Apprentissage de règles associatives temporelles pour les séquences temporelles de symboles

Mathieu Guillame-Bert Guillame-Bert

▶ To cite this version:

Mathieu Guillame-Bert Guillame-Bert. Apprentissage de règles associatives temporelles pour les séquences temporelles de symboles. Mathématiques générales [math.GM]. Université de Grenoble, 2012. Français. NNT : 2012GRENM081 . tel-00849087

HAL Id: tel-00849087
https://theses.hal.science/tel-00849087

Submitted on 30 Jul 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique et Mathématique Appliquée**

Arrêté ministérial : 7 aôut 2006

Présentée par

Mathieu GUILLAME-BERT

Thèse dirigée par **James L. CROWLEY**

préparée au sein **du Laboratoire d'Informatique de Grenoble á l'INRIA Rhône-Alpes**
et de **l'Ecole Doctorale de Mathématiques, Sciences et Technologies de l'Information**

Learning temporal association rules on Symbolic time sequences

Thèse soutenue publiquement le **23 Novembre 2012**,
devant le jury composé de :

Mr. Augustin Lux
Professor at INPG-ENSIMAG, Président
Mr. Malik Ghallab
Research Director at LAAS-CNRS, Rapporteur
Mr. Paul Lukowicz
Professor at the Universite of Passau, Rapporteur
Mr. Artur Dubrawski
Senior Systems Scientist, Director of the Auton Lab, CMU, Examinateur
Mr. James L. CROWLEY
Professor at INPG-ENSIMAG, Directeur de thèse

Abstract

The learning of temporal patterns is a major challenge of Data mining. We introduce a temporal pattern model called Temporal Interval Tree Association Rules (Tita rules or Titar). This pattern model can be used to express both uncertainty and temporal inaccuracy of temporal events. Among other things, Tita rules can express the usual time point operators, synchronicity, order, and chaining, disjunctive time constraints, as well as temporal negation. Tita rules are designed to allow predictions with optimum temporal precision. Using this representation, we present the Titar learner algorithm that can be used to extract Tita rules from large datasets expressed as Symbolic Time Sequences. This algorithm based on entropy minimization, apriori pruning and statistical dependence analysis. We evaluate our technique on simulated and real world datasets. The problem of temporal planning with Tita rules is studied. We use Tita rules as world description models for a Planning and Scheduling task. We present an efficient temporal planning algorithm able to deal with uncertainty, temporal inaccuracy, discontinuous (or disjunctive) time constraints and predictable but imprecisely time located exogenous events. We evaluate our technique by joining a learning algorithm and our planning algorithm into a simple reactive cognitive architecture that we apply to control a robot in a virtual world.

Résumé

L'apprentissage de modèles temporels constitue l'une des grandes problématiques de *l'Exploration de Données* (Data Mining).

Dans cette thèse, nous avons développé un nouveau modèle temporel appelé *TITA Rules* (Règle associative temporelle basée sur des arbres d'intervalles). Ce modèle permet de décrire des phénomènes ayant un certain degré d'incertitude et/ou d'imprécision. Ce modèle permet entre autres d'exprimer la synchronicité entre évènements, les contraintes temporelles disjonctives et la négation temporelle. De par leur nature, les *TITA Rules* peuvent êtes utilisées pour effectuer des prédictions avec une grande précision temporelle.

Nous avons aussi développé un algorithme capable de découvrir et d'extraire de manière efficace des *TITA Rules* dans de grandes bases de données temporelles. Le cœur de l'algorithme est basé sur des techniques de minimisation d'entropie, de filtrage par "Apriori" et par des analyses de co-dépendance.

Notre modèle temporel et notre algorithme ont été appliqués et évalués sur plusieurs jeux de données issues de phénomènes réels et de phénomènes simulés.

La seconde partie de cette thèse a consisté à étudier l'utilisation de notre modèle temporel sur la problématique de la *Planification Automatique.*

Ces travaux ont mené au développement d'un algorithme de planification automatique. L'algorithme prend en entrée un ensemble de TITA Rules décrivant le fonctionnement d'un système quelconque, une description de l'état initial du système, et un but à atteindre. En retour, l'algorithme calcule un plan décrivant la meilleure façon d'atteindre le but donné.

Par la nature même des TITA Rules, cet algorithme est capable de gérer l'incertain (probabilités), l'imprécision temporelle, les contraintes temporelles disjonctives, ainsi que les événements exogènes prédictibles mais imprécis.

Acknowledgments

First of all, I would like to thank my supervisor Prof. James L. Crowley who provide me a stimulating research environment and introduce me to the research community. I am also grateful to Prof. Marie-Christine Rousset, Prof. Malik Ghallab, Prof. Paul Lukowicz and Dr. Artur Dubrawski for their interest in my work and for agreeing to be in the jury of this thesis.

I would also like to thank all the current and past members of PRIMA for their suppost and fun during those last three years. I further thank Thibault Lhermitte for his remarks and corrections.

Lastly, and most importantly, I would like to thank my family for their support and encouragement during all my studies.

Contents

1 Introduction

Chapter Contents

1.1 Scope

One of the main goals of physicists, and more generally of scientists, is to develop new models to explain and predict existing phenomena. However, scientists are not the only ones to develop models. In the everyday life, people and especially children invent models of the world. These conscious and unconscious models are vital for us. They represent a significant part of our knowledge. Examples of such models are the basic laws of physics (e.g. People know that objects tend to fall down, that fire is hot, and that they cannot go through walls) and the basic understanding of human actions and behaviors.

Since the earliest civilizations, humanity has tried express such knowledge with mathematical descriptions. The first attempts appear to have been done from a philosophical point of view. It is believed that Aristotle made the first work on Formal Logic around 400 BC, which is the earliest known mathematical representation of knowledge and reasoning. Through the ages, investigation of mathematics, logic, philosophy and psychologies have contributed to human understanding of natural and artificial phenomena.

With the development of Computer Science, the problem of mathematically representing knowledge has taken a new turn. A significant number of theories have been proposed on *how to represent knowledge*, *how to learn knowledge*, *how to transmit knowledge* and *how to use knowledge*. With the decrease of the price of memory, the increase of computers power, and the digitizing of absolutely every piece of information available, these theories become more and more useful, and increasingly used. One of the great challenges of the Computer science is to represent, learn and use automatically and efficiently models of any phenomena. A non exhaustive list of phenomena of great interest are: the weather, the earth's climate, the behaviors of people and animals, epidemic spread, the state of artificial systems (bridges, car, plane, space station, etc.), and basic physic. The ability to represent and use such knowledge is the key to technologies such as autonomous robotics, computer assisted medical diagnostics, weather and climate forecasting, automatic system fault prediction and detection, etc.

1.2 Goals and motivations

Time is a fundamental physic quantity that is present in all aspects of the real world. It is at the core of the Physics. It is well modeled but not well understood. Time at large scales (scale of the universe) or small scales (scale of particles) is still a very active field of research in physics. Time is well modeled at human scale (size, speed and duration), where time can be represented by a scalar quantity. The fig. 1.1 illustrates with examples the usual linear interpretation of the time.

At human scales, time has several properties that make it different from other quantities. For example, instants of time are unique, time can be ordered, and time has the property of transitivity (if A happens before B, and if B happens before C, therefore A happens before C).

In computer science, time is discretized for two reasons. First, time is generally represented by fixed or floating point values because common computers (Turing machines) cannot numerically represent real numbers. The second reason is the link between the complexity of algorithm and the discretization of time. Some algorithms discretize time into bins, and the time and space complexity of these algorithms depend on the number of bins. Therefore,

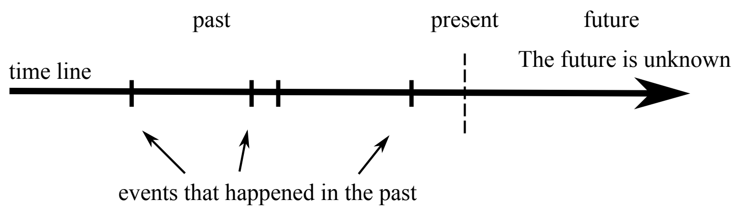


Figure 1.1: Illustration of the linear interpretation of the time. Examples of zero-duration events are given.

time is discretized into a finite number of bins in order for the algorithms to be run in reasonable time. In the case of learning algorithms, a third reason exists: The discretisation of time constraints is a kind of generalization. Therefore, it provides a solution to the sampling problem. The fig. 1.2 illustrates the linear interpretation of the time from the point of view of a computer.

Because of the omnipresence and the particular properties of time, the study of algorithms dealing with time has continued to be of a great interest. In this thesis, we address two of those domains: The learning of temporal models, and the temporal planning and scheduling.

Temporal learning

The temporal learning problem is to develop algorithms that can automatically build models of temporal phenomena. Similarly to scientists, the input of such technique is a set of observations of a phenomenon. The output is a model of the phenomenon. Models can be trivial (e.g. "If you touch fire, then you will be burned"), extremely complex or even stochastic (i.e. probabilistic). Models do not necessary have to be exact, but they have to be valid in some cases. Generally, models can be applied to predict the future state of a studied phenomenon. They can also be used in more complex operations such as fault detection, characterisation, planning or scheduling. The fig. 1.3 illustrates the temporal learning problem.

The temporal learning has to struggle with several problems. Below is a non-exhaustive listing of these problems:

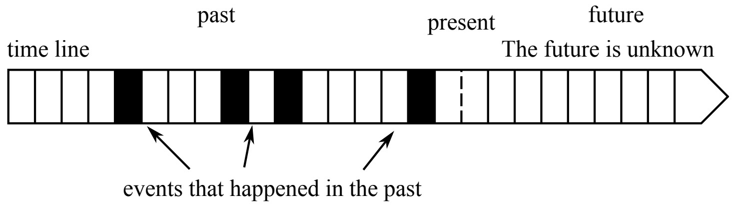


Figure 1.2: Illustration of the linear and discretized interpretation of the time from the point of view of a computer.

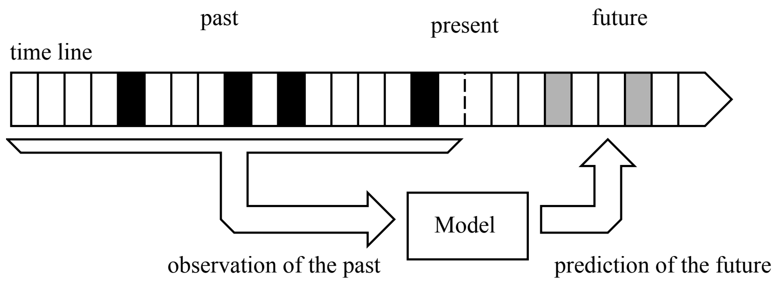


Figure 1.3: Illustration of the temporal learning problem. In this example, a model is used to predict the future of a phenomenon. Models can also be used to predict unobserved past of a phenomenon. The core of the temporal learning problem is to lean the model.

1. How to represent observations about phenomena?
2. How to represent models?
3. How to characterize a "good model"?
4. Since there is generally an infinite number of models, how to find the good ones without having to test them all?
5. Since there is generally an infinite number of good models, how to find the best ones?
6. How to detect trivial or conclusive, redundant and over specialized models?

The problem of the representation of observations about phenomena is a domain specific problem. Trivial and non trivial solutions exist. The five other problems are tightly interlaced since the complexity of dealing with a model is correlated with the structure of the model.

Temporal Planning and Scheduling

Automated Planning and Scheduling is the study of computer programs designed to build strategies that lead to desired states. Domains of application include autonomous robotics, automatic system maintenance and project management. The input of a temporal planning and scheduling algorithm is a description of a 'world', a list of possible doable actions, and a desired goal (or a set of desired goals). The term *world* is a generic term that represents an initial state and the model of a system (this model is often presented a set of rules). The output of a Planning algorithm is a plan, i.e. a description of the actions to perform in order to achieve a goal. The fig. 1.4 illustrates the planning and scheduling problem.

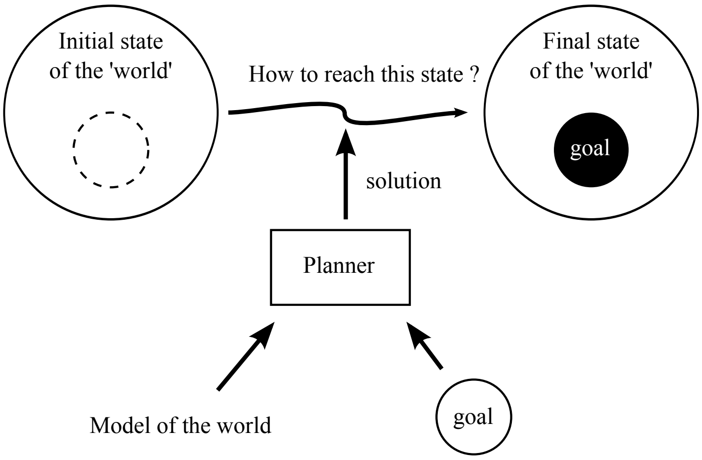


Figure 1.4: Illustration of the planning problem. An initial state of the world, a model of the word and a goal is given to the planner. The output is a plan that leading the world to a final state where the goal is reached.

A system for temporal planning and scheduling must resolve several problems, including:

1. Select a way to represent data (description of the world's states, description of the world's mechanisms, and description of plans).
2. How to choose from a possibly infinite number of plans (i.e. choose the plans that lead to the goals) without having to enumerate all the possible plans?
3. How to deal with the uncertainty?
4. How to adapt plan to new and unpredicted observations?
5. How to take into account the probability of (positive of negative) side effects of a plan.

1.3 Contribution of this study

During this thesis, we addressed two hard challenges of Computer Science: The Automated Learning and Automated Planning and Scheduling.

This work begun by the study of temporal patterns (patterns describing phenomenon with a temporal aspect) described in the literature. Our initial goal was to efficiently assist people in their everyday life throughout robotic interaction. In order to provide an efficient help to people, a context-aware building had to be able to "understand" the behaviors of occupants. Therefore, our first challenge was to represent the behaviors of people from the point of view of a context-aware building. As an example, a typical rule to lean and use is "If somebody is knocking at the door of a room, and there is somebody in this room, then the knocking person will probably enter the room in the following minutes". Another example of a rule to learn is "If the music is playing, a person is about to leave the room, and if there are no more people of this room, then this person will probably switch-off the music before leaving the room.".

The same example from the point of view of an intelligent and active environment is: "If the music is playing, a person is leaving the room, and if there are no more people of the room, then the music should be switched-off automatically.".

The pattern we tried to represent were not especially complex, but since people are by nature unpredictable, the patterns and the learning techniques had to be able to represent imprecise and inaccurate rules. Because no patterns and no algorithms met our needs, we developed a new temporal pattern called Titar. This temporal model has been designed to express a certain class of temporal correlations. This class of temporal correlation had to be large enough to describe some real world phenomena, but it had also to be simple enough to be learned efficiently. The initial goal of this temporal model was to be able to describe human activity for robotic applications. Surprisingly, experiments showed that our temporal model could be successfully applied to describe other domains.

The vast majority of temporal pattern in the literature can be seen as systems of temporal constraints. One of the core problems of temporal learning is the learning of these temporal constraints. Informally, this problem consists in choosing a finite set of 'good temporal constraints' from an infinite set of candidate temporal constraints. We call this problem the *Temporal Constraint Selection problem*. This problem is discussed and illustrated below in chapter 4.5.

As with most of the literature on temporal learning, the learning of Tita rules has to deal with this problem. Surprisingly, we observed that the majority of literature that we are aware of is proposing inefficient solutions to this problem. In this work, we have proposed a novel solution for this problem. In opposition to the current literature, our method does not suppose any initial properties for observations, and it does not perform exhaustive testing on all the candidate temporal constraints. This solution is justified and validated through several experiments. Based on our solution for the *Temporal Constraint Selection problem*, we present an efficient learning algorithm called Titarl.

The second part of this thesis develops a Planning and Scheduling algorithm based on our model of temporal rules. Given a world (or a system) described by a set of Tita rules and a goal, our planning algorithm computes a plan to achieve this goal in the described world.

We show that this combination is able to produce robust and adaptable plans, where robust means that the plans are able to deal with the uncertainty of the world, and adaptable means that the plans have a certain amount of freedom. A technique for representing temporal inaccuracy is presented.

The result is validated by experiments: Our learning algorithm is integrated with our planning and scheduling algorithm to yield a cognitive policy.

1.4 A guided tour of the chapters

Below is given an informal overview of each chapter.

- **Chapter 2** reviews the existing literature concerning the problem addressed in this thesis. It begins by introducing the problems tackled by Data mining (association rule learning, clustering, regression and classifications). Next, it presents the common time representations that are used in temporal data mining. A time representation defines how to represent temporal observations such as events or states (e.g. It is raining. It begins to rain. The rain is over. etc.) We adopt the terminology defined by Mörchen (Mörchen, 2007) to describe the various time representations. Next, this chapter describes the temporal patterns used in the literature.

  Temporal patterns define how to represent correlation/relation/patterns between events (e.g. If I press the switch, then the light will be on. Every night, it becomes dark. etc.). The presented patterns are the *general temporal constraint systems*, the *temporal constraint network*, the various types of *episodes*, the *chronicles*, the *temporal node Bayesian network* and a type of probabilistic and temporal rule. Next, we present existing temporal learning techniques found in the literature. We define classical notions about temporal learning and we present a small overview of each technique. Finally, we present the problem of temporal planning and scheduling, and we summarizes several works from the literature.

- **Chapter 3** details the various mathematical notions and representations that will be used in this thesis. Several of these notions are quite common but have several writing conventions and terminology. New simple notions are introduced to make the writing of equation more compact and easier to read. We also present our temporal pattern model and our temporal plan. We use the first model called Titar (for Temporal Interval Tree Association Rule) to represent temporal relation about observations. We present several examples of Titar. The second model called Meta Titar Plan is based on our Titar. We use it to represent plan (description of action that will lead to a given goal). We present several examples of Meta Titar Plans.

- **Chapter 4** details the technique we have designed to automatically learn Titar from sets of observations. This chapter begins by a discussion on a core issue for temporal learning algorithms: What should be a 'good' temporal pattern? One of the main difficulties of temporal learning is the learning of these constraints. We call this problem the *Temporal Constraint Selection Problem*. This problem is common to all temporal learning techniques, and it is addressed in various ways in the literature. We propose and justify a new approach to solve this problem.

  From this approach, we design and present the Titar learner algorithm which is presented in this chapter. This algorithm is based on four simple operations: the creation of trivial unit rules, the addition of condition, the division of rule and the refinement of rules. The first section presents and illustrates these operations. The algorithm is composed of several stages called *improvement policies* and based on the four simple operations. The second section presents and illustrates these policies. The third section is a listing of the algorithm. The fourth section details a step by step application of the algorithm. The last section introduces the Temporal Constraint Selection Problem. This definition is used to discuss the Titar algorithm and compares it to related literature.

- **Chapter 5** tackles the problem of temporal planning and scheduling. It introduces the aspect of time in a classical planning and scheduling problem. In this approach, worlds (or systems) are described with Tita rules. Since Tita rules can represent imprecise (non-deterministic) and inaccurate temporal relations, the Planning and Scheduling technique we are introducing has to be able to deal with such aspects. The tree structure of Tita rules (by opposition to the complete graph structure of usual pattern models) allows efficient planning with disjunctive time constraints and inaccurate temporal relations. Time is considered to be continuous.

  The tree structure of our pattern model (in opposition to the complete graph structure found in most of the literature (Dousson and Duong, 1999; Mannila *et al.*, 1997)) provides an interesting trade-off between the power of expression of patterns, and the complexity of learning them. From the point of view of automated planning, the tree structure is also easier to solve than the complete graph structure. This structure is the key to the planning algorithm we developed. It is able to deal with uncertainty (expressed with probabilities), temporal inaccuracy (expressed with ranges of value and probability distribution), and discontinuous (or disjunctive) time constraints (which is an NP-hard problem in a complete graph structure).

The first stage of the algorithm, called Titar planner, is able to build a plan from a goal and a single rule. The grammar of plan we develop is called Titar Plan. The created plans have a certain amount of freedom and do not necessarily impose an exact time for actions to be performed: For example, if the achievement of an action at any time in a given period lead to a goal, a good plan would define this period instead of assigning an arbitrary time-stamp to the actions. Since a world is generally not described with one single rule, we design an algorithm able to combine rules to build a plan. However, because of the temporal inaccuracy of rules, merging high confidence rules does not always lead to a high confidence plan. This point is discussed in details in section 3.9. Examples are given.

In order to tackle this issue, we propose a plan model called Meta Titar plan. Informally, a Meta Titar plan is a set of Titar plans connected by temporal constraints. A Meta-plan is more expressive than a plan. A Meta-plan can express concepts such as 'waiting on an expected exogenous event before continuing the execution of a sequence of actions'.

We present an algorithm able to build Meta-plans from a goal, a set of rules, and additional observations of the world. The last part of the algorithm is the scheduling stage. The scheduling stage is the selection of the time samples of actions to perform according to arbitrary criteria. We detail a step by step run of our technique. Several illustrated examples of plan and Meta-plan are given. We conclude the chapter with the analysis of the time complexity of the planning algorithm.

- **Chapter** 6 presents the various experiments we made to validate our work i.e. data models and algorithms. The first section of the chapter presents the experiments on temporal learning. The second section presents one experiment on planning where the input of the technique is the actual output of our learning algorithm. The first section begins with a presentation of several evaluation measures. We define simple common measures such as confidence, support, range and precision. We discuss the problem of evaluation temporal learning. Based on this discussion, we introduce a measure called 'Score'. The score expresses the 'value' of a temporal association rule. It is computed from confidence, support and prediction range. We also introduce a measure called Global Support Map (GSM). The GSM expresses the relation between minimum confidence of a predictor, minimum temporal precision of a predictor, and percentage of event explained by this predictor. This measure is especially interesting to get overview of a set of temporal rules. Examples of GSMs are given. Our learning algorithm is applied on several temporal domains:

– First, we apply our algorithm on a computer generated dataset. This dataset is composed of 100 sub-parts with different temporal patterns having various complexity and noises. These patterns are available for evaluation of learning. This dataset also contains *reference predictions* (or *prediction ground truth*) i.e. the best predictions that can be done. Reference predictions allow more detailed evaluation of temporal learning techniques than 'events to predict ground truth'. This point is discussed in the chapter. Our learning algorithm and two other techniques from the literature are evaluated.

– In this second experiment, we evaluate the ability of our algorithm to learn the behavior of a person. This experiment is based on a 28 days of sensor data and activity annotations about one person living in an apartment. In this experiment, we use the algorithm to predict the activities of the person based on the sensor data. Two algorithms from the literature are also applied on this problem, and the results are compared with our algorithm.

– In the third experiment, we use our algorithm to predict the motions of people in a building. This experiment is based on data recorded by over 200 proximity sensors. The algorithm successfully learns rules between the sensors signal. The experiment is concluded with the reconstruction of the topological structure of the building from the learned rules.

– In the fourth experiment, we apply our algorithm to the Foreign exchange market (Forex). Our goal is to train our system to predict the upward/downward trends of the markets, and automatically make buying/selling orders. The results are conclusive, and they leaded us to believe that our algorithm can be used as the core of automatic Forex trading systems.

In the second section of this chapter, we evaluate our temporal planning and scheduling algorithm. We simulate a virtual stochastic world. A robot is placed in this world with the goal of getting resources. Initially, the robot does not have any knowledge about the world. Through 'try and observe' behavior, the robot learns a model of the world, and uses this model to design plans to get food. The robot is controlled with a simple cognitive policy relying on our learning and planning algorithms.

2

State of the art

Chapter Contents

The Encyclopaedia Britannica gives the following definition to Data mining:

> *Data mining, also called Knowledge Discovery in Databases in computer science, is the process of discovering interesting and useful patterns and relationships in large volumes of data. The field combines tools from statistics and artificial intelligence with database management to analyze large digital collections, known as datasets.*
>
> – *Encyclopaedia Britannica*

Several other definitions exist with subtle differences. However the main ideas over the definitions are generally similar and the small differences do not have any practical implications. The definitions can be summed up to the following points:

1. A (possible very large) volume of data is available.
2. This dataset is not entirely random and contains some patterns.
3. These patterns are more of less interesting according to arbitrary criteria.
4. Data mining (or Knowledge Discovery in Databases) is the study of algorithms that can find (learn, extract, mine, etc.) efficiently the more interesting patterns from the dataset.

Techniques for Data mining have been applied to a wide range of domains ranging from computer science and business to medicine and economics.

Techniques for Data Mining can be divided into four categories:

Association rule learning is the search for of association rules. An association rules is a "$conditions \rightarrow implications$" pattern. A famous example of association rule is the fact that 'between 5 p.m. and 7 p.m., men who buy diaper tend also to buy beer' (Thomas Blischok – manager of a retail consulting group at Teradata, 1992).
Clustering is the search for groups of 'similar' elements such that the elements of a same group are more similar to each other than to elements of

other groups.
Regression is the analysis of the dependence of variables. Regression is usually the research of a function which gives an approximation of the value of a (dependant) variable, given the value of other variables.
The Classification is the task to organize a new set of data into a known structure.

Following this classification, the work presented in this thesis can be seen as a form of association rule learning.

A variety of data models have been explored with varieties depending on the process that is applied and the nature of the information that is used.

The simplest data model used for association rule learning is the transaction data model.

Definition 2.0.1 *A* transaction *is a set of items (or itemset) i.e. symbols. A* transaction dataset *is a set of* transactions*. An example of transaction dataset is* $\{\{A, B\}, \{C\}, \{A, B, C\}\}$.

In the case of transaction data model, an association rule is a set of symbols called a 'condition' and a set of symbols called a 'head'.

More complex data models have been created by enriching the items of transaction datasets with extra information such as, numerical values, probability, temporal localization, first order logic terms, etc.

In this thesis, we are focusing on datasets augmented with temporal and probabilistic information. More precisely, we are dealing with 'symbolic time sequence' datasets. This term will be defined in the next section.

The next section presents various temporal data models. Symbolic time sequences are one of them (see definition in section 2.1). The section 2.2 presents several types of patterns used to describe correlation in temporal datasets. We emphasize patterns related to Symbolic time sequences. The section 2.3 presents Data mining algorithms of the current literature which are designed to lean 'symbolic time sequences'. The section 2.4 presents an overview of the current literature on automated temporal planning and scheduling.

2.1 Time representation

Mörchen (Mörchen, 2007) proposes a general definition for the different temporal data models used by the Data mining community. Other naming conventions exist. Depending on the naming convention, a given name can refer to different temporal data models. The different temporal data models are defined as follow:

- *A* ***time series*** *is a set of unique time points (This definition supposes that the intervals between the time points are similar).*
- *A* ***time sequence*** *is a multi set of time points.*
- *A pair of time points defines a* ***time interval****, inclusively.*
- *Two intervals* ***overlap*** *if there is at least one time point that lies within both intervals.*
- *An* ***interval series*** *is a set of non overlapping time intervals.*
- *An* ***interval sequence*** *can include overlapping and equal time intervals.*
- *The series data types can be* ***univariate*** *or* ***multivariate****.*

The *series data types* can be symbol, set of symbols, numerical values, etc. For example, a symbolic time sequence is a multi-set of time points labelled with a symbol. Fig. 2.1 shows examples of data models.

Scalar time series are generally used to represent repetitive sampling a raw data (e.g. $< 1.5, 8, 2, 3 >$) The sampling the temperature in a room every ten minutes is an example of *scalar time series*.

Symbolic time series are generally used to represent series of ordered (but non time-sampled) events (e.g. $< a, b, d, a, c >$ or $< abdac >$ [the two notations are equivalent]). A sentence is a symbolic time series where each letter is a symbol. In biological research, *DNA sequencing* uses symbolic time series to represent fragments of DNA. In some papers, symbolic time series sometime refer to series of partially ordered events (e.g. $< a(abc)g(bc) >$).

Time sequences are used when the time sampling is important. For example, suppose a dataset which is a record of credit card operations. Since people tend to have patterns in the use of their credit card, banks can use such dataset to detect stolen credit card. In this scenario, the fact that twenty purchases have been done in an hour on the same credit card might be a sign that this credit card has been stolen. While, the fact that twenty purchases have been done over a month might be normal. This example is a simple illustration of the importance of time sampling in temporal datasets.

By opposition to time series, time sequences can deal with several events defined at the same temporal location. Time sequences can also be used to express interval series and interval sequence by giving begin and end points of intervals. Fig. 2.2 is an example of real world symbol interval sequence dataset.

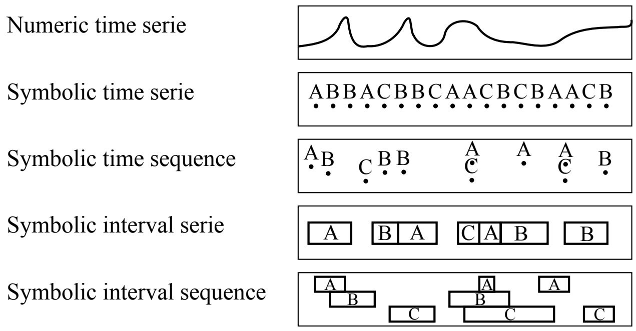


Figure 2.1: Several examples of temporal data models used in temporal data mining.

Figure 2.2: Example of real world symbol interval series dataset (van Kasteren *et al.*, 2008a). The display is done with the software Event Viewer developped diring this thesis.

2.2 Temporal patterns

Various temporal patterns have been designed for each data models. In this thesis we are using the temporal pattern called Symbolic time sequences.

Since several patterns for symbolic time sequences are inspired by patterns for symbolic time series, we also present a small overview of symbolic time series patterns and symbolic time series mining algorithms.

2.2.1 Patterns for Symbolic time series

The more common pattern for Symbolic time series is called *Sequential Patterns*. A *Sequential Patterns* is actually a *symbolic time series* (according to the naming convention in the thesis). For example, the sequential pattern $< abac >$ indicates an event a, followed by an event b, followed by an event a and finally followed by an event c. Similarly to the Time representation, in some papers, patterns for symbolic time series can be series of partially ordered events (e.g. $< a(abc)g(bc) >$).

A sequential pattern does not contain information about temporal duration. Therefore, it is not suited for a large range of domains. In addition, a sequential pattern used for perdition/forecasting does not give information about the temporal localization of the events to predict.

As an example, suppose the sequential pattern $< abc >$ (i.e. a followed by b followed by c). This pattern can be extended into a sequential association rule: $< ab \to c >$ (i.e. if an event of type a is followed by an event of type b, then an event of type c will occur). When this rule is applied, it predicts an event of type c, but it does not give more information about the temporal location of this event: The event c might happens right after b, or after an arbitrarily long period of time.

2.2.2 Patterns for Symbolic time sequences

This section presents several patters for symbolic time sequences mining.

2.2.2.1 Time-Interval Sequential Patterns

Chen et al. (Chen *et al.*, 2003) extend the sequential patterns used in symbolic time series mining to the mining of symbolic time sequence. This new pattern is called *Time-Interval Sequential Patterns* (TISP). A *TISP* is essentially a sequential pattern $< A_0, I_1, A_1, I_2, A_2, \cdots, I_n, A_n >$ where $\{A_i\}_i$ are regular symbols, and the $\{I_i\}_i$ are temporal intervals i.e. $I_i = [a, b]$. I_i defines the range of time allowed between the events A_{i-1} and A_i. In Chen's et al. definition, the temporal intervals $\{I_i\}_i$ are limited to be elements of

$TI = \{[0, T_1], [T_1, T_2], [T_2, T_2], \cdots, [T_{r-1}, T_r], [T_r, \infty]\}$ where $\{T_i\}_i$ are user defined boundaries.

Suppose $TI = \{I_0 = [0, 3], I_1 = [3, 6], I_2 = [6, 9], I_3 = [9, \infty]\}$. The TISP $< a, I_2, b, I_1, c >$ defines an event a followed between 6 and 9 time units by an event b, followed between 3 and 5 time units by an event c.

TISPs cannot express relations such as "an event a followed between 0 and 10 time units by an event b, followed between 0 and 5 time units by an event c".

2.2.2.2 Generalized Sequential Pattern with Item Intervals

Hirate (Hirate and Yamana, 2006) defines another extension of the sequential patterns used in symbolic time series mining. This pattern is called *Generalized Sequential Pattern with Item Intervals* (GSPII). A GSPII is essentially TISP where the events can be partially ordered. In addition, four constraints are introduced:

- C1: The minimum interval between two events.
- C2: The maximum interval between two events.
- C3: The minimum interval between the head and the tail of a sequence.
- C4: The maximum interval between the head and the tail of a sequence.

The main interest of these four constraints is to limit the temporal distance between non ordered events. The C1 and C3 constraints are set to 0 in Hirate's experiments.

In Hirate's definition, the temporal intervals $\{I_i\}_i$ are limited to be $I_i = [i\Delta, (i+1)\Delta]$, where Δ is a user constant. In the experiments, Δ is set to 60×60 (one hour), $60 \times 60 \times 8$ (eight hours), $60 \times 60 \times 256$ (256 hours) and $60 \times 60 \times 24$ (one day).

Suppose $\Delta = 60 \times 60$ (one hour if the time unit is the second). The times intervals are $[0 - 1\ hour], [1 - 2\ hours], [2 - 3\ hours], etc.$ The GSPII $< a, [1 - 2\ hours], (bc), [0 - 1\ hour], d >$ defines an event a followed between one and two hours by two events b and c, followed between zero and one hour by an event d. In addition, this GSPII should satisfy the four constraints C1-C4.

Similarly to TISPs, GSPIIs cannot express relations such as "an event a followed between 0 and 10 time units by an event b, followed between 0 and 5 time units by an event c".

2.2.2.3 Temporal Constraint System

A *Temporal Constraint System* (TCS) is a collection of existing conditions (over events) and a set of constraints (conditions) on these existing conditions. In the literature, the constraints are generally binary operators over events, but constraints based on operators with more than two arguments can also be built (and they are not always equivalent to aggregation of binary constraints). As an example, consider the ternary constraint over the three events located at times t_1, t_2 and t_3: "the distance between t_1 and t_2 should be greater than the distance between t_2 and t_3".

A TCS restricted to constraints as binary operators can be represented by directed graphs where each node is labelled with a symbol (i.e. existing condition of an event with such symbol), and each edge is labelled with a constraint between two existing conditions. We will use the graph's terminology to describe such a TCS.

The simplest constraints over two temporal points (called *time point binary operators*) are *before*, *equals* and *after*.

A *rule* is a TCS divided into two parts. One of the parts is called *head*, the other is called *body*. A rule predicts that its head will be observed in the dataset when its body is found. Rules are sometime augmented with additional information or conditions such as confidence, time distribution, etc. Every TCS can be transformed into a rule, and all rules can be transformed into a TCS. However, a good rule (rule with high confidence, high support or any other measure) does not necessary give a good GCS, and vice versa.

2.2.2.4 Temporal Constraint Network

Dechter et al. (Dechter *et al.*, 1991) define a TCS called *Temporal Constraint Network* (TCN). A TCN is a TCS where constraints are restricted to be binary operators such as $t_1 \in \mathbb{R} \times t_2 \in \mathbb{R} \mapsto \begin{cases} 1 & \text{if } t_1 - t_2 \in C \\ 0 & \text{otherwise} \end{cases}$, where C is a subset of $\mathbb{R}$. TCN can be represented as a graph. The structure of this graph is unrestricted. The restriction of this problem on solving TCN where the constraints are restricted to interval (i.e. of the form $[a, b]$) is called STP (Simple Temporal Problem).

Dechter et al. (Dechter *et al.*, 1991) define a sub-class of TCN called STP (for Simple Temporal Network). STPs are TCNs where constraints are restricted to be intervals. An interval is a set of numbers such that any number that lies between two numbers in the set is also included in the set.

2.2.2.5 Episodes

An *Episode* is a TCS where constraints are restricted to be the 'before' operator i.e. $t_1 \in \mathbb{R} \times t_2 \in \mathbb{R} \mapsto t_1 < t_2$ (episodes are less expressive than TCNs). In other words, an episode is a partially ordered collection of events. The edges can only be labelled with the *before* condition. A *serial episode* is a completely ordered episode i.e. any two nodes are connected by an edge labelled with a *before* constraint. A *parallel episode* is an episode with no order i.e. the graph does not have any edge. A *k-episode* is an episode with k symbols (k existing conditions). Several authors use Episodes with an extra constraint to avoid episodes with very spread events. Such augmented episodes can still be expressed by a TCS.

Fig. 2.3 shows three examples of episodes and one example of a rule.

Garriga's Episodes

Garriga (Casas-Garriga, 2003) uses episodes with an additional constraint between time points. In the technique proposed by Garriga, an Episode is defined as a partially ordered collection of events such that the maximum distance between the two ordered events is bounded. This maximum distance is called 'time unit separation'. This constraint can be expressed by adding extra edges in the graph representing this TCS.

Mannila's et al. Episodes

Mannila et al. (Mannila *et al.*, 1997; Mannila and Toivonen, 1996) also uses episodes with an additional constraint between time points. For Mannila et al., an episode is defined as a partially ordered collection of events such that, the distance between any two events is bounded. This maximum distance is called the 'window size'. This constraint can be expressed by adding extra

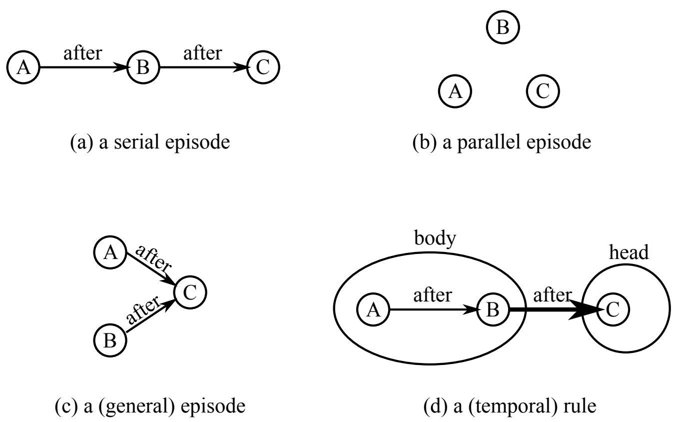


Figure 2.3: Four examples of temporal patterns

edges in the graph representing this TCS. For efficiency in the learning process, Mannila et al.'s episode are restricted to be completely ordered (serial episode) or not ordered at all (parallel episode).

