary and related diagnostic ICD codes). Since 2018, the SNIIRAM (i.e. the database of health care reimbursements) is an important building block of the SNDS (National System of Health Data, so called Health Data Hub).

Due to this broad coverage of the french population it provides opportunities to detect weak signals (pathologies with low incidences, rare events). Data extractions from the SNIIRAM data warehouse are available after authorization of regulatory agencies for medical researches.

Another advantage of the SNDS is that it contains structured data and readily available for analysis. It is the opposite situation with inhospital data warehouses that holds textual medical reports, images (radiography, scanners, etc), signals ... these information requires tremendous preprocessing to access the structured information to process. In addition, the SNDS uses taxonomies to structure the information. For example, the drugs are described by their code in the international ATC taxonomy. In the Anatomical Therapeutic Chemical (ATC) classification system, the active substances are divided into different groups according to the organ or system on which they act and their therapeutic, pharmacological and chemical properties. Drugs are classified in groups at five different levels. An issue with such taxonomy is that a drug or a chemical substance can have several codes. For example, the *alprostadil* is classified by codes beginning by $C01EA$ and $G04BE$. It means that if we consider the *alprostadil* delivery as an event, all the codes linked to it have to be processed using the ATC taxonomy.

The main drawback of using medico-administrative databases for pharmaco-epidemiology studies is that these databases are not designed for this purpose but only for reimbursement management. There is a gap between the administrative semantic and the medical information.

Furthermore, a part of the data is not accurate, incorrect or incomplete. For example, in the case of the SNDS, only the delivery of a drug is known and not the real medication time. It is due to the fact that the logged information dedicated to the reimbursement is done in the pharmacy and not when the medication is really taken. Thereby, an event of drug delivery could be logged but the drug never taken. Hypotheses then have to be done based on known guidelines to deduce the real medication time. Diagnoses made by doctors are also missing from the SNDS, a reimbursement was done for a specific out-hospital consultation but the reason is unknown. We only have disease information for hospital stays. Finally, the human factor may also impact the data quality. Practitioners have to fill this information accurately. This is our assumption even if we know that is not necessarily true. The quality of the data requires a great effort to combine them with expert knowledge.

B.2 GENEPI study

The GENEPI study (GENeric substitution of anti-EPIleptic drugs) is a typical example of pharmaco-epidemiology study [Polard et al., 2015]. Its goal was to assess the association between seizure-related hospitalization and drugs substitutions (more especially the substitution between brand-names and generic drugs). Polard et al. [2015] conduct a study to validate or invalidate the hypothesis that a switch from a brand-named anti-epileptic drug to a generic anti-epileptic drug can imply an epileptic seizure.

The studied population is made of controlled epilepsy patients i.e. people known to have a stable treatment before than the seizure occurred. To be sure to obtain data concerning generic substitution, a larger population was requested from the SNDS. The data obtained from the SNDS

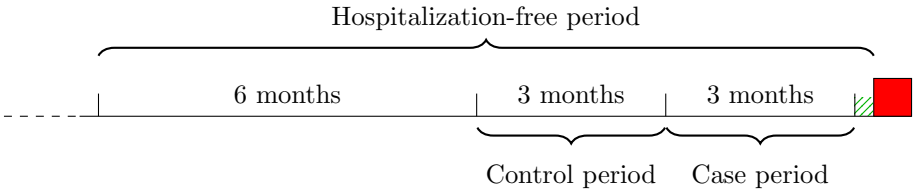


Figure 2.3: Illustration of the heathcare pathway of patients concerned by GENEPI. The index event is the hospitalization with epilepsy diagnosis represented by the red rectangle on the illustration. Before this, a 3-day induction period is removed from the data. The case and control periods are the two 3-month exposure window occurring before the induction period.

		Case period	
		No	Yes
Control	No	7222	478
period	Yes	491	188

Table 2.1: Number of patients according to the occurrence of brand-to-generic substitution in case and control periods. The majority of patients is not concerned by generic substitutions. The number of patients for which a generic substitution occurs only in the case period is similar to the number of patients for which generic substitution only occurs in the control period.

concern all adult patients affiliated to the French national health insurance scheme, 18 years old or older in January 2009, who had at least one reimbursement between 2009 and 2011 for at least one of the following anti-epilieptic drugs: *carbamazepine*, *lamotrigine*, *levetiracetam*, *topiramate*, or *valproic acid*, referred to as selected anti-epileptic drugs. These drugs had a brand name and at least one "A-rated" generic form available on the French market by this time (2009-2011) and were widely prescribed for epilepsy, allowing identification of epilepsy patients. They were extracted from the SNDS database using their ATC codes: $N03AF01$, $N03AX09$, $N03AX14$, $N03AX11$ or $N03AG01$.

The collected data were then pruned to remove specific populations that could bias the results. Patients with a medical history of cancer and women who gave birth within the study period were excluded because of the high risk of repeated seizures as well as patients receiving fatty acid derivatives through formulations registered as mood-stabilizing drugs such as valpromide (DEPAMIDE) or divalproex sodium (DEPAKOTE) who may not be epileptic patients. The events related to cancer or birth were detected by their ICD-10 codes. The International Statistical Classification of Diseases and Related Health Problems (ICD), is a taxonomy similar to ATC but dedicated to the diseases [WHO et al., 1993].

Finally, the target population is selected from the remaining data. To study the effect of generic substitution on epileptic seizures, patients known to be hospitalized for such seizure between January 2010 and December 2011 were selected. Such selection is made by selecting all patients concerned by an event with the ICD-10[9] codes $G40.x$ (epilepsy) or $G41.x$ (status epilepticus) as codes of interest in primary or secondary hospital discharge diagnosis position. The controlled epilepsy patients were at last defined to have a hospitalization-free period for the epilepsy ICD-10 codes of at least a year before the selected hospitalization. Furthermore, a regular dispensation was defined as at least 10 dispensation claims within a year for the same anti-epileptic drug.

After all those pre-processing steps, a statistical model was designed to test the hypothesis. For each patient, a case cross-over protocol was done. In this protocol, the case period i.e. the three months before the hospitalization for epileptic seizure was compared with the control period i.e. three months before. Figure 2.3 illustrates the studied periods of the healthcare pathway. The statistical model lies in four cases: people for whom a generic substitution occurs in both phases, those for whom a generic substitution occurs only before the seizure, those for whom a generic substitution occurs only in the first phase and those for whom no generic substitution occurred. Table 2.1 regroups the different number of patients in each case. Intuitively, the epileptic seizure

[9]ICD-10: International Classification of Diseases 10th Revision

seems to not be correlated with generic substitution occurring in case period. The conclusion of this study was negative i.e. the hypothesis was invalidated [Polard et al., 2015].

B.3 Reproducing GENEPI study through sequential data analysis

Our objective is now to illustrate how to use sequential pattern mining to conduct an experiment that could answer the initial question of GENEPI. It aims at highlighting the potential opportunities and limitations of a classical pattern mining approach to analyze a real data set and thus to motivate some needs of improvements.

The use of sequential pattern mining seems here interesting because GENEPI investigates sequential events of being delivered with generic AED (anti-epileptic drug) and then, with brand-named AED (or vice-versa). Then, the general idea is to use a frequent sequential pattern mining algorithm to extract patterns that would reveal frequent sequential observations in care pathways. If we identify patterns involving generic and brand-named AED we would have identified situation of interest.

But, prior to execute state-of-the-art sequential pattern mining algorithms, we need to define sequences of events. It raises three questions:

- ▷ **which patient care pathway contributes to the data set?** We select patients from the GENEPI data set who had an epileptic seizure. Indeed, we are interested in revealing patterns that may explain occurrences of seizure.
- ▷ **which time period of the care pathway of each will be considered?** We select a period of time preceding an epileptic seizure (case period). In the original study, the case period is 3 months, then we used the same duration. The reason of this choice is to identify trajectory patterns that may explain occurrences of seizure: 1) events must precede a seizure event and be not too distant in time to a seizure event.
- ▷ **how to define the items of which the sequence is made?** We select all drugs delivery events and form items by using a triple : ATC code of drug, specialty group[10] and generic status (0 or 1). This means that the vocabulary of items is made of a list of integers and each integer correspond to a triple. When a drug with the characteristics of the item e is delivered at time t, we append the event (e, t) to the sequence of the patient.

Remark 3 (Sequence engineering)**.** *The above choices may be discussed. This raises the question of preparing a dataset to discover meaningful information, especially for secondary usage data. This step is definitively very important and sensitive in the whole processing chain of knowledge discovery.*

In practice, the data preparation is a tedious task which requires a lot of trials and errors to identify which features are really interesting to enable data analytic algorithms to reveal meaningful information.

Chapter 5 is dedicated to these questions.

The resulting dataset contains $17,862$ sequences (one patient may have experienced several seizure), with a mean length of ≈ 2.83 items. The standard deviation of the lengths is high. Some patients have very few medical events (including none), but some patients may have more than 50 medical events. The vocabulary size is $7,181$. We can now apply our favorite sequential pattern mining algorithm to extract patterns. We use *PrefixSpan* with a minimal threshold set up to 100 (in pharmaco-epidemiology, a hundred of similar cases is sufficient to investigate potential problems), and we tested several different constraints to reduce the pattern set. The number of patterns that we obtained are the following:

- ▷ without constraints: $10,818$ patterns,
- ▷ with closure constraint: $10,602$ patterns,
- ▷ with maxgap constraint (35 days): $10,800$ patterns,
- ▷ with closure and maxgap constraints (35 days): $10,584$ patterns.

[10] The specialty group identifies the drug presentation: international non-proprietary name, strength per unit, number of units per pack and dosage form.

Remark 4. *Faced with the large number of patterns, it would be interesting to filter out patterns that do not involve ADE but a basic implementation of PrefixSpan does not allow to add such constraint to extracted patterns (i.e. to patterns that do not contain a given event). It would require to modify the code without any guarantee of having really interesting patterns to exploit.*

What would be interesting is to have a tool to ease adding new constraints and allowing the specification of complex constraints. This problem will be addressed in Chapter 4.

Let now have a look at some extracted patterns. We focus on *valproic acid* (ATC $N03AG01$, specialty group 438) an AED which is delivered in generic and brand-name forms. The 16 maximal frequent patterns of length 3 that involve both and only generic (383) and originator (114) deliveries of *valproic acid* are listed in the Table 2.2.

Pattern	Support
$\langle(114, 383)\ (114, 383)\ 114\rangle$	100
$\langle(114, 383)\ (114, 383)\ 383\rangle$	100
$\langle(114, 383)\ 114\ 114\rangle$	156
$\langle(114, 383)\ 114\ 383\rangle$	156
$\langle(114, 383)\ 383\ 114\rangle$	156
$\langle(114, 383)\ 383\ 383\rangle$	156
$\langle 114\ (114, 383)\ 114\rangle$	171
$\langle 114\ (114, 383)\ 383\rangle$	171
$\langle 114\ 114\ 383\rangle$	368
$\langle 114\ 383\ 114\rangle$	236
$\langle 114\ 383\ 383\rangle$	236
$\langle 383\ (114, 383)\ 114\rangle$	154
$\langle 383\ (114, 383)\ 383\rangle$	154
$\langle 383\ 114\ 114\rangle$	274
$\langle 383\ 114\ 383\rangle$	274
$\langle 383\ 383\ 114\rangle$	415

Table 2.2: Selection of sequential patterns extracted from the Genepi dataset. Patterns with a support greater than 100 and that involve both 383 (generic valproic acid) and 114 (brand-name valproic acid).

The two most frequent patterns, $\langle(114), (114), (383)\rangle$ and $\langle(383), (383), (114)\rangle$, are also the most interesting. Indeed, they describe a change in the drug deliveries from two generic to brand-name deliveries or from two brand-name to generic deliveries. They can be interpreted as the beginning of a delivery switch that appears prior to seizure. The care pathway of each pattern has now to be analyzed by epidemiologist to interpret the results. The patterns simply identify some trajectories that are statistically relevant, but frequent patterns have not to be misinterpreted.

Remark 5 (Need for more expressive patterns)**.** *Sequential patterns only inform epidemiologists about sequences of events. It poorly exploits the temporal information about care pathway. Nonetheless, it would be interesting to have information about the temporal delay between the events: are they close to each other or are they distant in time? The second case is less interesting as two distant events may not be related. Is a delivery switch close in time to a seizure? If not, an epidemiologist will not relate the switch to seizure. Finally, a trajectory pattern only shows what happens and does not inform about what did not happen. Some deliveries (including other deliveries of valproic acid) may occur beside the occurrence of a pattern and change its interpretation.*

We conclude that patterns with more expressive description of the trajectories would be more interesting. And, more especially, patterns which capture temporal information. In the next Chapter, we present our proposal to extract patterns with metric temporal information.

Chapter 3

Time in sequential patterns

In this chapter, we consider the problem of defining new pattern domains to support temporal knowledge discovery. A pattern domain defines the general shape of considered patterns but it is also associated to a search space. A pattern mining algorithm is at a crossroad of semantic and algorithmic issues:

- ▷ from a semantic point of view, a pattern domain is a kind of lens through which data are seen. This lens enables to zoom-in on particular dimension of the data, but it makes an observer blind to some others. The more complex the pattern domain, the more interesting the extracted knowledge for the expert.
- ▷ from an algorithmic point of view, the more complex the pattern domain, the larger the search space, the less tractable will be the algorithmic mining task. And, as we saw in the preliminaries (see Section A of Chapter 2), the efficiency of pattern mining comes with good structure of its search space (namely, the anti-monotonicity property of the measure of interest).

The historical pattern mining example is purchase basket mining. It illustrates the differences between the pattern domain of itemsets [Agrawal et al., 1993] and the pattern domain of sequences [Agrawal and Srikant, 1995]. Let's consider a client which came to the supermarket twice. He/She first purchases the following items (A, A, B, C) and, 2 days after, he/she purchases items (B, C, D). When mining itemsets, the purchase baskets of a client have to be seen as a single set of purchases without considering repetitions neither the order of purchases, for instance, all the items are put together: $\{A, B, C, D\}$. This illustrates the shape of patterns extracted by itemset mining algorithms. The knowledge is limited to the product a client purchase "at some time". Sequential pattern mining considers the data with a more complex model, $\langle (A, B, C)(B, C, D) \rangle$, where repetitions of B and C or the sequentiality between purchase of A and D are taken into account. The lens of sequential pattern domain offers a wider view on the data compared to the one of itemset domain. A user gets more insights from sequential patterns than from itemset patterns. The counterpart of this wider pattern domain is the additional computation cost of mining sequences. In fact, the size of itemsets search space is lower than the search space of sequential patterns.

Choosing the right pattern domain is of particular importance. In our objective of providing patterns to an expert, it is important to choose some *lens* that will provide abstract views corresponding to the user interest. And more important is to choose pattern domains that will prevent from confusion or miss-understanding of the data. For instance, if it is of particular importance to not confuse event concurrency and event sequentiality, then itemsets should not be used. First, the expert may be unsatisfied by results providing any sequentiality information but, more insidiously, itemset patterns extracted according to a counterintuitive view of the data may confuse the expert.

We have previously seen that several decades of research yield efficient algorithm to extract sequential patterns from large databases (see Section A of Chapter 2). But, the temporal information captured by classical sequential pattern algorithms is limited to the sequentiality (vs concurrency) of events, i.e. all events are ordered sequentially. We can notice that the time delay between purchases is not taken into account in sequential pattern mining. In some contexts, the quantitative

temporal dimension of data provides fundamental information that should be taken into account in the mining processes and returned as part of the extracted knowledge.

When dealing with traces of dynamical system, the temporal information is of particular importance. For example, in web log analysis, the browsing patterns can be refined by the time spent to explore the web pages. The comparison between the quantitative time spent on a web page and the required time to browse its content may be useful to decide whether some part of the information has not been read. In [Quiniou et al., 2003] different kinds of cardiac arrhythmia are discriminated thanks to the quantitative temporal intervals between events. The structure of cardiac wave sequences ($P-QRS-T$) may not provide enough information to distinguish diseases, while the duration of the events (e.g. long P wave associated to a long QRS) witnesses the diseases class. The sequentiality of events does not provide enough information and new mining approaches have to be proposed to extract meaningful patterns.

The question of this chapter is to **study whether injecting temporal information in sequential patterns is 1) practically tractable and 2) really brings new interesting information**.

My contribution to this research question has been to explore quantitative temporal patterns. A quantitative temporal pattern associates a classical sequential pattern with quantitative notions of time. In the following, we present three algorithms illustrating this approach:

- in Section B, we introduce the mining of **interval-based temporal** sequences. In this work, the problem was to explore temporal sequences whose events are qualified by timestamps and duration. We will see that considering only timestamps may not be appropriate, especially in case of event repetitions in sequences. Several algorithms have been proposed, implemented and evaluated.
- in Section C, we introduce **discriminant chronicle mining**. Discriminant sequential patterns have been studied as a alternative to the frequent pattern mining task or in the pattern-based classification [Bringmann et al., 2009]. The contribution of Y. Dauxais PhD thesis was to extract discriminant temporal patterns, the rich model of chronicle was chosen in this case.
- in Section D, we introduce **negative temporal patterns**. Negative sequential patterns have been proposed by Cao et al. [2016] in the e-NSP algorithms. A negative sequential pattern holds information about event types that do not occur in matched sequences. The idea of this work is to inject quantitative temporal information in negative patterns.

A Temporal pattern mining: related works

In this section, we propose a brief overview of the main approaches of temporal pattern mining concerned by enhancing sequential patterns with temporal information.

Temporal sequence mining is a subdomain of temporal data mining [Mitsa, 2010]. Temporal pattern mining involves two different kinds of issue: sequential pattern mining (SPM) algorithms with time constraints [Lee and De Raedt, 2004; Masseglia et al., 2008], and temporal relation discovery [Giannotti et al., 2006; Höppner, 2002; Hwang et al., 2004; Yoshida et al., 2000]. Time constraints approaches, make use of minimum and maximum gap values that must be defined as the gap constraint by the user. Then the algorithms, such as SeqLog [Lee and De Raedt, 2004] or GTC (Graph for Time Constraints) [Masseglia et al., 2008], extract ordered sequences of items that respect these gaps. However, the algorithm does not discover the time constraints. The gaps are used to get rid of insignificant patterns and rules so as to reduce the number of generated patterns and rules. Temporal relation discovery approaches use extended definitions of temporal sequences. As our challenge is to have temporal information as part of the extracted knowledge, we do not review works on specification of temporal constraints.

Let first have a look at the representation of temporal information in sequences or patterns. The grid presented in Figure 3.1 organizes representations in three dimensions:

- in rows, **punctual vs interval** based representation of events. A punctual event is an event that has no duration, while interval based events have a certain time span.

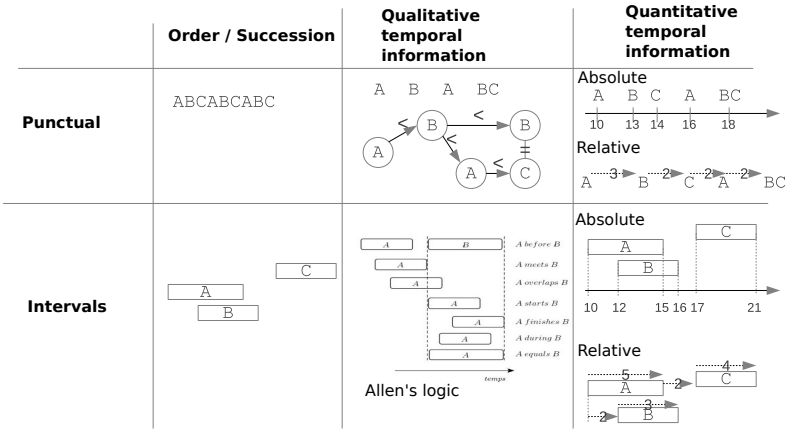


Figure 3.1: Illustration of the classes of temporal information representation in temporal sequences.

Patterns / Sequence	Punctual, qualitative	Punctual, quantitative	Intervals, qualitative	Intervals, quantitative
Punctual, qualitative	GSP PrefixSpan SPADE DiffSet WinEPI MinEPI ...			
Punctual, quantitative				TAS Chronicles Delta-patterns TGSP
Intervals, qualitative			Hoeppner IEMiner KarmaLego Armada Arrangements	
Intervals, quantitative				DepthPIMS QtempIntMiner QTIPrefixSpan

Figure 3.2: Categorization of state of the art temporal pattern mining algorithms. In raw, characteristics of temporal sequences and in columns, characteristics of temporal patterns. Grey cells are impossible configurations.

- ▷ in columns, **quantitative vs qualitative** representation of temporal information. In this dimension, we distinguish mainly three groups of approaches: 1) the representation of the linear order of events, such as sequences represents time; 2) the representation of qualitative relations between events. McDermott [McDermott, 1982] and Allen logics [Allen, 1984] are the classical logics for representation and reasoning about respectively punctual or interval events; 3) the representation of quantitative relations between events. The quantitative representation of time associate the time (duration of timestamps) with figures. This figures may be real-valued (metric) or integer-valued. Integer-valued representations of time refer to Linear Temporal Logics [Balbiani and Condotta, 2002] and real-values representations refer to Metric Temporal Logics [Ouaknine and Worrell, 2008].
- ▷ **absolute vs relative** representation of time. The absolute representation of time denotes timestamped events, i.e. events associated to a time instant or time period with is rooted on a timeline with a start time. The relative representation of time denotes the representation of the delays between the events.

Representation of time is a core issue in the artificial intelligence community and many other dimensions of time omitted in the brief overview. For instance, representing and reasoning with granularity of time in temporal data [Combi et al., 2004] is also very important, but we did not addressed this dimension in our work. The reader that would be interested in deep insights about the representing and reasoning about time may refer to Chittaro and Montanari [2000], or for medical applications to Combi et al. [2010].

The representation of time proposed in Figure 3.1 are the inputs of the grid of analysis in Figure

3.2. It organizes the state-of-the art of temporal pattern mining in two dimensions:
- ▷ the modeling of temporal information in temporal sequences
- ▷ the modeling of temporal information in temporal patterns

Quali/Quantitative punctual events sequences with qualitative patterns There are many articles to extract qualitative patterns from temporal sequences. Indeed, mining sequential patterns was issued at the early age of the pattern mining field and . the classical sequential pattern mining such as GSP [Agrawal and Srikant, 1995], *PrefixSpan* [Pei et al., 2001] or Spade [Zaki, 2001] handle the quantitative temporal dimension of events in a sequence by the notion of succession or itemsets. We refer the reader to Mooney and Roddick [2013] for a survey of sequential pattern mining algorithms.

Mannila et al. [1997] introduces the notion of episode which generalize the notion of sequential order to an arbitrary partial order of symbols. This representation of patterns allow to capture notion of simultaneity of events. Laxman et al. [2007] proposes an elegant paper about generalized episode mining.

Quantitative interval-based sequences with qualitative patterns In [Höppner, 2002; Kam and Fu, 2000; Papapetrou et al., 2009; Moskovitch and Shahar, 2015; Patel et al., 2008; Winarko and Roddick, 2006], qualitative temporal relations on intervals, i.e. Allen temporal relations, are associated with sequence items. Allen's interval algebra [Allen, 1984] is a set of 7 relations between intervals ($before$, $meet$, $overlap$, $start$, $during$, $finishes$, $equality$).
Höppner [2002] extracts patterns as a graph of events linked by Allen's relations. The author proposes a level-wise algorithm inspired from *GSP* [Agrawal and Srikant, 1995] to extract efficiently both event sequences and temporal relations.
Papapetrou et al. [2009] proposed the notion of *arrangement* which uses a subset of Allen's relations between successive pairs interval-based events. An arrangement is then an alternation of items and operators, e.g. $A|B \star A|C \star B > C$. Arrangements are a subclass of Höppner's patterns, thus computationally more efficient to extract. Papapetrou et al. [2009] propose a depth first strategy with an adaptation of the prefix projection strategy [Pei et al., 2004].
KarmaLego algorithm [Moskovitch and Shahar, 2015] extracts similar types of patterns. It consists of two main phases: 1) all of the frequent two-sized patterns are extracted, and 2) a recursive process extends the frequent 2-sized patterns. Sacchi et al. [2007] applied the same principle to time series analysis by proposing a formalism of knowledge-based temporal abstraction which enable to represent time series as a sequence of temporal intervals.

Finally, Quiniou et al. [2003] use inductive logic programming (a first-order relational learning method) to construct a graph of temporal constraints. In this framework, temporal constraints are predefined through a bias language turning the mining problem to a classical constrained pattern mining problem.

Quantitative punctual event sequences with quantitative patterns In [Dousson and Duong, 1999; Giannotti et al., 2006; Hwang et al., 2004; Yoshida et al., 2000; Yen and Lee, 2013], the temporal information associated with temporal sequences is abstracted by quantitative intervals in a patterns. Giannotti et al. [2006] propose an extension of the sequence mining paradigm to linear *temporally-annotated* sequences, where each transition $A \rightarrow B$ in a sequence is annotated with a typical transition time t, denoted by $A \xrightarrow{t} B$. The notion of *Delta-Pattern*, proposed by Yoshida et al. [2000] is quite similar to temporally annotated sequential patterns. *Delta-Patterns* have the form $A \xrightarrow{[0,3]} B$, denoting a sequential pattern $A \rightarrow B$ that frequently occurs in the data set with transition times from A to B that are contained in $[0, 3]$. Hirate and Yamana [2006] propose a generalized sequential pattern mining with the capability to handle two kinds of item interval measurements, item gap and time interval. An example of patterns extracted by Hirate and Yamana [2006] is $\langle (0, H), ((0, 1day], H), ((0, 1day], H)\rangle$, meaning that a second H item occurs between 0 and 1 day after the first H item (that occur at time 0, and a third H event occurs between 0 and 1 day after the second H item. The FACE system [Dousson and Duong, 1999] extracts patterns with temporal extents of the interval between pairs of events. Temporal extents are intervals representing a constraint on the delay between events. Such models are called *chronicles*

and is detailed in Section C.1. The same kind of patterns are extracted by ASTPminer [Alvarez et al., 2013].

Giannotti et al. [2006], TGSP [Yen and Lee, 2013] and Yoshida et al. [2000] use a clustering method to extract the representative values of the transitions. Yoshida et al. [2000] provide an heuristic for finding some frequent *Delta-Patterns* but their method is not exhaustive. Giannotti et al. [2006] propose to combine a candidate sequence generation step with a clustering step, but do not provide details nor evaluations of their method. In these two methods, the main issue of intertwining a pattern growing method with temporal interval generation based on the analysis of interval distributions is not explicitly addressed. TGSP [Yen and Lee, 2013] uses a different technique. It first extracts the sequential patterns and then, for each patterns, it computes temporal constraints using a density-based clustering algorithm.
FACE or Hirate and Yamana [2006] adapt classical sequential pattern mining algorithms (respectively *GSP* and *PrefixSpan*) and make use of anti-monotonic constraints. They are incomplete but do not request clustering algorithm. However, Cram et al. [2012] proposed a complete and correct algorithm to mine chronicles. Sahuguède et al. [2018] proposed an algorithm to mine chronicles similar to the KarmaLego principle (first mining pairs of events and then combine them).

Quantitative interval-based sequences with quantitative patterns Only few approaches addressed the problem of mining quantitative temporal patterns while sequences at hand are interval-based sequences. Most of them derived from our own work that is presented in next section. Similarly to the TAS [Giannotti et al., 2006], it consists in extracting patterns as an interval-based sequence which is a representative of a set of sequences.

Alternatively, Gomariz [2014] proposed a collection of algorithms to mine temporal patterns, including DepthPIMS and BreadthPIMS which are able to mine quantitative patterns composed of both points and intervals based sequences. PIMS algorithm represents a pattern as set of items and a matrix of constraints between pairs of patterns that are discovered. Constraints may be qualitative (Allen's relations) but also quantitative. For instance, a constraint "$<$ [2]" between events A and B specifies that the end of event A must be 2 time units before the beginning of event B.

B Mining patterns with temporal intervals[1]

In this section, we consider the problem of discovering frequent temporal patterns in a database of temporal sequences, where a temporal sequence is a set of items with associated a date and a duration.

Let us first redefine a temporal sequence by extending Definition 12 with the temporal events with date and duration.

Definition 14 (Temporal sequence)**.** *An* event *is a couple* $(e, [l, u])$ *where* $e \in \mathcal{I}$ *is the event type and* $[l, u]$ *is a non empty temporal interval* $[l, u]$*.* $l, u \in \mathbb{R}, l < u$ *are timestamps.* $(e, [l, u])$ *is strictly before an event* $(e', [l', u'])$*, denoted* $(e, [l, u]) < (e', [l', u'])$*, iff* $l < l' \vee (l = l' \wedge (e < e' \vee (e = e' \wedge u < u')))$.

A temporal sequence $\boldsymbol{s}$ *is a set of events* $\boldsymbol{s} = \langle (e_1, [l_1, u_1])\ (e_2, [l_2, u_2])\ \ldots\ (e_n, [l_n, u_n]) \rangle$ *such that* $(e_i, [l_i, u_i]) < (e_{i+1}, [l_{i+1}, u_{i+1}])$ *for all* $i \in [n-1]$.

l_i (resp. u_i) is the beginning (resp. end) occurrence date of the interval-based event in the temporal sequence. Events in a sequence are ordered by start dates and then by lexicographic order.

Figure 3.3 illustrates the temporal sequence $\boldsymbol{s_{ex1}} = \langle (A, _{[1,2]}), (C, _{[1.5,4]}), (B, _{[3,4]}), (C, _{[5,6]}) \rangle$.

Intuitively, the problem of mining frequent temporal patterns from $\mathcal{D}$, a collection of temporal sequences (with interval-based events), is similar to the problem of mining temporal sequences: we are looking for some *temporal patterns* that occur at least σ times in the $\mathcal{D}$. A temporal pattern, so called QTI pattern (Quantitative Temporal Interval pattern), is also a temporal sequence with interval-based events to capture information about the duration of events.

The challenge is then to extract both the information about the sequence of events and the information of temporal intervals.

[1] This section is a joint collaboration with René Quiniou

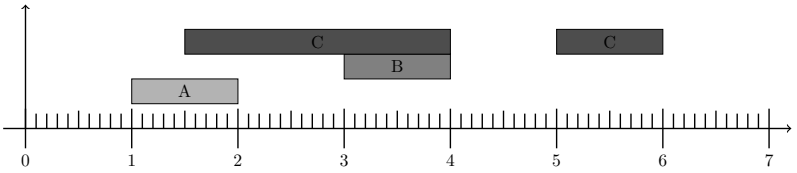


Figure 3.3: The interval sequence $\boldsymbol{s_{ex1}} = \langle (A, [1, 2]), (C, [1.5, 4]), (B, [3, 4]), (C, [5, 6]) \rangle$.

Figure 3.4: Alternative representation of the interval-based temporal sequence in a sequence of events (on the left) or in a sequence of timestamped events (on the right).

When initially defined [Guyet and Quiniou, 2008], this pattern mining task was a new challenge. Some approaches addressed the mining of temporal patterns with interval-based events but using only symbolic temporal relations between the pattern events [Höppner, 2002] while temporal pattern mining like [Dousson and Duong, 1999; Giannotti et al., 2006; Yen and Lee, 2013] extract quantitative information from temporal sequence with events without duration. By exploring QTI patterns, we aim at extracting more temporal knowledge from the data than the former approaches.

In the remainder of this section, we first motivate the work by illustrating that explicit representation of event duration can not be reduced to a representation by classical (temporal) sequences. Then, we present the general principles of QTI pattern mining algorithms. Then we briefly introduced two algorithms, namely QTempIntMiner and QTIPrefixSpan, in Section B.2.4 and we compare them experimentally in Section B.3.

B.1 Motivations for explicit representation of durations

While defining a new pattern domain or a new pattern mining task, the first question is "can it be addressed by a good old-fashioned algorithm?".

Indeed, some approaches proposed to represent QTI patterns with more classical (temporal) sequences such that they can apply the classical temporal pattern mining algorithms:

1. Chiu et al. [2003] propose to represent the event's duration by the repetition of a same event having a fixed atomic duration.
2. Wu and Chen [2007] propose to represent the event's duration by a pair of timestamped events: one event for the start and one event for the end.

Example 12. *Figure 3.4 illustrates the two alternative representations of sequence* $\boldsymbol{s_{ex1}}$ *(see Figure 3.3). The atomic duration of an event is 0.5, then* $\boldsymbol{s_{ex1}}$ *is represented by the* $\widetilde{\boldsymbol{s_{ex1}}} = \langle A\ (AC)\ C\ C\ (BC)\ (BC)\ \emptyset\ \emptyset\ C\ C \rangle$*. In this representation, we denote* $\emptyset$ *the empty itemset.*

For the second solution, each event generates two different events representing the start and the end of an interval. For instance, As (resp. Ae) is the event type for the start (resp. end) of A. Then $\boldsymbol{s_{ex1}}$ *is represented by the* $\widetilde{\boldsymbol{s_{ex1}}} = \langle (As, 1)\ (Cs, 1.5)\ (Ae, 2)\ (Bs, 3)\ (Be, 4)\ (Ce, 4)\ (Cs, 5)\ (Ce, 6) \rangle$.

Unfortunately, these intuitive transformations are misleading while mining patterns. In fact, patterns mining algorithms may extract some patterns that are actually not in the original dataset.

Let us first illustrate the problems encountered with the representation with repeated events with the following set of sequences.

$$\mathcal{D} = \begin{cases} \boldsymbol{s_1} = \langle (A, {}_{[0,3]})\ (B, {}_{[3,5]}) \rangle \\ \boldsymbol{s_2} = \langle (A, {}_{[0,3]})\ (B, {}_{[3,5]}) \rangle \\ \boldsymbol{s_3} = \langle (A, {}_{[0,3]})\ (B, {}_{[3,5]}) \rangle \\ \boldsymbol{s_4} = \langle (A, {}_{[0,2]})\ (B, {}_{[2,5]}) \rangle \\ \boldsymbol{s_5} = \langle (A, {}_{[0,2]})\ (B, {}_{[2,5]}) \rangle \end{cases}$$

its translation is then

$$\widetilde{\mathcal{D}} = \begin{cases} \widetilde{\boldsymbol{s_1}} = \langle A\ A\ A\ B\ B \rangle \\ \widetilde{\boldsymbol{s_2}} = \langle A\ A\ A\ B\ B \rangle \\ \widetilde{\boldsymbol{s_3}} = \langle A\ A\ A\ B\ B \rangle \\ \widetilde{\boldsymbol{s_4}} = \langle A\ A\ B\ B\ B \rangle \\ \widetilde{\boldsymbol{s_5}} = \langle A\ A\ B\ B\ B \rangle \end{cases}$$

The two closed patterns that occur at least 3 times in the dataset $\widetilde{\mathcal{D}}$ are:

- ▷ $\langle A\ A\ A\ B\ B \rangle$ which is $\langle (A, {}_{[0,3]})\ (B, {}_{[3,5]}) \rangle$: it is the pattern that is expected to be extracted
- ▷ $\langle A\ A\ B\ B \rangle$ which is $\langle (A, {}_{[0,2]})\ (B, {}_{[2,4]}) \rangle$: but this pattern actually does not exist in the dataset. Indeed, the two interesting behaviors in the data set are "a long A and then a short B" or "a short A and then a long B". The extracted pattern says that there are "a short A and then a short B" which is obviously undesirable.

The problem with this representation is that it introduces a confusion between the duration and the repetition. The temporal sequences $\langle (A, {}_{[0,2]})\ (A, {}_{[2,3]}) \rangle$ and $\langle (A, {}_{[0,3]}) \rangle$ have the exact same representation using the technique of Chiu et al. [2003]. However, in some applications it is necessary to distinguish a repetition of events from a single occurrence of this event: for example, two apneas of $30s$ each should not be confused with a single apnea of $1min$. This distinction is only possible with a quantitative representation of the time information.

The similar problem is encountered with a representation with timestamped events: some different temporal sequences have the same representation with timestamped events. For instance, these following sequences have the exact same representation $(\langle (As, 0)\ (As, 4)\ (Ae, 5)\ (Ae, 7) \rangle)$.

$$\begin{cases} \boldsymbol{s_1} = \langle (A, {}_{[0,5]})\ (A, {}_{[4,7]}) \rangle \\ \boldsymbol{s_2} = \langle (A, {}_{[0,7]})\ (A, {}_{[4,5]}) \rangle \end{cases}$$

This confusion may leads to generate patterns that have no meaning for the user.

Then, to prevent from extracting misleading patterns, the methods we proposed mine temporal sequences made of events, stamped with date and duration.

B.2 General approach for mining frequent QTI patterns

In this section, we first give some additional definitions used for interval based temporal sequences, and then some generic principles of mining QTI patterns.

B.2.1 Preliminary definitions

In this section we give additional definitions used for representing the interval based temporal sequences.

Definition 15 (Symbolic signature)**.** *Let $\boldsymbol{s} = \langle (s_i, [l_i, u_i]) \rangle_{i \in [n]}$ be a temporal sequence, the symbolic signature of $\boldsymbol{s}$, denoted $\underline{\boldsymbol{s}}$, is the sequence of its symbols: $\underline{\boldsymbol{s}} = \langle s_1\ \ldots s_n \rangle$. The order of symbols in the symbolic sequence is the same as in the temporal sequence.*

Definition 16 (Multi-dimensional interval)**.** *A multi-dimensional interval of dimension n is a tuple of n intervals $I = ([l_i, u_i])_{i \in [n]}$.*

Definition 17 (Projection over a symbolic signature, π)**.** *Let $\boldsymbol{s} = \langle (s_i, [l_i, u_i]) \rangle_{i \in [n]}$ be a temporal sequence and $\underline{\mathcal{M}} = \langle m_1\ \ldots\ m_k \rangle$ be a sequence of length k. A projection of $\boldsymbol{s}$ over the symbolic*

pattern signature $\underline{\mathcal{M}}$*, denoted* $\pi_{\underline{\mathcal{M}}}$*, is a multi-dimensional interval of dimension* k*,* $([l_{j_i}, u_{j_i}])_{i\in[k]}$ *such that* $(j_i)_{i\in[k]} \in [n]$ *is an increasing*[2] *tuple and* $s_{j_i} = m_i$ *for all* $i \in [k]$*.*

If $\underline{\mathcal{M}}$ *is not a subsequence of* $\underline{\boldsymbol{s}}$ *then* $\pi_{\underline{\mathcal{M}}}(\boldsymbol{s}) = \emptyset$*.*

$\pi(\boldsymbol{s})$ *denotes the projection of the whole temporal sequence over its proper signature:* $\pi_{\underline{\boldsymbol{s}}}(\boldsymbol{s}) = ([l_i, u_i])_{i\in[n]}$*.*

Example 13. *Let* $\boldsymbol{s_{ex1}}$ *be the sequence* $\langle (A, [1,2])\ (C, [1.5,4])\ (B, [3,4])\ (C, [5,6])\rangle$ *depicted in Figure 3.3. Then* $\underline{\boldsymbol{s_{ex1}}} = \langle A\ C\ B\ C\rangle$*,* $\pi_{\langle A,B\rangle}(\boldsymbol{s_{ex1}}) = ([1,2],[3,4])$ *and* $\pi(\boldsymbol{s_{ex1}}) = ([1,2],[1.5,4],[3,4],[5,6])$*.*

Definition 18 (Temporal sequence dissimilarity)**.** *Let* $\boldsymbol{s}$ *and* $\boldsymbol{s'}$ *be two temporal sequences.* $d(\boldsymbol{s}, \boldsymbol{s'})$*, the dissimilarity between* $\boldsymbol{s}$ *and* $\boldsymbol{s'}$*, is defined by:*

$$d(\boldsymbol{s}, \boldsymbol{s'}) = \begin{cases} \infty \text{ if } \underline{\boldsymbol{s}} \neq \underline{\boldsymbol{s'}}, & \textit{(different symbolic signature)} \\ \delta\left(\pi(\boldsymbol{s}), \pi(\boldsymbol{s'})\right) \end{cases}$$

where δ *is a distance measure on multi-dimensional intervals.*

Several distance measures between multi-dimensional intervals have been given in [Irpino and Verde, 2008]. In the following, we consider three distances: the Haussdorf distance, noted d_H, the CityBlock distance (sum of the absolute difference between corresponding interval bounds), noted d_{CB}, and Lebesgue distance (volume intersection ratio), d_L. Given $I = ([l_k^I, u_k^I])_{k\in\mathbb{N}_n}$ and $J = ([l_k^J, u_k^J])_{k\in\mathbb{N}_n}$, we have:

$$\begin{aligned} d_H(I,J) &= \sum_{k=1}^{n} \max\left(|l_k^I - l_k^J|, |u_k^I - u_k^J|\right), \\ d_{CB}(I,J) &= \sum_{k=1}^{n} |l_k^I - l_k^J| + |u_k^I - u_k^J|, \\ d_L(I,J) &= \frac{\max\left(\prod_{i=1}^{n} u_i - l_i, \prod_{i=1}^{n} u_i' - l_i'\right)}{\max\left(0, \min(u_i, u_i') - \max(l_i, l_i')\right)} \end{aligned}$$

B.2.2 Frequent temporal pattern extraction

Definition 19 (QTI pattern)**.** *A QTI pattern is a representative temporal sequence of a set of temporal sequences (instances of the pattern).*

As Giannotti et al. [2006], our pattern definition relies on the notion of sub-sequence representativity. A temporal pattern represents a set of temporal sequences that contains a subsequence that is closed to the representative. This notion relies on the chosen similarity between multidimensional intervals, and on a threshold ϵ. Then, we can define a containment relation between a pattern $\boldsymbol{p}$ and a sequence $\boldsymbol{s}$.

Definition 20 (QTI pattern ϵ-covering)**.** *A QTI pattern* $\boldsymbol{p} = \langle (p_i, [l_i^p, u_i^p])\rangle_{i\in[n]}$ *ϵ-covers a sequence* $\boldsymbol{s} = \langle (s_i, [l_i^s, u_i^s])\rangle_{i\in[m]}$*, denoted* $\boldsymbol{s} \preceq_\epsilon \boldsymbol{p}$*, if and only if there exists a projection of* $\boldsymbol{s}$ *over* $\underline{\boldsymbol{p}}$ *such that its dissimilarity with* $\boldsymbol{p}$ *is less than* ϵ*:*

$$\exists J \subseteq [m], \quad d\left(\boldsymbol{p}, \left\langle \left(s_j, \left[l_j^s, u_j^s\right]\right)\right\rangle_{j\in J}\right) < \epsilon$$

Example 14. *Let* $\boldsymbol{p_1} = \langle (C, [1,2])\ (A, [2,4])\rangle$*,* $\boldsymbol{p_2} = \langle (A, [1,2])\ (C, [2,4])\rangle$ *and* $\boldsymbol{p_3} = \langle (A, [1,2])\ (C, [4,5])\rangle)$ *be three QTI patterns. Let* $\epsilon = 1$ *and* δ *be the interval distance CityBlock.* $\boldsymbol{p_1}$ *does not* ϵ*-cover* $\boldsymbol{s_{ex1}}$ *(cf. Example 13) because* $\boldsymbol{s_{ex1}}$ *does not contain the sub-sequence* $\langle C\ A\rangle$*.* $\boldsymbol{p_2}$ *ϵ-covers* $\boldsymbol{s_{ex1}}$ *because* $\delta\left(\ ([1,2],[2,4]), ([1,2],[1.5,4])\ \right) = 0.5 < \epsilon$*.* $\boldsymbol{p_3}$ *does not ϵ-cover* $\boldsymbol{s_{ex1}}$ *because the dissimilarity of each occurrence of* $\boldsymbol{p_3}$ *in* $\boldsymbol{s_{ex1}}$ *is greater than* ϵ*:* $\delta\left(\ ([1,2],[4,5]),\ ([1,2],[1.5,4])\ \right) = 3.5$ *and* $\delta\left(\ ([1,2],[4,5]),\ ([1,2],[5,6])\ \right) = 2$*.*

[2] $\forall i,\ i < k,\ j_i < j_{i+1}$

Definition 21 (Partial order on QTI patterns). *Let* $\boldsymbol{p} = \langle (s_i, [l_i, u_i]) \rangle_{i \in [n]}$ *and* $\boldsymbol{p'} = \langle (p'_i, [l'_i, u'_i]) \rangle_{i \in [m]}$ $\boldsymbol{p}$ *is less specific than* $\boldsymbol{p'}$ *iff there exists a sequence of integers* $(i_j)_{j \in [m]} \in [n]$ *such that* $p'_i = p_{j_i}$ *and* $[l_{j_i}, u_{j_i}] \subseteq [l'_i, u'_i]$ *for all* $i \in [m]$.

Proposition 4 (QTI lattice). *Let QTI denotes the set of QTI patterns then,* $(QTI, \preceq_\epsilon)$ *is an infinite semi-lattice.*

Proposition 5 (Anti-monotonicity of ϵ-cover). *ϵ-cover is a is anti-monotonic on the set of QTI patterns. Let* $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *be two QTI patterns and* $\boldsymbol{s}$ *a temporal sequence. If* $\boldsymbol{p}$ *is more specific than* $\boldsymbol{q}$ *then* $\boldsymbol{p}$ *ϵ-covers* $\boldsymbol{s}$ *if* $\boldsymbol{q}$ *ϵ-covers* $\boldsymbol{s}$.

Thus, QTI patterns are structures in a lattice, and we have a support measure that is anti-monotonic. Then, we fit the classical framework of frequent sequential pattern mining (see Section A), we can define a new task of frequent QTI pattern mining.

Definition 22 (Frequent QTI pattern mining). *Given a frequency threshold* $f_{min} \in [0, 1]$, *a maximal dissimilarity* $\epsilon \in \mathbb{R}^+$ *and a database of temporal sequences* $\mathcal{D}$, *the frequent QTI pattern extraction consists in generating QTI patterns* $\boldsymbol{p}$ *such that the frequency of sequences that ϵ-cover a temporal pattern* $\boldsymbol{p}$ *is greater than* f_{min}.

B.2.3 General principles of QTI patterns mining

Algorithm 4 gives a pseudo-algorithm abstracting the two approaches that will be detailed in next sections. Each of the concrete algorithm has its own implementations of the main functions (NextCandidate, TemporalCharacteristics).

The general principle of these algorithms is inspired by the classical pattern mining algorithm. It is a generate and test approach. A candidate sequential pattern $\boldsymbol{p}$ is generated by the NextCandidate function from the previous frequent patterns. It it worth noticing that the function generates this symbolic signature of a QTI pattern denoted $\underline{\boldsymbol{p}}$, i.e. simply a sequential pattern, but not a QTI (symbolic signature and multidimensional interval). Then, if this symbolic signature is frequent (line 3-4) in the dataset then lines 5-8 generates candidate multidimensional intervals to assign to $\underline{\boldsymbol{p}}$. Indeed, we have an anti-monotonicity property: if the symbolic signature is not frequent, any QTI with this signature is not frequent either (more specific). Thus, it prevents from generating a priori infrequent candidate QTI.

The generation of candidate multidimensional intervals is based on the projection of symbolic signature (function TemporalCharacteristics, line 8). This function identifies clusters of similar projections and generalizes them by representative multidimensional intervals.

Lines 9 to 13, candidate QTI patterns are generated combining the symbolic signature $\underline{\boldsymbol{p}}$ with a multidimensional interval. The support of each of such patterns, $\boldsymbol{tp}$, is evaluated to decide whether $\boldsymbol{tp}$ is frequent or not.

The principle of Algorithm 4 is to use temporal constraints as soon as possible to prune quickly patterns that are not frequent. This strategy is significantly different from algorithms, such as TGSP [Yen and Lee, 2013], that first extract all the candidate symbolic signatures and in a final step, extract the temporal constraints for each of them. Our intuition is that evaluating temporal pattern support along the mining process enables to prune as early as possible the infrequent patterns and that despite the additional computation cost, it improves the overall efficiency of the algorithm. It is more especially efficient for datasets whose temporal dimension is of particular importance, i.e. which contains similar symbolic signatures, but very different temporal constraints.

The TemporalCharacteristics function, line 8, generalizes a set multidimensional intervals by a more general multidimensional interval that may be frequent. This task maybe addressed with a complete and correct algorithm [Kaytoue et al., 2011]. But, the computational cost would be too high and the number of candidates would explode. For this reasons, the two algorithms presented in next section apply clustering algorithms which extract only representatives multidimensional intervals.

B.2.4 Algorithms for QTI pattern mining

We proposed three different algorithms:

Algorithm 4: General principle of the interval-based temporal pattern mining algorithms.

Input: f_{min}: minimal support, $\mathcal{D}$: temporal sequences dataset
Output: $\mathcal{P}$: frequent interval-based temporal patterns

```
1  p ← ⟨⟩;
2  while p ≠ ∅ do
3      EvaluateSymbolicSupport(p) ;
4      if p is frequent then
           // Compute temporal projections of p on the dataset
5          M ← ∅;
6          for s ∈ D do
7              M ← M∪ ComputeProjections(s, p) ;
           // Generate candidate multidimensional intervals
8          C ← TemporalCharacteristics(p, M) ;
9          for c ∈ C do
               // Generate a candidate QTI pattern
10             tp ← (p, c);
               // Evaluate the candidate QTI pattern support
11             EvaluateSupport(tp) ;
12             if tp is frequent then
13                 P ← P ∪ {tp};
14     p ← NextCandidate(P) ;
15 return P;
```

- ▷ QTempIntMiner [Guyet and Quiniou, 2008]: an algorithm based on a hyper-cube representation of temporal sequences which uses a density estimation of the distribution of event intervals to extract quantitative temporal information. It uses a breadth first search strategy to traverse the set of QTI patterns.
- ▷ QTIPrefixSpan [Guyet and Quiniou, 2011]: this class of algorithms is based on a distance-based clustering of temporal intervals intertwined with sequential pattern extraction steps. It uses a depth first search strategy to traverse the set of QTI patterns.
- ▷ QTIAPriori: this class of algorithms is also based on the distance-based clustering of temporal intervals but it uses a breadth first search strategy to traverse the set of QTI patterns.

B.3 Comparative evaluation

In the following experiments, we evaluate computing performances of the proposed algorithms and we assess the quality of extracted patterns on synthetic datasets. We compare QTempIntMiner [Guyet and Quiniou, 2008], QTIAPriori and QTIPrefixSpan [Guyet and Quiniou, 2011] (with different multidimensional interval distances).[3]

B.3.1 Synthetic dataset generation

The data generation process generates random temporal sequences based on a *temporal pattern prototype* to which is added temporal noise and structural noise. Our temporal sequence generator generate datasets that can be tuned with respect to the particular features of hidden patterns (particularly their numerical temporal features) and of the sequence database (number of sequences, noise level). The principle used to generate a temporal sequence database consists in building sequences from temporal pattern prototypes and, next, in adding random sequences (noise) in the sequence database at a rate rP, $0 \leq rP \leq 1$.

[3] All algorithms have been implemented in Matlab. QTempIntMiner implementation uses the MixMod library [Biernacki et al., 2006] to compute Gaussian mixture models. The source code of the algorithms and of the data generator can be downloaded from http://people.irisa.fr/Thomas.Guyet/QTempIntMiner/

A temporal pattern prototype is a set of 5-tuples $\{(E_i, \mu_{b_i}, \sigma_{b_i}, \mu_{d_i}, \sigma_{d_i})\}_{i \in \mathbb{N}_n}$ where E_i is a symbol, μ_{b_i}, μ_{d_i} (resp. $\sigma_{b_i}, \sigma_{d_i}$) are the means (resp. standard deviation) of the beginning date and of the duration of E_i. A prototype specifies a temporal pattern $(\{(E_i, [\mu_{b_i}, \mu_{b_i} + \mu_{d_i}])\}_{i \in \mathbb{N}_n})$ to be discovered. The instances of such a pattern are generated from a gaussian distribution of dates (resp. durations) centered on μ_b (resp. μ_d) with a standard deviation σ_b (resp. σ_d). More the standard deviation is large, more the temporal variations are important and more the extraction of the original pattern will be difficult. To simplify the result presentation, we fix a unique parameter $tN \in \mathbb{R}^+$ for quantifying all the standard deviation values. This parameter quantifies the *temporal noise* of a dataset.

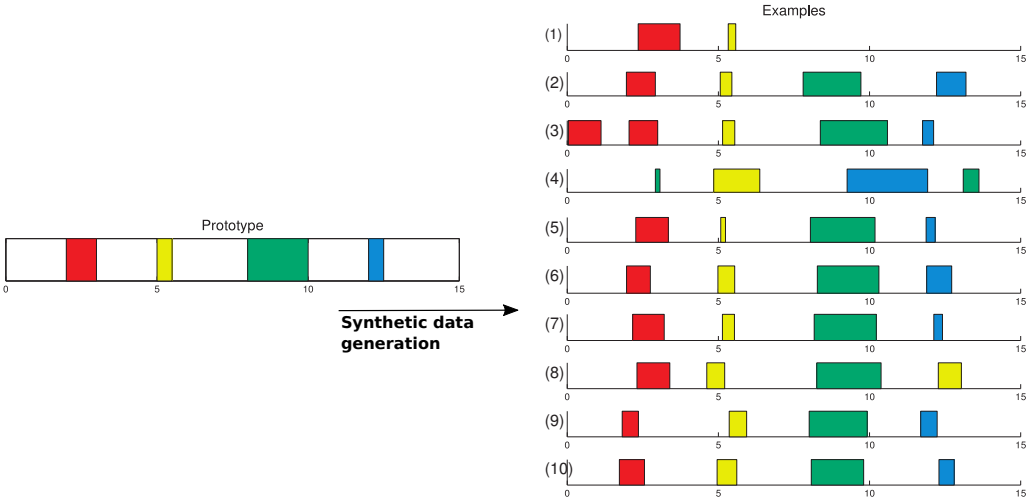


Figure 3.5: Prototype (on the left) and 10 generated examples (on the rights). First sequence illustrates possible random deletion. Fourth and eighth sequences illustrate corruptions of event types. Third sequence illustrates event insertion. Finally, all temporal intervals are randomly generated.

Figure 3.5 illustrates the temporal sequence generation process ($nb = 10$, $sN = 0.1$). The fourth sequence has been randomly generated ($rP = 0.1$), several sequences such as sequence number 8, have been structurally corrupted ($sN = 0.1$) and time bounds fluctuates due to the temporal noise ($tN = 0.2$).

Then, the problem of the QTI pattern miners is to retrieve the prototype from the noisy examples.

B.3.2 Results

All curves presented in the sequel were obtained by averaging the results on several (from 5 to 10) different datasets generated with the same parameters. Where not stated otherwise, the following default parameter values were used: $|\mathcal{D}| = 100$, $rP = 0.4$, $tN = 0.2$, $f_{min} = 0.1$ and $\epsilon = \infty$. We used the Hausdorff distance and looked for the temporal pattern $\{(A, _{[2,3]})\ (B, _{[5,5.5]})\ (C, _{[8,10]})\ (B, _{[12,12.5]})\}$.

Figures 3.6-(a) and 3.6-(b) compare the computation time of algorithms QTempIntMiner, QTIAPriori and QTIPrefixSpan with respect to the size of the sequence database, on the one hand, and to the size of hidden patterns, on the other hand. In both case, we can see that the computation times are exponential. In addition, QTIAPriori and QTIPrefixSpan are significantly faster than QTempIntMiner.

More precisely (cf. Figure 3.6-(c)), QTIPrefixSpan runs twice as fast as QTIAPriori, whether it is associated to KMeans or to Affinity Propagation (AP) [Frey and Dueck, 2007]. The KMeans clustering leads to better performances than the AP clustering. QTIAPriori-KMeans is even faster than QTIPrefixSpan-AP for patterns of size greater than 4. Figure 3.6-(d) illustrates the ability of QTIPrefixSpan-KMeans to cope with complexity. The computation time rises exponentially with respect to the number of sequences but the rate is equal to 1.26, so it is quite

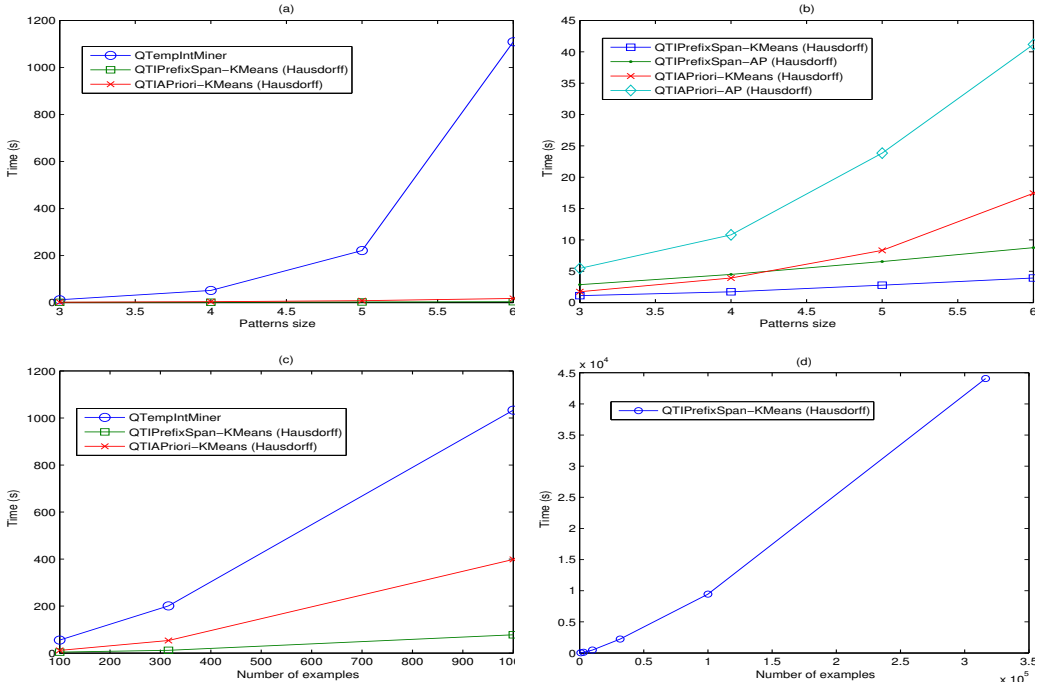


Figure 3.6: Mean computing time (in seconds) with respect to the size of hidden temporal patterns (a and b) and to the number of sequences (c and d).

close to 1. The mean time for extracting temporal patterns of size 4 from a database of 100,000 sequences, corresponding to 600,000 symbols, is about 15 min.

The quality of patterns extracted by QTIAPriori and QTIPrefixSpan is evaluated with respect to several criteria: i) the presence or absence among the extracted patterns of some pattern that ϵ-covers the prototype; ii) the dissimilarity between the temporal intervals in the prototype and in the extracted patterns, i.e. the precision of temporal intervals associated to extracted patterns; iii) the number of extracted patterns having the same symbolic signature as the prototype's, so called "clones"[4]. This last criterion concerns results readibility: the lower the number of clones, the easier the results analysis will be to the user.

The main factors that impact these criteria are the temporal noise level tN, the dissimilarity threshold ϵ and the minimal frequency threshold f_{min}. These factors are related: if data are very noisy then a high ϵ will be needed to insure that noisy instances are classified in the same cluster. But if ϵ is too high (with respect to tN) then the clusters will be larger and pruning using f_{min} will be less efficient and this will affect the overall algorithm performances.

For sufficiently high values of ϵ and low values of f_{min} the three algorithms can extract the prototype pattern from the sequence dataset. Moreover, when we add a second prototype having the same symbolic signature but different temporal intervals the two patterns are still correctly extracted. In the latter case, GSP or PrefixSpan can extract only one pattern.

Figure 3.7 illustrates the quality of patterns that are extracted by the algorithms from simulated data that contains only one prototype among random sequences. Figure 3.7-(a) gives, for QTIPrefixSpan, the total number of frequent patterns FP and the number of prototype clones (same symbolic signature and temporal constraints closest than ϵ to the prototype temporal constraints). When ϵ is small ($\epsilon < .1$), few patterns having the size of the prototype are generated: the expected similarity of pattern instances imposes a too strong constraint with respect to noise. For $.1 < \epsilon < .8$, the number of generated patterns grows very much. Due to the strong similarity that is still required by the value of ϵ, the temporal clustering induces many clusters but with a number

[4]Our method may extract several frequent patterns with the same symbolic signature but with discriminative temporal intervals.

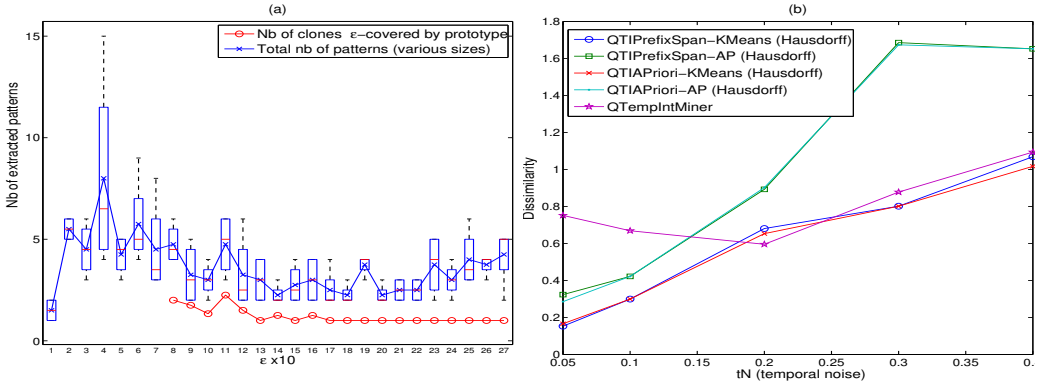


Figure 3.7: (a) mean number of extracted patterns with respect to ϵ for QTIPrefixSpan-KMeans. (b) mean dissimilarity between the best extracted pattern and the prototype with respect to the temporal noise.

of instances that remains below f_{min}. For $.8 < \epsilon < 1.7$, several frequent patterns are extracted. For $1.7 < \epsilon$, the number of extracted frequent patterns stabilizes to 1: ϵ is sufficiently large for considering that the dissimilarity between temporal intervals is quite reasonable.

We can see on Figure 3.7-(b) that even with noise, the best temporal pattern extracted by each algorithm is close to the hidden prototype. A Hausdorff distance equal to 1 is very low for a temporal pattern where temporal intervals may last 12 time units. For QTIAPriori and QTIPrefixSpan, when the temporal noise moves up, the precision logically moves down. We can see also, that the choice of the clustering method has more impact on the quality of results than the choice of the search method (GSP or PrefixSpan). KMeans can cope with harder temporal noise than AP: the mean dissimilarity obtained with KMeans is less than that obtained with AP. Finally, QTempIntMiner is more precise than the two other algorithms for large patterns: the quality induced by a clustering based on EM is better than that of KMeans or AP.

B.4 Discussion

In this section, we propose some additional discussion about the quality of QTI patterns extracted by QTempIntMiner and QTIPrefixSpan.

B.4.1 Extracting patterns with twin-items

Other experiments have been performed to illustrate that our algorithm can extract (1) temporal patterns with temporal items that overlap or (2) temporal patterns with twins, i.e. several identical events (associated with different intervals). Contrary to existing approaches, e.g. Dousson and Duong [1999], such temporal patterns do not introduce difficulties to QTIPrefixSpan thanks to the hyper-cube representation that separates the temporal information for each item. In the case of temporal patterns with twins, two dimensions can be separated if two representative temporal extents are extracted.

Thus, the QTI pattern illustrated is Figure 3.3 can be extracted by our algorithms, and most other algorithms can not.

B.4.2 About representativeness of interval-based temporal patterns

QTempIntMiner and QTIPrefixSpan are based on the notion of "representative pattern". The way a representative pattern is extracted strongly impacts the quality of the results.

In the QTempIntMiner algorithm, we developed an approach whose representative pattern is created from the statistical distributions using an *ad-hoc* principle and based on statistical hypothesis (Gaussian distribution assumption). But, in some cases, the hypothesis is false, and it is interesting to study the results to better understand the limits of the "representativeness approach".

We identified two cases where our assumption is false. The first one occurs when the variance of the lower (or upper) bound of a prototype interval is high compared to the duration of this interval (see Figure 3.8, left). In this case, the extracted temporal interval is larger than expected. The second case occurs when the variance of the lower bound of a prototype interval is low compared to the variance of its duration (see Figure 3.8, right). In this case, temporal interval is shifted. In fact, the distribution is not Gaussian anymore: it is rather exponential. As a consequence, the data set must be such that the variance of interval lower bounds must be *low* but *sufficiently high* compared to their respective duration variance.

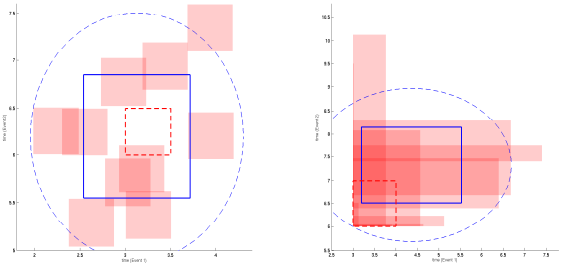


Figure 3.8: Two cases of inadequate interval extraction. Plain line rectangles show the temporal bounds extracted from the computed Gaussians (ellipsis). Dashed line rectangles show the expected temporal bounds.

The distance based clustering used in the QTIPrefixSpan aims at solving two problems encountered using the QTempIntMiner approach. First, QTIPrefixSpan does not create enlarged or shifted patterns that do not actually exist in the data. Clustering algorithms based on the notion of *medoid* enables to have a representative of the cluster that really exists. That is the case for KMedoid and AP algorithms. Second, QTIPrefixSpan introduces more flexibility in the notion of "representativeness" by allowing the user to choose its similarity measure.

B.4.3 Multiple projections

Multiple projection designates the fact that a temporal projection of a sequence on a symbolic signature may lead not to a unique multi-dimensional interval but on a set of multi-dimensional intervals. For instance, the symbolic signature AB has two temporal projection on the sequence $\langle (A, {}_{[1,2]})\ (B, {}_{[4,5]})\ (C, {}_{[4.5,6]})\ (B, {}_{[5,7]})\rangle$: $\langle [1,2],[4,5]\rangle$ and $\langle [1,2],[5,7]\rangle$.

If not all temporal projections are considered then we may miss some patterns because some recurrent multi-temporal intervals are hidden by other projection and do not appear as frequent during the clustering step.

Taking into account all projections may be computationally hard. Repetitions of an item in a sequence leads to generate a very high number of multi-dimensional interval to cluster (more than the number of sequences). The computational cost is thus important. For some datasets with item repetitions, the only tractable solution is to consider only one temporal projection.[5]

Considering that datasets we process contains small sequences with repetitions, we simplify our algorithm by taking into account only the first computed temporal projection in our application.

It is interesting to notice that with this simplification we skipped an interesting but difficult problem we faced later when mining discriminant chronicles. This problem comes from the differences between the clustering, that enumerates temporal projections, and the mining process that enumerates sequences. But if multiple projections is possible in a single sequence, enumerating temporal projections overestimates the number of sequences. This problem is well-known in machine learning and is a "multiple instance" machine learning task. We will come back on this problem in the Section C.5.1.

[5]The problem of choosing one temporal projection remains open.

B.4.4 Interpretation of empty periods

One issue encountered with pattern mining is the potential misinterpretation of patterns due the fact that they only shows what does happen, but does not inform about what did not happen.

In case of a sequential pattern, e.g. $\boldsymbol{p} = \langle A\, B\rangle$, some events may occur between the occurrences of A and B (e.g. $\boldsymbol{p}$ occurs in $\langle A\ C\ C\ B\rangle$).

In case of a temporal patterns, e.g. $\boldsymbol{p} = \langle (A, {}_{[1,2]})\,(B, {}_{[4,5]})\rangle$ there is an ambiguity behind the absence of event during period $[2, 4]$. On the one side, it may mean that in a frequent set of sequences, there is no event between A with time span $[1, 2]$ and B with time span $[4, 5]$. On the other side, it may mean that no event frequently occur between occurrences of A and B (but some event occurs).

The practical solution we adopted to disambiguate the absence of event in temporal pattern consists in fulfill each sequence of the dataset with a specific event stating the absence of any other regular event. By doing this, we increase significantly to number of events and increase the mining hardness.

Section D proposes a solution to address explicitly the absence of events.

B.4.5 Clustering *vs* Separating

We developed an alternative approach of our algorithm based on the idea of interval **separation** instead of interval **clustering**. This approach tackles both the problem of "representative pattern" and the anti-monotonicity constraint.

The difference between these two approaches is illustrated in Figure 3.9. In the interval clustering approach, similar interval occurrences illustrated by black horizontal lines are cluster to create two patterns with different temporal characteristics (yellow symbols 1 and 2). We have seen that, at the beginning of the process, i.e. without conditional information from co-occurrence of other events, the clustering may create not really interesting patterns due to the data noise or temporal variability of the intervals. But, if cluster exists then, it prunes efficiently the search space

The separate approach splits interval occurrences only when they are "well-separated" from the other intervals. For instance, two set of intervals are separated iff there exists a wide-enough gap in the interval distribution. Such a criteria is interesting for two computational reasons:

- ▷ it is monotonic: if intervals of event A are "well-separated" in the dataset, they will be "well-separated" also in the interval dataset projected on a larger symbolic signature (e.g. AB)
- ▷ it is easy to compute: on the contrary to the clustering, this criteria is computed in a linear complexity with the number of intervals
- ▷ it does not require any "representative". Similarly to a PrefixSpan approach, the pattern is a symbolic signature with a projected database representing the multi-dimensional intervals.

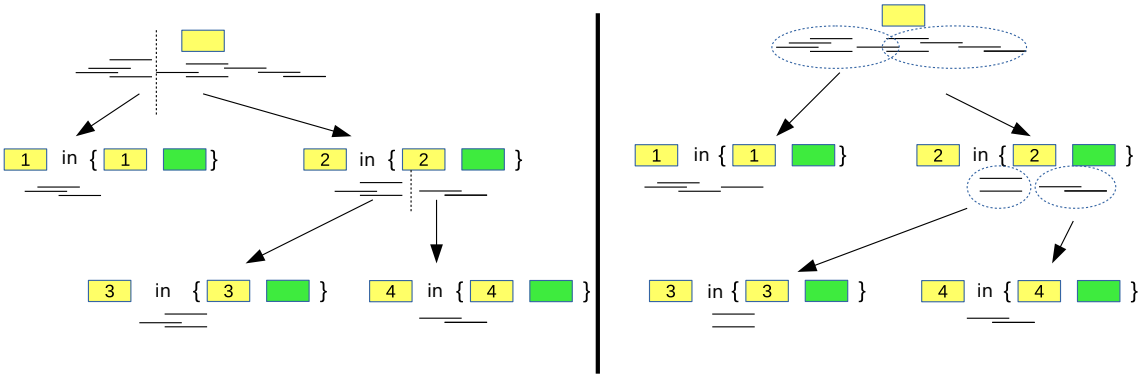


Figure 3.9: Illustration of the difference between cluster (on the right) and separate (on the left) approaches.

Figure 3.10 illustrates the first results obtained with separation strategy. Synthetic datasets are those generated in Section B.3.1. Unsurprisingly, the computation time is reduced of one order

of magnitude and the precision of the extracted pattern is lower (the considered pattern is the one specified by separation boundaries). As shown in Figure 3.10 on the left, the boundaries does not match the hidden pattern, but remember that extracting accurate boundaries remains possibles by clustering intervals.

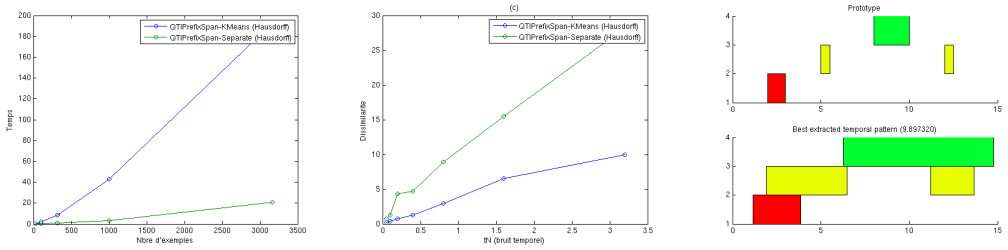


Figure 3.10: Computing time wrt to the number of examples (on the left) and best pattern distance with respect to temporal noise (in the middle) for two approaches: QTIPrefixSpan in blue vs separate strategy, in green. Figure on the right illustrates the hidden pattern and the extracted pattern with the separation strategy.

To improve the quality of temporal patterns, the final set of projections maybe post-process to extract "representative" intervals. Thus, representatives are only used to present the pattern to the user but it has no impact on extracted patterns.

In this section, we presented two algorithms to extract quantitative temporal patterns from interval based sequences. On the one side, the existing approaches to handle interval-based data extract temporal patterns with qualitative temporal information and thus are less precised. On the other side, applying mining techniques extracting quantitative temporal information from sequences of punctual events required to transform interval-based temporal sequences in punctual events temporal sequences. This transformation may lead to misinterpretation. In both cases, the proposed algorithms enables to reveal meaningful information that the other approaches can not extract.

C Discovering discriminant temporal constraints[6]

In this section, we consider labeled temporal sequences (Table 3.1 illustrates a dataset that is explore in this section):

- ▷ sequences are made of timestamped events;
- ▷ labels associate a sequence to class.

For instance, a sequence may be a care pathway and the class indicates whether the patient has been cured or not.

In such a case, the objective is to extract behaviors that characterize for each sequences class. A possible approach consists in extracting frequent patterns for each set of sequences (one set per class) and to post-process pattern sets to extract meaningful information. We prefer to mine the whole dataset and to extract directly discriminant patterns, i.e. patterns that appears more in a class than in the others. In our case, a pattern is discriminant if it appears g_{min} times more in one sequence class than in others.

Discriminant pattern mining has been addressed in the literature with different names: emerging patterns [Dong and Li, 1999], constrast patterns [Bay and Pazzani, 2001], subgroup discovery [Novak et al., 2009]. Fradkin and Mörchen [2015] proposed discriminant pattern mining for sequences inspired by the framework of Bringmann et al. [2009] on pattern based classification.

Remark 6 (Discriminant patterns and rule patterns)**.** *Mining discriminant patterns is different from mining rules from sequential patterns. Both aims at doing some classifications, but discrim-*

[6]This part is mainly the Y. Dauxais PhD work, supervised by D. Gross-Amblard, A. Happe and myself. It also benefits from a collaborative work with P. Besnard (CNRS/IRIT).

Table 3.1: Set of six sequences labeled with two classes $\{+, -\}$.

SID	Sequence	Label
1	$(A, 1)$, $(B, 3)$, $(A, 4)$, $(C, 5)$, $(C, 6)$, $(D, 7)$	$+$
2	$(B, 2)$, $(D, 4)$, $(A, 5)$, $(C, 7)$	$+$
3	$(A, 1)$, $(B, 4)$, $(C, 5)$, $(B, 6)$, $(C, 8)$,$(D, 9)$	$+$
4	$(B, 4)$, $(A, 6)$, $(E, 8)$, $(C, 9)$	$-$
5	$(B, 1)$, $(A, 3)$, $(C, 4)$	$-$
6	$(C, 4)$, $(B, 5)$, $(A, 6)$, $(C, 7)$, $(D, 10)$	$-$

inant patterns predict a label while rules predict the type of future events. Then, mining temporal association rules [Sacchi et al., 2007] is different from our objective.

Sequence classification is encountered in numerous application fields: comparing groups of clients (e.g. large family *vs* single child family), analyzing data related to supermarket purchases, identifying behavior of customers (who churns *vs* loyal customers) or characterize care pathways specificities for patients who have a disease. In all these contexts, the temporal dimension may hold the key information to discriminate sequences from one class to the other. For instance, a short delay between the delivery of two adverse drugs may help discriminate sick from healthy patients. By taking quantitative temporal constraints into account, we aim at improving classification accuracy, but discriminant patterns will also provide better insights about discriminant care sequences.

The problem that is addressed in this section is the **enhancement of discriminant patterns with temporal dimension**. Indeed, as we strongly believe that temporal information may be meaningful to characterize dynamic systems, it has to be meaningful to discriminate different types of behaviors observed in a dataset.

The contribution of Y. Dauxais PhD thesis is the extraction of discriminant temporal constraints. The main idea may be roughly introduced as follow: QTempIntMiner or QTIPrefixSpan extract temporal constraints by adding a temporal characterization of a symbolic signature by *clustering* its temporal projections. Discriminant temporal pattern mining proposes to use a supervised learning technique to extract temporal constraints to characterize a symbolic signature.

More precisely, we investigate the pattern-domain of chronicles. A chronicle is a temporal graph whose vertices are items and whose edges are numerical temporal constraints between items occurrences. This kind of pattern have been introduced to monitor stream of events [Dechter et al., 1991; Dousson and Maigat, 2007] and several algorithms have been proposed to extract frequent chronicles from databases [Dousson and Duong, 1999; Cram et al., 2012; Alvarez et al., 2013; Huang et al., 2012]. It is very expressive and easy to understand by non-expert. Thus, it is very interesting for approaches whose objective is to bring insight about the data to the analyst.

The contribution of this work is twofold:

1. we propose the new setting of mining discriminant chronicles, from labeled sequences of timestamped events.
2. we propose the *DCM* algorithm, which relies on rule learning algorithm to extract discriminant temporal constraints. To the best of our knowledge, this is the first approach that extracts discriminant patterns with quantitative temporal information.

C.1 Chronicles

Let us first introduce the pattern domain of chronicles. We first give a formal definition of chronicle and its graphical representation. Then, we adapt the classical definition of partial order and support of a chronicle in a temporal sequence to fit the definition of Section A of Chapter 2.

Definition 23 (Chronicle: multiset and temporal constraints)**.** *A **chronicle** is an ordered pair $\mathscr{C} = (\mathcal{E}, \mathcal{T})$ where*

- ▷ *$\mathcal{E}$ is a finite ordered **multiset**, i.e., $\mathcal{E}$ is of the form $\{\!\{e_1, \ldots, e_n\}\!\}$ (in which repetitions are allowed) such that*

  - $e_i \in \mathbb{E}$ *for* $i = 1, \ldots, n$
  - $e_1 \leq_{\mathbb{E}} \cdots \leq_{\mathbb{E}} e_n$;

▷ $\mathcal{T}$ *is a set of* ***temporal constraints****, i.e, expressions of the form* $(e, o_e)[t^- : t^+](e', o_{e'})$ *such that* [7]
  - $e, e' \in \mathcal{E}$
  - $t^-, t^+ \in \overline{\mathbb{R}}$
  - $o_e, o_{e'} \in [n]$
  - $t^- \leq t^+$
  - $o_e < o_{e'}$
  - $e_{o_e} = e$
  - $e_{o_{e'}} = e'$.

In the above definition of a chronicle as an ordered pair $\mathscr{C} = (\mathcal{E}, \mathcal{T})$, the multiset $\mathcal{E}$ is required to be finite but there is no such requirement for $\mathcal{T}$. In addition, in the most general case, nothing forbid to have several constraints between the same pair of events.

Nonetheless, in the following, we assume there is at most one temporal constraint between two events of the multiset.[8] Hence, $\mathcal{T}$ is finite. This limitation is made by all chronicle mining algorithm.

Example 15. *Graphical representation of chronicles*

Figure 3.11 is a graphical representation of the chronicle $(\mathcal{E}, \mathcal{T})$ *where* $\mathcal{E} = \{\!\{ABBC\}\!\}$ *and*

$$\mathcal{T} = \left\{ \begin{array}{c} (A,1)[1:5](B,2) \\ (A,1)[9:15.5](B,3) \\ (B,2)[7:+\infty](B,3) \\ (A,1)[-5:-3](C,4) \end{array} \right\}.$$

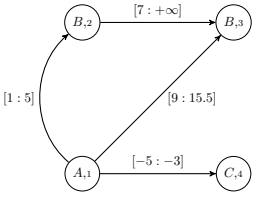


Figure 3.11: Graphical representation of the chronicle of Example 15.

Definition 24 (Preorder on chronicles)**.** *Let* $\mathscr{C} = (\mathcal{E}, \mathcal{T})$ *and* $\mathscr{C}' = (\mathcal{E}', \mathcal{T}')$ *be chronicles.* $\mathscr{C}$ *is* ***at least as specific as*** $\mathscr{C}'$*, to be denoted* $\mathscr{C}' \preceq \mathscr{C}$*, iff* $\mathcal{E}' \Subset \mathcal{E}$ *by some multiset embedding* θ *such that for each* $(e'_i, o)[l' : u'](e'_j, p)$ *in* $\mathcal{T}'$ *there is a temporal constraint* $(e_{\theta(i)}, \theta(o))[l : u](e_{\theta(j)}, \theta(p))$ *in* $\mathcal{T}$ *satisfying*[9] $(e_{\theta(i)}, \theta(o))[l : u](e_{\theta(j)}, \theta(p)) \trianglelefteq (e'_i, o)[l' : u'](e'_j, p)$*, where* $(e_1, o_1)[l : u](e_2, o_2) \trianglelefteq (e'_1, o'_1)[l' : u'](e'_2, o'_2)$ *iff* $e_1 = e'_1, e_2 = e'_2$ *and* $[l : u] \subseteq [l' : u']$.

Remark 7 (Poset but not lattice)**.** *Contrary to the intuition, the set of chronicles* $(\mathcal{C}, \preceq)$ *is not a semi-lattice but only a poset [Besnard and Guyet, 2019].*

We now deal with a formal definition underlying the notion of chronicles as patterns to be found in event sequences.

Definition 25 (Chronicle occurrences and embeddings)**.** *An* ***occurrence*** *of a chronicle* $\mathscr{C} = (\{\!\{e'_1, \ldots, e'_m\}\!\}, \mathcal{T})$ *in a stripped sequence* $\boldsymbol{s} = \langle (e_1, t_1), \ldots, (e_n, t_n) \rangle$ *is a subset* $\{(e_{f(1)}, t_{f(1)}), \ldots, (e_{f(m)}, t_{f(m)})\}$ *of* $\boldsymbol{s}$ *such that*

1. $f : [m] \to [n]$ *is an injective function,*
2. $f(i) < f(i+1)$ *whenever* $e'_i = e'_{i+1}$,
3. $e'_i = e_{f(i)}$ *for* $i = 1, \ldots, m$,
4. $t_{f(j)} - t_{f(i)} \in [t^- : t^+]$ *whenever* $(e'_i, i)[t^- : t^+](e'_j, j) \in \mathcal{T}$.

We call f *an* embedding.

A chronicle $\mathscr{C}$ occurs *in* $\boldsymbol{s}$*, denoted* $\mathscr{C} \preceq \boldsymbol{s}$*, iff there is at least one occurrence of* $\mathscr{C}$ *in* $\boldsymbol{s}$.

A temporal projection *of* $\mathscr{C}$ *on* $\boldsymbol{s}$ *is the* m*-dimensional vector* $\{t_{f(1)}, \ldots, t_{f(m)}\}$

It is worth noting that f usually fails to be increasing. In fact, there is a difference between (i) the order $\lessdot$ on the events in the sequence (see Definition 12, page 17), and (ii) the order $\leq_{\mathbb{E}}$ on the event types in the multiset of the chronicle.

[7] $\overline{\mathbb{R}}$ is to denote $\mathbb{R}$ extended with two elements, $-\infty$ and $+\infty$, resp. least and greatest for $\leq$.

[8] We refer the reader to the collaborative work with P. Besnard for a deep understanding of chronicles [Besnard and Guyet, 2019]. In [Besnard and Guyet, 2019], the chronicles used in this manuscript are called simple chronicles.

[9] Since θ is a multiset embedding for $\mathcal{E}'$ into $\mathcal{E}$, both $e'_i = e_{\theta(i)}$ and $e'_j = e_{\theta(j)}$.

As is naturally expected, it can happen that a chronicle has several occurrences in the same temporal sequence: see the next example (depicted in Figure 3.12).

Example 16. *Two occurrences of a chronicle in a temporal sequence*

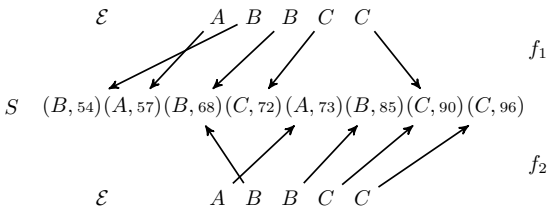


Figure 3.12: Chronicle occurrences in a temporal sequence (Example 16).

Let $\boldsymbol{s}$ *be the temporal sequence* $\langle (B,54)\ (A,57)\ (B,68)\ (C,72)\ (A,73)\ (B,85)\ (C,90)\ (C,96)\rangle$
Consider the chronicle $\mathscr{C} = (\mathcal{E}, \mathcal{T})$ *where* $\mathcal{E} = \{\!\{ABBCC\}\!\}$ *and*

$$\mathcal{T} = \left\{ \begin{array}{c} (A,1)[-6,-1](B,2) \\ (A,1)[10,12](B,3) \\ (B,2)[25,50](C,4) \\ (B,3)[3,7](C,5) \end{array} \right\}$$

In view of f_1 *and* f_2*,*

f_1 :

$[m]$	$\to$	$[n]$
1	$\mapsto$	2
2	$\mapsto$	1
3	$\mapsto$	3
4	$\mapsto$	4
5	$\mapsto$	7

i.e.

$\mathcal{E}$	$\to$	S
$(A,1)$	$\rightsquigarrow$	$(A,57)$
$(B,2)$	$\rightsquigarrow$	$(B,54)$
$(B,3)$	$\rightsquigarrow$	$(B,69)$
$(C,4)$	$\rightsquigarrow$	$(C,72)$
$(C,5)$	$\rightsquigarrow$	$(C,90)$

f_2 :

$[m]$	$\to$	$[n]$
1	$\mapsto$	5
2	$\mapsto$	3
3	$\mapsto$	6
4	$\mapsto$	7
5	$\mapsto$	8

i.e.

$\mathcal{E}$	$\to$	S
$(A,1)$	$\rightsquigarrow$	$(A,73)$
$(B,2)$	$\rightsquigarrow$	$(B,68)$
$(B,3)$	$\rightsquigarrow$	$(B,85)$
$(C,4)$	$\rightsquigarrow$	$(C,90)$
$(C,5)$	$\rightsquigarrow$	$(C,96)$

it is clear that there are two occurrences of $\mathscr{C}$ *in* $\boldsymbol{s}$*.*

Where arrows cross in Figure 3.12, f_k *is not increasing (indeed, Condition 1. in Definition 25 precludes crossing of arrows just in case the event type is the same).*

Definition 26 (Chronicle support)**.** *The* ***support*** *of a chronicle* $\mathscr{C}$ *in a sequence set* $\mathcal{S}$ *is the number of sequences in which* $\mathscr{C}$ *occurs:*

$$supp(\mathscr{C}, \mathcal{S}) = |\{\boldsymbol{s} \in \mathcal{S} \mid \mathscr{C} \preceq \boldsymbol{s}\}|$$

Example 17. *Chronicle* $\mathscr{C}$ *(see Figure 3.13 at the top left), occurs in sequences* 1, 3 *and* 6 *of Table 3.1 (the columns Label may be ignored in this example). We notice there are two occurrences of* $\mathscr{C}$ *in sequence* 1*. Nonetheless, its support is* $supp(\mathscr{C}, \mathcal{S}) = 3$*. The two other chronicles, denoted* $\mathscr{C}_1$ *and* $\mathscr{C}_2$*, occur respectively in sequences* 1 *and* 3*; and in sequence* 6*. Their supports are* $supp(\mathscr{C}_1, \mathcal{S}) = 2$ *and* $supp(\mathscr{C}_2, \mathcal{S}) = 1$*.*

The support of chronicles is anti-monotone using the previously defined partial order $\preceq$. Indeed for two chronicles $\mathscr{C}_1$ and $\mathscr{C}_2$ and a sequence set $\mathcal{S}$, $\mathscr{C}_1 \preceq \mathscr{C}_2$ implies $supp(\mathscr{C}_1, \mathcal{S}) \leq supp(\mathscr{C}_2, \mathcal{S})$. This property derives from the fact that the most specific chronicle $\mathscr{C}_1$ can not occur in more sequences than $\mathscr{C}_2$ because it is more constrained. This property has been formally proved and used in previous works on chronicles [Dousson and Duong, 1999; Cram et al., 2012; Huang et al., 2012; Alvarez et al., 2013].

From the global point of view of our contributions we presented at the beginning of this chapter, the multiset of a chronicle can be seen as the *symbolic signature* of the pattern. The *temporal projection* of a chronicle on a sequence is given by the set of timestamps of the sequence events matching the chronicle in an embedding. More precisely, a chronicle models the delays between events. Then, the frequent chronicle mining algorithms can be related to QTempIntMiner or

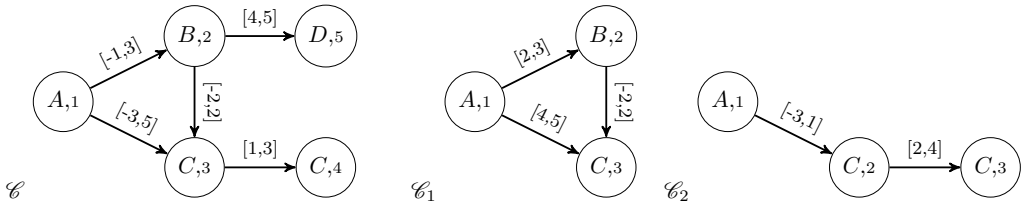


Figure 3.13: Example of three chronicles occurring in Table 3.1 (cf. Examples 17). No edge between two events is equivalent to the temporal constraint $[-\infty, \infty]$.

QTIPrefixSpan, but instead of clustering the delays between the events of embeddings as the TGSP algorithm [Yen and Lee, 2013], the algorithms use the convex envelop to generalize the delays.

In this work, we blend the view of a temporal pattern (QTI pattern or chronicle) as a symbolic signature combines with temporal information generalizing temporal projections, and the objective to have discriminant temporal information. Thus, instead of generalizing the temporal projections, we could learn discriminant temporal projections.

C.2 Discriminant chronicles

A preliminary definition of the discriminant mining task is: Given a growth rate threshold g_{min} and a dataset $\mathcal{D}$ of labeled sequences, mining discriminant chronicles is extracting all chronicles $\mathcal{C}$ from $\mathcal{D}$ that have a growth-rate above g_{min}, but chronicles with a low support in the positive set of sequences and that do not occur in the negative ones are both numerous and meaningless.

Then, we redefine the discriminant mining task as follow.

Definition 27. *Let $\mathcal{D}$ be two sets of sequences, and $\sigma_{min} \in \mathbb{N}^+$, $g_{min} \in [1, \infty]$ be two parameters. A chronicle $\mathscr{C}$ is **discriminant** for label $l \in \mathbb{L}$ iff*

1. *$supp(\mathscr{C}, \mathcal{D}^l) \geq \sigma_{min}$,*
2. *$supp(\mathscr{C}, \mathcal{D}^l) \geq g_{min} \times supp(\mathscr{C}, \mathcal{D} \setminus \mathcal{D}^l)$ and*
3. *$supp(\mathscr{C}, \mathcal{D} \setminus \mathcal{D}^l) > 0$,*

where $\mathcal{D}^l$ is the set of sequence with label l.

In the above definition, a minimum support constraint (with an additional parameter σ_{min}) prevents from generating unfrequent, and so insignificant, chronicles. Pruning unfrequent chronicles is efficient thanks to the anti-monotonicity of support. More specifically, if a chronicle[10] $(\mathcal{E}, \mathcal{T}_\infty)$ is not frequent, then no chronicle of the form $(\mathcal{E}, \mathcal{T})$ will be frequent. This means that temporal constraints may be extracted only for frequent multisets. Contrary to the minimal support constraint, the minimal growth constraint is not anti-monotone.

Example 18. *With chronicle $\mathscr{C}$ of Figure 3.13, $supp(\mathscr{C}, \mathcal{D}^+) = 2$, $supp(\mathscr{C}, \mathcal{D}^-) = 1$, where $\mathcal{D}^+$ (resp. $\mathcal{D}^-$) is the sequence set of Table 3.1 labeled with $+$ (resp. $-$). Then, $\mathcal{C}$ is discriminant if $g_{min} \leq 2$. For chronicles $\mathscr{C}_1$ and $\mathscr{C}_2$, $supp(\mathscr{C}_1, \mathcal{D}^-) = 0$ then it is not discriminant due to the third constraint and $supp(\mathscr{C}_2, \mathcal{D}^+) = 0$ and $supp(\mathscr{C}_2, \mathcal{D}^-) = 1$ then $\mathscr{C}_2$ is not discriminant, but $\mathscr{C}_1$ is for any g_{min} value.*

C.3 *DCM* algorithm

Complete search strategy may be proposed to extract the complete set of discriminant and frequent chronicles, but it appears to be practically intractable considering the huge number of possible chronicles to extract.

In the Y. Dauxais work [Dauxais, 2018], we explored several strategies to find a good tradeoff between computational efficiency and accuracy/representativeness of chronicles. Several strategies have been explored:

[10] $\mathcal{T}_\infty$ is the set of temporal constraints with all bounds set to ∞.

Algorithm 5: Algorithm *DCM* for discriminant chronicles mining

Input: $\mathcal{S}^+$, $\mathcal{S}^-$: sequences sets, σ_{min} : minimal support threshold, g_{min} : minimal growth threshold

1 $\mathbb{M} \leftarrow$ ExtractMultiSets($\mathcal{S}^+$, σ_{min}) // $\mathbb{M}$ is the frequent multiset set
2 $\mathbb{C} \leftarrow \emptyset$ // $\mathbb{C}$ is the discriminant chronicle set
3 **foreach** $ms \in \mathbb{M}$ **do**
4 **if** $supp(\mathcal{S}^+, (ms, \mathcal{T}_\infty)) > g_{min} \times supp(\mathcal{S}^-, (ms, \mathcal{T}_\infty))$ **then**
   // Discriminant chronicle without temporal constraints
5 $\mathbb{C} \leftarrow \mathbb{C} \cup \{(ms, \mathcal{T}_\infty)\}$;
6 **else**
7 $\mathcal{M} \leftarrow$ ExtractDiscrConstraints($\mathcal{S}^+$, $\mathcal{S}^-$, ms, g_{min}, σ_{min}) ;
8 **forall** $\mathcal{T} \in \mathcal{M}$ **do**
9 $\mathbb{C} \leftarrow \mathbb{C} \cup \{(ms, \mathcal{T})\}$ // Add a new discriminant chronicle
10 **return** $\mathbb{C}$

- ▷ a complete algorithm traversing the search space first to extract frequent chronicles (from bottom to top) and then pruning non-discriminant ones (from top to bottom).
- ▷ using formal concept analysis to extract closed set of frequent chronicles in order to reduce the number of chronicles [Guyet et al., 2017b].
- ▷ heuristic approaches based on rule learning algorithm to extract the most accurate chronicles

In the section, we present *DCM* [Dauxais et al., 2019], an algorithm which implement the third approach. *DCM* extracts discriminant chronicles in two steps (see Algorithm 5): the extraction of frequent multisets, and then the specification of temporal constraints for each non-discriminant multiset.

At first, line 1 of Algorithm 5, ExtractMultiSets extracts, $\mathbb{M}$, the frequent multiset set in $\mathcal{S}^+$. In a second step, lines 3 to 8 of Algorithm 5 extract the discriminant temporal constraints of each multiset. The naive approach would be to extract discriminant temporal constraints for all frequent multisets. A multiset $\mathcal{E}$ (i.e. a chronicle $(\mathcal{E}, \mathcal{T}_\infty)$) which is discriminant may yield numerous similar discriminant chronicles with most specific temporal constraints. We consider them as useless and, as a consequence, line 4 tests whether the multiset ms is discriminant. If so, $(ms, \mathcal{T}_\infty)$ is added to the discriminant patterns set. Otherwise, lines 7-8 generate chronicles from discriminant temporal constraints identified by ExtractDiscrConstraints.

The general idea of ExtractDiscConstraints is to extract discriminant temporal constraints using a classical numerical rule learning task (e.g. [Cohen, 1995]). Similarly to QTempIntMiner and QTIPrefixSpan which extract "representative" temporal constraints using clustering algorithm, *DCM* aims at extracting discriminant temporal constraints using supervised classification algorithm from temporal projections of chronicles.

The choice of using the learning algorithm is motivated by the following points:

- ▷ input constraint: the algorithm has to extract a model from numerical vectors (it discards symbolic machine learning algorithms for instance);
- ▷ output constraint: the outputted models have to be translated in interval-based temporal constraints (machine learning algorithms using black-box models (such as SVM or neural network) are also discarded: the model has to be an explicit model);

The remaining learning models are decision trees or numerical rules. These two types of algorithms extract models as a conjunction of conditions that are inequalities (e.g. $attr \geq 10$). It turns out that these inegalities are temporal constraints of a chronicle.

More formally, let $\mathcal{E} = \{\!\{e_i\}\!\}$ be a frequent multiset. A relational[11] dataset, denoted $\mathcal{D}$, is generated by computing all temporal projections of $\mathcal{E}$ in $\mathcal{S}$. The numerical attributes of $\mathcal{D}$ are inter-event duration between each pair (e_i, e_j), denoted $\mathcal{A}_{e_i \to e_j}$. An example is labeled by its

[11] In some context, relational dataset designates is dataset whose organization is based on a complex relational model of data. In this context, the relational model is simply a unique relation.

SID	$\mathcal{A}_{A\to B}$	$\mathcal{A}_{B\to C}$	$\mathcal{A}_{A\to C}$	Label
1	2	2	4	+
1	−1	2	1	+
2	5	−2	3	+
3	3	0	3	+
5	−1	3	1	−
6	6	−1	5	−

Table 3.2: Relational dataset for the multiset $\{\!\{A, B, C\}\!\}$.

sequence label ($L \in \mathbb{L} = \{+, -\}$). If a sequence has several occurrences of $\mathcal{E}$, then each occurrence yields one example.

A rule learning algorithm induces numerical rules from $\mathcal{D}$. A rule has a label in conclusion and its premise is a conjunction of conditions on attribute values. Conditions are inequalities in the form: $\mathcal{A}_{e_i\to e_j} \geq x \wedge \mathcal{A}_{e_i\to e_j} \leq y$, where $(x, y) \in \overline{\mathbb{R}}^2$. Such rule is then translated as a set of temporal constraints, $\mathcal{T} = \{(e_i, i)[x, y](e_j, j)\}$. The couple $(\mathcal{E}, \mathcal{T})$ is then a potential discriminant chronicle. At this stage, we are not sure that the chronicle is discriminant for the sequences. In fact, the rule learning extracts discriminant temporal constraints based on the dataset $\mathcal{D}$. The multiple-instances of the multiset is managed a posteriori (see section C.5.1).

Example 19 (Extracting discriminant temporal constraints)**.** *Table 3.2 is the relational dataset obtained from the occurrences of* $\{\!\{A, B, C\}\!\}$ *in Table 3.1. The attribute* $\mathcal{A}_{A\to B}$ *denotes the duration between A and B. We notice that several examples can come from the same sequence.*

The rule $\mathcal{A}_{A\to B} \leq 5 \wedge \mathcal{A}_{B\to C} \leq 2 \implies +$ *perfectly characterizes the examples labeled by + in Table 3.2. It is translated into the discriminant temporal constraints* $\{(A,1)[-\infty, 5](B,2), (B,2)[-\infty, 2](C,3)\}$ *which gives the discriminant chronicle* $\mathcal{C} = (\{\!\{A, B, C\}\!\}, \{(A,1)[-\infty, 5](B,2), (B,2)[-\infty, 2](C,3)\})$.

C.4 Experiments

This section reports the experiments that compare the prediction accuracy of DCM[12] with $BIDE\text{-}D$ [Fradkin and Mörchen, 2015], an algorithm that mines discriminant sequences. $BIDE\text{-}D$ is a pattern-based classifier Bringmann et al. [2009]. It extracts frequent sequential patterns and then a SVM classifier is trained on a dataset encoding sequences with absence/presence vectors of frequent sequential patterns. Then, the $BIDE\text{-}D$ model does not integrate quantitative temporal information. The objective of these experiments is to study how temporal information can improve classification accuracy. More experiments on synthetic data may be found in Dauxais [2018].

In these experiments, we use a subset of the data initially proposed to evaluate $BIDE\text{-}D$. These data come from a variety of applications. The subset of data has been chosen to be neither too simple like *blocks* where prediction rates approach 100% nor too difficult like *Auslan*2 where $BIDE\text{-}D$ approaches hardly exceed 30%. Then, we chose to focus on *asl-bu*, *asl-gt* and *context*.

The following results compare classification performances of discriminant chronicles with the discriminant sequences extracted by $BIDE\text{-}D$. The couples $\langle \mathscr{C}, L \rangle$ of discriminant chronicles, $\mathscr{C}$ and tags, L, are used to predict the label of a sequence. A label L is predicted for a sequence in case this sequence contains the chronicle $\mathscr{C}$. In the case of several chronicles appearing in the sequence, we will retain the label associated with the chronicle with the highest growth rate.

The results discussed below are presented in the Table 3.3. The maximal size of the chronicle multiset is set to 6 in order to ease their extraction and limit their number.

On *asl-bu* the results of DCM are somewhat lower than those of $BIDE\text{-}D$. The accuracy using $g_{min} = 2$ is equivalent from $\sigma_{min} = 0.4$ to $\sigma_{min} = 0.6$, whereas the number of extracted patterns is reduced from more than $30,000$ to $1,600$. The standard deviation is somewhat better for the accuracy of discriminant chronicles than for $BIDE\text{-}D$. We can notice that *asl-bu* is a hard type

[12] Y. Dauxais implemented DCM in C++. It relies on pre-existing implementations of LCM [Uno et al., 2004] to extract frequent multisets and $Ripper_k$ [Cohen, 1995] to extract rules. The main advantage of $Ripper_k$ is to extract *unordered* rules. Indeed, in ordered rules, a rule is valid only if the "previous" rules are not valid. In our case, each rule yield a chronicle. Thus, rules must be independent from each others, i.e. unordered.

Table 3.3: Evolution of accuracy (in %) depending on the parameters σ_{min}, g_{min} and the dataset. The standard deviation of the accuracy is specified in the parentheses.

		g_{min}			
dataset	σ_{min}	**2**	**3**	**4**	**5**
asl-bu	**0.4**	52.64 (±2.21)	49.20 (±0.51)	52.41 (±3.51)	48.28 (±1.99)
	0.5	51.26 (±1.31)	51.72 (±1.99)	49.66 (±6.82)	48.51 (±5.60)
	0.6	51.72 (±3.35)	50.34 (±3.58)	44.14 (±3.31)	39.08 (±3.72)
asl-gt	**0.2**	31.55 (±0.91)	31.61 (±0.92)	30.20 (±1.82)	30.15 (±0.92)
	0.3	31.17 (±0.44)	29.18 (±1.53)	27.75 (±1.58)	26.96 (±1.89)
	0.4	27.34 (±2.10)	25.82 (±0.42)	25.91 (±0.12)	25.32 (±0.19)
	0.5	25.44 (±0.34)	25.20 (±0.13)	24.68 (±0.50)	24.12 (±0.41)
	0.6	24.30 (±0.42)	23.92 (±0.53)	23.89 (±0.52)	23.13 (±0.44)
context	**0.2**	64.78 (±2.83)	57.39 (±4.76)	46.09 (±3.89)	53.48 (±6.07)
	0.3	56.09 (±5.83)	42.61 (±7.62)	52.61 (±3.22)	36.96 (±7.53)
	0.4	47.83 (±4.07)	39.57 (±3.57)	50.43 (±5.41)	47.39 (±4.96)
	0.5	53.91 (±4.46)	38.70 (±0.97)	30.43 (±5.10)	47.83 (±7.37)
	0.6	50.87 (±2.48)	34.78 (±4.35)	30.87 (±4.18)	28.70 (±4.46)

of dataset for chronicle mining due to multiple occurrences of same event types. As a result we do not obtain a result for $\sigma_{min} = 2$ or $\sigma_{min} = 3$.

On *asl-gt* the results are unfortunately very bad. Where accuracy range from 27.31% for $\sigma_{min} = 0.6$ to 82.94% for $\sigma_{min} = 0.2$ for *BIDE-D*, it hardly exceeds 30 for discriminant chronicles. The standard deviation is similar to that of *BIDE-D*. Many patterns are extracted but are poorly distributed among the dataset labels. These results may be due to the maximum size constraint of the chronicles or show the limit of the use of $Ripper_k$ in a multiple instance context.

Finally on *context*, the results therefore our best results since the results presented for the *BIDE-D* approaches oscillate between 27.08% and 53.47%. The standard deviation is higher than for the two other datasets but is still similar to this of *BIDE-D*. We note that the low results of *BIDE-D* approches are surely related to the use of a different minimum support threshold strategy for this dataset. The difference in the number of patterns extracted by the $\sigma_{min} = 0.2$ or $\sigma_{min} = 0.6$ parameters is much less than for *asl-bu*. It runs from 360 patterns for $\sigma_{min} = 0.2$ and $g_{min} = 2$ to 145 for $\sigma_{min} = 0.6$ and $g_{min} = 4$.

We can conclude from these results that the use of discriminant chronicles as rules can produce similar and sometimes better results in classification than the use of sequential patterns as features. The comparison were made with the best results obtained by the three approaches using directly sequential patterns as features (*BIDE*, *BIDE-D* and *BIDE-DC*) but we can notice that our approach produces often better results than *SMBT* and *SMBT-FS* which extract decision tree of discriminant sequential patterns to be used as features in a *SVM* classifier. Finally, we can obtain those results with an effortless setup where a grid search is needed to optimize the *SVM* parameters used in *BIDE-D*.

C.5 Discussions

To the best of our knowledge *DCM* is the first algorithm to extract discriminant quantitative temporal patterns, i.e. an algorithm which can discover quantitative temporal information for discriminating behaviors from temporal sequences. Inductive logic programming and subgroup discovery proposed algorithms to extract discriminant patterns. Our approach enhance them with a rich temporal information that is meaningful while time is a critical dimension in the data.

The approach of consisting in using a supervised machine learning algorithm to extract temporal constraints can be applied to a large collection of symbolic signature. Multiset as symbolic signature leads to extract chronicles models. The model of chronicles turns out to be a very interesting to investigate for several reasons:

- ▷ chronicles, derived from temporal networks [Dechter et al., 1991], are semantically very rich to represent complex dynamical behaviors in sequences of events
- ▷ chronicles enjoy some formal properties that make them computationally interesting for pattern mining
- ▷ chronicles has a graphical representation and they are quickly adopted by users. They answer the recent urge of interpretable models.