Based on this definition of episode, Mannila et al. define two types of temporal rules:

The first type of rule is used in the WinEpi algorithm. Such rules are defined as $\alpha \Rightarrow \beta$, where α and β are episodes, and β subsumes (is more general than) α. The semantic is: If the episode α is found in the window $[w, w + \text{window size}]$, then the episode β will be found in the same window. In the case of ordered episodes, the temporal relationship between α and β gives additional restriction on the temporal location of β.

The second type of rule is used in the MinEpi algorithm. These rules are defined as $\alpha[win_1] \Rightarrow \beta[win_2]$, where α and β are episodes, and β subsumes (is more general than) α. The semantic is: If the episode α is found in the minimal occurrence $[t_b, t_e]$ with $t_e - t_b \leq win_1$, then the episode β occurs in the window $[t_b, t_e + win_2]$. In the case of ordered episodes, the temporal relationship between α and β gives additional restriction on the temporal location of β.

Méger (Méger and Rigotti, 2004) uses rules equivalent to Mannila's rules based on Mannila's serial episodes.

Vilalta et al. (Vilalta and Ma, 2002) is using rules equivalent to Mannila's rules based on Mannila's parallel episodes.

2.2.2.6 Chronicles

Dousson et al. (Dousson and Duong, 1999) defines a TCS called *Chronicle*. The definition of Chronicle is equivalent to the definition of the Temporal Constraint Network (TCN). However, for efficiency in the learning process, a Chronicles' constraints are restricted to be convex subset of $\mathbb{R}$.

2.2.2.7 Probabilistic Association Rule

Oates et al. (Oates *et al.*, 1997) define a simple type of probability rule of the form $x[t] \Rightarrow y[t+a-a/2, t+a+a/2]$. Here, a condition x is a non temporal conjunction of first order logic atoms. If the condition x is true a time t, then the rule predicts the observation of an event y between $t+a-a/2$ and $t+a+a/2$. In Oates's algorithm, the variable a is a parameter fixed by the user. Therefore, the core of this algorithm is a non temporal learning, therefore we are not explaining the behavior of this algorithm.

2.2.2.8 Temporal Node Bayesian Network

Arroyo-Figueroa et al. (Arroyo-Figueroa and Suear, 1999) propose an extension of the Bayesian Networks (Pearl, 1988) to the case of temporal events. This type of Bayesian network is called Temporal Node Bayesian Network (TNBN). Each node of a TNBN is associated with a type of event and a set of Boolean random variable. Each of these random variables is associated with an interval (i.e. a convex subset of $\mathbb{R}$).

Let n_1 and n_2 be two nodes of a TNBN, and $v_{1\to 2}$ be a vertex from n_1 to n_2. n_1 is associated with the event A. n_2 is associated with the event B and has a random variable X_{n_2} associated with the interval $T := [t_1, t_2]$. Suppose an event of type A occurs at time t. The random variable X_{n_2} will be true if and only if an event of type B occurs at time t' with $t' - t \in T$. The fig. 2.4 presents an example of TNBN.

This kind of pattern can express sets of simple temporal relation between events. Several intervals can be associated with each node , and each of these intervals have different random variables and therefore different probabilities. Therefore, TNBNs can express temporal relation between two events that none of the previously presented temporal patterns could express. However, since temporal interval are mapped to nodes (and not to edges as it is the case in TCS), the temporal relation between a node and all its parents are the same. Because of that, the two following relations cannot be expressed in the same TNBN: "If A occurs at time t, then C will occur between $t+5$ and $t+10$" and "If B occurs at time t, then C will occur between $t+10$ and $t+20$". In addition, TNBNs cannot express rules with conjunction of conditions in the body. For example, the following relation cannot be expressed with TNBNs: "If A occurs at time t, and B occurs at time t' with $t < t' < t+10$, then C will occur between $t'+5$ and $t'+10$".

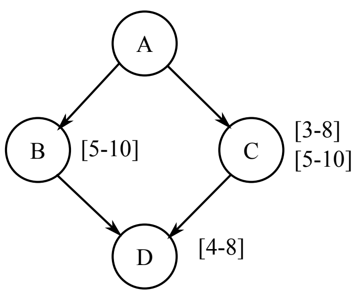


Figure 2.4: An example of TNBN.

2.2.2.9 Time Finite Automata with Granularities

Bettini et al. (Bettini *et al.*, 1998) propose the definition of *Time Finite Automata with Granularities* (TFAGs). A TFAG is a TCS where constraints are *Temporal Constraints with Granularity* (TCG).

Informally, a *Temporal type* is a discretization of the time. Common temporal types are *hour*, *day*, *week*, *mouth*, *years*, etc.

Given a time t and a temporal type μ, $\lceil t \rceil^{\mu}$ is the projection of t according to μ. For examples, $\lceil 2012/07/18\ 20:00 \rceil^{day} - \lceil 2012/07/21\ 07:00 \rceil^{day} = 3$, $\lceil 2012/07:18\ 20:00 \rceil^{day} - \lceil 2012/07/18\ 23:00 \rceil^{day} = 0$ and $\lceil 2012/07/18\ 23:59 \rceil^{day} - \lceil 2012/07/18\ 00:01 \rceil^{day} = 1$.

Suppose a temporal type μ. Suppose two natural numbers $n \in \mathbb{N}$ and $m \in \mathbb{N}$. Suppose two events respectively occurring at t_1 and t_2. A *Temporal Constraints with Granularity* $[m, n]\mu$ over these two events is true if and only if $\lceil t_2 \rceil^{\mu} - \lceil t_1 \rceil^{\mu} \in [m, n]$.

If the granularity is precise enough, every TCN can be converted into a TAG. In addition, all TAGs can be converted into a TCN (See proof 4.5.1).

2.2.2.10 Calendric Association Rules

Özden et al. (Özden *et al.*, 1998) propose a type of association rule called *Cyclic Association Rules* or *Calendric Association Rules*. These rules are essentially usual association rules (association rules used in transaction datasets) with a cyclic condition. The cyclic condition is a temporal condition that depends on the calendar. Examples of Cyclic Association Rules are '$(day = monday) \wedge A \rightarrow B$' or '$(day = monday) \wedge (hour = 8) \wedge A \rightarrow B$'.

2.3 Temporal Data mining algorithms

2.3.1 Apriori algorithm

The *Apriori algorithm* (Agrawal and Srikant, 1994) is a classical algorithm used to learn frequent itemsets (set of symbols) and association rules on itemsets on transaction datasets. It is not a Temporal Data mining algorithm but it inspired a consequent number of Temporal Data mining algorithms. Report to definition 2.0.1 for the definition of transaction dataset and itemset. A *k-itemset* is an itemset with k items (or symbols). The *support* of an itemset is the number of occurrences of this itemset in the dataset. A *frequent itemset* is an itemset with a support greater than the 'minimum support' parameter. The *apriori property* tells that any subset of a frequent itemset must be a frequent itemset. For example, $\{a, b, c\}$ is a frequent itemset if and only if $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$ and $\{b, c\}$ also are frequent itemsets.

The apriori algorithm enumerates all the frequent itemsets. Instead of enumerating all possible itemsets (and test if they are frequent), the algorithm eliminates a part of non-frequent itemsets with the apriori property i.e. if all subsets of an itemset are not frequent, then this itemset is not frequent. The algorithm begins by enumerating all the frequent 1-itemsets. At each algorithm loop, it generates the k+1-itemsets based on previously exacted k-itemsets.

2.3.2 Data mining algorithms for Symbolic time series

This section gives an overview of some of the techniques developed to extract temporal patterns from Symbolic time series. These techniques can be classified into two categories: The Apriori based algorithms and the Pattern growth based algorithms. Apriori based algorithms explore the frequent patterns with a Breadth-first search strategy (see the definition of the Apriori algorithm in section 2.3.1). The exploration is optimized with the *Apriori property*. Pattern growth based algorithms explore the frequent patterns with a Depth-first search strategy.

The *GSP algorithm* (Srikant and Agrawal, 1996) is a direct extension of the Apriori algorithm to the case of symbolic time series.

The *Spade algorithm* (Zaki, 1998) is an improvement of the GSP algorithm: Instead of scanning several time the dataset, the Spade algorithm builds a *vertical dataset* from the original dataset (also called *horizontal dataset*). The vertical dataset indexes directly the frequent patterns and their occurrence in the dataset. With this format of dataset, the algorithm is executed more efficiently.

The *Prefix Span algorithm* (Pei *et al.*, 2001) is a pattern growth based algorithm: When a pattern is considered by the algorithm, the dataset is *projected* according to this pattern i.e. informally, the parts of the dataset that are not supporting the pattern are removed, and the parts of the dataset that does support the pattern are optimized. In the Prefix Span algorithm, the dataset is recursively projected while patterns are enriched.

2.3.3 Data mining algorithms for Symbolic time sequences

This section presents some of the techniques developed to extract temporal patterns from Symbolic time sequences.

As presented in section 2.2, a symbolic time sequence pattern can be a *temporal constraint system* (TCS) or a *temporal rule*. A pattern a is said to subsume a pattern b if all matching of b is also a matching of a i.e. a is more general than b. Except for trivial cases, the number of temporal patterns in a dataset is exponential with the number of events. Therefore, mining algorithms needs to focus on extracting the more 'interesting' patterns. Depending on the author and depending on the type of pattern, there are several measures of the level of how interesting is a pattern.

A TCS (e.g. Chronicle, Episode, etc.) is usually considered to be interesting if it is matched at least a minimum number of time. The number of occurrences of a TCS is called the *support* (the support of a TCS is an integer number). In addition, an interesting TCS a may need to be closed (or minimal) i.e. a does not subsume any TCS b such that the support of b is equal to the support of a.

The *confidence* of a rule is the percentage of occurrences of the rule's body which also matches the body and the head of the rule i.e. $\frac{support(body)}{support(head+body)}$. The *support* of a rule is the percentage of occurrences of the rule's head which also matches the body and the head of the rule i.e. $\frac{support(head)}{support(head+body)}$. The *(temporal) precision* (or *accuracy*) of a rule is a measure of freedom of the rule's head considering the body fixed e.g. inverse of the average range of prediction. A rule is usually interesting if it holds a minimum confidence, minimum support and minimum precision (maximum prediction range). The large majority of the algorithms that learn symbolic time sequence patterns need for the user to define arbitrarily the precision of the generated rules.

Since rules are by nature more complex than TCSs (a rule is a TCS with some extra information), some rule learning algorithms begin by extracting TCSs, and then, they convert theses TCSs into rules. In this approach, the learning of TCSs is the core problem, while the conversion into rule is considered to be straight forward.

Notice that a good TCS (according to the previous definition) does not necessary give a good rule. Symmetrically, some good rules may be shaped from bad TCSs.

Example 2.3.1 *Suppose a symbolic time sequence dataset containing four types of events A, B, C and D. 1000 events of type A and B randomly located. 20 events of type C randomly located. Also, suppose that for half of the events of type C located at time t, there is an event of type D at time $t-5$.*

A very good rule (confidence 100%, support 50%, infinite precision) is 'If there is an event D at time t, then there is an event C at time $t+5$'. The TCS correlated with is rule is 'There are an event C at time t and an event D at time $t+5$'. This TCS has a low support (10) in comparison to any TCSs based on A and B (support close to 1000).

This example shows a good rule correlated with a bad TCS. It proves that mining TCSs in order to extract good rules is not an optimum (complete) solution i.e. all good patterns are not guaranteed to be found.

In the general case, rules that predict rare events (events with a relative low number of occurrences) are not associated with good TCS. This is due to the fact that a TCS is generally considered to be good if it occurs frequently. By definition, TCSs associated with rules predicting rare event have a low number of occurrences. Thus, techniques extracting TCSs to build rules are bad at learning rules that predict rare events.

Several mining algorithms for symbolic time sequences have been inspired from the *Apriori algorithm*. Such algorithms are said to be 'apriori like algorithms' (see the definition of the Apriori algorithm in section 2.3.1).

2.3.4 Face algorithm

The *Face algorithm* has been developed by Dousson et al. (Dousson and Duong, 1999) to extract Chronicles from symbolic time sequences. This algorithm is inspired from the Apriori algorithm. Chronicles are particularly interesting because they are more expressive than Episodes (which is the commonly used pattern model). An *Unconstrained Chronicle* is a Chronicle without order or time constraints (between the existing conditions). An *Unconstrained Chronicle* is equivalent to a *parallel episode*. A k-Chronicle is a chronicle with k existing condition (k symbols).

There is no meaning of search completely unconstrained pattern in a time sequence. The author is not clear about time point. Therefore we have been considering an extra condition of maximum size such as (Casas-Garriga, 2003) or (Mannila *et al.*, 1997) in our implementation.

1. The algorithm begins by constructing 1-Chronicles.
2. Then, on every algorithm loop k (staring at k=1):
    (a) The algorithm computes the unconstrained k+1-Chronicles from the unconstrained k-Chronicles.
    (b) Next, it gets rid of the non-frequent k+1-Chronicles i.e. Chronicles with low support.
    (c) Then, it computes a k+1-Chronicle from the k-Chronicles and each unconstrained k+1-Chronicles.
3. The looping process continues until there is no k+1-Chronicles.

The computing of the frequent unconstrained k+1-Chronicles from the unconstrained k-Chronicles (step 2.a) is based on the apriori property i.e. an unconstrained k+1-Chronicle is frequent is and only if all its sub chronicles are frequent. Unconstrained chronicles don't have any conditions; therefore, the computation of the sub chronicles of an unconstrained chronicle is simple to compute.

The computation of the k+1-Chronicles from the k-Chronicles and the unconstrained k+1-Chronicles (step 2.c) is different for the case k=1 and for the other cases (k $>$ 1). In the case of k=1, the condition $A \subset \mathbb{R}$ of a 2-Chronicle c' based on an unconstrained 2-Chronicle c is done such that: (a) A is a non empty convex subset of $\mathbb{R}$ i.e. $A := [t_1, t_2]$ (A convex subset of $\mathbb{R}$ is a subset such as $\forall x < y < z \in \mathbb{R}, x \in A \wedge z \in A \Rightarrow y \in A$). (c) The support of c if greater than the support of c' times the parameter it_{min}. (d) The surface of A (i.e. $t_2 - t_1$) is minimized.

In the case of k $>$ 1, the computation of a k+1-Chonicle c' from a set of k-Chronicles C and an unconstrained k+1-Chonicle c'' (step 2.c) is done such that: (a) The symbols of c' are the same as the ones of c''. (b) The condition A between two symbols s_1 and s_2 of c' is defined as $A = \bigcup_i B_i$ where B is the set of condition between the symbols s_1 and s_2 in the k-Chronicles C.

2.3.5 WinEpi algorithm

WinEpi is a well known algorithm developed by Mannila et al. (Mannila *et al.*, 1997) to learn Episodes and association rules (based on the episodes). Basically, WinEpi discretizes a symbolic time sequence into a transaction dataset with a fixed length window (parameter of the algorithm), and apply the Apriori algorithm.

The three parameters of the algorithm are the *window size*, the *window increment* and the *minimum support*. Suppose a time sequence from T_b to T_e (boundaries). The window is positioned at the beginning of the time sequence. Then, the window's position is incremented until it reaches the end of the sequence. At every new increment, a new itemset is generated such that it contains all the symbols that are present at least one time in the window. The fig. 2.5 shows an example of window discretisation. Once the itemsets are computed, WinEpi computes the frequent itemsets with the apriori algorithm. Every frequent itemset is converted into a parallel

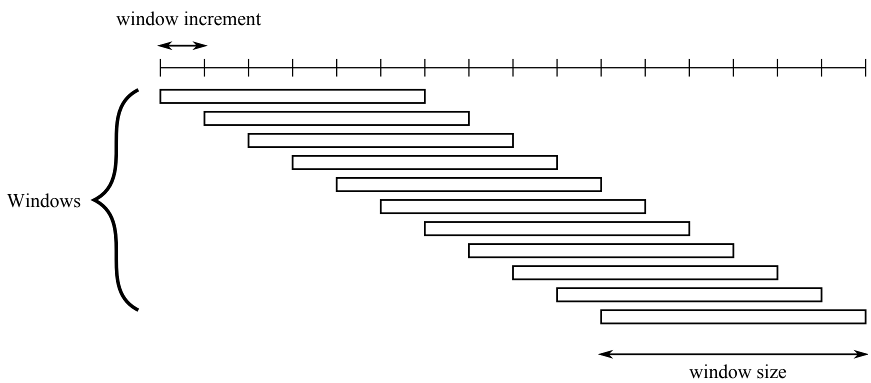


Figure 2.5: An example of WinEpi window discretisation.

Episode with the extra condition that the distance between any two events is bounded by the window size.

The process is similar for the research of serial episodes: The window discretisation produces completely ordered item lists instead of itemsets

The second part of the WinEpi algorithm computes association rules from the extracted episodes.

One of the main drawbacks of this algorithm is that the user fixes arbitrarily the temporal condition in the episodes. Therefore, the algorithm is not able to extract episodes with other temporal condition.

2.3.6 MinEpi algorithm

The *MinEpi algorithm* is an improvement of the WinEpi algorithm proposed by Mannila et al. (Mannila *et al.*, 1997). MinEpi's grammar is richer than the one used by Winepi, and MinEpi does not need to go through the all dataset several times. However, MinEpi is more space consuming than WinEpi.

MinEpi extracts episodes with a minimum number of *minimal occurrences*. An interval $[t_1, t_2]$ is a *minimal occurrences* of an episode α if and only if (a) α occurs in $[t_1, t_2]$, and if (b) α does not occurs in any subsets of $[t_1, t_2]$. A minimal occurrence is an occurrence. Like WinEpi, MinEpi is an apriori like algorithm. The algorithm is:

1. Extract 1-episodes.
2. Computes minimal occurrences for these 1-episodes.
3. Get rid of the 1-episodes without a minimum number of minimal occurrences.
4. Do
   (a) Computes k+1-episodes from k-episodes.
   (b) Minimums occurrences of k+1-episodes from minimal occurrences of k-episodes.
   (c) Get rid of the k-episodes without a minimum number of minimal occurrences.
5. Until there is no more k-episodes.

The second part of the MinEpi computes association rules from the extracted episodes. As presented in the *Temporal patterns* section, MinEpi rules are defined as $\alpha[win_1] \Rightarrow \beta[win_2]$, where α and β are episodes. The user needs to fix the parameters win_1 and win_2 (win_1 and win_2 are the same of all the rules).

Similarly to WinEpi, one of the main drawbacks of the MinEpi algorithm is that the user fixes arbitrarily the temporal conditions of the episodes (window sizes parameters). This prevents the algorithm to learn any temporal patterns with different temporal conditions than the one specified by the user.

2.3.7 EpiBF algorithm

The *EpiBF algorithm* has been developed by Garriga (Casas-Garriga, 2003). This algorithm is inspired from WinEpi and MinEpi. One of the main differences between EpiBF and WinEpi/MinEpi is that, EpiBF uses local constraints between couple of elements instead of having a global window constraint between all the elements of an episode. A Garriga's episode is defined as a partially ordered collection of events such that, the maximum distance between the two ordered events is bounded. This maximum distance is called 'time unit separation' and need to be fixed by the user.

This definition of episodes allows to compute more precise association rules than WinEpi or MinEpi association rules. On the other hand, this kind of episodes is more expensive to compute than WinEpi like episodes.

2.3.8 WinMiner algorithm

The *WinMiner algorithm* has been developed by Méger et al. (Méger and Rigotti, 2004). WinMiner extracts rules (called FLM rules) based on serial episodes (completely ordered episodes) with a maximum gap between all events (same as Mannila's definition (Mannila *et al.*, 1997)), and a structural constraint: The head of an FLM rule is always the single symbol such that there are no other symbols with higher order. Suppose the serial episode $A \rightarrow B \rightarrow C$. From this episode, the only FLM rule is $A \rightarrow B \Rightarrow C$, where $A \rightarrow B$ is the body and C the head.

Unlike to WinEpi, WinMiner extracts FLM rules in a depth first search. Like MinEpi, WinMiner computes occurrences of size k+1 from occurrences of size k.

2.3.9 GenFCE and GenREAR algorithms

The *GenFCE and GenREAR algorithms* have been developed by Harms et al. (Harms *et al.*, 2001). GenFCE extracts *closed episodes* from symbolic time sequences. GenREAR uses GenFCE to extract closed episodes and uses them to build a set of *representative association rules*.

Definition 2.3.2 *Suppose the mapping β that associate to a set of time windows W (set of convex intervals), the set of episodes E such that every* *episode of E occurs in all windows of W. Suppose the mapping α that associates to a the set of episodes E, the set of time windows W such that for every time window w of W, every episode of E occurs at least one time in w. A* closed set of episodes *is a set of episodes E such that $\beta(\alpha(E)) = E$.*

The cover *of a rule $r : X \Rightarrow Y$, denoted by $C(r)$, is the set of association rules that can be generated from r i.e.*

$$C(r) = \{X \cup U \Rightarrow V | U, V \subseteq Y, U \cap V = \emptyset, \text{ and } V \neq \emptyset\}$$

A set of representative association rules *with a minimum support s and a minimum confidence c is a set of rules R such that every rule r of R has a confidence and support greater than c and s, and there are no rules $r' \in C(r)$ such that the confidence and support of r' are greater than c and s.*

GenFCE is based on the apriori algorithm. From the result of GenFCE, GenREAR enumerates all association rules, beginning with the more complex. A rule r is added to the set of representative association rules if r is not covered by the already enumerated rules.

2.3.10 Frecpo

The *Frecpo* (frequent closed partial order) algorithm has been developed by Pei et al. (Pei *et al.*, 2006) to extract closed partial orders from sets of symbolic time series. This algorithm cannot be applied directly on time sequences. However, given a global or local window size, symbolic series can be extracted from symbolic time sequences through discretization (Mannila *et al.*, 1997; Casas-Garriga, 2003). Frecpo is reported to be efficient; therefore it will be presented here.

The core of Frecpo is simple: First, Frecpo enumerates the *transitive closure* of each symbolic series. For example, given the series $S = (a, c, d, b)$. The transitive closure of S is the set $\{ac, ad, ab, cd, cb, db\}$. Next, Frecpo considers this set of transitive closure as an itemset, and searches for frequent closed itemsets. Finally, each frequent closed itemsets (set of transitive closure) are converted to partial order. For example, suppose the frequent closed itemset $S = \{ab, bc, db\}$. The fig. 2.6 shows the partial order generated from S. The enumeration of the frequent itemsets is based on a depth-first

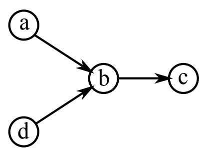


Figure 2.6: Partial order generated from the transitive closure $\{ab, bc, db\}$

search. This depth-first search is optimized by a pruning. This pruning is based on several properties on transitive closures.

2.3.11 TNBN learning

Pablo Hernandez and his colleagues (Hernandez-Leal *et al.*, 2011) are developing an algorithm to learn TNBNs. As detailed earlier in the chapter, TNBNs are relatively limited to express temporal pattern with more than two events, but the algorithm is an interesting solution to select the temporal intervals (we call this problem the "interval selection problem", and we present and discuss it in chapter 4). We are only presenting here the learning of intervals (we suppose the structure of the TNBN to be already known).

Let n_1 and n_2 be two nodes of a TNBN, and $v_{1\to 2}$ be a vertex from n_1 to n_2. n_1 is associated with the event A, and n_2 is associated with the event B. The current problem is to select the "good" intervals for the node n_2. A good interval is not necessarily an interval that implies a high confidence or a high precision. Pablo Hernandez et al. are supposing that the distribution $P(t' - t|A$ occurs at t and B occurs at $t')$ can be approximated by a Gaussian mixture model. The Expected-Maximization (EM) algorithm (Dempster *et al.*, 1977) is used to estimate the Gaussian mixture model. Next, an interval $[\mu - \sigma, \mu + \sigma]$ is added to the node for each Gaussian of the mixture model, where μ and σ are respectively the mean and the standard deviation of the Gaussian model. It is important to remark that this algorithm is based on several relatively weak hypotheses on the dataset:

1. The good intervals can be found by the analysis of the shape of the distribution.

2. The distribution is a Gaussian mixture model.

3. Given a Gaussian, the good interval is considered to be $[\mu - \sigma, \mu + \sigma]$ (and not $[\mu - 0.5\sigma, \mu + 0.5\sigma]$ or $[\mu - 2\sigma, \mu + 2\sigma]$ for example).

These hypotheses are discussed in the chapter 4.

2.3.12 Rules for rare events

Vilalta et al. (Vilalta and Ma, 2002) developed an algorithm that is especially efficient at learning rules that explain rare events. The algorithm extracts directly association rules without going through a step of generating TCSs. As explained in example 2.3.1, a good rule is not necessary associated with a good TCS, especially in the case of a rule predicting rare events.

This algorithm is based on a window 'fixed' on the events to predict (called target events). We define *the targets* to be the symbols of the events to predict. The window size W is a user parameter.

First, the algorithm computes the *frequent event-sets*.

An *event-set* is a set of symbols. Following the definition of the author, an event set $\{s_i\}$ *occurs* at time t if for all symbol s_i, there is an occurrence of an event of type s_i in the interval $[t - W, t]$ i.e. $\forall s \in \{s_i\} \, \exists s_i[t']$ with $t' \in [t - W, t]$.

A target event $e_i[t_i]$ is *supported* by an *event-set* E is and only if E occurs at time t_i. Suppose a set of target events $\{e_i[t_i]\}$ (the events that we try to predict). The *support* of an *event-set* is the percentage of targets events *supported* by this event-set. Given a minimum support $minS \in [0, 1]$, a *frequent event-set* is an *event-set* with a support greater or equal to $minS$.

Similarly to the apriori algorithm, this algorithm iteratively scans the data-set to evaluate the support of events-sets, generate new events-set candidates and prune them with the apriori property. The figure 2.7 shows an example of dataset and event-sets.

Once the frequent event-sets S are computed, a rule is associated with each of them i.e. for each $\{s_i\}$, the rule $\{s_i\} \rightarrow target$ is created. By convention, the confidence of an item-set is the confidence of the rule associated with the item-set. The confidence of each frequent event-set is computed. We define the *positive dataset* to be the dataset that contains only the events $e[t]$ of the (original) dataset such that there is a target event in the interval $[t, t + W]$. The confidence of an event-set is $\frac{x}{y}$ where y is the number of occurrences of the event-set in the dataset, and x is the number of occurrences of the event-set in the *positive* dataset.

Finally, the remaining event-sets are partially ordered according to confid-

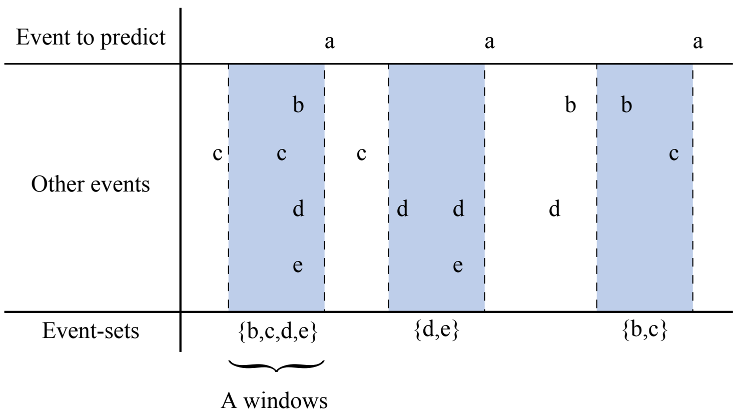


Figure 2.7: An example of dataset and event-sets for (Vilalta and Ma, 2002)

ence and support. Next, a process selects iteratively a *maximal event-set* s from the ordered list of event-sets, stores s, removes s from the list, and removes all the sub-sets of s contained in the list i.e. more general event-sets. The algorithm stops when there are no more event-sets to select. The rules associated with these item-sets are returned.

2.3.13 I-Apriori and I-PrefixSpan

Chen et al. (Chen *et al.*, 2003) have adapted the two well known Apriori and PrefixSpan algorithms to the case of Symbolic Time Sequences. The pattern used by these algorithms is called the Time-Interval Sequential Patterns.

The main difference between the Apriori algorithm and the I-Apriori algorithm is the generation of candidates. The generation of candidates of size $k = 2$ is done with a direct enumeration of all possible candidates i.e. $L_1 \times TI \times L_1$, where TI is the set of possible time interval and L_1 is the set of frequent symbols. The generation of candidates of size $k > 2$ is optimized with the Apriori property and the candidates of size $k - 1$. To optimise the evaluation of the support of candidates, the candidates are stored in a hash tree structure.

The main difference between the PrefixSpan algorithm and the I-PrefixSpan algorithm is the projection of the dataset. In PrefixSpan, when a dataset is projected according to a prefix, one tree is created. In the case of I-PrefixSpan, because of the temporal aspect, several trees can be created. Each of these trees needs to be computed to find the frequent *cells* (A cell is a tuple of a time interval and a symbol).

2.4 Automated Planning and Scheduling

Automated Planning and Scheduling is the study of computer programs designed to build strategies that lead to desired states. Some algorithms produce plans with several degrees of freedoms. Given a plan with several degrees of freedom, a *scheduling algorithm* produces a plan without degree of freedom.

Example 2.4.1 *An example of plan with no degree of freedom is "Execute action A1 at time 40". An example of plan with one degree of freedom is "Execute action A1 between times 20 and 30". Example of a plan with two degrees of freedom is "Execute action A1 between times 20 and 30, and execute action A2 between 5 and 10 after A1".*

Planning and scheduling programs are generally used by autonomous systems (robots, unmanned vehicle, etc.) and decision support systems.

Non-temporal planning

Classical planning problems can be solved by forward chaining, backward chaining, SAT reduction, model checking, heuristics, among other techniques.

Strips (Fikes and Nilsson, 1971) is a simple algorithm that works with *backward chaining* (or retrograde analysis). The underling idea of the backward chaining is the following one: Strips begins by searing for the rules that can produce the main goal (the main goal is one input of the algorithm). If these rules are directly applicable, they are included in the plan. Otherwise, a set of sub goals corresponding to the rules requirement to be applicable replace the original goal. The operation is repeated until a valid plan is found. This technique is called backward chaining because it achieved a research starting by the conclusion (goal), and looking recursively for the required pre-conditions.

GraphPlan (Blum and Furst, 1997) works with a straight forward chaining to build a graph that represent all the possible reachable states.

SatPlan (Kautz *et al.*, 2006) converts a planning problem into a SAT problem (Boolean satisfiability problem), which is then solved with usual SAT solvers.

Probabilistic Planning and Scheduling is in concern with worlds with uncertainty e.g. worlds with non-deterministic rules (Markov Decision Processes (MDP)) and/or worlds with incomplete/probabilistic observation (Partially Observable MDP).

Temporal planning

Temporal Planning and Scheduling extends classical planning with a temporal aspect. The temporal aspect allows non zero-duration and overlapping actions, and inaccurately located events.

Deviser (Vere, 1983) is a planning and scheduling algorithm that solves goals such as "Make X true between t_1 and t_2 for at least duration n". The algorithm allows non-zero duration events. The algorithm needs a deterministic (non-probabilistic) description of the world (states and rules). The planning stage of the algorithm produces a plan (represented as PERT chart). A PERT chart is a graphical representation used in project management that is equivalent to a *Temporal Constraint System* (see definition in the Temporal learning related work). The plans of the first stage of Deviser have several degrees of freedom. From such plans, the scheduling stage of the algorithm produces a plan with zero degree of freedom i.e. a list of nominal time of execution for every action of the plan. Deviser has been applied experimentally in planning activities for the Voyager spacecraft in its encounter with Uranus in 1986.

The Temporal Constraint Satisfaction Problem (TCSP) is the problem of determining if the time-samples of a given TCN can be assigned in such way as to make all the constrains of the TCN valid. TCSP is a scheduling problem. It is also intractable in the general case (NP-hard problem). Solving STP (TCN where constraints are restricted to be intervals) is a polynomial problem.

Balaban et al. (Balaban and Rosen, 1999) define a sub-class of TCN for the TCSP. This class defines the TCN that can be solved with a divide and conquer strategy. The problem of checking this sub-class is called STCSP (Structured Temporal Constraint Satisfaction Problem).

Algorithms such as DT-POP (Schwartz and Pollack, 2000) propose heuristic based strategies to deal with this problem.

The world is generally considered to be partially uncertain. Algorithms able to do planning with uncertainty have been proposed. STPU (Vidal and Fargier, 1999) consider the Temporal Constraint Satisfaction Problem where some nodes' time-stamps are unknown and cannot be specified. In the case of STPU, constraints are restricted to be non disjunctive.

2.5 Conclusion

This chapter presented and summarized common temporal patterns used in machine learning, frequent-sets mining algorithms, temporal mining algorithms, classical planning algorithm and temporal planning and scheduling algorithms.

The next chapter presents some common notations, the mathematical objects used in the thesis and the temporal pattern we have developed to express temporal relations and temporal plans.

3

Notation and event representation

Chapter Contents

This chapter presents the various notations and mathematical objects used in the thesis. Several of these notations and mathematical objects are already widely used, sometime with various syntaxes and namings. This chapter gives the terminology and notation used in this thesis. It also presents notation for some new concepts such as a Tita rules and Tita plans.

3.1 Probability distributions

Definition 3.1.1 *A* probability distribution *describes the probability of each value (or interval of values) of a random variable.*

Four particular continuous probability distributions are widely used in this thesis. They are presented in Fig. 3.1.

- The *uniform distribution* between two points a and b is noted $\mathbb{U}_{a,b}$ (see equation 3.1).

$$\mathbb{U}_{a,b} : x \mapsto \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

- The *normal distribution* is noted $\mathbb{N}_{\mu,\sigma^2}$ (see equation 3.2).

$$\mathbb{N}_{\mu,\sigma^2} : x \mapsto \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3.2}$$

- The *Dirac distribution* is noted δ_μ (see equation 3.3).

$$\begin{aligned} \delta_\mu : x \mapsto &\begin{cases} +\infty & \text{if } x = \mu \\ 0 & \text{otherwise} \end{cases} \\ \text{with } &\int_{\mathbb{R}} \delta_\mu(x)\, dx = 1 \end{aligned} \tag{3.3}$$

- The *histogram distribution* is noted $\mathbb{H}_{\{\alpha_i\},\{\beta_i\}}$ (see equation 3.4).

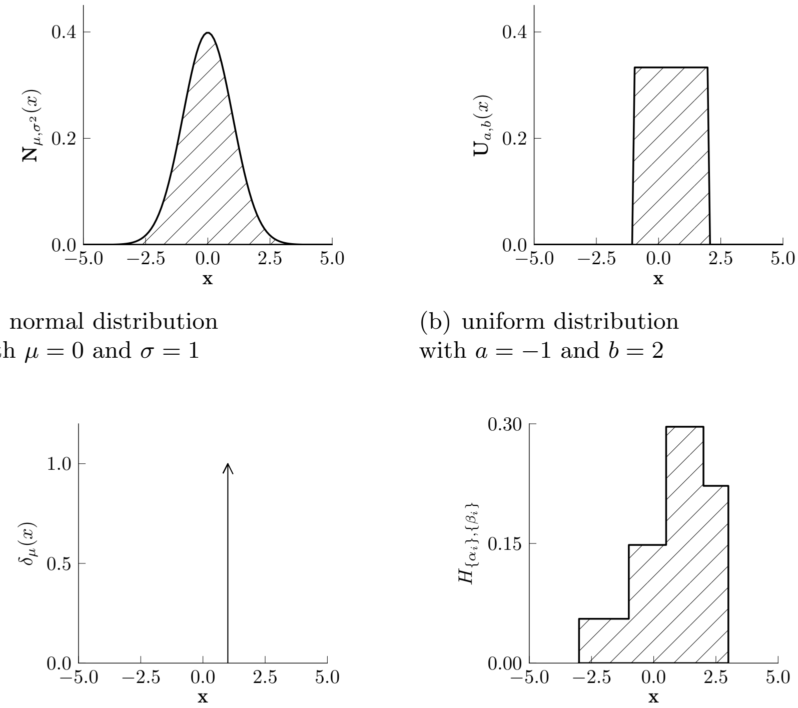
$$\mathbb{H}_{\{\alpha_i\},\{\beta_i\}} : x \mapsto \frac{1}{\sum\limits_{i=1,\cdots,n} \alpha_i} \sum_{i=1,\cdots,n} \alpha_i \mathbb{U}_{\beta_i,\beta_{i+1}}(x) \tag{3.4}$$

Suppose a probability distribution of a continuous variable $f : \mathbb{R} \to \mathbb{R}^+$. By convention, the probability distribution $f' := f + x$ with $x \in \mathbb{R}$ is defined as $f' : t \mapsto f(t-x)$. This operation can be interpreted as a translation of the distribution.