In addition, from the purely scientific point of view, the set of chronicles surprisingly has hidden complexity which make them very interesting to investigate.

While investigating discriminant chronicle mining we encountered some interesting difficulties that seems interesting to discuss in the following.

C.5.1 Problem of multiple instances

Let us start by introducing the problem of multiple instance learning in the case of discriminant chronicle mining with an example continuing Example 19.

Example 20 (Multiple instance (continuing Example 19))**.** *We remind that Table 3.2 illustrates the temporal projection of* $\{\!\{A, B, C\}\!\}$ *in the sequences of Table 3.1. It is worth noting that* $\{\!\{A, B, C\}\!\}$ *has two occurrences in sequence* ***1****. Then, Table 3.2 contains to lines with SID* ***1****.*

This raised two problems while using $Ripper_k$*:*

- ▷ *first, there is a mismatch between the growth rate of a rule evaluated by* $Ripper_k$ *(computed with the 6 examples of Table 3.1 corresponding to sequences* $\{1, 1, 2, 3, 5, 6\}$*) and the growth rate of a pattern (computed on the set of sequences* $\{1, 2, 3, 4, 5, 6\}$*): the growth rate of* $Ripper_k$ *does not take into account the fourth sequence but twice the first sequence.*
- ▷ *second, the extracted patterns may have larger constraints than necessary (and then be less accurate). Indeed,* $Ripper_k$ *extracted the discriminant temporal constraints* $\{(A,_1)[-\infty, 5](B,_2), (B,_2)[-\infty, 2](C,_3)\}$ *from Table 3.2, but* $\mathcal{T}_{mid} = \{(A,_1)[\mathbf{0}, 5](B,_2), (B,_2)[-\infty, 2](C,_3)\}$ *are sufficient to discriminate sequences (but not examples). Indeed, the chronicle* $(\{\!\{A, B, C\}\!\}, \mathcal{T}_{mid})$ *occurs in all positive sequences but in not any negative sequence.*

Y. Dauxais proposed several strategies to solve this difficult issue including the use of specific multiple instance machine learning algorithm (miSVM [Andrews et al., 2003] or MITI [Blockeel et al., 2005], a multiple instance decision tree learning algorithm). For sake of computational efficiency, the *DCM* algorithm post-processes the rules to prune those that are not discriminant according to a growth-rate computed with the number of different SID.

C.5.2 Generalizing the pattern-domain of discriminant chronicles

Chronicles have been presented as a couple: a multiset and temporal constraints. For discriminant patterns for a given multiset, the temporal constraints specifies the boundaries of the inter-event durations.

The graphical model constraint the boundaries to split the space using hyperplans orthogonal to the axes. This type of model is not necessarily the most accurate. A chronicles $(\{\!\{A, B\}\!\}, \{(A, 1)[1, 2](B, 2)\})$ specifies boundaries for the duration between event to be such that $1 \leq t_B - t_A \leq 2$, but learning a linear model $t_B = \beta + \alpha \times t_A$ with $\beta \in [0.5, 1.5]$ and $\alpha \in [0.5, 2]$ maybe more accurate. In any case, more complex models than quantitative rules could be learn from a relational dataset of temporal constraints (SVM, neural networks, etc). They could discover more complex temporal relationships between events. The drawback with such model is that we loose the interpretability of the model. Rule learning enables to visualize the chronicle while more complex models, and especially blackbox models, does not.

If the objective is to extract chronicles to make automatic labeling of sequences, visualizing chronicle is not a requirement and generalizing our approach maybe interesting. But, when the objective is to support an expert to get insights about the data at hand, interpretability is a real strength of chronicles.

By the way, it turns to be interesting to generalize discriminant chronicle mining by extracting patterns made of a couple $\langle \mathcal{M}, \mathcal{P} \rangle$ where $\mathcal{M}$ is a multiset of size n and $\mathcal{P}$ is a (learnable) classifier

having $n \times (n-1)$ numerical attributes as input. The Algorithm 5 for extraction patterns from sequence datasets remains identical and the principle of discriminant temporal constraint extraction has just to be adapted for the type of classifier. In such a pattern model, the temporal constraints are captured by the learnable models. This model is then used on the temporal projection of a multiset embedding to determine the sequence class.

The main interesting question is to know whether having a more complex temporal model improves sufficiently the accuracy of chronicles to accept loosing the interpretability. Indeed, using rule learning technique enable to graphically represent the temporal constraint, but it is no more possible with SVM classifier for instance. This tradeoff between interpretability and accuracy has been studied in Dauxais [2018] but more user experiences are now required to conclude. In any case, it opens a lot of possible alternative of encoding for the DCM algorithm.

C.5.3 On the semantic of discriminant chronicles: patterns at the boundaries

The previous section shows that using a machine learning algorithm yields a discriminant model. A discriminant model is a boundary between classes of sequences. Contrary to the clustering approaches that build representative, the supervised machine learning algorithm extract a boundary.

Then while extracting rules to yield chronicles temporal constraints, the values of the attributes are boundaries that discriminate sequences classes. As a consequence, the discriminant chronicles have to be carefully interpreted by users. The temporal constraint of a discriminant chronicles are not the representative of the sequence class, but at the boundaries between two behaviors.

For instance, the temporal constraint of the simple discriminant chronicle $(\{\!\{A, B\}\!\}, \{(A,_1)[-\infty, 5](B,_2)\}$ has to be interpreted as: "*if lower than* 5 *then class* $+$*, otherwise class* $-$".

$Ripper_k$ makes a choice of boundaries that will generalize the best the discrimination power of the classifier. But, the generalisation of examples is not necessarily the one that represents the best the examples. We shown that generalizing as a rule some examples with real valued attributed opens interpretability question of attribute-value rules [Besnard et al., 2019].

C.5.4 Decision making vs interpretability of discriminant chronicles

Finally, the practical use of discriminant chronicles for classifying new sequences appears to be practically complex. Indeed, mining discriminant chronicles leads to extract a lot of chronicles and each chronicle can be considered as a classification rule. Decisions from all chronicles on a new examples are not necessarily consistent and must be merged (e.g. by majority vote).

In pattern based classification [Bringmann et al., 2009], the principle is to use a classification algorithm to learn how to make a meta-classification. Nonetheless, Dauxais [2018] argues that with such solution, the semantic of a discriminant chronicle is changed as a chronicle is simply used as a feature to describe a sequence. Any chronicle may argue pro or cons a class, whatever the class the chronicle predict. We do not think that it is a good way to proceed and the problem is still open.

My opinion is that discriminant chronicles are interesting to get insights about data at hand but they have to be carefully manipulated for sequence classification.

D Negative temporal patterns[13]

In the previous two sections, we designed new pattern domains that enrich sequential patterns with temporal information. In this section, we enrich our sequential pattern with a new type of information: non-occurring events. It appears to be interesting for some specific application and we will see that the algorithmic principles (and difficulties) are shared with temporal pattern mining.

Analyzing non-occurring behaviors [Cao et al., 2015] is a recent data analysis task consisting in analyzing behaviors specifying absence of events. The classical data analytic tools have to be

[13] This section is a joint collaboration with René Quiniou, D. Lagarde (EDF), M. Boumghar (EDF) and L. Pierre (EDF).

adapted to take into account the specificities of such question: clustering, behaviors comparison and pattern mining.

Identifying non-occurring events is of particular importance when dealing with sequential data. In a medical context, a, b and c may be drug administration while d and e some medical events, respectively, patient declared cured and patient suffering complications. The situation that is illustrated by our synthetic dataset is the case of adverse drugs reaction. Being exposed to drug b while being treated by drugs a and c leads to complications. Mining positive patterns in a medical database would miss such adverse drug reaction. Then, mining frequent negative sequential patterns is of utmost interest to discover actionable rules taking into account absent events.

In this section, we first introduce the mining of negative sequential patterns, then we present the work developed in collaboration with EDF[14] on the mining of temporal negative patterns. Again, our contribution is threefold:

- ▷ a deep study of the semantics of negative patterns and their properties[Guyet and Quiniou, 2020] which leads us to run counter the state of the art by showing that some semantics enjoy anti-monotonicity properties,
- ▷ NegPSpan [Guyet and Quiniou, 2020]: an algorithm based on the depth first search strategy of PrefixSpan and that enables to extract complete and correct set of negative sequential patterns,
- ▷ NTGSP [Tsesmeli et al., 2018]: an algorithm to extract negative quantitative temporal patterns.

In this section, we focus on the third contribution to pursue the presentation of a general framework of (negative) sequential patterns with metric temporal information.

D.1 Negative sequential patterns

This section presents Negative Sequential Patterns (NSP) extraction. NSP refers to sequential patterns that specifies non-occurring, or absent, items in a sequence of events [Wang and Cao, 2019]. Given the example of Kamepalli et al. [2014], suppose we have a positive sequential pattern $\boldsymbol{p_1} = \langle a\ b\ c\ d\ f\rangle$ and a negative one $\boldsymbol{p_2} = \langle a\ b\ \neg c\ e\ f\rangle$. Pattern $\boldsymbol{p_1}$ means that events a, b, c, d and f usually happen in a row. But pattern $\boldsymbol{p_2}$ further specifies that one usual behavior is that events a and b happen in a row, then events e and f occur in a row, but event c does not occur between b and e.

D.1.1 Definitions and semantics

In this section, items are so-called literals and we consider sequences of itemsets. The symbol $\neg$ is used before an itemset to specify that it is a negative itemset. The definition are partly borrowed from Guyet and Quiniou [2020].

Definition 28 (Negative sequential patterns)**.** *A negative pattern is a sequence of positive and negative itemsets:* $\langle p_1\ \dots\ p_n\rangle$ *where* $\forall i$, p_i *is an itemset of literals* ($p_i = \{p_i^j\}$, $p_i^j \in \mathcal{I}$) *or a negated itemset of literals* ($p_i = \neg\{q_i^j\}$, $q_i^j \in \mathcal{I}$).

The subsequence of the pattern that holds the positive itemsets of a pattern $\boldsymbol{p}$ *is called the* positive part *of* $\boldsymbol{p}$*, denoted* $\boldsymbol{p}^+$.

According to this definition, patterns are made of negative or positive itemsets but an itemset is not made of both positive and negative items.

Let's now define the notion of occurrence of a negative sequential pattern. This definition is central to define the notion of support and it clarifies the semantic of the negative itemsets.

Definition 29 (Negative pattern occurrence)**.** *Let* $\boldsymbol{s} = \langle s_1\ \dots\ s_n\rangle$ *be a sequence and* $\boldsymbol{p} = \langle p_1\ \dots\ p_m\rangle$ *be a negative sequential pattern, pattern* $\boldsymbol{p}$ *occurs in sequence* $\boldsymbol{s}$ *iff* $n \geq m$ *and* ***there exists*** *an embedding* $(e_i)_{i\in[m]}$*, such that* $\forall i \in [m]$, $e_i \in [n] \cup \epsilon$*:*

$$\begin{cases} p_i \subseteq s_{e_i} & \text{if } p_i \text{ is positive} \\ e_i = \epsilon \wedge \forall j \in [e_{i-1}+1, e_{i+1}-1],\ p_i \not\subseteq s_j & \text{if } p_i \text{ is negative} \end{cases}$$

[14] EDF is the largest french electricity supplier company.

This definition of pattern occurrence illustrates that a negative pattern would be seen as a couple $\langle \boldsymbol{p}, \mathcal{C} \rangle$ where $\boldsymbol{p}$ is a sequential patterns (without negated itemsets) and $\mathcal{C}$ is a set of constraints. Contrary to the temporal patterns, the constraints specify absence of a items between the occurrence of two positive itemsets. This view of a negative pattern makes it very similar to temporal patterns. In accordance of previous algorithms, the positive part can be seen as the **symbolic signature** of the negative pattern and the notion of **projection** will be later defined on this basis.

Definition 29 is the "existing absence" semantic for negative sequential patterns. There also exists another semantic which replaces the existence of an embedding by a "for all embeddings". We denote this semantics as an "absolute absence". Their differences are the following:

- ▷ *absolute absence*: a negative pattern $\boldsymbol{p}$ occurs in a sequence $\boldsymbol{s}$ iff for all occurrences of the positive part of the negative pattern in the sequence, the negation constraints is satisfied
- ▷ *existing absence*: a negative pattern $\boldsymbol{p}$ occurs in a sequence $\boldsymbol{s}$ iff there exists at least one occurrence of the positive part of the negative pattern in the sequence for which the negation constraints is satisfied

The difference between the two semantics is related to the multiple occurrence issues. Since there is only one unique occurrence of the positive part of the pattern, occurrence decisions are identical for the two definitions. When there is multiple embeddings of the positive part in a sequence, occurrence decision may change. This is illustrated by the following example.

Example 21 (Negative pattern occurrence)**.** *Let's give an example and consider the pattern* $\boldsymbol{p} = \langle a\ b\ \neg c\ d \rangle$ *and two sequences* $\boldsymbol{s_1} = \langle a\ b\ e\ d \rangle$ *and* $\boldsymbol{s_2} = \langle a\ b\ c\ a\ d\ e\ b\ d \rangle$*. The positive part of* $\boldsymbol{p}$ *is* $\langle a\ b\ d \rangle$*. It occurs once in* $\boldsymbol{s_1}$ *so there is no difference between the two semantics. But, in the second case, the positive part occurs twice with embeddings* $(1, 2, 5)$ *and* $(4, 7, 8)$*. The first occurrence does not satisfy the negative constraint* $(\neg c)$ *while the second occurrence does. According to the "existing absence" semantics, the sequence* $\boldsymbol{s_2}$ *matches the pattern while it does not with the "absolute absence" semantic.*

The "*absolute absence*" is more restrictive, but depending on the problem and the data at hand, the first could be useful. The two semantics could be found in the literature (see D.1.2). The classical definition of pattern support, frequent patterns and pattern mining task comes naturally with the notion of occurrence of a negative sequential pattern, no matter the choice.

We refer the reader to [Guyet, 2019] for an extended formalization of different semantics for negative sequential patterns and their properties.

D.1.2 State of the art of negative sequential patterns

Wang and Cao [2019] recently survey the mining of negative patterns. This section presents briefly three significant algorithms for NSP mining (PNSP, Neg-GSP and e-NSP), and our own algorithm NegPSpan.

PNSP algorithm and Neg-GSP algorithm PNSP (Positive and Negative Sequential Patterns mining) [Hsueh et al., 2008] is the first algorithm proposed for mining both positive and negative sequential patterns. PNSP extends GSP to deal with the mining of negative sequential patterns. It consists of three phases: It first mines frequent positive sequential patterns, by using GSP, then it preselects negative sequential itemsets. A negative itemset must be infrequent but with a support greater than another threshold ($miss\_freq$). In the third phase, candidate negative sequences are level-wised generated. The sequence dataset is re-scanned to compute the support of these candidates, which are pruned when infrequent. Zheng et al. [2009] also proposed a negative version of GSP to find negative sequential patterns with similar principles.

These algorithms do not extract the complete set of negative sequential patterns due to their pruning strategy that is not correct [Zheng et al., 2009]. Thus, they potentially miss frequent negative patterns.

e-NSP algorithm e-NSP (efficient NSP) [Cao et al., 2016] identifies NSP by computing only frequent positive sequential patterns without re-scanning the database. This approach is far more

efficient than previous approaches that need for several re-scans of the database, in order to calculate the support of negative candidates (NSC). Nonetheless, it is important to notice that it is neither complete and than they do not extract the same set of patterns. Their definition of occurrence is based on an "absolute occurrence" semantic while the other existing approaches was based on the "existing occurrence" semantic.

eNSP first extract frequent sequential patterns and then evaluates the frequency of some negative pattern by arithmetic operation on the support of sub-patterns. The most simple case is a negative pattern with a single negative itemset, e.g. $\boldsymbol{p} = \langle a\ b\ \neg c\ d\rangle$. The support of $\boldsymbol{p}$ can be computed as the difference between the support of $\langle a\ b\ d\rangle$ and the support of $\langle a\ b\ c\ d\rangle$. The second pattern is called the positive partner of $\boldsymbol{p}$. The first is called the positive part of the negative pattern. The article shows that the support of a negative pattern can always be deduced from the support of frequent positive patterns. This simple reasoning on occurrence sets prevents from scanning sequences to compute negative patterns support.

This make the algorithm efficient considering that it only require to compute frequent positive sequential patterns but the requirements are restrictive: the positive partner has to be frequent, and occurrence of a negative pattern follows an "absolute absence" semantic. From our point of view, eNSP misses potentially interesting pattern in the database.

NegPSpan **algorithm** NegPSpan [Guyet and Quiniou, 2020] is a complete and correct algorithm to extract negative sequential pattern under the "exists occurrence" semantic. This algorithm is inspired by the depth first search strategy of PrefixSpan. The completeness is ensured thanks to the anti-monotonicity of the support evaluated by the "exists occurrence" semantic. The main advantages of this algorithm is first its low memory consumption compare to eNSP. eNSP stores a large set of frequent sequential patterns to generate NSP. The depth first search strategy prevents from storing these patterns. In addition, the second advantage of NegPSpan is its theoretical foundations that enables to take advantage of decades of research in pattern mining to improve the algorithm. In practice, we combine negative sequential patterns with gaps constraints (see Section A.3.2). This constraint is very meaningful in sequential pattern mining and it significantly improves algorithms' efficiency. In practice, we shown that NegPSpan is as time-efficient as eNSP while using weak[15] gap constraints.

D.2 Introducing time in negative sequential patterns

In this section, we introduce NTGSP – Negative Time Gap Sequential Pattern. This algorithm extracts *negative temporal patterns* from sequences of timestamped itemsets [Tsesmeli et al., 2018]. NTGSP is based on TGSP [Yen and Lee, 2013] to extract metric temporal inter-event duration constraints. It uses a density-based clustering algorithm (namely CLIQUE [Agrawal et al., 2005]) to discover typical inter-event durations represented by intervals of inter-event duration. This solution is in between *DCM* (it extracts inter-event duration intervals) and QTIPrefixSpan (it uses a clustering algorithm to extract temporal constraints).

D.2.1 Negative temporal pattern mining

Lets consider a dataset of temporal sequences $\mathcal{D}$ where temporal sequences are made of timestamps events, i.e. the temporal domain of events is $\mathbb{R}$.

Definition 30 (Negative Temporal Pattern (NTP))**.** *A negative temporal pattern* $\langle p_1, \tau_1, p_2, \tau_2, \ldots, \tau_{n-1}, p_n\rangle$ *interleaves itemsets of literals,* p_i*, and negative temporal constraints,* $\tau_i = \{[u_i, l_i], q_i\}$ *where:*

- ▷ $[u_i, l_i]$ *is a temporal constraint specifying the bound of admissible duration between* p_i *and* p_{i+1}*.*
- ▷ q_i *is a negative itemset listing all items that must not occur between occurrences of* p_i *and* p_{i+1}*.*

[15] A weak gap constraint means for instance that we do not take *maxgap* to 2 or 3 items, but larger values that make sense with respect to the mean length of the sequences.

Definition 31 (Occurrence of a negative temporal pattern in a sequence)**.** *A negative temporal pattern* $\boldsymbol{p} = \langle p_1, \tau_1, p_2, \tau_2, \ldots, \tau_{n-1}, p_n \rangle$ *occurs in a time sequence* $\boldsymbol{s} = \langle (s_i, t_i) \rangle$ *iff there exists at least one embedding* $(e_i)_{i \in [n]}$ *such that:*

- ▷ $\forall i \in [n],\ p_i \subseteq s_{e_i}$ *(positive itemsets matches),*
- ▷ $\forall i \in [n-1],\ u_{e_i} \leq t_{e_{i+1}} - t_{e_i} \leq l_{e_i}$ *(temporal constraints),*
- ▷ $\forall i \in [n-1],\ \forall t \in]t_{e_i}, t_{e_{i+1}}[,\ q_i \nsubseteq s_t$ *(negations)*

In addition, we can define some notations in accordance with previous sections. Let $\boldsymbol{p} = \langle p_1, \tau_1, p_2, \tau_2, \ldots, \tau_{n-1}, p_n \rangle$ be a negative temporal pattern:

- ▷ $\langle p_1, p_2, \ldots, p_n \rangle$ is the *symbolic signature* of the patterns. All itemsets of a symbolic signature are positive itemsets.
- ▷ let $\boldsymbol{s} = \langle (s_i, t_i) \rangle$ be a temporal sequence. The *temporal projection* of $\boldsymbol{s}$ on $\boldsymbol{p}$ is the vector of dimension $n-1$: $[(t_{e_2} - t_{e_1}), (t_{e_3} - t_{e_2}), \ldots, (t_{e_n} - t_{e_{n-1}})]$ where $(e_i)_{i \in [n]}$ is an *embedding* of a negative temporal pattern. According to the definition of TGSP, the set of temporal projections is a called a *time-gap table.*

Example 22 (Negative temporal pattern)**.** *For example, the pattern* $\boldsymbol{p} = \langle A, \{[1,3], \neg C\}, E, \{[2,2], \emptyset\}, D \rangle$ *is a NTP. It indicates that there is at least 1 and at most 3 time units between the occurrences of items A and E and also that item C does not appear between occurrences of items A and E. Then, item D must occur exactly 2 time units after E. The symbolic signature of* $\boldsymbol{p}$ *is* $\langle A\ E\ D \rangle$*.*

Without temporal constraints, the negative sequential pattern would be $\langle A\ \neg C\ E\ D \rangle$*.*

Let $\boldsymbol{s} = \langle (D,37)\ (A,38)\ (E,41)\ (C,42)\ (D,43) \rangle$ *be a time sequence. The sequence* $\boldsymbol{s}$ *supports* $\boldsymbol{p}$ *(or* $\boldsymbol{p}$ *occurs in* $\boldsymbol{s}$*) with the embedding* $(2,3,5)$*. Positive itemsets of the patterns occurs at the given positions without occurrence of C between A and E. We also find the event E three time units after the event A, a fact that satisfies the temporal constraint of at least one time unit and at most three time units between the events A and E; and exactly two time units between D and E.*

The temporal projection of $\boldsymbol{s}$ *on the pattern* $\boldsymbol{p}$ *is the vector of dimension 2:* $\{3,2\}$*.*

Then, classically, mining frequent negative temporal patterns consists in extracting all NTP $\boldsymbol{p}$ such that $\boldsymbol{p}$ occurs in more sequence of a database $\mathcal{D}$ than a given minimal frequent threshold f_{min}.

Remark 8. *This model of temporal patterns – without negations – is strongly related to chronicles. It corresponds to chronicles with a shape of a chain of items linked by positive temporal constraints (sequentiality). For example,* $\langle A[1,3]B[2,2]C \rangle$ *could be written as a chronicle* $\langle \{\!\{A, B, C\}\!\}, \{(A,1)[1,3](B,2), (B,2)[2,2](C,3)\} \rangle$*.*

D.2.2 NTSGP algorithm

The general structure of NTGSP combines ideas from the e-NSP algorithm for the extraction of sequential patterns with negation and from TGSP for the extraction of time intervals between itemsets of a sequential pattern step-wised.

Let $\mathcal{S} = \{\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n\}$ be a data set of n time sequences, σ the minimum support threshold of positive sequential patterns, μ the minimum support threshold of negative sequential patterns, ϵ the length of temporal units (for the CLIQUE algorithm) and δ the minimum density threshold of temporal units (for the CLIQUE algorithm).

The main steps of NTGSP are as follow:

1. **Frequent patterns extraction** The first step of the algorithm aims at extracting all the sequential patterns from $\mathcal{S}$. For this task, we simply use the PrefixSpan algorithm [Pei et al., 2004].
2. **Negative Sequential Candidates (NSC) generation** For any sequential pattern $\boldsymbol{s}$ extracted in step 1, all possible negative sequential candidates (NSC) are generated by turning one or more positive itemsets of $\boldsymbol{s}$ to negative.
3. **Frequent negative sequence patterns extraction** This step prunes all negative sequential patterns that have a support lower than μ.

4. **Negative temporal patterns extraction** Having extracted all frequent negative sequence patterns, the fourth and last part of the algorithm extracts the temporal dimension of patterns. This step searches for frequent time intervals between the positive itemsets of the negative sequence patterns. Similarly to DCM, temporal projection are computed from positive partners of each frequent negative sequence pattern. Temporal projections are then clustered using CLIQUE [Agrawal et al., 2005]. CLIQUE is a density based clustering algorithm. It decomposes the $(n-1)$-dimensional space in rectangle of length ϵ and prunes those that are not dense (according to the δ threshold). Finally, the clusters are decomposed in maximum $(n-1)$-dimensional rectangles to generate temporal constraints. Each rectangle yields a new temporal pattern.

D.3 Experiments

In this section we use synthetic dataset to study the behavior of NTGSP.[16] The identification of embeddings of a negative sequential pattern is done by regular expression matching algorithm. In fact, regular expressions enables to specify negation and their matching algorithm benefit from years of research and are computationally very efficient.

D.3.1 Synthetic data generation

We developed a random sequence generator which hides negative temporal patterns in the generated database. This generator follow the same principle presented in Section B.3.1.

On the one side, our generator enables to generate dataset with controlled characteristics: dataset size (default value: $1,000$)[17], mean length of sequences (30), vocabulary size (20). On the other side, it allows to compare the NTGSP results with expected patterns[18]. The frequency of the hidden patterns is controlled by two parameters: 1) a minimum occurrence frequency of the negative sequential pattern (0.1) and 2) a minimum threshold for the positive partner of hidden negative patterns. Following the e-NSP principle a negative pattern that occur frequently but whose positive partner does not occur will not be extracted. By default, the ratio of the frequencies between negative and positive patterns has to be between 0.2 and 0.5. Finally, the range of the duration between event ($[0, 100]$) set up randomly generated temporal patterns.

With this information, we can use the sequence database as input for our NTGSP algorithm, run the algorithm and finally verify if our algorithm extracts the planted NSP.

D.3.2 Experiments

Synthetic datasets are used to study the NTGSP behavior wrt its different parameters: σ, μ and ε. The impact of parameter δ is weak on results, so we decided to not report experiments on this parameter. In the following, each experiment has been repeated on 20 different datasets generated with the same characteristics. Figures below display the mean and standard deviation of measurements through value distributions. The experiments focus on the behavior of three metrics: the numbers of *sequential patterns*, *negative sequential patterns* and *frequent negative temporal patterns*.

Figure 3.14 displays the three metrics variation wrt the minimum support threshold μ and parameter $\sigma = 0.08 * \mu$, while parameters $\varepsilon = 40$ and $\delta = 0.005$ remain constant. Parameter ε was set to 40 because small values yielded few negative temporal patterns that led to unvaluable comparisons.

These experiments show that NTGSP extracts negative temporal patterns for very low minimum support thresholds, while the number of negative sequential patterns that do not include temporal information between events is significantly greater. This comes from a feature of the

[16] NTGSP has been implemented in Python and C++. PrefixSpan is implemented in C++, the main procedure and the CLIQUE algorithm are implemented in Python.
Some more information about implementation of negative sequential patterns can be found here: http://people.irisa.fr/Thomas.Guyet/negativepatterns/.

[17] values in parenthesis give the default values of the parameter.

[18] The generator ensures that the hidden patterns are frequent, but due to the random insertion of patterns, it may happens that some other (negative) patterns could by frequent in the database.

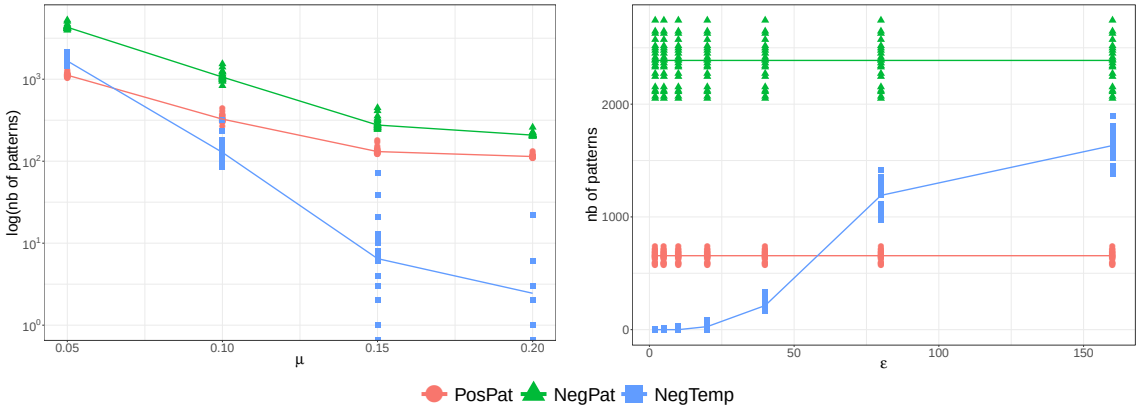


Figure 3.14: Number of positive, negative and temporal negative patterns wrt minimum negative threshold (μ) on the left, and wrt temporal unit length (ε) on the right.

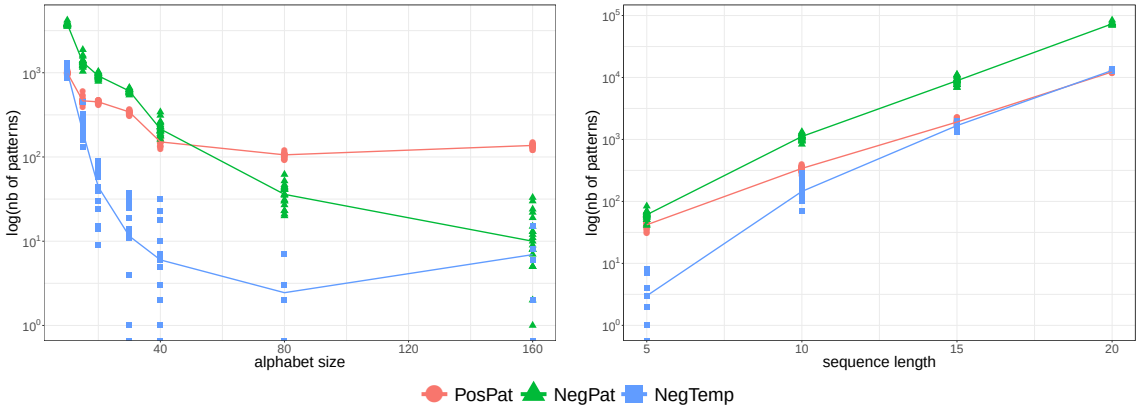


Figure 3.15: Number of positive, negative and temporal negative patterns wrt alphabet size (ql) on the left, and wrt mean sequence length (l) on the right.

sequence database synthesis that does not generate negative temporal patterns with high support. Furthermore, Figure 3.14-left shows that the number of negative sequence patterns is closely related to the number of sequential patterns extracted by PrefixSpan. This is due to the basic idea of negative pattern extraction borrowed from e-NSP [Cao et al., 2016], which generates negative candidates based on frequent positive patterns. Finally, from Figure 3.14-right we observe that higher ε values yield more negative temporal patterns, as it defines larger temporal units in the data points (patterns) space. Thus, temporal units include more data points and so the final clusters will also include more negative temporal patterns.

Figure 3.15-left displays how varies the number of patterns wrt the alphabet size. It shows that the number of patterns increases when the alphabet size decreases: for alphabets of small size, sequences are built on a more restricted set of items and, so, the generated sequences have a higher probability to contain common subsequences. Similarly, Figure 3.15-right shows that the number of patterns increases exponentially with the mean sequence length: longer sequences contain more items and, so, have a higher probability to contain common sub-sequences.

Figure 3.16 displays the runtime performance of NTGSP. The runtime increases exponentially when the minimum support decreases. This was expected since the runtime is closely correlated to the pattern number and varies accordingly with respect to both the minimum support (Figure 3.16-left) and the mean sequence length (Figure 3.16-right). The execution time for mining negative pattern and the execution time of CLIQUE are quite similar. The execution time of PrefixSpan takes a negligible part of the global time. This comes partly from an efficient implementation of PrefixSpan in C++ whereas the other modules are implemented in Python.

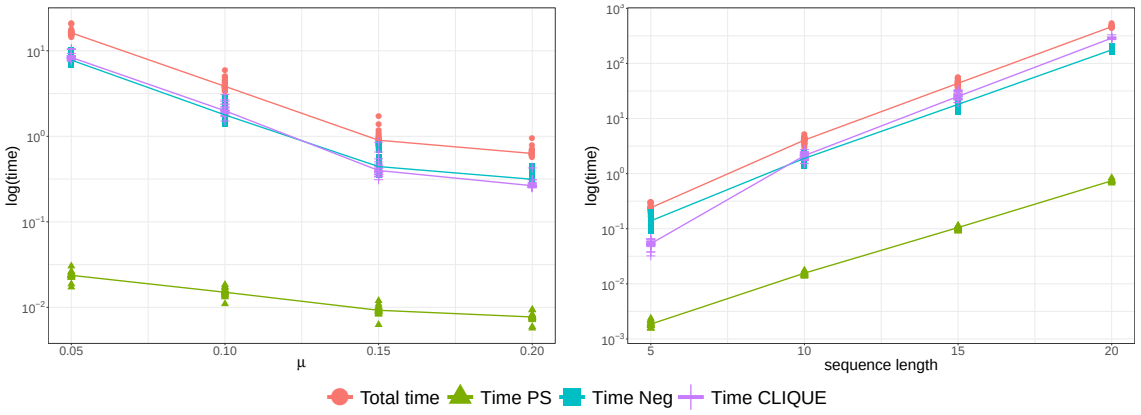


Figure 3.16: Execution times wrt minimum support (μ) on the left, and wrt mean sequence length (l) on the right. *Time PS* (PrefixSpan), *Time Neg* (Negative pattern generation) and *Time CLIQUE* are computing times of each algorithm part. *Total time* is the total time.

D.4 Discussion

In this section, we introduced the notion of negative sequential patterns and extend this pattern domain with temporal information.

From the application needs to the experimental studies through formal developments, this line of work leads us to touch the essence of pattern mining. The initial challenge was to formalize patterns that capture some specific information (absent events). Starting from the first patterns shapes, we had to formalize the pattern domain and explore the properties of several alternatives.

Beyond the formal algorithm properties of the patterns, it invites us to though about the semantic of the extracted pattern (what "absent event" means for users?). This is definitively a central question while dealing with pattern mining algorithms. Indeed, as stated in the introduction, beyond the predictive power of patterns, we are interested in conveying information about hidden behaviors in the sequences to a user. This requires to take care of the (mathematical) meaning of outputted patterns and how it will be understood by users.

With NTGSP, we proposed a new original pattern domain that combines quantitative temporal information and negation. Nonetheless, the proposed has two weaknesses:

- ▷ First, the efficiency of the algorithm may be improved with a better memory management. In fact, the three main tasks are executed sequentially in our algorithm. This requires to keep all patterns in memory. This approach is surely not an optimal memory management. More precisely, even if we have not encountered practical difficulties in our experiments, time-gap tables may require a lot of memory for dense datasets. Using a depth-first search strategy would improve significantly our algorithm. In this strategy, instead of keeping all time-gap tables in memory, it is sufficient to keep the unique time-gap table corresponding to the current negative pattern. This optimization could be achieved by embedding negative generation and CLIQUE algorithm within the PrefixSpan process. The algorithmic complexity would remain identical.
- ▷ Second, the completeness of the algorithm could be improved. Our negative candidate generation is close to the e-NSP strategy that derives negative patterns from positive patterns. As mentioned in Section D.1, this leads to miss frequent negative patterns. Using a negative pattern generation that does not impose constraints on the positive partner would be more costly but more complete.

To address this issues, we are currently working on the adaptation of NegPSpan to generate a more complete set of frequent negative temporal patterns. Then, the overall algorithms would be very close to QTIPrefixSpan: for each pattern, the CLIQUE algorithm (or another clustering algorithm) is applied on the temporal projections to extract representative temporal constraints corresponding to the current NSP.

E Case study

This section applies our algorithms (*DCM*, QTempIntMiner and NTGSP) to the GENEPI study, i.e. the analysis of care pathways of epileptic patients. The rational of this study can be found in Section B of Chapter 2. We simply remind that we investigate the association between medication substitutions (so called switches) with epileptic seizures for patients with long term treatment with anti-epileptic (AE) medication.

In Section B.3 of Chapter 2, we presented results applying sequential pattern mining to a dataset of sequences originates from the SNIIRAM database. The same data are used to experiment our algorithms, we simply take into account the temporal dimension of the data by adding the dates of drugs deliveries or medical procedures. The temporal granularity of the data is *day*. For each algorithm, we present data preprocessings and some patterns extracted to bring some insights to epidemiologists.

Our objective is not to detail the results but to give a flavor of how temporal patterns may be useful to analyze temporal data. More especially, we do not deeply comment the medical interpretation of the patterns. In our idea (see Chapter 5), pattern mining supports epidemiologists in their data interpretation. The extracted pattern identifies situation that epidemiologists would be interesting in investigating.

E.1 *DCM* results

E.1.1 Dataset preparation

In this experiment, the data set has to be preprocessed to generate positive and negative examples where each example is a sequence of timestamped events. Since all patient sequences have at least one seizure event, we adapt the case-crossover protocol to apply our *DCM* algorithm. This protocol, consisting in using a patient as his/her own control, is often used in PE studies. The dataset is made of two sets of $8,379$ labeled sequences. A 3-days induction period is defined before the first seizure of each patient. Drugs delivered within the 90 days before inductions yield the positive sequences and those delivered within the 90 days before the positive sequence, i.e. the 90 to 180 days before induction, yield the negative sequences. We discarded all other events (medical procedures, medical visits, etc). The temporal granularity of the data is the day.

E.1.2 *DCM* results

Set up with $\sigma_{min} = 5.10^{-3}$, i.e. 42 patients[19], and $g_{min} = 1.4$, we generated 777 discriminant chronicles. Chronicles involved 510 different multisets and 128 different event types.

Three types of pattern are of specific interest for clinicians: (1) sequences of AE generic and brand-named drug deliveries, (2) sequences of same AE drug deliveries, (3) sequences with AE drug deliveries and other drug types deliveries. According to these criteria, we selected 55 discriminant chronicles involving 16 different multisets to be discussed with clinicians. For the sake of conciseness, we choose to focus the remaining of this section on chronicles related to *valproic acid* (*N03AG01* ATC code, with different presentations) but our results contain chronicles related to other AE drugs like *levetiracetam* or *lamotrigin*.

We start with patterns representing switches between different presentation of *N03AG01*. In the following, drug id 114 stands for brand name *valproic acid* while 383 stands for generic *valproic acid*. Fig. 3.17 illustrates all discriminant patterns that have been extracted. It is noteworthy that all chronicles have temporal constraints, this means that multisets without temporal constraints are not discriminant. This results is consistent with Polard et al. [2015] which concluded that brand-to-generic AE drug substitution was not associated with an elevated risk of seizure-related hospitalization. But temporal constraints was not taken into account in the later. The four extracted chronicles suggest that for some small patient groups, drug switches with specific temporal constraints are more likely associated to seizure.

[19] This number of patients have been initially estimated important by epidemiologists to define a population of patients with similar care sequences associated to seizure.

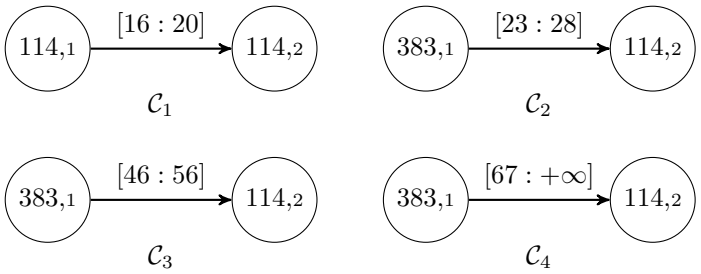


Figure 3.17: Four discriminant chronicles describing switches between same type of *valproic acid* (N03AG01) generic (383) and brand-named (114). $supp(\mathcal{C}_i, \mathcal{S}^+)$ respectively for $i = 1$ to 4 equals 43, 78, 71 and 43 and $supp(\mathcal{C}_i, \mathcal{S}^-)$ equals 23, 53, 39 and 30. Please remind that the second figure in a chronicle node represents simply the position of its event in the multiset.

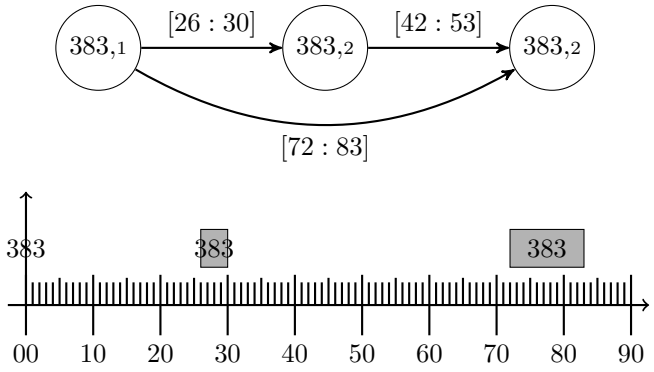


Figure 3.18: Above, a chronicle describing repetitions of *valproic acid* (N03AG01) generic (383) and, below, its timeline representation. The chronicle is more likely related to epileptic seizure: $supp(\mathcal{C}, \mathcal{S}^+) = 50$, $supp(\mathcal{C}, \mathcal{S}^-) = 17$.

The two first chronicles represent delivery intervals lower than 30 days, respectively from brand-to-generic and generic-to-brand. The third one represents an interval between the two events greater than 30 days but lower than 60 days. The temporal constraint of the last one could be interpreted as $[67, 90]$ because of the bounded duration of the study period (90 days). This chronicle represents a switch occurring more than 60 days but most of the time less than 90 days. The temporal depth of collected data ensures to prevent from missing data in the studied information.

These behaviors may correspond to unstable treatments. In fact, AE drug deliveries have to be renew every month thus, a regular treatment corresponds to a delay of ≈ 30 days between two AE drug deliveries.

We next present in Fig. 3.18 an example of discriminant chronicle that involves three deliveries of *N03AG01* (no chronicle involves more deliveries of this AE drug).

The growth rate of this chronicle is high (2.94). It is moreover simple to understand and, with their DTC, it can be represented on a timeline (see Fig. 3.18, below). It is noteworthy that the timeline representation loses some constraint information. The first delivery is used as starting point (t_0), but it clearly illustrates that last delivery occurs too late after the second one (more 30 days after). As well as previous patterns, this chronicle describes an irregularity in deliveries. More precisely, the irregularity occurs between the second and the third deliveries as described by the temporal constraints $[42, 53]$ and $[72, 83]$.

We conclude from observations about the previous two types of patterns that the precise numerical temporal information discovered by *DCM* is useful to identify discriminant behaviors. Analyzing pure sequential patterns does not provide enough expression power to associate switch of same AE deliveries with seizure. Chronicles, specifying temporal constraints, allow us to describe

Table 3.4: Patterns involving *valproic acid* switches with their supports computed by NegPSpan.

Pattern	support NegPSpan
$\boldsymbol{p_1}^{NSP} = \langle 7\ \neg 86\ 7 \rangle$	3125
$\boldsymbol{p_2}^{NSP} = \langle 7\ \neg 114\ 7 \rangle$	3191
$\boldsymbol{p_3}^{NSP} = \langle 7\ \neg 383\ 7 \rangle$	3188
$\boldsymbol{p_4}^{NSP} = \langle 7\ 7 \rangle$	3195
$\boldsymbol{p_5}^{NSP} = \langle 114\ \neg 7\ 114 \rangle$	2963
$\boldsymbol{p_6}^{NSP} = \langle 114\ \neg 86\ 114 \rangle$	2848
$\boldsymbol{p_7}^{NSP} = \langle 114\ \neg 383\ 114 \rangle$	2933
$\boldsymbol{p_8}^{NSP} = \langle 114\ 114 \rangle$	2969
$\boldsymbol{p_9}^{NSP} = \langle 383\ \neg 7\ 383 \rangle$	3428
$\boldsymbol{p_{10}}^{NSP} = \langle 383\ \neg 86\ 383 \rangle$	3248
$\boldsymbol{p_{11}}^{NSP} = \langle 383\ \neg 114\ 383 \rangle$	3378
$\boldsymbol{p_{12}}^{NSP} = \langle 383\ 383 \rangle$	3442

conditions under which a switch of same AE deliveries is discriminant for epileptic seizure.

E.2 Negative temporal patterns

This experiment required less preprocessing. Indeed, we simply extract negative temporal patterns from the set of positive temporal sequences generated for the *DCM* experiment.

We start by extracting temporal negative sequential patterns as it does not need data preprocessing. Thus, the input is the set of 18,499 timestamp sequences of drugs deliveries preceding epileptic seizure.

E.2.1 Negative sequential pattern extraction

Let us start with some negative sequential patterns extracted by NegPSpan (without temporal constraints). Initially, we set up NegPSpan with $\sigma = 15\%$ (2,774 sequences), a maximum pattern length of $l = 3$, $\tau = 3$. NegPSpan extracts only 12 patterns (including positive and negative patterns). In this analysis, we pay attention to the specialty of *valproic acid* which exists in generic form (event 383) or brand-named form (event 114). Table 3.4 presents patterns starting and finishing with event 383. 7 is *levetiracetam* an alternative anti-epileptic drugs, but 86 is *paracetamol*.

Then, we changed the settings of NegPSpan to focus on patterns involving a switch from generic form to brand-named form of *valproic acid*. The settings were $\sigma = 1.2\%$, $l = 3$ and max gap constraint $\tau = 5$. The only two frequent positive patterns are $\langle 114\ 383\ 114 \rangle$ and $\langle 114\ 114 \rangle$. These patterns are not conclusive about the impact of a switch from 114 to 383 on epileptic seizures. But NegPSpan extracts a pattern $\langle 114\ \neg 383\ 114 \rangle$, which specifies that the absence of a switch is frequent for patients having epileptic seizure. Thanks to gap constraints, this pattern describes interesting behaviors that shed light on the fact that switches and non-switches from generic form to brand-named form of *valproic acid* are both frequent behaviors in care pathways of patients having epileptic seizure. This result is in accordance with Polard et al. [2015]. This particular case illustrates that NSP are essential to have true insights about the meaning of frequent positive patterns.

E.2.2 Negative temporal pattern extraction

Then, we applied the temporal extension of NegPSpan to extract negative temporal patterns.[20] The set up for negative temporal patterns $\sigma = 2.70\%$ (500 sequences), $l = 3$, $\tau = 3$. The minimal

[20] In this section, we use a more recent implementation of negative temporal patterns. Its principle is similar to NT-GSP but uses NegPSpan to extract negative sequential patterns and MAFIA [Nagesh et al., 2001], an improvement of CLIQUE clustering [Agrawal et al., 2005].

Table 3.5: (Negative) temporal patterns involving *valproic acid* switches with their supports.

Temporal pattern	Support
$\boldsymbol{p_1}^{NTP} = \langle 114 \xrightarrow[\neg 383]{[27,31]} 114 \rangle$	896
$\boldsymbol{p_2}^{NTP} = \langle 114 \xrightarrow{[27,31]} 114 \rangle$	899
$\boldsymbol{p_3}^{NTP} = \langle 383 \xrightarrow[\neg 114]{[24,28]} 383 \rangle$	836
$\boldsymbol{p_4}^{NTP} = \langle 383 \xrightarrow[\neg 114]{[23,32]} 383 \rangle$	778
$\boldsymbol{p_5}^{NTP} = \langle 383 \xrightarrow{[22,27]} 383 \rangle$	597
$\boldsymbol{p_6}^{NTP} = \langle 383 \xrightarrow{[28,32]} 383 \rangle$	1164

frequency threshold has been diminished to take into account that occurrences are clustered that contains necessarily less elements.

Again, we focus our attention on *valproic acid* switches (events 383, brand-name presentation and 114, generic presentation). Table 3.5 presents some extracted patterns involving both events 114 and 383. The support may be compared to those of Table 3.4.

Pattern $\boldsymbol{p_1}^{NTP}$ illustrates that a single temporal constraint $[27, 31]$ is extracted from the negative sequential pattern $\langle 114\ \neg 383\ 114 \rangle$. This temporal constraint is not really interesting has it simply retrieves the facts that drugs are delivered about every 30 days in France. The reversed switch ($\langle 383\ \neg 114\ 383 \rangle$) is split in two temporal patterns $\boldsymbol{p_3}^{NTP}$, with temporal constraint $[22, 27]$, and $\boldsymbol{p_4}^{NTP}$, with temporal constraint $[28, 32]$. We do not explain $\boldsymbol{p_3}^{NTP}$, but $\boldsymbol{p_4}^{NTP}$ is again the French constraint of monthly delivery.

The support of negative temporal patterns are interesting to compare with the support of their temporal positive partners. We can notice that $\boldsymbol{p_1}^{NTP}$ and $\boldsymbol{p_2}^{NTP}$ have almost the same temporal constraints and support, meaning that there are only 3 occurrences of $\langle 114\ 114 \rangle$ with a 383 switch preceding epileptic seizures. On the opposite, there are 386 occurrences of $\boldsymbol{p_6}^{NTP}$ that are not occurrences of $\boldsymbol{p_4}^{NTP}$, meaning that in a third of occurrence of $\boldsymbol{p_6}^{NTP}$, there are a switch with 114 before an epileptic seizure. Thus, it seems interesting for epidemiologists to further investigate these cases.

E.3 QTIPrefixSpan results

In this section, we present some results obtained with QTIPrefixSpan. QTIPrefixSpan processes a sequence of interval-based events. In the data at hand, we have punctual events corresponding to drug deliveries. Then, we apply a pre-processing to transform drug deliveries in drug exposure. This preprocessing is detailed in next Section, next we present the results.

E.3.1 Dataset preparation

The transformation of drug deliveries in drug exposure in a three steps procedure:

1. each drug delivery event (e, t) is transformed into an interval based event $(e, [t, t + 30])$. In the French system, a drug delivery must be renew at least every month. Then, we assume that a drug delivery correspond to a 30 day drug exposure.
2. joint intervals are recursively merged. If two interval-based event $(e, [l_1, u_1])$ and $(e, [l_2, u_2])$ are such that $u_1 + 7 \geq l_2$ then the two intervals are merged into an interval $(e, [l_1, u_2])$. We made the hypothesis of a delivery renew every 30 days. The value 7 allows to take into account a certain flexibility to consider that successive deliveries are contiguous.

The dataset of $18,499$ sequences preceding seizure have been preprocessed to generate as much interval-based sequences. The transformation reduces the number of events from $341,100$ to $188,512$.

E.3.2 QTIPrefixSpan

The setup of QTIPrefixSpan is the following: minimal frequency threshold $f_{min} = 15\%$, similarity threshold $\epsilon = 30$ with Hausdorff distance. We bounded the size of the symbolic signature to 4 items.

The algorithm extracted 142 patterns in few seconds. While focusing on patterns having both 114 and 383 events, two patterns are more especially interesting (illustrated in Figure 3.19):

- ▷ $\boldsymbol{p_1}^{QTI} = \langle(86, [0, 93])\ (114, [33, 85])\ (383, [63, 98])\rangle$
- ▷ $\boldsymbol{p_2}^{QTI} = \langle(86, [6, 36])\ (383, [6, 36])\ (114, [61, 105])\ (86, [61, 111])\rangle$

Figure 3.19: Illustration of two patterns extracted by QTIPrefixSpan from interval-based temporal sequence of drug exposure preceding an epileptic seizure.

These two examples are interesting because they illustrate two types of switches between 114 and 383 drugs preceding epileptic seizures. $\boldsymbol{p_1}^{QTI}$ illustrates a change from 114 to 383. The overlap between the two intervals has to be carefully analyzed because, it does not necessarily means that the patient had both drugs at the same time. Indeed, we have seen that the *representative* nature of QTI patterns leads algorithms to enlarge the intervals extends (see discussion in Section B.4).

$\boldsymbol{p_2}^{QTI}$ illustrate a switch from 383 to 114. The event 86 which (*paracetamol*) is here also split in two different events. This pattern illustrates that our approach can extract repetitive patterns.

F Conclusion and perspectives

This Chapter presented pattern domains and algorithms to extract frequent patterns where quantitative temporal information is part of the extracted knowledge. It contributes to answer the challenge of enhancing sequential patterns with metric temporal information.

More precisely, we had initially two questions about extracting temporal pattern mining:

- ▷ **is it practically tractable?** Our answer is that the proposed solutions are practical for mid-size data set (data set with less than $100,000$ sequences of length lower than 30). Indeed, the experiments on synthetic dataset shown that extracting temporal information requires computational resources (time and memory). Such data characteristics allows to address a wide range of applications, even if it requires to prepare the dataset (e.g. by sampling sequences). Large data set may be difficult to mine with our algorithm.
- ▷ **does it bring new interesting information?** Our case study on care pathway analysis illustrates potential interest for having such kind of expressive pattern while analyzing temporal data. Nonetheless, we put light on the necessity to carefully discuss the semantic of pattern while proposing a new pattern domain (e.g. boundaries of discriminant patterns, negation in negative sequential patterns). Simple syntax may hide complex semantics and possible misinterpretation.

A focus on pattern semantics

The main motivation of our work is to extract *meaningful* patterns. Contrary to most of the pattern mining algorithms, we do not think about the computational efficiency first, but about the semantic enhancement of patterns first. Then, we explore the formal properties of the desired pattern domains (e.g. chronicles, or negative sequential patterns) to propose algorithm strategies to extract patterns in reasonable time on mid-size dataset. Scaling up our algorithm is still a huge challenge.