Definition 3.1.2 *The* range *of a probability distribution d is defined as $\int_{-\infty}^{+\infty} (\begin{cases} 1 & \text{if } d(x) > 0 \\ 0 & \text{otherwise} \end{cases})\, dx$. This can be understood as the 'surface of the non null part' of the distribution. The range of the uniform distribution $\mathbb{U}_{a,b}$ is $b-a$. The range of the normal distribution $\mathbb{N}_{\mu,\sigma^2}$ is ∞. The range of the Dirac distribution δ_μ is 0.*

3.2 Boolean function

Definition 3.2.1 *A* Boolean function *is a function $\mathbb{R} \to \{0,1\}$. In this work, Boolean functions are used to represent sub-sets of $\mathbb{R}$: A Boolean function b represents the set $\{x|b(x) = 1\}$. We chose to represent sub-sets of $\mathbb{R}$ by functions because it implies a simpler writing of the equations in this work.*



(a) normal distribution with $\mu = 0$ and $\sigma = 1$

(b) uniform distribution with $a = -1$ and $b = 2$

(c) Dirac distribution with $\mu = 1$

(d) histogram distribution with $\{\alpha_i\} = \{1, 2, 4, 2\}$ and $\{\beta_i\} = \{-3, -1, 0.5, 2, 3\}$

Figure 3.1: Four particular continuous probability distributions

Definition 3.2.2 *A* Boolean interval function *is a particular case of Boolean function where the property 3.5 holds.*

$$\forall x, y, z\ (f(x) = 1 \text{ and } f(z) = 1 \text{ and } x \leq y \leq z) \Rightarrow f(y) = 1 \tag{3.5}$$

We define the Boolean interval function $\mathbb{B}_{a,b}$ as:

$$\mathbb{B}_{a,b} : x \mapsto \begin{cases} 1 & \text{if } x \in [a, b] \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Property 3.2.3 *Every Boolean function can be expressed as a (possibly infinite) sum of Boolean interval functions.*

Suppose a Boolean function b. By convention, the Boolean function $-b$ is defined as $(-b) : x \mapsto b(-x)$.

3.3 Convolution

Definition 3.3.1 *A* convolution *is a mathematical operation on two functions, producing a third function. The convolution of f and g is written $f*g$ and is defined as follow:*

$$(f*g)(x) = \int_{-\infty}^{+\infty} f(x-t)g(t)dt \tag{3.7}$$

We present several properties of the convolution. These properties enable important speed ups for implementing the convolution.

- The convolution of two normal distributions is a normal distribution (Eq. 3.8).
$$\mathbb{N}_{\mu,\sigma^2} * \mathbb{N}_{\mu',\sigma'^2} = \mathbb{N}_{\mu+\mu',\sigma^2+\sigma'^2} \tag{3.8}$$
- The convolution of any function f by a Dirac distribution δ_μ (Eq. 3.9).
$$f*\delta_\mu = x \mapsto f(x-\mu) \tag{3.9}$$
- The convolution of a two Boolean interval functions is defined in Eq. 3.10. Fig 3.2 illustrates this equation.

$$\begin{aligned}(\mathbb{B}_{a,b} * \mathbb{B}_{c,d})(x) &= \begin{cases} x-a-c & \text{if } x \in [a+c, a+c+s] \\ s & \text{if } x \in [a+c+s, a+c+S] \\ a+c+S+s-x & \text{if } x \in [a+c+S, a+c+S+s] \\ 0 & \text{otherwise} \end{cases} \\ \text{with } s &= \min(b-a, d-c) \\ S &= \max(b-a, d-c) \end{aligned} \tag{3.10}$$

- Based on the convolution Boolean interval function and the property 3.2.3, the convolution of a two Boolean functions is defined in Eq. 3.11. The result is a piecewise continuous linear function that can be computed and stored efficiently.

$$\begin{aligned} f*g &= \sum_i \sum_j f_i * g_j \\ \text{with } \{f_i\} \text{ and } \{g_i\} & \text{ the decomposition of } f \text{ and } g \text{ into} \\ & \text{Boolean interval functions} \\ \text{i.e. } f(x) = \sum_i & f_i(x) \text{ and } g(x) = \sum_i g_i(x). \end{aligned} \tag{3.11}$$

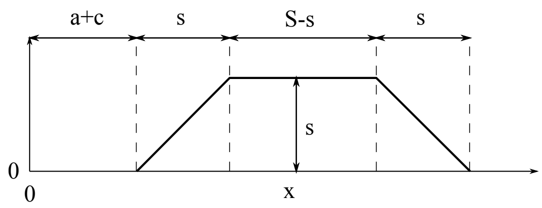


Figure 3.2: Convolution of two Boolean interval functions $\mathbb{B}_{a,b}$ and $\mathbb{B}_{c,d}$. Refer to Eq. 3.10 for notation.

- From Eq. 3.11, the convolution of a histogram distribution and a Boolean function is defined in Eq. 3.12. The result is also a piecewise linear function that can be computed and stored efficiently.

$$f * \mathbb{H} = \frac{1}{\sum_k \alpha_k} \sum_i \sum_j \frac{\alpha_j f_i * \mathbb{B}_{\beta_j,\beta_{j+1}}}{\beta_{j+1} - \beta_j} \tag{3.12}$$

with $\{f_i\}$ the decomposition of a Boolean function f and a histogram distribution $\mathbb{H}_{\{\alpha_i\},\{\beta_i\}}$

Convolution between Boolean functions is equivalent to Dechter's composition $\otimes$ operator (Dechter *et al.*, 1991). Convolution between a Boolean function and a probability distribution extends the Dechter's composition operator to probabilistic time distribution.

3.4 Temporal events and temporal states

Definition 3.4.1 *A* (temporal) event e *is a symbol (called type and noted* symbol_e*) and a time of occurrence* (time_e)*. The writing convention is* $e := \text{symbol}_e\,[\text{time}_e]$.

Example 3.4.2 *Suppose* $e := A\,[7.5]$ *to be an event. Literally* e *means that an event of symbol* A *occurs at time 7.5.*

Given a set of temporal events, we say that "an event of type A occurs at time t" if the set of events contains the event $A\,[t]$.

Definition 3.4.3 *A* (temporal) fuzzy event *is an uncertain temporal event and an imprecise temporal location. It is defined by a symbol (called type and noted* symbol_e*), a confidence* (conf_e) *and a temporal distribution* (dist_e)*. The writing convention is* $e := \text{symbol}_e\,[\text{conf}_e, \text{dist}_e]$.

A temporal event is a particular case of temporal fuzzy event with a confidence equal to one and a Dirac distribution as time distribution. Fuzzy events are particularly useful to represent imprecise and uncertain temporal predictions.

Example 3.4.4 *Suppose* $e := A\,[50\%, \mathbb{U}_{5,10}]$ *to be a fuzzy event. Literally* e *means that an event of symbol* A *occurs with 50% chance between times 5 and 10.*

Definition 3.4.5 *A* state s *is a Boolean function* $\mathbb{R} \rightarrow \{0, 1\}$ *that maps a value for every time location (i.e. real number). If* $s(t) = 1$*,* s *is said to be true at time* t*. Otherwise,* s *is said to be false at time* t*.*

Definition 3.4.6 *A* fuzzy state s *is a function* $\mathbb{R} \rightarrow [0, 1]$ *that maps a probability for every time location. If* s *is said to be true at time* t *with a probability of* $s(t)$*. Fuzzy state is an (potentially) uncertain state.*

3.5 Temporal Interval Tree Association Rules

3.5.1 Definition

A Temporal Interval Tree Association Rule (Tita rule) is a temporal pattern with the semantic of a rule i.e. a condition and a head. Several graphical examples of rules are given in fig. 3.3.

Like Chronicles (see definition in section 2.2), Tita rules can express the usual time point relation 'before/after', order and chaining, with different level of flexibility. Tita rules can also express Negation such as e.g. 'there are not occurrences of events of type A during a given interval' or 'there are no chains of 'A's followed by B followed by C'.

Definition 3.5.1 *A* type 1 condition c *is a symbol* (symbol_c) *and a set of type 2 conditions* (conds_c)*. The writing convention is* $c := \langle \text{symbol}_c, \text{conds}_c \rangle$*.*

Given a set of events E, a type 1 condition c is true at time t if:

- E contains an event e of symbol symbol_c and time t i.e. $\text{symbol}_e = \text{symbol}_c$ and $\text{time}_e = t$.
- All type 2 conditions $c' \in \text{conds}_c$ are true at time t (see definition below).

Definition 3.5.2 *A* type 2 condition c *is either:*

- *The negation of a type 2 condition c_2 (written $c := \text{not}\, c_2$). Here, c is true at time t if and only if c_2 is false at time t.*

- *A condition over a state s (written $c := s$). Here, c is true at time t if and only if s is true at time t i.e. $s(t) = 1$.*

- *An association between a Boolean function m and a type 1 condition c_3 (written $c := [m, c_3]$). Here, c is true at time t if and only if $\exists t'$ with $m(t' - t) = 1$ and c_3 is true at time t'.*

Definition 3.5.3 *A* Temporal Interval Tree Association Rule *(Tita) r is a symbol (symbol_r), a confidence (conf_r), a non null temporal distribution (dist_r) and a type 1 condition (cond_r). $\text{dist}_r(t - t')$ is the probability density of having an event of symbol symbol_r at time t while the condition cond_r being true at time t'. The writing convention is $r := \text{cond}_r \Rightarrow \text{head}_r \langle \text{conf}_r, \text{dist}_r \rangle$.*

When the condition cond_r of a Tita rule r is true at time t (also written as $\text{cond}_r(t)$), r is said to predict an event of symbol head_r with a probability of conf_r and with a temporal distribution of $t + \text{dist}_r$. An event e of symbol $\text{symbol}_e = \text{head}_r$ is said to verify such prediction if the density of the prediction is not equal to zero at time time_e i.e. $f'(\text{time}_e) > 0$ with $f' := t + \text{dist}_r$.

Definition 3.5.4 *Suppose a rule r, a set of events E of type head_r and $P \subset E$ to be the set of events predicted by the rule r. The* support *of the rule r (supp_r^E or supp_r) according to the set of events E is the ratio of events $e \in E$ predicted by r i.e. $\text{supp}_r = \frac{card(P)}{card(E)}$.*

By convention the *standard deviation* std_r of a rule r is the standard deviation of its temporal distribution dist_r. Rules with low standard deviation are generally more interesting than rules with high standard deviation because there are more "temporally accurate".

By convention the *prediction range* (written range_r) of a rule r is the range of its temporal distribution dist_r (see definition 3.1.2).

The standard deviation and the prediction range of a rule are two measures of the "temporal accuracy" of a rule. Depending on the application of the rules, one measure can be more interesting to use than the other. The *precision* of a rule r is defined as $\frac{1}{\text{range}_r}$.

Several particular cases of Tita rules are used in the thesis.

Definition 3.5.5 *A* unit Tita rule *is a Tita rule with the pattern $\langle \_, \emptyset \rangle \Rightarrow \_ \langle \_, \_ \rangle$ with $\_$ the unbound term i.e. a unit rule is a direct correlation between two events. Unit Tita rule expresses direct correlation such as an event A at time t implies an event B between $t + 5$ and $t + 10$.*

Definition 3.5.6 *A* trivial unit Tita rule *is a unit rule with the pattern* $\langle x, \emptyset \rangle \Rightarrow y \, \langle 100\%, \mathbb{U}_{-\infty,+\infty} \rangle$. *This type of rule is called trivial because as soon as there is at least one occurrence of* x *and one occurrence of* y*, this rule's confidence and support are 100%.*

Definition 3.5.7 *A* positive Tita rule *is a Tita rule with no negations. Positive Tita rules have strong properties useful for optimizing planning and predictions (see chapter 5).*

Definition 3.5.8 *A* linear Tita rule *is a Tita rule where type 1 conditions have at most one type 2 conditions. Linear Tita rules express chains of conditions.*

Definition 3.5.9 *A* forward Tita rule *is a Tita rule where the temporal distribution function is null for values below 0 i.e.* $\forall x < 0 \Rightarrow \mathrm{dist}_r(x) = 0$*, and all Boolean functions are null for values over 0. Forward Tita rules produces predictions in the future i.e. the conditions of a forward Tita rule are allays anterior to its head. The forward Tita rules are useful for on-line predictions.*

3.5.2 Examples

We are now presenting four examples of Tita rules in order to illustrate their power of expression. We specify a graphical representation to help the understanding of the rules' structure. Graphical representations of next example rules are given in fig. 3.3.

Example 3.5.10 *This example shows a unit rule. Suppose the rule* $r_1 := \langle A, \emptyset \rangle \Rightarrow B \, \langle 95\%, \mathbb{U}_{10,15} \rangle$. *Literally,* r_1 *expresses that if an event of type* A *occurs at time* t*, then, an event of type* B *will occur between* $t+10$ *and* $t+15$ *with* 95% *chance. This rule expresses a simple and direct correlation between two events.*

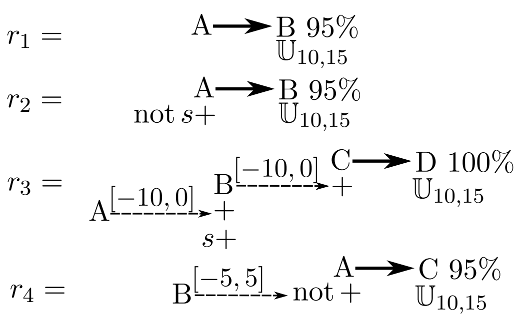


Figure 3.3: Graphical representation of Tita rules

Example 3.5.11 *This example shows a rule with a state negation. Suppose the rule* $r_2 := \langle A, \{\text{not}\, s\}\rangle \Rightarrow B\,\langle 95\%, \mathbb{U}_{10,15}\rangle$*. Literally,* r_2 *expresses that if an event of type* A *occurs at time* t *and the state* s *is not true at time* t*, then, an event of type* B *will occur between* $t+10$ *and* $t+15$ *with* 95% *chance.*

Example 3.5.12 *This example shows a rule with a chain of conditions. Suppose the rule* $r_3 := \langle C, \{[\mathbb{B}_{-10,0}, \langle B, \{s, [\mathbb{B}_{-10,0}, \{\langle A, \emptyset\rangle\}]\}\rangle]\}\rangle \Rightarrow D\,\langle 100\%, \mathbb{U}_{10,15}\rangle$ *. Literally,* r_3 *expresses that if an event of type* C *occurs at time* t_c *followed by an event of type* B *at time* t_b *(with a maximum interval of 10 seconds i.e.* $t_b - 10 \leq t_c \leq t_b - 0$*) followed by an event of type* A *at time* t_a *(with a maximum interval of 10 seconds i.e.* $t_a - 10 \leq t_b \leq t_a - 0$*) and* s *is true at time* t_b*, then, an event of type* D *will occur between* t_a+10 *and* t_a+15 *with* 100% *chance.*

Example 3.5.13 *This example shows a rule with a negation of the occurrence of an event. Suppose the rule* $r_4 := \langle A, \{\text{not}\,[\mathbb{B}_{-5,5}, \langle B, \emptyset\rangle]\}\rangle \Rightarrow C\,\langle 95\%, \mathbb{U}_{10,15}\rangle$*. Literally,* r_4 *expresses that if an event of type* A *occurs at time* t *and no events of type* B *occur between* $t-5$ *and* $t+5$ *i.e. there are no events of type* B *around the event of type* A*, then, an event of type* C *will occur between* $t+10$ *and* $t+15$ *with* 95% *chance.*

3.6 Information gain

The *entropy* is a measure of disorder or unpredictability. In the information theory, the entropy measures the uncertainty of a random variable. The *entropy* H_X of a Boolean random variable X is $H_X = h(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ with p the probability of X to be true. Such entropy function is called the *binary entropy function*. Fig. 3.4 displays this function.

In the information theory, the *information gain* of a piece of information is a measure of entropy change while taking into account, or not taking into account this piece of information.

Suppose a Tita rule r. Following this convention, the entropy of r is $h(\text{conf}_r)$. Suppose the rule r to be enriched (addition of condition) to become a new rule r'. Suppose the followings variables:

- a is the number of (correct or incorrect) predictions of r also predicted by r'.
- as is the number of correct predictions of r also predicted by r'.
- $\neg sa$ is the number of incorrect predictions of r also predicted by r'.
- $\neg as$ is the number of correct predictions of r not predicted by r'.

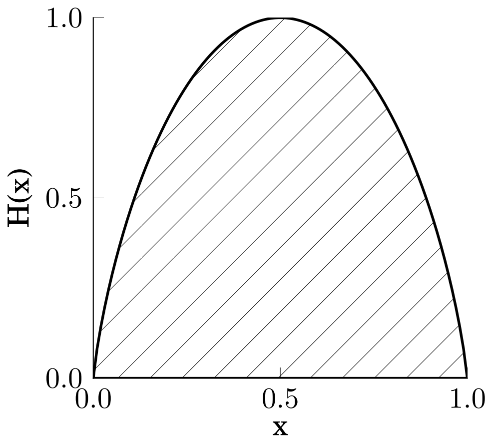


Figure 3.4: The Binary entropy function.

- $\neg s \neg a$ is the number of incorrect predictions of r not predicted by r'.

The eq. 3.13 shows the information gain of the enrichment from r to r'.

$$\begin{aligned} HG(r \rightsquigarrow r') = h(\mathrm{conf}_r) - \frac{\mathrm{supp}_{r'}}{\mathrm{supp}_r}(-\frac{as}{a}\log\frac{as}{s} - \frac{\neg sa}{a}\log\frac{\neg sa}{a}) \\ -(1 - \frac{\mathrm{supp}_{r'}}{\mathrm{supp}_r})(-\frac{\neg as}{\neg a}\log\frac{\neg as}{\neg s} - \frac{\neg s \neg a}{\neg a}\log\frac{\neg s \neg a}{\neg a}) \end{aligned} \tag{3.13}$$

3.7 Graph coloring

Definition 3.7.1 *A* graph *is a mathematical object defined as a set of vertices V (or points) and a set of edges E (directed or undirected links between two points). A* cycle *is closed path of edges. An undirected graph is* connected *if and only if any two vertices are connected by a path (in the case of a directed graph, this path should follow edges' directions). An* undirected tree *is an undirected connected graph without cycles. A* directed tree *is a directed acyclic graph in which there exists a single source vertex (called trunk) that has a unique path to every other vertex. A* leaf *is a vertex v without children i.e. without any edges from v to any other vertex. A* forest *is a disjoint union of trees.*

The graph (vertex) coloring problem consists to assign a color to every vertex of an undirected graph such that no connect vertices share the same color. The goal is to use the smallest number of different color.

More formally, given an undirected graph (V, E) where V is the set of vertices and E the set of edges. The graph (vertex) coloring problem consists to find the mapping $c : V \to \{1, \dots, n\}$ such that any two connected vertices v and v' (two vertices connected any an edge) do not share the same color ($c(v) \neq c(v')$ with n as small as possible. The smallest n is called the *chromatic number*. Fig. 3.5 shows an example of graph coloring.

The problem is NP-complete. Several polynomial heuristic exits to approximate this problem. I will use the DSAT heuristic (Brélaz, 1979) in this thesis.

3.8 Tita plan

In the automatic planning and scheduling domain, a *plan* is generally understood as a set of actions leading to a goal. As an example, 'sleeping' is a simple plan for the goal of 'not to be tired'. A plan has to be applied on a *world* (or a system). A world is a generic term referring to an initial state and a description of behavior. From the point of view of a person considered as a system, the rule 'if it is night and I sleep, then I will not be tired' is valid. The *plan* of 'sleeping' is a solution to the *goal* of 'not to be tired' based on the *rule* 'if it is night and I sleep, then I will not be tired'.

A *Tita plan* is a plan based on the Tita rules' semantic. In the chapter 5, a Tita plan is generated from a Tita rule and a set of observations (set of events). We define the notion of Tita plan and give some examples.

General planning considers two types of events: The events that can be triggered directly (and observed), and the events that can only be observed. Base on the same concept, we define the *doable symbols* to be the set of types of events that can be triggered and observed, and the *exogenous symbol* to be the set of types of events that can only be observed. As an additional constraint, every doable symbol is associated to a Boolean function describing when it is doable. In the case of on-line planning, this constraint is a way to specify that it is not possible to trigger actions in the past.

Definition 3.8.1 *A* Tita plan *is a directed forest (disjoint union of tree*

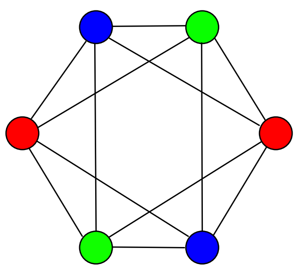


Figure 3.5: Exemple of graph coloring. Minimum number of color such that two connected vertices have different colors.

graphs) expressing a set of constraints over temporal events. Edges are oriented from the head to the leaves. Every vertex v is labelled with a tuple containing a symbol (symbol_v*), a Boolean function (*Acst_v*) and a set of state symbols (*states*). The Boolean function of a vertex is called the* absolute constraint*. Every edge e is labelled with a Boolean function noted* Rcst_e*, and a label 'positive' or 'negative'. The Boolean function of an edge is called the relative constraint.*

The plan semantic is given as follow. A plan is said valid if all its vertices are valid. A vertex v is said valid at time t if:

- All the states of the set of states states_v are true at time t.
- If the vertex symbol is doable, then the vertex symbol is doable at time t
- If the vertex symbol is exogenous, then an event of symbol equal to the vertex symbol occurs at time t
- The absolute constraint of the vertex is true a time t i.e. $\text{Acst}_v(t) = 1$.
- For all edges $e_{v \to v'}$ from v to v':
  If e is positive, there is at least one t' with v' validated at time t' and $\text{Rcst}_e(t' - t) = 1$

  If e is negative, there is no t' with v' validated at time t' and $\text{Rcst}_e(t' - t) = 1$.

A *positive Tita plan* is a Tita plan without negative edges. A plan can achieve its objective at any time t such that the tree's trunk is valid at time t. The Fig. 3.6 shows an example of Tita plan.

Definition 3.8.2 *Given a valid Tita plan, an* instance of Tita plan *is a set of (temporal) events that makes all doable vertices (vertices with doable symbol) of the Tita plan valid. An instance of Tita plan can be viewed as a plan without degree of freedom.*

3.9 Meta Tita plan

The core of planning techniques found in the literature, such as the *event calculus*, is to combine pieces of knowledge to build a plan. For the large majority of these techniques, the knowledge is expressed as association rules.

The following example illustrates the use of two association rules to build a plan:

Suppose the two following rules:

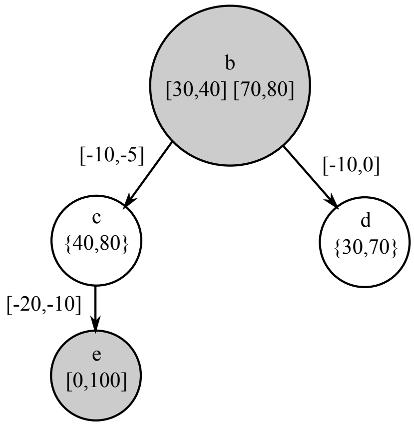


Figure 3.6: Example of Tita plan. The grey vertices are vertices with doable symbols. The symbols e and b are doable in the interval $[0, 100]$. c and d are exogenous. c is observed at times 40 and 80. d is observed at times 30 and 70. An instance of this plan is valid if it contains an event of type b during $[30, 40]$ or $[70, 80]$, if it contains an event of type e during $[0, 100]$, and if the relative contains are verified. $\{b[32], e[10]\}$ is a valid instance of this Tita plan.

rule 1 : If it is night and I sleep, then I will not be tired.

rule 2 : If I am not tired, then I will pass my exams.

Suppose that my goal is to 'pass my exams'. From rule 2, I know that I need to 'not to be tired'. But, since I cannot just choose to 'not to be tired', I need to find a way to 'not to be tired'. From the rule 1, I know that if I sleep at night, I will 'not to be tired'. To sum-up, the two rules are combined to find a plan for the goal of 'pass the exams': The plan is to 'sleep at night'. This causes 'not to be tired' which will cause 'pass the exams'.

Different rules can have very different temporal precisions. It is not always possible to combine several temporal rules into a simple list of actions to perform (a Titar plan), and expect to have a high confidence plan, even if the rules that describe the world have high confidence. The example 3.9.1 illustrates this problem.

Example 3.9.1 *Suppose two Tita rules:*

$$r_1 := \langle A, \emptyset \rangle \Rightarrow B \, \langle 100\%, \mathbb{U}_{0,100} \rangle$$
$$r_2 := \langle C, \{[B_{-10,-5}, \emptyset]\} \rangle \Rightarrow D \, \langle 100\%, \mathbb{U}_{10,15} \rangle$$

Suppose the objective to be the symbol D. Suppose A and C being the only doable symbols. A plan would be to 'fire' A. (that will cause B) and finally *fire C. But r_1 generates B with low (temporal) precision (range of 100), and r_2 need a maximum precision range of 5 to reach a maximum confidence. Therefore, even with two rules having 100% confidence, this plan has a* $5\%(=\frac{5}{100})$ *confidence to success. The fig. 3.7 is a graphical representation of this example.*

Therefore, we do not combine several Tita rules to make a single Tita plan (which is possible), but we combine several Tita plans to make a Meta Tita plan (i.e. a plan of plans). With the same rules as the previous example, a Meta-plan is: Fire A, wait for B to occur, and then fire C. This Meta-plan has 100% confidence. Meta Tita plans can express the notion of 'intermediate objectives to wait for' and the disjunction of solution for an objective (or intermediate objectives).

Definition 3.9.2 *A* Meta Tita plan *is a directed tree graph expressing a set of constraints over Tita plans. The edge are allays oriented from the head to the leaves. There are two types of vertices: 'solution vertices' and 'problem vertices'. A* solution vertex *is only connected to* problem vertices. *A* problem vertex *is only connected to* solution vertices. *The trunk of the graph is a problem vertex.*

Every solution vertex v is labelled with a Tita plan (written subPlan_v*). Every problem vertex v' is labelled with a leaf of the plan of v (written* $\text{subLeaf}'_v$*) such that there is an edge e from v to v', with a Boolean function called absolute constraint and a symbol. The leaf of a plan connected to a problem vertex of a Meta-plan is called a 'caused' vertex.*

Example 3.9.3 *Suppose a 'caused' vertex v of a Tita plan p associated to a problem vertex v' of a Meta Tita plan rp. The 'caused' vertex v is a sub-objective of the plan p which is solved by the plans associated to children of v'. The fig 3.8 represents this example.*

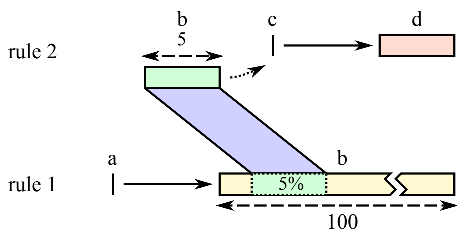


Figure 3.7: Example showing that the combination of high confidence rules doesn't always give high confidence plan. The two rules are the ones presented in example 3.9.1. Since the rule r_1 has low temporal precision, the time of execution of C cannot be a priori estimated. The maximum confidence of this plan is $5\%(= \frac{5}{100})$.

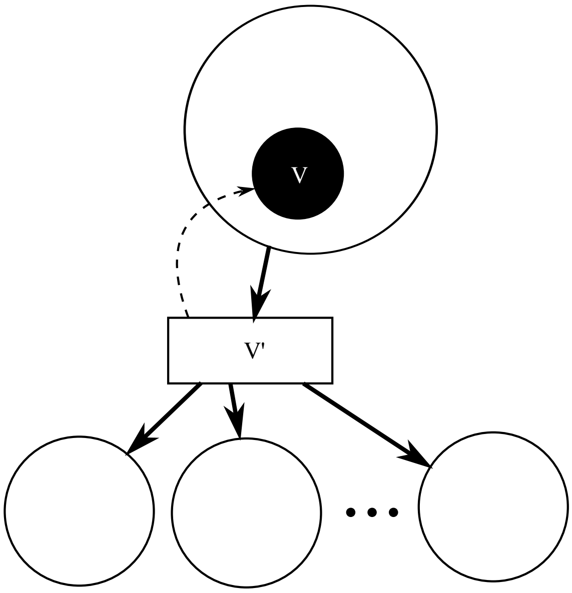


Figure 3.8: Representation of the example 3.9.3

The Fig. 3.9 shows an example of Meta Tita plan.

We define several types of particulars Meta Tita plans.

Definition 3.9.4 *A* Positive Meta Tita plans *is a Meta Tita plans that contains only positive Tita plans.*

Definition 3.9.5 *A* Weak linear Meta Tita plans *is a Meta Tita plans such that every plan contains at most one caused vertex. Weak linear Meta Tita plans are a sub class of Meta Tita plans with nice properties for computational issues.*

Definition 3.9.6 *A* Strong linear Meta Tita plans *is a Meta Tita plans with one leaf i.e. it is a path graph. A* Strong linear Meta Tita plans *is also a* weak linear Meta-plan*. Every weak linear Meta Tita plan can be decomposed (or unfolded) into a finite set of strong linear Meta Tita plan (see chapter 5 for more details).*

3.10 Conclusion

This chapter presents some common notations, the mathematical objects used in the thesis and the temporal patterns we have developed to represent temporal relations and temporal plans. The next chapter introduces a data mining algorithm able to extract such temporal pattern from temporal datasets.

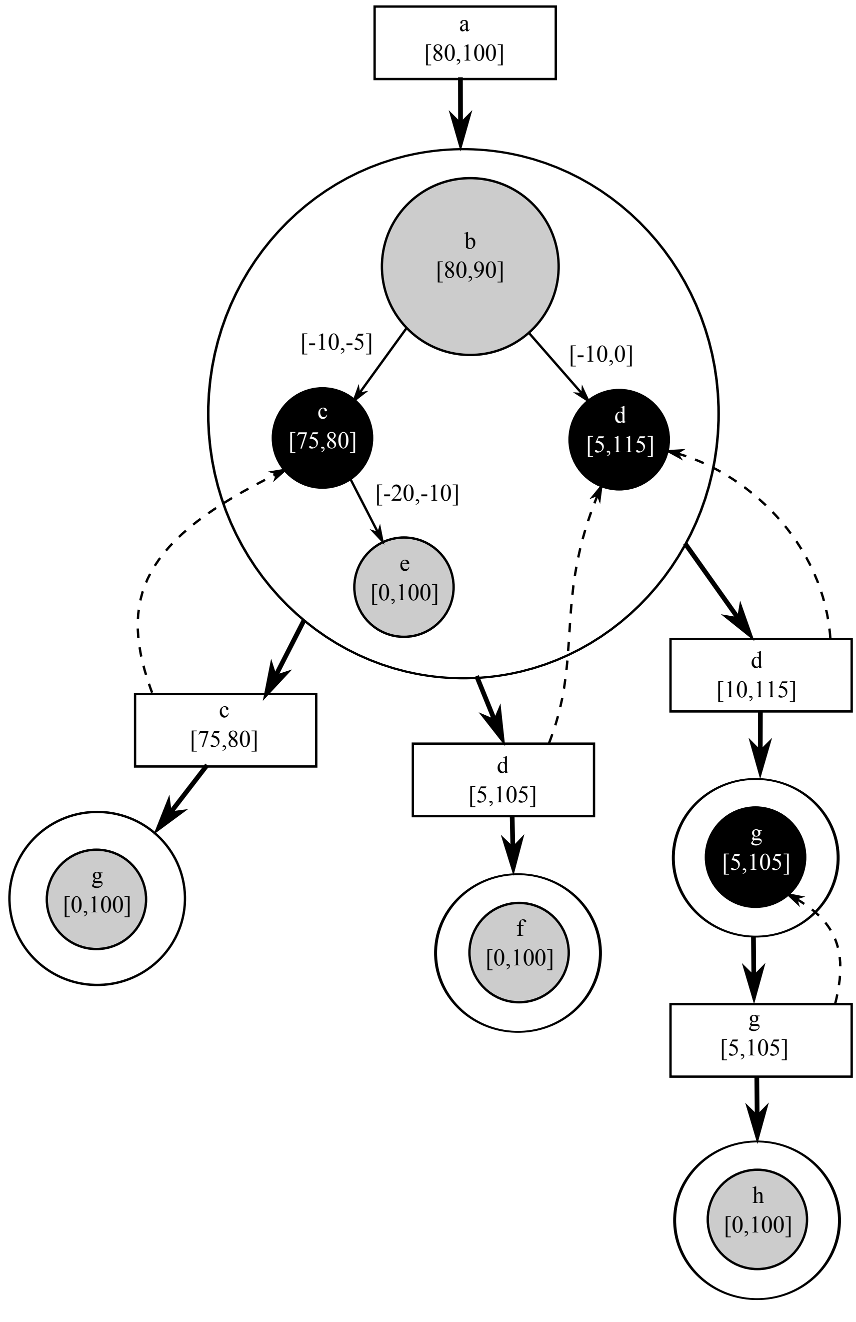


Figure 3.9: Example of Meta Tita plan. The main objective is A. The grey vertices are vertices with doable symbols. The black vertices are 'caused' vertices. Dashed arrows represent the labels of the edges. This Meta-plan is not (weak or strong) linear.

4 Temporal learning

Chapter Contents

This chapter presents the algorithm for learning Tita rules (see definition in section 3.5). This algorithm named ‘Titarl’ (for Temporal Interval Tree Association Rules Learner) is partially inspired by the apriori algorithm (see definition in section 2.3.1) and the ID3 algorithm (Quinlan, 1986).

As discussed in the related work chapter, most of the techniques that learn temporal rules begin by the learning temporal constraint systems (TCS), and then, they ‘convert’ these constraint systems into temporal rules. This is generally done by labeling a part of a constraint system to be the body of the rule, and labeling the remaining part to be the head of the rule. The example 2.3.1 shows that a good TCS is not guaranteed to give a good rule.

Because of this fact, we design our algorithm to directly learn temporal rules without going through a generation of constraint systems.

The core of a constraint system (or of a temporal rule) is the temporal constraints between the events. As discussed in the related work chapter, for the large majority of temporal patterns, a condition can be expressed as a subset of $\mathbb{R}$ (see section 2.2). This class of constraint systems is called temporal constraint network. We call the learning of such conditions the 'temporal interval selection problem' (see section 4.5.2).

Different algorithms allow different degrees of freedom on the conditions and different solutions for this problem (learning of the temporal interval). Most of the techniques found in the literature ask for the user to specify this temporal constraint (as an input parameter). In addition, this constraint often needs to be a convex subset of $\mathbb{R}$ that contains 0.

We believe (and this is shown by experimentation) that a good algorithm that learns constraint systems cannot use a fixed temporal constraint for two reasons:

First, because giving the constraint intervals as a user input implies that all the temporal constraint are the same. Therefore, fixing the temporal constraint is equivalent to cutting the vast majority of the research space (all the temporal intervals of a constraint system should be the same). Except for specific datasets, there is no reason to believe that the hidden relations of the dataset validate this constrain.

Secondly, because a family of rules whose only differences are the temporal constraints can contains several 'good' elements, and it is not possible to chose 'the best' temporal constraint or 'the best' rule. Therefore, there is no reason to fix the temporal constraint to be unique. In fact, we often observe such families of good rules with a "trade off" of confidence, support and temporal precision between The figure 4.1 presents a situation where there are several possible good intervals for such a family of rule.

From these ideas, we choose to develop a technique that does not require for the temporal constraints to be fixed by the user. Considering this point, the main problem is that when the temporal constraints are not fixed, it is not possible to enumerate all the good intervals (and all the good rules) as it is done in some data mining algorithms. In order to be tractable, we had to choose criteria to limit the number of enumerated temporal constraints. And because of the substantial difference between the temporal constraint of the head of a rule and the temporal constraints of the body of a rule, we must use two different criteria.

The underling idea of the algorithm is the following: Titarl begins by computing the set of trivial unit rules. These rules are *decorrelated*, *refined* and *stored* in R. Next, until a stopping criterion is met (maximum number of rules, maximum duration of learning, etc.) the algorithm picks a rule r in

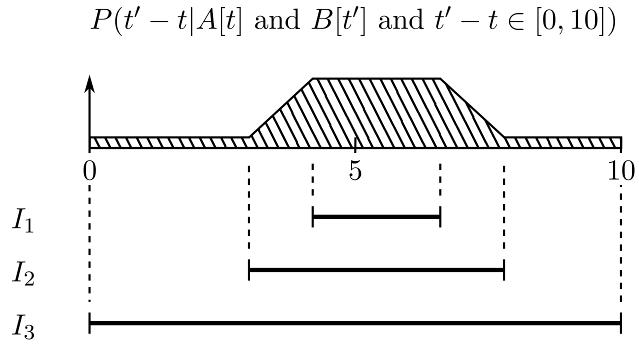


Figure 4.1: We suppose a dataset where the distribution of the probability $P(t' - t|A[t]$ and $B[t']$ and $t' - t \in [0, 10])$ is given. The problem is to find a good interval for a TCS between events A and B in the rule $A \rightarrow B$. I_1, I_2 and I_3 are three possible good intervals with a "trade off" between confidence, support and temporal precision: The support and the confidence from the interval I_3 is the highest. The range from the interval I_1 is the better. The ratio between confidence and range or support and range is the highest for the interval I_2.

the set R, it adds a condition to r, decorrelates it, refines it, and it adds the result back to the set R (the result can be several rules). The figure 4.2 shows the global architecture of the Titarl algorithm.

The creation of *trivial unit rule* is a simple step described in the next section. The improvement of a rule is based on three different operations (Addition of condition, division (or decorrelation) and refinement). These three operations are also described in the next section.

Given a rule, the number of different parameters for the improvements (Addition of condition, division and refinement) is generally infinite. Therefore, for each of these three operations, we associate an *improvement policy*. These policies are described and justified in the section 4.2.

The section 4.3 gives the listing of the algorithm. The section 4.4 presents a step by step detailed run of the algorithm. The section 4.5 presents a comparison between our algorithm and other related algorithms. This last section discusses the different approaches to the *temporal interval section problem*.

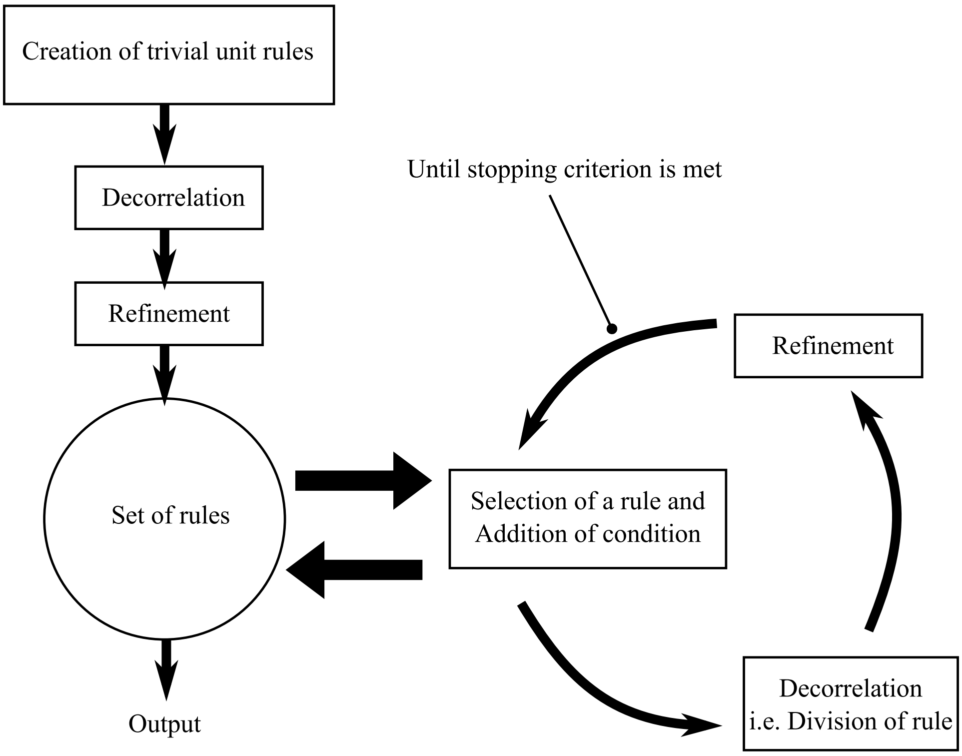


Figure 4.2: Schema of the Titarl algorithm.

4.1 The four operations

4.1.1 Creation of trivial unit rules

Given two symbols A and B, a trivial unit rules is $\langle A, \emptyset \rangle \Rightarrow B \langle 100\%, \mathbb{U}_{-\infty,+\infty} \rangle$. The *confidence* and the *support* of a trivial unit rule is 100%. The *(temporal) precision* of a trivial unit rule is $0 = \frac{1}{\infty}$. Given n symbols, n^2 trivial unit rules can be generated.

4.1.2 Addition of a condition to a rule

Given a rule r, c_1 a type 1 condition of r and c_2 a type 2 condition. The addition of a condition c_2 to a rule r at the location c_1 gives a rule r' such that r' is the rule r with the condition c_1 replaced by the condition $\left\langle \text{symbol}_{c_1}, \text{conds}_{c_1} \cup \{c_2\} \right\rangle$. Notice that $\left\langle \text{symbol}_{c_1}, \text{conds}_{c_1} \cup \{c_2\} \right\rangle$ is a type 1 condition.

Property 4.1.1 *All events predicted by the rule r' are also predicted by r. Therefore,* $\text{supp}_{r'} \leq \text{supp}_r$.

Fig. 4.3 shows an example of addition of condition.

4.1.3 Division of a rule

The division of a rule consists in the splitting a rule into a set of rules.

Definition 4.1.2 *A* division function d *is a function* $d : \mathbb{R} \to \mathbb{N}$ *with a maximum i.e.* $\exists n\ \forall x\ d(x) \leq n$.

Given a rule $r := \mathrm{cond}_r \Rightarrow \mathrm{head}_r \langle \mathrm{conf}_r, \mathrm{dist}_r \rangle$ with a probability distribution dist_r. The division of r according to the division function d produces a set of rules $\{r_i\}$ with:

$$\mathrm{cond}_{r_i} = \mathrm{cond}_r \tag{4.1}$$

$$\mathrm{head}_{r_i} = \mathrm{head}_r \tag{4.2}$$

$$\mathrm{dist}_{r_i}(x) = \begin{cases} \mathrm{dist}_r(x) & \text{if } d(x) = i \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

Property 4.1.3 *Suppose a rule* r *divided into the rules* $\{r_1, \ldots, r_n\}$. *The followings equations hold:*

$$\forall i\ \ \mathrm{supp}_{r_i} \leq \mathrm{supp}_r \tag{4.4}$$

$$\forall i\ \ \mathrm{conf}_{r_i} \leq \mathrm{conf}_r \tag{4.5}$$

$$\forall i\ \ \mathrm{std}_{r_i} \leq \mathrm{std}_r \tag{4.6}$$

$$\sum_i \mathrm{conf}_{r_i} \geq \mathrm{conf}_r \tag{4.7}$$

Fig. 4.4 shows an example of division of rule.

4.1.4 Refinement to a rule

We suppose two types of rule refinement: The refinement of a rule's head and the refinement of a rule's body. The refinement of a rule's head consists in changing the temporal distribution of the head of a rule. The refinement of a rule's body consists in changing the temporal condition of a rule.

The figure 4.5 shows two examples of rule refinements.

$$r_5 = \quad \mathrm{A} \longrightarrow \underset{\mathbb{U}_{10,15}}{\mathrm{B}\ 95\%}$$

$$r_6 = \quad \mathrm{C} \xrightarrow{[-10,0]} \overset{\mathrm{A}}{+} \longrightarrow \underset{\mathbb{U}_{10,15}}{\mathrm{B}\ 95\%}$$

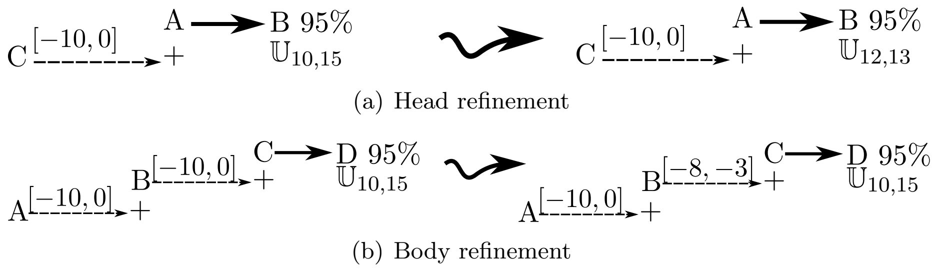
Figure 4.3: Addition of the condition $[T_{-10,0}, \langle C, \emptyset \rangle]$ to the rule $r_5 := \langle A, \emptyset \rangle \Rightarrow B \langle 95\%, \mathbb{U}_{10,15} \rangle$. The result is $r_6 := \langle A, \{[T_{-10,0}, \langle C, \emptyset \rangle]\} \rangle \Rightarrow B \langle 95\%, \mathbb{U}_{10,15} \rangle$.

$$r_7 = \quad \mathrm{A} \xrightarrow[\mathbb{U}_{5,15}]{} \mathrm{B}$$

$$r_8 = \quad \mathrm{A} \xrightarrow[\mathbb{U}_{5,10}]{} \mathrm{B}$$

$$r_9 = \quad \mathrm{A} \xrightarrow[\mathbb{U}_{10,15}]{} \mathrm{B}$$

Figure 4.4: division of r_7 into $\{r_8, r_9\}$ with the division function



(a) Head refinement

(b) Body refinement

Figure 4.5: Two examples of rule refinements

4.2 Improvement policies

This subsection presents the three improvement policies used in the algorithm. Each of these policies takes as input a rule (or a set of rules), and gives in return a parameter for the corresponding operation. For example, given a rule r, the policy for the division of rule returns the 'best' *division function* for r.

4.2.1 Policy for the Addition of condition

Given a set of rules R, this policy selects (a) a rule $r \in R$, (b) a location in r and (c) a condition to add to r in order to improve it (see definition of adding of condition in section 4.1.2). The value of improvement is given by the *information gain* of the addition of a condition (see definition of information gain in section 3.6). A given percentage of the time (fixed at 90% in the experiments), the process selects the rule and the condition to add in order to maximize the information gain. The other ten percent of the time, the process selects a random rule and a random condition. In this last case, the probability of a rule and a condition to be chosen is non-uniform and depends on the number of conditions of the rule. This policy helps preventing the algorithm from falling into a local minimum. The selection of the highest information gain is inspired from the ID3 algorithm used to generate decision trees (Quinlan, 1986). In this context, the information gain of the adding of a condition to a rule measures how much confidence is gained and how low the support becomes.

In the case of *a condition over a state*, the algorithm find of the best condition to add i.e. the condition with the higher information gain, with a simple enumeration of all the possible conditions.

In the case of condition over the occurrence of temporal events, the algorithm need to select the temporal constraint (defined as a subset of $\mathbb{R}$ i.e. a Boolean function) associated with the condition. However, since the number of possible temporal constraint is infinite, the algorithm cannot enumerate and test of all of them.

Our solution to this problem is to perform a "kind of gradient descent" to compute the Boolean function m of the condition:

1. We compute the family of Boolean function $\{b_i\}_{1 \leq i \leq n}$ based on the *histogramBounds* parameter (*histogramBounds* is a list of $n+1$ ordered real numbers). Fig. 4.10 shows several examples of histogram bounds. The following equation defines $\{b_i\}$.

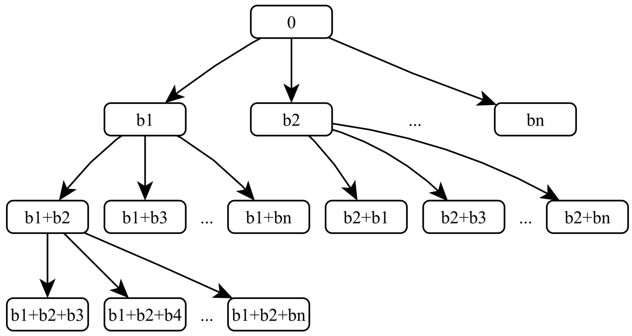
$$b_i(t) = \begin{cases} 1 & \text{if } t \in [\text{histogramBounds}_i, \text{histogramBounds}_{i+1}] \\ 0 & \text{otherwise} \end{cases}$$

2. The Boolean function m is defined as the sum of a subset of elements of $\{b_i\}$ i.e. $m = \sum_{i \text{ if } x_i=1} b_i$ where $X = (x_1, x_2, \ldots, x_n)$ is a Boolean vector. The number of candidates for m (or for X) is $2^n - 1$. Therefore, all the candidates cannot be explored (In our experiments, n is generally defined between 10 to 50). The selection of the best candidate is done with a gradient descent on the vector X. The gradient descent tries to maximize the information gain.

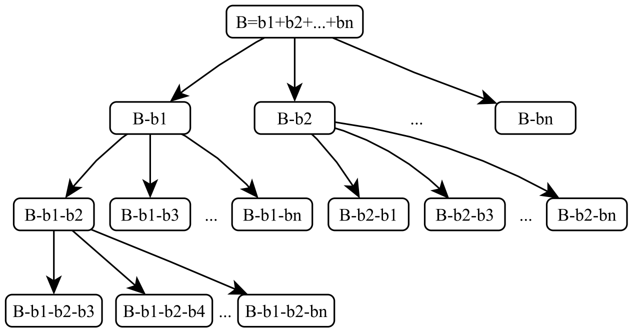
   In the case of a positive condition, the gradient descent iterates on the addition of elements of $\{b_i\}$ to m. m is considered empty are the beginning i.e. $X = (0, 0, \ldots, 0)$ at the beginning.

   In the case of a negative condition, the gradient descent iterates on the suppression of elements of $\{b_i\}$ to m. m is considered to be $m = \sum_i b_i$ at the beginning i.e. $X = (1, 1, \ldots, 1)$ at the beginning. The fig. 4.6 illustrates the gradient descent in the case of positive and negative conditions.

   Experiment shows that in small datasets (datasets with a small number of events), the restriction for m to be a *Boolean interval function* is a good solution to limit over training of rules. With this restriction, the number of candidates for m is $\frac{n(n-1)}{2}$. In this case, the selection of the best candidate (the candidate with the highest information gain) can be done with an enumeration of all the candidates (instead of a Gradient descent on entropy).



(a) Tree of all possible iterations for the construction of a Positive condition. During the construction of a condition, only one path is explored.



(b) Tree of all possible iterations for the construction of a Negative condition

Figure 4.6: Illustration of the gradient descent of the policy for the addition of positive and negative conditions

4.2.2 Policy for the Division of rules

We begin the presentation of this policy with an example.

Example 4.2.1 *Suppose a rule r that produces predictions which always match two events or none. Suppose, we divide this rule (see definition of rule division in section 4.1.3) into two rules r_1 and r_2, and that r_1 and r_2 produce predictions which always match one event or none. Therefore,* $\text{conf}_{r_1} = \text{conf}_{r_2} = \text{conf}_r$. *Moreover* $\text{std}_{r_1} \leq \text{std}_r$ *and* $\text{std}_{r_2} \leq \text{std}_r$ *(property 4.1.3). In conclusion, r_1 and r_2 are more precise than r but their confidence is similar.*

The goal of this policy is to produce more temporally precise rules (decrease rule standard deviation) while loosing as less as possible confidence and support (like the division of r into r_1 and r_2).

Several possible solutions have been studied to chose the 'best divisions', including the analysis of the shapes of histograms (detection of maximums, detection of inflexions points) and the maximization of the likelihood on models mixtures (Gaussian, exponential, uniform, log-normal, etc.). From the experiments, we observe that the best solution is the following one:

Given a rule r, the policy selects a 'good' *division function* to divide r into a set of rules $\{r_i\}$ such that the rules $\{r_i\}$ are independent (see definition of rule division in section 4.1.3). This dependency division is based on graph coloration techniques.

Suppose a rule r, and cond_r the conditions of r. By convention $\mathrm{cond}_r(t)$ is a Boolean predicate which is true if and only if cond_r is true at time t. The policy relies on the analysis of the probability distribution of the rule $\mathrm{dist}_r = P(t' - t'' | \,\mathrm{head}_r[t] \text{ and } \mathrm{cond}_r(t'))$.

1. The first step is to compute H, an N categories histogram of dist_r. The bounds of this histogram are defined by the user parameter **histogramBounds** (H is not necessary uniform). Each category i of the histogram corresponds to the interval I_i.

2. The second step is to compute an N by N co-occurrence matrix M such as: Given a prediction $t' + \mathrm{dist}_r$ of the rule r ($\mathrm{cond}_r(t')$ is true) and an event $\mathrm{head}_r[t_1]$ matching this prediction ($(t' + \mathrm{dist}_r)(t_1) > 0$) with $t_1 - t' \in I_i$, $M_{i,j}$ is the probability of having an event $\mathrm{head}_r[t_2]$ also matching this prediction with $t_2 - t' \in I_j$.

$$\begin{aligned} M_{i,j} = P(\,&\mathrm{head}_r[t_2] \wedge (t' + \mathrm{dist}_r)(t_2) > 0 \wedge (t_2 - t') \in I_j \\ &|\,\mathrm{cond}_r(t') \wedge \mathrm{head}_r[t_1] \wedge (t' + \mathrm{dist}_r)(t_1) > 0 \\ &\wedge (t_1 - t') \in I_i) \end{aligned}$$

   $M_{i,j}$ is computed by considering each predictions of the rule r, and each events which are matched by those predictions. In the case of large datasets, $M_{i,j}$ can be estimated by only considering a subset of the predictions of r.

3. Next, we apply a set of thresholds $\{\sigma_i\}$ on the covariance matrix $M_{i,j}$. The next operations will be applied on each of threshold matrices.

4. For each matrix $M_{i,j}$, we compute the graph $G := (\{v_i\}, \{e_i\})$ using $M_{i,j}$ as an *adjacency matrix*. $\{v_i\}$ are the vertices of the graph G. $\{e_i\}$ are the edges of the graph G. An adjacency matrix is defined a follow: If $M_{i,j} = 1$, then there is an edge between the vertices v_i and v_j. Otherwise, there is no edge between the vertices v_i and v_j.

5. We compute the vertices coloring $c : V \to \mathbb{N}$ of G i.e. labeling of the graph's vertices with colors such that no two vertices sharing the same edge have the same color (see the complete definition of graph coloring in section 3.7).

   In this context, two vertices of different colors represent two intervals that should be separated in the probability distribution of the rule r.

6. Finally, the division function d is defined as follow:

$$d(x) = c(i) \text{ with } x \in I_i \tag{4.8}$$

   The division function defines the temporal distribution of the newly generated rules. We merge all the division functions and remove the duplicates. The Thresholds are fined between 0 and 1. In our experiments, the thresholds σ_i are set to be $\sigma_i = \alpha \exp^{\frac{x-p}{2}}$ for $i \in [0, p[$ with $p = 10$ and $\alpha = 0.2$. Decreasing the number of thresholds increases the speed of the algorithm, but in the case of noisy dataset, the algorithm will produce rules with less accurate temporal constraints (rule confidence, support and temporal accuracy will decrease). If the dataset is noisy, increasing the number of thresholds increases the number of generated rules and reduces the speed of the algorithm. If the dataset is not noisy, increasing the number of thresholds does not impact the number of generated rules nor reduce the speed of the algorithm.

   In the worst case (dataset with a very specific type of noise), the number of generated rules is bound by $\min(\frac{N(N+1)}{2}, \frac{N(N+1)-(N-p)((N-p)+1}{2})$ with N the number of categories of the histogram. In the Home Dataset experiment 6.1.3, the average number of created rule during this step is 1 and in the computer simulated dataset experiment 6.1.2, it is about 10 in the part with the highest level of noise.

Example 4.2.2 *The following example details step by step the application of policy for the division of rules. Suppose two event symbols with the following property: If there is an event A at time t, then there are an event B at time $t+5$ and an event B at time $t+15$. Additionally, other events B are uniformly present in the dataset. Suppose the rule $r_1 := \langle A, \emptyset \rangle \Rightarrow B \langle 100\%, \mathbb{U}_{0,20} \rangle$. Fig. 4.7 represents the steps of the process.*

(a) The input rule r_1.

(b) The histogram of the distribution $P(t' - t'' | A[t'']$ and $B[t'])$.

(c) The co-occurrence matrix $M_{i,j}$.

(d) The result of applying the threshold on $M_{i,j}$.

(e) The graph with the vertices coloration.

(f) The colored distribution.

(g) And finally the output rules r_2 and r_3.

The division function is $d : x \to \begin{cases} 1 & \text{if } x \in [0, 10] \\ 0 & \text{if } x \in [10, 20] \end{cases}$.

The result of the division on rule r_1 is

$$r_2 := \langle A, \emptyset \rangle \Rightarrow B \langle 100\%, \mathbb{U}_{0,10} \rangle$$
$$r_3 := \langle A, \emptyset \rangle \Rightarrow B \langle 100\%, \mathbb{U}_{10,20} \rangle$$

4.2.3 Policy for the Refinement of a rule

The refinement of a rule r is the modification of the temporal intervals of the conditions of r or the modification of the temporal distribution of r. The refinement has two objectives: First, in the case of the modification of the temporal distribution, it can increase a rule confidence and precision without decreasing significantly the support. Second, in the case of the modification of the condition, it can improve conditions, and therefore, increase a rule confidence and support. The example 4.2.3 illustrates this second case.

Example 4.2.3 *Suppose a dataset that contains the rule r_1 represented in fig. 4.8.*

In this figure, I_1 and I_2 are two temporal intervals. When applied to the dataset, Titarl begins to learn r_2. However, because of the noise in the dataset, I'_1 is slightly different from I_1. Next, Titarl learns r_3. Again, because of the noise in the dataset, and because of the inaccuracy of I'_1, I'_2 is different from I_2.

When applying to the rule r_3, the refinement recomputes I'_1 to I''_1 and produce r_4. Since I'_1 is initially estimated for r_2 and I''_1 is estimated for r_3, then I''_1 is possibly more accurate than I'_1. As result, r_4 is more accurate than r_3.

And again, I'_2 of r_4 can be refined to produce r_5 possibly more precise than r_4. The refinement process can continue indefinitely.Experiments show that too much refinement tends to over fit the rule. Therefore, we limit the refinement to be applied once on every condition.

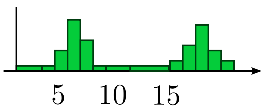
Head refinement

The head refinement consists in:

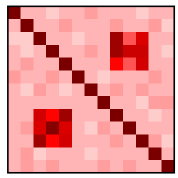
1. Compute a histogram of the distribution of a rule's head.

$$r_1 = \quad A \longrightarrow \underset{\mathbb{U}_{0,20}}{B}$$

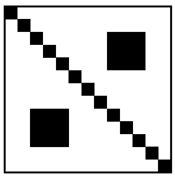
(a) input rule



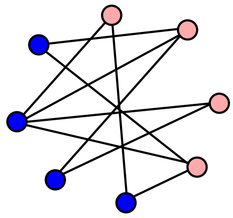
(b) histogram



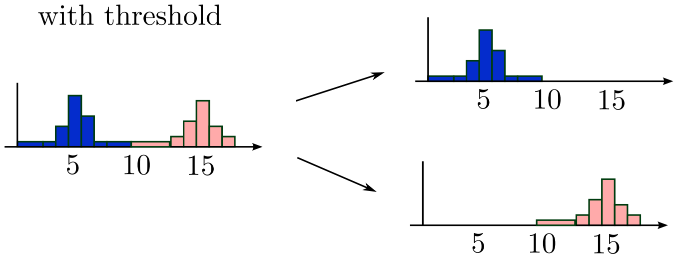
(c) co-occurrence matrix



(d) co-occurrence matrix with threshold



(e) colored independence graph



(f) colored distribution

$$r_2 = \quad A \longrightarrow \underset{\mathbb{U}_{0,10}}{B}$$

$$r_3 = \quad A \longrightarrow \underset{\mathbb{U}_{10,20}}{B}$$

(g) output rules

Figure 4.7: The steps of the policy for the division of rule.

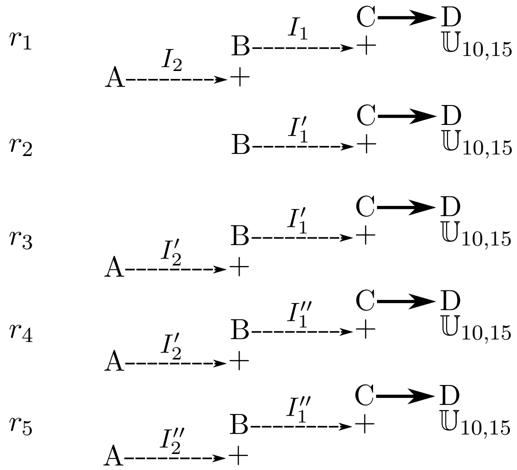


Figure 4.8: Five rules used to explain the refinement of a rule's body.

2. Apply a small Gaussian filter on the distribution.
3. Threshold this histogram with the **ruleRefinementThreshold** parameter.
4. Set the new distribution of a rule's head to be the thresholded histogram.

The figure 4.9 presents graphical this operation.

Body refinement

The body refinement consists in:

1. Compute all the successful occurrences of the rule i.e. successful predictions. For each prediction, we need to keep track of each conditions

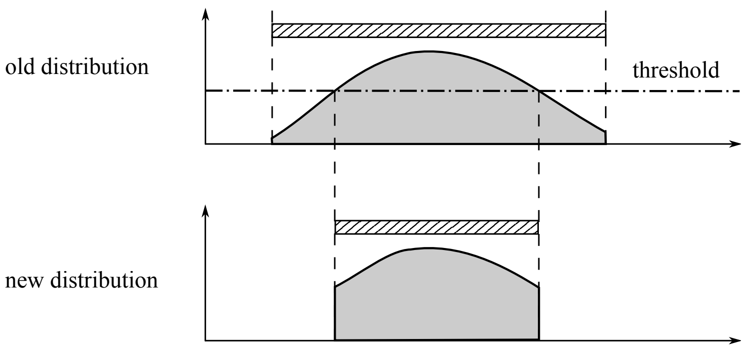


Figure 4.9: Graphical representation of the head refinement.

matching time i.e. for each predictions, we need to keep track of the mapping 'type 2 condition of the rule' → 'matching time'.

2. For each type 2 conditions of the rule c, compute the histogram h of $D := \{\text{time}_c - \text{time}_{\text{parent}(c)}\}$, where given a prediction, time_c is the time of matching of the condition c, and $\text{parent}(c)$ is the type 2 parent condition of c.

   Note: $\forall d \in D, m(d) = 1$ with $c := [m, c']$.

3. Threshold the histogram h with the **ruleRefinementThreshold** parameter.

4. Set the new Boolean function of the condition c to be m' with $m'(t) = 1$ if $h(t) > 0$, $m'(t) = 0$ otherwise.

4.3 The TITARL Algorithm listing

This section presents the TITARL algorithm. This algorithm is based on the four operations and the three policies presented in the previous sections.

The algorithm uses several user specified parameters:

- **minConfidence** as the minimum confidence of the rules to generate.
- **minSupport** as the minimum support of the rules to generate.
- **minPrecision** as the minimum temporal precision of the rules to generate.
- **maxRangeCondition** (or **mrc**) as the maximum size (or range) of a condition. If this parameter is very large (in comparison to the size of the dataset – this parameter can be infinity), rules representing correlations between very distant events will be extracted.
- **maxLoop** as the maximum number of loops of the algorithm. Other stopping criteria have been tried (maximum number of rules, maximum computation times, minimum entropy gain, etc.). These other criteria are not described in this manuscript.
- **minProbabilitiesDependency** is a set of real numbers. They express a minimum probability of dependency i.e. co-occurrence. This parameter is used on the process of rule division.
- **histogramBounds** is a set of real numbers describing the shape of histograms used to represent probability distribution. Fig. 4.10 shows several examples of histogram bounds.

- **ruleRefinementThreshold** is the real number, between 0 and 1, used to threshold histograms in the rule refinement process.

Algorithm 1: Tita Miner

Data: A symbolic time sequence I
Result: The set R of rules
begin
  let Sym to be the set of all symbols
  ▷ initialization
  compute R' the set of all trivial unit rules i.e.
  $R' := \{\langle w, \emptyset \rangle \Rightarrow x \langle 100\%, \mathbb{U}_{-\mathbf{mrc},+\mathbf{mrc}} \rangle \,|\forall w, x \in Sym\}$
  while *stopping criterion is not met* **do**
    for $r \in R'$ **do**
      ▷ division
      generate with the a division function d for r (see section 4.2.2)
      divide r according to d and store the result in $\{r_i\}$
      for $r' \in \{r_i\}$ **do**
        if $\operatorname{supp} r' \geq$ ***minSupport and*** r' *not already in* R **then**
          refine r' (see section 4.2.3)
          put r' in R
    ▷ addition of condition
    clear R'
    select a rule $r \in R$, a type 1 condition c_1 of r and a condition c_2 (see section 4.2.1)
    add the condition c_2 to r and store the result in R'
    i.e. replace c_1 of r by $\left\langle \text{symbol}_{c_1}, \text{conds}_{c_1} \cup \{c_2\} \right\rangle$
  return R

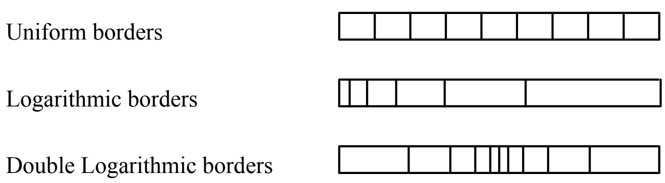


Figure 4.10: Three examples of histogram bounds.

4.4 Detailed step by step run

This section presents a step by step run of the algorithm on a simple dataset. This dataset is made of four types of events A, B, C and D. Simple correlations exist between these events:

(a) If there is an event A at time t, followed between 5 and 10 time units by and event B at time t', then there is an event C between $t'+5$ and $t'+10$.

(b) If there is an event B at time t', then there 50% chance to be an event C between $t'+5$ and $t'+10$.

(c) Additionally, events of type D are spread randomly through the dataset.

We are fixing the parameter **mrc** to 100 (maximum distance between two points in a condition).

1. The first task of the algorithm is to compute all the trivial unit rules:

$$\begin{aligned}
R' := \{&\langle A,\emptyset\rangle \Rightarrow A\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle B,\emptyset\rangle \Rightarrow A\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle C,\emptyset\rangle \Rightarrow A\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle D,\emptyset\rangle \Rightarrow A\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle A,\emptyset\rangle \Rightarrow B\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle B,\emptyset\rangle \Rightarrow B\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle C,\emptyset\rangle \Rightarrow B\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle D,\emptyset\rangle \Rightarrow B\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle A,\emptyset\rangle \Rightarrow C\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle B,\emptyset\rangle \Rightarrow C\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle C,\emptyset\rangle \Rightarrow C\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle D,\emptyset\rangle \Rightarrow C\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle A,\emptyset\rangle \Rightarrow D\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle, \langle B,\emptyset\rangle \Rightarrow D\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\\
,&\langle C,\emptyset\rangle \Rightarrow D\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\ \langle D,\emptyset\rangle \Rightarrow D\,\langle\sim 100\%,\mathbb{U}_{-100,+100}\rangle\}
\end{aligned}$$

2. We begin the first loop of the algorithm

3. Next, these trivial rules are divided (if necessary). R now contains rules such as:

$$\begin{aligned}
R := \{&r_1 := \langle B,\emptyset\rangle \Rightarrow C\,\langle 50\%, \mathrm{dist}_{r_1}\rangle\\
,&r_2 := \langle A,\emptyset\rangle \Rightarrow C\,\langle ?\%, \mathrm{dist}_{r_2}\rangle\\
,&\ldots\}
\end{aligned}$$

   R also contains a lot of rules with low precision.

   The distributions dist_{r_1} and dist_{r_2} are sampled on the dataset with a histogram. dist_{r_1} is a uniform distribution $\mathbb{U}_{5,10}$. dist_{r_2} has the shape of a convolution between two square functions. The fig. 4.11 represents the distributions dist_{r_1} and dist_{r_2}.

4. Next the rules are refined. Since we are considering here that the rules are perfectly learned, the refinement has a small impact.

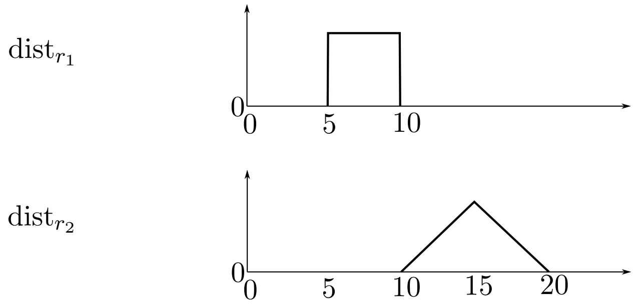


Figure 4.11: Distributions dist_{r_1} and dist_{r_2} of the step by step algorithm run.

5. Next, the algorithm selects a rule to augment (addition of condition). The addition of the condition $[T_{-10,-5}, \langle A, \emptyset \rangle]$ to the rule $\langle B, \emptyset \rangle \Rightarrow C \, \langle 100\%, \text{dist}_{r_1} \rangle$ has the highest information gain.

6. The rule $r_3 := \langle B, \{[T_{-10,-5}, \langle A, \emptyset \rangle]\} \rangle \Rightarrow C \, \langle 100\%, \text{dist}_{r_3} \rangle$ is created and added to the result.

At this point, the rule r_2 and r_3 express the correlations (a) and (b) of the dataset. Depending on the stopping criterion, the learning will continue a certain amount of time (the algorithm will continue to loop).

4.5 Discussion

This section presents a theoretical comparison between our work and the techniques presented in the chapter on related work (chapter 2). Two different aspects are discussed:

First, we discuss the expressive power of the temporal patterns. The power of expression of a pattern has direct implication on the complexity of learning. A pattern with a weak power of expression will be simple and computationally inexpensive to learn, but it might not be able to capture the complexity of a process. On the other hand, a pattern with too much power of expression might be too complex to be learned efficiently. In the next section, several common temporal patterns are listed, and a hierarchy is presented.

In the second part, we define and discuss the *Temporal Constraint Selection problem*. Since most temporal patterns are based on temporal constraints, this problem is common to most temporal learning techniques.

4.5.1 The expressive power of temporal patterns

The definitions of the temporal patterns listed in this section are given in chapter 2.

The *Temporal Constraint Network* (TCN) of Dechter et al. (Dechter *et al.*, 1991) was initially designed for automated planning and scheduling. TCNs is the most expressive of the temporal patterns we are studying here, therefore its formalism is a great tool to compare other temporal patterns. As presented in section 2.2, TCN are sets of temporal constraints over couples of events (the constraint is a binary operator), and these constraints can be represented as subsets of $\mathbb{R}$.

The *Chronicles* of Dousson et al. (Dousson and Duong, 1999) are equivalent to TCN. However, for efficiency in the learning process, the class of learnable Chronicles is restricted. The temporal intervals of learnable Chronicles have to be convex (this is equivalent to STP (Dechter *et al.*, 1991)).

The temporal patterns of Tita rules are less expressive than (unrestricted) Chronicles because Tita rules have a tree structure while Chronicles have an unconstrained (possibly complete graph) structure. However, Titars and learnable Chronicles can both express patterns that the other temporal pattern cannot express. Tita rules can have non-convex and negative temporal constraints.

There are several classes of Episodes depending of the learning algorithm and the author. All the Episode classes are less expressive than learnable Chronicles.

Tita rules are more expressive than any serial Episode class.

Oates et al. (Oates *et al.*, 1997) probability rules are less expressive than both restricted Chronicles and Tita rules.

The *Calendric Association Rules* introduced by Özden et al. (Özden *et al.*, 1998) uses the fact that a large number of phenomena are correlated with particular discretization of time (called cycles). For example, the schedule of people is generally cyclic over days and weeks e.g. people tend to wake up or eat approximatively at the same time every day and on all similar week days.

However, Calendric Association Rules cannot have more than one temporal condition. For example, Calendric Association Rules cannot express "If it is raining today, and if it was raining yesterday, then it will rain tomorrow".

The *Time Finite Automata with Granularities* (TFAG) pattern introduced by Bettini et al. (Bettini *et al.*, 1998) also uses the fact that a large number of phenomena are correlated with particular discretization of the time. TFAGs can express such patterns as "If it is raining today, and if it was raining yesterday, then it will rain tomorrow". It is important to remark that "it will rain tomorrow" is different from "it will rain in 24 hours" : If it is

23:59pm, "it will rain tomorrow" might be two minutes in the future.

If the granularity is precise enough, every TCN can be converted into a TAG. In addition, all TFAGs can be converted into a TCN (See proof 4.5.1).

Proof 4.5.1 *A TFAG of size 1 (i.e. a TAG with only one element) has no temporal constraint (Temporal Constraints with Granularity - TCG), therefore, it is also a TCN.*

Suppose a TFAG of size 2. Suppose the temporal constraint to be $[m,n]\mu$ *(m and n are two real numbers, and μ is a temporal type). By convention, Δ_μ is the number of time units between two new cycles of μ (i.e. if μ is 'day', and the time unit is the second, then $\Delta_\mu = 60 \times 60 \times 24$). Suppose the event e_μ to occur each time μ start a new cycle (i.e. if μ is 'day', then e_μ will occurs every day at midnight). Fig. 4.12 shows by construction how to convert this TFAG of size 2 into a TCN of size 4. Fig. 4.12 also shows an example of such transformation.*

Suppose a TFAG of size p with q temporal constraints ($q \leq \frac{p(p+1)}{2}$). By the same process, this TFAG can be converted into a TCN of size $p + 2 \times q$. Fig. 4.13 shows an example of conversion of a TFAG of size 3 into a TCN of size 9.

As it is discussed in 2.2.1, Sequential Patterns cannot capture information about time duration and time localization. The two types of extended Sequential Patterns presented in chapter 2.2.1 can express restricted information about time duration and time localization.

The fig. 4.14 sums-up the comparison of the power of expression of the different temporal patterns.

4.5.2 Temporal Constraint Selection problem

The *Temporal Constraint Selection problem* is composed of two sub-problems:

The first sub-problem is the number of possible temporal constraints: If the time is considered to be continuous, the number of possible temporal constraints is infinite. However, for the vast majority of learning techniques, the time is discretized. In this case, given a time interval discretized into n bins, the number of possible temporal constraints is 2^n. With such discretized time, and with m possible symbols, there is $(m^p)(m^p - 1)2^n$ possible TCN with p nodes. In conclusion, except for special cases, the number of possible temporal constraints is infinite or not enumerable. Examples of special case are Episodes: Episodes' constraints are restricted to be either $[-\infty, +\infty]$, $[-\infty, 0]$, or $[0, +\infty]$.

Another example of special case is the *Generalized Sequential Patter with Item Intervals* or the *Time-Interval Sequential Patterns*: In the case of these

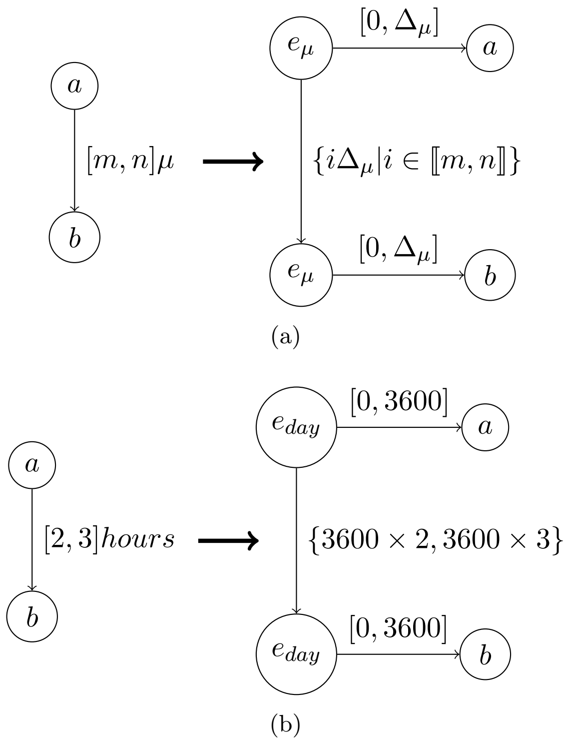


Figure 4.12: Definition and example of conversion of a size 2 TFAG into a size 4 TCN. The time unit is the second.