Despite the limitations of proposed algorithms, mining temporal sequences attracts the interest of data analysts in various applications (e.g. health care pathways, emotion detection from facial

images, electricity consumption, ECG), and some extensions have been proposed by other research groups. The *DCM* algorithm has been used in two other contexts:

- ▷ Sellami et al. [2018] used *DCM* for predictive maintenance purposes in an industrial context,
- ▷ K. Fauvel (PhD Candidate in LACODAM) explored chronicles and *DCM* for predicting oestrus of cows during the I. Harada internship

The QTempIntMiner and QTIPrefixSpan algorithms have been extended by several research teams in various directions:

- ▷ in Ruan et al. [2014], they proposed to parallelize the algorithm.
- ▷ in Hassani et al. [2016], they explore alternative distance.
- ▷ in Dermouche and Pelachaud [2016], they propose to use a vertical format to use a search strategy inspired from SPADE Zaki [2001].

Finally, NTGSP has been used by EDF to analyze their customer pathways.

Toward a general framework

This research line started with the proposal of QTempIntMiner. To the best of our knowledge, this algorithm proposed in 2008 [Guyet and Quiniou, 2008] was the first temporal pattern mining algorithm to combine pattern domain exploration and machine learning. It initiates our work which leads to propose several algorithms presented in this Chapter: QTempIntMiner, QTIPrefixSpan, *DCM*, NTGSP. Their principles are similar and based on temporal pattern definition in two parts:

- ▷ **a symbolic signature: the structural skeleton of the pattern**; Symbolic signatures are structured in a poset (ideally a concept lattice) such that the classical tricks of frequent pattern mining ensures their efficient extraction. The different types of symbolic signatures are: sequences, multisets or negative patterns.
- ▷ **temporal constraints: it enhances the symbolic signature with temporal information**; Then, through the "projection" of each symbolic signature on sequences, we build a tabular dataset representing timestamps of occurrences. A tabular dataset is processed by a clustering (KMeans or CLIQUE) or classification algorithm ($Ripper_k$) to extract meaningful temporal information.

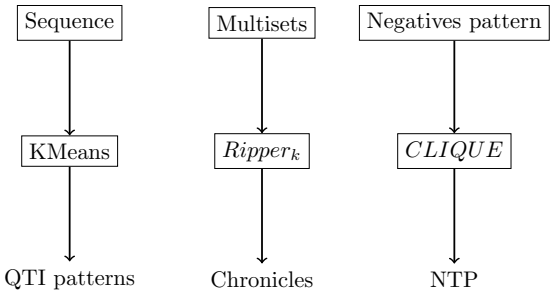


Figure 3.20: Parallelism between the three proposed algorithms, from left to right: QTempIntMiner, *DCM* and NTGSP.

Figure 3.20 illustrates the three proposed algorithm. It shows the obvious similarities between this approach, and it suggest that a generic framework may be developed.

Perspectives

Then, the first perspective of this research line is to unify some of our approaches, namely chronicles and negative patterns. With NTGSP, we shown that modeling absent events and modeling temporal delay between events may be seen as constraints on the occurrences of a sequential pattern. With chronicles, multisets are preferred to sequential patterns with constraints between pairs of events. A unification would be to have couples $(\mathcal{E}, \mathcal{T})$ where $\mathcal{E}$ is a multiset of event types and $\mathcal{T}$ is a set of constraints that are temporal constraints or negation constraints. This perspective

of designing such a pattern domain combines the expressivity of negative patterns and chronicles; and we expect inheritance of their computational properties to propose efficient algorithms.

To improve the efficiency of our approaches, we believe in sampling approaches (see Section A.3.3 of Chapter 2). Approaches based on MCTS are dedicated to the exploration of large search space, and the solution developed by Bosc et al. [2018] to extract diverse set of patterns may be adapted to temporal patterns.

Finally, there is some space to better promote our algorithm implementations. Indeed, the most population implementation of sequential pattern mining, namely SPMF library Fournier-Viger et al. [2016], does not have any algorithm to extract temporal patterns. Releasing solid implementation of our algorithms as a library of temporal pattern mining may be useful for a broad range of applications.

Chapter 4

Declarative Sequential Pattern Mining[1]

The challenge of mining a deluge of data is about to be solved, but is also about to be replaced by another issue: the deluge of patterns. In fact, the size of the complete set of frequent patterns explodes with the database size and density [Lhote, 2010]. The data analyst cannot handle such volumes of results. A broad range of research, from data visualization [Perer and Wang, 2014] to database sampling [Low-Kam et al., 2013] is currently attempting to tackle this issue. The data-mining community has mainly focused on the addition of expert constraints on sequential patterns [Pei et al., 2004].

Recent approaches have renewed the field of Inductive Logic Programming (ILP) [Muggleton and De Raedt, 1994] by exploring declarative data mining. Declarative data mining consists in encoding data mining or machine learning tasks in declarative languages (SAT, CP, Linear Programming, Logic Programs, etc) [De Raedt, 2012]. One of the objective is to benefit from their versatility to ease the addition of constraints. This approach has been developed for clustering [Adam et al., 2013; Dao et al., 2018], for supervised machine learning tasks (e.g. rule learning, ILP), but more especially for pattern mining tasks.

Many approaches addressed the problem of declarative pattern mining, especially itemset mining [Guns et al., 2015; Järvisalo, 2011; Jabbour et al., 2018]. Some propositions have extended the approach to sequence mining [Negrevergne and Guns, 2015; Métivier et al., 2013; Jabbour et al., 2013; Coquery et al., 2012]. Their practical use depends on the efficiency of their encoding to process real datasets. Thanks the improvements on satisfiability (SAT) or constraints programming (CP) solving techniques and solvers, such approaches become realistic alternatives for highly constrained mining tasks. Their computational performances closely reach those of dedicated algorithms.

The long term objective is to benefit from the genericity and versatility of solvers to let a user specify a potentially infinite range of constraints on the patterns. Thus, we expect to go from specific algorithm constraints to a rich query language for pattern mining.

This part presents the use of Answer Set Programming (ASP) to mine sequential patterns. ASP is a high-level declarative logic programming paradigm for high level encoding combinatorial and optimization problem solving as well as knowledge representation and reasoning. In ASP, the programmer has to specify a search space (choice rules) and the description of the search space elements to select (constraints). This maps to the problem of pattern mining in which interesting patterns (e.g. frequent ones) are selected among the space of patterns. Thus, ASP is a good candidate for implementing pattern mining with background knowledge, which has been a data mining issue for a long time.

The contributions of this chapter are the following:

[1] This part is a collective work with Yves Moinard, René Quiniou, Torsten Schaub, Martin Gebser and Javier Romero

- ▷ We propose **encodings of the classical frequent sequential pattern mining task** within two representations of embeddings (*fill-gaps* vs *skip-gaps*).
- ▷ We illustrate the expressiveness of ASP by presenting encodings of **different variant of sequential pattern mining** tasks (closed, maximal, constraints, rares, etc.)
- ▷ We apply our framework to the mining of care pathways. It shows the advantage of ASP compare to other declarative paradigms to **blend mining and reasoning**.

A ASP – Answer Set Programming

In this section we introduce the Answer Set Programming (ASP) paradigm, syntax and tools. Section A.1 introduces the main principles and notations of ASP. Section A.2 illustrates them on the well-known graph coloring problem.

A.1 Principles of Answer Set Programming

ASP is a declarative programming paradigm. From a general point of view, declarative programming gives a description of what is a problem instead of specifying how to solve it. Several declarative paradigms have been proposed, differing in the modeling formalism they use. For instance, logic programming [Lallouet et al., 2013] specifies the problem using a logic formalism, the SAT paradigm encodes the problem with boolean expressions [Biere et al., 2009], the CP (constraint programming) paradigm specifies the problem using constraints [Rossi et al., 2006]. ASP belongs to the class of logic programming paradigms, such as Prolog.

An *ASP program* is a set of rules of the form

$$\mathtt{a_0} \texttt{ :- } \mathtt{a_1}, \ldots, \mathtt{a_m}, \texttt{not } \mathtt{a_{m+1}}, \ldots, \texttt{not } \mathtt{a_n}. \tag{4.1}$$

where each $\mathtt{a_i}$ is a propositional atom for $\mathtt{0} \leq \mathtt{i} \leq \mathtt{n}$ and `not` stands for *default negation*. In the body of the rule, commas denote conjunctions between atoms. Contrary to Prolog, the order of atoms is meaningless. In ASP, rule (4.1) may be interpreted as "*if* $\mathtt{a_1}, \ldots, \mathtt{a_m}$ *are all true and if none of* $\mathtt{a_{n+1}}, \ldots, \mathtt{a_n}$ *can be proved to be true, then* $\mathtt{a_0}$ *is true.*"

If $\mathtt{n} = \mathtt{0}$, i.e. the rule body is empty, (4.1) is called a *fact* and the symbol "`:-`" may be omitted. Such a rule states that the atom a_0 has to be true. If $\mathtt{a_0}$ is omitted, i.e. the rule head is empty, (4.1) represents an integrity constraint.

Semantically, a logic program induces a collection of so-called *answer sets*, which are distinguished models of the program determined by answer sets semantics; see Gelfond and Lifschitz [1991] for details. For short, a model assigns a truth value to each propositional atoms of the program and this set of assignments is valid. An answer set is a minimal set of true propositional atoms that satisfies all the program rules. Answer sets are said to be minimal in the way that only atoms that have to be true are actually true.

To ease the use of ASP in practice, several extensions have been developed. First of all, rules with *variables* are viewed as shorthands for the set of their ground instances. This allows for writing logic programs using a first order syntax. Such kind of syntax makes program shorter, but it hides the grounding step and its specific encoding issues, especially from the memory management point of view.

Further language constructs include *conditional literals* and *cardinality constraints* [Simons et al., 2002]. The former are of the form

$$\mathtt{a} : \mathtt{b_1}, \ldots, \mathtt{b_m}$$

the latter can be written as

$$\mathtt{s}\ \{\mathtt{c_1}; \ldots; \mathtt{c_n}\}\ \mathtt{t}$$

where `a` and $\mathtt{b_i}$ are possibly default negated literals for $0 \leq i \leq m$, and each $\mathtt{c_j}$ is a conditional literal for $1 \leq i \leq n$. The purpose of conditional literals is to govern the instantiation of a literal `a` through the literals $\mathtt{b_1}, \ldots, \mathtt{b_m}$. In a cardinality constraint, `s` (resp. `t`) provides the lower (resp. upper) bound on the number of literals from $\mathtt{c_1}; \ldots; \mathtt{c_n}$ that must be satisfied in the model.

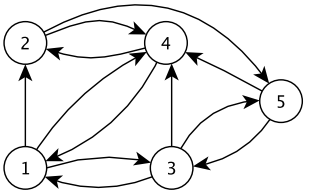


Figure 4.1: An example graph for the graph coloring problem.

A cardinality constraint in the head of the rule defines a specific rule called a *choice rule*:

$$\texttt{s}\ \{\texttt{c}_1;\ldots;\texttt{c}_\texttt{n}\}\ \texttt{t :- a.}$$

If `a` is true then all atoms of a subset $\mathcal{S} \subset \{c_1, \ldots, c_n\}$ of size between s and t have to be true. All such subsets are admissible according to this unique rule, but not in the same model. All such subsets contribute to alternative answer sets. It should be noted that alternative models are solved independently. It is not possible to specify constraints that involve several models.

The practical value of both constructs becomes more apparent when used in conjunction with variables. For instance, a conditional literal like `a(X) : b(X)` in a rule's antecedent expands to the conjunction of all instances of `a(X)` for which the corresponding instance of `b(X)` holds. Similarly, `2 {a(X) : b(X)} 4` holds whenever between two and four instances of `a(X)` (subject to `b(X)`) are true.

ASP problem solving is ensured by efficient solvers [Lifschitz, 2008] which are based on the same technologies as constraint programming solvers or satisfiability checking (SAT) solvers. *smodels* [Syrjänen and Niemelä, 2001], *dlv* [Leone et al., 2006], *ASPeRiX* [Lefèvre and Nicolas, 2009] or *clingo* [Gebser et al., 2011] are well-known ASP solvers. Due to the computational efficiency it has demonstrated and its broad application to real problems, we use *clingo* as a basic tool for designing our encodings.

The basic method for programming in ASP is to follow a *generate-and-test* methodology. Choice rules generate solution candidates, while integrity constraints are tested to eliminate those candidates that violate the constraints. The programmer should not have any concern about how solutions are generated. He/she just has to know that all possible solutions will be actually evaluated. From this point of view, the ASP programming principle is closer to CP programming than to Prolog programming. Similarly to these declarative programming approaches, the difficulty of programming in ASP lies in the choices for the best way to encode problem constraints: it must be seen as the definition of the search space (*generate* part) or as an additional constraint on solutions within this search space (*test* part). This choices may have a large impact on the efficiency of the problem encoding.

A.2 A simple example of ASP program

The following example illustrates the ASP syntax on encoding the graph coloring problem. Lines 9-10 specify the problem as general rules that solutions must satisfy while lines 1-6 give the input data that defines the problem instance related to the graph in Figure 4.1.

The problem instance is a set of colors, encoded with predicates `col/1` and a graph, encoded with predicates `vertex/1` and `edge/2`. The input graph has 5 vertices numbered from 1 to 5 and 12 edges. The 5 fact-rules describing the vertex are listed in the same line (Line 2). It should be noted that, generally speaking, `edge(1,2)` is different from `edge(2,1)`, but considering that integrity constraints for the graph coloring problem are symmetric, it is sufficient to encode directed edge in only one direction. Line 6 encodes the 3 colors that can be used: `r`, `g` and `b`. Lower case letters represent values, internally encoded as integers, while strings beginning with upper case letters represent variables (see line 9 for instance).

Lines 9 and 10 specify the graph coloring problem. The predicate `color/2` encodes the color of a vertex: `color(X,C)` expresses that vertex `X` has color `C`. Line 10 is an integrity constraint. It

```
1 % instance of the problem: the graph and colors
2 vertex(1). vertex(2). vertex(3). vertex(4). vertex(5).
3 edge(1,2). edge(1,3). edge(1,4). edge(2,4). edge(2,5).
4 edge(3,1). edge(3,4). edge(3,5). edge(4,1). edge(4,2).
5 edge(5,3). edge(5,4).
6 col(r). col(b). col(g).
7
8 % graph coloring problem specification
9 1 { color(X, C) : col(C) } 1 :- vertex(X).
10 :- edge(X, Y), color(X, C), color(Y, C).
```

Listing 4.1: Encoding of graph coloring – ASP syntax and encoding example

forbids neighbor vertices `X` and `Y` to have the same color `C`[2]. The ease of expressing such integrity constraints is a major feature of ASP.

Line 9 is a choice rule indicating that for a given vertex `X`, an answer set must contain exactly one atom of the form `color(X,C)` where `C` is a color. The grounded version of this rule is the following:

```
1 { color(1, r), color(1, b), color(1, g) } 1.
1 { color(2, r), color(2, b), color(2, g) } 1.
1 { color(3, r), color(3, b), color(3, g) } 1.
1 { color(4, r), color(4, b), color(4, g) } 1.
1 { color(5, r), color(5, b), color(5, g) } 1.
```

The variable `X` is expanded according to the facts in line 2 and for each vertex, a specific choice rule is defined. Within the brackets, the variable `C` is expanded according to the conditional expression in the rule head of line 9: the only admissible values for `C` are color values (`r`, `g` and `b`). For each line of the grounded version of the program, one and only one atom within brackets can be chosen. This corresponds to a unique mapping of a color to a vertex. Line 9 can be seen as a search space generator for the graph coloring problem.

The set `color(1,b) color(2,r) color(3,r) color(4,g) color(5,b)` is an answer set for the above program (among several others).

For more detailed presentation of ASP programming paradigm, we refer the reader to the recent article of Janhunen and Niemelä [2016].

A.3 The *Potassco* collection of ASP tools

The *Potassco* collection is a set of tools for ASP developed at the University of Potsdam. The main tool of the collection is the ASP solver *clingo* [Gebser et al., 2011]. This solver offers both a rich syntax to facilitate encodings[3] and a good solving efficiency.

The *clingo* solving process follows two consecutive main steps:

1. *grounding* transforms the initial ASP program into a set of propositional clauses, cardinality constraints and optimisation clauses. Note that grounding is not simply a systematic problem transformation. It also simplifies the rules to generate the as short as possible equivalent grounded program.
2. *solving* consists in finding from one to all solutions of the grounded program. This step is performed by *clasp* which is a conflict-driven ASP solver. The primary *clasp* algorithm relies on conflict-driven *nogood* learning. It is further optimized using sophisticated reasoning and implementation techniques, some specific to ASP, others borrowed from CDCL-based SAT solvers.

[2] It is important to notice that the scope of a variable is the rule and each occurrence of a variable in a rule represents the same value.

[3] *clingo* is fully compliant with the recent ASP standard: https://www.mat.unical.it/aspcomp2013/ASPStandardization

asprin [Brewka et al., 2015] is an extension of the *clingo* solver which provides a generic framework for implementing a broad range of preferences relations on ASP models and can easily manage them. An ASP-based system enables expressing combinations of qualitative and quantitative preferences among answer sets. A *preference relation* is defined via a declaration of the form

$$\texttt{\#preference(p,t)}\ \ \{\texttt{c}_1,\ldots,\texttt{c}_n\}.$$

where `p` and `t` are the name and type of the preference relation, respectively, and each $\texttt{c}_j$ is a conditional literal. The directive `#optimize(p)` instructs *asprin* to search for `p`-optimal answer sets, being maximal wrt the strict preference relation associated with `p`. While *asprin* already comes with a library of predefined primitives and aggregate preference types, like `subset` or `pareto`, respectively, it also allows for adding customized preferences. To this end, users provide rules defining an atom `#better(p)` that indicates whether an answer set X is preferred to another Y wrt the preference relation associated with `p`. Both sets X and Y are reified by *asprin* via unary predicates `holds/1` and `holds'/1`,[4] whose instances are drawn upon in the definition of `#better(p)` .

B Frequent sequential pattern mining with ASP

In this section, we present several ASP encodings for frequent sequential pattern mining. We assume that the database contains sequences of itemsets. But for the sake of simplicity, we will restrict patterns to sequences of items (each itemset is a singleton).

Sequential pattern mining has strong connections with ASP principles. The sequential pattern mining task rises two issues: 1) exploring a large search space, i.e. the potentially infinite set of sequences and 2) assessing the frequency constraint (with respect to the given database). Thus, sequential pattern mining can be considered as a *generate and test* process which makes it straightforward to encode the mining task using ASP principles: 1) choice rules will define the search space and 2) the frequency assessment will be encoded using integrity constraints.

Our proposal is borrowed from Järvisalo [2011]: the solution of the sequential pattern mining ASP program is all the answer sets (AS), each of which contains the atoms describing a single frequent pattern as well as its occurrences in database sequences. The solution relies on the "generate and test principle": 1) generate all possible patterns with their occurrences in the sequences database; and 2) test whether they satisfy the specified constraints.

B.1 Modeling database, patterns and parameters

A sequence database $\mathcal{D}$ is modeled by the predicate `seq(T,Is,I)` which holds if sequence `T` contains item `I` at index `Is`. Itemsets are encoded by several distinct items at the same index.

Example 23 (A sequence database encoded in ASP)**.** *Table 4.1 reminds the sequence database of Example 6, page 7.*

Id	Sequence
s_1	$\langle a\ c\rangle$
s_2	$\langle a\ b\ (bc)\ c\rangle$
s_3	$\langle d\ b\rangle$
s_4	$\langle a\ (bc)\ c\rangle$
s_5	$\langle a\ b\ d\rangle$
s_6	$\langle a\ c\ b\ c\rangle$
s_7	$\langle a\ (bc)\ c\rangle$

Table 4.1: Dataset of sequences (see Example 6).

The following facts encode the database of Table 4.1:

[4] That is, `holds(a)` (or `holds'(a)`) is true iff $\texttt{a}\in X$ (or $\texttt{a}\in Y$).

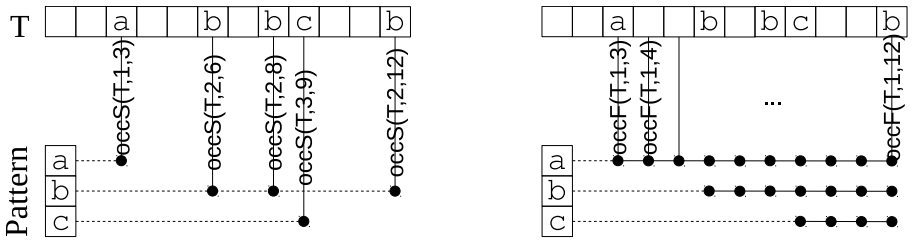


Figure 4.2: Illustration of embeddings strategies. On the left, *skip-gaps*, on the right, *fill-gaps*.

```
seq(1,1,a). seq(1,2,c).
seq(2,1,a). seq(2,2,b). seq(2,3,b). seq(2,3,c). seq(2,4,c).
seq(3,1,d). seq(3,1,b).
seq(4,1,a). seq(4,2,b). seq(4,2,c). seq(4,3,c).
seq(5,1,a). seq(5,2,b). seq(5,2,d).
seq(6,1,a). seq(6,2,c). seq(6,3,b). seq(6,4,c).
seq(7,1,a). seq(7,2,b). seq(7,2,c). seq(7,3,c).
```

Similarly, the current pattern is modeled by predicate `pat(Ip,I)` which holds if a pattern contains item `I` at index `Ip`.

For example, the pattern $\langle a\ b\ c\rangle$ is modeled by the following atoms:
`pat(1,a).pat(2,b).pat(3,c).`

In addition, we define two program constants:

- ▷ `#const th=23.` represents $f_m in$, the minimal frequency threshold, i.e. the requested minimal number of supporting sequences
- ▷ `#const maxlen=10.` represents the maximal pattern length.

Let S be a set of ground atoms and $P \subset S$ the set of `pat(Ip,I)` atoms in S, according to the Järvisalo's encoding principle we would like an ASP program π such that S is an answer set of π iff the pattern defined by P is a frequent sequential pattern in the database $\mathcal{D}$.

B.2 Two encodings for sequential pattern mining

The main difficulty in declarative sequential pattern mining is to decide whether a pattern $\boldsymbol{p} = \langle p_1\ \ldots\ p_n\rangle$ supports a sequence $\boldsymbol{s} = \langle s_1\ \ldots\ s_m\rangle$ of the database. According to Definition 32, it means that it exists an embedding $e = (e_i)_{1\leq i\leq n}$ such that $p_i = s_{e_i}$. Unfortunately, this definition is not usable in practice to implement an efficient ASP encoding. The difficulty comes from the possible multiple instances of a pattern in a single sequence. On the other hand, the detailed mapping description is not required here, we simply have to define embeddings that exist iff a pattern supports a sequence. An embedding of a pattern in a sequence is given by the description of a relation between pattern item indexes and sequence item indexes.

This section presents two encodings of sequential pattern mining. These two encodings differ in their representation of embeddings, as illustrated in Figure 4.2. Two embedding strategies have been defined and compared in our results: *skip-gaps* and *fill-gaps*.

More formally, let $\boldsymbol{p} = \langle p_1\ \ldots\ p_n\rangle$ be a pattern sequence and $\boldsymbol{s_T} = \langle s_1\ \ldots\ s_m\rangle$ be the T-th sequence of $\mathcal{D}$. In the *skip-gaps* strategy, an embedding $\mathcal{E}$ is a relation over $[1,m]\times[1,n]$ such that $\forall(i,j)\in\mathcal{E},\ i\leq j \wedge p_i = s_j$ and $\forall(i,j),(i',j')\in\mathcal{E},\ i<i' \implies j<j'$. In the *fill-gaps* strategy, an embedding $\mathcal{E}'$ is the same relation as $\mathcal{E}$ (i.e. $(i,j)\in\mathcal{E} \implies (i,j)\in\mathcal{E}'$) with the additional specification: $\forall i\in[1,m],\ \forall j,j'\in[1,n],\ (i,j)\in\mathcal{E}' \wedge j<j' \implies (i,j')\in\mathcal{E}'$. This additional specification expresses that once a pattern item has been mapped to the leftmost (having the lowest index, let it be j), the knowledge of this mapping is maintained on remaining sequences items with indexes $j'>j$. So, a *fill-gaps* embedding makes only explicit the "leftmost admissible matches" of $\boldsymbol{p}$ items in sequence $\boldsymbol{s_T}$.

```
1 item(I) :- seq(_, _,I).
2
3 %sequential pattern generation
4 patpos(1).
5 0 { patpos(Ip+1) } 1 :- patpos(Ip), Ip<maxlen.
6 patlen(L) :- patpos(L), not patpos(L+1).
7
8 1 { pat(Ip,I): item(I) } 1 :- patpos(Ip).
9
10 %pattern embeddings
11 occS(T,1   ,Is):- seq(T,Is,I), pat(1,   I).
12 occS(T,Ip+1,Is):- seq(T,Is,I), pat(Ip+1,I), occS(T,Ip,Js), Js<Is.
13
14 %frequency constraint
15 support(T) :- occS(T,L,_), patlen(L).
16 :- #count{T: support(T) } < th.
```

Listing 4.2: Encoding of frequent sequential pattern mining – skip-gaps strategy

Relations $\mathcal{E}$ and $\mathcal{E}'$ are interesting because (i) that can be computed in a constructive way (i.e. without encoding guesses) and (ii) they contains the information required to decide whether the pattern supports the sequence.

The two following sections detail the ASP programs for extracting patterns under each embedding strategy.

B.2.1 The skip-gaps approach

In the first ASP encoding, an embedding of the current pattern $\boldsymbol{p} = \langle p_1 \ \ldots \ p_n \rangle$ in sequence $\boldsymbol{s} = \langle s_1 \ \ldots \ s_m \rangle$ is described by a set of atoms `occS(T,Ip,Is)` which holds if the pattern item occurring at index `Ip` is identical to the sequence item occurring in sequence `T` at position `Is` (formally, $p_{\texttt{Ip}} = s_{\texttt{Is}} \wedge \langle p_1 \ \ldots \ p_{\texttt{Ip}} \rangle \preceq \langle s_1 \ \ldots \ s_{\texttt{Is}} \rangle$). The set of valid atoms `occS(T,_,_)` encodes the relation $\mathcal{E}$ above and is illustrated in Figure 4.2 (on the left).

Example 24 (Illustration of skip-gaps embedding approach)**.** *Let $\boldsymbol{p} = \langle a \ c \rangle$ be a pattern represented by terms `pat(1,a)` and `pat(2,c)` Here, we provide the embeddings of pattern $\boldsymbol{p}$ in the sequences of example 23:*

```
occS(1,1,1) occS(1,2,2)
occS(2,1,2) occS(2,2,4)
occS(4,1,1) occS(4,2,3)
occS(5,1,1)
occS(6,1,1) occS(6,2,2) occS(6,2,4)
occS(7,1,1) occS(7,2,3)
```

The pattern could not be fully identified in the fifth sequence. There are two possible embeddings in the sixth sequence. Atom `occS(6,1,1)` is used for both. Nonetheless, this sequence must be counted only once in the support.

Listing 4.2 gives the ASP program for sequential pattern mining. The first line of the program is called a *projection*. It defines a new predicate that provides all items from the database. The symbol "`_`" denotes an anonymous (or don't care) variable.

Lines 4 to 8 encode pattern generation. Predicate `patpos/1` defines allowed indexes for a sequential pattern, beginning at index 1 (line 4). Line 5 is a choice rule that generates the successive pattern positions up to an ending index iterating from 2 to `maxlen`: `patpos(Ip+1)` is true if there is a pattern position at index `Ip` and `Ip` is lower than `maxlen`. Line 6 defines the length of a pattern: `patlen(L)` holds if `L` is the index of the last pattern item (there is no pattern item with a greater index). This predicate is used to decide whether an embedding has been completed or

```
10 %pattern embeddings
11 occF(T,1,Is)  :- seq(T,Is,I), pat(1,I).
12 occF(T,Ip,Is) :- occF(T, Ip-1, Is-1), seq(T,Is,I), pat(L,I).
13 occF(T,Ip,Is) :- occF(T, Ip,   Is-1), seq(T,Is,_).
14
15 %frequency constraint
16 seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
17 support(T) :- occF(T, L, LS), patlen(L), seqlen(T,LS).
18 :- #count{ T : support(T) } < th.
```

Listing 4.3: Encoding of frequent sequential pattern mining – fill-gaps strategy (see Listing 4.2 for first lines).

not. Finally, line 8 is a choice rule that associates exactly one item with each position `X`. We can note that each possible sequence is generated once and only once. So, there is no redundancy in the search space exploration.

Lines 11 to 12 encode pattern embedding search. Line 11 guesses a sequence index for the first pattern item: `occS(T,1,Is)` holds if the first pattern item is identical to the `Is`-th of sequence `T` (i.e. $p_1 = s_{\texttt{Is}}$). Line 12 guesses sequence indexes for pattern items at indexes strictly greater than 1. `occS(T,Ip,Is)` holds if the `Ip`-th pattern item is equal to the `Is`-th sequence item (i.e. $p_{\texttt{Ip}} = s_{\texttt{Is}}$) and the preceding pattern item is mapped to a sequence item at an index strictly lower than `Is`. Formally, this rule expresses the following implication $(\texttt{Jp}, \texttt{Is}-1) \in \mathcal{E} \wedge p_{\texttt{Ip}} = s_{\texttt{Is}} \wedge \texttt{Ip} > \texttt{Jp} \implies (\texttt{Ip}, \texttt{Is}) \in \mathcal{E}$ and recursively, we have $\langle p_1 \ \ldots \ p_{\texttt{Ip}} \rangle \preceq \langle s_1 \ \ldots \ s_{\texttt{Is}} \rangle$. It should be noted that this encoding generates all the possible embeddings of some pattern.

Finally, lines 15 to 16 are dedicated to assess the pattern frequency constraint. `support(T)` holds if the database sequence `T` supports the pattern, i.e. if an atom `occS` holds for the last pattern position. The last line of the program is an integrity constraint ensuring that the number of supported sequences is not lower than the threshold `th` or, in other words, that the support of the current pattern is greater than or equal to the threshold.

B.2.2 The fill-gaps approach

In the *fill-gap* approach, an embedding of the current pattern P is described by a set of atoms `occF(T,Ip,Is)` having a slightly different semantics than in the *skip-gap* approach. `occF(T,Ip,Is)` holds if at sequence index `Is` it is true that the `Ip`-th pattern item has been mapped (to some sequence index equal to `Is` or lower than `Is` if `occF(T,Ip,Is-1)` holds). More formally, we have $\langle p_1 \ \ldots \ p_{\texttt{Ip}} \rangle \preceq \langle s_1 \ \ldots \ s_{\texttt{Is}} \rangle$. The set of atoms `occF(T,_,_)` encodes the relation $\mathcal{E}'$ above and is illustrated in Figure 4.2 (on the right).

Example 25 (Fill-gaps approach embedding example)**.** *Pattern $P = \langle a \ c \rangle$ has the following fill-gaps embeddings (represented by `occF` atoms) in the sequences of the database of example 23:*

```
occF(1,1,1) occF(1,1,2) occF(1,2,2)
occF(2,1,2) occF(2,1,3) occF(2,1,4) occF(2,2,4)
occF(4,1,1) occF(4,1,2) occF(4,1,3) occF(4,2,3)
occF(5,1,1) occF(5,1,2)
occF(6,1,1) occF(6,1,2) occF(6,1,3) occF(6,1,4)
                        occF(6,2,2) occF(6,2,3) occF(6,2,4)
occF(7,1,1) occF(7,1,2) occF(7,1,3) occF(7,2,3)
```

Contrary to the skip-gap approach example (see Example 24), the set of `occF(T,Ip,Is)` atoms alone is not sufficient to deduce all occurrences. For instance, occurrence with indexes $(3, 8, 9)$ is masked.

Listing 4.3 gives the ASP program for sequential pattern mining with the fill-gaps strategy. The rules are quite similar to those encoding the skip-gaps method. The main difference comes from the computation of embeddings (lines 11-13). As in listing 4.2, line 11 guesses a sequence

index for the first pattern item: `occF(T,1,Is)` holds if the first pattern item is identical to the `Is`-th of sequence `T` (i.e. $p_{\texttt{Ip}} = s_{\texttt{Is}}$).

Line 12 guesses sequence indexes for pattern items at indexes strictly greater than 1. `occS(T,Ip,Is)` holds if the `Ip`-th pattern item is equal to the `Is`-th sequence item and the preceding pattern item is mapped to some sequence item at some index strictly lower than `Is`. More formally, we have that $p_{\texttt{Ip}} = s_{\texttt{Is}} \wedge (\texttt{Ip}-1, \texttt{Is}-1) \in \mathcal{E}' \implies (\texttt{Ip}, \texttt{Is}) \in \mathcal{E}'$.

Line 13 simply maintains the knowledge that the `Ip`-th pattern item has been mapped all along the further sequence indexes, i.e. `occF(T,Ip,Is)` holds if `occF(T,Ip,Is-1)` holds. More formally, $(\texttt{Ip}-1, \texttt{Is}-1) \in \mathcal{E}' \implies (\texttt{Ip}, \texttt{Is}) \in \mathcal{E}'$. In combination with previous rules, we thus have recursively that `occF(T,Ip,Is)` is equivalent to $\langle p_1 \ \ldots \ p_{\texttt{Ip}} \rangle \preceq \langle s_1 \ \ldots \ s_{\texttt{Is}} \rangle$.

Line 17 a sequence is supported by the pattern an `occF` atoms exists at the last position `LS` of the sequence, computed line 16. The remaining rules for testing whether it is greater than the threshold `th` are identical to those in the skip-gaps approach.

B.3 Sequential pattern mining improvements

The main objective of this subsection is to introduce three attempts to improve the computational efficiency of our encodings of sequential pattern mining. The encoding details can be found in Guyet et al. [2017c].

Filter out unfrequent items The first improvement consists in generating patterns from only frequent items. According to the anti-monotonicity property, all items in a pattern have to be frequent.

Using projected databases The idea of this alternative encoding is to use the principle of *projected databases* introduced by algorithm PrefixSpan [Pei et al., 2001]. Let $\boldsymbol{p} = \langle p_1 \ \ldots \ p_n \rangle$ be a pattern, the projected database of $\mathcal{D} = \{\boldsymbol{s_1}, \ldots, \boldsymbol{s_n}\}$ is $\{\boldsymbol{s'_1}, \ldots, \boldsymbol{s'_n}\}$ where $\boldsymbol{s'_i}$ is the projected sequence of $\boldsymbol{s_i}$ with respect to $\boldsymbol{p}$. Let $\boldsymbol{s_i} = \langle s_1 \ \ldots \ s_m \rangle$ be a sequence. Then the projected sequence of $\boldsymbol{s_i}$ is $\boldsymbol{s'_i} = \langle s_{k+1} \ \ldots \ s_m \rangle$ where k is the position of the last item of the first occurrence of $\boldsymbol{p}$ in $\boldsymbol{s_i}$. If $\boldsymbol{p}$ does not occur in $\boldsymbol{s_i}$ then $\boldsymbol{s'_i}$ is empty.
The idea is to improve the candidate generation part of the algorithm by using items from projected databases. Instead of generating a candidate sequential pattern by extending a frequent pattern with an item that is frequent in the whole database, the pattern extension rule uses only items that are frequent in the database projected along this pattern.

Mixing itemsets and sequences mining Järvisalo [2011] has shown that ASP can be efficient for itemset pattern mining. The main idea of this last alternative approach is to mine frequent itemsets and to derive sequential patterns from them.
This time, the itemset mining step extracts a frequent itemset pattern $I = (e_i)_{i \in [n]}$, $e_i \in \mathcal{I}$. A sequential pattern $S = (s_i)_{i \in [m]}$ is generated using the items of the itemset, i.e. $\forall i \in [m]$, $\exists j \in [n]$, $s_i = e_j$ taking into account that items may be repeated within a sequential pattern and that every item from I must appear in S. If not, there would exist a subset $J \subset I$ that would generate the same sequence s. This would lead to numerous redundant answer sets for similar frequent sequences and would cause a performance drop.

B.4 Experiments

Having demonstrated that modeling in ASP is powerful yet simple, it is now interesting to examine the computational behavior of ASP-based encodings.

The first experiments compare the performance, in runtime and memory requirements, of the various ASP programs presented before. The objective is to better understand the advantages and drawbacks of each encoding. The questions we would like to answer are: which of the two embedding strategies is the best? does the encoding improvement really reduce computing resources needs? what is the behaviour of our encoding with added pattern constraints?

Next, we compare our results with the CP-based ones of CPSM [Negrevergne and Guns, 2015]. CPSM constitutes a natural reference since it aims at solving a mining task similar to the present one and since CPSM adopts a semi-declarative approach, in particular, occurrence search is performed by a dedicated constraint propagator. CPSM is based on the CP solver *gecode*.

Table 4.2: Sequence generator parameters.

Parameter	Default value	Description
D	500	number of sequences in the database
l	20	sequence mean length (sequence length follows a normal law)
n	20	number of different patterns
lp	5	pattern mean length
th_D	10%	minimum number of occurrences generated for each pattern
k	50	alphabet size. The distribution of item occurrences follows a normal law ($\mu = .5$ and $\sigma = 0.05$). Some items occur more often than others.

In all presented experiments, we use the version 4.5 of *clingo*, with default solving parameters. For benchmarking on synthetic data, the ASP programs were run on a computing server with 8Go RAM without using the multi-threading mode of `clingo`. Multi-threading reduces the mean runtime but introduces variance due to the random allocation of tasks. Such variance is inconvenient for interpreting results with repeated executions. For real datasets, we used the multi-threading mode with 4 threads and 20Go shared RAM. This large amount of memory is required for large datasets.

B.4.1 Encodings comparisons on synthetic datasets

The first experiments were conducted on synthetic databases to control the most important features of data. We designed a sequence generator with same principle of the generator of Section B.3.1 of Chapter 3. Generator parameters and their default values are sum up in Table 4.2. Default values are those used when not explicitly specified.

The task to be solved is the extraction of the complete set of frequent patterns (see Section A of Chapter 2). It should be noted that every encoding extracts exactly the same set of patterns. Resource requirements are thus fairly comparable. The computation runtime is the time needed to extract all the patterns. It includes both grounding and solving of the ASP programs using the quiet *clingo* mode (no printed output). The memory consumption is evaluated from the size of the grounded program, i.e. the number of grounded atoms and rules. This approximation is accurate to compare ASP encodings. The solving process may require additional memory. This memory requirement is negligible compared to grounding.

We start with an overall comparison of the different encodings and their refinements with respect to parameters th_D and l. Figure 4.3 compares the runtimes for different encodings and the two embedding strategies, fill-gaps and skip-gaps. For each setting, 6 databases with the same characteristics were generated. Figure shows the mean runtime of the successful executions, i.e. those that extract the complete set of frequent pattern within the timeout period. The timeout was set to 20 minutes.

The exponential growth of the runtime when the threshold decreases is a classical result in pattern mining considering that the number of patterns grows exponentially. Every approach conforms to this behaviour. In more details:

- ▷ the longer the sequences, the greater the runtime. Most problem instances related to databases with $l = 10$ can be solved by any approach. When the mean length of sequences increases, the computation time increases also and the number of instances solved within the timeout period decreases. This can be easily explained by the combinatorics of computing embeddings which increases with the sequence length.
- ▷ all proposed improvements do improve runtime for high frequency thresholds on these small synthetic databases. For $f_{min} = 20\%$, the curve of every proposed improvement is below the curve of the basic encoding. For high thresholds, prefix-projection and itemsets improvements are significantly better. Nonetheless, the lower the threshold, the lower the difference between computation times. This shows that, except for prefix-projection, the improvements are not so efficient for hard mining tasks.

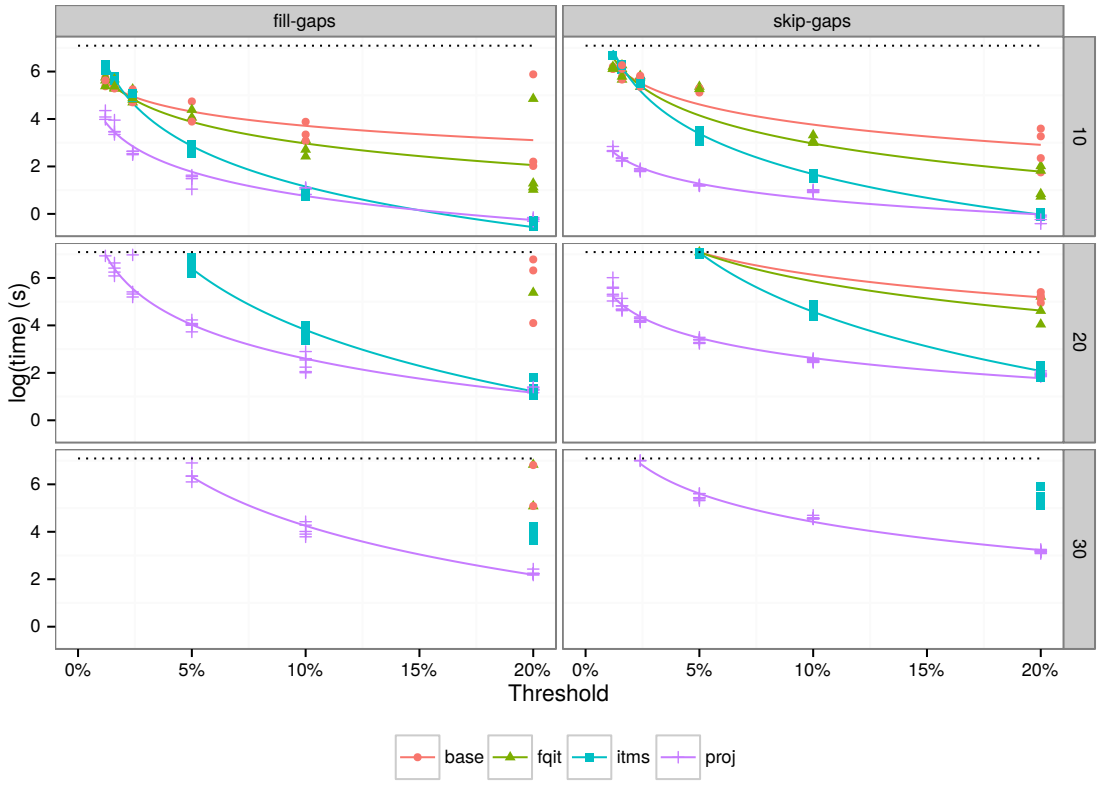


Figure 4.3: Mean computation time for synthetic databases with sequences length from 10 to 30 (in rows). Each curve represents an improvement to the basic encoding: none (base), frequent items (fqit), itemsets (itms), projection (proj). Each dot represents the mean of results on 6 datasets. The left-hand (resp. right-hand) column gives the results for the fill-gaps (resp. skip-gaps) strategy. The dashed horizontal line denotes the timeout of 20 minutes.

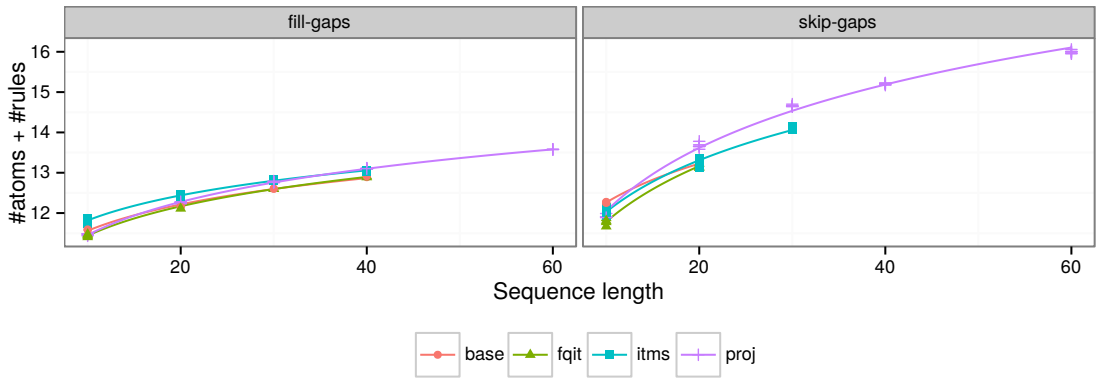


Figure 4.4: Memory requirement with respect to sequence length. Problem size (estimated memory requirement) for synthetic databases of sequences of length from 10 to 30 (in rows), under frequency threshold of 20%.

- ▷ the prefix-projection improvement is the fastest and reduces significantly the computation time (by 2 to 3 orders of magnitude).
- ▷ the skip-gaps strategy is more efficient than the fill-gaps strategy for these small datasets. The skip-gaps strategy requires less time to extract the same set of patterns than the fill-gaps strategy, for the same encoding improvements.

We will see below that this last result does not accurately predict which strategy should be preferred for mining real datasets. Before, we analyse the memory requirements of the different encodings.

We first note that the memory consumption is not related to the frequency threshold. This is a specificity of declarative pattern mining. Thus, Figure 4.4 compares the embedding strategies only for a unique frequency threshold $f_{min} = 20\%$. The curves show the number of grounded atoms and rules. As it represents a tight approximation of the memory requirement, we will refer to memory in the sequel.

Unsurprisingly, the richer the encoding is, the more memory is required. But the differences are not really significant, except for the prefix-projection programs (proj) which requires the highest number of atoms. We can see that using frequent itemsets (itms) is efficient to reduce the memory requirement. This means that the grounding step was able to exploit the additional rules to avoid the creation of useless atoms and rules. Such a kind of rules is really interesting because, as the algorithmic complexity of the mining task is not high, the efficiency of the ASP program is related to his grounding size.

In addition, from this last point of view, we can note that the fill-gaps strategy requires several order less memory than the skip-gaps strategy. The longer the sequences, the larger the difference. This result is illustrated by Figure 4.5. For each problem instance, the ratio of memory usage is computed by dividing the memory required by encoding with skip-gaps strategy with the memory required by the similar encoding with the fill-gaps strategy. Figure 4.5 illustrates with boxplots the dispersion of these ratios for different sequence lengths. Figure 4.5 clearly shows that the longer the sequences are, the more efficient the fill-gaps strategy is for memory consumption.

To end this overall comparison, it is interesting to come back to runtime. The overall results of Figure 4.3 show that the skip-gaps strategy seems better, but considering that the fill-gaps strategy requires less memory, it is interesting to analyse the evolution of computation time with respect to database size.

Figure 4.6 illustrates the ratio of runtimes in both strategies when the database size increases. The support threshold, f_{min}, is fixed to 10% and the sequence mean length to 20. We used the prefix-projection encoding for this experiment. Similarly to the previous figure, the ratios were individually computed for each pair of results (fill-gaps/skip-gaps) and the figure shows statistics

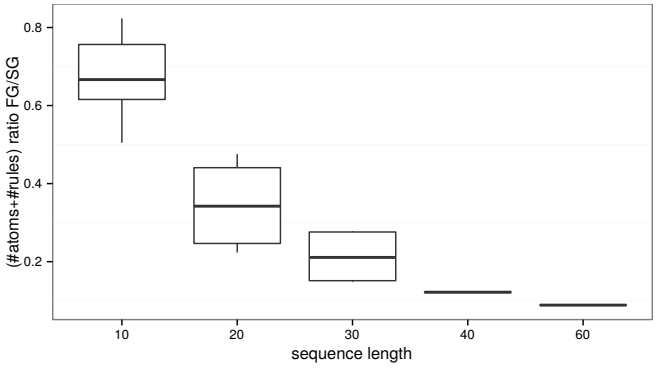


Figure 4.5: Dispersion of ratios of memory consumption obtained for the skip-gaps strategy to those obtained for the fill-gaps strategy. Boxplots were computed for problem instances with threshold at 20% and for all lengths and all encodings.

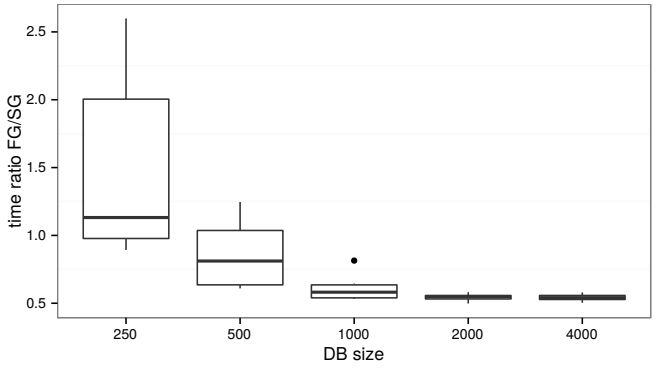


Figure 4.6: Dispersion of ratios of runtime of the skip-gaps strategy to the runtime of the fill-gaps strategy.

about these ratio. It shows that when the database size increases, the fill-gaps strategy becomes more efficient than the skip-gaps strategy.

From these experiments, we can conclude that combining prefix-projection with the fill-gaps strategy gives the best encoding. Thus, in the next subsection, we will compare this encoding with CPSM.

B.4.2 Real dataset analysis

In these experiments, we analyse the proposed encodings on processing real datasets. We use the same real datasets as selected in [Negrevergne and Guns, 2015] to have a representative panel of application domains:

- ▷ JMLR: a natural language processing dataset; each transaction is a paper abstract from the Journal of Machine Learning Research,
- ▷ UNIX: each transaction is a series of shell commands executed by a user during one session,
- ▷ iPGR: each transaction is a sequence of peptides that is known to cleave in presence of a Trypsin enzyme,
- ▷ FIFA: each transaction is a sequence of webpages visited by a user during a single session.

The dataset characteristics are sum up in Table 4.3. Some of them are similar to those of simulated datasets.

Table 4.3: Dataset characteristics: alphabet size, number of sequences and items, max and mean length of sequences, dataset density.

Dataset	$\|\mathcal{I}\|$	$\|D\|$	$\|\|D\|\|$	$max\|T\|$	$avg\|T\|$	*density*
Unix user	265	484	10935	1256	22.59	0.085
JMLR	3847	788	75646	231	96.00	0.025
iPRG	21	7573	98163	13	12.96	0.617
FIFA	20450	2990	741092	100	36.239	0.012

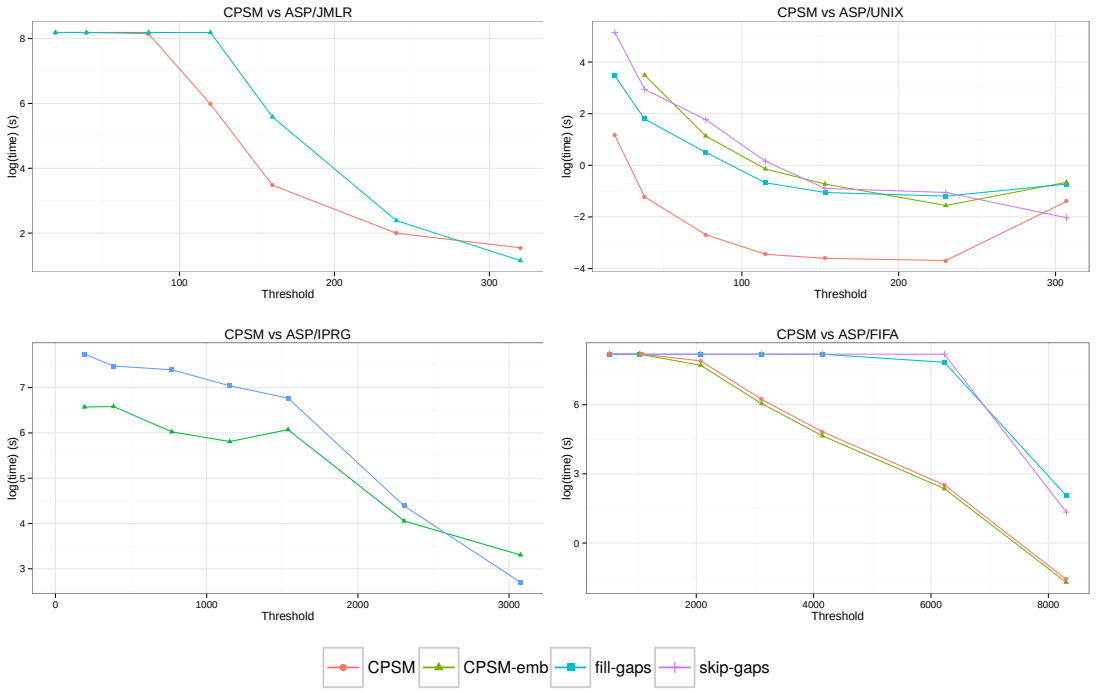


Figure 4.7: Runtime for mining frequent patterns with four approaches: CPSM, CPSM-emb, ASP with fill-gaps, ASP with skip-gaps.

B.4.3 Comparison of frequent pattern mining with CPSM

Figure 4.7 compares the runtimes of ASP-based sequence mining (using *clingo* solver) and CPSM. We ran the two versions of CPSM. CPSM makes use of global constraints to compute embeddings. This version is known to be very efficient, but it cannot cope with embedding constraints, while CPSM-emb does but is less efficient. We do not compare our approach with dedicated algorithms, which are known to be more efficient than declarative mining approaches (see Negrevergne and Guns [2015] for such comparisons). The timeout was set to 1 hour.