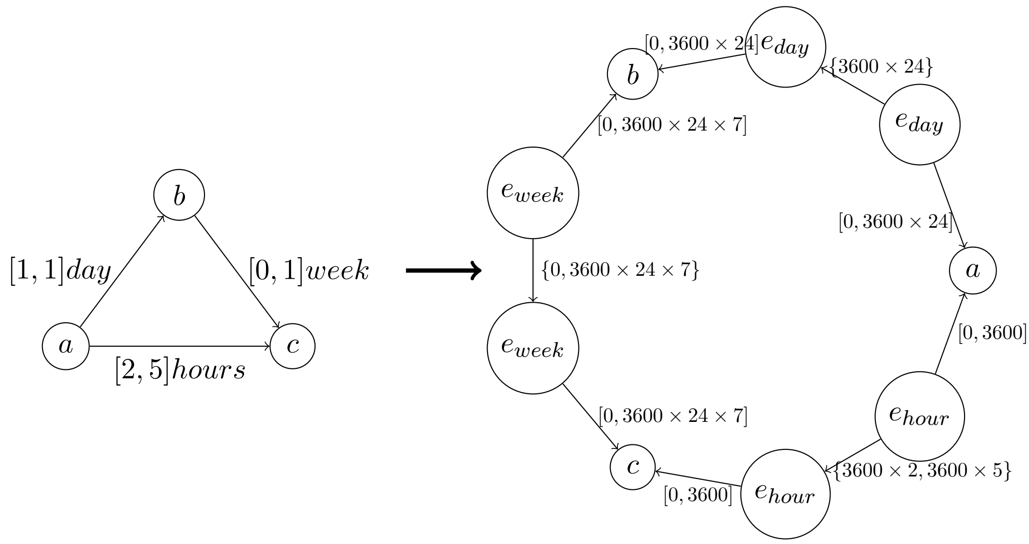


Figure 4.13: Example of conversion of a size 3 TFAG into a size 9 TCN. The time unit is the second.

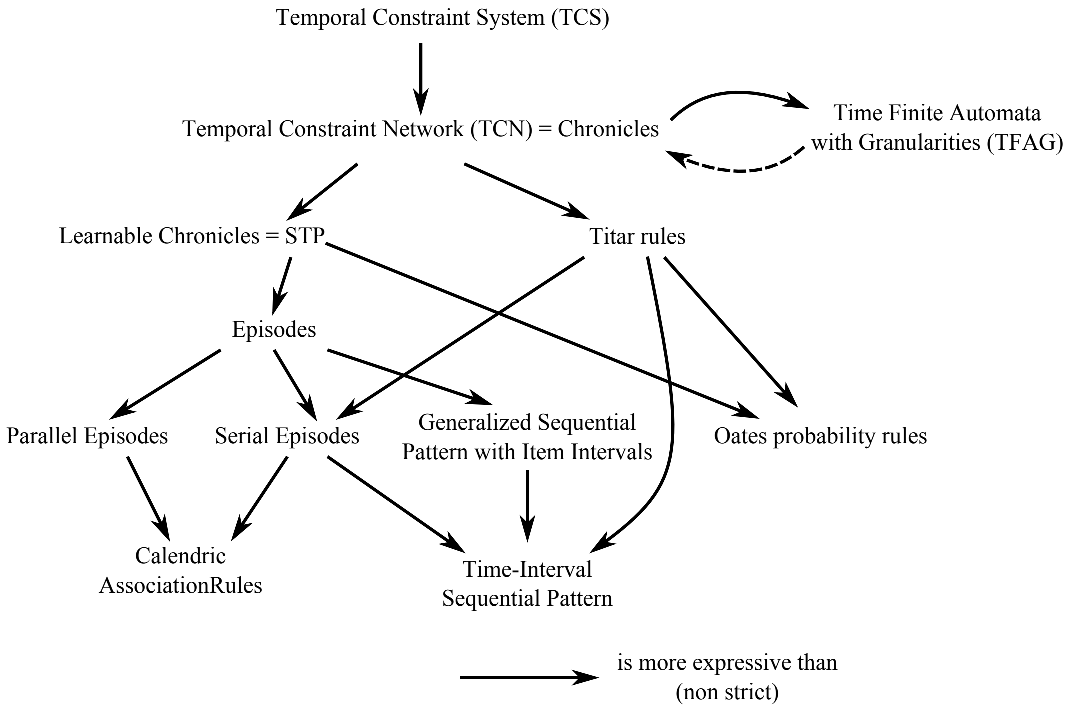


Figure 4.14: Expressive power relation between the different temporal patterns.

two patterns, the constraints are restricted to be in $\{[0, T_1], [T_1, T_2], [T_2, T_3], \cdots, [T_{r-1}, T_r], [T_r, \infty]\}$ where $\{T_i\}_i$ are user defined boundaries.

The second problem with the learning of temporal constraints is the fact that the "quality" of a temporal constraint, and more generally the "quality" of a TCN, depends on the domain of application. As far as we know, and as far as we have searched, there is no universal metric for the quality of a TCN. Existing solutions rely on weight, threshold of fixed value, simple metrics such as confidence, support, range, or on closure.

In the case of temporal association rule learning, temporal constraints for the head of a rule (temporal constraint of prediction) and temporal constraints for the body of a rule (temporal constraint of condition) have different semantics. Several techniques found the literature use the same criterion to learn temporal constraints for the head and temporal constraints for the body of rules. Our solution takes into account these differences: Two different criteria are proposed and justified. The next two sections discuss these two aspects.

We believe that the *Temporal Constraint Selection problem* it is the main problem of (TCN) temporal learning.

Temporal Constraint Selection for rules' head

The most common metric to evaluate rules is the *confidence*. This metric expresses 'how true' a rule is.

The second most common metric to evaluate rules is the *support*. This metric expresses the quantity of events a rule can predict.

It is usually easy to extract rules with high confidence and low support, or rules with low confidence and high support. However, these both kinds of rules are generally not very interesting. The confidence and the support of a rule are correlated: It is possible to increase the confidence of a rule while decreasing its support and vice versa. Therefore, learning algorithms have to tackle the following problem: What is the good compromise between confidence and support for a rule? The usual solution in data-mining is to define a minimum support and a minimum confidence. Every rule that holds these two constraints is considered to be good. Other solutions include linear and non linear combinations of confidence, support and other simple metrics (Geng and Hamilton, 2006).

The *apriori* (Agrawal and Srikant, 1994) is a technique frequently used to enumerate patterns with a minimum support. Unfortunately, no such technique exists to enumerate patterns with a minimum confidence.

In the context of this thesis, temporal association rules need to have a third metric to express how precise they are. Indeed, it is trivial to extract temporal rules with 100% confidence and 100% support in symbolic time sequence: Suppose two symbols A and B such that there is at least one occurrence of each of them. The rule saying 'If the event A occurs at time t, then the event B occurs between $[t-\infty, t+\infty]$' has a 100% confidence and support. However such a rule (called unit trivial rule) is useless. To deal with this problem, a third metric must be introduced: The *prediction time range* of a rule expresses how accurate the predictions of this rule are. This is a measure of the size of temporal interval of the rule's head. We call the *precision* of a rule, the inverse of its prediction time range. A good rule has usually a low prediction time range i.e. a high precision. The temporal constraint of the head of a rule has a direct implication on the temporal precision of a rule. The three metrics confidence, support and precision are highly dependent. Increasing the precision of a rule usually decease its confidence and support.

In the EpiBF algorithm (Casas-Garriga, 2003), the user defines the 'time unit separation' parameter which fixes the rule's head temporal constraint to
$[-time\_unit\_separation, +time\_unit\_separation]$.

In the MinEpi algorithms (Mannila *et al.*, 1997), the user defines the 'w2 $\in \mathbb{R}^+$' parameter that directly fixes the rule's head temporal constraint to $[0, w2]$.

In the WinEpi algorithm (Mannila *et al.*, 1997) and most of the other algorithm used to learn Episodes, the user defines the 'window size $\in \mathbb{R}^+$' parameter that directly fixes the rule's head temporal constraint to $[0, window]$.

In the Face algorithm (Dousson and Duong, 1999), the user defines a parameter '$it_{min} \in [0,1]$'. This parameter is related (in a non trivial way) to the support/confidence/precision of the rules that can be extracted from Chronicles. The temporal constraint of a 2-Chronicle c is the smallest connect interval such that the rule has a frequency of it_{min} times the frequency of c', where c' is the unconstrained Chronicle associated with c.

Given two types of events, the algorithm of Pablo Hernandez et al. (Hernandez-Leal *et al.*, 2011) selects several "good" temporal constraint based on the analysis of the distribution $P(t'-t|A \text{ occurs at } t \text{ and } B \text{ occurs at } t')$. The algorithm supposes that:

1. The good temporal constraint (intervals) can be found by the analysis of the shape of the distribution.
2. The distribution is a Gaussian mixture model.
3. Given a Gaussian, the good interval is $[\mu-\tau, \mu+\tau]$ (and not $[0.5\mu - \tau, 0.5\mu+\tau]$ or $[2\mu-\tau, 2\mu+\tau]$ for example).

In the I-Apriori, I-PrefixSpan and the Generalized Sequential Pattern Mining with Item Intervals algorithms (Chen *et al.*, 2003; Hirate and Yamana, 2006) the user defines the 'window size $\in \mathbb{R}^+$' parameter that directly fixes a set of possible temporal constraints. These temporal constraints are restricted to be convex, mutually disjunctive and to not include negative numbers.

The algorithm we have developed does not require the user to fix the rule's head temporal constraint.

In the four algorithms EpiBF, MinEpi, WinEpi and Face, the temporal constraints must be convex. Therefore, these algorithms cannot learn patterns with non-convex prediction ranges. An example of rule with a non convex prediction range is 'If A occurs are time t, then B will occur between $t+5$ and $t+10$, or between $t+15$ and $t+20$'.

Remark 4.5.2 *The semantic of a rule with a non convex prediction range cannot be expressed with rules with convex prediction ranges. For example, the rule 'If A occurs are time t, then B will occur between $t+5$ and $t+10$, or between $t+15$ and $t+20$', it not equivalent to the rules 'If A occurs are time t, then B will occur between $t+5$ and $t+10$' and 'If A occurs are time t, then B will occur between $t+15$ and $t+20$'.*

Titarl can learn rules with non convex prediction ranges. The algorithms EpiBF, MinEpi and WinEpi can only learn rule with rule's head temporal constraint that contains 0. For example, these three algorithms cannot learn the rule 'If A occurs are time t, then B will occur between $t+5$ and $t+10$'. The algorithm we developed can learn rules without 0 in the rule's head temporal constraint.

Experiments shows that Face can deal with very small datasets, but it is not robust to noise. In addition, Face is not robust to learn rules in dense dataset i.e. when occurrences of rules are mixed together. Finally, in the case of non-noisy and non-mixed datasets, Face may be unable to learn rules. The example 4.5.3 shows a step by step application of the Face algorithm to support the last declaration.

Example 4.5.3 *Suppose a dataset composed of two types of events A and B. All the events occur between times* 0 *and* 1000*. 50 events of type A are randomly located. No events of type A are closer to each other than 10 time units i.e. the dataset is not dense. For every event A located a time t, an event B is located between time $t+5$ and time $t+6$. Therefore, the rules are not mixed together. There are no other events i.e. there is no noise. The rule to learn is 'if an event A occurs at time t, then an event B occurs between time $t+5$ and time $t+6$. This rule can be expressed with a 2-Chronicle (Chronicles are the temporal pattern used by the Face algorithm). This chronicle contains one temporal constraint. This temporal constraint will be the interval* $[5,6]$*. The prediction range is the length of the temporal constraint (the prediction range is 1 in this case).*

The Face algorithm is presented in section 2.3. In this algorithm, the selection of the temporal constraint of a 2-Chronicle is done at step 2.c with the special case k=1. This temporal constraint is an interval. Let's call this interval T. The Face algorithm uses one user specified parameter called $it_{min} \in [0,1]$. Let's F_{max} be the frequency (the number of occurrences) of the Chronicle is the temporal constraint was $]-\infty,+\infty[$. it_{min} define the frequency of the Chronicle such as 'frequency of the extracted Chronicle' $\leq it_{min}F_{max}$. In this particular scenario, if it_{min} is in the interval $[\frac{1}{50},\frac{2}{50}[$, the correct temporal constraint will be extracted. Therefore, the correct Chronicles will also be generated. If it_{min} is lower than $\frac{1}{50}$, no chronicle will be extracted. If it_{min} is higher of equal than $\frac{2}{50}$, the extracted interval will be too large, and the chronicle will be imprecise. More precisely, the expected prediction range of the extracted chronicle will be $1+(it_{min}-\frac{1}{50})\times\frac{1000}{2}$. The prediction range of the extracted chronicle is unstable: Small change on the parameter will change dramatically the solution. The experience shows that, given a dataset, there is only a small window of good values of the parameter (which should be fixed by the user).

In all these techniques, the selection of the temporal constraint is either fix by the user, either computed directly or indirectly from the distribution

$P(t' - t|A$ occurs at t and B occurs at $t')$. However, this distribution is is not a reliable information to select temporal constraints. This distribution is especially ineffective if the dataset is noisy, if the dataset is dense , if the temporal constraint of the rules as an unusual shape, or if rules interfere with each other.

The example 4.5.4 shows two different datasets with different temporal rules, and where the temporal distributions are the same.

Example 4.5.4 *Suppose two datasets D_1 and D_2. Those two datasets only containt events of type a or b.*

The dataset D_1 is defined as follow:

> *Events of type a are randomly distributed. If there is an event of type a at time t therefore: There is an event of type b around $t + 5$. There is an event of type b around $t + 15$.*

The dataset D_2 is defined as follow:

> *Events of type a are randomly distributed. If there is an event of type a at time t therefore: There is an event of type b a bit before $t + 5$ or a bit after $t + 15$. There is an event of type b a bit after $t + 5$ or a bit before $t + 15$.*

Suppose the unit rule "If there is an event of type a at time t then, there is an event of type b between t and $t + 20$". In the two datasets D_1 and D_2, the estimated temporal distribution between the head and the body of this rule is the same (see fig. 4.15). So, any technique only relying on this distribution will behave exactly the same for those two datasets, and therefore the learning on one of those datasets will be incorrect.

This example illustrates that only analyzing temporal distributions between temporal events is not a sound solution to extract temporal patterns.

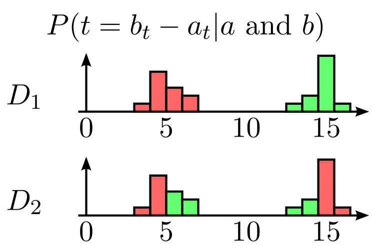


Figure 4.15: Temporal distribution in the two datasets of the example 4.5.4. The colors show the ideal splitting for the rule in the two datasets.

Therefore, in our approach, we do not rely on this distribution, but instead, we rely on the 'co-occurrence matrix' to split rules and estimate their temporal distributions.

Temporal Constraint Selection for rules' conditions

The vast majority of algorithms use the same or a similar process to select the temporal interval of prediction and conditions.

In the WinEpi algorithm (Mannila *et al.*, 1997), the temporal interval of conditions is defined by the user parameter 'window size $\in \mathbb{R}^+$'. This is the same parameter that defines the temporal interval of prediction.

In the MinEpi algorithm (Mannila *et al.*, 1997), the temporal interval of conditions is defined by the user parameter 'w1 $\in \mathbb{R}^+$'.

In the EpiBF algorithm (Casas-Garriga, 2003), the temporal interval of conditions is defined by the user parameter 'time unit separation' $\in \mathbb{R}^+$'.

In the Face algorithm (Dousson and Duong, 1999), the temporal interval of conditions is selected like the temporal interval of prediction (see previous section).

In our approach, we select temporal constraints for rule's body base on a entropy minimisation policy. In addition, the temporal constraints are refined each time a rule is augmented with a new condition.

5 Temporal planning

Chapter Contents

The previous chapter presents a technique to extract Tita rules from Symbolic Time Sequences. In this chapter, we present a planning and scheduling algorithm based Tita rules. In order to do planning and scheduling, we always suppose in this chapter that the set of input rules express causal relations. Example 5.0.5 shows a non causal rule.

Example 5.0.5 *Suppose a system where three types of events can occur: A, B and C. To help the understanding of this example, you can suppose the system to be a person inside a car, and the events A, B and C to be:*

1. *A : To start up the car.*
2. *B : Hearing an engine noise.*
3. *C : The lamps of the dashboard come on.*

Suppose two following rules in the system:

1. *If A occurs at time t, then B will occur at time $t + 5$.*
2. *If A occurs at time t, then C will occur at time $t + 7$.*

These two rules are true because of the mechanic of the car (we suppose the car is working correctly).

From the point of view of somebody who has no understanding of the mechanic of the car, the rule 'if B occurs at time t, then C will occur at time $t + 2$' also seems to be valid. This rule seems to be true because the person has no observation of counter examples. However, this rule is a non causal rule *since it is not actually the event B that triggers the event C.*

An example of counter example is: Suppose the same person. He is living in a quiet place, and he goes in a city. At some point, he is sitting in his car, and he ears the noise of the engine of another car passing by. In this situation, the lamps of the dashboard will not come on; therefore the rule 'if B occurs at time t, then C will occur at time $t + 2$' is invalid.

We developed a temporal planning and scheduling algorithm that is able to use Tita rules as input rule (see definition in sections 3.5). Tita rules can express uncertain (probabilistic) and (temporally) inaccurate associations between events and states. The algorithm deals with inaccurately located events and zero duration events.

Similar to the Strips algorithm (Fikes and Nilsson, 1971), our algorithm works with *backward chaining* (or retrograde analysis). As to the Deviser algorithm (Vere, 1983), our algorithm deals with time windows on actions and goals. Checking conflicts between positive and negatives part of the plan is time expensive. Therefore, in the case of rules with negations, our algorithm does not check for conflicts between the different parts of the plan. Our solution is to generate instances of plans and check if the goal of the plan is actually generated. This last part is a heuristic.

The algorithm is composed of four stages.

1. The first stage called TitaPlanner builds a Tita plan (see definition in 3.8) from a single Tita rule, a set of scheduled events and states, a goal symbol and a time window for this goal symbol.

2. The second stage called MetaTitaPlanner is building a Meta Tita plan (see definition in 3.9) from a set of Tita rules, a set of scheduled events and states, a goal symbol and a time window for this goal symbol. Informally, a Meta Tita plan is a set of Tita plans bond together to achieve an objective (Meta Tita plans have a greater power of expression than Tita plans). This second stage (MetaTitaPlanner) is based on the first one (TitaPlanner).

3. The third stage 'linearises' Meta-plans (output of the second algorithm) into strong linear Meta-plans (see definition in 3.9.6). Strong linear Meta-plans are a sub-class of linear Meta-plans. A linear Meta-plan can be decomposed into strong linear Meta-plans. This stage decomposes linear Meta-plans and returns the strong linear Meta-plan with the highest confidence.

4. The final stage performs scheduling. This stage takes as input a strong linear Meta-plan (output of the third algorithm), and return a list of nominal time-sampled actions.

Section 5.1 introduces the different stages of the algorithm. Section 5.2 presents a step by step detailed run of the algorithms. Section 5.3 discusses the complexity of the algorithms.

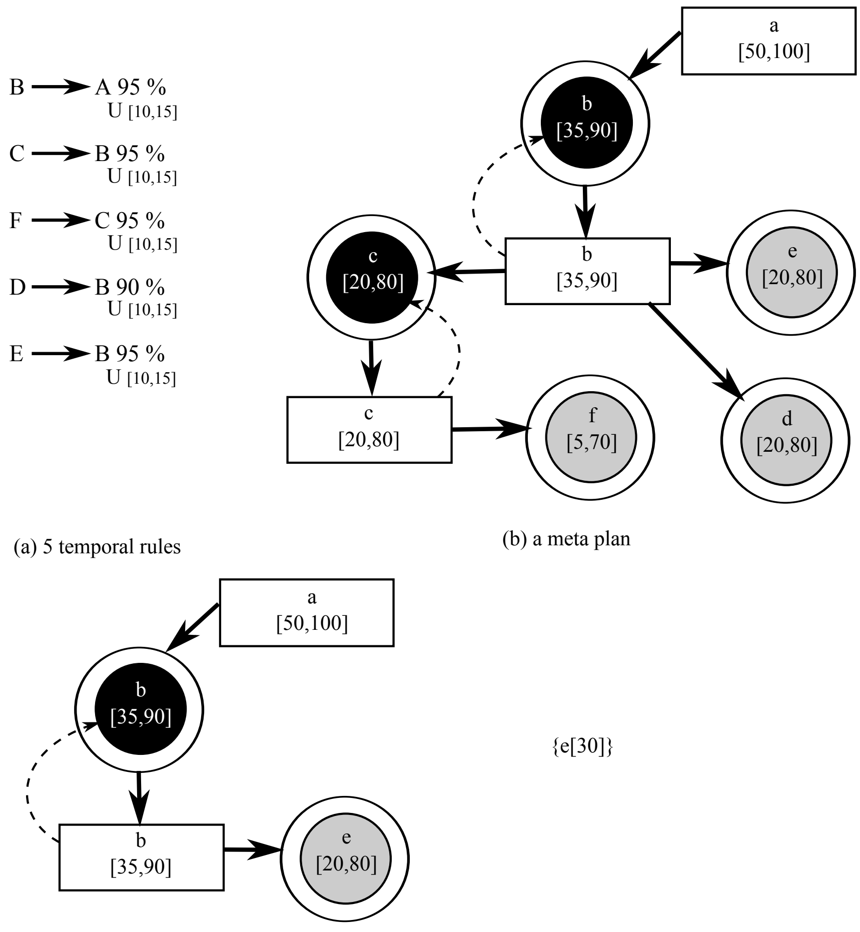
5.1 The algorithm

This section presents the four stages of the temporal planning and scheduling algorithm. Figure 5.1 shows a complete example of Meta-planning.

5.1.1 Stage 1: Tita planner

The Tita planner takes as input a Tita rule, a set of scheduled events and states, a goal symbol and a time window for this goal symbol. It generates a Tita plan describing how to produce the goal symbol in the time window request with the given rule. The given rule should have as a head symbol the requested goal symbol. The user provides a parameter defining the minimum confidence for the plan to build. The output plan has the same tree structure as the input rule.

Next, we present three simple examples that illustrate the behavior of the Tita planner. In these three examples, the objective is to find a way to generate an event of type A in the interval $R = [50, 100]$. These examples show the underling idea of the computation of a plan with each of these rules.



(a) 5 temporal rules (b) a meta plan

(c) the "best" strongly linear plan (d) an instance for this strongly linear meta plan

Figure 5.1: This figure shows a complete example of Meta-planning. The Meta-plan (b) is computed from the set of Tita rules (a). Next, a strong linear Meta-plan (c) is extracted from the Meta-plan (b). In this example, the strong linear Meta-plan (c) is the strong linear Meta-plan with the highest confidence that can be extracted from (b). Finally, an instance (d) is computed from the strong linear Meta-plan (c).

Example 5.1.1 *Suppose the rule r_1 introduced in fig. 5.2. Let $f(t) = ((-\operatorname{dist}_{r_1}) * R)(t)$. If an event of type B occurs at time t, then $f(t)$ is the probability of having an event of type A in the interval R because of the rule r_1. $f(x)$ threshold by $c \in \mathbb{R}$ defines a binary function b. The plan 'trigger an event B at time t with $b(t) = 1$' has a probability of c to generate an event of type A in the interval R. The fig. 5.2 shows R, b and f.*

Example 5.1.2 *Suppose the rule r_2 introduced in fig. 5.2. This rule is equivalent to the rule r_1 with an extra condition. Suppose R, f and b as defined in example 5.1.1. In the rule r_2, the temporal constraint between B and C is the Boolean function $m = \mathbb{B}_{-10,-5}$ (i.e. $m(x) = 1$ if and only if $x \in [-10, -5]$). If an event of type B is triggered at time t_1 and an event of type C is triggered at time t_2 with $m(t_2 - t_1) = 1$, then $f(t_1)$ is the probability of having an event of type A in the interval R because of the rule r_2.*

Example 5.1.3 *Suppose R, f, m and b as defined in examples 5.1.1 and 5.1.2. Suppose the rule r_2 introduced in fig. 5.2. Suppose that events of type C cannot be triggered (i.e. C is an exogenous symbol). Suppose that an event of type C occurs at time 50. Let $b' = \mathbb{B}_{55,60}$. In order for the*

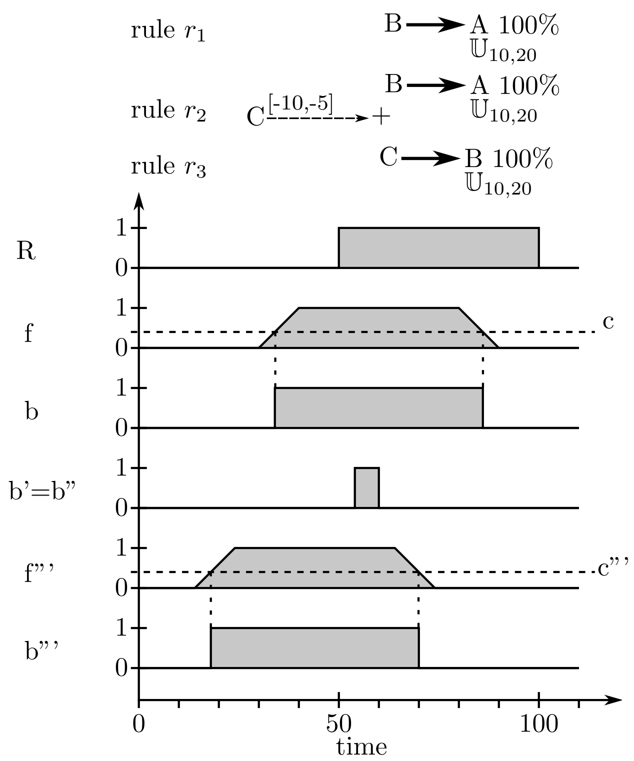


Figure 5.2: The rule r_1, r_2, r_3, R, b, f, b', b''', f''' used in the examples 5.1.1, 5.1.2, 5.1.3 and 5.1.4 of behavior of the Tita planner.

constraint between B and C to be valid, an event B has to occur at time t with $b'(t) = 1$. Let $b''(t) = b(t) \cdot b'(t)$. If an event of type B occurs at time t with $b''(t) = 1$, then $f(t)$ is the probability of having an event of type A in the interval R because of the rule r_2. If an event of type B occurs at time t with $b''(t) \neq 1$, then the rule r_2 cannot be applied or the generated event C is not in the R.

Stage 1 is given as follows:

1. **Build a Tita plan p from a Tita rule r**
   The Tita plan p is build recursively with the function TitaPlanner.

2. **Propagate the absolute constraints in the Tita plan p**
   The propagation can be achieved with two depth-first explorations of the plan's tree. If an absolute constraint becomes null, the plan is discarded (or invalid). The two equations for the propagation of the constraints are the following ones:

   For every edge $e := v \to v'$ of the plan p:

$$(\mathrm{Acst}_{v'} * (-\mathrm{Rcst}_e))(x) = 0 \Rightarrow \mathrm{Acst}_v(x) = 0 \tag{5.1}$$

$$(\mathrm{Acst}_v * \mathrm{Rcst}_e)(x) = 0 \Rightarrow \mathrm{Acst}_{v'}(x) = 0 \tag{5.2}$$

Function TitaPlanner

Data:
A Tita rule r ▷ The source of the plan
A Tita plan p ▷ The output plan
A request interval R ▷ The interval for the head of the rule to occur
The minimum confidence of the plan $minConf$
Result: The Tita plan p
begin
  initialise p to be an empty graph
  Let b be the Boolean function such as:

$$b : x \mapsto \begin{cases} 1 & \text{if } ((-\mathrm{dist}_r) * R)(x) \geq \frac{minConf}{\mathrm{conf}_r} \\ 0 & \text{otherwise} \end{cases}$$

  add a vertex v to p with $\mathrm{Acst}_v = b$
  call SubTitaPlanner(conds_r,v,'positive')

Function SubTitaPlanner(c:a type 1 condition, v:a vertex of the plan p, l:'positive' or 'negative')

Data:
A Tita plan p ▷ The output plan
A symbolic time sequence I
A set of doable symbols D
A time interval describing when the doable symbols can occur T

begin
- **set** $\text{symbol}_v = \text{symbol}_c$
- **let** m be the Boolean function of symbol_v activation i.e. $m(t) = 1$ when there is an event of type symbol_v at time t.
- **if** symbol_v *is doable i.e.* $\text{symbol}_v \in D$ **then**
  - **set** $m'(x) := \begin{cases} 1 & \text{if } m(x) = 1 \text{ or } T(x) = 1 \\ 0 & \text{otherwise} \end{cases}$
- **else**
  - **set** $m' = m$
- **set** $\text{Acst}_v(x) := \text{Acst}_v(x) \cdot m'(x)$
- **if** $\forall x, \text{Acst}_v(x) = 0$ *i.e.* Acst_v *is null* **then**
  - **discard** the plan
  - **return**
- **for** $c' \in \text{conds}_c$ **do**
  - **if** c' *is a negation of a type 2 condition* c'' **then**
    - **if** $l =$ *'positive'* **then**
      - **call** SubTitaPlanner(c'',v,'negative')
    - **else if** $l =$ *'negative'* **then**
      - **call** SubTitaPlanner(c'',v,'positive')
  - **else if** c' *is condition over a state* s **then**
    - **let** m be the Boolean function of s activation i.e. $m(t) = 1$ when s is true at time t.
    - **if** s *is doable i.e.* $s \in D$ **then**
      - **set** $m'(x) := \begin{cases} 1 & \text{if } m(x) = 1 \text{ or } T(x) = 1 \\ 0 & \text{otherwise} \end{cases}$
    - **else**
      - **set** $m' = m$
    - **set** $\text{Acst}_v(x) := \text{Acst}_v(x) \cdot m'(x)$
    - **if** $\forall x, \text{Acst}_v(x) = 0$ *i.e.* Acst_v *is null* **then**
      - **discard** the plan
      - **return**
    - **add** s to the set of states states_v of v
  - **else if** c' *is an association between a Boolean function* m *and a type 1 condition* c'' **then**
    - **add** a vertex v' to p with $\text{Acst}_{v'} = \text{Acst}_v * m$
    - **add** an edge $e := v \to v'$ with $\text{Rcst}_c = m$
    - **set** the label of e to be l
    - **call** SubTitaPlanner(c'',v','positive')

5.1.2 Stage 2: Tita Meta-planner

The Meta Tita planner takes as input a set of Tita rules, a set of scheduled events and states, a goal symbol and a time window for this goal symbol. It generates a Meta Tita plan that contains the information of how to produce the goal symbol in the time window request with a combination of the given rules. The user provides a parameter defining the minimum confidence the Meta-plan to build.

Stage 2 uses stage 1 to build Tita plans, and it combines them into a Meta Titar plan. The temporal constraints of the plans are propagated trough the Meta-plan. The following example illustrates the behavior the Meta Tita planner. It shows the combination of two rules into a Meta Tita plan.

Example 5.1.4 *Assume the two rules r_3 and r_4 displayed in fig. 5.2. As with to the last example, the objective is to generate an event of type A in the interval $R = [50, 100]$. Suppose that C is a doable symbol but that B is an exogenous symbol. The functions of this example are displayed in the fig. 5.2.*

First, the planner tries to apply the rule r_3 to generate A in the interval R. The condition for r_3 to be applied is to get an event of type B at time t with $b(t) = 1$, with b as defined in the example 5.1.1 (This step is similar to the example 5.1.1). Since B is an exogenous symbol, it needs to be generated by a rule.

The same process is applied again with the rule r_4 to generate an event of type B at time t with $b(t) = 1$. Let's define $f'''(t) = ((-\operatorname{dist}_{r_4}) * b)(t)$, and b''' the threshold of the function f''' with $c''' \in \mathbb{R}$. If there is an event C at time t' with $b'''(t') = 1$, then the rule r_4 generates an event of type B at time t with $b(t) = 1$, and then the rule r_3 generates an event of type A in the interval R.

The listing of the stage 2 is given as follows:

1. **Build a Meta Tita plan** rp
   The Meta Tita plan is build with the function MetaTitaPlanner.

2. **Propagate the absolute constraints in the Meta-plan and the plans**
   The propagation into the Meta-plan and plans can be achieved with two depth-first explorations.

   If the absolute constraint of a 'problem' vertex v becomes null, the vertex and all its children are deleted from the graph. If the 'caused' vertex v of a plan p labelled to a 'solution' vertex v' is not labelled by any problem vertex, the 'solution' vertex v' and all its children are deleted from the graph. If the recursive plan does not contain any vertices, it is discarded.

Propagation of constraints must satisfy the following two conditions.

a) The absolute constraint of every caused vertex v should be equal to the absolute constraint of the 'problem' vertex v' with is labelled to v i.e. $\text{subLeaf}_{v'} = v$ implies $\text{Acst}_v = \text{Acst}_{v'}$

b) Suppose a 'problem' vertex v. Suppose an edge $e := v \to v'$ with v' a 'solution' vertex labelled with a plan p. Suppose v'' the trunk of the plan p. The two equations of propagation are:

$$((-\operatorname{dist}_r) * \text{Acst}_v)(x) < \frac{minConf}{\text{conf}_r} \Rightarrow \text{Acst}_{v''}(x) = 0 \tag{5.3}$$

$$(\operatorname{dist}_r * \text{Acst}_{v''})(x) < \frac{minConf}{\text{conf}_r} \Rightarrow \text{Acst}_v(x) = 0 \tag{5.4}$$

With r the rule of the plan p and $minConf$ the minimum confidence of the plan p.

Function MetaTitaPlanner

Data:
A Meta Tita plan rp ▷ The output Meta-plan
A request symbol s ▷ The objective of the plan
A request interval R ▷ The interval for the objective to occur
The minimum confidence of the plan $minConf$
Result: The Meta Tita plan rp
begin
  initialise rp to be an empty graph
  add a 'problem' vertex v to rp with $\text{Acst}_v = R$ and $\text{symbol}_v = s$
  call SubMetaTitaPlanner(v,$minConf$)

5.1.3 Stage 3: Meta-plan linearization

The linearization (of a Meta-plan) is the extraction of the valid *strong linear Meta-plan* (see definition is section 3.9.6) with the highest confidence. Strong linear Meta-plans are important because they are the intermediate step into the computation of a (Meta) plan instance.

We are presenting an efficient algorithm to linearise Meta-plan. This algorithm requires as input a *weak linear Meta-plan* without negation of doable symbols.

The algorithm enumerates all the paths (strong linear Meta-plans) of the input Meta-plan (a Meta-plan has a tree graph structure). The confidence of a path is the product of the confidences of the plans associated with its vertices. In the worst case, the number of paths of a tree graph is equal to the number of nodes minus one.

If the algorithm is fed with a *weak linear Meta-plan* with negation, it can produce invalid plans. A simple heuristic to deal with such Meta-plans is to enumerate the Meta-plans and test its validity. In this case, we are returning the valid Meta-plan with the highest confidence.

5.1.4 Stage 4: Building of (Meta) plan instance

Given a plan we can compute an instance of this plan (see definition 3.8.2). By convention, an instance of a strong linear Meta-plan is an instance of the plan associated with the (unique) leaf of the Meta-plan.

To compute such instance, the plan structure is explored with a depth first search. At each doable node n of the plan, an event of type symbol_n and time time_t is added to the instance, such that time_t satisfy the absolute constraint of n and the relative constraint of the parent of n.

5.2 Detailed step by step run

This section describe step by step the TitaPlanner and MetaTitaPlanner algorithms presented in this chapter: The building of a plan and the building of a Meta-plan.

5.2.1 Building of a plan

We present the step of the TitaPlanner algorithm to build the plan shown in fig. 5.4 based on the rule presented in fig. 5.3. In this example, we set the minimum confidence to 0. The goal is to find a plan to produce an event a between 50 and 100.

1. The b Boolean function is computed to be $\mathbb{B}_{35,90}$.

2. The vertex v_1 associated with b is added.

3. The absolute constraint of v_1 is set to b.

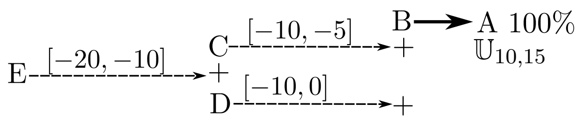


Figure 5.3: Example of Tita rule.

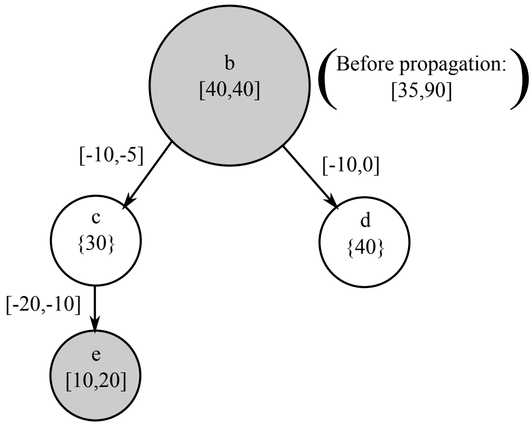


Figure 5.4: Example of Tita plan. A similar plan has already been presented and explained in fig. 3.6.

4. The vertex v_2 associated with c is added.
5. An edge is added between v_1 and v_2.
6. The relative constraint of this edge is set to $\mathbb{B}_{-10,-5}$ (based on the rule's condition).
7. The absolute constraint of v_2 is computed: c is exogenous. There is one occurrence of c at time 30. The absolute constraint of v_2 is $\mathbb{B}_{30,30}$
8. The vertex v_3 associated with e is added.
9. An edge is added between v_2 and v_3.
10. The relative constraint of this edge is set to $\mathbb{B}_{-20,-10}$ (based on the rule's condition).
11. The absolute constraint of v_3 is computed from the relative constraint between v_2 and v_3, and the absolute constraint of v_2.
12. The vertex v_4 associated with d is added.
13. An edge is added between v_1 and v_4.
14. The relative constraint of this edge is set to $\mathbb{B}_{-10,0}$ (based on the rule's condition).
15. The absolute constraint of v_4 is computed: d is exogenous. There is one occurrence of d at time 40. The absolute constraint of v_4 is $\mathbb{B}_{40,40}$
16. We propagate the equations 5.2 and 5.1: The absolute constraint of the nude labelled with b becomes $\mathbb{B}_{40,40}$.