The results show that the runtimes obtained with *clingo* are comparable to CPSM-emb. It is lower for IPGR, very similar for UNIX and larger for JMLR. These results are consistant to those presented in [Gerbser et al., 2016] for synthetic datasets. When sequences become large, the efficiency of our encoding decreases somewhat. The mean length for JMLR is 96 while it is only 12.96 for iPRG. For CPSM with global constraints, the runtime-efficiency is several order of magnitude faster. To be fair, it should be noted that ASP approach ran with four parallel threads while CPSM-emb ran with no multi-threading since it does not support it. It should also be noted that CPSM requires a lot of memory, similarly to ASP-based solving.

B.5 Discussion

Although our declarative encodings cannot fully compete with the low-level efficiency of dedicated data mining systems on basic sequence mining tasks, our approach has an edge over them as regards expressiveness and extensibility. Nonetheless, our empirical analysis revealed that its performance matches that of a closely related CP-based approach [Negrevergne and Guns, 2015]. Beyond that, we have seen that the incorporation of preferences can lead to a significant reduction of pattern sets. All in all, our approach thus aims at processing mid-size databases and the enablement of analysts to extract patterns that they are really interested in.

Our approach follows the encoding principle of Järvisalo [2011], who used ASP for itemset mining. Unlike itemset mining, however, sequence mining is more demanding since we need to take the order of items in a pattern into account. Unlike that, our encodings leave particular embeddings of a pattern implicit and focus on sequences for which some embedding exists. More generally, SAT lacks a modeling language, which makes it hard for any SAT-based approach to achieve a similar level of extensibility [Coquery et al., 2012; Jabbour et al., 2013].

The CPSM system [Negrevergne and Guns, 2015] pursues a CP-based approach and tackles sequential pattern mining tasks similar to ours. While we rely on fully declarative encodings, CPSM uses global constraints for embedding computation. A global constraint uses a procedural propagator which assess the validity of some constraints in the CP program. Although such propagators can be implemented by space-efficient special-purpose algorithms, they cannot easily be extended to accommodate further constraints.

Recent versions of *clingo* are also equipped with similar propagator functionalities. It will be interesting to implement similar CPSM constructs and to find a trade-off between declarativeness and effectiveness.

C Alternative sequential pattern mining tasks

In this section, we illustrate how the previous encodings can be modified to solve more complex mining tasks. Our objective is to show the impressive expressiveness of ASP which let us encode a very wide range of mining tasks. We focus our attention on the most classical alternative sequential pattern mining tasks: constrained sequential patterns and condensed representation of sequential patterns.

Negrevergne and Guns [2015] organize the constraints on sequential patterns in three categories, and we also identified a fourth category of constraints. Constraints in ASP are generally easy to classify as they involve different predicates of the program:

1. constraints on patterns involve the `pat/2` predicate,
2. constraints on patterns embeddings involve the `occ\3` predicate,
3. constraints on transaction (the fourth type) involve the `support/1` predicate,
4. constraints on pattern sets involve the models themselves, and in most cases, can not be handle by our framework

The following subsection shows that our ASP approach enables to add constraints on individual patterns (constraints of categories 1 and 2).

But, as ASP cannot compare models with each others, the third category of constraints can not be encoded directly. In Section C.3, we transform the classical definition of the most known constraints of the third category – the condensed representations – to encode them in pure ASP. Condensed representations (maximal and closed patterns) have been widely studied due to their monotonicity property, and to their representativeness with respect to frequent patterns. Concerning more generic constraints on pattern sets, such as the extraction of skypatterns [Ugarte et al., 2015], we have proposed in [Gerbser et al., 2016] an ASP-based approach for mining sequential skypatterns using *asprin* for expressing preferences on answer sets. This work is briefly presented in Section C.4.

Sections C.2.1 and C.2.2 present encoding of mining tasks where interesting patterns are not the frequent ones but respectively that rare and emerging ones. These are new types of transaction constraints. It illustrates that the designed strategy for frequent patterns may be partly reused efficiently to design easily new mining tasks.

C.1 Constraints on patterns, embeddings and transactions

In Section A.3.2 of Chapter 2, we introduced seven types of constraints on patterns and embeddings borrowed from Pei et al. [2007]. In this subsection, we describe each of these constraints keeping their original numbering. If not stated otherwise, the base encoding is the skip-gaps strategy and line numbers refers to Listing 4.2.

C.1.1 Encoding of classical constraints in sequential pattern mining

In a first approach, constraints on patterns and on embeddings may be trivially encoded by adding integrity constraints. But these integrity constraints acts a posteriori, during the test stage, for invalidating candidate models. A more efficient method consists in introducing constraints in the generate stage, specifically in choice rules, for pruning the search space early.

Constraint 1 – item constraint. An item constraint specifies what are the particular individual or groups of items that should or should not be present in the patterns. For instance, the constraint "patterns must contain item 1 but not item 2 nor item 3" can be encoded using `must_have/1` and `cannot_have/1` predicates: `must_have(1).cannot_have(2).cannot_have(3).`

To cope with this kind of constraint, Line 8 of Listing 4.2 is modified as:

```
8 1 { pat(X,I): item(I), not cannot_have(I) } 1 :- patpos(X).
9 :- #count{ X,I : pat(X,I), must_have(I) } < 1.
```

The encoding of Line 8 modifies the choice rule to avoid the generation of known invalid patterns, i.e. patterns with forbidden items. Line 9 is a new constraint that imposes to have at least one of the required items.

Constraint 2 – length constraint. A length constraint specifies a prerequisite on pattern length. The maximal length constraint is already encoded using the program constant `maxlen` in our encodings. A new constant `minlen` is defined to encode the minimal length constraint and a new rule is added to predicate `patpos/1` to impose at least `minlen` positions in patterns instead of only one.

```
#const minlen = 2.
patpos(1).
patpos(X+1) :- patpos(X), X<=minlen.
0 { patpos(X+1) } 1 :- patpos(X), X<maxlen.
```

Constraint 3 – super-pattern constraint. A super-pattern constraint enforces the extraction of patterns that contain one or more given sub-patterns. Mandatory sub-patterns are defined by means of the new predicate `subpat(SP,P,I)` expressing that sub-pattern `SP` contains item `I` at position `P`.

Predicate `issubpat(SP)` verifies that the sub-pattern `SP` is included in the pattern. An approach similar to embedding computation may be used:

```
issubpat(SP,1,P) :- pat(P,I), subpat(SP,1,I).
issubpat(SP,Pos+1,P) :- issubpat(SP,Pos,Q), pat(P,I),
                        subpat(SP,Pos+1,I), Q<P.
issubpat(SP) :- issubpat(SP,L,_), subpatlen(SP,L).
```

`issubpat(SP)` is true if the sub-pattern `SP` is a sub-pattern of the current pattern. This predicate is used to define the final integrity constraint:

```
:- #count{ SP : issubpat(SP), subpat(SP,_,_) } = 0.
```

Constraint 4 – aggregate constraint. An aggregate constraint is a constraint on an aggregation of items in a pattern, where the aggregate function can be *sum*, *avg*, *max*, *min*, *standard deviation*, etc. The only aggregates that are provided by *clingo* are `#sum`, `#max` and `#min`. For example, let us assume that to each item `I` is assigned a cost `C`, which is given by predicate `cost(I,C)`. The following constraint enforces the selection of patterns having a total cost of at least 1000.

```
:- #sum{ C,X : cost(I,C), pat(X,I) } < 1000.
```

As an integrity constraint, this rule means that it is not possible to have a total amount lower than 1000 for pattern. It should be noted that `C` values are summed for each pair `(C,X)`. Thus, item repetitions are taken into account.

Constraint 5 – Regular expression. Such a constraint is satisfied if the pattern is an accepted regular expression as stated by the user. A regular expression can be encoded in ASP as its equivalent deterministic finite automata. Expressing such a constraint is mainly technical but feasible. We do not detail it here. expressions.

Constraint 6 – Duration constraints. The duration (or span) of some pattern is the difference between its last item timestamp and its first item timestamp. A duration constraint requires that the pattern duration should be longer or shorter than a given time period. In the database encoding introduced Section B.1, predicate `seq(T,P,I)` defines the timestamp of `I` in sequence `T` as the integer position `P` . A global constraint such as *max-span* cannot be expressed through simple local constraints on successive pattern item occurrences, as gap constraints described in the next paragraph. In fact, the predicate `occS/3` does not describe the embeddings precisely enough to express the *max-span* constraint: for some pattern embedding, there is no explicit link between its first item occurrence and its last item occurrence. The proposed solution is to add an argument to `occS/3` to denote the position of the occurrence of the first pattern item:

```
11 %pattern embeddings (skip-gaps strategy)
12 occS(T,1,P,P)       :- seq(T,P,I), pat(1,I).
13 occS(T,Pos+1,P,IP) :- occS(T,Pos,Q,IP), seq(T,P,I), pat(Pos+1,I),
14                       P-IP+1<=maxspan, P-IP+1>=minspan.
```

Constraint 7 – Gap constraints. A gap constraint specifies the maximal/minimal number of positions (or timestamp difference) between two successive itemsets in an embedding. The maximal gap constraint is anti-monotonic while the minimal gap is not anti-monotonic. Contrary to pattern constraints, embedding constraints cannot be encoded simply by integrity constraints. In fact, an integrity constraint imposes a constraint on all embeddings. If an embedding does not satisfy the constraint, the whole interpretation – i.e. the pattern – is unsatisfied.

In the following we give an encoding for the *max-gap* and *min-gap* constraints. For such local constraint, the solution consists in modifying the embedding generation (lines 11-12 in Listing 4.2) for yielding only embeddings that satisfy gap constraints:

```
11 occS(T,1,P)      :- seq(T,P,I), pat(1,I).
12 occS(T,Pos+1,P) :- seq(T,P,I), pat(Pos+1,I), occS(T,Pos,Q),
13                    P-Q-1>=mingap, P-Q-1<=maxgap.
```

This encoding assumes that the value of constants `mingap` and `maxgap` have been provided by the user (using `#const` statements).

Constraints of type 6 and 7 can be mixed by merging the two encodings of `occS` above:

```
11 occS(T,1,P,P)       :- seq(T,P,I), pat(1,I).
12 occS(T,Pos+1,P,IP) :- seq(T,P,I), pat(Pos+1,I), occS(T,Pos,Q,IP),
13                                    P-Q-1>=mingap, P-Q-1<=maxgap,
14                                    P-IP+1<=maxspan, P-IP+1>=minspan.
```

C.1.2 Experiment: Comparison of constrained frequent pattern mining with CPSM

In this section, we detail the performance on constrained pattern mining tasks. We compare our approach with CPSM-emb, which enables *max-gap* and *max-span* constraints. In this experiments we took the same setting as the experiments of Negrevergne and Guns [2015]: we add first a constraint *max-gap=2* and then we combine it with a second constraint *max-span=10*. For each setting, we compute the frequent patterns with our ASP encoding and with CPSM for the four datasets.

Figure 4.8 shows the runtime and the number of patterns for each experiment. This figure illustrates results for completed searches. A first general remark is that adding constraints to ASP encodings reduces computation times. Surprisingly for CPSM, for some thresholds the computation

with some constraints requires more time than without constraints. This is the case for example for the iPRG dataset: CPSM could not solve the mining problem within the timeout period for thresholds 769 and 384. Surprisingly, it could complete the task for lower thresholds whereas the task should be more difficult. ASP required also more time to solve the same problem instances, but it could complete them. Again, we can note that the mean sequence length impacts the performance of ASP encodings. CPSM has lower runtime on JMLR than ASP while it is the opposite on iPRG.

The curves related to the number of patterns demonstrate that the number of extracted pattern decreases when the number of constraints increases. Since we present only the results of completed solving, CPSM and ASP yield the same set of patterns.

C.2 Transaction constraints

The classical transaction constraint is the minimal frequency threshold, but we also implemented alternative transaction constraints to propose new sequential pattern mining tasks

- ▷ Rare patterns: a rare pattern is a pattern with a support lower than a threshold,
- ▷ Discriminant sequential patterns: a discriminant pattern is a pattern that occurs more in a class of sequence than in other classes.

C.2.1 Mining rare patterns[5]

This section presents encodings for mining (minimal) rare sequential patterns. Rare patterns are patterns that are not frequent. For some applications, such patterns may be more interesting. Only few works addressed the problem of mining rare itemsets [Koh and Ravana, 2016; Szathmary et al., 2010] but none addressed the mining of rare sequential pattern mining.

The encoding that are proposed in this section complement the Listing 4.3 using the fill-gaps strategy of embedding (removing the support constraint).

Rare and minimal rare sequential patterns Let $\sigma \in \mathbb{N}^+$ be a frequency threshold given by the data analyst. A sequence $\boldsymbol{s}$ is *rare* iff $supp(\boldsymbol{s}) < \sigma$. In such case, the sequence $\boldsymbol{s}$ is called a rare sequential pattern or a *rare pattern* for short.

It is noteworthy that rare patterns are strongly related to the frequent patterns, i.e. the sequence $\boldsymbol{s}$ such that $supp(\boldsymbol{s}) \geq \sigma$. The set of rare patterns is the dual of frequent patterns. A pattern that is not rare is frequent and conversely.

Moreover, the rarity constraint is monotone, i.e. for two sequences $\boldsymbol{s}$ and $\boldsymbol{s}'$, if $\boldsymbol{s} \preceq \boldsymbol{s}'$ and $\boldsymbol{s}$ is rare, then $\boldsymbol{s}'$ is rare. This property comes from the anti-monotonicity of the support measure.

As the number of rare patterns may be very high, we introduce two additional definitions: *zero-rare* patterns and *minimal rare patterns*. A *zero-rare* pattern is a rare pattern with a null support. This means that this pattern does not occur in dataset sequences. A *non-zero-rare* pattern is a rare pattern which is not zero-rare.

Finally, a sequence $\boldsymbol{s}$ is a minimal rare pattern (mRP) iff it is rare but all its proper subsequences are frequent.

$$mRP = \{\boldsymbol{s} | supp(\boldsymbol{s}) < \sigma \wedge \forall \boldsymbol{s}' \prec \boldsymbol{s}, supp(\boldsymbol{s}') \geq \sigma\} \tag{4.2}$$

Mining rare sequential patterns with ASP Intuitively, the constraint of having frequent pattern may be replace by the following constraint which reverse the inequality sense. It states that the number of supported sequences must be strictly below the given threshold `th` (i.e. not above `th`). `th` is a program constant that can be specified while the solver is launched.

```
:- #count{ S : support(S) } >= th.
```

Despite the correctness of this constraint, we propose an alternative encoding that explicitly expresses the monotonicity property and guide the solver to prune non-zero-rare patterns. This encoding introduces a `rare/1` predicate (see listing 4.4). Instead of evaluating the support of pattern $\langle p_j \rangle_{j \in [n]}$, it evaluates independently the support of each of its prefix. For all $l \in [n]$,

[5] This work has been done in collaboration with A. Samet and B. Negrevergne and has been published in 2017 in ILP conference

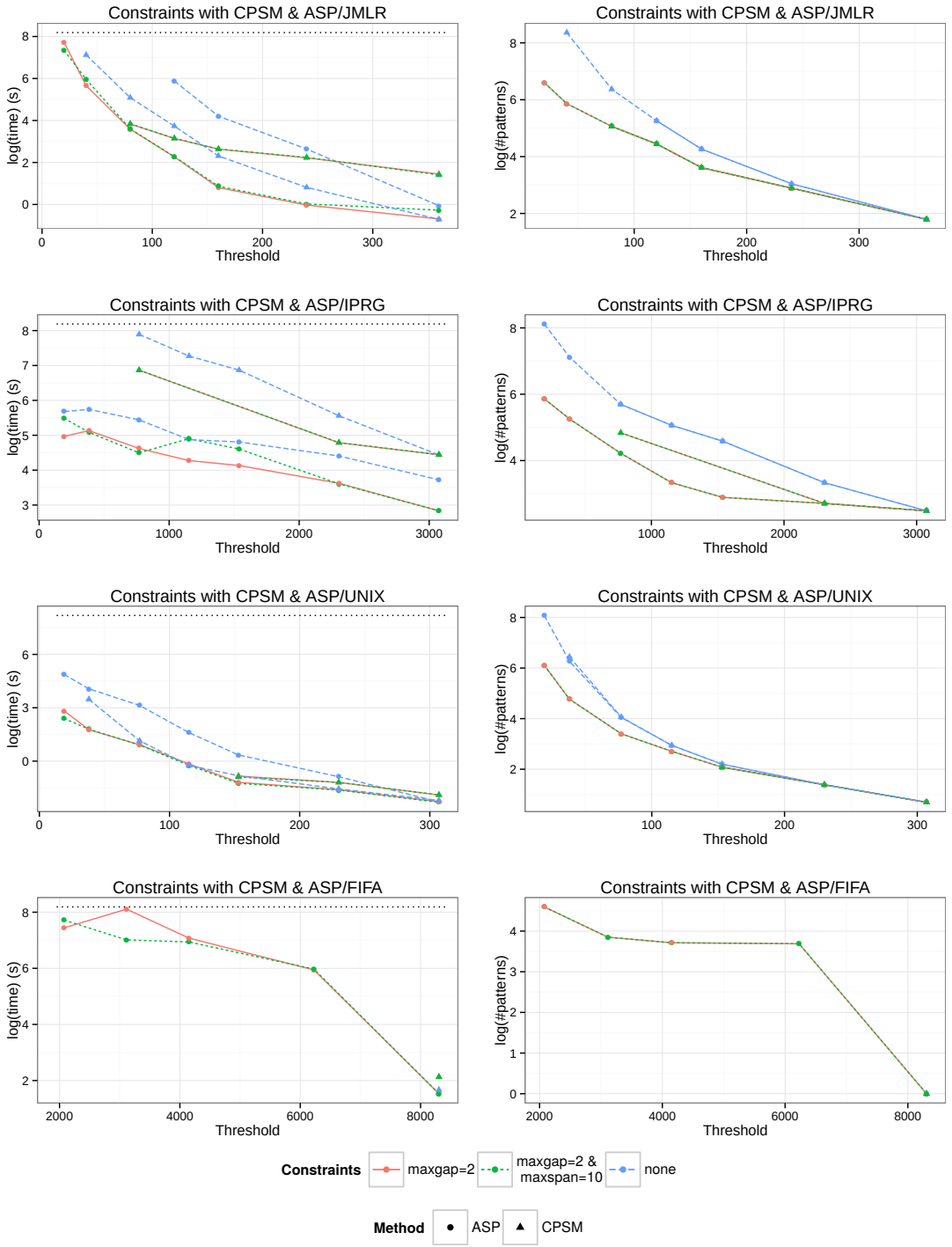


Figure 4.8: Results for constrained sequence mining tasks with ASP vs CPSM. On the left: runtime; on the right: number of patterns. The two figures are in log scales. From top to bottom, JMLR, IPRG, UNIX and FIFA. For each plot, the curves illustrate the results for different type of constraints (see legend). The horizontal dashed line figure out the 1 hour timeout.

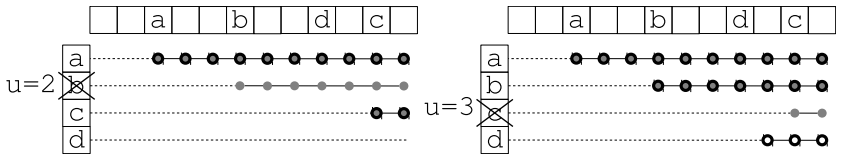


Figure 4.9: mRP occurences for pattern $\boldsymbol{p} = \langle a\ b\ c\ d \rangle$ and $\boldsymbol{u} = 3$, on the left, in case of a sequence that support $\boldsymbol{p}_u$ but not $\boldsymbol{p}$. Grey disks illustrate `occF/3` atoms and black circles illustrate `spocc/4`.

`rare(`l`)` means that the subpattern $\langle p_j \rangle_{j \in [l]}$ is rare. Line 18 imposes to have the atom `rare(`n`)` where n is the pattern length. Line 17 gives a simpler manner to yield such atom: if a prefix-pattern of length l is rare, then all patterns of length l', $l' > l$, are rare. This rule prevents the solver from evaluating all rules of line 15 and it potentially prevents from evaluating `occ/3` atoms. Finally, line 19 prunes zero-rare patterns imposing to have at least one supported sequence.

```
15 rare(IL) :- IL=1..L, patlen(L), #count{ S : occS(S, IL, LS),seqlen(S,LS) } < th.
16
17 rare(L) :- rare(L-1), L<=PL, patlen(PL).
18 :- not rare(L), patlen(L).
19 :- not support(S) : seq(S,_,_).
```

Listing 4.4: Encoding of rare sequence mining

Minimal Rare Sequence in ASP The encoding in Listing 4.5 mines minimal rare sequences. It is based on the observation that it is sufficient to evaluate the support of all sub-pattern of size $n-1$ to determine whether a pattern $\boldsymbol{p}$ of size n is a minimal. Let $\boldsymbol{p} = \langle p_i \rangle_{1 \in [n]}$ be a non-zero-rare pattern. $\boldsymbol{p}$ is a mRP iff the sub-pattern $\boldsymbol{p}_u = \langle p_i \rangle_{i \in \{1,\dots,u-1,u+1,\dots,n\}}$ is frequent (i.e. not rare) for all $u \in \{1, \dots, n\}$. According to Equation 4.2, $\boldsymbol{p}$ is a mRP iff $\forall \boldsymbol{p}' \prec \boldsymbol{p}$, $\boldsymbol{p}'$ is frequent. It is then clear that if $\boldsymbol{p}$ is a mRP, then $\boldsymbol{p}_u$ is frequent for all u. Conversely, for all sequence $\boldsymbol{s}$ such that $\boldsymbol{s} \prec \boldsymbol{p}$, there exists u such that $\boldsymbol{s} \prec \boldsymbol{p}_u$. Then, as $\boldsymbol{p}_u$ is frequent and by anti-monotonicity of the frequency constraint, $\boldsymbol{s}$ is frequent too.

Furthermore, the encoding takes advantage of two properties of sequential patterns: (1) a sequence that supports the pattern $\boldsymbol{p}$ supports each pattern $\boldsymbol{p}_u$, (2) if a sequence $\boldsymbol{s}$ is supported by $\boldsymbol{p}_u$ but not by $\boldsymbol{p}$, then for all $l \in [1, u-1]$, `occF(`$\boldsymbol{s}$`,`l`,`p_l`)` is part of the ASP model computed by Listing 4.3, where $(p_l)_{l \in [1,(n-1)]}$ is an occurrence of $\boldsymbol{p}_u$. In fact, as there is no difference between $\boldsymbol{p}_u$ and $\boldsymbol{p}$ before position u, there is no differences between their occurrences. These properties are helpful to decide whether $\boldsymbol{p}_u$ is rare. In Listing 4.5, line 20 enumerates all subpatterns, identified by `U`. Similarly to `occF/3` predicates, atoms `spocc(`$\boldsymbol{s}$`,`u`,`l`,`p_l`)` describe the occurrence of $\boldsymbol{p}_u$ in sequence $\boldsymbol{s}$ (Figure 4.9).

Lines 22 to 26 compute occurrences of pattern $\boldsymbol{p}_u$ according to the fill-gaps principle (see section B.2.2) with two differences: (1) `spocc/4` are yielded only if sequence $\boldsymbol{t}$ is not supported, otherwise the sequence is obviously supported knowing that $\boldsymbol{p}_u \prec \boldsymbol{p}$; and (2) `spocc/4` starts from pattern position u using `occF(`$\boldsymbol{t}$`,`$u-1$`,_)` available atoms to avoid redundant computation according to the second property above. Lines 28 to 33 determine whether the subpattern u is rare. This encoding differs Listing 4.4 mainly at lines 30-32. The total number of sequences supported by $\boldsymbol{p}_u$ is the number of sequences supported by $\boldsymbol{p}$ (line 32) plus the number of sequences supported by $\boldsymbol{p}_u$ but not by $\boldsymbol{p}$ (line 31), i.e. sequences for which `spocc/4` atoms reached the last pattern item. Finally, line 33 eliminates answer sets for which a subpattern is rare.

Note that minimal rare constraint is a pattern set constraint. Similarly to maximal or closed constraints, it may be encoded by using the *asprin* extension (see Section C.4).

C.2.2 Discriminant sequential patterns

In this Section, we address the discriminant pattern mining, i.e. a supervised pattern mining task. Similarly to the discriminant chronicle mining (see Section C), the main idea is to say that a pattern is interesting if its support is significantly larger in one dataset than in another [Dong and Li, 1999].

```
20 suppat(U) :- U=1..L, patlen(L), L>1.
21
22 spocc(S,1,2,P) :- seq(S,P,I), pat(2,I), not support(S).
23 spocc(S,U,U,P) :- suppat(U), occF(S,U-1,P), not support(S).
24 spocc(S,U,L  ,P+1) :- spocc(S,U,L,P), seq(S,P+1,_).
25 spocc(S,U,L+1,P+1) :- spocc(S,U,L,P), seq(S,P+1,I), pat(L+1,I).
26
27
28 sprare(U,U-1) :- sprare(U-1), suppat(U).
29 sprare(U,L)   :- sprare(U, L-1), L<=LP, patlen(LP).
30 sprare(U, IL) :- suppat(U), IL=U+1..L, patlen(L),
31                  #count{ S: spocc(S,U,IL,LS), seqlen(S,LS); S: support(S) } < th.
32
33 :- sprare(U,L), patlen(L).
```

Listing 4.5: Encoding part for minimal rare patterns

More formally, the discriminant sequential pattern (DSP) mining considers a set of databases $\mathcal{D}_1, \ldots \mathcal{D}_n$. We denote by $supp_{\mathcal{D}_i}(S)$ the support of the sequence S in the database $\mathcal{D}_i$. Let $\alpha \in [1, \infty]$ be a user defined "growth-rate threshold", an α-DSP of the database k is a sequence S such that $\forall l \neq k,\ supp_{\mathcal{D}_k}(S) > \alpha \times supp_{\mathcal{D}_l}(S)$. The database identifier can be seen as the label of a sequence in a single database of label-sequences.

For encoding DSP, we add the predicate `cl/2`. `cl(T,C)` specifies that the sequence `T` is in the database `C` (or is of class `C`). The predicate `class/1` is used to encode the list of the databases and `of_class/1` encodes the class of the current pattern. At the line 3 of the Listing 4.6, one atom of this predicate is generated.

The rules of the listing 4.6 replace the frequency constraints of the listing 4.3 (fill-gaps) or 4.2 (skip-gaps). With this program, a frequency of the pattern is computed with respect to each class. The atom `support(C,S)` indicates that there is `S` sequences supported by the pattern in the database `C`. For the pattern class, it is used as a frequency constraint rule (line 6).

```
1 %frequency constraint
2 class(C) :- cl(_,C). %set of the classes
3 1{ of_class(C) : class(C) } 1.
4
5 support(C, S) :- S={ in_support(T): cl(T,C) }, class(C).
6 :- support(C, S), S<N, threshold(N), of_class(C).
7
8 %emerging pattern with respect to the class K
9 #const alpha = 10
10 discr(K) :- support(K,KS), support(C,S), KS*(100+alpha)<S*100,
11                of_class(C), class(K), K!=C.
12 %discriminancy constraint
13 :- S=#count { K : not discr(K), class(K), K!=C}, of_class(C), S>0.
```

Listing 4.6: Discriminant sequential patterns

The lines 8 to 13 encode the constraint of an emerging pattern. As ASP manipulates only integers, the constant `alpha` encodes the value of $(\alpha - 1) \times 100$.

The rule of the lines 10-11 generate atoms of the `discr/1` predicate while a pattern is a discriminant pattern of the class `C` with respect to one class `K`. Finally, a pattern is discriminant while it is discriminant with respect to all the classes. The constraint of the line 13 eliminates patterns for which is not discriminant with respect to at least one class `K`.

C.3 Mining closed and maximal sequences

In this section, we study the encodings for two well-studied pattern types, closed and maximal patterns. A closed pattern is such that none of its frequent super-patterns has the same support.

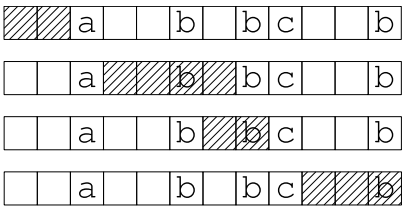


Figure 4.10: Illustration of the notion of insertable region on the example of Figure 4.2 for pattern $\langle abc \rangle$. Each line shows an insertable region, from top to bottom: insertion in the prefix, insertion between b and b, insertion in b and c, insertion in the suffix.

A maximal pattern is such that none of its super-patterns is frequent. Thus, it is necessary to compare the supports of several distinct patterns. Since a solution pattern is encoded through an answer set, a simple solution would be to put constraints on sets of answer sets. However, such a facility is not provided by basic ASP language and need the *asprin* extension of *clingo* [Brewka et al., 2015] (see section C.4 illustrating the use of *asprin* for mining skypatterns). Here, closure and maximality constraints have been encoded without any comparison of answer sets but as additional constraints on the requested patterns. The next section introduces the definitions of these alternative mining tasks and the properties that were used to transform the pattern set constraints as constraints on individual patterns. Section C.3.2 gives encodings for closed and maximal patterns extraction.

C.3.1 Definitions and properties

Basic definitions of condensed sequential patterns (maximal and closed patterns) have been given in Section A.3.1 of Chapter 2. But these original definition are not helpful to encode them in an ASP program because the ASP program put constraint on one answer set (i.e. one pattern in our framework). The definitions in Section A.3.1 of Chapter 2 make use of at least two patterns to define maximality/closure condition.

We introduce alternative maximality/closure conditions. The objective of these equivalent conditions is to define maximality/closure without comparing patterns. Such conditions can be used to encode the mining of condensed pattern representations. The main idea is to say that a sequence $\boldsymbol{s}$ is maximal (resp. closed) if and only if for every sequence $\boldsymbol{s}'$ s.t. $\boldsymbol{s}$ is a subsequence of $\boldsymbol{s}'$ with $|\boldsymbol{s}'| = |\boldsymbol{s}| + 1$, then $\boldsymbol{s}'$ is not frequent (resp. $\boldsymbol{s}'$ has not the same support as $\boldsymbol{s}$).

More precisely, a frequent pattern $\boldsymbol{s}$ is maximal iff any sequence $\boldsymbol{s}_a^j$, obtained by adding to S any item a at any position j, is non frequent. Such an a will be called an *insertable* item.

Proposition 6 (Maximality condition)**.** *A frequent sequence* $\boldsymbol{s} = \langle t_1 \ \dots \ t_n \rangle$ *is maximal iff* $\forall a \in \mathcal{I}$, $\forall j \in [0, n]$, $|\{\boldsymbol{t} \in \mathcal{D} | \boldsymbol{s} \prec \boldsymbol{t} \wedge \boldsymbol{s}_a^j \preceq \boldsymbol{t}\}| < \sigma$, *where* $\boldsymbol{s}_a^0 = \langle a, t_1, \dots, t_n \rangle$, $\boldsymbol{s}_a^j = \langle t_1, \dots, t_j, a, t_{j+1}, \dots, t_n \rangle$ *and* $\boldsymbol{s}_a^n = \langle t_1 \ \dots \ t_n, a \rangle$.

A frequent pattern S is closed iff for any frequent sequence S_a^j, obtained by adding any item a at any position j in S, any sequence T that supports S supports also S_a^j.

Proposition 7 (Closure condition)**.** *A frequent sequence* $\boldsymbol{s} = \langle t_1 \ \dots \ t_n \rangle$ *is closed iff* $\forall a \in \mathcal{I}$, $\forall j \in [0, n]$, $supp(\boldsymbol{s}_a^j) \geq \sigma \implies (\forall \boldsymbol{t} \in \mathcal{D}, \boldsymbol{s} \preceq \boldsymbol{t} \implies \boldsymbol{s}_a^j \preceq \boldsymbol{t})$, *where* $\boldsymbol{s}_a^0 = \langle a \ t_1 \ \dots \ t_n \rangle$, $\boldsymbol{s}_a^j = \langle t_1 \ \dots \ t_j \ a \ t_{j+1} \ \dots \ t_n \rangle$ *and* $\boldsymbol{s}_a^n = \langle t_1 \ \dots, t_n \ a \rangle$.

A consequence (the contraposition) of these properties is that if an item may be inserted between items of an embedding for at least f_{min} sequences (resp. for all supported sequences) then the current pattern is not maximal (resp. not closed). The main idea of our encodings is grounded on this observation.

The main difficulty is to construct the set of insertable items for each in-between position of a pattern, so-called insertable regions. Figure 4.10 illustrates the insertable regions of a sequence for the pattern $\langle a \ b \ c \rangle$.

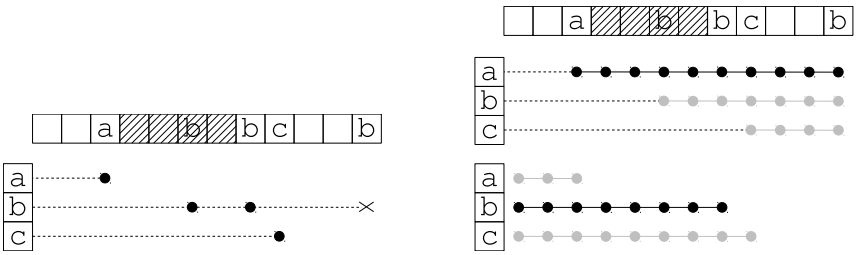


Figure 4.11: Illustration of the computation of an insertable region (hatched area) where insertable items are located between a and b and related to the first and second element of pattern $\langle abc \rangle$. On the left, valid `occS/3` atoms in the skip-gaps strategy. In the figures, the leftmost occurrences and the rightmost occurrences are the same. Concerning the fill-gaps strategy, `occF/3` and `roccF/3` atoms are illustrated on the right. Black occurrences are used to compute the hatched region.

Definition 32 (Insertable item/insertable region)**.** *Let* $\boldsymbol{p} = \langle p_i \rangle_{i \in [l]}$ *be an* l*-pattern,* $\boldsymbol{s} = \langle s_i \rangle_{i \in [n]}$ *be a sequence and* $e^j = (e_i^j)_{i \in [l]}$, $j \in [k]$ *be the* k *embeddings of* $\boldsymbol{p}$ *in* $\boldsymbol{s}$, $k > 0$. *An* insertable region $R_i = [l_i + 1, u_i - 1] \subset [n]$, $i \in [l+1]$ *is a set of positions in* $\boldsymbol{s}$ *where* $l_i \stackrel{\text{def}}{=} \min_{j \in [k]} e_{i-1}^j, i \in [2, l+1], u_i \stackrel{\text{def}}{=} \max_{j \in [k]} e_i^j, i \in [1, l])$, $l_1 \stackrel{\text{def}}{=} 0$, $u_{l+1} \stackrel{\text{def}}{=} n + 1$.

Any item $a \in s_p$, $p \in [l_i, u_i]$, $i \in [l+1]$ *is called an* insertable item *and is such that* $\boldsymbol{s}$ *supports the pattern* $\boldsymbol{p}'$ *obtained by inserting* a *in* $\boldsymbol{p}$ *at position* i *as follows:*

- ▷ $\boldsymbol{p}' = \langle a, p_1, \ldots, p_l \rangle$ *if* $i = 1$,
- ▷ $\boldsymbol{p}' = \langle p_1, \ldots, p_l, a \rangle$ *if* $i = l + 1$,
- ▷ $\boldsymbol{p}' = \langle p_1, \ldots, p_{i-1}, a, p_i, \ldots, p_l \rangle$ *otherwise.*

In the sequel, we present encodings for closed and maximal patterns which are based on the notations introduced in Definition 32. These encodings cope with the most general case of condensed patterns. It should be noted that, for efficiency reasons, most of procedural algorithms for condensed sequential pattern mining process backward-condensed patterns only. Specific ASP encodings for backward-condensed pattern mining can be found in Guyet et al. [2017c]. These encodings are known to be more efficient but are less generic.

C.3.2 Encoding maximal and closed patterns constraints

The encoding below describes how is defined the set of items that can be inserted between successive items of an embedding. These itemsets are encoded by the atoms of predicate `ins(T,X,I)` where `I` is an item which can be inserted in an embedding of the current pattern in sequence `T` between items at position `X` and `X+1` in the pattern. We give the encodings for the two strategies skip-gaps and fill-gaps: Listing 4.7 (resp. Listing 4.8) has to be added to the encoding of skip-gaps strategy (Listing 4.2), resp. fill-gaps strategy (Listing 4.3). We illustrate the way they proceed in Figure 4.11.

Listing 4.7 gives the encoding for computing insertable items using the skip-gaps strategy. This encoding is based on the idea that the insertable region i is roughly defined by the first occurrence of the $(i-1)$-th pattern item and the last occurrence of the i-th pattern item. However, not all occurrences of an item I represented by `occS/3` atoms are valid. For instance, in Figure 4.11, on the left, the last occurrence of b is not valid because it can not be used to define an occurrence of $\langle a\ b\ c \rangle$. The valid occurrences are those which have both a preceding and a following valid occurrence. Thus, this validity property is recursive. The encoding of Listing 4.7 selects two types of occurrences: the leftmost occurrences (resp. rightmost occurrences) corresponding to the earlier (resp. the later) embeddings.

Lines 19 and 23 are boundary cases. A leftmost occurrence is valid if it is the first occurrence in the sequence. Line 20 expresses that an occurrence of the X-th item is a valid leftmost occurrence if it follows a valid leftmost occurrence of the $(X-1)$-th item. Note that it is not required to

```
18 % leftmost "valid" embeddings
19 mlocc(T,1,P) :- occS(T,1,P), 0 { occS(T,1,Q): Q<P } 0, support(T).
20 mlocc(T,X,P) :- occS(T,X,P), mlocc(T,X-1,Q), Q<P, X>1, support(T).
21
22 % rightmost "valid" embeddings
23 mrocc(T,L,P) :- occS(T,L,P), 0 { occS(T,L,R): R>P } 0, patlen(L).
24 mrocc(T,X,P) :- occS(T,X,P), mrocc(T,X+1,R), R>P, X<L, patlen(L).
25
26 %insertable items
27 ins(T,1  ,I) :- seq(T,P,I), P<Q, mrocc(T,1,  Q).
28 ins(T,X  ,I) :- seq(T,P,I), P<Q, mrocc(T,X,  Q), P>R,
29                                   mlocc(T,X-1,R), X>1, patpos(X).
30 ins(T,L+1,I) :- seq(T,P,I), P>R, mlocc(T,L,  R), patlen(L).
```

Listing 4.7: Computation of insertable items – skip-gaps strategy

```
20 %embeddings in a reverse order
21 roccF(T,L,P) :- seq(T,P,I), pat(L,I), patlen(L).
22 roccF(T,L,P) :- roccF(T, L,   P+1), seq(T,P,_).
23 roccF(T,L,P) :- roccF(T, L+1, P+1), seq(T,P,C), pat(L,C).
24
25 %insertable items
26 ins(T,1  ,I) :- seq(T,P,I), roccF(T,1,  P+1).
27 ins(T,L+1,I) :- seq(T,P,I), occF(T,L,P-1), patlen(L).
28 ins(T,X,  I) :- seq(T,P,I), roccF(T,X,P+1),
29                              occF(T,X-1,P-1), patpos(X), X>1.
```

Listing 4.8: Computation of insertable items – fill-gaps strategy

compute a unique leftmost occurrence here. Line 24 does the same operation starting from the end of the sequence, precisely, the rightmost occurrence.

Lines 27-30 define insertable items. There are three cases. Lines 27 and 30 are specific boundary cases, i.e. insertion respectively in the prefix and in the suffix. The rule in lines 28-29 specifies that insertable items `I` are the items of a sequence `T` at position `P` such that `P` is strictly between a leftmost position of the `(X-1)`-th item and a rightmost position of the `X`-th item. In Figure 4.11 left, the hatched segment defines the second insertable region for pattern $\langle a\ b\ c\rangle$ (strictly between a and b).

The encoding of Listing 4.8 achieves the same task using the alternative semantics for predicate `occF/3` defining the fill-gaps strategy. As noted for the previous encoding, only the positions of the last and the first valid occurrences are required for any pattern item. It can be noted that the fill-gaps strategy provides the first valid occurrence of an item `X` as the first atom of the `occF(T,X,_)` sequence. Then, computing the last occurrence for each pattern item can be done in the same manner considering an embedding represented in reverse order . The right part of Figure 4.11 illustrates `occF/3` and `roccF/3` (reverse order) occurrences (see Listing 4.8, lines 21-23). We can notice that the hatched insertable region is the intersection of occurrences related to a and reverse occurrences related to b, after having removed intersection bounds.

The computation of insertable items, Listing 4.8 lines 26-29, exploits the above remark. Line 26 defines the insertable region in a prefix using `roccF(T,1,P)`. Since items are insertable if they are strictly before the first position, we consider the value of `roccF(T,1,P+1)`. Line 27 uses `occF(T,L,P)` to identifies the suffix region. Lines 28-29 combine both constraints for in-between cases.

We can now define the (integrity) constraints for closed and maximal patterns. These constraints are the same for the two embedding strategies.

To extract only maximal patterns, the following constraint denies patterns for which it is possible to insert an item which will be frequent within sequences that support the current pattern.

```
:- item(I), X = 1..maxlen+1, { ins(T,X,I) : support(T) } >= th.
```

The following constraint concerns the extraction of closed-patterns. It specifies that for each insertion position (from 1, in the prefix, to `maxlen`+1, in the suffix), it not possible to have a frequent insertable item `I` for each supported transaction.

```
:- item(I), X = 1..maxlen+1, { ins(T,X,I) } >=th, ins(T,X,I) : support(T).
```

Though interesting from a theoretical point of view, these encodings leads to more complex programs and should be more difficult to ground and to solve, especially the encoding in Listing 4.7. Backward-closure/maximality constraints are more realistic from a practical point of view.

Finally, it is important to notice that condensed constraints have to be carefully combined with other patterns/embedding constraints. As noted by Negrevergne et al. [2013], in such cases the problem is not clearly specified. For instance, with our database of Example 23, extracting closed patterns amongst the patterns of length at most 2 will not yield the same results as extracting closed patterns of length at most 2. In the first case, $\langle b\ c \rangle$ is closed because there is no extended pattern (of length at most 2) with the same support. In the second case, this pattern is not closed, even if its length is at most 2.

C.3.3 Experiments: analysis of condensed pattern extraction

Figure 4.12 illustrates the results for condensed pattern mining. This approach cannot be compared to CPSM since it does not propose means for encoding such kind of patterns.

This experiment compares the resource requirements in time and memory for mining closed/-maximal and backward-closed/maximal patterns. For each of these mining task, we compared the skip-gaps and fill-gaps strategies. The main encoding is still based on prefix-projection. Three real datasets have been processed (JMLR, UNIX and IPRG). The FIFA dataset was not processed due to its heavy memory requirement for some of these tasks.

We can first note that the difference between the number of extracted patterns is low. As expected, all encodings that complete a given mining task extract the same number of patterns. This result supports the correctness of our approach. From the memory point of view, we see that the encodings extracting condensed patterns requires several order of magnitude more memory, especially for (backward-)closed patterns. It is also interesting to note that the memory requirement for the fill-gaps strategy is not linked to the threshold, contrary to the skip-gaps strategy. Again, the fill-gaps strategy seems to be more convenient for small thresholds. We can note that there is a big difference between datasets concerning runtime. For instance, frequent patterns are faster to extract for JMLR and UNIX, but maximal patterns are faster to compute on IPRG. The density of this last dataset makes maximal pattern extraction easier. Uniformly, we can conclude that fill-gaps is faster than skip-gaps. The complexity of the encoding related to insertable items with skip-gaps makes the problem difficult to solve. Opposed to the experiments presented in Guyet et al. [2017c], we did not use any solving heuristic. For maximal patterns, a huge improvement of runtime was observed when using the *subset-minimal* heuristic[6].

C.4 Encoding pattern set constraints with preferences

We have demonstrated above how easily a basic encoding of frequent sequence mining can be extended in order to refine the patterns of interest with constraints. In the previous section, we have seen that patterns, embedding and transaction constraints can easily be included in a program, but pattern set constraints (like closed or maximal patterns) required a specific encoding effort.

To better fit the ideal principle of declarative encoding we need to propose a framework to encode pattern set constraints. We adopt the framework proposed by Ugarte et al. [2015] which proposes to encode pattern set constraints as preferences on the patterns. An interesting set of

[6]The use of subset-minimal heuristic keeps solving the maximal patterns problem complete.

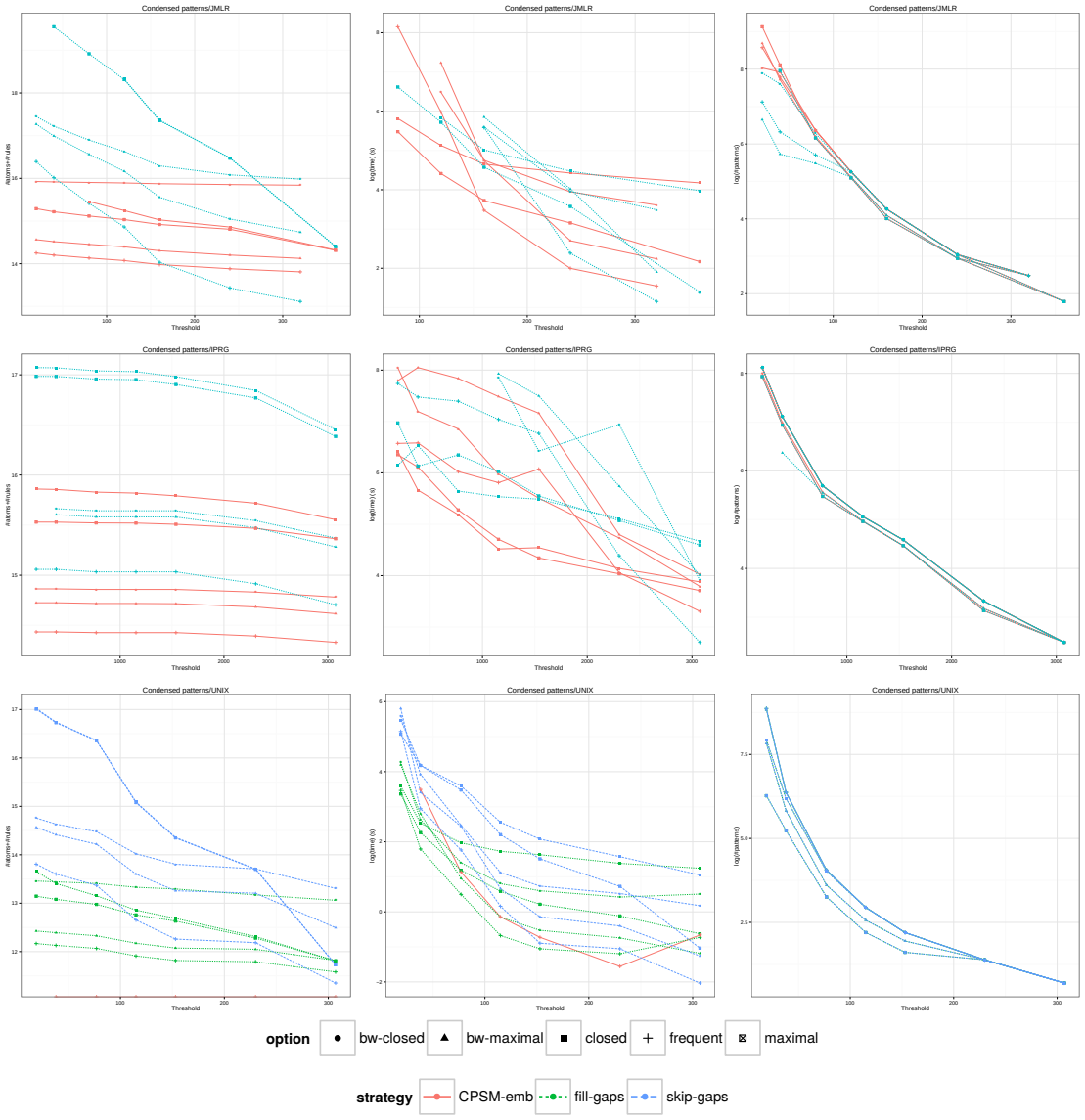


Figure 4.12: From left to right, problem size, runtime and number of extracted patterns with respect to the frequency threshold. Runtimes are shown only if the solving was complete, contrary to pattern numbers which show the number of extracted patterns within the timeout period. From top to bottom, JMLR, IPRG and UNIX. For each plot, the curves illustrate the results for different types of condensed patterns (see legend) and for the two embedding strategies (*fill-gaps* in red-plain line, *skip-gaps* in blue-dashed line).

```
1 cont(E,F)   :- item(E), F = #count{ T : seq(T,P,E) }.
2 freq(1..M)  :- M = #max{ F : cont(E,F) }.
3
4 hasfreq(F)    :- cont(E,F), pat(X,E).
5 hasfreq(F-1)  :- hasfreq(F), 1<F.
6 better(P) :- preference(P,aconf),
7              #sum{ 1,F,T : holds'(hasfreq(F)),
8                            holds (cover(T));
9                   -1,F,T : holds (hasfreq(F)),
10                           holds'(cover(T)) } > 0.
11 bettereq(P) :- preference(P,aconf),
```

Listing 4.9: Preference type implementation

patterns is a set of the preferred patterns: if a pattern is dominated by another preferred pattern, it is not included to the set of patterns.

This can be taken one step further by exploiting ASP's preference handling capacities to extract preferred patterns. The idea is to map a preference on patterns to one among answer sets (cf. Section A.3), which is accomplished in two steps with the *asprin* system. At first, we have to provide implementations of pattern-specific preference types that are plugged into *asprin*. Once done, the defined preference types can be combined freely with those available in *asprin*'s library to form complex preferences among the patterns given by answer sets.

Let us illustrate our approach by modeling the extraction of skypatterns [Ugarte et al., 2015], which follow the Pareto principle in combining several quality measures. To illustrate skypatterns, we consider the exemplary pattern-oriented preference types *area* and *aconf*, measuring the product of a pattern's support and length or the ratio of support and the highest frequency of its contained items, respectively. More precisely, given a database $\mathcal{D}$ and a (frequent) pattern $\boldsymbol{s} = \langle s_i \rangle_{1 \leq i \leq m}$, the associated *area* is $m \times \mathit{support}(\boldsymbol{s}, \mathcal{D})$, and the *aconf* measure is determined by

$$\frac{\mathit{support}(\boldsymbol{s}, \mathcal{D})}{\max_{1 \leq i \leq m} |\{\boldsymbol{t} \in \mathcal{D} \mid \langle s_i \rangle \preceq \boldsymbol{t}\}|} \,, \tag{4.3}$$

where greater values are preferred in both cases. We further consider *Pareto efficiency* to aggregate the two measures. That is, a pattern is preferred over another one if it is at least as good as the other in all criteria, and strictly better in some criterion.

While the *area* preference can be directly mapped to *asprin*'s built-in `cardinality` type (see below), a dedicated implementation of the *aconf* measure is given in Listing 4.9. To begin with, the auxiliary rules from Line 1 to 3 provide atoms of the form `cont(`e`,`f`)` and `freq(`f`)`. The former express that f sequences in $\mathcal{D}$ contain the item e, and the latter indicate that some item belongs to f or more sequences. Moreover, the rules in Line 5 and 6 derive atoms `hasfreq(1..`f`)` for items e in a candidate pattern, so that the number of such atoms corresponds to the highest frequency.

As described in Section A.3, it remains to define an atom `better(p)` to indicate that an answer set X, reified in terms of `holds(a)` atoms, is preferred over (a previous) one, Y, given by `holds'(a)`, according to a preference `p` of type *aconf*. This is accomplished by the rule from Line 8 to 12, whose precondition of the form `preference(p,aconf)` checks the type of preference `p`. The other literal is a `#sum` aggregate that casts ratios (4.3) to corresponding products via multiplication of the denominators. That is, each sequence $\boldsymbol{t}$ in $\mathcal{D}$ including the pattern comprised in answer set X contributes f' times the addend `1`, where f' is the highest frequency of items in the pattern given by Y. In turn, each sequence $\boldsymbol{t}$ including the pattern in Y amounts to f instances of the addend `-1` when f is the highest frequency for X. As a consequence, the `#sum` aggregate evaluates to a positive integer iff the ratio (4.3) obtained with the pattern in X is greater than the one for Y, which is then signaled by `better(p)`. With the rules in Listing 4.9 at hand, the preference type denoted by `aconf` is ready for use in preference declarations.[7]

[7]For brevity, Listing 4.9 does not show an analogous rule replacing "`> 0`" by "`>= 0`" for defining `bettereq(p)`,

```
1 #preference(p1,more(cardinality)){ pos(X) & cover(T) : X = 1..max, seq(T,P,E) }.
2 #preference(p2,aconf){ hasfreq(F) : freq(F); cover(T) : seq(T,1,E) }.
3 #preference(p3,pareto){ name(p1); name(p2) }.
4
5 #optimize(p3).
```

Listing 4.10: Preference declaration

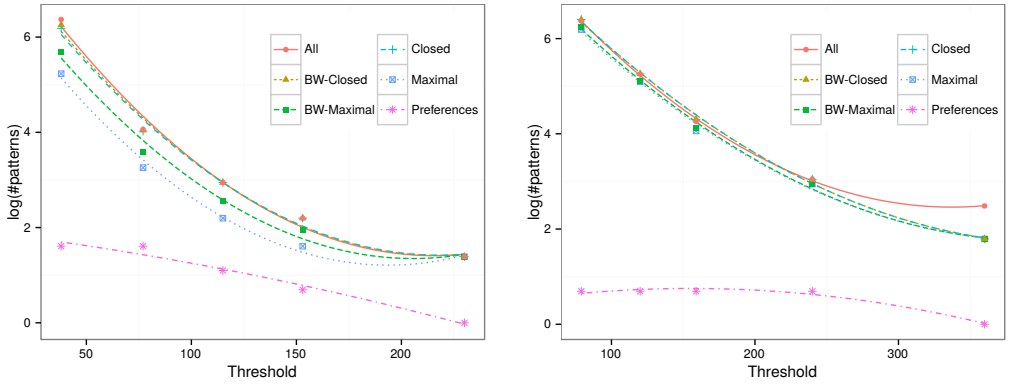


Figure 4.13: Numbers of frequent patterns wrt different thresholds. Top: `Unix user` database; bottom: `jmlr` database.

Listing 4.10 shows an *asprin* declaration of the preference relations specified above. In Line 1 and 2, the *area* preference is mapped to the built-in `cardinality` type, used to count conjunctions (`&`) of item positions in a pattern and sequences including it. The preference denoted by `p1` thus aims at maximizing $m \times \mathit{support}(\boldsymbol{s}, \mathcal{D})$. The second preference `p2` in Line 3 and 4 refers to the `aconf` type, whose implementation as above addresses the maximization of the ratio (4.3). In Line 5, both preference relations, referenced via dedicated atoms `name(p1)` and `name(p2)`, are aggregated by means of the built-in preference type `pareto`. This yields the nested preference `p3`, made subject to optimization by *asprin* in Line 7. For the database of Example 23, answer sets comprising the (frequent) patterns $\langle c \rangle$, $\langle a\ c \rangle$, and $\langle a\ b\ c \rangle$ turn out to be optimal in terms of Pareto efficiency. Their associated *area* and *aconf* measures are 6, 10, and 12 or 1, 5/6, and 2/3, respectively. While neither of the three patterns is preferred over another, they dominate the remaining patterns $\langle a \rangle$, $\langle b \rangle$, $\langle a\ b \rangle$, and $\langle b\ c \rangle$. Of course, other customized preference types and different combinations thereof can be conceived as well, eg. using built-in types like `lexico`, `and`, and `or` (with their literal meanings).

C.4.1 Experiments: Pattern set reduction

This experiment aims at comparing different pattern mining tasks: skypatterns (with *area* and *aconf* measures) as well as (BW/$\sqsubseteq_b$) closed and maximal frequent patterns. The objectives are to evaluate the effectiveness of our encodings on real databases and to illustrate how focusing on skypatterns drastically reduces the number of extracted patterns. We ran our respective encodings on two databases of the UCI collection `jmlr` and `Unix user`.

Figure 4.13 displays numbers of frequent patterns obtained wrt different thresholds. For the basic as well as condensed frequent pattern mining tasks, we see that the lower the threshold, the greater the number of patterns. Despite the exponential growth (or decrease, respectively) of candidate pattern sets, the number of skypatterns remains very low (at most five patterns).

In addition, we observed that the times for computing skypatterns are comparable to those for computing all frequent patterns. In fact, the number of models to compare remained quite low

used additionally for aggregation via Pareto efficiency.

although the search space is the same. Preferences rely upon optimization, but considering that we extract all frequent patterns, it is quite similar to extract only the best ones.

D Case study: Illustration of versatility at work[8]

In this Section, we continue exploring solution to our case-study. We remind that we want to analyze possible relationship between anti-epileptic drug exposition and seizure.

Such medical events have a low incidence, even in our database made for highlighting them by selecting 8,000 patients having epileptic seizures. Extracting using frequent sequential patterns mining requires to set up a low threshold. The consequence of this set up is a possible flood of patterns that makes the output useless for epidemiologists.

The regular frequent sequential pattern mining task is not necessarily the best set up to extract interesting sequential patterns. The proposed framework enables epidemiologists to investigate a wide range of sequential pattern mining tasks.

- ▷ various manner to yield items forming sequences,
- ▷ various types of mining tasks, e.g. frequent, rare, closed or emerging patterns, and possibly new ones,
- ▷ various constraints on patterns.

This section aims at showing the interest of the versatility of our ASP based pattern mining approach and its efficiency to extract interesting sequential patterns from care pathways using expert knowledge.

More especially, we believe that the definition of interesting patterns have to exploit in-depth the semantic complexity of care pathway events. This means to take into account both external knowledge (e.g. drugs and diagnosis taxonomies) and complex care event descriptions (e.g. units number, strength per unit, etc.). By this mean, we expect to extract less but more significant patterns for epidemiologists.

All constraints and tasks we presented in the previous sections of this chapters are agnostic. They are independent from to data source and may be meaningless for domain experts. Using knowledge expert in ASP program enables users to express constraints using the semantic of the original data.

Using expert knowledge, and more especially ontologies, to enhance the semantic of data mining is studied by [Dou et al., 2015]. They state that ontologies are introduced with the objectives 1) to bridge the semantic gap between the data, applications, data mining algorithms, and data mining results, 2) to provide data mining algorithms with a priori knowledge which either guides the mining process or reduces/constrains the search space and 3) to provide a formal way for representing the data mining flow, from data preprocessing to mining results. Definitively, we share the first two objectives. On the one hand, the use of expert knowledge aims at bridging the semantic gap between the raw medical data and the representation of sequences manageable by sequential pattern mining tools. On the second hand, constraints have to ease solving.

Figure 4.14 illustrates the general idea of our framework. Upper blue cylinders represent data and patterns in the semantic world of the application. The lower green cylinders represent data and patterns in the (simple) semantic world of sequential pattern mining. The knowledge base is used to translate the data or patterns from one semantic world to the other.

This framework can be denoted by "knowledge mediated pattern mining" referring to the notion of ontology-mediated query answering. Ontology-mediated query answering [Artale et al., 2017; Bienvenu, 2016] is a paradigm in data management that seeks to exploit the semantic knowledge expressed in ontologies to improve query answering over data. Our framework seeks to exploit knowledge to improve data mining by using formalized knowledge.

We illustrate this principle on our case study. It illustrates the approach effectiveness and it shows possible blending of reasoning techniques (use of taxonomies) with mining. To the best of our knowledge, this is the first concrete example of applied declarative pattern mining that show the ability of such class of method to effectively use formal

[8]This work has been published in AIME [Guyet et al., 2017a].

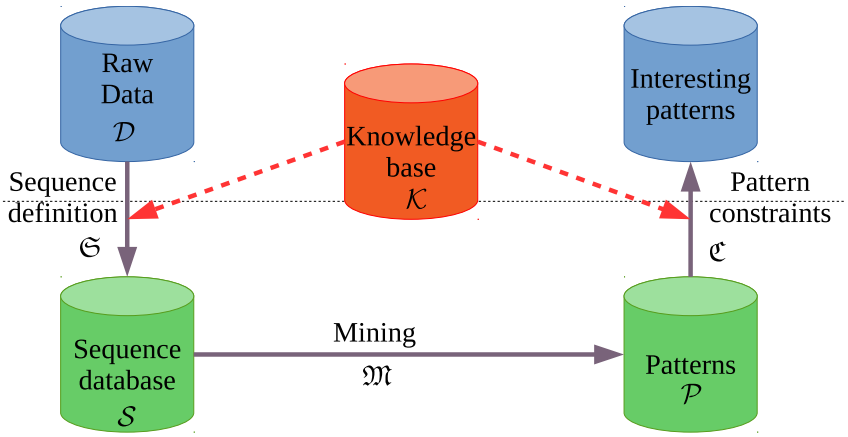


Figure 4.14: Semantic pattern mining.

D.1 Raw data and knowledge base

The first step is to encode the health care pathway data in ASP facts. This has been done for the digital cohort of 8,379 patients with a stable treatment for epilepsy (stability criterion detailed in [Polard et al., 2015] have been used). It represented 1,810,600 deliveries of 7,693 different drugs and 20,686 seizure-related hospitalizations. This information and the background knowledge (ATC[9] drugs taxonomy, ICD-10[10] taxonomy) are encoded as ASP facts.

For each patient $\boldsymbol{p}$, `sex(`$\boldsymbol{p}$`,`s`)` gives the patient sex and `birthYear(`$\boldsymbol{p}$`,`b`)` her/his birth year. Drug deliveries are encoded with `deliveries(`$\boldsymbol{p}$`,`t`,`d`,`q`)` atoms meaning that patient $\boldsymbol{p}$ got q deliveries of drug d at date t. Dates are day numbers. We use french CIP, Presentation Identifying Code, as drug identifier. The knowledge base links the CIP to the ATC and other related informations (e.g. speciality group, strength per unit, number of units or volume, generic/brand-named status, etc). Each diagnosis related to an hospital stay is encoded with `disease(`$\boldsymbol{p}$`,`t`,`d`)` meaning that patient $\boldsymbol{p}$ have been diagnosed with d at date t. The date is the hospital stay entrance and the diagnosis is an ICD code. Several disease may be diagnosed during the same stay. All of them are encoded at the same date. For our patient cohort, data $\mathcal{D}$ and related knowledge base $\mathcal{K}$ represent a total 2,010,556 facts.

The following set of facts illustrates the encoding of one patient, with identifier 0, male, born in 1960, who has been diagnosed with ICD G409 (epileptic seizure) and K700 (Alcoholic fatty liver) at the same stay.

```
sex(0,m). birthYear(0,1960).
delivery(0,1,3585053,1).     % Tramadol (CIP 3585053) delivery
delivery(0,1,3599730,1).     % Formoterol, asthma treatment
delivery(0,154,3599730,1).   % Formoterol
delivery(0,346,3599730,1).   % Formoterol
delivery(0,350,3599730,2).   % 2 boxes of Formoterol
disease(0,380,g409). disease(0,380,k700).
```

It is worth noticing that the above encoding is a direct encoding of relational tables at hand (SNIIRAM database). In the SNIIRAM database, a table contains information about the insured people, another table contains drugs deliveries and a third table contains hospitalisation events including reasons encoded with ICD code. There are many other tables representing specific events and each can be encoded by a dedicated predicate.

The following facts are an extract of the knowledge base related to facts of the previous patient events. The CIP code (identifier of a drugbox in France) gives details about the drug: its active

[9] ATC (Anatomical Therapeutic Chemical) is a taxonomy of active substances
[10] ICD-10: International Classification of Diseases, 10th revision

molecule represented by an instance of the ATC taxonomy, a speciality group, a generic vs brand-name status, and quantity information (number of pills and dosage). In addition, we encoded ATC and ICD taxonomies with `is_a/2` predicate. $R06AE09$ (tramadol molecule) is a piperazine derivatives ($R06AE$) which is a anthihistamines for systemic use ($R06A$) and so on. The $G40.9$ ICD code represents epilepsy diseases which are episodic and paroxysmal disorders ($G40 - G47$) which are classified in Chapter 6 of ICD that cover diseases of the nervous system.

```
%Details about CIP 3585053 (Tramadol)
cip_atc(3585053,r06ae09).    % ATC class of Tramadol: R09AE09
grs(3585053,364).            % Speciality group identifier (GRS)
generic(3585053).            % This CIP is a generic GRS
nbpills(3585053,28).         % 28 doses per blister
dosage(3585053,5).           % 5mg per dose

%ATC Taxonomy for R06AE09 ATC code
is_a(r06ae09, r06ae). is_a(r06ae, r06a). is_a(r06a, r06).

%ICD Taxonomy (G409 branch)
is_a(g409, g40). is_a(g40, g40g47). is_a(g40g47, chap6).
```

It is worth noticing that facts representing care pathway are the raw data of the medico-administrative database. The procedural sequential pattern mining approaches require a semantic transformation of the raw data to fit the for of a sequence of itemsets. Here, we only have a syntactic transformation of the database.

D.2 Declarative care pathway mining

The declarative pattern mining approach benefits from the versatility of logic programs. Different logic programs can easily and flexibly be combined to tackle different care pathway mining tasks.

Using notations introduced in Figure 4.14, the declarative care pathway mining is a tuple of ASP rule sets $\langle \mathcal{D}, \mathrm{S}, \mathcal{K}, \mathrm{M}, \mathrm{C} \rangle$ where is $\mathcal{D}$ is a raw dataset and $\mathcal{K}$ a knowledge base (see section D.1); M is the encoding of (frequent) sequence mining and C is a set of constraints. Finally, S is a set of rules yielding the sequences database: $\mathrm{S} \cup \mathcal{D} \cup \mathcal{K} \models \mathcal{S}$.

Depending on the study the user would like to perform, she/he has to provide S, a set of rules that specifies which are the events of interest and C, a set of constraints that specifies the patterns the user would like. It is noteworthy that the use of knowledge in constraints increases the expressiveness compare to the generic sequential pattern constraints.

In the following of this section, we give examples for S and C that can be used to define complex mining queries applied to our pharmaco-epidemiological study.

Defining sequences to mine with S. The following rules examples yield atoms `seq(`s`,` t`,` e`)` defining the sequence database to mine from the raw dataset $\mathcal{D}$. Listing 4.11 illustrates the sequences generation, only for males, that holds two types of events: anti-epileptic drugs (AED) deliveries and diagnosis events. In this listing, `aed(`i`,`c`)` lists the CIP code i, which are related to one of the ATC codes for AED ($N03AX09$, $N03AX14$, etc.). ASP enables to use a reified model of sequence, similar to those used in event calculus Kim et al. [2009] where events are functional literals: `delivery(X)` or `disease(D)`. `seq(P,T,delivery(X))` designates that patient `P` was delivered with drug `X` at time `T`.

```
aed(CIP,AED) :- cip_atc(CIP,AED),AED=(n03ax09;n03ax14;n03ax11;n03ag01;n03af01).
seq(P,T,delivery(CIP)) :- delivery(P,T,CIP,Q), sex(P,male), aed(CIP,AED).
seq(P,T,disease(D)) :- disease(P,T,D), sex(P,male).
```

Listing 4.11: Example of sequence database generation

Listing 4.12 illustrates the sequence generation of deliveries of anti-epileptic drug specialities within the 3 months before the first seizure event. It illustrates the use of the knowledge base to express complex sequences generation. In this listing, `aed(`i`,`c`)` are used to identify AED, and `firstseizure(`$\boldsymbol{p}$`,`t`)` is the date, t, of the first seizure of patient $\boldsymbol{p}$. A seizure event is a disease

event with one of the G40-G41 ICD code (see ICD-10 taxonomy). The first seizure is the one without any other seizure event before. In this sequence generation, sequence events are encoded with functional literals `delivery(AED,Gr,G)` where `AED` is an ATC code, `Gr` is drug speciality and `G` indicates whether the speciality is a generic drug or a brand-named one.

```
firstseizure(P,T) :- disease(P,T,D), is_a(D,g40;g41),
                     #count{Tp: disease(P,Tp,Dp), is_a(Dp,g40;g41), Tp<T}=0.

seq(P,T,delivery(AED,Gr,1)) :- delivery(P,T,CIP,Q), aed(CIP,AED), grs(CIP,Gr),
                               generic(CIP), T<Ts, T>Ts-90, firstseizure(P,Ts).

seq(P,T,delivery(AED,Gr,0)) :- delivery(P,T,CIP,Q), aed(CIP,AED), grs(CIP,Gr),
                               not generic(CIP), T<Ts, T>Ts-90, firstseizure(P,Ts).
```

Listing 4.12: Sequence with AED drugs within the 3 months before the first seizure.

Defining constraints on patterns. On the other side of our framework, C enables to add constraints on patterns the clinician looks for. As introduced in Section C.1, constraints are organized in pattern, embedding, transaction and pattern set constraints. Line 2 is a constraint for sequences generated by Listing 4.11 that constraint the mining program to extract patterns ending with a disease related to epileptic events. Constraints of lines 3 and 4 specify patterns that must have at least a generic and a brand-name drug in it. These constraints illustrate the use of knowledge and the expressive power of using reified events.

The transaction constraint of lines 7-9 specifies that we are interested in patterns that occur at least twice more for males than for females. `#count{ P : support(P),sex(P,0) }` enumerates male patients that support a pattern.

Finally, we illustrate the encoding of a *maxgap* constraint only for similar events. Here, the objective is to constraint the occurrence of the pattern event to be consecutive in time. This enables the clinician to interpret a pattern of consecutive similar drugs as a drug exposure. The temporal constraint as been set to 35 days because, in the french health care system, medics are delivered every months.

```
1 % Pattern constraints
2 :- patlen(L), not pat(L,disease(D)), is_a(D,g40;g41). % ending with seizure
3 :- { pat(X,delivery(AED,GRS,1)) }=0. % has generic drugs
4 :- { pat(X,delivery(AED,GRS,0)) }=0. % has not generic drugs
5 :- #sum{ Dose : pat(X,delivery(X)), nbdoses(X,Dose) }.
6
7 % Transaction constraints
8 :- M=#count{ P : support(P), sex(P,0) },
9    F=#count{ P : support(P), sex(P,1) }, M>2*F.
10
11 % Embedding constraints
12 #const maxgap=35.
13 :- occS(S,X,T), occS(S,X+1,Tp), Tp<T+maxgap, pat(X,D), pat(X+1,D).
```

Listing 4.13: Mining constraints.

These constraints illustrate some complex mining tasks that would be difficult to manage with classical sequential pattern mining algorithms. Constraints specification is very expressive thanks to the ability to use a knowledge base.

D.3 Example of query refinement

In this section, we present the results we obtained with various mining queries on the dataset presented in section D.1. Our objective is to make a pattern mining task to get insights about possible association between hospitalization for seizure and antiepileptic drug (AED) switch, i.e. a change in the AED treatments. A change to investigate is for instance between a generic drug to a brand-name drug.

In a first experiment, we extracted regular frequent sequential patterns of AED deliveries. The sequence database has been generated using listing 4.11 without the sex constraint. We used some of the previously presented constraints to specify the expected patterns: a frequency above $f_{min} = 50$, and a length between 3 and 5 items and their last event must be an epileptic seizure (cf. line 2 of listing 4.13). With this setting, 4,359 patterns are extracted in $387s$. This pattern number is obviously too large to be inspected manually. The principle of our framework is to refine constraints to get less but more interpretable patterns.

In a second experiment, we choose to use listing 4.12 to generate sequences preceding a seizure and to extract frequent patterns in this database while keeping same parameters for patterns and transaction constraints. In such case, less patterns are extracted, 3,888 patterns, but this number remains huge. Thus, we decided to refine the mining query by adding some constraints on the pattern. In fact, clinicians are interested in treatment switches, and more especially in generic/brand-name switches. By adding constraints of lines 3-4 of listing 4.13, we drastically reduce the number of extracted patterns (1,421 patterns extracted in $309s$), and such patterns are more interesting for clinicians.

D.4 Implementing a new mining task

The framework flexibility has been used to create a new mining query dedicated to the original question, i.e. a case-crossover study Polard et al. [2015]. For each patient, the S rules generate two sequences made of deliveries within respectively the 3 months before the first seizure (positive sequence) and the 3 to 6 months before the first seizure (negative sequence). In this setting the patient serves as its own control.

The mining query consists in extracting frequent sequential patterns where a patient is supported by the pattern iff the pattern appears in its positive sequence, but not in its negative sequence. A frequency threshold for this pattern is set up to 20 and we also constraint patterns 1) to have generic and brand-name deliveries and 2) to have exactly one switch from a generic to a brand-name AED (or the reverse). The solver extracts respectively 32 patterns and 21 patterns. With such very constrained problem, the solver is very efficient and extracts all patterns in less than 30 seconds. The two following patterns are representative of our results:

- ▷ $\langle(N03AG01, 438, 1), (N03AG01, 438, 1), (N03AX14, 1023, 0), (N03AX14, 1023, 0)\rangle$ is a sequence of deliveries showing a change of treatment from a generic drug of the speciality 438 of valproic acid to the brand-name speciality 1023 of levetiracetam.
- ▷ $\langle(N03AG01, 438, 1), (N03AG01, 438, 0), (N03AG01, 438, 0), (N03AG01, 438, 0)\rangle$ is a sequence of deliveries of the same speciality, 438, of valproic acid. The patient only switches from generic to brand-names AED.

According to our mining query, we found more than 20 patients which have this care sequence within the 3 months before a seizure, but not in the 3 previous months preceding this period. These new hypothesis of care-sequences are good candidates for further investigations and possible recommendation about AE treatments.

D.5 Conclusion

This section presented the principles of declarative sequential pattern mining with ASP and we shown that it is an interesting framework to flexibly design care-pathway mining queries that supports knowledge reasoning (taxonomy and temporal reasoning). This framework enables the user to define a knowledge-mediated pattern mining tasks in ASP rules, on the one side, the sequential database generation from raw data and, on the other side, the constraints on expected patterns.

We are aware that it would be possible to answer more efficiently some of these mining queries using mining algorithms by creating *ad hoc* datasets (except for cumulative dose constraint) and a constrained pattern mining algorithm. But, we strongly believe that our integrated and flexible framework empowers epidemiologists to quickly evaluate various pattern constraints and that it limits tedious pre-processing phases.

On the other hand, our framework is expressive enough to answer new mining tasks. We illustrated this by encoding a totally new mining query inspired from case-crossover studies. Numerous

new mining queries may be encoded using our framework to answer complex care-pathway analysis. Such approach may efficiently support epidemiologist in their analysis of complex care-pathways.

E Conclusion and perspectives

This section has presented a declarative approach of sequential pattern mining based on answer set programming. Generally speaking, we have demonstrated that the modeling capacities of ASP are suitable to express complex sequence mining tasks, where choice rules define the search space, integrity constraints eliminate invalid candidate patterns, and preferences distinguish optimal outcomes. We have also illustrated how to encode a broad range of mining tasks (including condensed representations and constrained patterns) in pure ASP. Thus, we have shown the first advantage of declarative pattern mining: for most well-specified tasks, the development effort is significantly lower than for procedural approaches. The integration of new constraints within our framework requires only few lines of code. This was made possible thanks to the flexibility of both ASP language and solvers.

Definitively, it shows the expressive power of ASP and its elegant manner to specify new problems. ASP enables to encode new patterns mining tasks in only few lines to add upon the generic encoding of sequential pattern mining. Beyond the presented encodings, our ASP framework also enables to capture the notions presented in the previous chapter (discriminant patterns, negative patterns and chronicles).

This work also contribute to better structure the constraints that can be expressed on sequential pattern mining. We defined four types of constraints: embedding, pattern, transaction and pattern set constraints. All along this chapter we have shown that a wide range of constraints can be defined and that makes sense to use declarative approaches mining to address sequential pattern mining.

A short term perspective is to **embed our large collection of constraint encodings in a sequential pattern mining toolbox** that would have ASP in backend. Ideally, the user will set up a mining task in few clicks by combining different constraints. This would enable users to quickly prototype the constraints that are the most desirable for their purpose.

But, the reader has to keep in mind that the presentation of these encodings may be misleading. The apparently simple encodings is the tip of the iceberg. The immersed part of encoding development is made of many attempts and of deep understanding of the solver mechanisms.

Another objective of this section was to give the intuition to the reader that while encoding a straightforward solution to a problem can be easy in ASP, writing efficient programs may be complex. This can be named the **curse of ASP: encoding problems is intuitive ... until it has to be efficient.** We have first seen encoding of the classical frequent sequential pattern mining in 13 lines, but the most efficient encodings are based on projected databases. This program is longer and not intuitive. In the advanced tasks encoding, the complex encodings we presented have simpler encodings (e.g. rare patterns, see Section C.2.1) but only advanced encodings enable to achieve better computation performances (or at least prevent from untractable resolutions).

In view of the data-intense domain at hand, some efforts had to be taken to develop efficient encodings of generic sequence mining tasks. **The drawback of the versatility of the approach is the poor computational performances compare to dedicated procedural algorithms, especially in memory requirement.** ASP and the *clingo* solver are currently not a reasonable approach for large dataset. Nonetheless, we demonstrated that our approach based on ASP competes with the other declarative approaches (SAT and CP) since they keep there declarative dimension.

We have presented several improvements of basic sequential pattern mining and two alternatives for encoding the main complex task, i.e. computing embeddings. These encodings have been extensively evaluated on synthetic and real datasets to draw conclusions about the overall efficiency of this approach (especially compared to the constraint programming approach CPSM) and about which are the best encodings among the proposed ones and in which context.

The first conclusion of these experiments is that our ASP approach has comparable computing performances with CPSM-emb as long as the length of the sequence remains reasonable. This can

be explained considering that solving the embedding problem is a difficult task for pure declarative encodings while CPSM relies on dedicated propagators. The propagators of CPSM solve the embedding problem using additional procedural code. It turns that, for solving the embedding problem in ASP, encoding using a *fill-gaps* strategy appears to be better than using the *skip-gaps* strategy on real datasets, thanks to lower memory requirements.

The second conclusion is that adding constraints on patterns reduces runtime, but increases memory consumption. For real datasets, the more constraints are added, the more memory is required. This is due, to encoding the constraints, but also to encoding the information that may be required to compute constraints. For example, encodings using the *maxspan* constraint require more complex embeddings (`occS/4` atoms) than encodings without this constraint.

To achieve better computational performances, SAT and CP approaches developed dedicated propagators (so called global constraints in CP [Kemmar et al., 2015]). A propagator is a procedural code that can be used within the solver to efficiently verify or falsify some specific atoms of the program. The new *clingo* 5 series integrates "ASP modulo theory" solving processes. This new facilities enables to combine ASP and propagators in an efficient way. Using propagator increases the computational performances, but it reduces the versatility of the approach. Indeed, the part of the resolution that is delegate to the black-box propagator is prewired and forbid interaction with the remaining part of the program. For instance, we are currently working on a propagator that assess whether a pattern supports a sequence (for short, it gives the truth value of the `support/1` atoms). Such propagator is very efficient because it prevents from having to handle embeddings in a declarative way. But, having such propagator forbids to specify any embedding constraint in a declarative way. **The solution of propagator may be considered to tackle larger datasets**, but we do not believe that it is the main strength of our approach.

In our opinion, **the major challenge of declarative pattern mining is to embed complex knowledge in mining task**, i.e. to come back to the ideal of inductive logic programming. Among the declarative paradigms, the logic programming language ASP is suitable for representing complex knowledge, including reasoning. For this reason, our research takes the lead of exploring purely declarative encodings to propose new ways for combining reasoning and mining.

This lead is illustrated by our case study (see Section D). It shows that knowledge reasoning and data mining can co-operate in a declarative framework and that it is helpful both to prototype new pattern mining tasks (from raw data to pattern selection) and to have constraints expressed at the applied semantic level. The pattern constraints are mediated by knowledge.

Our long term goal is to develop *epistemic measure of interestingness* for pattern mining. Declarative pattern mining highlights that a pattern mining tasks is a optimization problem since a interestingness patterns have been specified. In our work, we copy the state of the art procedural algorithms to specify a set of constraints that can be used to specify the interesting pattern: *I want patterns that are frequent!*, *my patterns shall not contain item A!*, *patterns must correspond to situation happening in a short time span!*, etc. These are simple interestingness specification. The notion of interestingness measure in knowledge discovery is deeply studied by Tijl de Bie [Kontonasios et al., 2012] mainly through the prism of unexpectedness. The interestingness of discovered patterns is evaluated by comparing these patterns to the background knowledge. The deviation of the pattern from the background knowledge is a measure of its interestingness.

Chapter 5

Temporal data wrangling

In this chapter, our concern is the design of a data analysis workflow to discover meaningful knowledge from raw (secondary usage) temporal data. A data analysis workflow (or knowledge discovery workflow) is a set of operations that have to be done to extract knowledge from the raw data. Fayyad et al. [1996] shed light on the importance of the different steps that compose a Knowledge Discovery and Delivery (KDD) process. The mains steps of a KDD process are: data selection, data preprocessing (cleaning), data transformation (feature engineering), data mining and data interpretation. And all along this process, there are feedbacks from the expert user that modifies the processing to refine the processing chain such that it delivers meaningful insights.

Choices made in the design of a data analysis workflow highly impacts the information that can be extracted from the data, and thus the knowledge extracted. One of the main steps of the KDD process is the feature engineering. For instance, in Section E.3, we proposed some preprocessing steps to transform the drugs deliveries into drug exposition to extract patterns with QTempIntMiner. This preprocessing can be seen as a feature engineering. Indeed, we created a new type of events (*drug exposure*) which structures the raw data information in such a way that our algorithms can extract knowledge from it. This preprocessing has two parameters: the duration of a treatment after a delivery (30 days) and the distance threshold between two periods to merge them (7 days). Modifying this values modifies significantly the dataset of temporal sequences mined by QTempIntMiner, and then its output. Several attempts has been required to tune the preprocessing of the raw data before being able to have conclusive outcomes.

The importance of the preprocessing steps is enhanced by two specific aspects of the data we are dealing with:

- **secondary usage data**. Secondary usage data are data collected for another purpose than the purpose of the data analysis workflow. The nature of secondary usage data requires feature engineering steps to feed data analytic algorithms with well structured information and meaningful content for analysis purposes. This is especially the case with our case study on medico-administrative database (see Section A).
- **temporal dimension** of raw data. The possibilities for feature engineering with temporal data are huge. The example of drug exposure illustrates a complex feature engineering and there are thousand of possible arrangement of temporal events to create new ones. This has been widely shown with temporal data manipulations in medical application, so called temporal abstraction Combi and Chittaro [1999].

But, designing a data analysis workflows is known to be tedious, human time consuming Verbruggen and De Raedt [2017]. It requires several trial and error attempts to identify which workflow (and which set up) will more likely generates meaningful results. Many recent approaches proposed to automatize the design of simple machine learning workflows. Indeed, the problem is again a traversal of a search space (i.e. the space of all the possible workflows). Instead of letting the user do the exploration, the idea is to explore the search space automatically and let the machine select the best workflows. This is the field of AutoML (automatic Machine-Learning) or Automatic Data Science. It leads to the development of many tools to automatically select the best set up of algorithm [Feurer and Hutter, 2019] or some time the base workflow (for instance, Auto-sklearn [Feurer et al., 2019], AutoWeka [Kotthoff et al., 2019]). The automatic feature engineering is an

extension of the classical feature selection task. For instance, the approach proposed by Boullé et al. [2019] aims at learning propositional features from a relational database, including complex aggregates. The transformed dataset is then used in a machine learning problem to extract Bayesian classification rules. The main challenge of this work is to deal with the very large search space of features combination. This challenge is addressed by MODL [Boullé, 2009], an generic approach to optimize a minimum description length criteria.

Our approach is interactive, we propose a system to support a user in designing its workflows. We believe that the trial and error process has more benefit for designing insightful, potential novel, processing workflows than automatic tools. But, we would like to prevent this tasks from being tedious. To do so, we would like a tool that:

- ▷ facilitates the engineering of new trials, i.e. to ease the adaptation of existing workflows,
- ▷ enables to automatize part of the exploration, e.g. exploring different values for a parameter.

The proposed solution is to formalize data analytic workflows. Formalization aims to propose an abstract representation of workflows. This representation has to be generic enough to represent a wide range of possible workflows. At the same time, we want an abstract representation that can be manipulated by users or automatically to address our expectations.

In this chapter, we present our work on the design of a **temporal data wrangling** which aims at empowering users of temporal data to prepare their data analysis processing workflows from raw data to temporal pattern mining algorithms.

- ▷ in Section A, we present the parallel between data analysis workflows and pharmaco-epidemiological studies. Indeed, this chapter originates from needs for such tools in this context. It motivates some of our choices.
- ▷ in Section B, we present the formalization of a temporal data wrangling as an algebra.
- ▷ in Section C, we focus on the problem of temporal pattern matching, which is a core concept of our algebra.

A Case study context and objectives

Let comes back to our case study of epidemiological studies. Our initial motivation was to support epidemiologists in designing epidemiological studies conducted on medico-administrative databases.

In this section, we draw connections between the field of epidemiological studies and data sciences. Our claim is that an epidemiological study is a data science workflow which manipulates care pathways (i.e. temporal sequences).

A.1 A digital environment of pharmaco-epidemiology studies from Medico-administrative databases

The development of the analysis protocol of a pharmaco-epidemiological study is a long and complex process. It is not straightforward, and it involves a lot of "trials and errors" like any data scientist experiences while trying to discover meaningful knowledge from large datasets. It is especially the case for pharmaco-epidemiological studies using SNIIRAM due to 1) the data richness (each patient is described by a huge number of variables), 2) the data complexity (taxonomies, temporal and spatial dimensions of records) and, above all, 3) the semantic gap between the raw data (e.g. drugs deliveries) and the required information (such as a "stable treatment for epilepsy").

After analysing pharmaco-epidemiological studies, we proposed a general scheme to carry out pharmaco-epidemiological studies form SNIIRAM. This scheme is illustrated by Figure 5.1. The main steps of the process are the following.

Cohort building: it selects a set of patients that have to be included in the study. The main difficulties for clinicians is to express inclusion and exclusion criteria (based on available raw data) that are sound and accurate for the clinical study, especially for its statistical representativeness of the population and the statistical significance of study outcomes.

Building care pathways: it transforms the raw data into a dataset of care pathways in a standard model. This step gives an abstract view on patients in order 1) to partly bridge the

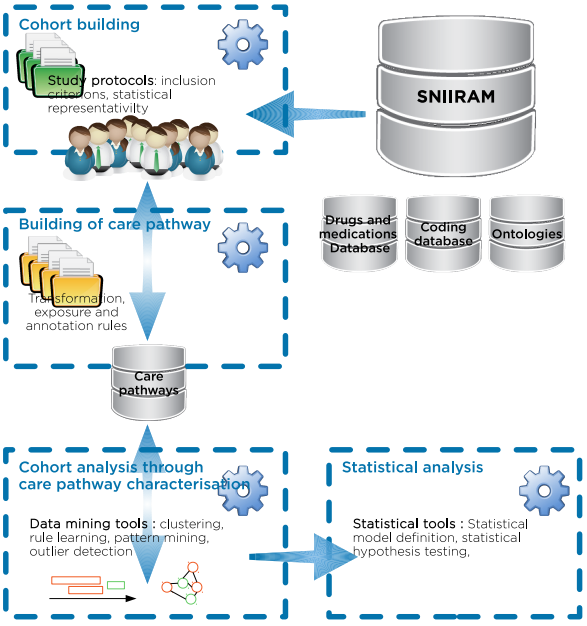


Figure 5.1: Illustration of principles of pharmaco-epidemiological studies.

semantic gap, and 2) to prune the data that the clinician assumed to have no incidence on the study. Such transformation may be implemented as transformation algorithms or rules. For example, in GENEPI, drugs deliveries are transformed into drugs exposures. The difficulty for clinicians is to define such algorithms that may have a lot of parameters.

Care pathways characterization: it consists in using supervised and unsupervised data mining algorithms to extract meaningful patterns from data, and more especially sequential patterns. Such patterns may help clinician to generate new hypothesis.

Statistical analysis: Statistical analysis are finally run to assess association between the outcome and the covariates.

All these steps are practically intertwined (see double-arrows): the definition of inclusion criteria may depend on the care pathway abstraction and the abstract process may be adjusted according to the first data mining results.

A.2 When pharmaco-epidemiology with MADB becomes data science

MADB drastically changes the way to carry the pharmaco-epidemiological studies in two different ways: in an epistemological way and in a practical way.

The major change is the way the data are gathered. Indeed, the data gathered to conduct the analysis of a classical study are collected for this specific purpose. This is the primary usage of the collected data contribute to the analysis. With MADB, the data has not been collected for the study to conduct. Such secondary usage data does not necessarily contains the expected information as it and they require pre-processing before being integrated in a data analysis (see step "Building care pathway" in Figure 5.1).

This is an epistemological change because it changes the work of epidemiologists. The main issues in classical epidemiological studies is "how to recruit patients in order to collect the data to apply a statistical test that will answer my research question". With MADB, the question becomes "how to transform the available raw data for thousands of patients in order to create a dataset on which apply a statistical test that answers my research question". In the former case, the problem is to create the rows of a dataset, in the latter case, the problem is to create its columns.

This changes started with the creation of large, non-specialized cohort of patients (such as CONSTANCE [Zins et al., 2015] or Nutrinet [Hercberg et al., 2010]), but it is facilitated by use of numerical data that can easily be shared and copied for multiple usages. And, it currently impacts

medical research in France with the creation of the Health Data Hub[1].

The second change is in the practical work of epidemiologists working on MADB. Epidemiologists become data scientists who have to muggle the data to feed data analysis tools. They have to mobilize their medical expertise but also to master the data management in order to make the data "speak the truth". Then, a parallel can be drawn between pharmaco-epidemiology and data science such that the work of the pharmaco-epidemiologist looks very close to the work of a data scientist. The parallel for the main steps of a pharmaco-epidemiological study (Figure 5.1) are as follow:

- ▷ Cohort building is a (SQL) query on the SNDS data warehouse: it also directly corresponds to the *data selection* of KDD process
- ▷ Building care pathways is a data wrangling step. Data wrangling can be defined as a process by which the data required by an application is identified, extracted, cleaned and integrated, to yield a dataset that is suitable for exploration and analysis [Furche et al., 2016]. It corresponds to the *data transformation* of the KDD process.
- ▷ Cohort analysis and statistical analysis are the application of data science algorithms. It correspond to the *data mining* of the KDD process.

Conversely, pharmaco-epidemiology with SNDS is a very interesting representative of data science, especially with temporal data. The model of the epidemiological study can be transposed to many other contexts in which the user is interested in discovering knowledge about the data at hand. The SNDS data enables to built patients longitudinal care pathways, i.e. a sequence of complex events, exactly like the logs gathered from various system are a set of sequences of complex events. The starting point of the data scientist is a data warehouse of data gathered from hither and thither (similarly to the SNDS, it is often secondary usage data) and, in most cases, a question about the system underlying the gathered data. The data scientist has to imagine how to muggle the data to create a dataset and a data analysis task that answer the question.

The epidemiologists muggle the data to feed data analysis tools. Verbruggen and De Raedt [2017] pointed out that 80% of a data scientist work is devoted to preprocessing and only 20% to the actual data analytic step. The same problem is currently observe with epidemiological studies, especially because the tools used in this domain appears to be restrictive. The problem of empowering the epidemiologist with data wrangling tools becomes a very important question.

Bacry et al. [2019] proposed to use statistical machine learning techniques to analyze care pathways. Their event based method requires a feature engineering to identify which are the right events to take into account in a epidemiological studies. To the best of our understanding, they propose a general feature engineering scheme and set up automatically its parameters. They use same medico-administrative database (SNIIRAM) to evaluate their workflow. Their approach does not have an explicit temporal model. The approach is based on feature engineering to transform the raw data of a patient (from the different tables of the database) to create new features based on templates. The whole patient data are represented as a vector (with high dimensionality). Then, the set of patient yields a tabular dataset that can be processed via classical machine learning techniques.

Our approach is "care-pathways-centered" meaning that we initially have a longitudinal view of the SNIIRAM data, representing its care pathways as a temporal sequence, and we advocate for the use of data analytic tools to process temporal sequences (e.g. temporal pattern mining algorithms we introduced in the previous chapters).

Then, our objective is to propose a tool to manipulate a temporal sequence and to induce new transformed temporal sequences. For instance, drug exposure is an example of such a transformation. In the same spirit, JTSA [Sacchi et al., 2015] proposes a framework for time series temporal abstraction. It provides collection of algorithms to perform temporal abstraction and preprocessing of time series and a framework for defining and executing data analysis workflows.

[1] Health Data Hub is an administrative structure to ease the access of french medical data for research purposes. It has been officially create on 1th Decembre.

B Temporal Sequences Wrangling Workflows[2]

The analysis workflows representing pharmaco-epidemiology studies are workflows to process care pathways, i.e. a dataset of temporal sequences. For sake of generality, we present an algebra to formalize the main steps of a wrangling of temporal sequences. We illustrate our algebra with pharmaco-epidemiological studies. This formalization can be used in any application involving temporal sequences.

We are interested in developing tools to support experts to prototype data analysis tool chains. As we seen in the introduction, the pattern mining tools are the final step of the data analysis process. Beforehand, many steps are required to constitute a dataset of sequences from the raw dataset: selection of the individuals, selection of the interesting events, transformation of events, etc. They are required to yield dataset in a suitable format for advanced data analytic tools (e.g. the algorithms presented in Chapter 3 require temporal sequence of itemsets). These pre-processing steps, known as data wrangling, require a lot of human effort and a strong knowledge about the data itself.

Our general challenge is to facilitate the building of temporal data analysis tool chains and to enable experts proceeding by trial and error. The proposal of a Domain Specific Language (DSL) dedicated to the analysis of temporal data addresses this challenge by formalizing a language which enables to specify quickly a processing chain, and to modify it while outputs are unsatisfactory. The tool chain itself being composed of data analytic components.

Beyond the query language itself, which enables experts to specify tool chains, a (DSL) is equipped with a computational model such that queries (i.e. processing chains) may be executed.

In this Section, we present our preliminary proposal to formalize the temporal data wrangling in two steps. Section B.1 proposes an algebra to formalize the main steps of the study. This algebra is made of abstract operators. They are parameterized by a description of temporal patterns (see Section C). This framework is then illustrated on the GENEPI case study in Section B.4 .

B.1 Temporal Data Mining Query Language

In this section, we introduce our domain specific query language for formalizing temporal data analysis tool chains.

We choose an algebraic syntax to formalize our query language. An algebraic syntax offers an abstract and flexible representation of the tool chain that makes it independent from the computational model.

In principle, the tool chain is represented by an expression like:

$$op^1_{p_1}\left(op^2_{p_2}\left(\langle \mathscr{C}_1, \mathcal{D}_1\rangle\right), op^3_{p_3}\left(\langle \mathscr{C}_2, \mathcal{D}_2\rangle\right)\right) \tag{5.1}$$

where $op^1_{p_1}$, $op^2_{p_2}$ and $op^3_{p_3}$ are elementary operators with parameters (p_1, p_2 and p_3) and $\langle \mathcal{C}_1, \mathcal{D}_1\rangle$ represents a dataset of sequences. Each operator also returns a dataset of sequences. The dataset of sequences is represented by a pair. $\mathcal{D}$ is the dataset itself representing and $\mathcal{C}$ is called the context. The later is an abstract representation of the dataset formalism. An operator processes one or more datasets of sequences and generates (at most) another dataset of sequences.

In the following, we formalize the different elements of a query starting by the model of a dataset of sequences.

In principle, an element of the dataset, so called *individual*, is described by two aspects:

- ▷ a tuple of sequence values $\langle c_1, \dots c_n\rangle$ that are not timestamped (for instance, the sex or the birthday of a patient),
- ▷ a temporal sequence of complex events: each event is represented by a tuple of values $\langle [t^-, t^+], v_1, \dots, v_n\rangle$ where $[t^-, t^+]$ is a temporal interval and $v_1, \dots, v_n$ are typed values. Intuitively, the event values (except timestamp) are described in a relational model.

Remark 9 (Interval vs punctual events)**.** *In this model the temporal events are associated to interval time span. Punctual events are temporal event that have a singleton interval, i.e., a temporal interval* $[t, t]$*.*

[2]This section presents results that are not yet published.

In addition, the dataset of sequences is equipped with a *context* which is a pair of abstract models of sequences values (constants for the patient) and event values (with timestamps). In short, the abstract model is a collection of attribute definitions which specify the type, the domain and the name of each attribute.

Definition 33 (Attribute definition)**.** *An attribute $a = \langle \nu, \tau, \delta \rangle$ is defined by:*

- ▷ *a name ν, i.e. a literal that identifies the attribute.*
- ▷ *a type τ among the common types (boolean, integer, float, string) and types for enumerates or taxonomies. Enumerate and taxonomy values enable to represent a finite collection of elements.*
- ▷ *(optionally) a domain δ. The domain is mandatory for type enumerate or taxonomy.*

Definition 34 (Dataset context)**.** *A context is a tuple $\langle \mathcal{C}, \mathcal{R}_1, \ldots, \mathcal{R}_r \rangle$ where $\mathcal{R}_1, \ldots, \mathcal{R}_r$ and $\mathcal{C}$ are sets of attribute definitions representing respectively the sequences attributes and the events attributes.*

In a set of attributes, two different attributes can not hold the same name.

Intuitively, the dataset context give the schema of a sequence: $\mathcal{C}$ is the relation schema for sequence attributes and $\mathcal{R}_1, \ldots, \mathcal{R}_r$ are relational schema that can be used to describe different types of events. We denote $\Re$ the set of sequence attributes schema, and r is the number of relation schema. Each relation schema $\mathcal{R} \in \Re$ represents events of type $\mathcal{R}$.

Remark 10 (Data model vs FOL representation of temporal sequences)**.** *This representation of temporal events has connections with the example of the manuscript introduction, and that we used in the case study of Section D in Chapter 4. The listing below is an example of information that we want to represent with our data model.*

```
1 sex(p1, 1).
2 birthYear(p1, 1979).
3 event(p1, 0, 0, drugDelivery(3430126, 1) ).
4 event(p1, 3, 5, procedure(ECGH011) ).
5 event(p1, 12, 12, drugDelivery(4991401, 1) ).
6 event(p1, 23, 23, drugDelivery(3588755, 2) ).
7 event(p1, 38, 38, procedure(YYYY600) ).
8 event(p1, 45, 45, drugDelivery(3723093, 1) ).
9 event(p1, 62, 66, procedure(GAQE001) ).
```

In this case, predicates `sex/2` and `birthYears/2` encodes the sequences values, while `event/4` predicate encodes temporal sequences of complex events.

In the FOL syntax, the context correspond to the set of predicates.

Definition 35 (Attributed events in context $\langle \mathcal{C}, \Re \rangle$)**.** *Let $\mathcal{R} = \{r_1, \ldots, r_m\} \in \Re$ be the set of m event attribute definitions, then an **attributed event** of type $\mathcal{R}$ is a tuple of values $([t^-, t^+], v_1, \ldots v_m)$ such that:*

- ▷ *$t^-, t^+ \in \mathbb{R}$ are timestamps such that $t^- \leq t^+$,*
- ▷ *v_i is a value of type τ_i in the domain δ_i for all $i \in [m]$, where $r_i = \langle \nu_i, \tau_i, \delta_i \rangle$ is the attribute definitions of the i-th attribute.*

For sake of conciseness, we denote by $\vec{v}$ the tuple of values.

Definition 36 (Attributed sequence in context $\mathscr{C} = \langle \mathcal{C}, \Re \rangle$)**.** *Let $\mathcal{C} = \{c_1, \ldots, c_n\}$ be the set of n sequence attribute definitions, then an **attributed sequence** is a pair $\left(id, \vec{v}, \boldsymbol{s}^1, \ldots, \boldsymbol{s}^r\right)$ such that:*

- ▷ *$id \in \mathbb{N}$ is an identifier of the individual,*
- ▷ *$\vec{v}$ is a tuple of values such that v_i is a value of type τ_i in the domain δ_i for all $i \in [n]$, where $c_i = \langle \nu_i, \tau_i, \delta_i \rangle$ is the attribute definitions of the i-th attribute,*
- ▷ *$\boldsymbol{s}^i = \langle s^i_1 \ \ldots \ s^i_{k^i} \rangle$ is a sequence of attributed events of type $\mathcal{R}_i$, i.e. s_i is an attributed event in context $\mathcal{R}_i$.*

The identifier of the individual *id* represents the individual from which the sequence originates. It is not an identifier of the sequence. More especially, this value is not necessarily unique in the dataset of sequences.

Our model of time for temporal events is based on intervals. This enables to represent both events with duration ($t^+ > t^-$) and punctual events ($t^+ = t^-$). One can note that we do not take into account the problem of multi-temporal granularity in the data [Euzenat and Montanari, 2005]. The timestamps are represented by real values. It can be seconds, minutes, days or whatever but the domain of time is unique for all sequences in a dataset of sequences.

Definition 37 (Dataset of sequences in context $\mathscr{C}$)**.** *Let $\mathscr{C} = \langle\{c_1, \ldots, c_n\}, \{r^1_1, \ldots, r^1_{m^1}\}, \ldots, \{r^r_1, \ldots, r^r_{m^r}\}\rangle$ be a context, a dataset of sequences with context $\mathscr{C}$ denoted $D_{\mathscr{C}}$ or $(\mathscr{C}, D)$, is a set of sequences in context $\mathscr{C}$.*

Example 26 (Care pathways)**.** *In this example, we propose to represent care pathways from data of the SNDS. An individual is a patient which has three attributes: its sex, its birth year and its localization code (zip code). Then we have $\mathcal{C} = \{\langle \mathsf{sex}, enum, \{F, M\}\rangle, \langle \mathsf{by}, Integer, [1900, 2100]\rangle, \langle \mathsf{zipcode}, Integer, \mathbb{N}^+\rangle\}$.*

The care pathway of a patient is made of drugs deliveries, medical procedure and hospitalization. The attributes of a drug delivery event are the ATC code, and the quantity of drug boxes. The attribute of a medical procedure event is the CCAM code. The attributes of an hospitalization event is the CIM 10 code (entrance principal diagnosis).

The $\mathcal{R}_{drug} = \{\langle \mathsf{code}, enum, ATC\rangle, \langle \mathsf{quantity}, Integer, \mathbb{N}^+\rangle\}$, $\mathcal{R}_{act} = \{\langle \mathsf{code}, enum, CCAM\rangle\}$ and $\mathcal{R}_{hosp} = \{\langle \mathsf{code}, enum, CIM10\rangle\}$.

We can now illustrate two instances of a sequence in the context $\mathscr{C} = \{\mathcal{C}, \mathcal{R}_{drugs}, \mathcal{R}_{act}, \mathcal{R}_{hosp}\}$:

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{drugs} = \langle([0,0], [C01DA02, 1])\ ([10,10], [R01AD52, 3])\ ([31,31], [C07AA07, 1])\rangle, \boldsymbol{s}_{acts} = \langle([8,8], [HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle([35,39], [K35.0])\rangle)$
2. $(2, [F, 1945, 27300], \boldsymbol{s}_{drugs} = \langle([0,0], [A01BB04, 4])\ ([0,0], [R01AD52, 1])\ ([30,30], [R01AD52, 3])\ ([31,31], [A01BA07, 1])\ ([90,90], [R01AD52, 1])\ \rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle([32,34], [K35.0])([95,99], [K35.0])\rangle)$

Let us now present the abstract notion of operator. Intuitively, an operator processes a collection input datasets to output one dataset of sequences. Then, this an atomic operation on a dataset of sequences.

Definition 38 (Operator)**.** *An **operator** op of arity $n \in \mathbb{N}$ is denoted $op_{[p_1=\rho_1,\ldots,p_m=\rho_m]}(\langle\mathscr{C}_1, \mathcal{D}_1\rangle, \ldots, \langle\mathscr{C}_n, \mathcal{D}_n\rangle)$ where:*

- *$p_1, \ldots, p_m$ are the parameters of the operator and $\rho_1, \ldots, \rho_m$ are parameters values or literals. For any $i \in [m]$, if ρ_i is a literal, then it represents possible alternative configurations for the operator. The value of the parameter will be given at the execution.*
- *$\langle\mathscr{C}_1, \mathcal{D}_1\rangle, \ldots, \langle\mathscr{C}_n, \mathcal{D}_n\rangle$ is an ordered list of datasets of sequences. This list can be empty. We denote by $lit(op)$ the list of the literals (or variables) used in the parameters of the operator op and $ar(op)$ the arity of the operator op.*

*If the operator in a not **terminal**, then the operator op yields a new dataset $\langle\mathscr{C}, \mathcal{D}\rangle$. A **terminal** operator does not yield any dataset. A terminal operator can not be of arity 0.*

Some remarks and precision about the definition of an operator:

- An operator of arity 0 (empty list of dataset) corresponds to a data source: it yields a dataset of sequences from not any other dataset.
- The parameters of an operator are typed. Their types are among the common types (*string*, *int*, *float*, *boolean*).
- A terminal operator can not be used as an argument for another operator. The terminal operator may have side effect: it can be used to output some results in files or to print results.

Now that operators have been defined, a workflow or a tool chain is simply a composition of operators. Instead of proposing an algebraic definition of a workflow, as illustrated in Eq. 5.1, we propose a definition based on a graph.

Definition 39 (Processing chain/workflow graph)**.** *A workflow is a directed graph of n operators $\langle\{op_1, op_2, \ldots, op_n\}, e : [n]\times[n] \mapsto \mathbb{N}\rangle$ where $\{op_1, op_2, \ldots, op_n\}$ are operators (vertices of the graph) and e are the directed edges between operators, and such that:*

- *each terminal operator has an outdegree[3] equals to 0,*

[3] The outdegree of a vertex is the number of edges pointing from it.

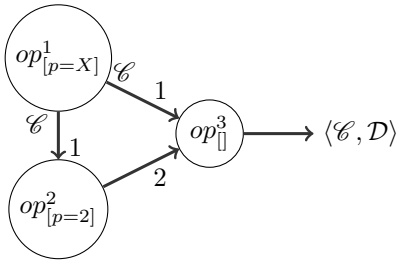


Figure 5.2: Graph of the workflow in Example 27: $\langle \mathscr{C}, \mathcal{D} \rangle = op^3_{[]}(\mathcal{C}, op^2_{[p=2]}(\mathcal{C} = op^1_{[p=X]}()))$.