5.2.2 Building of a Meta-plan

We present the step of the MetaTitaPlanner algorithm to build the Meta-plan shown in fig. 5.1 based on the rules presented in the same figure. In this example, we set the minimum confidence to 0 and the parameter N to 1. The goal is to produce a Meta-plan to produce the event a between 50 and 100.

1. Create a 'problem' node for a with an absolute constraint of $\mathbb{B}_{50,100}$.
2. There is only one rule that can produce a. Add a 'solution' node, and compute a plan for a based on this rule.
3. Label this 'solution' node with the computed plan.
4. The body b of this rule is exogenous. Add a 'problem' node for b.
5. There are three rules that can produce b. For every of these rules, add a 'solution' node and compute a plan for b based on this rule.
6. Label these 'solution' nudes with the computed plans.
7. The body c of one of these rules is exogenous. Add a 'problem' node for c.
8. There is only one rule that can produce c. Add a 'solution' node and compute a plan for c based on this rule.

5.3 Algorithms computation complexity

This section presents a computation complexity analysis of the four algorithms presented in the chapter.

5.3.1 Tita planner

The Tita planner is a recursive exploration of the input rule structure with a branch cutting optimization. Therefore, in the worst case, the number of calls of the main function SubTitaPlanner is equal to the number of conditions in the rule. The number of conditions of a rule is related to the size of the training dataset. In our experiments, the number of conditions of the rules rarely exceeds eight, and is generally under five. The more expensive operation in the SubTitaPlanner function is the dichotomic search of event which has an algorithmic complexity of $O(log\ n)$ with n the number of events (we suppose the number of types of events to be fixed).

Therefore, the worst case algorithmic complexity of the Tita planner is $O(c\ log\ n)$ with c the number of conditions of the rule and n the number of event in the dataset.

5.3.2 Meta Tita planner

The Meta Tita planner is a recursive exploration of different combinations of rules with a branch cutting optimization. In the worst case, the function SubMetaTitaPlanner calls recursively itself $O(m\ N\ 2^c)$ times, with m the number of rules fitting the current sub goal, N the parameter of the algorithm (usually fixed around 4) and c the maximum number of conditions of the m rules. The branch cutting works with a minimum plan confidence and a maximum search depth d.

Therefore, the worst case algorithmic complexity of the Meta Tita planner is $O(d^{m\ N\ 2^c})$. The worst case occurs when all the rules are recursive i.e. the rules have the same symbol in the head and in the body.

5.3.3 Meta-plan linearization

The Meta-plan linearization is a direct exploration of the structure of the Meta-plan. Its complexity is $O(n)$ with n the number of nodes in the Meta-plan. The Meta-plan linearization can be merged into the Meta Tita planner improves computer execution time.

5.3.4 Building of plan instance

The building of plan instance is a direct exploration of the plan. Its complexity is $O(n)$ with n the number of nodes in the plan. In the case of plan with negation, a simple heuristic produces instances until a valid instance is found.

5.4 Conclusion

This chapter presented our planning and scheduling algorithm. The algorithm takes as input a set of Titar rules and generates a plan called Meta Titar plan.

We believe, that one of the main interests of this algorithm, in comparison to classical planners, is its capacity to deal with imprecisely located (exogenous and no-exogenous [or endogenous]) events. This makes possible to use imprecise rules as input of the algorithm. Rules extracted by learning algorithms in real world applications have generally some level flexibility/imprecision. One of our motivations during the development of our planning algorithm was the fact that it can be theoretical combined with our learning algorithm. We tried this combination in one of our experiments, and the results are presented in the next chapter as well as the results of several other experiments.

Function SubMetaTitaPlanner(v:a vertex of the recursive plan rp, c:the current minimum confidence)

Data:
A set of Tita rules $\{r_i\}$
A Meta Tita plan rp ▷ The output Meta-plan
A symbolic time sequence I
A set of doable symbols D
A time interval describing when the doable symbols can occur T
An integer N describing the 'confidence discretisation'
Result: The Meta Tita plan rp
begin
    for *All the rules* $r \in r_i$ *such that* $\text{head}_r = \text{symbol}_v$ *and* $\text{conf}_r \geq c$ **do**
        for $j \in [0, N-1]$ **do**
            set $c' = c + j\frac{\text{conf}_r - c}{N-1}$ ▷ c' is the minimum confidence of the (sub)plan that will be created for this node.
            set $\{c_i\}$ to be the set of type 1 conditions of r such that $\text{symbol}_{c_i} \notin D$ ▷ $\{c_i\}$ is the set of conditions for which we need to search a (sub)plan for (the condition over exogenous events).
            for *All the subsets* C *of* $\{c_i\}$
            Note: If we select only the subsets of cardinality lower or equal to one, the Meta-plan will be weak linear. **do**
                compute the plan p with TitaPlanner, the rule r, the request interval Acst_v, the minimum confidence c' and the doable symbols D' with D' the symbols of C.
                ▷ Each type 1 condition of C is associated with a vertex of p. These vertices are 'caused' vertices
                if *the plan* p *is valid* **then**
                    add a 'solution' vertex v' to rp labelled with the plan p
                    add an edge $e := v \to v'$ to rp
                    set $c''' = \frac{\text{number of 'caused' vertices of } p}{\text{conf}_p}$; ▷ c''' is the remaining amount of confidence. The remaining confidence is spread equally on the (sub) plans. Solutions with different distributions can be considered. However, the next sections mainly focus on weak linear plan, and weak linear plans have at most one 'caused' vertex. Therefore, we are not considering alternative confidence distribution.
                    for *all the 'caused' vertices* v'' *of* p **do**
                        add a 'problem' vertex v''' to rp with $\text{Acst}_{v'''} = \text{Acst}_{v''}$, $\text{symbol}_{v'''} = \text{symbol}_{v''}$ and labelled with v''
                        call SubMetaTitaPlanner(v''',c'')

6 Experimental evaluation

Chapter Contents

In order to understand, evaluate and compare our algorithms to the related work, we have performed several experimental evaluations. This first section presents the experiments on temporal learning. The second section details the experiments on temporal planning and scheduling. The input data used in these experiments are either real world datasets or computer generated/simulated datasets.

Generated (or simulated) datasets make possible to evaluate algorithms with a large variety of controlled situations. Since we generated these datasets, we know the temporal patterns and level of noise they contain. Therefore, we can evaluate the responses of techniques to different types of noise, different level of noise, complexity of the dataset, etc.

Real world datasets are also important. They give us the guarantee that our algorithms work on real applications.

6.1 Temporal learning

6.1.1 Evaluation measures

In this thesis, the result of a temporal learning process returns a set of temporal patterns. Several measures exist to evaluate 'how good/interesting' is a pattern, a set of patterns, or a pattern in a set of patterns. A lot a different measures have been developed and studied for general (non temporal) data-mining patterns (Geng and Hamilton, 2006). In the case of temporal learning, a part of these measures are not relevant. This section presents the different measures for temporal rules used in this work.

6.1.1.1 Simple metrics

The simple metrics are the *confidence* (conf), the *support* (supp), the *prediction range* (range) and the *standard deviation* (std). These four measures are defined in section 3.5.

When a reference rule (or ground truth rule) r_{ref} is available, we can compute the ratio between the metrics of a rule to evaluate r and the metrics of the reference rule i.e. $\frac{\text{conf}_r}{\text{conf}_{r_{ref}}}$, $\frac{\text{supp}_r}{\text{supp}_{r_{ref}}}$, $\frac{\text{range}_r}{\text{range}_{r_{ref}}}$ and $\frac{\text{std}_r}{\text{std}_{r_{ref}}}$. Since reference rules are generally not known, such ratios can't be computed.

6.1.1.2 Normalized confidence and normalized support

The normalized metrics are used to compare the confidence and the support of a set of rules with various prediction ranges. The prediction ranges of the rules $\{r_i\}$ are reduced to be equal to the prediction ranges of a reference rule r_{ref} while their confidence is maximized.

The result is the rule $\{r'_i\}$ with $\text{range}(r'_i) = \text{range}(r_{ref})$. The *normalized confidence* of r_i is the confidence of r'_i. The *normalized support* of r_i is the support of r'_i. These metrics are easy to read. However, we need a reference prediction range to compute them, and such reference prediction range is not always available.

6.1.1.3 Confidence and support temporal lift

The *lift* is a common metric for classical/non-temporal association rules. The lift compares the performance of a rule against the performance of the randomness. The performance can either be the confidence or the support of the rule. Suppose a non temporal rule $A \to B$. The lift of confidence of this rule is $\frac{P(B|A)}{P(B)}$ with:

- $P(B|A)$ is the probability of having the head of the rule true when the body of the rule is also true i.e. the confidence of the rule.
- $P(B)$ is the probability of having the head of the rule true.

With the same notation, the lift of support of this rule is $\frac{P(A|B)}{P(A)}$.

We extend the measure of lift to the temporal association rules. Suppose a temporal rule $A \to B\ [r]$ where A is the body of the rule, B is the head of the rule and $r \in \mathbb{R}$ is the prediction range of the rule i.e. the average size of the prediction interval. In the case where the events A and B are independent, the (independent) confidence and the (independent) support of the rule are given by the equations (6.1) and (6.2).

$$\text{conf}_{indep} = 1 - (1 - \frac{r}{d})^{|B|} \qquad \text{Independent confidence} \qquad (6.1)$$

$$\text{supp}_{indep} = 1 - (1 - \frac{r}{d})^{|A|} \qquad \text{Independent support} \qquad (6.2)$$

Where $|B|$ is the number of time the rule's head event occurs in the dataset, $|A|$ is the number of time, the rule is triggered i.e. the number of predictions, and d is the duration of the dataset.

With this notation, the *temporal confidence lift* is $\frac{\text{conf}}{\text{conf}_{indep}}$ and the *temporal support lift* is $\frac{\text{supp}}{\text{supp}_{indep}}$.

By opposition to non temporal lifts (i.e. classical lift), temporal lifts cannot be use directly because they will give very high score to rules with small prediction ranges and small support, in comparison to rules with higher prediction range and support.

In order to tack this problem and obtain a measure to rank the rules, we define the *score* of a rule to be the product of the temporal confidence lift, and the temporal support.

$$score(r) = \frac{\text{conf}}{\text{conf}_{indep}} \cdot \text{supp} \qquad (6.3)$$

By opposition to temporal lift, score gives credit to rules with high confidence, high support and low prediction range. Score is a measure without unit.

Experiments show that the score is sometime giving too much importance to prediction range (rules with highest score have confidence that is too low to be useful). If needed, the definition of the score can be slightly modified to give more a less importance to confidences, support and prediction ranges of the rules (for example, by adding factor parameters to the score equation i.e. $\frac{\text{conf}^{\alpha}}{\text{conf}_{indep}} \cdot \text{supp}^{\beta}$). This alternative score does not have the semantic of $lift \times support$, but experiments show that it is also a useful tool to analyze and understand rules.

Given a dataset composed of two of events A and B with the given correlation: If an event A occurs at time t, then an event B occurs at time $t + \mathbb{U}_{T_1,T_2}$, with $\mathbb{U}$ the uniform distribution. Suppose the family of rule $\{A_t \to B[t+X, t+Y] | \forall X, Y \text{ with } 0 < Y - X \leq M\}$, i.e. if an event A occurs at time t, then an event B occurs between $t+X$ and $t+Y$. The limit M is used in the next proof. Each of the rules of this family is associated with a score. If the range of the rule is small in comparison of the range of the dataset i.e. $M << d$, and if the dataset is sparse i.e. $|B| << \frac{d}{M}$, then the rule $A_t \to B[t+T_1, t+T_2]$ is the rule in the dataset with the highest score.

Proof 6.1.1 *The confidence, the independent confidence and the support of the rule $A_t \to B[t+X, t+Y]$ are :*

$$
\begin{aligned}
\text{conf} &= \max\left(0, \frac{\min(Y,T_2) - \max(X,T_1)}{T_2 - T_1}\right) && \textit{Confidence} \\
\text{supp} &= \text{conf} && \textit{Support} \\
\text{conf}_{indep} &= 1 - (1 - \frac{Y-X}{d})^{|B|} && \textit{Independent confidence} \\
&= \frac{(Y-X)|B|}{d} + O((\frac{(Y-X)}{d})^2)) && \textit{Series expansion at } \frac{Y-X}{d} = 0
\end{aligned}
$$

We approximate the independent confidence.

$$
\text{conf}_{indep} \approx \frac{(Y-X)|B|}{d}
$$

We are looking for X and Y such that the score is maximized.

$$\begin{aligned}
&X \textit{ and } Y \textit{ maximize} && \text{score} \\
\Leftrightarrow &X \textit{ and } Y \textit{ maximize} && \frac{\left(\max\left(0, \frac{\min(Y,T_2)-\max(X,T_1)}{T_2-T_1}\right)\right)^2}{1-(1-\frac{r}{d})^{|B|}} \\
\Leftrightarrow &X \textit{ and } Y \textit{ maximize} && \frac{(\min(Y,T_2)-\max(X,T_1))^2}{1-(1-\frac{r}{d})^{|B|}} \\
\Leftrightarrow &X \textit{ and } Y \textit{ maximize} && \frac{(\min(Y,T_2)-\max(X,T_1))^2}{Y-X} \\
\Leftrightarrow &X = T_1 \textit{ and } Y = T_2
\end{aligned}$$

Therefore, the rule $A_t \to B[t+T_1, t+T_2]$ *has the highest score.*

Experiments show that if we add noise to the dataset, the rule $A_t \to B[t+10, t+20]$ is still the rule with the highest score.

For readability and computational convenience, we compute the log of the score:

$$\begin{aligned}
\log score =& \frac{\text{conf}}{\text{conf}_{indep}} \cdot \text{supp} \\
=& \log \text{conf} + \log \text{supp} - \log \text{conf}_{indep}
\end{aligned}$$

6.1.1.4 Rule gain

The *gain* of a rule is a measure of 'how much' a rule will produces/wins/prevents/etc. in a given process/use. For example, in Forex (foreign exchange market) trading, the gain of a rule can be the expected value and the standard deviation of the monetary gain.

6.1.1.5 Global Support Map

A *Global Support Map* (GSMap) is a tool used to evaluate a set of predictors (such as association rules) on a temporal dataset. A support map $m : [0,1] \times [0,\infty] \to [0,1]$ is a function mapping a *minimum confidence* (probability of the predictions to be true) and a *maximum prediction range* (or size of prediction's window) to a *global support*. The *global support* is the percentage of events predicted by rules with a confidence greater or equal than the minimum confidence, and a prediction range lower or equal than the maximum prediction range. The main advantage of GSMaps over classical metrics (e.g. confidence or support) is to not require an arbitrary fixed window size as it is usually done in related literature. A GSMap can be represented as a two dimensional picture. Two GSMaps can also be compared together.

The figure 6.1 shows a GSMap with uniform and logarithmic range axis. The figure 6.2 shows the subtraction of two GSMaps. This type of figure helps the comparison of sets of rules.

In order to compute the GSMap, the confidences of the rules to evaluate need to be estimated. We need to estimate the confidence of the rules to compute GSMaps. This estimation is an empirical probability; therefore, the confidence of rules with a low number of uses cannot be accurately estimated. Therefore, GSMaps are sensible to rules with a low number of uses. In consequence, rules with low number of uses are discarded when computing GSMaps.

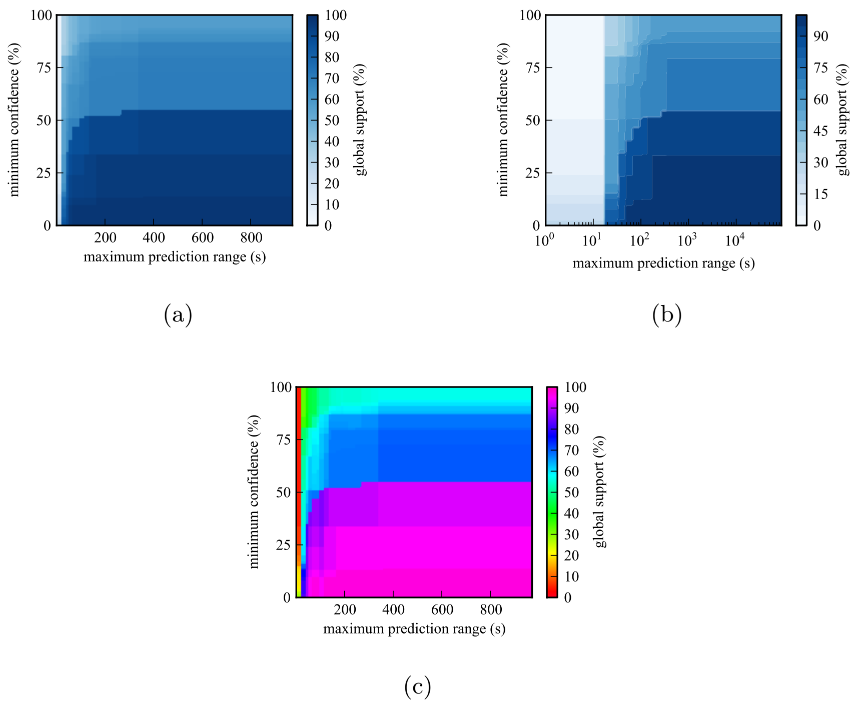


Figure 6.1: Three graphical representations of the same GSMap. (a) uses a uniform scale for the range axis. (b) uses a logarithmic scale for the range axis. (c) uses a uniform scale for the range axis with a multi-color map. It is easy to read global tendencies when using a mono-color color map (see (a) and (b)). It is convenient to read punctual support value when using a multi-color map (see (c)).

The GSMap measure is interesting because it describes sets of rules (when other measures describe only one rule), and because it does not need to have

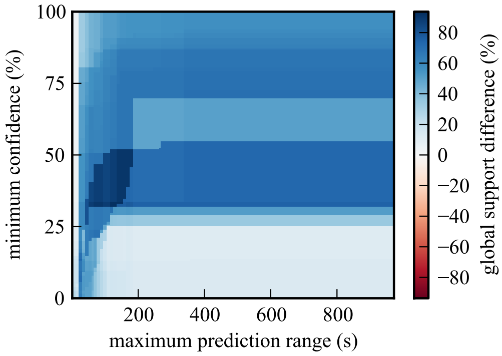


Figure 6.2: Difference of two GSMaps g_1 and g_2. The higher is the value, the better is the GSMap g_1 in comparison to g_2. In this example, the set of rules evaluated by g_1 it globally better than the set of rules evaluated by g_2. In addition, the difference is especially important for confidences in the interval $[30\%, 50\%]$ and for prediction ranges in the interval $[50, 170]$.

references rules.

Property 6.1.2 *Given a dataset d, a finite number of rules $\{r_i\}$, and the support map m of $\{r_i\}$ on d. Let r be the largest prediction range of the rules $\{r_i\}$ i.e. $r = \max_i(\text{range}_{r_i})$. The following property holds $\forall c, \forall r' \geq r, m(c, r') = m(c, r)$.*

6.1.2 Computer generated dataset

Dataset generation

This section presents our symbolic time sequence dataset. We have constructed this dataset because of the lack of good symbolic time sequence datasets available to evaluate and compare symbolic time sequence learning algorithms. Most available temporal datasets are either too small to be used or built from a pre-processed time series. The conversion from time series to symbolic time sequences is a complex problem with no perfect solution. In this last case, existing solutions bias the input data of symbolic time sequence learning algorithms in such way that it becomes difficult to compare two techniques.

Moreover, as far as we know, there is currently no available symbolic time sequence dataset associated with a 'ground truth'. For a symbolic time sequence, a ground truth is not a set of the actual events to predict, but a set of best possible predictions to do. With such a ground truth, it is possible to evaluate how good an algorithm is, independently from any other algorithm, and even with patterns with low confidence/support. If the dataset contains a large enough number of different patterns with various complexities, it becomes possible to analyze precisely the robustness, completeness and power of expression of any algorithm.

Based on these observations we decided to build a dataset, with a large number of different patterns with various complexities (the patterns are documented), with various levels and types of noises, with a record of the best possible predictions, and with a tool for automatically evaluate the various metrics of a given set of predictions.

This computer generated dataset provides a powerful tool for temporal data mining and temporal learning algorithms.

The dataset is divided into 100 *parts*. Each part is constructed in this following way: The algorithm randomly generates a temporal association rule (structure and metrics) called *reference rule*. The confidence is chosen between 60% and 90%, the support is chosen between 40% and 90%, the number of conditions is chosen between 1 and 4, the number of uses of the pattern is chosen between 1000 and 50000. The ratio of noisy events/useful events is selected between 0% and 1000%. Several types of noise are considered: A noise non-correlated with the pattern, as well as noise with different types of correlation with the pattern. Next, the algorithm generates a dataset's part with the selected rule and parameters. This process is a heuristic, and the metrics of the rule need to be re-evaluated on the generated part. The generated part contains an infinite number of temporal rules (e.g. a dataset where the reference rule is $A \dashrightarrow B \longrightarrow C$ also contains the rule $B \longrightarrow C$ with a lower confidence). However, the generative algorithm guaranties that the *reference rule* is the rule with the highest score according to Equ. 6.3.

Every part is divided into two sub-parts in order to perform cross validation. In addition, the dataset contains (for every part) a list of reference predictions i.e. a 'ground truth'. These predictions are (with a small error rate) the best predictions that can be expected to do. We use the Mersenne Twister pseudo-random number generator as source of randomness for the patterns and additional noises. Fig. 6.3 shows three examples of patterns of the dataset.

The fig. 6.4 shows a sample of the html document which describes each part of the dataset.

The dataset is available online (Guillame-Bert, 2011). A Python script is provided to automatically compare user predictions (the predictions from the set of rules you want to evaluate) with reference predictions. Fig. 6.5 shows a screen-shot of the script's interface. The script computes the confidence, support and temporal precision of the user and reference predictions. In addition, it computes ratio of measure and normalized measures (see defini-

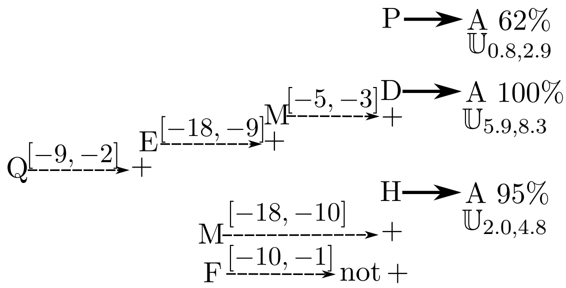


Figure 6.3: Three example of patterns of the computer generated dataset.

tion is previous section). It also computes various analyses of the predictions such as relations between predictions, confidence and several measure of the complexity of the dataset. The output report is also presented as an Html page.

Learning evaluation

We apply our algorithm for 60 seconds on each part of the dataset. We applied the Minepi (Mannila *et al.*, 1997) until it reaches the fourth loop. The time bound is fixed to 100s and the window sizes parameter is fixed to all integer between 1 and 100 i.e. $1, 2, 3, \cdots, 99, 100$. The algorithm is run one time for each of these values and the results are combined.

We applied the Face (Dousson and Duong, 1999) algorithm until it stops. On every part of the dataset, the FACE is run ten times with the pMin parameter fixed between 0.1 and 1 i.e. $pMin \in \{0.1, 0.2, 0.3, \cdots, 0.9, 1\}$. The algorithm is run one time for each of these values and the results are combined.

The running time of our algorithm took a couple of minutes on a multi-threaded machine. Face and Minepi algorithms took several days.

Each generated rule is scored according to score defined in section 6.1.1.3. Since each part of the dataset is based on only one temporal rule (called a reference rule), and because the temporal distribution of this rule is always a uniform distribution, the reference rule is guaranteed to be the rule with the highest score in a given part of the dataset. Therefore, for each algorithm, we only keep the best learned rule (according to the score) from each part of the dataset.

The cross validation is performed in the following way: (a) We extract rules for each part of the dataset (b) We select the rule with the highest score. (c) Finally, we evaluate the rule on another part of the dataset that contains the same pattern. We know the exact patterns (reference rules) for each parts of the dataset. The ranges of rule predictions are normalized to the ranges of the reference predictions in order to compare supports and confidences

Sub part name : part_59

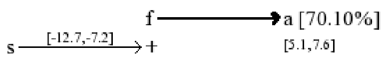
Events
Number of events to predict : 1908
Time range of dataset : [0 , 10000]
Density of events to predict : 0.1908
Number of different symbols : 20

Patterns
Number of patterns : 1
Maximum future of prediction : 10.0000
Temporal precision : 3.0000
Maximum condition window distance : 20.0000
Maximum condition window size : 10.0000
Number of conditions : 2

Noise
Maximum ratio number of noise events/number of events to predict : 0.1000
Symbols noises ratio: a:0.0000, b:0.0214, c:0.0117, d:0.0338, e:0.0952, f:0.0929, g:0.0305, h:0.0535, i:0.0545, j:0.0380, k:0.0019, l:0.0822, m:0.0004, n:0.0185, o:0.0880, p:0.0115, q:0.0443, r:0.0795, s:0.0838, t:0.0592
Probability of type 0 noise : 50.00%
Probability of type 1 noise : 50.00%
Probability of type 2 noise : 0.00%
Type 1 and 2 noise probability parameter : 30.00%
Rule random signature : 0 , 1

Average of all cross-validations
Total number of events : 6427
Symbols (+number of occurrences) : a:1621, b:24, c:13, d:39, e:111, f:1863, g:35, h:62, i:63, j:44, k:1, l:96, m:0, n:21, o:102, p:13, q:51, r:93, s:2106, t:69

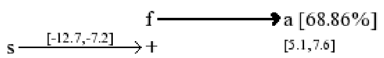
Patterns :



confidence : 70.10%
support : 60.31%
number of use : 1630
standard deviation : 0.7161
number of conditions : 2

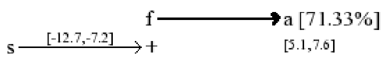
Cross-validation 1 (part_59_1)
Total number of events : 5444
Symbols (+number of occurrences) : a:1343, b:21, c:11, d:33, e:95, f:1585, g:30, h:53, i:54, j:38, k:1, l:82, m:0, n:18, o:87, p:11, q:44, r:79, s:1800, t:59

Patterns :



confidence : 68.86%
support : 60.39%
number of use : 1352
standard deviation : 0.7161
number of conditions : 2

Cross-validation 2 (part_59_2)
Total number of events : 7422
Patterns :



confidence : 71.33%
support : 60.24%
number of use : 1908
standard deviation : 0.7161
number of conditions : 2

Figure 6.4: Sample of the html description provided for the part 59 of the dataset.

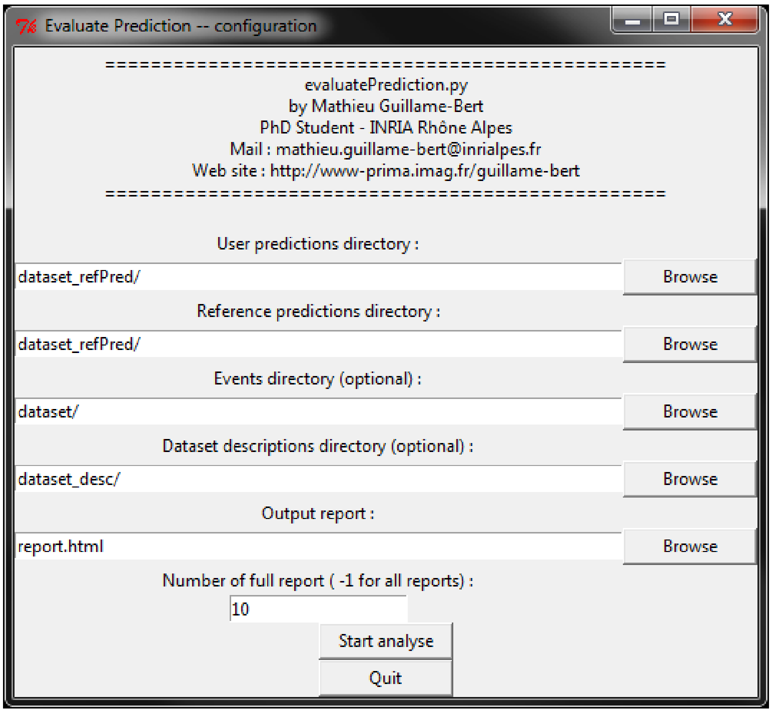


Figure 6.5: Screen-shot of the comparison script interface.

(called normalized support and normalized confidences).

Table 6.1 shows the average ratio between the normalized confidence, normalized support, average range and number of predictions of the learned rules and the confidence, support, average range and number of predictions (n.o.p.) of the reference rules.

Table 6.1: Evaluation of Titarl, MINEPI and FACE algorithms on the computer generated dataset. The table shows the average of the ratio measure/reference measure for the normalized confidence, normalized support, average range and number of predictions.

algorithm	**Normalized confidence**	**Normalized support**	**av. range /reference av. range**	**n.o.p. /reference n.o.p.**
Face	0.49	0.71	16.29	2.21
Minepi	0.51	0.78	3.75	1.43
Titarl	**0.90**	**0.99**	**2.09**	**1.11**
Reference	1.00	1.00	1.00	1.00

Minepi gives a slightly better normalized confidence and normalized support than Face. We also observe that our algorithm outperforms Minepi and Face on normalized confidence and normalized support. Minepi prediction ranges are far better than Face prediction range. Our algorithm outperforms Minepi and Face on prediction range.

6.1.3 Home dataset

Dataset presentation

The 'Home activities dataset' created by Tim van Kasteren et al. (van Kasteren *et al.*, 2008b) is a record of 28 days of sensor data and activity annotations about one person performing activities within an apartment. An apartment is equipped with sensors on doors, cupboard, fridge, freezer, etc. Activities of the person are annotated (prepare breakfast, dinner, having a drink, toileting, sleeping, leaving the house, etc.). The dataset is divided into two categories: sensor events (start_sensor_fridge, end_sensor_fridge, start_sensor_frontdoor, etc.) and change of activities (start_action_get_drink, end_action_get_drink, start_action_prepare_dinner, etc.). S is composed of 24 states describing the time of the day (it_is_1am, it_is_2am, it_is_3am, etc.). In this experiment, the algorithm is applied in order to predict activity change events according to sensor events and states describing the time. E contains 42 types of events and 2904 occurrences of events. We expect to observe patterns in the person's activity.

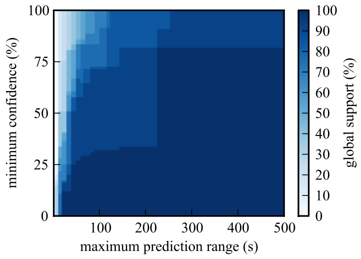
Remark on learned rules

We apply our algorithm to the dataset. The algorithm is executed for 120 seconds for every type of event to predict. The confidence, support and prediction range of rules depend of the kind of activity to predict.

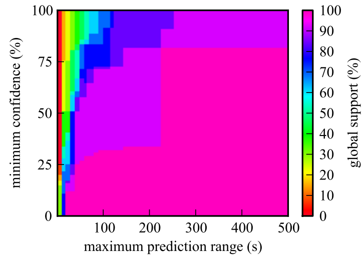
While actions such as *sleeping* or *using the bathroom* are fairly well explained (relatively high confidence, support and low prediction range – see fig. 6.6), actions such as *getting a drink* or *leaving the house* cannot be explained with high confidence, high support and low prediction range at the same time. Fig. 6.6 shows the global support maps of different types of actions for our technique.

As an example, fig. 6.6 shows that we can predict all of the uses of bathroom with at least 58% confidence and a temporal precision of less than 100 seconds, or with 100% confidence and a temporal precision range of 400 seconds.

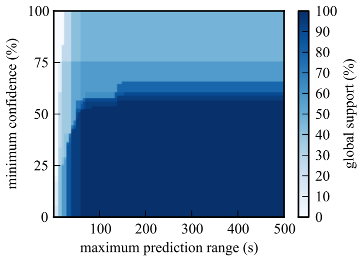
The algorithm extracts rules of different complexity. For example, the direct implication between the use of the toilet flush and the action of using toilets (fig. 6.7.a - confidence: 92%, support: 82%, prediction range: 220 seconds and standard deviation: 46 seconds), or the use of the front door



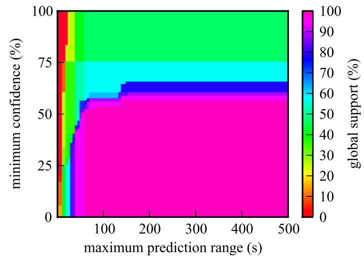
(a) Going to bed



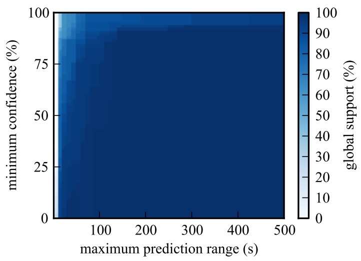
(b) Going to bed (multi-color map)



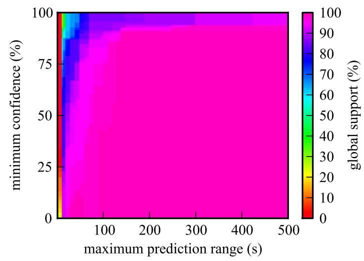
(c) Leaving house



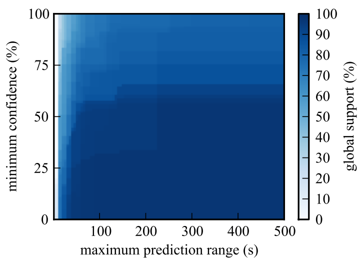
(d) Leaving house (multi-color map)



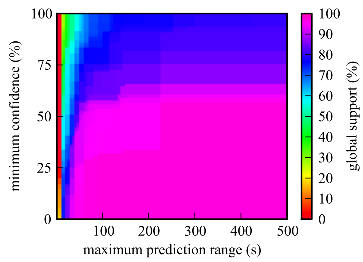
(e) Use of bathroom



(f) Use of bathroom (multi-color map)



(g) Average of all actions



(h) Average of all actions (multi-color map)

Figure 6.6: Global support maps for the 'Home activities dataset'

and the action of leaving the flat (fig. 6.7.b - confidence: 51%, support: 100%, prediction range: 50 seconds and standard deviation: 11 seconds).

More complex rules are based on several conditions. As an example, the action of going to bed is predicted with a 100% confidence, 66% of the time by the rule represented on fig. 6.7.d. Some other examples of complex rules are presented in fig. 6.7.

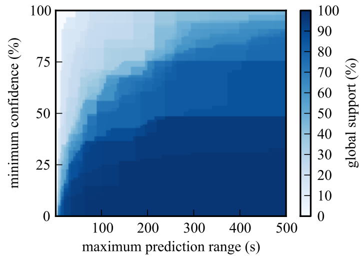
GSMap comparison with other techniques

The dataset is divided into two parts of equal sizes. We apply our algorithm, the Face algorithm and the Minepi algorithm to the first part, and evaluate the results with the second part. Our algorithm is applied for 120 seconds. Face and Minepi are applied until they stop. Since the number of rules generated by the Minepi algorithm is too large to be analysed, we only keep the 10000 rules with the highest scores.

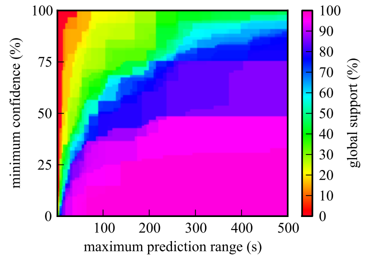
The fig. 6.8 shows the average GSMaps of the rules learned with the Face, Minepi and our algorithm. The figure 6.9 shows the subtraction of these GSMaps. Table 6.2 shows the statistics of the subtraction of these GSMaps.

We observe (fig. 6.9 and table 6.2) that globally, Face is more efficient than Minepi: With the same confidence and range constraints, Face explains an average of 41% more of events than Minepi. Titarl (our technique) outperforms both Face and Minepi algorithms: With the same confidence and range constraints, Titarl explains an average of 10% more of events than Face and 51% more of events than Minepi.