▷ *each operator vertex has an indegree*[4] *equals to the arity of the operator.*

Let o and o' be two operators, then $e(o, o') > 0$ iff o is a parameter of the o' operator. The edge direction indicates the data flow from the producer to the consumer. The value $e(o, o')$ indicates the position of o in the list the parameters of o'.

The workflow does not constraint the outdegree of non terminal operators. This means that the output of an operator can be used by several operators. In this case, the outdegree. In the algebraic representation of a workflow, we use renaming of output dataset to represent its multiple usage (see Example 27). In addition, a non terminal operator without outedge is an output of the workflow. Thus, the workflow may have several outputs.

We will see that operators may have some requirements for their operands (for example, having an attribute with a specific name). These requirements are integrity constraints that are specified by each operator. We do not represent these constraints in our formalization of a workflow. Indeed, we relegate these constraint to the workflow execution process. The operator is executed iff the constraints are satisfied. Otherwise, there is no output.

Example 27 (Algebraic and graph representation of a workflow)**.** *Let op^1, op^2 and op^3 be three operators of arity 0, 1 and 2 (respectively) and let $\langle \mathscr{C}, \mathcal{D} \rangle = op^3_{[]}(\mathcal{C}, op^2_{[p=2]}(\mathcal{C} = op^1_{[p=X]}()))$ be a workflow. In this workflow, $op^1_{[p=X]}$ is a data source (null indegree). The value of parameter p is given by a literal (X). A value is assigned to X at the execution of the workflow. The value of the parameter p in operator op^2 is 2. The syntax $\mathcal{C} = op^1_{[p=X]}()$ denotes the renaming of the output of op^1: once in op^1 and once in op^3. It illustrates the multiple usage of the output of the operator op^1. The Figure 5.2 illustrates the graph representation of this algebraic expression.*

Definition 40 (Query)**.** *A **query** is a triple $(\mathcal{G}, \mathcal{O}, \mathcal{X})$ where:*

▷ *$\mathcal{G}$ is a workflow involving operators $\{op^1, op^2, \ldots, op^n\}$,*

▷ *$\mathcal{O} \subseteq \{op^1, op^2, \ldots, op^n\}$ is a set of output operators for which we would like the dataset,*

▷ *$\mathcal{X}$ is an assignment of parameters literals of $\bigcup_{i \in [n]} lit(op^i)$.*

The assignment of parameter literals is global. This means that the same value replaces all use of a parameter literal X in the workflow.

At the time, there is no query pre-checking mechanism in our tools. Some operators have requirements (for instance a number of input dataset, or event requirement on the context of the input dataset). Then, the requirements are checked while running the workflow. It would be interesting to induce the context without running operators to have static pre-checking of the workflow, and avoid running workflow with prototyping errors.

B.2 Workflow execution

The previous section introduces our data model and our generic notions of workflow and query. In this section, we introduce our computational model, i.e., the description of how to execute a workflow.

[4]The indegree of a vertex is the number of edges pointing to it.

Algorithm 6: ExecuteOperator : execution of an operator.

input : op: an operator of arity n, $\mathcal{X}$: literals assignment,
$\mathcal{G} = \langle \{op^1, op^2, \ldots, op^n\}, e : [n] \times [n] \mapsto \mathcal{N} \rangle$: a workflow.
output: R: a datasets of sequences

$\mathcal{P} \leftarrow [p_1 = \emptyset, \ldots, p_n = \emptyset]$// parameters of the operator op.
// Collect op operand, i.e. dataset of sequences:
foreach $op' \in \{o | e(o, op) > 0\}$ **do**
  $p_{e(op',op)} \leftarrow$ ExecuteOperator$(op', \mathcal{G})$;
// Execute the operator:
$R \leftarrow op_{[p_1 = \mathcal{X}(\rho_1), \ldots, p_m = \mathcal{X}(\rho_m)]}(p_1, \ldots, p_n)$;

Algorithm 7: Query execution

input : $(\mathcal{G} = \langle \{op^1, op^2, \ldots, op^n\}, e : [n] \times [n] \mapsto \mathcal{N} \rangle, \mathcal{O}, \mathcal{X})$: a query
output: $\mathcal{R}$: a set of datasets of sequences

foreach $op \in \mathcal{O}$ **do**
  $\mathcal{R} \leftarrow \mathcal{R} \cup$ ExecuteOperator$(op, \mathcal{X}, \mathcal{G})$

Algorithm 7 gives the principle of a query execution. The execution of a query starts by calling the ExecuteOperator function on output operators. Indeed, to run only the necessary operators, we start from the output to backtrack in the workflow graph. The function ExecuteOperator (see Algorithm 6) is a recursive function that first executes the antecedent operators in the graph (i.e. the parameters of an operator) to collect the dataset of sequences to proceed and then execute the operator itself.

Before executing the query, the parameters literals are substituted by their values defined in $\mathcal{X}$. For any rho_i, $\mathcal{X}(\rho_i)$ denotes the value ρ_i is ρ_i is a value or it denotes the assigned value of ρ_i in $\mathcal{X}$ if ρ_i is a parameter literal.

Note that the computational model of our DSL has not been optimized. More especially, there are several scientific workflow management systems [Liu et al., 2015a] that provides very efficient (and distributed) system to execute complex workflows. We simply prevent from computing twice the datasets of sequences in case of multiple usage of an operator.

To sum up, this section presents the notions of query that represents a workflow for which each operator processes dataset of sequences. We formalized the notion of dataset of sequences and gives the principle of the execution of a query. The core of the domain specific query language lies in the definition of a set of operators that can be used to design a query.

B.3 Abstract operators for dataset of sequences

We now introduce a collection of abstract operators for processing dataset sequences. For each of these operators, we briefly describe transformation of contexts and sequences; and we give some examples of concrete operators on the running example 26.

In this section we denote by $(id, \vec{v}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_r)$ and $(id', \vec{v}', \boldsymbol{s}'_1, \ldots, \boldsymbol{s}'_r)$ two attributed sequences in a dataset of sequences $\mathcal{D}_{\mathscr{C}}$ upon the context $\mathscr{C} = \langle \{c_1, \ldots, c_n\}, \{r^1_1, \ldots, r^1_{m^1}\}, \ldots, \{r^r_1, \ldots, r^r_{m^r}\} \rangle$. $\vec{v}$ and $\vec{v}'$ are two vectors of size n.

Datasource (S, nullary) A data source operator creates a dataset of sequences from a file or a relational database. It creates both the context and the dataset of sequences.

Projection ($\pi_{[a_1,\ldots,a_k]}$, unary) A projection operator restricts the sequence attributes (π^S) or the event attributes ($\pi^E_{[(r_1,a_1),\ldots,(r_k,a_k)]}$) or sequence types ($\pi^T$) to the attributes or types with names $a_1, \ldots, a_k$. It discards any other attribute. Event attributes are identified by the name of the sequence type and the name of the attribute.
This operator simplifies the context of a dataset. The output context contains only the attributes that have been projected and information that are out of the context are removed.

Example 28. *With the dataset introduced in Example 26, the operator* $\pi^{E}_{[(drugs,code),[acts,code],[hosps,code]]}$ *removes the drugs quantities and thus yields the following sequences:*

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{drugs} = \langle([0,0],[C01DA02])\ ([10,10],[R01AD52])\ ([31,31],[C07AA07])\rangle,$ $\boldsymbol{s}_{acts} = \langle([8,8],[HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle([35,39],[K35.0])\rangle)$
2. $(2, [F, 1945, 27300], \boldsymbol{s}_{drugs} = \langle([0,0],[A01BB04])\ ([0,0],[R01AD52])\ ([30,30],[R01AD52])([31,31],$ $[A01BA07])\ ([90,90],[R01AD52])\ \rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle([32,34],[K35.0])([95,99],[K35.0])\rangle)$

The operator $\pi^{T}_{[acts]}$ *removes drugs deliveries and hospitalizations:*

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{acts} = \langle([8,8],[HNCA006])\rangle)$
2. $(2, [F, 1945, 27300], \boldsymbol{s}_{acts} = \langle\rangle)$

Filtering ($\theta_{[\varphi_1,\ldots,\varphi_r]}$, unary) A filtering operator selects sequence events satisfying the condition specified by $\varphi_1, \ldots, \varphi_r$ where r is a number of event types. There is one condition per event type. Events that do not satisfy the condition are discarded from the sequences.
For all attributed sequence $(id, \vec{v}, \boldsymbol{s}_1, \ldots, \boldsymbol{s}_r)$ in $\mathcal{D}$, the operator yields an attributed sequence $(id, \vec{v}, \boldsymbol{s}'_1, \ldots, \boldsymbol{s}'_r)$ such that $\boldsymbol{s}'_i$ is a subsequence of $\boldsymbol{s}_i$, $\boldsymbol{s}'_i \preceq \boldsymbol{s}_i$, and $s \in \boldsymbol{s}'_i \Leftrightarrow \varphi_i(s)$ for all $i \in [r]$.

Example 29. *With the dataset introduced in Example 26, the operator* $\theta_{[\varphi_{drugs},\varphi_{acts},\varphi_{hosps}]}$ *where* φ_{drugs} *is true if* code *equals* $R01AD52$*; and* φ_{acts} *and* φ_{hosps} *are false, then, the operator yields the following sequences:*

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{drugs} = \langle([10,10],[R01AD52,3])\rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle\rangle)$
2. $(2, [F, 1945, 27300], \boldsymbol{s}_{drugs} = \langle([0,0],[R01AD52,1])\ ([30,30],[R01AD52,3])\ ([90,90],[R01AD52,1])\rangle,$ $\boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle\rangle)$

Selection ($\sigma_{[\varphi]}$, unary) A selection operator selects a sequence in $\mathcal{D}$ that satisfies the condition φ. Sequences that does not satisfy the condition are discarded from the yielded dataset. The context of the yielded dataset is the same as the input dataset. The condition concerns both sequence attributes and sequence events.

Example 30. *With the dataset introduced in Example 26, the operator* $\sigma_{[\varphi]}$ *with* $\varphi = (sex = M \wedge (\exists e$ *s.t.* $e.type = drugs \wedge e.code = R01AD52))$ *removes the second individuals (just because of* sex*). Thus it yields the following sequences:*

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{drugs} = \langle([0,0],[C01DA02,1])\ ([10,10],[R01AD52,3])\ ([31,31],[C07AA07,1])\rangle,$ $\boldsymbol{s}_{acts} = \langle([8,8],[HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle([35,39],[K35.0])\rangle)$

Join ($\bowtie_{id}$, binary) A join operator has two input datasets $\langle\mathscr{C}, \mathcal{D}\rangle$ and $\langle\mathscr{C}', \mathcal{D}'\rangle$ and yields a dataset $\langle C \cup \mathscr{C}', \mathcal{D}''\rangle$. if $id = id'$, then the operator yields a sequence in $\mathcal{D}''$ where sequence attributes are merged and sequence events are merged. To merge sequence events, events with same type and same temporal interval (same date and same duration) merge their attributes.[5]

TimeJoin ($\bowtie^{T}$, binary) A TimeJoin operator has two input datasets $\langle\mathscr{C}, \mathcal{D}\rangle$ and $\langle\mathscr{C}', \mathcal{D}'\rangle$ and yields a dataset $\langle\mathscr{C}, \mathcal{D}''\rangle$. A sequence $(id, \vec{v}, \boldsymbol{s}'')$ of $\mathcal{D}''$ where there exists $(id, \vec{v}, \boldsymbol{s})$ in $\mathcal{D}$ and $(id, \vec{v}', \boldsymbol{s}')$ in $\mathcal{D}'$ such that for all $s = ([t^-, t^+], \boldsymbol{v}) \in \boldsymbol{s}''_i$, $s \in \boldsymbol{s}_i$ and there exists $s' = ([t'^-, t'^+], \boldsymbol{v}') \in \boldsymbol{s}'_j$, $[t^-, t^+] \subseteq [t'^-, t'^+]$.

Example 31. *In this example, we first need to introduce a new dataset of sequences* $\mathcal{D}'$ *with only time periods (attributes are useless):*

1. $(1, [], \boldsymbol{s} = \langle([0,37],[])\rangle)$
2. $(2, [], \boldsymbol{s} = \langle([0,30],[])\ ([60,90],[])\rangle)$

The application of the TimeJoin operator on the dataset introduced in Example 26, yields the following sequences:

1. $(1, [M, 1980, 35700], \boldsymbol{s}_{drugs} = \langle([0,0],[C01DA02,1])\ ([10,10],[R01AD52,3])\rangle, \boldsymbol{s}_{acts} = \langle([8,8],$ $[HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle\rangle)$
2. $(2, [F, 1945, 27300], \boldsymbol{s}_{drugs} = \langle([0,0],[A01BB04,4])\ ([0,0],[R01AD52,1])\ ([30,30],[R01AD52,3])$ $([90,90],[R01AD52,1])\rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle\rangle)$

TimeTransformer ($\delta_{[l,u]}$, unary) A TimeTransfomer operator makes transformations of the temporal intervals. $u : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ and $l : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}$ are two functions to transform the temporal interval to a timestamp. For all sequence of the dataset, it transforms each event $([t^-, t^+], \vec{v})$ into an event $([l(t^-, t^+), u(t^-, t^+)], \vec{v})$.

[5] Note that in case values are inconsistent, there is no predefined merging strategy.

Labeler ($\lambda_{[a,l]}$, unary) A labeler operator creates a new sequence attributes a within a domain $\mathcal{A}$. Then, the context of the output dataset is $\langle\{c_1,\ldots,c_n,a\},\{r^1_1,\ldots,r^1_{m^1}\},\ldots,\{r^r_1,\ldots,r^r_{m^r}\}\rangle$. For each sequence of the dataset, the value of attribute a is computed by the labeling function l from other sequence attributes and sequence events ($l(\boldsymbol{s}) \in \mathcal{A}$).

Example 32. *We consider the operator $\lambda_{[\mathsf{addict},l]}$ with the dataset introduced in Example 26, where l is a function that is true if the total number of $R01AD52$ deliveries is above 4 and false otherwise. Then, the operator yields the following dataset:*

1. $(1,[M,1980,35700,\textbf{\textit{False}}],\boldsymbol{s}_{drugs} = \langle([0,0],[C01DA02,1])\ ([10,10],[R01AD52,3])\ ([31,31],[C07AA07,1])\rangle,$ $\boldsymbol{s}_{acts} = \langle([8,8],[HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle([35,39],[K35.0])\rangle)$
2. $(2,[F,1945,27300,\textbf{\textit{True}}],\boldsymbol{s}_{drugs} = \langle([0,0],[A01BB04,4])\ ([0,0],[R01AD52,1])\ ([30,30],[R01AD52,3])$ $([31,31],[A01BA07,1])\ ([90,90],[R01AD52,1])\ \rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} = \langle([32,34],[K35.0])([95,99],[K35.0])\rangle)$

The context of the new dataset now contains a new sequence attribute $\langle\mathsf{addict}, bool, \{True, False\}\rangle$.

Induction ($\iota_{[\varphi,\psi]}$, unary) An induction operator creates new events a in a sequence in case condition φ on the sequence attributes and events. ψ yields the events timestamps and attributes.

Example 33. *With the dataset introduced in Example 26, with ψ a function that creates a new event exposure of in the sequence* **drug** *while there are three deliveries of the same drug (φ condition). The timestamps of new events are the smallest lower bound and the greatest upper bound of drugs deliveries. It yields the following sequences:*

1. $(1,[M,1980,35700],\boldsymbol{s}_{drugs} = \langle([0,0],[C01DA02,1])\ ([10,10],[R01AD52,3])\ ([31,31],[C07AA07,1])\rangle,$ $\boldsymbol{s}_{acts} = \langle([8,8],[HNCA006])\rangle, \boldsymbol{s}_{hosps} = \langle([35,39],[K35.0])\rangle)$
2. $(2,[F,1945,27300],\boldsymbol{s}_{drugs} = \langle([0,0],[A01BB04,4])\ ([0,0],[R01AD52,1])\ ([30,30],[R01AD52,3])$ $([31,31],[A01BA07,1])\ ([90,90],[R01AD52,1])$ ***([0,90], [exposure,1])***$\rangle, \boldsymbol{s}_{acts} = \langle\rangle, \boldsymbol{s}_{hosps} =$ $\langle([32,34],[K35.0])([95,99],[K35.0])\rangle)$

Rename ($\rho^{\cdot}_{[\mathsf{old}=\mathsf{new}]}$, unary) Rename the attribute name old into new. ρ^S renames sequence attributes and ρ^T renames sequence types. $\rho^E_{[k,\mathsf{old}=\mathsf{new}]}$ renames event attributes of type k.

SequenceMerger ($\tau^{\cup}_{[k,l]}$, unary) Merge the events of sequence of type l within the events of type k. It requires that l and k have same types and same domains.

Example 34. *With the simplified dataset computed in Example 28, the operator $\tau^{\cup}_{[drugs,acts,hosps]}$ merge all code events*[6]*:*

1. $(1,[M,1980,35700],\boldsymbol{s} = \langle([0,0],[C01DA02])\ ([10,10],[R01AD52])\ ([31,31],[C07AA07])\ ([8,8],$ $[HNCA006])\ ([35,39],[K35.0])\rangle)$
2. $(2,[F,1945,27300],\boldsymbol{s} = \langle([0,0],[A01BB04])\ ([0,0],[R01AD52])\ ([30,30],[R01AD52])\ ([31,31],$ $[A01BA07])\ ([90,90],[R01AD52])\ ([32,34],[K35.0])\ ([95,99],[K35.0])\rangle)$

The above operators are *abstract operations* on our data model. Their concrete semantic depends on the conditions and functions that specifies their actual behavior. We seen in our example that for each operator we had to instantiate them by providing definition of conditions or functions. We propose two complementary alternatives to represent them:

- ▷ **dedicated algorithms.** Dedicated algorithms enables to implement computationally efficient methods. In addition, it reduces the setting effort for each operator. The user will only have to set up the algorithm parameters. The drawback of this solution is that each new operator behavior requires a specific operator development.
  In practice, it is especially interesting to implement conditions by pattern matching algorithms. The user specifies a pattern according to which a sequences (Selection operator) of events (Filtering operator) must be kept or discarded. We implemented two simple pattern matching algorithms: MustHave which specifies that an event must occurs in a sequence, ChroniclePattern which is a chronicle recognition algorithm (see Section C.1).
  The functions of TimeJoin operator are implemented by linear transformation of the interval of the input event.
- ▷ **First-order-logic (FOL) programs.**[7] Functions and constraints are implemented by FOL programs, $\Pi(X)$. X is a set of parameters. Program parameters are part of the operator

[6] In this case it is an abusive example as the code attributes does not have the same domains.
[7] A FOL program designates here a set of logical rules that induces new facts.

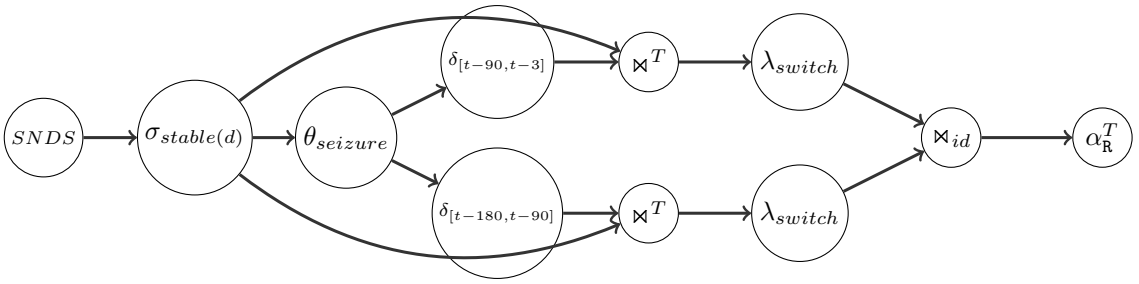


Figure 5.3: GENEPI workflow

parameters (see parameters literals in a workflow). Internally, a sequence in our data model is translated into a set of facts (with predefined predicates, we refer the reader to section C.2). An inductive logic program is used to generate events attributes or sequence attributes (e.g. for the induction operator). In such case, the expected output of the function or condition are predicates that must be assigned by Π.

The main advantage of FOL program is the versatility: it is easy to adjust the operator behavior to the needs. But this requires much human effort and skills to implement it.

In addition, our framework also integrates data analytic operators. A data analytic operator is a terminal operator that does not output any dataset. It generates the result of an analysis of a dataset as a file or a graphic which is out of our data model. We propose two operators:

Attribute analytics ($\alpha^T_{[\mu]}$, unary). An attribute analytics operator conducts data analytics on the sequence attributes only. The set of sequence attributes is a tabular dataset that can be processed by classical data analytics tools such as `R` or SAS. μ represents the data analysis program on the sequence attributes dataset.

Sequence analytics ($\alpha_{[k,\mu]}$, unary). A sequence analytics operator conducts data analytics on the sequence of type k. It is typically a temporal pattern mining algorithm. μ represents the parameters the mining tasks: the minimal threshold, but also the constraint.

The strengths of the our system lies in the possibility to combines easily different operators to formalize a processing chain from the raw data to the data analytics. Its specificity is the nature of the data model which is centered on temporal data. The complex nature of these data leads us to define, at this stage, abstract operators. We call them *abstract* because, they do not specifies by themselves the data processing, but they offer a conceptual framework to represent processing on temporal data. Their expressiveness of the operators lies in the expressiveness of the constraints and functions that parameterize them.

The set of abstract operators represents the essential processing that can be done on dataset of sequences. Thus, any data wrangling procedure can be represented as a workflow of these operators.

B.4 Usage example: GENEPI

The Figure 5.3 illustrates the workflow corresponding to the GENEPI study (see Section B.2 of Chapter 2).[8] The workflow is made of the following operators:

- ▷ $SNDS$ is a data source representing the care pathways of patients from the SNDS data warehouse
- ▷ $\sigma_{stable(d)}$ is a selection of patients that have stable anti-epileptic treatments. The treatment is stable in case the patient had d drugs deliveries of anti-epileptic drugs within 1 year. d is a parameter of the condition ($d = 10$ in the original study). We can notice that this condition is a complex condition as it involves both temporal constraints and ontological knowledge (see Section C for more details).

[8] To focus our attention on the most interesting part of the workflow, we do not included the filtering of patients having cancer or being pregnant.

- ▷ $\theta_{seizure}$ is a filtering of the epileptic seizure events. These events are simply recognized by CIM10 codes in the sequence of hospitalization (see Section B.2, in Chapter 2).
- ▷ $\delta_{[t-90,t-3]}$ (resp. $\delta_{[t-180,t-90]}$) is a time modifier which identifies the care period (resp. control period). The case period (resp. control period) is defined as the period of time from 3 to 90 days (resp. 180 to 90 days) before a seizure event.
- ▷ $\bowtie^T$ makes the temporal join of the dataset from σ_{stable} with the case (resp. control) period. Intuitively, it selects the events that occurs within the case (resp. control) period. Note that in case of several seizure events for a patient, it generates as many individuals in the dataset.
- ▷ λ_{switch} is a labeler which create a new sequence attributes in case a switch in drugs deliveries has been found. There is a switch in case a patient had brand-named and generic anti-epileptic drug deliveries.
- ▷ $\bowtie_{id}$ gathers the sequences with the same *id*. The merge of events is not interesting in this case. The objective is to merge information generated by the labelers: For each individual, we have the information of the presence or the absence of a switch in the case and in the control period.
- ▷ $\alpha^T_{\mathtt{R}}$: finally, the sequences attributes are given to `R` that creates the contingency table (see Table 2.1) and performs a statistical test to evaluate the hypothesis of the impact of drug switches on epileptic seizures.

This example illustrates that the proposed DSL formalizes a real pharmaco-epidemiological study. In addition, it illustrates we expected benefit of the DSL. Indeed, the original study requires to make assumptions, for instance, about the notion of stable treatments for instance. The treatment is said stable while at least 10 deliveries of anti-epileptic drugs has been found. This figure has been set based on medical knowledge but may be discussed. The DSL enables to evaluate easily different values for the parameter d. This enables to make more informed choices.
In addition, this assumption may appear too simple to represent the complex notion of stable treatment. In particular, there is no temporal notion in it. The reason of this simplification is technical: with the available tools, expressing more complex patterns is a difficult task.

To empower the epidemiologist with a tool that foster the design of complex workflow, it requires to propose expressive tools for specifying complex behaviors. A complex behaviors is a constraint on the temporal sequence that can be used as parameters for the operators σ, θ or ι. The next section focuses on this problem.

Remark 11 (There no unique workflow for the same task.)**.** *The same workflow may be expressed in different manners. More especially, a selection operator $\sigma_{[\varphi]}(\mathcal{D})$ which enables to express a complex condition can be represented by a $\sigma_{[a]}(\iota_{[\varphi,a]}(\mathcal{D}))$ where ι induces a new event in condition φ, and $\sigma_{[a]}$ selects individuals having event a.*

This simple example illustrates the non-minimality of the collection of operators. Some operator may be simplified keeping the same language expressiveness. Again, our challenge is to formalize the pharmaco-epidemiological studies. The language redundancy is not our current interest.

B.5 Proof of concept implementation

We implemented our data model, the operators and workflow management system in C++. The objective of this implementation was to make a proof of concept and to experiment some simple workflow. The workflow is specified in a XML format that describes the workflow graph and the set up of each operator. Figure 5.4 illustrates the graphical user interface to visualize a workflow. Our tool is based on a simplified data model with only one type of sequence per individual and does not yet fully support variables.

The development of this tool is still on-going. Further developments are required to release a prototype that may be used out of the lab. Our tool have been implemented from scratch but future implementations may benefit from workflow management.

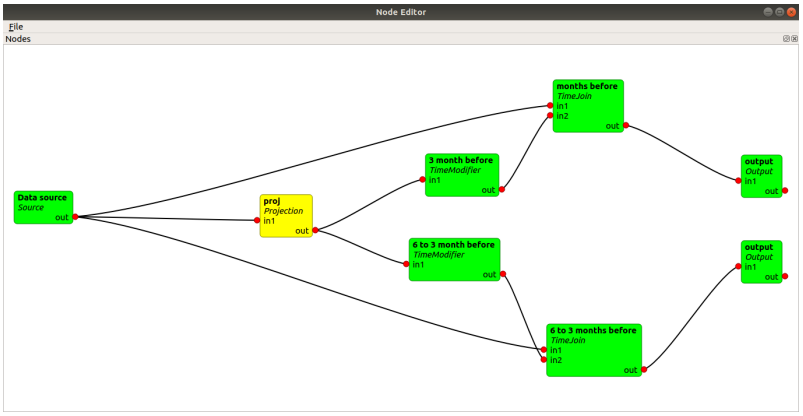


Figure 5.4: Illustration of the graphical interface of our implementation of the workflow manager for temporal sequence database analysis.

C Matching complex patterns in temporal sequences[9]

In the previous section, we defined a domain specific language (DSL) to represent queries that process dataset of temporal sequences from the raw data transformation to data analytics. This DSL is formalized as an algebra of abstract operators. Most of its operators are parameterized by constraints or functions on the temporal sequence, i.e. the care pathways. For instance, the selection operator $\sigma_{[\varphi]}$ is parameterized by a formula φ. What $\sigma_{[\varphi]}$ selects is defined by φ. Then, the semantic of a query relies on operators constraints that are concretely defined in the workflow of a query.

Intuitively, the general form of an operator constraint specifies a situation or a behavior that can be match in a sequence. For instance, in the GENEPI study, $\sigma_{stable(d)}$ selects patients having a stable treatment (i.e. a situation where a patient has at least d anti-epileptic drug deliveries in a year). The specification of a behavior occurring in a sequence is, by definition, a complex pattern. As soon as a complex pattern matches a sequence, then it activates the operator. For instance, if it is a selection operator, it selects the individual whose sequence matches the pattern.

Then, our objective is to equip abstract operators with complex pattern recognition. This requires to propose both:

1. a complex pattern language, i.e. a formalism to represent situation or a behavior to match in sequences,
2. a recognition method that will assess whether a complex pattern matches a sequence.

A complex pattern is very similar to patterns we encountered in the previous chapters: it is a structured representation of a complex behavior observed in sequences. But, the task of complex pattern matching is less computational demanding than pattern mining. More especially, there is no need to have a structured set of complex patterns because we do not search among patterns. Thus, the model may be more complex and foster expressiveness.[10]

The challenge is to have a formalism for an expressive pattern language that support clinicians to express a broad range of complex situations from care pathways. The more expressive the patterns language, the more situation it can represents, and the less matching error. Matching errors are false-positives (i.e. a care pathway that match the pattern but that does not interest the clinician) and false-negatives (i.e. a care pathway that does not match the pattern, while it should be).

Remind that the SNDS database is a secondary usage database, meaning that the clinically meaningful events are not necessarily readily available. To detect accurately some clinical events,

[9]This part of the work is the subject of the J. Bakalara PhD Thesis. This PhD is co-supervized by O. Dameron (IRISA/Dyliss), E. Oger (CHU Rennes/REPERES) and A. Happe (CHR Brest/REPERES). The work on comparing chronicle recognition models started with a collaboration with Y. Rivault (EHESP/REPERES), L. Le Meur (EHESP/REPERES) and O. Dameron (IRISA/Dyliss).

[10]This explains why we use the term “complex pattern” in this section.

epidemiologists have to built dedicated *proxies* or "algorithms" which exploit the raw data to identify the clinical events (temporal abstraction). Such *proxies* may then be validated, shared and reused as a building block for more complex workflows. For instance, the RedSiam network[11] edits a repository of validated "algorithms" to identify many diseases from the SNDS. In epidemiology, such algorithms are also called *phenotypes*, as to describe what is visible from the care pathway point of view while exposed to a drug, a disease and so on. Each of these algorithms are interesting to formalize to identify periods of exposure or periods when patient suffering from diseases, etc.

Two dimensions are significant with SNDS care pathways: the temporal dimension and the ontological dimension. All events in the SNDS database are timestamped, and the descriptions of events refer to standard taxonomies (ATC, CIM-10, etc). It is expected by the user to be able to specify temporal constraints in patterns and to abstract events using taxonomies. For instance, the definition of a stable patient in the GENEPI study requires to specify a duration in which d deliveries of anti-epileptic drugs occurred. In addition, the notion of anti-epileptic drugs is defined thanks to taxonomies to avoid enumerating all drugs names.

First-order logic is the first class citizen solution to represent constraints including temporal and ontological reasoning. The drawback to the expressiveness are the possible complexity to write patterns for clinicians and the computational efficiency. On the one hand, the objective is to empower epidemiologists to create their own patterns. The formalism has to be simple enough to be wrote and understood by non-computer scientists. These are requirements for trustworthy and fully harnessed tools. On the other hand, the computational efficiency is mandatory to have practical tools. Indeed, the evaluation of patterns is intensively called while executing a query.

Again, we face a tradeoff between our willing of expressiveness and our need for efficiency. In the following, we investigate two solutions of complex pattern:

- ▷ **an efficient but expressively limited solution** based on chronicles. Chronicle (see Section C.1 in Chapter 3) is an example of formalism to represent temporal patterns. Its temporal constraints enables to represent complex situation and it has efficient dedicated algorithms (see next section). Moreover, it has a graphical representation that facilitate the prototyping by users.
- ▷ **an expressive, but less efficient solution** based on Answer Set Programming constraints. ASP empowers users with expressive first order logic constraints and solvers such as *clingo* resolve the problem of the constraint satisfiability. But, despite the solvers efficiency, we expect to pay a computational cost of its genericity.

C.1 Chronicles recognition

A chronicle is a description of a complex pattern occurring in a sequence. The chronicle recognition assesses a sequence contains a chronicle or not.

Chronicle recognition handles sequences of timestamped events. But, in our data model, an individual is made of a collection of sequences with events qualified with temporal intervals. This means that it uses a simplified view on the model of sequence. To fit chronicle recognition:

- ▷ the duration of any event is ignored and
- ▷ one of the event attributes is chosen to serve as sequence item, then the event type of a sequence is defined as a pair (t, a) where t is the sequence type and a is the attribute value.

Example 35 (Chronicle for stable patient)**.** *Figure 5.5 represents a chronicle which recognizes a sequence of at least 10 deliveries of the anti-epileptic drug $N03AX09$ within a period of a year. Each pair of successive deliveries are not more distant than 62 days (2 months). To recognize this chronicle in a sequence with the data model presented in Example 26 page 103, the* code *attribute of the drugs sequence is chosen as item.*

Remark 12. *The limitations on the sequence format can be mitigated thanks to processing workflows.*

Chronicles have been introduced in Section C.1 of Chapter 3). In the remainder of this section, we present an efficient algorithm to match (or to recognize) a chronicle in a sequence. A sequence

[11] https://www.redsiam.fr

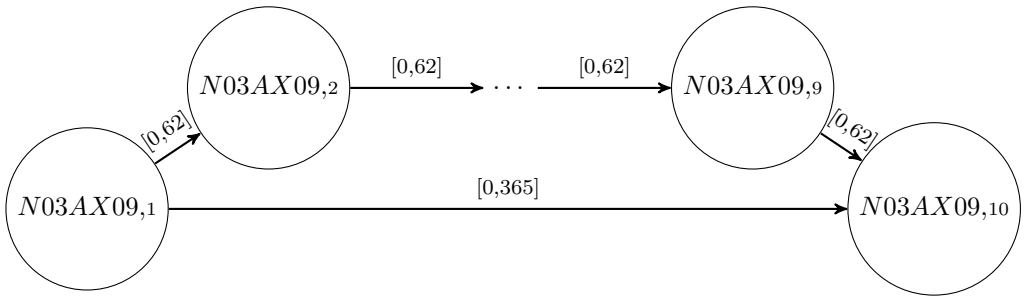


Figure 5.5: Chronicle representing a patient with stage anti-epileptic treatment.

of length n is a sequence of n timestamped events denoted $\langle (e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n) \rangle$. The formal definition of a chronicle $\mathscr{C} = (\mathcal{E} = \{\!\{e'_1, \ldots, e'_m\}\!\}, \mathcal{T})$ is given in Definition 23 page 39. The definition below reminds the notion of chronicle occurrence (see Definition 25 in Section C.1 in Chapter 3).

Definition 41 (Chronicle occurrence)**.** *An occurrence of a chronicle* $\mathscr{C} = (\{\!\{e'_1, \ldots, e'_m\}\!\}, \mathcal{T})$ *in a sequence* $\boldsymbol{s} = \langle (e_1, t_1), \ldots, (e_n, t_n) \rangle$ *is a subset* $\{(e_{f(1)}, t_{f(1)}), \ldots, (e_{f(m)}, t_{f(m)})\}$ *of* $\boldsymbol{s}$ *such that*

1. $f : [m] \to [n]$ *is an injective function,*
2. $f(i) < f(i+1)$ *whenever* $e'_i = e'_{i+1}$,
3. $e'_i = e_{f(i)}$ *for* $i = 1, \ldots, m$,
4. $t_{f(j)} - t_{f(i)} \in [t^- : t^+]$ *whenever* $(e'_i, i)[t^- : t^+](e'_j, j) \in \mathcal{T}$.

The chronicle $\mathscr{C}$ occurs *in* $\boldsymbol{s}$*, denoted* $\mathscr{C} \preceq \boldsymbol{s}$*, iff there is at least one occurrence of* $\mathscr{C}$ *in* $\boldsymbol{s}$.

We can now define two specific tasks: the *sequence adherence* and *occurrence enumeration*. In the first case, we are just interested in identifying whether a chronicle occur in a sequence, but in the second case, we are interested in finding out all possible occurrences of a chronicle in a sequence. For some abstract operators (selection, labeler) sequence adherence seems suitable, but occurrence enumeration may be required for some other abstract operators (induction, filtering).

Definition 42 (Sequence adherence)**.** *Let* $\boldsymbol{s}$ *be a sequence and* $\mathcal{C}$ *be a chronicle. Sequence adherence answers the question "does chronicle* $\mathcal{C}$ *occur in* $\boldsymbol{s}$*?".*

Definition 43 (Occurrence enumeration)**.** *Let* $\boldsymbol{s}$ *be a sequence and* $\mathcal{C}$ *be a chronicle. Occurrence enumeration outputs all possible occurrences of* $\mathscr{C}$ *in* $\boldsymbol{s}$.

Note that a single sequence is processed by a single chronicle. For some application, it is interesting to find occurrences of several different patterns at the same time. In such case, specific techniques to improve the overall computing time may be used. For instance, if two chronicles patterns have a sub-chronicle in common, using hierarchical chronicles recognition [Dousson and Maigat, 2007] may be efficient. Similarly, when dealing with large datasets, specific strategies may be proposed to lower the memory usage to tackle very large dataset of sequences.

The second specificity compared to classical chronicle recognition tools is that we are not dealing with a stream of events but a static sequence. This enables to process events in a favorable order for efficiency instead of being constrained by the temporal order of continuously incoming events.

Algorithm 8[12] presents a chronicle enumeration algorithm [Guyet et al., 2020]. Its principle is to refine progressively the intervals in which chronicle events $e \in \mathcal{E}$ may occur in the sequence $\boldsymbol{s}$. These intervals are so called *admissible positions*. The algorithm transverses the set of events $e \in \mathcal{E}$ and propagates the temporal constraints of the chronicle to narrows intervals until intervals has a singleton intervals which are an occurrence of the chronicle. Algorithm 8 makes recursive calls of the Algorithm 9 which assumes that the $k-1$ first event types have been located in the sequence $\boldsymbol{s}$. This means that the k first intervals of the admissible intervals adm are singleton intervals. The recursive call looks for event type c_k in the admissible part of sequence for k-th event (lines 4-5). If found, it is a candidate for further refinements and temporal constraints of the chronicle

[12]This algorithm is available in Python as a pip package: https://pypi.org/project/pychronicles/.

Algorithm 8: Occurrences of a chronicle $\mathscr{C}$ in a sequence $\boldsymbol{s}$.

Input: $\mathscr{C} = (\mathcal{E} = \{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T})$, $\boldsymbol{s} = \langle (e_1, t_1) \ \ldots \ (e_n, t_n) \rangle$
Output: *occs*

```
1  adm ← {[−∞,∞],...,[−∞,∞]} // Vector of m intervals
2  occs ← ∅ // Set of occurrences
3  foreach (e,t) ∈ s do
4      if e = c_1 then
           // create a new occurrence
5          o ← adm;
6          o_1 ← [t,t];
           // propagate chronicle constraints
7          foreach (c_1,1)[t^−,t^+](c,p) ∈ T do
8              o_1 = [max(0,t+t^−), min(t+t^+|p|−1)];
9              loccs ← RecRecognition(o,1,C,s) ;
10             loccs ← occs ∪ loccs;
11 return occs
```

are propagated. The constraint $(c_k, k)[t^-, t^+](c, p)$ is a constraint from (c_k, k) event to the event type c at position p in the multiset. It is used to possibly narrow the admissible interval of event p (line 10). In case the new interval is inconsistent (line 11) then this candidate occurrence can not satisfies the temporal constraints and it is discarded (*satisfiable* is set to $false$). If the candidate occurrence is satisfiable, the recursive call attempts to refine further this occurrence (line 20). Note that only forward constraints are propagated. Indeed, backward constraints (i.e. constraint to event at position lower than k in the multiset) have already been taken into account in parent calls.

Example 36 (Algorithm example)**.** *Let* $\boldsymbol{s} = \langle E\ B\ C\ E\ A\ B\ C\ E\ C\ C \rangle$ *and* $\mathscr{C} = (\{\!\{A, B, C\}\!\}, \{(A,1)[-2,2](B,2), (A,1)[-3,5](C,3), (B,2)[-1,3](C,3)\})$.

1. *Processing of event* A
   - ▷ *generates a single occurrence* $o = ([5,5], [-\infty,\infty], [-\infty,\infty])$
   - ▷ *constraints propagation:*
     - $(A,1)[-2,2](B,2)$*:* $o = ([5,5], [3,7], [-\infty,\infty])$
     - $(A,1)[-3,5](C,3)$*:* $o = ([5,5], [3,7], [2,10])$
2. *Processing of event* B
   - ▷ *narrow intervals with occurrences:* $(B,2)$ *is invalid* ($2 \notin [3,7]$)*, but* $(B,6)$ *satisfies the admissible interval* $[3,7]$ *so the admissible intervals can be updated* ($o = ([5,5], [6,6], [2,10])$)
   - ▷ *constraints propagation:*
     - $(B,2)[-1,3](C,3)$*:* $o = ([5,5], [6,6], [2,10] \cap [5,9]) = ([5,5], [6,6], [5,9])$
3. *Processing of event* C
   - ▷ *narrow intervals with occurrences:* $(C,3)$ *and* $(C,10)$ *are invalid, but* $(C,7)$ *and* $(C,9)$ *are valid, then the two occurrences of the chronicle are obtained by updating the admissible intervals* ($([5,5], [6,6], [7,7])$ *and* $([5,5], [6,6], [9,9])$)*.*

Algorithm 8 presents the main principle of chronicle recognition in a sequence. It could be improved by simple heuristics that exploit the characteristics of sequences and chronicles. Indeed, the algorithm efficiency is impacted by the order in which chronicle events are processed. In the above example, a recognition starting by event C would be more resource consuming as the initial number of candidate occurrence would be larger. A first heuristic is to order the event by their frequency in the sequence in order to start by less frequent events. This yields less candidates and prevent for heavy memory consumption. A second heuristic is to select dynamically the next event to proceed that yield less candidates knowing the admissible interval. Counting them would not be efficient, but it can be estimated by $c(e_i) \times (t_i^+ - t_i^-)$, where $c(e_i)$ is the count of event e_i in the sequence. Nevertheless, our experiments shown that simplest heuristics outperforms the more complex heuristics on synthetic datasets [Guyet et al., 2020].

To conclude, chronicle recognition is an interesting solution to represent complex behaviors to

Algorithm 9: RecRecognition$(o, k, \mathscr{C}, \boldsymbol{s})$.

Input: *adm*: admissible intervals, k: recursion level, $\mathscr{C} = (\mathcal{E} = \{\!\{c_1, \ldots, c_m\}\!\}, \mathcal{T})$, $\boldsymbol{s} = \langle (e_1, t_1) \ \ldots \ (e_n, t_n) \rangle$

Output: *occs*

```
occs ← ∅ // Set of occurrences
if k = m + 1 then
    return adm
foreach (e, t) ∈ s s.t. t ∈ adm_k do
    if e = c_k and then
        // create a new occurrence
        o ← adm;
        o_k ← [t, t];
        // propagate chronicle constraints
        satisfiable ← true;
        foreach (c_k, k)[t⁻, t⁺](c, p) ∈ T do
            o_p ← [max(o_p⁻, t + t⁻), min(o_p⁺, t + t⁺)];
            if o_p⁻ > o_p⁺ then
                satisfiable ← false;
                break;
        if satisfiable then
            // Recursive call
            loccs ← RecRecognition(o, k + 1, C, s) ;
            loccs ← occs ∪ loccs;
return occs
```

recognize in sequences. But, its expressiveness has some limitations: it can not handle several attributes of event sequences neither combining events from different types; it does not deal with taxonomies; its temporal relations are limited to delay intervals; there are no specification of absent events. Our intuition is that the model may be enriched with at least taxonomy and complex events without loosing the recognition efficiency. These extensions are part of our perspectives.

C.2 Complex patterns in ASP

The use of ASP is a specific case of using a FOL program to specify operator conditions φ. In principle, φ is a FOL formula. The condition φ is satisfied for a sequence S iff $S \models \varphi$.

With ASP programs, the condition φ is satisfied on a sequence if the ASP program $\phi \cup \mathcal{S} \cup \mathcal{B}$ has answer sets where ϕ is ASP program encoding ϕ, $\mathcal{S}$ is the set of ASP facts encoding the sequence and $\mathcal{B}$ is the formalized background knowledge (e.g. taxonomies).

Let us start with the encoding of a sequence. Let $S = (id, \boldsymbol{v}, \boldsymbol{s}_1, \ldots \boldsymbol{s}_r)$ be an individual of the dataset $\mathcal{D}_{[c_1,\ldots,c_n],\Re}$. The sequence state, denoted $\mathcal{S}$, is represented by a set of ASP facts:

- ▷ $\mathtt{c}_i(v_i)$ which represents the value v_i for attribute c_i for all $i \in [n]$. c_i is a predicate.
- ▷ $\mathtt{seq}(t^-, t^+, \mathtt{R}_k(v_1, \ldots, v_{m^k}))$ for each event $s = ([t^-, t^+], [v_1, \ldots, v_n]) \in \boldsymbol{s}_k$ where $[t^-, t^+]$ is the temporal interval of the event, v_j is the value of the j-th event attribute, $j \in [m^k]$ and $\boldsymbol{s}_k$ is a sequence of the type R_k.

Example 37 (ASP representation of the care pathway of Example 26 page 103)**.** *The set of facts of the first patient is:*

```
sex(m). by(1980). zipcode(35700).
seq(0,0, drug(c01da02,1)).
seq(10,10, drug(r01ad52,3)).
seq(31,31, drug(c07aa07,1)).
seq(8,8, act(hnca006)).
seq(35,39, hosp(k350)).
```

```
% find at least 10 delivery dates
10{ deliveries(T) : seq(T,T,drug(n03ax09,_)) }.

% two deliveries date may never be more distant than 365 days
:- deliveries(T), deliveries(T2), T2>T+365.
```

Listing 5.1: Recognition of stable patients ($d = 10$)

```
% mapping of sequence events with chronicle states
1{ map(X,T) : seq(T,_,S) } 1 :- state(X,S).

% satisfaction of transitions defined for some pairs of events
:- transition(X,Y,LB,UB), map(X,T1), map(Y,T2), T2>UB+T1.
:- transition(X,Y,LB,UB), map(X,T1), map(Y,T2), T2<LB+T1.

% two states corresponding to the same event S have to be ordered
:- state(X,S), state(Y,S), X<Y, map(X,T1), map(Y,T2), T1>=T2.
```

Listing 5.2: Chronicle recognition in ASP

Note that the *id* of the sequence does not appear in the facts. Pattern matching is independent task from one sequence to another. Thus, a program never hold information from several sequences at the same time and a new resolution is ran for each sequence.

Beside the ASP encoding of a sequence, the operator condition, φ, is also encoded by a set of ASP rules, denoted ϕ. ϕ can specify temporal constraints but also use taxonomies classes to specify constraints on event attributes. It can combine constraints on several event types and also absent event. Then, all missing features of chronicles can easily be encoded with ASP constraints. For instance, Listing 5.1 encodes the stability constraint to select patients in the GENEPI study. This encoding is both the situation description and the method to find it in sequences.

Writing constraints in ASP may be difficult for domain expert, but generic pattern recognition programs can be implemented in ASP and benefit from advanced reasoning capabilities of ASP. Then, an expert has only to provide a pattern description. For instance , we developed the generic chronicle recognition ASP program (see Listing 5.2). Then, a domain expert simply has to specify a chronicle to recognize. For instance, Listing 5.3 is the ASP encoding of a chronicle that can be recognize by Listing 5.2. A chronicle is encoded by two predicates: `state/2` for the event multiset and `transition/4` for the temporal constraints (lower and upper bounds for each pair of state). The chronicle recognition program looks for potential mappings of the chronicle events to the sequence events and it assesses the temporal constraint satisfiability.

```
state(0..10,drug(n03ax09,1)).
transition(X,X+1,0,62) :- X=0..9.
transition(0,10,0,365).
```

Listing 5.3: ASP encoding of the chronicle of Figure 5.5 for recognition by program in Listing 5.2.

The next Section compares the efficiency of the two chronicle recognition strategies. It evaluates the computational cost of a declarative strategy for complex pattern recognition compared to a dedicated procedural approach.

C.3 Efficiency comparison

This section presents an experiment on the efficiency between three strategies to recognize chronicles: a procedural algorithm (see Section C.1), a constraint solving approach (see Section C.2) and a web semantic approach using SPARQL. The details of the encoding of chronicle recognition with SPARQL can be found in [Rivault, 2019].

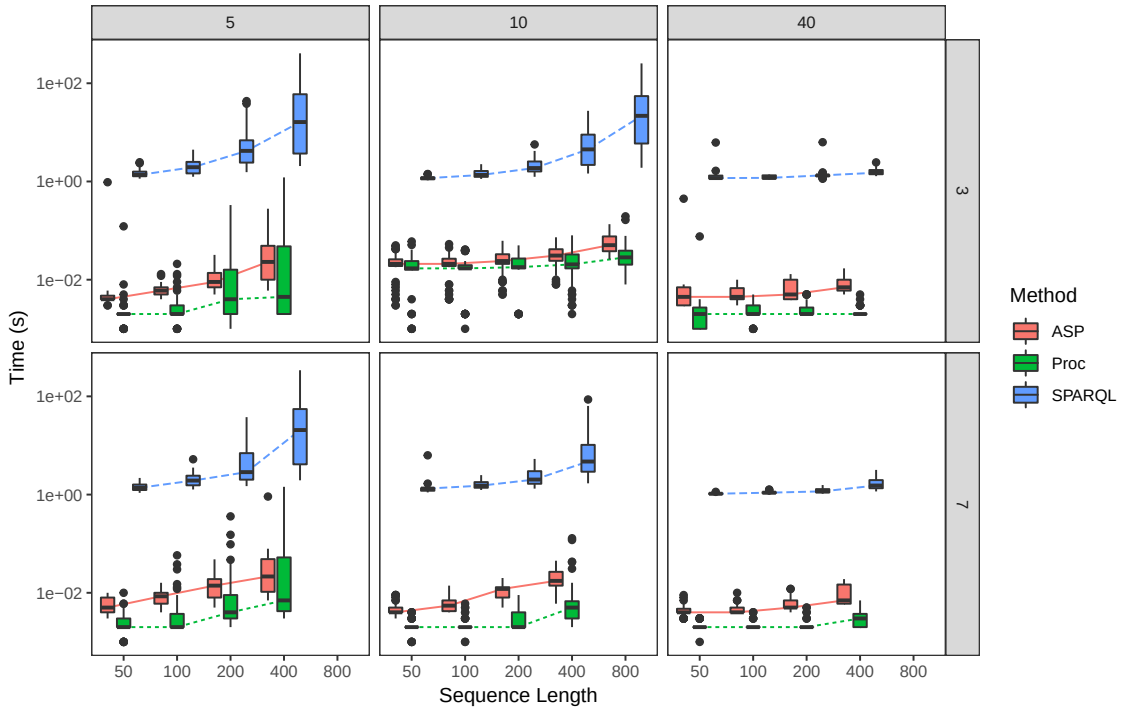


Figure 5.6: Computation times for chronicle recognition with respect to sequence length. From left to right, the plots compares the computing time with different vocabulary size (5, 10 and 40). Upper plots give computing times with chronicles having temporal constraints with a mean interval duration of 3 or 7.

In this experiment, we generated random sequences of length m with an event type vocabulary of size v ($v = 5$, 10 or 40 and m goes from 20 to 800). The event types are randomly drawn using a equiprobable multinomial distribution. The event dates are generated using a uniform distribution of the delay between successive events. We also generated random chronicles. The multiset of a chronicle of size $n = 4$ is randomly drawn according to an equiprobable multinomial distribution on the vocabulary. The lower bound of a time constraint is generated randomly with the same uniform distribution as the sequence delays. The upper bound is randomly drawn such that the interval duration is a normally distributed around 3 or 7. Chronicles with inconsistent constraints are discarded.

Figure 5.6 presents computing times of the three chronicle recognition systems. It illustrates the influence of three parameters for all approaches: sequence length, vocabulary size and extent of the chronicle temporal constraints. At a glance, we see that the longer the sequence, the longer it takes; the smaller the vocabulary, the longer it takes; and the larger the chronicle intervals, the longer it takes. Comparing the approaches, the procedural approach is unsurprisingly the most efficient. The less efficient approach is SPARQL. Note that, for fairness, the times evaluated for SPARQL only takes into account the query execution but not the graph building. ASP is less efficient than the procedural approach, but the difference is not so important.

These results confirm our initial intuition that matching patterns is more efficient with dedicated algorithms than with generic tools. Today, we did not identified a class of patterns that has the expressiveness for representing a significant range of the constraints requested by epidemiologist. Chronicles are a good candidate, but have to be enhanced with more features (taxonomies, absence of events). Then, a solution that enables to encode first order logic constraints is still reasonable. Then, the solution based on ASP turns out to be the best compromise between expressiveness and efficiency.

D Conclusions and Perspectives

In this section, we formalized epidemiological studies as a data science workflow, and more especially a workflow to proceed temporal sequences databases. This framework is based on 1) an algebra to manipulate datasets of sequences and 2) complex pattern recognition tools. We illustrated with chronicles that complex patterns can be recognized efficiently with a procedural algorithm but also with ASP programs. This latter solution improves the expressiveness.

Our initial motivation was to support epidemiologists to conduct studies using a medico-administrative database. Our formalization enables to represent complex epidemiological studies such as the GENEPI study, but many more studies may be represented in our framework. Our objective is now to demonstrate the expressiveness of our proposal to represent a wide range of epidemiological studies. More than just representing studies, we proposed a computational model and implemented a tool to execute the workflow. This tool is still experimental but it is a proof of concept for our approach.

We think that this approach is original and that it may improve significantly both the efficiency of the conduction of epidemiological studies and the accuracy of these studies by proposing tools that better exploit the data. It also sheds light on the need for tools to support data preparation (data wrangling) especially with temporal sequences.

In the following, we start by highlighting that this Chapter is strongly related to the case study of the previous Chapter. Then, we proposes some perspectives of this work.

D.1 Continuum between declarative and procedural approaches

The more declarative, the more expressive but the less computational efficient. This universal law is again verified for developing tool to analyze temporal sequences.

In the proposed framework, the algebra on sequences datasets is the backbone of the framework, and algebra operators are equipped with temporal constraints. We illustrated how to represent temporal constraints in procedural or declarative manners. The declarative manner enables to propose the expression of more flexible and expressive constraints but is less computationally efficient than using procedural chronicle constraints.

But the continuum of our solution from the procedural to the declarative approach can be extended to the ASP framework we presented in the case study of the previous chapter (see Section D of Chapter 4) Indeed, there are strong connections between our domain specific query language and the combination of ontological reasoning and mining to analyze care pathways. In the previous chapter we adopted the view of mining mediated by ontologies. Another way to see the same framework is to say that the ontology mediated mining task, $\langle \mathcal{D}, \mathrm{S}, \mathcal{K}, \mathrm{M}, \mathrm{C} \rangle$, represents a workflow. Rules from S represents the pre-processings steps to induce interesting features of a sequence and C are preprocessing rules. This is illustrated by encoding the GENEPI study in the two frameworks: the declarative and the procedural. In both case, we implemented the transformation of the raw data into sequences to be analyzed by sequential pattern mining tools. The declarative framework has less restrictions on what can be expressed that the procedural framework.

The pure ASP encoding of the data wrangling, with pure ASP temporal constraint is illustrated in the case study of the previous chapter. But, there is some possibilities to improve its efficiency by using propagators if some constraints are more especially interesting to have in the framework. For instance, the chronicle recognition algorithm could be used as a propagator such that ASP program may request the use of an efficient *global* constraint about a chronicle recognition. Beyond the potential improvement of time efficiency, it may improve the memory usage of ASP solver. Today, memory usage is the main limitation to the use of ASP. Then, a perspective would be to benefit from the recent research on lazy grounding [Taupe, 2018], even if we do not think that it could be a reasonable solution to scale.

This continuum of solutions to analyze temporal sequences databases is illustrated in the Figure 5.7. The more left, the more declarative.

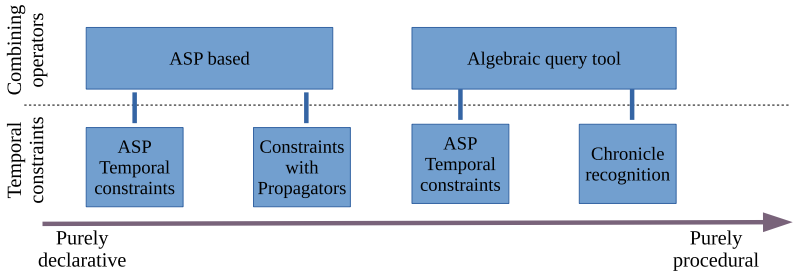


Figure 5.7: Illustration of the continuum of analyzing temporal dataset.

D.2 Need for workflow management systems

The approach we propose can be seen as a workflow management system for knowledge discovery from sequential databases. It may benefit from researches on workflow management systems, especially for scientific applications [Liu et al., 2015b] and life science applications [Cohen-Boulakia et al., 2017].

Investigate workflow management systems may improve our approach in three directions:

- ▷ **scaling up.** As our ambition is to process the very large SNDS database (with several millions of patients), the tools we developed require too much memory to scale up and have too large response time. Workflow management systems enable parallel execution of data-intensive workflows and exploit resources distributed in different infrastructures such as grid and cloud.
- ▷ **reusing part of the workflows.** Our second ambition is to ease the work of epidemiologists. And one way to support them is to provide tools to help them to create workflows. Scientific workflow management systems promote workflow sharing for reproducibility of the research results but also for reusing parts of workflows. As the SNDS database is a unique data source that aims to be used for a wide range of epidemiological studies (but also for other types of studies), we expect to see common needs shared by several users. Then, it would be a benefit to be able to share some parts of the workflow.
- ▷ **automatic workflows optimization.** At the cross-road between the workflow management system and auto-ML [Hutter et al., 2019], our promise was to empower epidemiologists with tools that can explore automatically alternative workflows to identify which set up make speak the data. At the time, we are able to handle some variables, but the workflow execution is not good enough to run efficiently an automatic exploration of variables. More especially, all operators must be ran again for each variable values even if variables have no impact on the operator output. Using workflow management systems may benefit our approach with smarter execution strategies.

D.3 Toward a deeper exploration of temporal constraints

The temporal constraints proposed in our framework illustrates how workflow operators may be enriched by complex temporal constraints. The expressivity of our framework is finally more based on these temporal constraints than on the operator themselves. For instance, the expressivity of the operator of patient selection (σ_φ) lies in the possibility to define complex selection criteria, including temporal criteria, as illustrated by chronicles.

The more expressive are these criteria the better. But, from the computational point of view, criteria have to be efficiently computed and, from the usability point of view, it has to be easily expressed by users.

Our main perspective is to explore temporal constraints to propose solution that could be a good compromise between efficiency, expressiveness and usability.

This problem is currently explored by J. Bakalara for care pathways. She studies the problem of specifying complex behaviors to recognize in temporal data in care pathways. Specifying care pathways requires to manipulate: temporal concepts (time constraints and time window), medical concepts and knowledge (ontologies). The ideal formal framework should capture these dimensions,

enable intuitive queries to be expressed for a wide range of pharmaco-epidemiological studies and be computationally efficient. It is of paramount importance to base choices on solid theoretical foundations. Expressiveness and efficiency are known to be antagonist objectives [Levesque, 1986]. A well-founded approach would be the basis for proposing long-term solutions, make possible future improvements and facilitate its application to a broad range of contexts (i.e., various databases, queries). Bakalara [2019] identified different research lines to address this problem:

- In the field of model checking [Clarke Jr et al., 2018], the dynamic systems and their properties are formalized with temporal logics such as LTL, CTL or MTL (temporal LTL). This logics may be associated with dedicated discrete event models such as timed Petri Net or timed automata. Our problem is then a problem of satisfiability of a formula in a trace.
- In the field of Complex Event Processing (CEP) [Giatrakos et al., 2017], streams of of temporal events are processed to detect or to locate complex events (or *patterns*) defined by the user. This domain defines formalisms that aim at being very efficient to process streams and expressive to specify patterns. Temporal constraint networks [Cabalar et al., 2000] or Chronicles [Dousson and Maigat, 2007] are simple temporal models that are interesting for their graphical representation, but are limited to simple relational events. Some more complex formalisms, e.g. ETALIS [Anicic et al., 2011] or logic-based event recognition [Giatrakos et al., 2017], propose very expressive representations of complex events, including reasoning techniques (including ontologies) which enrich the capabilities of CEP. In our context, care trajectories are logs, and care pathways are the complex events. We are not interested in the stream dimension of these approaches, but their formalisms to represent complex events may be adapted in the context of static logs.
- Temporal databases [Snodgrass, 1986] extends the notion of database to timestamped data. Databases issued data representation problems but also specific querying language problems. We gather in this family the temporal extension of relational databases (e.g. TSQL) but also web semantic approaches which combine query languages (e.g. SPARQL) and expressive description languages. Care trajectories are facts in the temporal database and the querying of care pathways becomes a problem of specifying care pathways in the query language. Rivault et al. [2019] shown that semantic web is an interesting approach for our problem, but does not explicitly address the problem of temporal queries.
  As we tackle our problem through a formal approach, we are interested in Description Logic (DL). DL contains the notion of ontology that may be useful to designate easily classes of events. More especially, the temporal extension of DL [Artale et al., 2017] are of particular interest in our problem. For instance, OConnor et al. [2009] developed a tool supported by OWL for research data management with a temporal reasoning in a clinical trial system.
- Knowledge Representation and Reasoning (KR) [Levesque, 1986] is "the study of how what we know can at the same time be represented as comprehensibly as possible and reasoned with as effectively as possibly". In this research domain, temporal KR is focused on representing and reasoning about time. It gives rise to several logics Long [1989], for instance: Allen's logic, McDermott's logic, Event Calculus or Halpern & Shoham's logic. KR is, by definition, the most generic framework to study how to represent care trajectories and how to represent querying of care pathways as a reasoning.

Today, the chronicle models turns out to have a lot of advantages (graphical representation, interpretability and metric temporal constraints), but miss some essential features for epidemiologist. Then, our short term perspective is to extent chronicles (and it recognition algorithm) with new features such as taxonomies and absent events.

Chapter 6

Conclusions and perspectives

This manuscript presented some research directions to enhance sequential pattern mining for analyzing complex logs of dynamical systems. In the preliminary section, we introduce sequential pattern mining and we highlight some of its limitation to extract accurate sequential patterns. These limitation are: 1) a poor exploitation of the metric temporal nature of temporal sequences, 2) the difficulties to integrate expert knowledge to guide the mining toward meaningful patterns, and 3) the need for time consuming and tedious data wrangling to transform raw data into data that sequential pattern mining can exploit.

Our proposal is to enhance sequential pattern mining with time and reasoning:

- ▷ In Chapter 3, we proposed to better take into account the temporal nature of the data by proposing new temporal pattern domains and algorithms to extract such patterns from set of temporal sequences. Temporal patterns describe metric temporal constraints that are more informative and more accurate. For the occurrences of a unique sequential pattern $\langle A\ B\rangle$, our approaches may extract several more specific patterns, e.g. $\langle A \xrightarrow{[2,45]} B\rangle$ and $\langle A \xrightarrow{[30,78]} B\rangle$, that may be meaningful to better understand two underlying behaviors of the system.
- ▷ In Chapter 4, we addressed the same objective but through reasoning. The approach we developed is in the vain of declarative pattern mining. We explored Answer Set Programming as declarative language to encode sequential pattern mining (and many of its variants) and to formalize expert knowledge. In this approach the pattern domain remains identical (in most cases), but the capabilities of the mining procedure to take into account complex constraints enriches the extracted pattern. It is not only a frequent sequential pattern (i.e. the description of a system behavior), but a sequential pattern satisfying constraints (i.e. the description of a behaviors that occurs under certain conditions). Thus, it enhances the semantic of sequential patterns.
- ▷ In Chapter 5, we proposed an approach for temporal data wrangling that unleashes the possibility to exploit complex databases, and more especially for secondary usage temporal data. Secondary usage data are readily available for making them speak by data analytics, but they have to be carefully prepared. Our approach consists in having reasoning (data abstraction, data induction, etc) prior to the application of data analytics.

The application to care pathway analysis serves as a running example. This application originates from the ANSM/PEPS project which aims at proposing tools to support epidemiologists to conduct pharmaco-epidemiological studies using the SNDS, the medico-administrative database of french insured people.

This application illustrates the general context of our problem: experts (epidemiologists) collect or access data that contain longitudinal information about a system (the patients). Our general context is to support the expert to understand the functioning of the system by analyzing its logs. In addition, the nature of the SNDS data (medico-administrative databases) highlights the interest of 1) pattern mining approaches, 2) enriching patterns with temporal information and 3) to have reasoning to make pattern mining capable of taking into account the available formalized medical information such as taxonomies and ontologies. The use of pattern mining is motivated

by the interactive approach of pharmaco-epidemiology [Happe and Drezen, 2018]. Epidemiologists have a clear need to understand what are the care pathways of groups of patients. Then, they used to manipulated care pathways to make them speak. In this context, patterns turn out to be interesting thanks to their interpretability and their possible use both as a nugget of knowledge extracted from the data or as a formalized knowledge describing a complex situation. Then, the temporal dimension of patterns is very meaningful, because having drugs exposure few days before a disease has not the same meaning that a month before.

The data analysis situation that is described is encountered in our different applied contexts, and the same approach developed in our work applies. The exact same experiments that have been conducted in each Chapter may be conducted in each of the concrete application introduced in the introduction: understanding the habits of customer through purchase analysis (Example 1), capacity planning of a server cluster (Example 3) or intrusion detection (Example 4).

Cross-cutting remarks

The pattern domain of **chronicle turns out to be a corner stone** of the different approaches:

- ▷ Chronicles model complex temporal situations that can be match in temporal sequences. They offer an interesting tradeoff between expressiveness and efficiency. Then, it can be used as a complex pattern to specify situations to recognize in temporal sequences, as we did in Chapter 5. The perspective we had is to extend chronicles with different types of constraints (e.g. absence of events) to enhance its expressiveness.
- ▷ Chronicles are interpretable. Thanks to their graphical representation, expert domain can easily manipulate them. Nonetheless, we warn the potential users about the intuitiveness of chronicle. The temporal constraints may sometime be difficult to handle and even be counterintuitive.
- ▷ Chronicles can be efficiently extracted by pattern mining algorithms. Cram et al. [2012] give a complete and correct algorithm to extract frequent chronicles, and we derived new strategies to extract discriminant ones.
- ▷ Chronicles are naturally suitable for constraint programming approaches. Indeed, chronicles is defined as a set of constraints over occurrences of a multiset. Then, matching chronicle is essentially checking the constraints satisfaction and the declarative pattern mining approaches we developed in Chapter 4 for sequential pattern mining may be adapted for chronicles.

In two of our problems, we investigate the tradeoff between, on the one side, the versatility and expressiveness of declarative approaches; and, on the other side, the efficiency of procedural approaches. These two different approaches have been used to develop sequential pattern mining or sequential data wrangling.

The versatility and expressiveness of declarative approaches have also to be balanced with the usage simplicity. The declarative approach such as ASP is interesting to formalize problems and to prototype quickly solutions. The problem is that the ideal of the declarative programming, which is to simply encode the problem, is more technical in practice. Then, from a practical point of view, we do not believe that pure declarative programming is suitable. It has to be curbed to became practical. The difficult question is to decide which part of the solution has to be confined and which part can be unleashed. This is the core question of this tradeoff.

Even if, by answering this question, we have solution for a procedural encoding. ASP remains very interesting to investigate the possible tradeoff. The versatility of ASP programs is very useful to quickly prototype alternative tasks in few lines, and to evaluate their potential.

In our past work, we propose opposite solutions and in the future a fine tuning of the tradeoff has to be investigated: procedural code in declarative frameworks, or declarative code in a procedural framework.

Some perspectives

The field of pattern mining was very active in the 2000s, but the rise of supervised learning methods and the difficulties encountered by pattern extraction methods have reduced their interest. Two

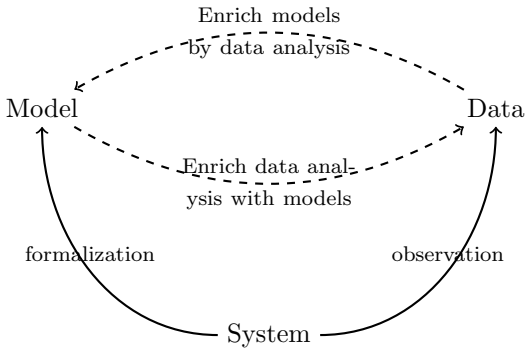
of these difficulties can be mentioned:

- ▷ the deluge of patterns: frequent pattern mining algorithms extract a lot of patterns, sometime more than the number of examples. Then, experts have to identify those that are really relevant. This task is tedious and limits the interest in the method.
- ▷ the operationalization of the extracted patterns, i.e. the use of patterns in automatic tools after their extraction. For instance, patterns may be used to make some prediction, to optimize system trajectories or other kind of reasoning (planning).

We would like our methods to extract patterns that are insightful for users and usable in automatic data analysis tools without requiring too much human resources.

The main direction of my research aims to provide solutions to these problems by **developing methods combining formal reasoning and algorithmic data mining**.

The data analysis community (machine learning, data mining) has traditionally focused on inductive learning from partial (potentially noisy) observations of a system (i.e. data). The formal methods community has traditionally favored automated deduction, for example by using automatic reasoning methods, theorem proving or model checking as the main methods of exploring how system works, with a particular emphasis on formal models and proof of their accuracy. The illustration below illustrates these two visions of exploring a system.



The future of pattern mining lies in the complementarity between data mining, which exploits logs of a system, and formal reasoning, which exploits formalized knowledge. Where machine learning approaches successfully exploit data at a signal level (e.g. images, signals, time series), pattern mining and formal reasoning methods (planning, combinatorial optimization, argumentation, etc.) process data at a high level of semantic. This semantic level makes these approaches more interpretable by users. As our objective is to support experts understanding, we believe that investigating such techniques is promising.

A decade of research in data science shown that data analytics and constraint programming are cross-fertile areas. It has been initiated with ILP (Inductive Logic Programming) and now declarative pattern mining approaches benefit from modern solving techniques (e.g. conflict-driven clause learning used in SAT or CP solvers). Today, there is space for research aiming to make declarative pattern mining even more viable and attractive. The first step is to deeply integrate reasoning and pattern mining to meet new tasks that combine the strengths of a system model and the potential of event logs.

In one direction, it aims to provide solutions to use pattern mining to address ambitious questions involving formalized knowledge. While pattern mining uses statistical criteria to specify pattern to extract (*frequent* patterns, *discriminant* patterns), our ambitious is to extract patterns *which optimize the system trajectory* or *that can be used as a counter-arguments to an attack*. For these examples, the learning objective is guided by reasoning and exploits pre-existing knowledge about the system. This is potentially a solution to the deluge of patterns.

In the other direction, data analytic may be helpful for reasoning tasks (argumentation, planning, optimization). Most of the reasoning technique use a system model but no massive data. Then, the open question is how data analytics may be part of reasoning.

In the following, we give three concrete research directions toward the reconciliation of mining and reasoning.

Enrich data analysis through the use of formal models The research directions we propose address two sub-problems independently.

- ▷ **Develop the notion of "Epistemic measure of interestingness"**. This research axis is in line with the collaboration between reasoning and pattern mining. The notion of "subjective measure of interestingness" is developed in the ERC of De Bie [2013] or by Silberschatz and Tuzhilin [1995] as a way to prevent the deluge of patterns by means of a statistical model. The statistical model formalizes expert knowledge about the system. Patterns that does not fit this model are said "novel" and are preferred in the mining process. Our research axis aims to develop a parallel approach based on a formal model of the system. The idea is that patterns that are not "explainable" by the formal model are potentially new or particularly interesting. Reasoning about knowledge is used to evaluate the interestingness of a pattern and that is why we called this approach "Epistemic measure of interestingness". It provides an original entry point for deeply integrating knowledge-based reasoning into pattern mining. Proof of concept on the notion of novelty will open the way to more complex modes of reasoning such as planning or argumentation (see examples above).
- ▷ **Address the challenges of efficiency**. Today, the most viable tools for integrating pattern mining and reasoning are constraints solvers (SAT, CP, ASP, theorem prover, etc.). The main interest of solvers is their versatility. However, their ability to process large amounts of data remains limited. Our objective is to explore the new generic framework in the community of constraint programming or SAT, to evaluate their new strategies on our problems. More especially, certain solvers goes beyond the usual framework of SAT problem solving. Two leads emerge
  - exploiting the ability of some solvers to use domain theories (SAT Modulo theory [Barrett and Tinelli, 2018], i.e. solvers of formulas that mix logical expressions and expressions of a theory). The challenge is to identify which are the theories that are interesting to propose. In our case, a natural objective is a theory for chronicle matching.
  - exploring QBF solvers (Quantified Boolean Formula) [Amendola, 2018] that address problems of classes of complexity higher than 3-SAT problems. We expect that problems mixing extraction and reasoning will belong to this type of high complexity class.

  Moreover, as our main problem with ASP is the memory requirement, a short term perspective is to evaluate the potential benefit of lazy grounding strategy [Taupe, 2018; Lefèvre and Nicolas, 2009].

Enriching models with data The use of pattern mining to enrich the formal models of a system requires an improvement in the formalization of the patterns themselves. Since the 1990s and the first work in pattern mining, a considerable effort has been made on the algorithmic properties of patterns, leading to a significant improvement of the algorithm performances. The question of their semantics finally remains an area that needs to be investigated and better formalized.

In this research axis, we are interested in continuing our recent work on the **formalization and characterization of complex pattern domains** (chronicles, negative patterns) that provide valuable knowledge about pattern semantic.

Formalizing and characterizing patterns must be considered within the general framework of integrating mining and formal reasoning. Then, the desired properties of pattern domains are no longer only the antimonotonicity of a support measure (i.e. the property required for pattern mining), but also properties that would allow to reason about patterns: their actionability (in decision making), their usefulness for optimization or planning problems, etc. What are these properties? Are they compatible with pattern discovery objective? These are some of the challenging questions we face off.

Converge between knowledge representation (logic, ontology) and pattern mining The era of big data had to find in the data the answers to the questions that companies, scientists, doctors, etc. ask themselves about systems they are trying to understand or to manage. These massive raw data are often collected without any specific purpose, and may be difficult to use (secondary usage data).

The second major difficulty encountered by pattern mining (difficulty of operationalization) is

the difficulty of bridging the following semantic gaps:

- ▷ the semantic gap between raw data and data that can be used by data analysis algorithms,
- ▷ the semantic gap between the user semantic level and the data analysis semantic level (interpretability issue).

These difficulties include the issues raised by the automation of the data mining process, and more especially the data wrangling phase [De Raedt et al., 2018]. We remind that our ambition is not in full-automation of data science process but one way to support domain experts in their analysis is to partially automation the data science workflows. The problems to address are thus similar.

In some areas (particularly in the biomedical informatics), formalized knowledge is available in large quantities and is often underutilized. These knowledge bases represent important research efforts to make the expert knowledge usable in an interoperable way (in particular, with semantic web technologies).

Just as the field of data query has been revolutionized by the use of knowledge to answer user questions, so the field of data mining could be revolutionized by using all this knowledge. Mugnier [2018] proposed the *Ontology-Mediated Query Answering* concept, which consists in using ontologies to mediate the interrogation of a database by ontologies, i.e. to bridge the semantic gap that may exist between raw data and users. We propose to explore the concept of **Ontology-Mediated Pattern Mining** which would aim to mine patterns at a low semantic level while presenting useful information to the user at a high semantic level (more suitable for users). It is nothing more or less than injecting ontological reasoning into pattern extraction, which is one instance of the first challenge.

Our research project aims to explore these fundamental questions in the context of **temporal data analysis**, driven by applications that motivate the need for in-depth data mining. Indeed, temporal data are suitable for exploring these challenges. On the one hand, they are now very widely collected and their temporal dimension is semantically very rich. Their use should allow users to gain a significant amount of information compare to approach that to not exploit time. On the other hand, time have been widely studied in the different communities that feed our work (data mining, machine learning, artificial intelligence, constraint programming and discrete event systems). The originality of our research project lies in the interactions between methods from these different communities.

Bibliography

Achar, A., Laxman, S., and Sastry, P. S. (2010). A unified view of automata-based algorithms for frequent episode discovery. *CoRR*, abs/1007.0690.

Adam, A., Blockeel, H., Govers, S., and Aertsen, A. (2013). Sccql: A constraint-based clustering system. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 681–684. Springer.

Agrawal, R., Gehrke, J., Gunopulos, D., and Raghavan, P. (2005). Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11(1):5–33.

Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pages 207–216.

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proceedings of the 20th Internationnal Conference in Very Large Databases*, pages 487–499. Morgan Kaufmann.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the International Conference on Data Engineering*, pages 3–14.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.

Alvarez, M. R., Felix, P., and Carinena, P. (2013). Discovering metric temporal constraint networks on temporal databases. *Artificial Intelligence in Medicine*, 58(3):139 – 154.

Amendola, G. (2018). Towards quantified answer set programming. In *RCRA@ FLoC*.

Andrews, S., Tsochantaridis, I., and Hofmann, T. (2003). Support vector machines for multiple-instance learning. In *Advances in neural information processing systems*, pages 577–584.

Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., and Studer, R. (2011). ETALIS: Rule-based reasoning in event processing. In *Proceedings of Reasoning in event-based distributed systems*, pages 99–124. Springer.

Artale, A., Kontchakov, R., Kovtunova, A., Ryzhikov, V., Wolter, F., and Zakharyaschev, M. (2017). Ontology-mediated query answering over temporal data: A survey (invited talk). In *24th International Symposium on Temporal Representation and Reasoning (TIME 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Bacry, E., Gaffas, S., Leroy, F., Morel, M., Nguyen, D. P., Sebiat, Y., and Sun, D. (2019). Scalpel3: a scalable open-source library for healthcare claims databases.

Bakalara, J. (2019). Temporal models of care sequences for the exploration of medico-administrative data. In *Proceedings of the 17th Conference on Artificial Intelligence in Medicine (AIME)*, pages 1–7, Poznan, Poland.

Balbiani, P. and Condotta, J.-F. (2002). Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *International Workshop on Frontiers of Combining Systems*, pages 162–176.

Barrett, C. and Tinelli, C. (2018). Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer.

Bay, S. D. and Pazzani, M. J. (2001). Detecting group differences: Mining contrast sets. *Data Mining and Knowledge Discovery*, 5(3):213–246.

Besnard, P. and Guyet, T. (2019). *Chronicles.* to appear.

Besnard, P., Guyet, T., and Masson, V. (2019). Admissible generalizations of examples as rules. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1480–1485.

Bienvenu, M. (2016). Ontology-mediated query answering: harnessing knowledge to get more from data. In *IJCAI: International Joint Conference on Artificial Intelligence*.

Biere, A., Heule, M., van Maaren, H., and Walsh, T. (2009). *Handbook of satisfiability. Frontiers in Artificial Intelligence and Applications*, volume 185. IOS Press.

Biernacki, C., Celeux, G., Govaert, G., and Langrognet, F. (2006). Model-based cluster and discriminant analysis with the MIXMOD software. *Computational Statistics and Data Analysis*, 51(2):587–600.

Bistarelli, S. and Bonchi, F. (2007). Soft constraint based pattern mining. *Data & Knowledge Engineering*, 62(1):118–137.

Blockeel, H., Page, D., and Srinivasan, A. (2005). Multi-instance tree learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 57–64.

Bonchi, F., Giannotti, F., Lucchese, C., Orlando, S., Perego, R., and Trasarti, R. (2006). Conquest: a constraint-based querying system for exploratory pattern discovery. In *Proceedings of the International Conference on Data Engineering*, pages 159–159.

Bosc, G., Boulicaut, J.-F., Raïssi, C., and Kaytoue, M. (2018). Anytime discovery of a diverse set of patterns with monte carlo tree search. *Data Mining and Knowledge Discovery*, 32(3):604–650.

Boulicaut, J. and Jeudy, B. (2005). Constraint-based data mining. In Maimon, O. and Rokach, L., editors, *The Data Mining and Knowledge Discovery Handbook.*, pages 399–416. Springer.

Boullé, M. (2009). A parameter-free classification method for large scale learning. *J. Mach. Learn. Res.*, 10:1367–1385.

Boullé, M., Charnay, C., and Lachiche, N. (2019). A scalable robust and automatic propositionalization approach for bayesian classification of large mixed numerical and categorical data. *Machine Learning*, 108(2):229–266.

Brewka, G., Delgrande, J. P., Romero, J., and Schaub, T. (2015). asprin: Customizing answer set preferences without a headache. In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pages 1467–1474.

Bringmann, B., Nijssen, S., and Zimmermann, A. (2009). Pattern-based classification: a unifying perspective. In *Proceedings of the 2nd workshop 'From Local Patterns to Global Models', LeGo*, page 10.

Cabalar, P., Otero, R. P., and Pose, S. G. (2000). Temporal constraint networks in action. In *ECAI*, pages 543–547.

Cao, L., Dong, X., and Zheng, Z. (2016). e-nsp: Efficient negative sequential pattern mining. *Artificial Intelligence*, 235:156–182.

Cao, L., Yu, P. S., and Kumar, V. (2015). Nonoccurring behavior analytics: A new area. *IEEE Intelligent Systems*, 15:4–11.

Cheng, H., Yan, X., and Han, J. (2004). Incspan: incremental mining of sequential patterns in large database. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD, pages 527–532.

Chittaro, L. and Montanari, A. (2000). Temporal representation and reasoning in artificial intelligence: Issues and approaches. *Annals of Mathematics and Artificial Intelligence*, 28:47–106.

Chiu, B., Keogh, E., and Lonardi, S. (2003). Probabilistic discovery of time series motifs. In *Proceedings of the 9th International Conference on Knowledge Discovery and Data Mining*, pages 493–498.

Clarke Jr, E. M., Grumberg, O., Kroening, D., Peled, D., and Veith, H. (2018). *Model checking.* MIT Press.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the International Conference on Machine Learning*, pages 115–123.

Cohen-Boulakia, S., Belhajjame, K., Collin, O., Chopard, J., Froidevaux, C., Gaignard, A., Hinsen, K., Larmande, P., Le Bras, Y., Lemoine, F., et al. (2017). Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities. *Future Generation Computer Systems*, 75:284–298.

Colas, S., Collin, C., Piriou, P., and Zureik, M. (2015). Association between total hip replacement characteristics and 3-year prosthetic survivorship: A population-based study. *JAMA Surgery*, 150(10):979–988.

Combi, C. and Chittaro, L. (1999). Abstraction on clinical data sequences: an object-oriented data model and a query language based on the event calculus. *Artificial Intelligence in Medicine*, 17(3):271–301.

Combi, C., Franceschet, M., and Peron, A. (2004). Representing and reasoning about temporal granularities. *Journal of Logic and Computation*, 14(1):51–77.

Combi, C., Keravnou-Papailiou, E., and Shahar, Y. (2010). *Temporal information systems in medicine*. Springer Science & Business Media.

Coquery, E., Jabbour, S., Saïs, L., and Salhi, Y. (2012). A SAT-Based approach for discovering frequent, closed and maximal patterns in a sequence. In *Proceedings of European Conference on Artificial Intelligence (ECAI)*, pages 258–263.

Cram, D., Mathern, B., and Mille, A. (2012). A complete chronicle discovery approach: application to activity analysis. *Expert Systems*, 29(4):321–346.

Dao, T., Kuo, C., Ravi, S. S., Vrain, C., and Davidson, I. (2018). Descriptive clustering: ILP and CP formulations with applications. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1263–1269.

Dauxais, Y. (2018). *Discriminant chronicle mining*. PhD thesis, University of Rennes 1, France.

Dauxais, Y., Gross-Amblard, D., Guyet, T., and Happe, A. (2019). Discriminant chronicle mining. In Pinaud, B., Guillet, F., Gandon, F., and Largeron, C., editors, *Advances in Knowledge Discovery and Management (vol 8)*, Advances in Knowledge Discovery and Management, pages 89–118. Springer, Cham.

De Bie, T. (2013). Subjective interestingness in exploratory data mining. In *Advances in Intelligent Data Analysis XII*, pages 19–31. Springer.

De Bie, T. and Spyropoulou, E. (2013). A theoretical framework for exploratory data mining: recent insights and challenges ahead. In *Machine Learning and Knowledge Discovery in Databases*, pages 612–616. Springer.

De Raedt, L. (2012). Declarative modeling for machine learning and data mining. In *International Conference on Formal Concept Analysis*, pages 2–2. Springer.

De Raedt, L., Blockeel, H., Kolb, S., Teso, S., and Verbruggen, G. (2018). Elements of an automatic data scientist. In *International Symposium on Intelligent Data Analysis*, pages 3–14. Springer.

Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial intelligence*, 49:61–95.

Dermouche, S. and Pelachaud, C. (2016). Sequence-based multimodal behavior modeling for social agents. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, ICMI 2016, pages 29–36.

Diop, L., Diop, C. T., Giacometti, A., Li, D., and Soulet, A. (2018). Sequential pattern sampling with norm constraints. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 89–98.

Dong, G. and Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of ACM SIGKDD*, pages 43–52.

Dou, D., Wang, H., and Liu, H. (2015). Semantic data mining: A survey of ontology-based approaches. In *Proceedings of the 9th international conference on semantic computing*, pages 244–251.

Dousson, C. and Duong, T. (1999). Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 620–626.

Dousson, C. and Maigat, P. L. (2007). Chronicle recognition improvement using temporal focusing and hierarchization. In Veloso, M. M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 324–329.

Egho, E., Gay, D., Boull, M., Voisine, N., and Clrot, F. (2015). A parameter-free approach for mining robust sequential classification rules. In *2015 IEEE International Conference on Data Mining*, pages 745–750.

Euzenat, J. and Montanari, A. (2005). Chapter 3 - time granularity. In Fisher, M., Gabbay, D., and Vila, L., editors, *Foundations of Artificial Intelligence*, volume 1, pages 59 – 118. Elsevier.

Ezeife., C. and Monwar, M. (2007). A PLWAP-based algorithm for mining frequent sequential stream patterns. *International Journal of Information Technology and Intelligent Computing*, 2(1):89–116.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37.

Feurer, M. and Hutter, F. (2019). Hyperparameter optimization. In Hutter et al. [2019], pages 3–33.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J. T., Blum, M., and Hutter, F. (2019). Auto-sklearn: Efficient and robust automated machine learning. In Hutter et al. [2019], pages 113–134.

Fournier-Viger, P., Lin, J. C.-W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., and Lam, H. T. (2016). The spmf open-source data mining library version 2. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 36–40. Springer.

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., and Thomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77.

Fradkin, D. and Mörchen, F. (2015). Mining sequential patterns for classification. *Knowledge and Information Systems*, 45(3):731–749.

Frey, B. J. and Dueck, D. (2007). Clustering by passing messages between data points. *Science*, 315(5814):972–976.

Froese, T. and Ziemke, T. (2009). Enactive artificial intelligence: Investigating the systemic organization of life and mind. *Artificial Intelligence*, 173(3):466–500.

Furche, T., Gottlob, G., Libkin, L., Orsi, G., and Paton, N. W. (2016). Data wrangling for big data: Challenges and opportunities. In *EDBT*, pages 473–478.

Garofalakis, M., Rastogi, R., and Shim, K. (1999). SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the International Conference on Very Large Data Bases*, pages 223–234.

Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., and Schneider, M. (2011). Potassco: The Potsdam answer set solving collection. *AI Communications*, 24(2):107–124.

Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385.

Gerbser, M., Guyet, T., Quiniou, R., Romero, J., and Schaub, T. (2016). Knowledge-based sequence mining with ASP. In *Proceedings of Internation Join Conference on Artificial Intelligence (IJCAI)*, pages 1497–1504.

Giannotti, F., Nanni, M., Pedreschi, D., and Pinelli, F. (2006). Mining sequences with temporal annotations. In *Proceedings of the Symposium on Applied Computing*, pages 593–597.

Giatrakos, N., Artikis, A., Deligiannakis, A., and Garofalakis, M. (2017). Complex event recognition in the big data era. *Proc. VLDB Endow.*, 10(12):1996–1999.

Gomariz, A. (2014). *Techniques for the Discovery of Temporal Patterns*. PhD thesis, Universidad de Murcia.

Guns, T., Dries, A., Nijssen, S., Tack, G., and De Raedt, L. (2015). MiningZinc: A declarative framework for constraint-based mining. *Artificial Intelligence*, page In press.

Guyet, T. (2007). *Interprétation collaborative de séries temporelles. Application à des données de réanimation médicale*. PhD thesis, INPG.

Guyet, T. (2019). Semantic(s) of negative sequential patterns. In *Actes des Journées d'Intelligence Artificielle Fondamentale (IAF)*.

Guyet, T., Besnard, P., Samet, A., Ben Salha, N., and Lachiche, N. (2020). Énumération des occurrences d'une chronique. In *Actes de la conférence Extraction et Gestion des Connaissances (EGC)*, pages 253–260.

Guyet, T., Dauxais, Y., and Happe, A. (2017a). Declarative sequential pattern mining of care pathways. In *Proceedings of Conference on Artificial Intelligence in Medicine (AIME)*, pages 261–266.

Guyet, T. and Quiniou, R. (2008). Mining temporal patterns with quantitative intervals. In Zighed, D., Ras, Z., and Tsumoto, S., editors, *Proceedings of the 4th International Workshop on Mining Complex Data*, page 10.

Guyet, T. and Quiniou, R. (2011). Extracting temporal patterns from interval-based sequences. In *Proceedings of International Join Conference on Artificial Intelligence (IJCAI)*, pages 1306–1311.

Guyet, T. and Quiniou, R. (2012). Incremental mining of frequent sequences from a window sliding over a stream of itemsets. In *Actes des Journée Intelligence Artificielle Fondamentale (IAF)*, page 9.

Guyet, T. and Quiniou, R. (2020). NegPSpan: efficient extraction of negative sequential patterns with embedding constraints. *Data Mining and Knowledge Discovery*, 34(2):563–609.

Guyet, T., Quiniou, R., and Masson, V. (2017b). Mining relevant interval rules. In *International Conference on Formal Concept Analysis*, Supplementary proceedings of International Conference on Formal Concept Analysis (ICFCA).

Guyet, T., Quiniou, R., Moinard, Y., and Schaub, T. (2017c). Efficiency analysis of ASP encodings for sequential pattern mining tasks. *Advances in Knowledge Discovery and Management*, 7:41–81.

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. (2000). FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 355–359. ACM.

Happe, A. and Drezen, E. (2018). A visual approach of care pathways from the French nationwide SNDS database - from population to individual records: the ePEPS toolbox. *Fundamental and Clinical Pharmacology*, 32(1):81 – 84.

Hassani, M., Lu, Y., Wischnewsky, J., and Seidl, T. (2016). A geometric approach for mining sequential patterns in interval-based data streams. In *Fuzzy Systems (FUZZ-IEEE), 2016 IEEE International Conference on*, pages 2128–2135. IEEE.

Hercberg, S., Castetbon, K., Czernichow, S., Malon, A., Mejean, C., Kesse, E., Touvier, M., and Galan, P. (2010). The nutrinet-santé study: a web-based prospective study on the relationship between nutrition and health and determinants of dietary patterns and nutritional status. *BMC public health*, 10(1):242.

Hirate, Y. and Yamana, H. (2006). Generalized sequential pattern mining with item intervals. *Journal of computers*, 1(3):51–60.

Ho, C.-C., Li, H.-F., Kuo, F.-F., and Lee, S.-Y. (2006). Incremental mining of sequential patterns over a stream sliding window. In *IWMESD Workshop at ICDM*, pages 677 –681.

Höppner, F. (2002). Learning dependencies in multivariate time series. In *Proceedings of the Workshop on Knowledge Discovery in (Spatio-)Temporal Data*, pages 25–31.

Hsueh, S.-C., Lin, M.-Y., and Chen, C.-L. (2008). Mining negative sequential patterns for e-commerce recommendations. In *Proceedings of Asia-Pacific Services Computing Conference*, pages 1213–1218. IEEE.

Huang, J.-W., Tseng, C.-Y., Ou, J.-C., and Chen, M.-S. (2006). On progressive sequential pattern mining. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, CIKM '06, pages 850–851.

Huang, Z., Lu, X., and Duan, H. (2012). On mining clinical pathway patterns from medical behaviors. *Artificial Intelligence in Medicine*, 56(1):35–50.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2019). *Automated Machine Learning - Methods, Systems, Challenges*. The Springer Series on Challenges in Machine Learning. Springer.

Hwang, S.-Y., Wei, C.-P., and Yang, W.-S. (2004). Discovery of temporal patterns from process instances. *Computers in Industry*, 53(3):345–364.

Irpino, A. and Verde, R. (2008). Dynamic clustering of interval data using a wasserstein-based distance. *Pattern Recogn. Lett.*, 29(11):1648–1658.

Jabbour, S., Mana, F. E., Dlala, I. O., Raddaoui, B., and Sais, L. (2018). On maximal frequent itemsets mining with constraints. In *International Conference on Principles and Practice of Constraint Programming*, pages 554–569. Springer.

Jabbour, S., Sais, L., and Salhi, Y. (2013). Boolean satisfiability for sequence mining. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 649–658. ACM.

Janhunen, T. and Niemelä, I. (2016). The answer set programming paradigm. *AI Magazine*, 37:13–24.

Järvisalo, M. (2011). Itemset mining as a challenge application for answer set enumeration. In *Proceedings of the conference on Logic Programming and Nonmonotonic Reasoning*, pages 304–310.

Kam, P.-S. and Fu, A. W.-C. (2000). Discovering temporal patterns for interval-based events. In *Data Warehousing and Knowledge Discovery (DaWaK)*, pages 317–326.

Kamepalli, S., Sekhara, R., and Kurra, R. (2014). Frequent Negative Sequential Patterns – a Survey. *International Journal of Computer Engineering and Technology*, 5, 3:115–121.

Kaytoue, M., Kuznetsov, S. O., and Napoli, A. (2011). Revisiting numerical pattern mining with formal concept analysis. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Kemmar, A., Loudni, S., Lebbah, Y., Boizumault, P., and Charnois, T. (2015). Prefix-projection global constraint for sequential pattern mining. In *International Conference on Principles and Practice of Constraint Programming*, pages 226–243. Springer.

Kim, T.-W., Lee, J., and Palla, R. (2009). Circumscriptive event calculus as answer set programming. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*.

Koh, Y. S. and Ravana, S. D. (2016). Unsupervised rare pattern mining: A survey. *Transactions on Knowledge Discovery from Data*, 10(4):1–29.

Kontonasios, K., Spyropoulou, E., and Bie, T. D. (2012). Knowledge discovery interestingness measures based on unexpectedness. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(5):386–399.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2019). Auto-weka: Automatic model selection and hyperparameter optimization in WEKA. In Hutter et al. [2019], pages 81–95.

Lallouet, A., Moinard, Y., Nicolas, P., and Stéphan, I. (2013). Programmation logique. In Marquis, P., Papini, O., and Prade, H., editors, *Panorama de l'intelligence artificielle : ses bases méthodologiques, ses développements*, volume 2. Cépaduès.

Lam, H. T., Mrchen, F., Fradkin, D., and Calders, T. (2014). Mining compressing sequential patterns. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 7(1):34–52.

Laxman, S. and Sastry, P. S. (2006). A survey of temporal data mining. *Sadhana*, 31(2):173–198.

Laxman, S., Sastry, P. S., and Unnikrishnan, K. P. (2007). A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419.

Lee, S. D. and De Raedt, L. (2004). *Database support for data mining applications: discovering knowledge with inductive queries*, chapter Constraint based mining of first order sequences in SeqLog, pages 154–173. Springer.

Lefèvre, C. and Nicolas, P. (2009). The first version of a new ASP solver: ASPeRiX. In *Proceedings of the conference on Logic Programming and Nonmonotonic Reasoning*, pages 522–527.

Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., and Scarcello, F. (2006). The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562.

Levesque, H. J. (1986). Knowledge representation and reasoning. *Annual review of computer science*, 1(1):255–287.

Lhote, L. (2010). Number of frequent patterns in random databases. In *Advances in Data Analysis*, Statistics for Industry and Technology, pages 33–45. Skiadas, Christos H.

Lifschitz, V. (2008). What is answer set programming? In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pages 1594–1597.

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015a). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493.

Liu, J., Pacitti, E., Valduriez, P., and Mattoso, M. (2015b). A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493.

Long, D. (1989). A review of temporal logics. *The Knowledge Engineering Review*, 4(2):141–162.

Low-Kam, C., Raïssi, C., Kaytoue, M., and Pei, J. (2013). Mining statistically significant sequential patterns. In *Proceedings of the IEEE International Conference on Data Mining*, pages 488–497.

Mannila, H., Toivonen, H., and Verkamo, A. I. (1997). Discovering frequent episodes in event sequences. *Journal of Data Mining and Knowledge Discovery*, 1(3):210–215.

Martin-Latry, K. and Bégaud, B. (2010). Pharmacoepidemiological research using french reimbursement databases: yes we can! *Pharmacoepidemiology and drug safety*, 19(3):256–265.

Masseglia, F., Poncelet, P., and Teisseire, M. (2008). Efficient mining of sequential patterns with time constraints: Reducing the combinations. *Expert Systems With Applications*, 40(3):to appear.

Mathonat, R., Nurbakova, D., Boulicaut, J.-F., and Kaytoue, M. (2019). SeqScout: Using a Bandit Model to Discover Interesting Subgroups in Labeled Sequences. In *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Washington, United States.

McDermott, D. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155.

Métivier, J.-P., Loudni, S., and Charnois, T. (2013). A constraint programming approach for mining sequential patterns in a sequence database. In *Proceedings of the Workshops of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD)*.

Mitsa, T. (2010). *Temporal data mining*. Chapman and Hall/CRC.

Mooney, C. H. and Roddick, J. F. (2013). Sequential pattern mining – approaches and algorithms. *ACM Journal of Computing Survey*, 45(2):1–39.

Moskovitch, R. and Shahar, Y. (2015). Fast time intervals mining using the transitivity of temporal relations. *Knowledge and Information Systems*, 42(1):21–48.

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.

Mugnier, M.-L. (2018). Reasoning on data: the ontology-mediated query answering problem. In *Handbook of the 6th World Congress and School on Universal Logic*, page 76.

Nagesh, H., Goil, S., and Choudhary, A. (2001). Parallel algorithms for clustering high-dimensional large-scale datasets. In *Data mining for scientific and engineering applications*, pages 335–356. Grossman, Robert L. and Kamath, Chandrika and Kegelmeyer, Philip and Kumar, Vipin and Namburu, Raju R.

Negrevergne, B., Dries, A., Guns, T., and Nijssen, S. (2013). Dominance programming for itemset mining. In *Proceedings of the International Conference on Data Mining*, pages 557–566.

Negrevergne, B. and Guns, T. (2015). Constraint-based sequence mining using constraint programming. In *Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR*, pages 288–305.

Novak, P. K., Lavrač, N., and Webb, G. I. (2009). Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining. *Journal of Machine Learning Research*, 10:377–403.

Nowak, E., Happe, A., Bouget, J., Paillard, F., Vigneau, C., Scarabin, P.-Y., and Oger, E. (2015). Safety of fixed dose of antihypertensive drug combinations compared to (single pill) free-combinations: A nested matched case–control analysis. *Medicine*, 94(49):e2229.

Ouaknine, J. and Worrell, J. (2008). Some recent results in metric temporal logic. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 1–13. Springer.

OConnor, M. J., Shankar, R. D., Parrish, D. B., and Das, A. K. (2009). Knowledge-data integration for temporal reasoning in a clinical trial system. *International journal of medical informatics*, 78:77–85.

Papapetrou, P., Kollios, G., Sclaroff, S., and Gunopulos, D. (2009). Mining frequent arrangements of temporal intervals. *Knowledge and Information Systems*, 21(2):133.

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*, pages 393–404. ACM.

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *icccn*, page 0215.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2004). Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. *IEEE Transactions on knowledge and data engineering*, 16(11):1424–1440.

Pei, J., Han, J., and Wang, W. (2007). Constraint-based sequential pattern mining: The pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160.

Perer, A. and Wang, F. (2014). Frequence: interactive mining and visualization of temporal frequent event sequences. In *Proceedings of the international conference on Intelligent User Interfaces*, pages 153–162.

Polard, E., Nowak, E., Happe, A., Biraben, A., and Oger, E. (2015). Brand name to generic substitution of antiepileptic drugs does not lead to seizure-related hospitalization: a population-based case-crossover study. *Pharmacoepidemiology and drug safety*, 24(11):1161–1169.

Public Policy Committee *et al.* (2016). Guidelines for good pharmacoepidemiology practice (GPP). *Pharmacoepidemiology and drug safety*, 25(1):2.

Quiniou, R., Carrault, G., Cordier, M.-O., and Wang, F. (2003). Temporal abstraction and inductive logic programming for arrythmia recognition from electrocardiograms. *Artificial Intelligence in Medicine*, 28:231–263.

Rivault, Y. (2019). *Care trajectory analysis using medico-administrative data : contribution of a knowledge-based enrichment from the Linked Data*. Theses, Université Rennes 1.

Rivault, Y., Dameron, O., and Le Meur, N. (2019). queryMed: Semantic Web functions for linking pharmacological and medical knowledge to data. *Bioinformatics*.

Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of constraint programming*. Elsevier.

Ruan, G., Zhang, H., and Plale, B. (2014). Parallel and quantitative sequential pattern mining

for large-scale interval-based temporal data. In *Big Data (Big Data), 2014 IEEE International Conference on*, pages 32–39. IEEE.

Sacchi, L., Capozzi, D., Bellazzi, R., and Larizza, C. (2015). JTSA: an open source framework for time series abstractions. *Computer Methods and Programs in Biomedicine*, 121(3):175–188.

Sacchi, L., Larizza, C., Combi, C., and Bellazzi, R. (2007). Data mining with temporal abstractions: learning rules from time series. *Data Mining and Knowledge Discovery*, 15(2):217–247.

Sahuguède, A., Le Corronc, E., and Le Lann, M.-V. (2018). An ordered chronicle discovery algorithm. In *3nd ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data, AALTD'18*.

Sellami, C., Samet, A., and Tobji, M. A. B. (2018). Frequent chronicle mining: Application on predictive maintenance. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1388–1393. IEEE.

Silberschatz, A. and Tuzhilin, A. (1995). On subjective measures of interestingness in knowledge discovery. In *KDD*, volume 95, pages 275–281.

Simons, P., Niemel, I., and Soininen, T. (2002). Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1-2):181–234.

Snodgrass, R. T. (1986). Temporal databases. In *Proc. IEEE computer*.

Srikant, R. and Agrawal, R. (1996a). Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17.

Srikant, R. and Agrawal, R. (1996b). Mining sequential patterns: Generalizations and performance improvements. *Advances in Database TechnologyEDBT'96*, pages 1–17.

Syrjänen, T. and Niemelä, I. (2001). The smodels system. In *Proceedings of the conference on Logic Programming and Nonmotonic Reasoning*, pages 434–438.

Szathmary, L., Valtchev, P., and Napoli, A. (2010). Generating rare association rules using the minimal rare itemsets family. *International Journal on Software and Informatics*, 4(3):219–238.

Tatti, N. and Vreeken, J. (2012). The long and the short of it: Summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, pages 462–470.

Taupe, R. (2018). Speeding up lazy-grounding answer set solving. In *Technical Communications of the 34th International Conference on Logic Programming (ICLP)*, pages 20:1–20:9.

Termier, A. (2013). *Pattern mining rock: more, faster, better*. PhD thesis, Université de Grenoble.

Tsesmeli, K., Boumghar, M., Guyet, T., Quiniou, R., and Pierre, L. (2018). Fouille de motifs temporels négatifs. In *Actes de la Conférence Internationale sur l'Extraction et la Gestion des Connaissances (EGC)*, pages 263–268.

Ugarte, W., Boizumault, P., Crémilleux, B., Lepailleur, A., Loudni, S., Plantevit, M., Raïssi, C., and Soulet, A. (2015). Skypattern mining: From pattern condensed representations to dynamic constraint satisfaction problems. *Artificial Intelligence*, page In press.

Uno, T. (2004). http://research.nii.ac.jp/ũno/code/lcm_seq.html.

Uno, T., Kiyomi, M., and Arimura, H. (2004). LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 126.

Uno, T., Kiyomi, M., and Arimura, H. (2005). Lcm ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, pages 77–86.

Verbruggen, G. and De Raedt, L. (2017). Towards automated relational data wrangling. In *Proceedings of AutoML 2017@ ECML-PKDD: Automatic selection, configuration and composition of machine learning algorithms*, pages 18–26.

Wang, J. and Han, J. (2004). BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the International Conference on Data Engineering*, pages 79–90.

Wang, W. and Cao, L. (2019). Negative sequences analysis: A review. *ACM Computing Survey*, 52.

Weill, A., Païta, M., Tuppin, P., Fagot, J.-P., Neumann, A., Simon, D., Ricordeau, P., Montastruc, J.-L., and Allemand, H. (2010). Benfluorex and valvular heart disease: a cohort study of a million people with diabetes mellitus. *Pharmacoepidemiology and drug safety*, 19(12):1256–1262.

WHO et al. (1993). *The ICD-10 classification of mental and behavioural disorders: diagnostic criteria for research*, volume 2. World Health Organization.

Winarko, E. and Roddick, J. F. (2006). ARMADA - an algorithm for discovering richer relative temporal association rules from interval-based data. *Data & Knowledge Engineering*, 63(1):76–90.

Wu, S.-Y. and Chen, Y.-L. (2007). Mining nonambiguous temporal patterns for interval-based events. *IEEE transactions on knowledge and data engineering*, 19(6):742–758.

Yan, X., Han, J., and Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the SIAM Conference on Data Mining*, pages 166–177.

Yen, S.-J. and Lee, Y.-S. (2013). Mining non-redundant time-gap sequential patterns. *Applied intelligence*, 39(4):727–738.

Yoshida, M., Iizuka, T., Shiohara, H., and Ishiguro, M. (2000). Mining sequential patterns including time intervals. In *Proceedings of the conference on Data Mining and Knowledge Discovery: Theory, Tools and Technology II*, pages 213–220.

Yuan, D., Lee, K., Cheng, H., Krishna, G., Li, Z., Ma, X., Zhou, Y., and Han, J. (2008). Cispan: Comprehensive incremental mining algorithms of closed sequential patterns for multi-versional software mining. In *Proceedings of SIAM*, pages 84–95.

Zaki, M. J. (2000). Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, pages 422–429.

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Journal of Machine Learning*, 42(1/2):31–60.

Zhao, J., Henriksson, A., and Bostrom, H. (2015). Cascading adverse drug event detection in electronic health records. In *International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–8.

Zheng, Z., Zhao, Y., Zuo, Z., and Cao, L. (2009). Negative-GSP: An efficient method for mining negative sequential patterns. In *Proceedings of the Australasian Data Mining Conference*, pages 63–67.

Zins, M., Goldberg, M., et al. (2015). The french constances population-based cohort: design, inclusion and follow-up. *European journal of epidemiology*, 30(12):1317–1328.