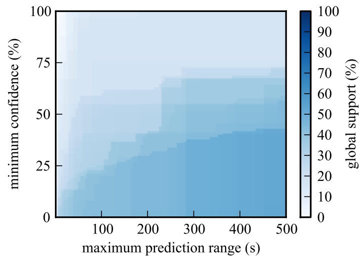
Figure 6.7: Example of learned rules from the HomeDataSet (experiment 2)



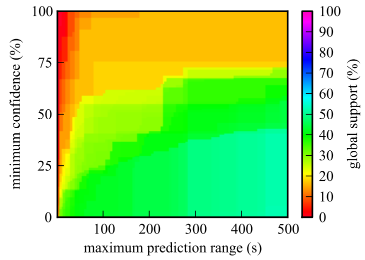
(a) Face algorithm



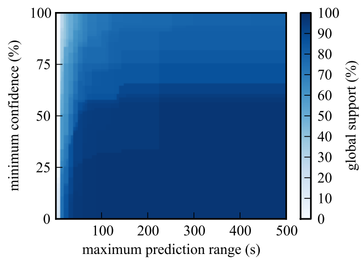
(b) Face algorithm (multi-color map)



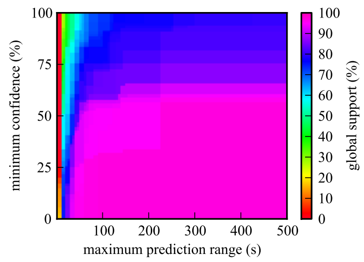
(c) Minepi algorithm



(d) Minepi algorithm (multi-color map)



(e) Titarl



(f) Titarl (multi-color map)

Figure 6.8: Global support maps for the 'Home activities dataset'

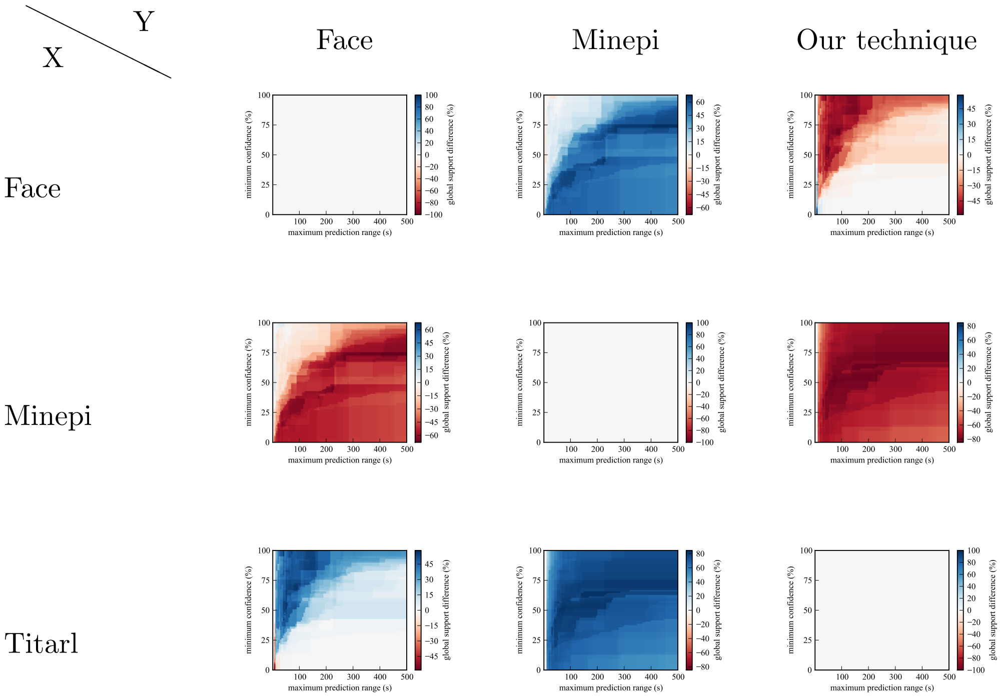


Figure 6.9: Subtraction of the GSMaps of the Face, Minepi and our algorithm. Each cell represents the GSMap of the algorithm X (written on left column) minus the GSMap of the algorithm Y (written on the top line). A positive support value z means that given the same confidence and range constraints, the algorithm X predicts more events than the algorithm Y. The given value is a difference (and not a ratio) e.g. if one technique as a support of 50% and the other technique as a support of 100%, then the difference is 50%.

Remark 6.1.3 *The given percentage represents means of differences of global support. If the techniques A and B have respectively a global support of* 50% *and* 100%*, the B explains* +50% *of events than A.*

Fig. 6.10 shows the evolution of the average of the GSMap for the Titarl algorithm according to training time. Most of the actions can be explained with the *trivial rules* learned during the first loop (spike at time = 2). Next, the rules are refined and the metrics are improved until they stabilize.

Best rule comparison with other techniques

For this comparison, we use the set of learned rules from the previous comparison. Next, each generated rule is scored according to score defined in

Table 6.2: Statistics of the subtraction of the GSMaps display in fig. 6.9. The 'strict pos.' is the percentage of the global support strictly greater than 0. The GSMaps are computed with a prediction range between 0s an 500s with 100×100 cells.

Y / X	**Face**	**Minepi**	**Titarl** (Our technique)
Face		mean : 41.65% median : 47.33% std : 17.51% strict pos. : 97.70%	mean : -15.83% median : -9.82% std : 17.68% strict pos. : 26.84%
Minepi	mean : -41.65% median : -47.33% std : 17.51% strict pos. : 1.72%		mean : -57.42% median : -59.59% std : 12.20% strict pos. : 1.78%
Titarl (Our technique)	mean : 15.83% median : 9.82% std : 17.68% strict pos. : 73.16%	mean : 57.42% median : 59.59% std : 12.20% strict pos. : 98.22%	

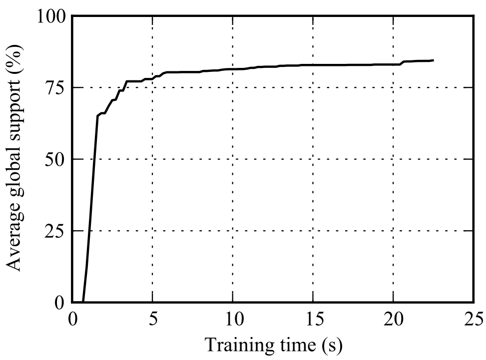


Figure 6.10: Evolution of the GSMap average with the training time i.e. the curve shows the average (on the different actions) of the average (for a given GSMap) of the GSMaps according to the training time.

section 6.1.1.3. For each algorithm and each type of event to predict, we only keep the rule with the highest score (called the 'top rule').

Table 6.3 shows the 'log score/confidence/support/average range' of the top rule for each algorithm and each type of event to predict. We only display result for actions that occur at least 10 times in both the learning and evaluation dataset.

As expected (see section 6.1.1.3), the top rules have relatively low support and prediction range. Minepi gives a higher confidence than Face (57% vs. 36%) and more precise range ($6.25s$ vs. $18s$) but Face gives higher support (60% vs. 11%) than Minepi. Our (complete) algorithm gives a higher confidence than Face and Minepi.

Table 6.3: Evaluation of our algorithm, Minepi and Face algorithms on the Home activities dataset. Each cell contains the log score, the confidence, the support and the average prediction range of the best rule (rule with the highest score). Example: The best Face's rule for predicting going to bed' has a *log* score of 6.02, a confidence of 30%, a support of 54% and a prediction range of 30 seconds.

Action	Face	Minepi	Titarl
go to bed	6.02 30% 54% 30	7.47 75% 15% 5	6.42 100% 8% 9.50
leave house	5.84 25% 79% 30	5.69 55% 5% 5	6.05 75% 11% 9.63
prepare Breakfast	8.01 70% 50% 10	7.05 67% 10% 5	8.15 100% 40% 10.0
use toilet	6.53 20% 44% 2	4.25 32% 14% 10	6.38 76% 48% 9.59
Average	36% 60% 18	57% 11% 6.25	88% 27% 9.68

State comparisons

In the previous evaluation, we were focused on predicting events describing beginning and ending of actions or states. For examples, events such as 'going to bed' and 'leaving bed' define the state of 'sleeping'. With such events and for every time point, we can compute the probability for the person to be in a given state. The fig. 6.11 shows example of conversion from events to states.

In this experiment, we use the *beginning* and *ending* predictions of actions to compute the probability for the person to be in any of the possible states. For example, if at a given time, the probability for the person to sleep is 100%, and if we predict that the person wakes up with 70% chance, then the probability for the person to sleep becomes 30%, and the probability for the person to be awake becomes 70%. We evaluate our predicted states with the cross validation protocol.

In real applications, the cost of a temporal imprecision is generally proportional to the duration of the state to predict. For example, having a 1 minute error on predicting a 2 minute state is worst than having a 1 minute error

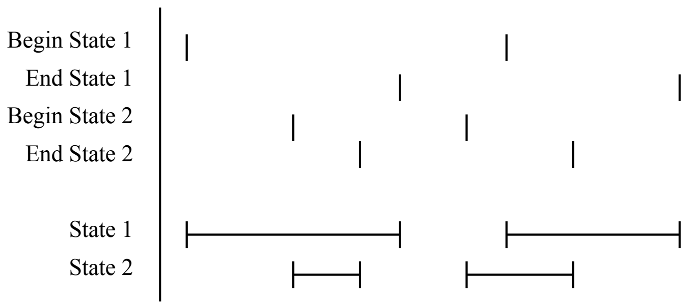


Figure 6.11: Example of conversion events ↔ states

on predicting a 2 hours state. In this case, evaluating the states instead of the beginning and ending events is meaningful. In the home dataset, actions with long durations are 'sleeping' and 'being out'.

Table 6.4 shows the precision and recall for the prediction of sleeping and being out.

Table 6.4: Precision and Recall of the prediction of the person long duration states

State	Precision	Recall
Sleeping	97.97%	99.03%
Being out	97.23%	96.74%

Since the 'going to bed' and 'leaving bed' events were well predicted (see fig. 6.6), it is not a surprise that the precision and recall of sleeping are high. The interesting fact is that, even if the 'leaving home' and 'entering home' events were not as well predicted as 'going to bed' and 'leaving bed' (see fig. 6.6), the precision and recall of 'being out' are equally high.

Table 6.3 shows that the *score* is favoring rule with low prediction range (all the top rules have a prediction range lower than 30s).

6.1.4 In office people motion prediction

Dataset presentation

The MERL Motion Detector Dataset created by Chris Wren et al. (Wren *et al.*, 2007) is a one year record of over 200 infra-red proximity sensors placed in the MERL (Mitsubishi Electric Research Labs) office building. The sensors are ceiling mounted every two meters in the public places of the building (corridors, lobbies and meetings rooms). They are arranged such that there is no overlap into their different fields of view. The fig. 6.12 shows the floor plan of the building. The fig. 6.13 shows the location of the infra-red proximity sensors.

Each time a person walks in front of a sensor, it records a time sampled event $s_n[t]$, where n is the sensor id and t is the current time. In addition to proximity sensor events, the dataset contains information such as the weather, the list of meeting, the fire department visits, etc. In this experiment, we are only using the proximity sensors data. The dataset is divided into several sequences. We are using the first sequence ('0114' sequence). This sequence contains around 4 500 000 time sampled events and 147 types of events.

Figure 6.12: Plan of the building. The observed areas are shaded.

Learning

Our experiment on the dataset is inspired from the following idea: When people are in corridors, they are usually walking (people tend to sit or stand in offices and walk in corridors). Therefore, if a corridor sensor s fires, and if we suppose that the person that triggers the sensor is walking, then one of the neighbor sensors of s will fire in the following seconds. This rule is true if the person is not entering a space which is not cover by sensors (private offices, toilets, etc.). The fig. 6.14 illustrates this idea.

The goal of the experiment is the following one: We learn the temporal correlations between the sensor events, and we expect to be able to construct a topological representation of the building from those rules.

We are using Titarl to learn temporal rules between sensor events. To avoid learning rules that bond temporally and spatially distant sensors, temporal intervals are restrained to be contained in $[-5, 0]$. The minimum confidence and minimum support is respectively fixed to 25% and 5%.

Since the dataset is quite massive, the first loop can take almost a minute. The algorithm is run for 50 seconds once the first loop is achieved.

The topological graph is constructed as following: A node is created and labelled with each of the proximity sensors. For each learned rule r with a

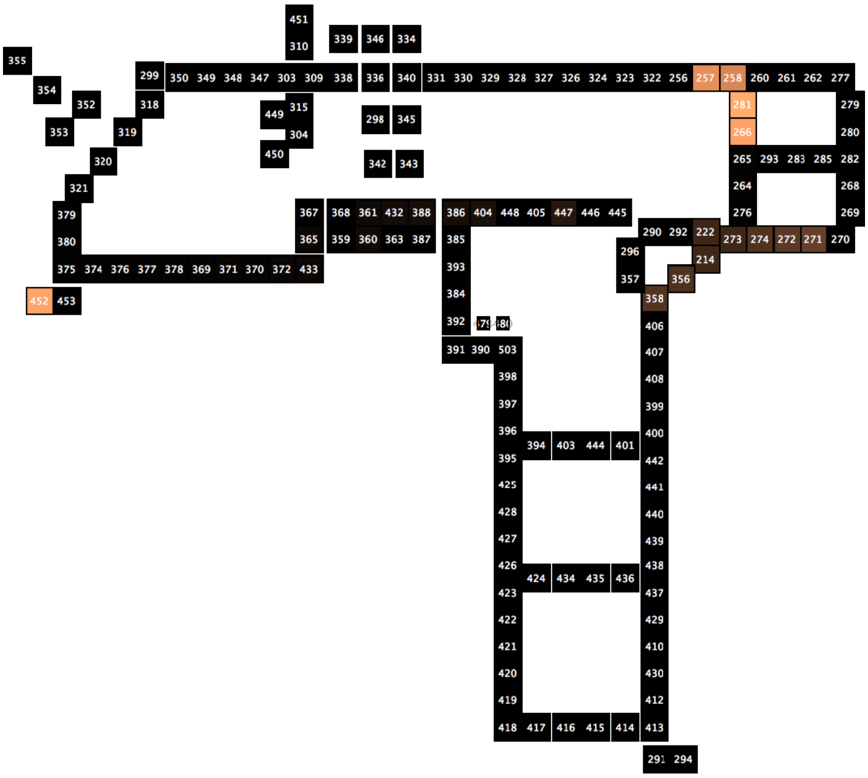


Figure 6.13: Location and id of the infra-red proximity sensors in the building.

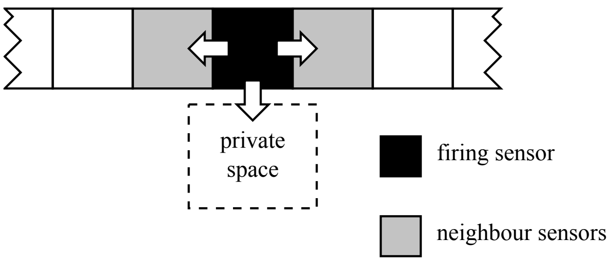


Figure 6.14: Illustration of the correlation between sensors.

head symbol head_r and first condition symbol $\text{symbol}_{\text{cond}_r}$, an edge is added between the nodes head_r and $\text{symbol}_{\text{cond}_r}$.

Result

During the learning process, around 1200 rules are extracted. The topological graph contains 146 nodes and 425 edges. An example of unit rule from the first learning is given in the fig. 6.15. Simple rules tend to correlate symbols of spatially close sensors. An example of complex rule from the second learning is given in the fig. 6.16. Complex rules tend to be 'chains' of aligned sensors. The fig. 6.17 shows the topological graph. The fig. 6.18 shows the topological graph superposed to the building plan. Except for the connections between the sensors 426, 423 and 424, all the structure of the building is captured. The analysis of the dataset shows that rules between 426, 423 and 424 have confidence lower than 25% (the limit chosen in the experiment).

Figure 6.15: Unit rule between two neighbor sensors. The sensors 327 and 326 are neighbours.

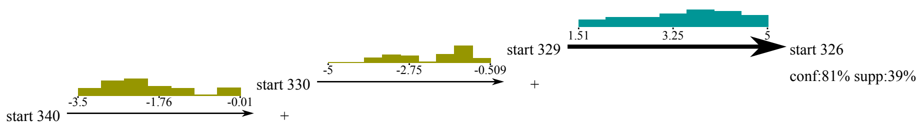


Figure 6.16: Complex rule between aligned sensors. The sensors 340, 330, 329 and 326 are aligned in a corridor. Therefore if a person activates the sensors 340, 330 and 329, there is a high probability for the sensor 326 to be activated.

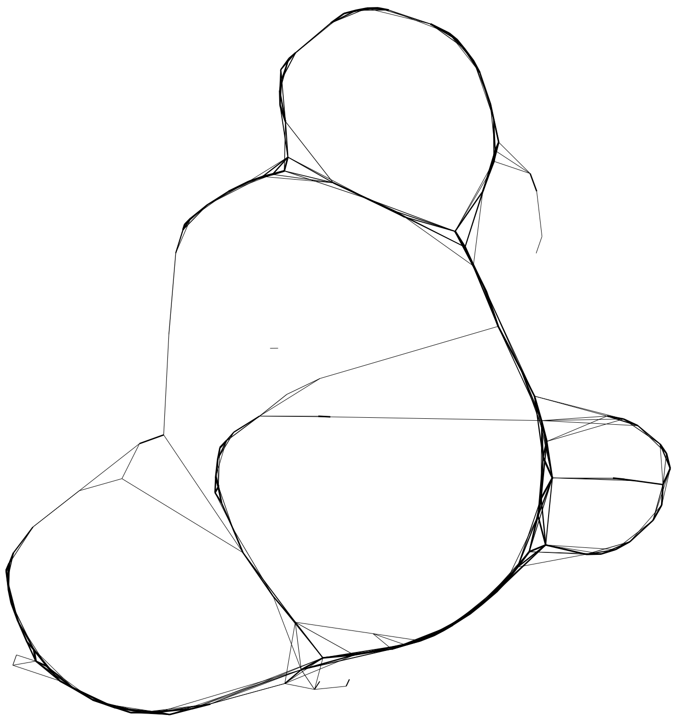


Figure 6.17: Topological graph of the Merl buidling.

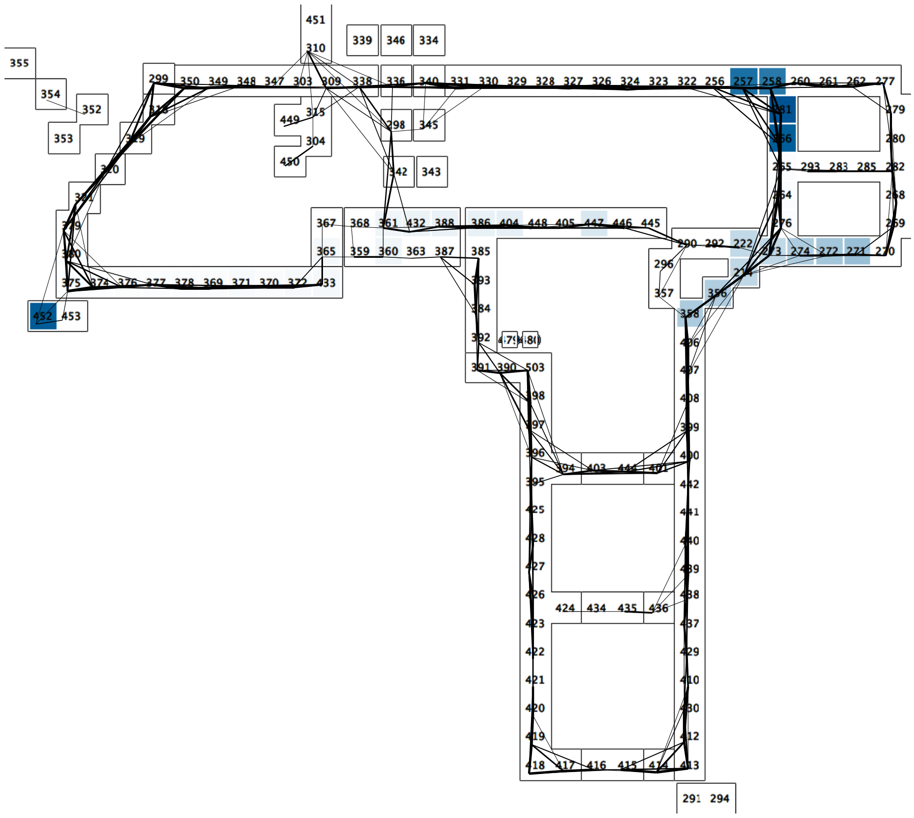


Figure 6.18: Topological graph of the Merl buidling aligned with the plan of the building.

6.1.5 Foreign exchange market trading

Stock exchange markets

A *stock exchange market* is a market used by *traders* to exchange products such as shares, bonds, warrants, index funds, currencies, oil, etc. In these markets, the selling/buying prices (also called ask/bid prices) are automatically calculated according to the supply and demand (the buying price is always higher than the selling price). The difference between the buying price and the selling price is called the *spread*. Since the value of goods are constantly changing, the goal of the brokers is to buy products at low price, and resale them later at higher price. The work of a trader is to forecast the evolution of the product prices.

Traders are either professional workers in financial institutions or individual investors. Professional traders invest their company money, while individual investors use their own resources. Trader's predictions are based on market observation, market knowledge and the world news.

A large panel of theories and tools has been developed to assist the traders. Two families of tools can be distinguished: (a) The *information systems* that produce more or less complex signals and alerts interpretable by the traders and (b) The *automated or robotic traders* that directly send buy/sell orders.

Tools from both families need precise parameter tuning. Therefore in the case of automated trading, a human trader should still be present to adjust the parameters of the automated trader.

Forex

The Foreign exchange market (Forex) is a type of stock exchange market where traders trade currencies (e.g. US Dollars, Euros, Yen, etc.).

This market is the largest market in the world in term of money transaction (approximately $4 trillion are exchanged every day). The market is open all week, it has a high liquidity (the difference between selling and buying price is small), and it has not fixed cost by order. This market is especially sensitive to employment reports, economic forecasts, economic reading and banks activities.

Since currencies are bonded to countries (or groups of countries), values of a currencies are correlated to countries' stability, the countries' resources, and the countries' interest rate. The fluctuation of exchange rate between currencies depends on the value of each currency individually. These fluctuations are generally small, and traders need to use *leverage* to make significative gains.

The *leverage* is the borrowing of a large amount of money, to invest it in a pair of currency, resale the pair, and reimburse the 'loan'. This operation allows traders with small amount of money to make reasonable benefit. Leverage increases the benefit but also increases the losses. The Forex operators (brokers) erquire a guarantee for the loan (the loan is between 50× and 400×). If the lost of an order exceeds the guarantee, the broker closes the order, takes back the loan, and the trader looses the guarantee. This approximate description gives an idea of how leverage works.

The next example shows a detailed Forex buying transaction.

Example 6.1.4 *Consider the ask (asking price) of the EURUSD to be 1.33075 (i.e. 1 Euro gives 1.33075 Dollars). A trader predicts that the EURUSD will increase, therefore he buys 100 000€ for \$133 075. With a leverage of* 100×, *the trader has to give a guarantee of \$1 330.75. Thirty minutes later, the bid of the EURUSD is 1.33198. The trader sell its 100 000 € for \$133 198. He reimburses the \$133 075 loan. At the end of the operation, the trader has won \$111 = \$133 198 - \$133 075.*

The fig. 6.19 shows an example of a (selling) transaction.

Forex exchange rates are updated in real time (up to the micro second for (automated) high frequency trading). In order to be used by humans, the exchange rates are discretized (usual discretization periods are 1mn, 5mn, 15mn, 30mn, 1h and 4h). Most traders use 30mn or 1h periods. The orders of human traders generally have duration of a couple of hours.

Traders use *indicators* (or indexes) to understand the situation of the market (SMA, MACD, etc.). Basic indicators are presented in the next section. Trader can also be assisted by *Information Systems*. Such systems give indications and alerts on the probable evolution of the exchange rates.

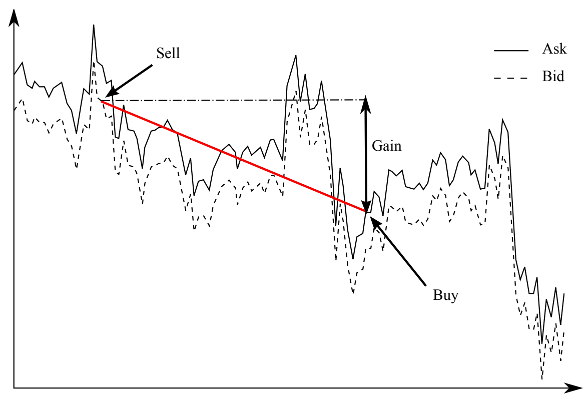


Figure 6.19: Example of Forex transaction with positive profit

On the other hand, fully automated trade systems are computer programs that send buying and selling orders automatically. Such programs are generally based on production rules such as:

```
Buy - When the 30-day moving average (MA) crosses above the 60-day MA
Sell - When the 30-day MA crosses below the 60-day MA
Stop - Maximum loss of 10 units
Target - Target of 10 units
```

The work of the trader is to configure the parameters of the program. Automated trade systems are widely used, especially by large financial institutions (in 2006, 25% of Forex orders where generated by fully automated trade systems). The fact that a trader is using an automated trade system is generally known, but the rules used by the system are kept secret.

High-frequency trading is a special kind of trading made by automated traders. This kind of trading is very fast (orders can be as short as 10 milliseconds) and produces a large amount of order (millions of simultaneous orders). Such a trading system tries to spot increases or decreases of rate before the other actors of the market.

Because of the high update frequency (large dataset), direct outcomes, and complexity of the problem, the Forex is one of the great challenges for machine learning. Many techniques have been applied to Forex trading problem to produce Information Systems and Fully automated traders. These include Neural network (Gan and Ng, 1995; Yao and Tan, 2000), Support vector machines (Cao and Tay, 2001; Kamruzzaman *et al.*, 2003) and Evolutionary algorithms (Myszkowski and Bicz, 2010; Dittmar *et al.*, 1996; Connor and Madden, 2006). The paper (Atsalakis and Valavanis, 2009) is a summary and analysis of more than one hundred articles dealing with Forex forecasting with Neural Networks.

In this experiment, we designed an automated trade system based on the Titarl algorithm. The experiment has two goals. The first goal is to evaluate the capacity of the Titar leaner to deal with a large dataset (we use a 3 years record of the EURUSD (Euro/US Dollars) exchange rate with a 30 minute sample period). The second goal is to evaluate the 'profit' and the 'risk' of the automated trade system. The next section presents the experiment setup, report and discusses the results.

6.1.5.1 Experiment setup

For this experiment, we use a record of three years of the EURUSD exchange rate sampled approximately every minute (from June 5 2008 to June 5 2011). From the dataset, we extract 'trends' (see definition below). We compute three basic indicators (MACD(12,26,9), SMMA(14) and Awesome – see definitions below) with a 30 minute window size. The signals of the indicators are discretized with two types of states (X-is-greater-than-Y and X-is-lower-than-Y) and two types of events (X-becomes-greater-than-Y and X-becomes-lower-than-Y), where X is an indicator and Y is a fixed value. Next, the record is split into three 1 year long parts. The first part of the dataset is used for the learning (June 5 2008 – June 5 2009), the second part is used for the over training analysis (June 5 2009 – June 5 2010), and the last part is used for evaluation (June 5 2010 – June 5 2011). We use the Titarl algorithm to learn and predict the future trends according to the discretized signals.

Definition 6.1.5 *A* Moving Average *(MA) is a* finite impulse response filter *(i.e. the response is computed from a finite set of sample) used to smooth out small fluctuations. It is computed as an average of the last N values of a signal (or from N values sampled with a window size in the case of continuous signal). By convention, MA(N) is a moving average based on the last N values.*

The Simple Moving Average *(SMA) is a moving average where the last N values have the same weight. If v_i represent the i^{th} value. The SMA at time i is $SMA_i = \frac{v_i + v_{i-1} + v_{i-2} + \cdots + v_{i-(N-1)}}{N}$. The fig. 6.20 gives an example of SMA.*

With the same convention, the Exponential Moving Average *(EMA) is defined as $EMA_0 = v_0$ and $EMA_i = \alpha v_i + (1-\alpha)EMA_{i-1}$ with $\alpha = \frac{2}{N+1}$.*
The SMoothed Moving Average *(SMMA) is an EMA with different at initialization. Additionally, the N parameter of the SMMA (written N_{SMMA}) is defined as $N_{SMMA} = \frac{N_{EMA}+1}{2}$ with N_{EMA} the N parameter of the EMA.*

Definition 6.1.6 *A* Moving Average Convergence Divergence (MACD(X,Y,Z)) *index is composed of two output signals. The first one is the difference*

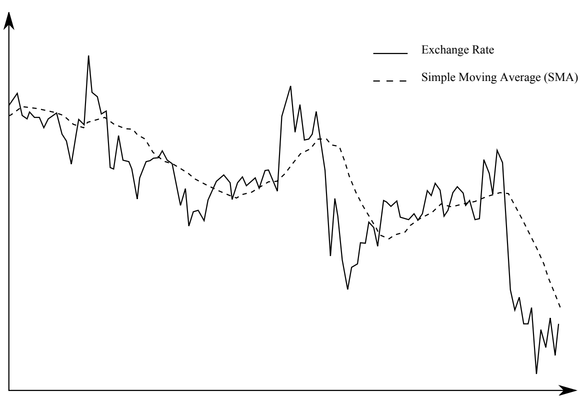


Figure 6.20: An example of signal, and the simple moving average (SMA) on this signal.

between an EMA(X) and an EMA(Y) of the exchange rate. The second one is an EMA(Z) of the first output signal.

Definition 6.1.7 *The* Awesome *index is the difference between a SMA(5) and a SMA(34). These SMAs are computed from the 'middle points' of the exchange rate. Given a signal and a window, the* middle point *is the average of the highest value and the lower value of the signal in this window.*

Definition 6.1.8 *A* trend *is a tendency of the exchange rate to increase (upward trend) or decrease (downward trend) from a time scale (from decades to minutes).*

In this work, we characterize trends with four parameters: A direction (upward or downward), a beginning time, a duration and a gradient.

There is an upward trend between times t_1 and t_2 (the duration of the trend is $t_2 - t_1 > 0$, and the gradient is $\frac{v(t_2)-v(t_1)}{t_2-t_1} > 0$), if the value $v(t)$ of signal a time $t \in [t_1, t_2]$ is always greater than $v(t_1) + (v(t_2) - v(t_1))\frac{t-t_1}{t_2-t_1}$.

Symmetrically, there is a downward trend between times t_1 and t_2 if the value $v(t)$ of signal a time $t \in [t_1, t_2]$ is always lower than $v(t_1)+(v(t_2)-v(t_1))\frac{t-t_1}{t_2-t_1}$. In this case, the gradient is $\frac{v(t_2)-v(t_1)}{t_2-t_1} < 0$.

In this experiment, the duration and gradient of the trends are limited to pre-defined sets of values. Several trends can occur simultaneously, but we avoid having overlapping trends with similar durations and gradients. We use a greedy algorithm to extract trends from the exchange rate signal. For every possible durations and gradients, we associate a symbol e.g. 'start_upward_trend_grad_is_X_duration_is_Y'. The fig. 6.21 gives an example of upward and downward trends.

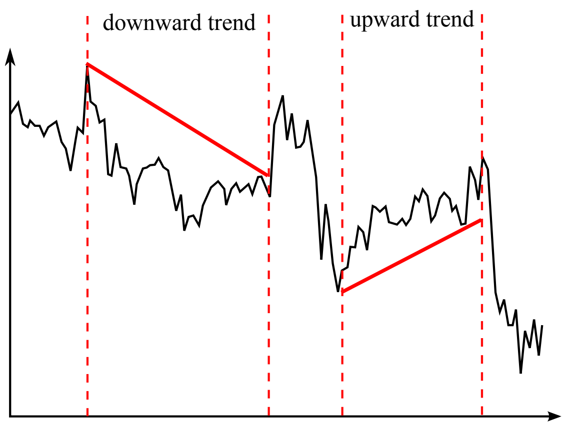


Figure 6.21: An upward an a downward trends.

Often, Traders cannot spot trends from their beginning, but with some delay. A late spotted trend can still generate profit if it is long enough.

In order to be able to use such trends, and because we configure the Titar learner to do prediction in the future only, we extract and learn to predict 'delayed trends'. The trends are delayed between 0% and 60% of their duration. We associate a symbol to each type of delayed trend e.g. 'start_upward_trend_grad_is_X_duration_is_Y_delays_Z'.

The final dataset contains:

- 264 601 exchange rate from the 1 minute samples
- 42 253 exchange rate from the 30 minute samples
- 598 076 discretized events and states

Fig. 6.22 shows 24 hours of signal, indicators and discretized events.

6.1.5.2 Results

The Titarl algorithm is applied in the first dataset (June 5 2008 – June 5 2009). The input events and input states are the discretized indicators. The output (or target) events are the discretized trends. The learning uses a logarithmic histogram bounds (the temporal precision decreases with the temporal distance) selected such that the predictions are always in the future (from 0 to 12 hours in the future). The number of conditions is limited to eight. The minimum confidence is set to 30%, the minimum support is set to 5%, and the minimum number of uses of the rules is set to 20. We run the algorithm for '300 seconds + initialization time' on every type of discretized trend. The algorithm returns 886 343 rules. The repartition of the number of conditions is given by fig 6.23. Fig.6.24 shows an example of a learned Tita rule.

In the rest of the document, we will evaluate the profit of learned rules. By writing convention, the average profit of a rule is the average profit of the orders made by this rule. The gain and the risk of a prediction are proportional to the initial investment and the leverage. Some automated trading systems can select different initial investment. In our case, we are set all the initial investment to be 100 000 money unit (called a lot). We are also using a leverage of 100x, and we are supposing a spread of 1 pip (i.e. a difference of 0.0001 between the buying price and the selling price) on the EURUSD. These values are common values.

First try The confidences of the rules are estimated on the first part of the dataset (June 5 2008 – June 5 2009). The average annual profits, annual

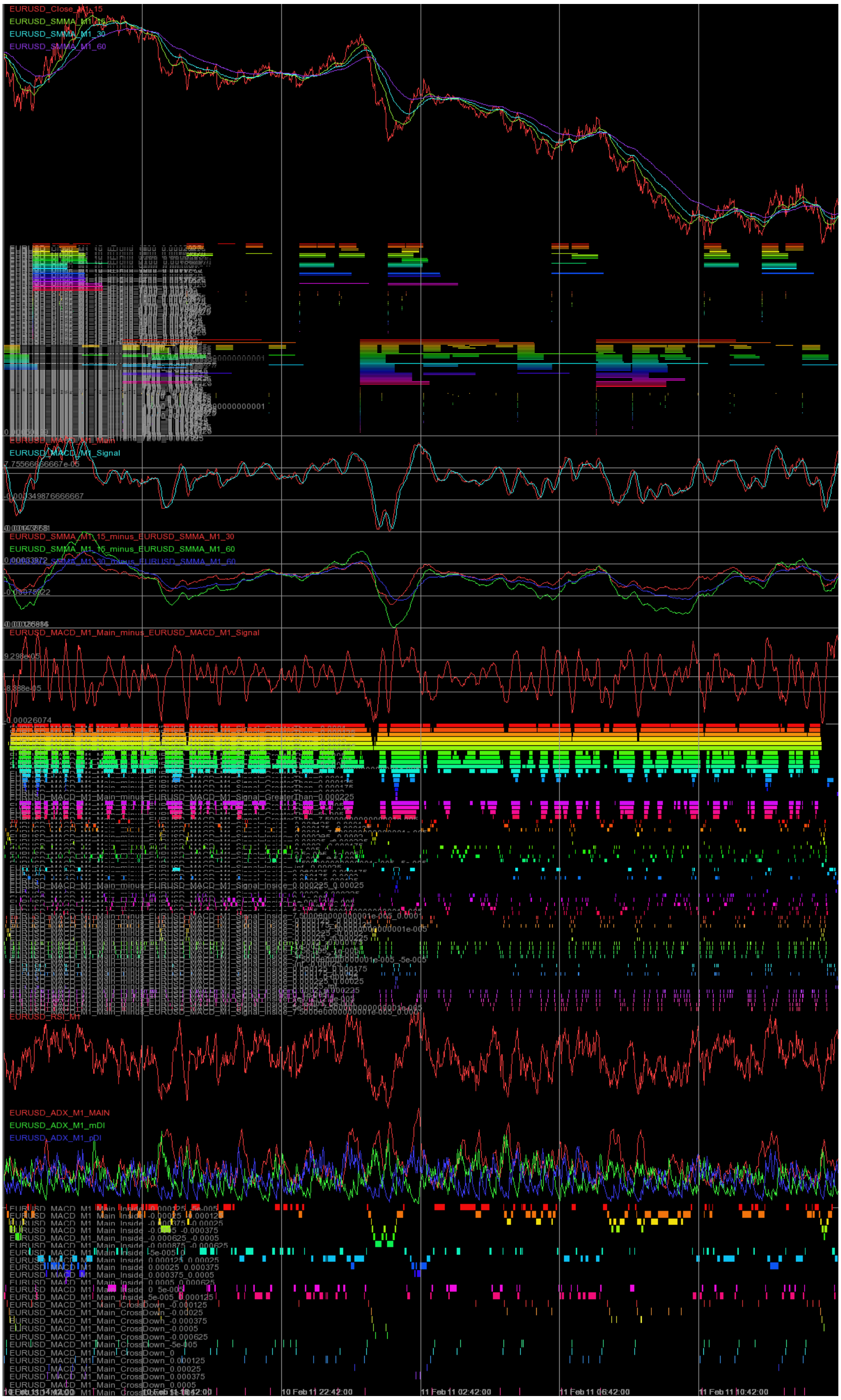


Figure 6.22: Twenty four hours preview of signal, indicators and discretized events of the EURUSD exchange rate.

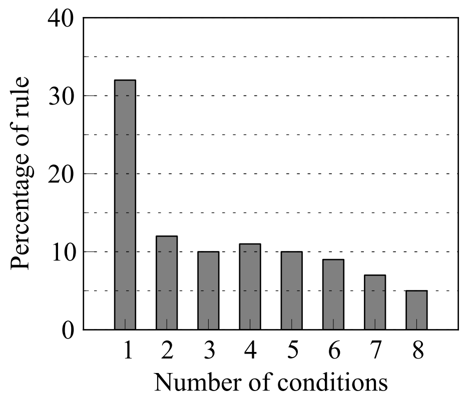


Figure 6.23: Repartition of the number of conditions for the Forex learned rules.

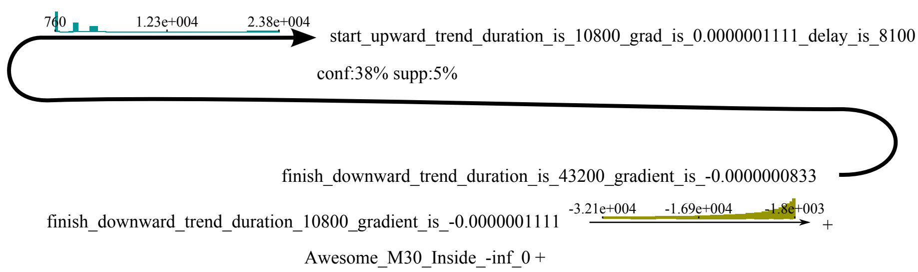


Figure 6.24: Example of learned rules that predicts the beginning of an upward trend.

profits and number of orders are estimated on the second part of the dataset (June 5 2009 – June 5 2010).

The fig. 6.25(a) shows the correlation between the number of orders of the rules in the first and the second dataset parts. The fig. 6.25(b) shows the correlation between the average annual profit of the rules in the first and the second dataset parts. We observe a strong correlation between the number of uses of rules from the first and second dataset (fig. 6.25(a)). However, the average annual profits from the first and second datasets appear to be decorrelated (fig. 6.25(b)). The fig. 6.26 shows the correlation between the confidence of the rules, the average annual profits, the annual profits and the number of orders. We observe a strong correlation between the confidence of a rule, and the number of uses of this rule: Rules with low confidence are applied more time than rules with high confidence (fig. 6.26(a)). However, there is no correlation between the confidence of a rule, and the average annual profits (fig. 6.26(b)). The average profits and average losses of rules

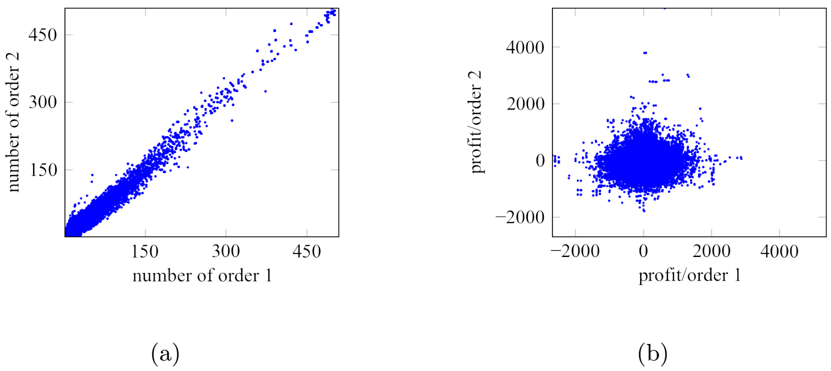


Figure 6.25: Correlation (Average annuals profit and number of orders) between the first and second datasets

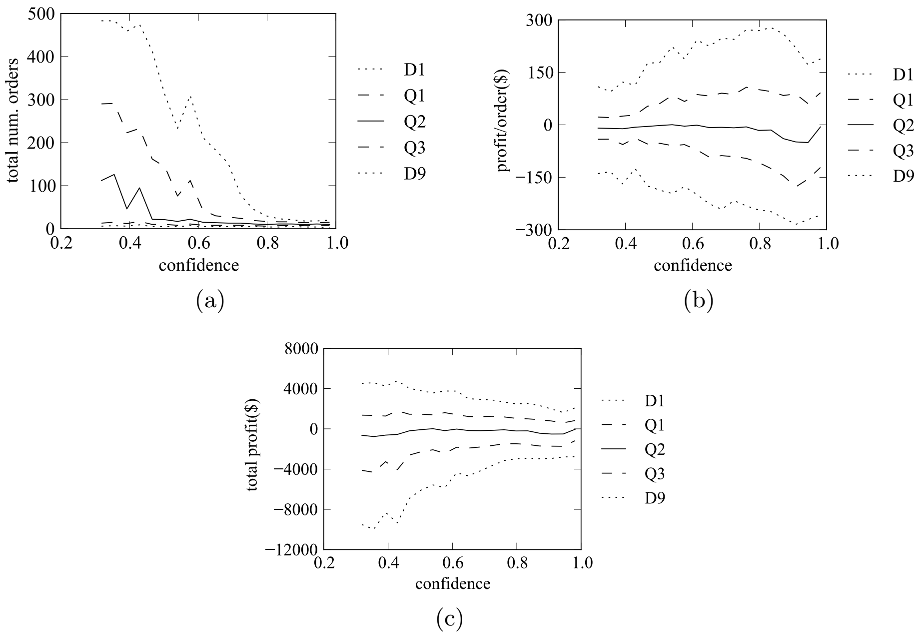


Figure 6.26: Correlation between the confidence, the average annual profit, the annual profits and the number of orders of rules. Q1, Q2 and Q3 are respectivly the first, second and third quantiles. The second quantile is the median. D1 and D9 are the first and ninth deciles.

with high confidence are both higher than the ones of rules with low confidence. There are no correlations between the confidence of a rule, and the annual profits of a rule (fig. 6.26(c)). The annual profits and the annual losses of rules with low confidence are higher than the ones of rules with high confidence. Finally, the average profits, and the annual profits is close to zero and slightly negative (fig. 6.26(b) and fig. 6.26(c))

This first try of the experiment is non-conclusive. The learned rules seem to be globally over-trained (the confidence of the rule is globally uncorrelated between the first and second dataset parts).

Second try In the second part of the experiment, we filter the over-trained rules (learned from the first part of the dataset) with the second parts of the dataset (June 5 2008 – June 5 2009 and June 5 2009 – June 5 2010). The rules that are kept will be evaluated on the third part of the dataset (June 5 2010 – June 5 2011).

Observations reveal that the predictions of some type of trends show a correlation between the average profits on the dataset parts 1 and 2. The figs. 6.27(a) and 6.27(b) show two trends with correlation. The fig. 6.27(c) shows a trend with no correlation through the two datasets.

We propose two measures to characterize the correlation between the average profits:

1. Given a set of rules predicting a given discretized trend, the first measure is the percentage of the rules with positive average profits in the two dataset parts. Given a threshold α, we are keep trends with such ratio greater than α. In addition, we are only keeping the rules with positive average profits in the two dataset.

2. Given a set of rules predicting a given discretized trend, the second measure is the square mean of the difference of average profits on the two dataset. Given a threshold β, we are keep trends with such ratio lower than β. In addition, we are only keeping the rules with positive average profits in the two dataset.

We evaluate all the type of trends with theses two measures. We consider several values for the thresholds of the two measures (from 0 to 1 for α, and from 0 to 2 for β).

The fig. 6.28 represent graphically the experiment as well as the two measures to filter the rules.

Figs. 6.29(a), 6.29(b), 6.29(c) and 6.29(d) show the number of remaining rules, the number of orders, the average profits and the annual profits of the rules filtered with the first measure. Figs. 6.30(a), 6.30(b), 6.30(c) and

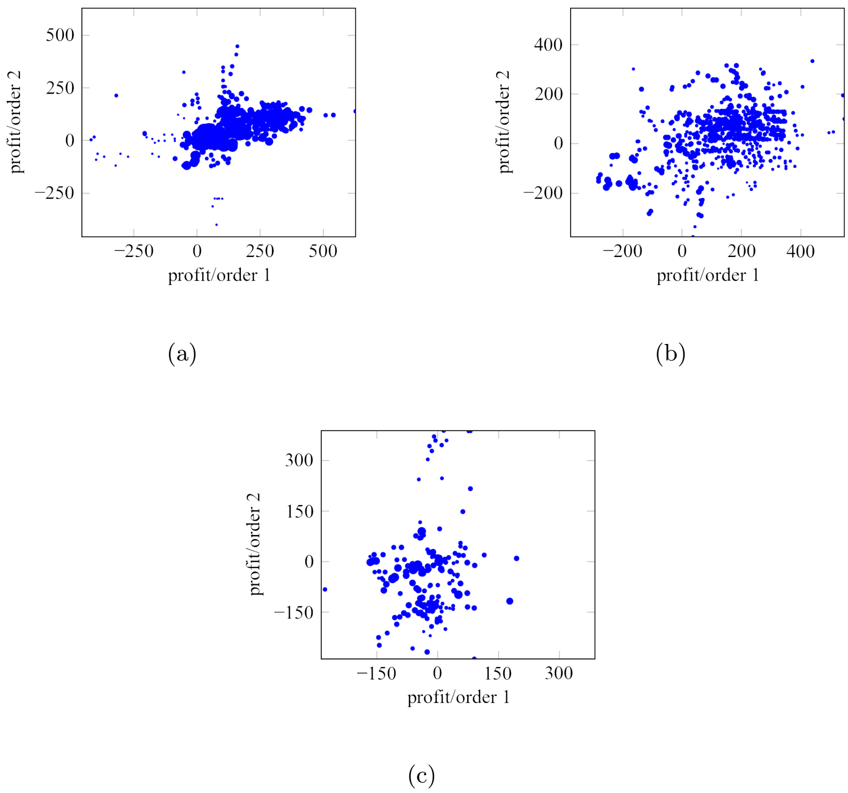


Figure 6.27: Correlation of average profits between the first and second datasets for three different trends.

6.30(d) show the number of remaining rules, the number of orders, the average profits and the annual profits of the rules filtered with the second measure.

For the first measure, the number of rules and the number of orders by rule (approximately) decreases exponentially when the threshold α increases (figs. 6.29(a) and 6.29(b)). The average profits and the annual profits increase with α (figs. 6.29(c) and 6.29(d)). This profit is computed with the third part of the dataset. The distances between quantiles of the average profits are constant. With $\alpha = 80\%$, it remains 1186 rules with an average profit by order of \$480, the average annual profits of rules over the year is \$3061, and the average number of orders by year is 6.4.

For the second measure, the number of rules and the number of orders by rule increases when the threshold β increases (figs. 6.30(a) and 6.30(b)). The average profits and the annual profits decrease with β (figs. 6.30(c) and 6.30(d)). This profit is computed with the third part of the dataset. With $\beta = 0.4\%$, it remains 443 rules with an average profit by order of \$194,

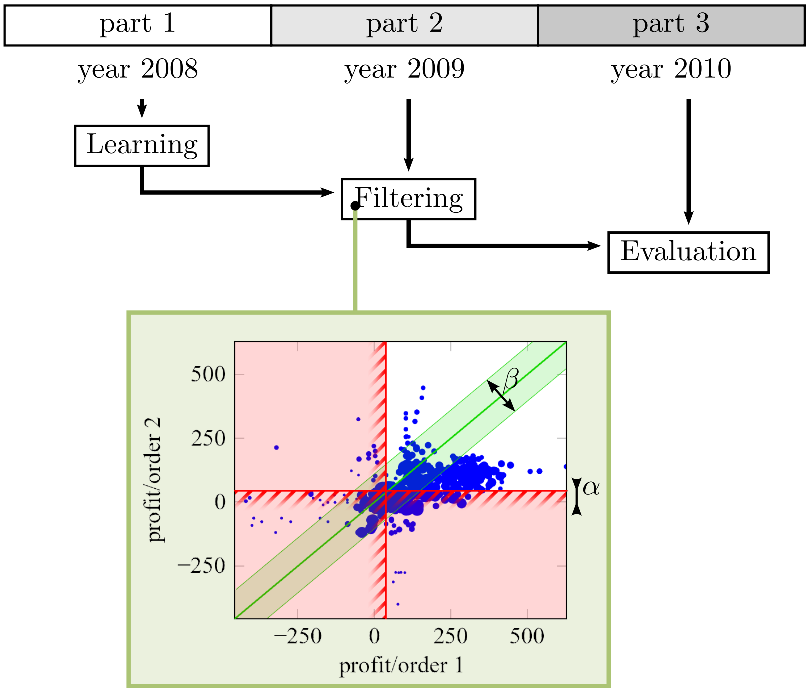


Figure 6.28: Graphical representation of the experiment. The rules are filtered according to their average profile on the years 2008 and 2009. The two parameters (α and β of the filters are displayed.

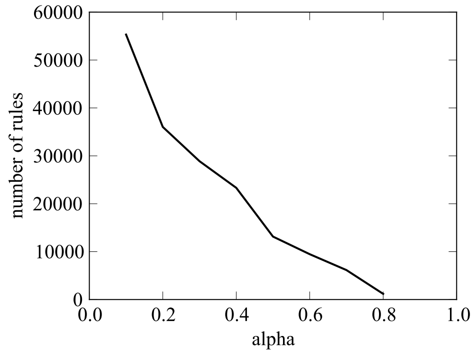
the average annual profits of rules over the year is $2733, and the average number of orders by year is 14.6.

6.1.5.3 Discussion

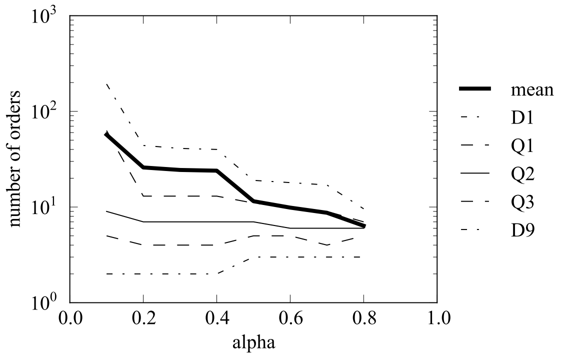
To demonstrate the scalability of the Titar algorithm, we apply our algorithm to a large dataset.

We developed a simple methodology to convert time series of Forex records into symbolic time sequences. This method is basic, exhaustive and non-specific for Forex trading. Having a deeper knowledge of the Forex would have certainly leaded us to design a more suitable method with better numerical results. However, this simple method shows that our technique is non-specialized and can be use in other domains.

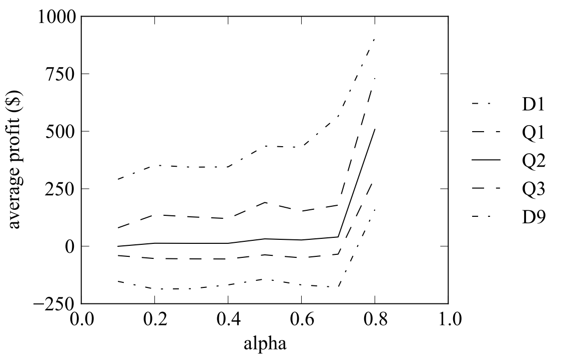
We applied our learner and evaluate the result with a cross-validation protocol. We chose to split the dataset into three temporal continuous subsets of one year each. The other option would have been to split the dataset day by day (of week by week), and assign each day (or each week) to one of the subset. Fig. 6.31 shows these two types of splitting. This second option would have lead to more 'uniform' subsets. However, this would have also



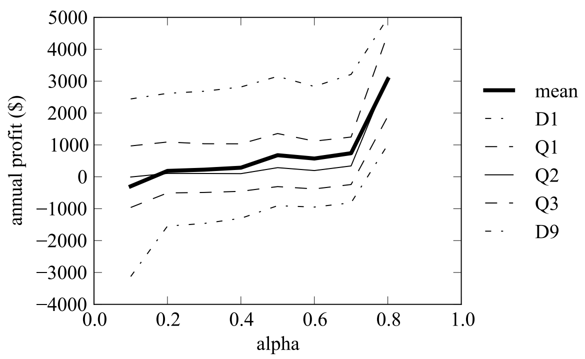
(a)



(b)

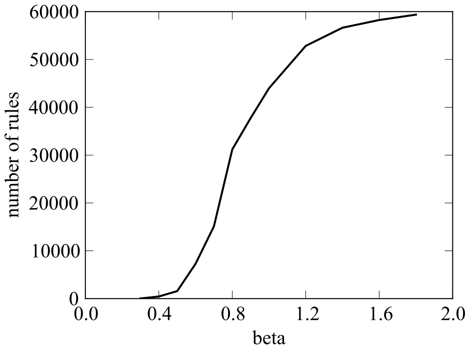


(c)

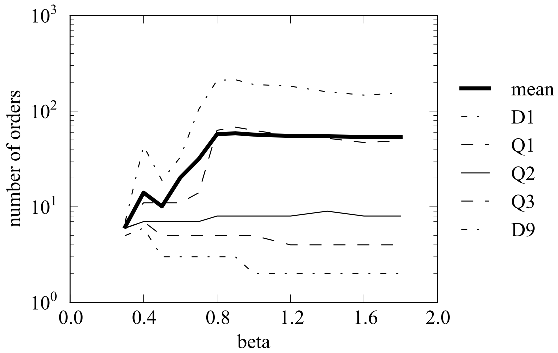


(d)

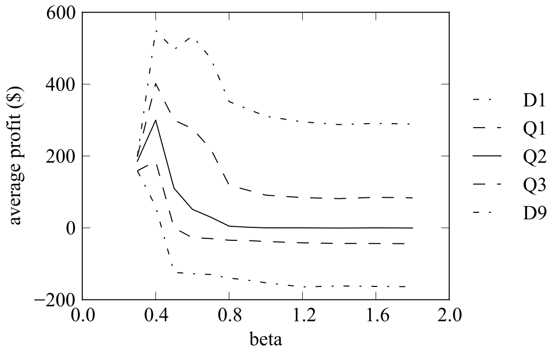
Figure 6.29: Number of rules, number of orders, the average profits and the annual profits of the rules filtered with the first measure. Q1, Q2 and Q3 are respectivly the first, second and third quantiles. The second quantile is the median. D1 and D9 are the first and ninth deciles.



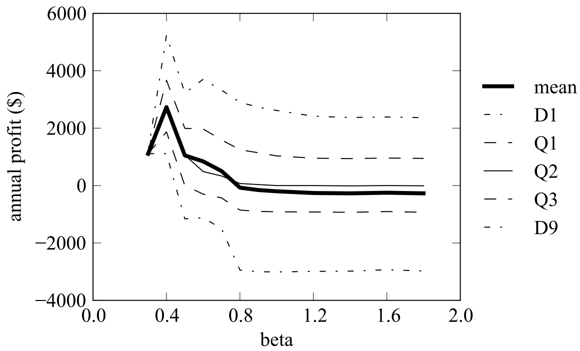
(a)



(b)



(c)



(d)

Figure 6.30: Number of rules, number of orders, the average profits and the annual profits of the rules filtered with the second measure. Q1, Q2 and Q3 are respectivly the first, second and third quantiles. The second quantile is the median. D1 and D9 are the first and ninth deciles.

required to learn from data in the future in order to predict events in the past.

Since the Forex changes over time, some of the rules that were true in 2008 are no longer valid in 2009 or 2010. We believe this is one of the reasons for the absence of correlation between the confidence and the profit of the rules learned in 2008 and evaluated in 2009 (first experiment). However, we have supposed that some of the rules of the Forex are constant over time periods longer than couple of years. This hypothesis has been confirmed by the conclusive results of the second part of our experiment.

In conclusion, we developed a global methodology including preprocessing of Forex record, cross-selection and cross-validation to evaluate our algorithm to predict Forex market. The results are conclusive.

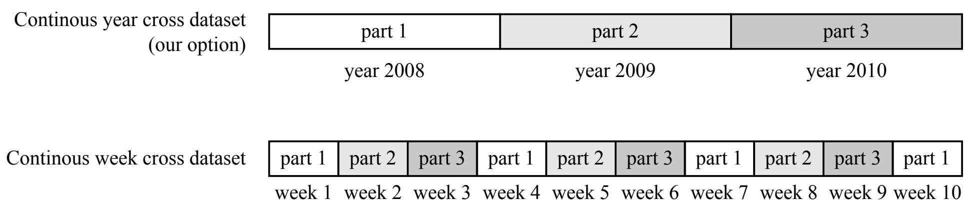


Figure 6.31: Division of the dataset in three part with continuous years or continuous weeks.

6.2 Temporal planning

6.2.1 Robot in a simulated world

This experiment shows one of the possible uses of the planning algorithm presented in chapter 5 and the learning algorithms presented in chapter 4. We shows how a robot located in an (initially) unknown world, will learn the rules of the world, and use this knowledge to achieve its goals. With the Titarl algorithm, the robot has the ability to understand the world. With the Titarl Planner algorithm, the robot has the ability to generate plans to reach its goals.

In this experiment, we assume that the world is non deterministic (an event cannot be predicted with 100% confidence), the world is inaccurate (predictions can only be made on time ranges and not on time points), and the world has unpredictable processes running i.e. even if the robot does not do anything, some events will still occur. By opposition to the usual experiments in planning and scheduling, time is considered to be continuous.

The world is composed of three buttons, three lights and a treat dispenser. The goal of the robot is to get treats. There are several mechanisms between the buttons, lights and the treat dispenser. These mechanisms have different levels of complexity and uncertainty. These mechanisms are unknown from the robot. The hidden "rules" of the world are presented in the next section. Fig. 6.32 is a visual representation of the problem.

Virtual world rules

The world is composed of three buttons (called b1, b2 and b3), three lights (called l1, l2 and l3) and a treat dispenser (called treat). The goal of the robot is to get as many treats as possible. The rules of the world are the following ones:

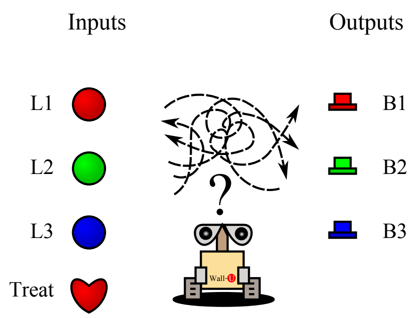


Figure 6.32: Visual representation of the "Robot in simulated world" problem.

- r_0 **:** The lights l1, l2 and l3 flashes randomly with a respective average of a flash every 50, 40 and 100 seconds.
- r_1 **:** There is a 50% chance that a treat is given to the robot between 3 and 4 seconds after the button b1 is pressed, if the button b1 was not pressed in the previous 20 seconds.
- r_2 **:** There is a 90% chance that the light l2 flashes between 0 and 10 seconds after the button b3 is pressed, if the button b3 was not pressed in the previous 2 seconds.
- r_3 **:** There is a 90% chance that a treat is given to the robot between 3 and 4 seconds after the button b2 is pressed, if the light l2 has flashed in the previous 3 seconds, and if the button b2 was not pressed in the previous 3 seconds.

The rule r_0 gives an external noise in the world that the robot cannot predict. The goal of this external noise is to simulate the uncertainty of the real world.

This rule r_1 directly correlates an action to the reward. This rule is the simplest rule that predicts treat. However, this rule has a low probability (50%) and it cannot be used more than once every 20 seconds.

The rule r_2 combined with the rule r_3 can be combined to get a treat. This combination is the best solution to maximize the number of treats. These two rules are not perfectly accurate (r_2 and r_3 have respectively a 10 and 3 seconds range). The combination of these two rules is partially similar to the example 3.9.1: The optimal plan is to trigger the rule r_2, wait for the light l2 to be activated (l2 might never be activated), and when (and if) l2 is activated, to trigger the rule r_3. This combination is especially complex because the robot has to wait for the light l2, but it has no guaranties that it will actually happen. Such Meta-plan has optimally a confidence of $81\% = 90\% 90\%$. If the robot combines these two rules into a plan (not a Meta-plan), then the resulting plan would have a confidence of $8.1\% = 90\%\ 90\%\ \frac{4-3}{10-0}$.

The robot can also wait for the rule r_0 to tiger the light l2, and use the rule r_3 to get a treat.

Robot cognitive policy

The robot cognitive policy is composed of two states: the exploration state and the planning state.

Initially, the robot does not know the world and will only do random actions at random times. The action to perform is chosen with a random uniform selection through the possible robot actions. The amount of time the robot waits between actions is also uniform random between 0 and 20 seconds.

After a period of random acting, the robot analyses its behavior and the behavior of the world with the Titarl algorithm. It extracts a set of Tita rules and enters the planning state.

In this state, ten times a second, the robot builds and executes a linear Meta Tita plan with the objective of getting food in the next 30 seconds, based on the learned rules. The robot only stores the plan instances (the robot does not store/keep in mind the all linear Meta-plan). The Meta-plan is recomputed every time. This process is a simple architecture with the ability to react to the uncertainty of the world e.g. if, while a plan is executed, a new and better plan appears to be possible (because of the external sources of randomness of the world for example), the old plan will be replaced by the new one. This aspect is discussed in the conclusion of the experiment.

Results of the experiment

Table 6.5 shows the duration, number of events, number of triggered rules and number of treats of the exploration and planning phases of the experiment.

Table 6.5: Duration, number of events, number of triggered rules and number of treats of the exploration and planning phases of the experiment. The abbreviation “N.o.” replace “Number of”.

Phase	Duration	N.o. events				N.o. rule tigers		N.o. treats
Exploration	10000s	b1:	554	l1:	211	r1:	183	190
		b2:	622	l2:	590	r2:	529	
		b3:	545	l3:	258	r3:	99	
Planning	5000s	b1:	0	l1:	98	r1:	0	301
		b2:	335	l2:	375	r2:	365	
		b3:	365	l3:	117	r3:	335	

The figures 6.33 and 6.34 present respectively a sample the record of the exploration and planning phases. The events trigger_r1, trigger_r2 and trigger_r3 represent the triggering of the rules r1, r2 and r3. Of course, the robot does not have access to these signals.

During the experiment, the confidence spread is fixed to 5. Reducing this parameter will produces smaller Meta-plan but will possibly “miss” some solutions. A confidence spread of 5 produces Meta-plans too large to be printed. Therefore, we will use a confidence spread 1 for the following figures. As expected, we see experimentally that, the Meta-plans built with a confidence spread of 1 gives less efficient plan that the ones built with a confidence spread of 5. However, the results (number of treats, structure of the graph, etc.) are still close enough to be presented.

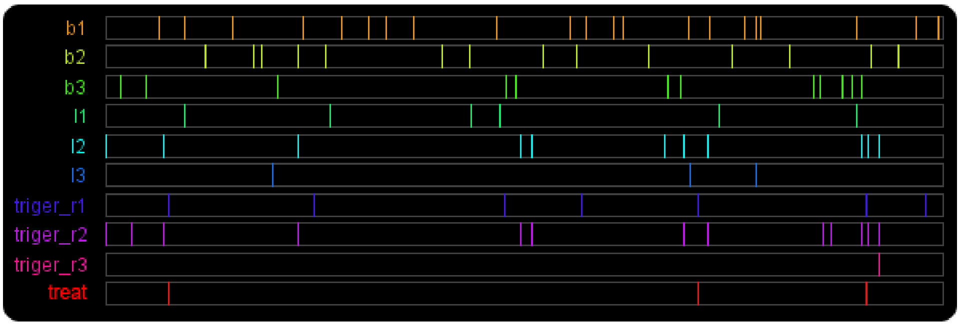


Figure 6.33: Extract of the exploration phase.

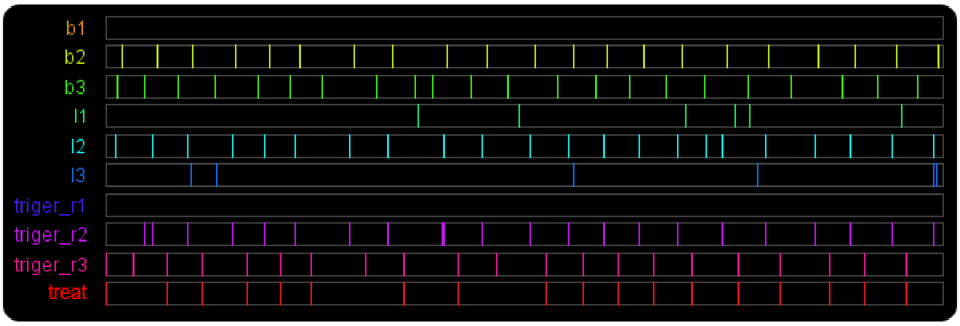


Figure 6.34: Extract of the planning phase.

The fig. 6.35 shows one of the Meta-plans generated during the planning phase. The fig. 6.36 shows the linear Meta-plan extracted from this Meta-plan.

Conclusion

The first observation about the experiment is that the robot successfully understood the hidden rules of the world, and it successfully uses them to get treats. Second, the robot did not extract the exact rules of the world: The negative parts of the conditions of the rules r_2 and r_3 are not correctly understood. However, the robot reaches its goal with the learned approximate rules. The analysis of the rules learned by the robot shows that the negation of the rules r_2 and r_3 are actually detected but the time intervals associated with the conditions are not correctly estimated. Because of this incorrect estimation, the rules with negations have low confidence and are discarded by the planning algorithm.

Other simulations with longer exploration phase or with less activation of the rule r_0 (external random source) shows that the robot can actually

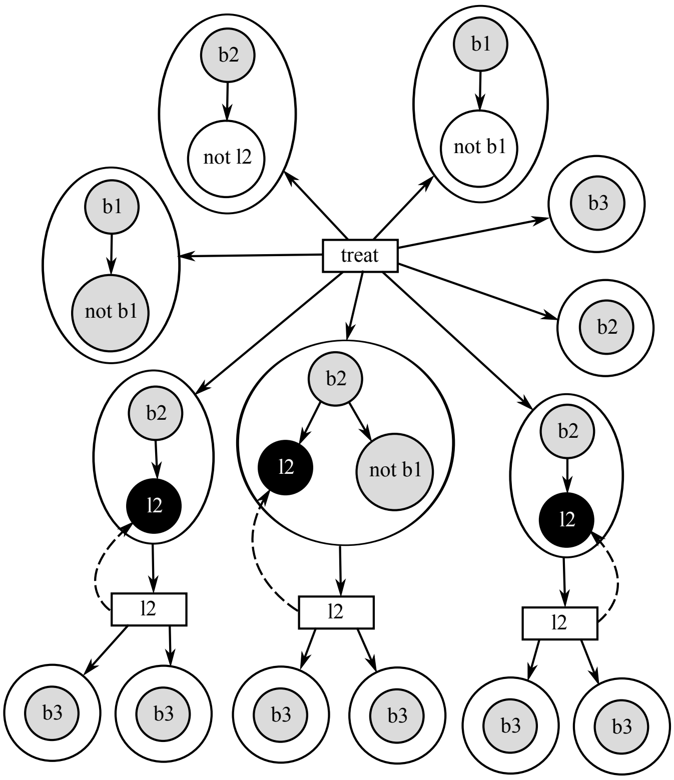


Figure 6.35: Representation of the (non linear) Meta-plan built with a confidence spread of 1. The graph is too large to be detailed on a page, the main interest of this figure is to show the global structure of the graph. The relative and absolute constraints are not displayed.

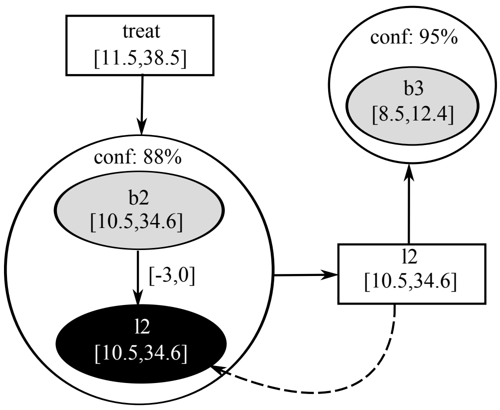


Figure 6.36: Representation of the linear Meta-plan built with a confidence spread of 1. Goal was to produces 'treat' in the interval [8.5,38.5] when the current time was 8.5 (No actions cannot be done before 8.5). The global confidence of the Meta-plan is the product of the nodes' confidences i.e. $95\% \cdot 88\% = 84\%$.

correctly learn the negations of the rules r_2 and r_3 with larger or less noisy training examples. This is an example of how an approximate/incorrect representation of a system (the rules are not perfectly learned) can still be good enough to accomplish some goals.

7

Conclusion and perspective

Chapter Contents

This thesis addresses two challenges from two distinct domains of Computer Science.

The first challenge, which part of the Data Mining domain, is to define temporal pattern models (models able to describe temporal processes) and to develop algorithm able to extract (or learn) these patterns from sets of observations. The core of this challenge is for the models and the algorithms to be respectively efficient in describing temporal processes, and to be efficient in extracting these temporal patterns.

The second challenge is part of the Planning and Scheduling domain. Suppose a theoretical world described by rules. The second challenge is to automatically build plans that will lead to desired states in this theoretical world.

7.1 Contributions

The outcome of this thesis can be summarized with four main contributions:

- First, we introduce a new temporal pattern model for temporal association rule. We called this pattern *Tita rules* (or *Titar*). This pattern model can express both uncertainty and temporal inaccuracy of temporal events. Among other things, it can express the usual time point operators, synchronicity, order, and chaining, as well as temporal negation.

- The second contribution is the development of an efficient learning algorithm for Tita rules in large and noisy time sequences.
- Since Tita rules can describe imprecise and inaccurate phenomena, usual planning algorithms and planning grammars are not suited to deal with phenomena described with Tita rules. Our third contribution is the introduction of a grammar to express temporal plans based on Tita rules. We called this grammar *Meta Titar Plan*.
- The last contribution is the development of a planning and scheduling algorithm that automatically build Meta Titar Plan from a world described with Tita rules.

7.2 Impact and Perspectives

The main work of this thesis was to propose a novel technique to represent and learn temporal patterns. The core problem of temporal learning (the Temporal Constraint Selection problem) has been identified, and an efficient solution was proposed and successfully evaluated on several of our domains of interest. The learning of temporal patterns has a large number of possible direct and indirect applications:

First of all, it is a powerful tool to analyze and understand complex stochastic processes. Examples of such processes are automated fault prediction and detection: In complex systems, each component has a certain probability to fail. Moreover, the probability of each component to fail is correlated with the use of the system and the behavior of other components. In the case of large, complex and critical systems, being able to locate and predict components fails is essential.

Second, temporal learning can be use to automatically build models, and use these models to predict/infer future/hidden phenomena. For examples, Decision aids tools for human biology: The human body is by nature extremely complex to understand, and it seems to be subject to a large amount of randomness from a physician point of view. In this case, temporal learning can be used to analyze medical records, to build generalized models of the human biology, and finally to infer diagnostics when a new medical record is presented. Another good example of such use is the prediction of human behavior: Being able to learn the habits of people is the key to powerful Home automation systems.

Third, temporal learning can be integrated into more complex automatic systems in order to give them a certain degree of autonomy. This is especially useful in domains where human cannot operate because of inability to communication with the system, communication latency with the system, too slow human reactivity or too important cost of operators.

From our point of view, an interesting extension of our technique would be the Spatio-temporal learning problem. Similarly to Time, Space is a non trivial dimension in Machine Learning: The space is a multi-dimensional space with very specific properties. We think that our solution for the Temporal Constraint Selection Problem can be adapted to a solution for the 'Spatial Constraint Selection Problem' that would represent the core of spatial learning. The result would lead to an efficient Spatio-temporal learning algorithm.

Such techniques could be used in a new range of domains such as epidemic spread analysis or animal migration analysis.

In this work, we were able to identify a non trivial use for our temporal pattern model: Planning and Scheduling. We investigate the implications of using our model for this task, and we discovered new useful theories and tools such as the Meta-plans. Our experiment of merging our temporal learning algorithm and our temporal planning algorithm, in order to control a robot, illustrates the possible combination of temporal learning and temporal planning.

We believe that the meeting of different scientific domains is a great source for new discoveries. Therefore, similarly to the planning and scheduling domain, we believe that the use of the Titar model in other domains would be very interesting, and would possibly lead to new theories and new tools.

Bibliography

Agrawal, R. and Srikant, R. "Fast algorithms for mining association rules in large databases". In "Proceedings of the 20th International Conference on Very Large Data Bases", VLDB '94, pages 487–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1994). ISBN 1-55860-153-8.

Anderson, J.R., Bothell, D., Byrne, M.D., Douglass, S., Lebiere, C. and Qin, Y. "An integrated theory of the mind". In *PSYCHOLOGICAL REVIEW*, 111:1036–1060 (2004).

Arroyo-Figueroa, G. and Suear, L.E. "A temporal bayesian network for diagnosis and prediction". In "Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence", UAI'99, pages 13–20. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999). ISBN 1-55860-614-9.

Atsalakis, G.S. and Valavanis, K.P. "Surveying stock market forecasting techniques - part ii: Soft computing methods". In *Expert Syst. Appl.*, 36(3):5932–5941 (2009). ISSN 0957-4174.

Balaban, M. and Rosen, T. "Stcsp : structured temporal constraint satisfaction problems". In *Annals of Mathematics and Artificial Intelligence*, 25:35–67 (1999). ISSN 1012-2443.

Bettini, C., Sean Wang, X., Jajodia, S. and Lin, J.L. "Discovering frequent event patterns with multiple granularities in time sequences". In *IEEE Trans. on Knowl. and Data Eng.*, 10(2):222–237 (1998). ISSN 1041-4347. doi:10.1109/69.683754.

Blum, A.L. and Furst, M.L. "Fast planning through planning graph analysis". In *Artif. Intell.*, 90(1-2):281–300 (1997). ISSN 0004-3702.

Brélaz, D. "New methods to color the vertices of a graph". In *Commun. ACM*, 22:251–256 (1979). ISSN 0001-0782.

Cao, L. and Tay, F.E. "Financial forecasting using support vector machines". In *Neural Computing Applications*, 10:184–192 (2001). ISSN 0941-0643. 10.1007/s005210170010.

Casas-Garriga, G. "Discovering unbounded episodes in sequential data". In N. Lavrac, D. Gamberger, L. Todorovski and H. Blockeel, editors, "Knowledge Discovery in Databases: PKDD 2003", volume 2838 of *Lecture Notes in Computer Science*, pages 83–94. Springer Berlin / Heidelberg (2003). ISBN 978-3-540-20085-7.

Chen, Y.L., Chiang, M.C. and Ko, M.T. "Discovering time-interval sequential patterns in sequence databases". In *Expert Systems with Applications*, 25(3):343 – 354 (2003). ISSN 0957-4174. doi:10.1016/S0957-4174(03)00075-7.

Connor, N.O. and Madden, M.G. "A neural network approach to predicting stock exchange movements using external factors". In *Knowledge-Based Systems*, 19(5):371 – 378 (2006). ISSN 0950-7051. ¡ce:title¿AI 2005 SI¡/ce:title¿.

Dechter, R., Meiri, I. and Pearl, J. "Temporal constraint networks". In *Artif. Intell.* (1991).

Dempster, A.P., Laird, N.M. and Rubin, D.B. "Maximum likelihood from incomplete data via the em algorithm". In *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38 (1977).

Dittmar, R., Neely, C.J. and Weller, P. "Is technical analysis in the foreign exchange market profitable? a genetic programming approach". CEPR Discussion Papers 1480, C.E.P.R. Discussion Papers (1996).

Dousson, C. and Duong, T.V. "Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems". In "Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1", pages 620–626. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999).

Fikes, R.E. and Nilsson, N.J. "Strips: A new approach to the application of theorem proving to problem solving". In *Artificial Intelligence*, 2(3-4):189–208 (1971).

Gan, W.S. and Ng, K.H. "Multivariate forex forecasting using artificial neural networks". In "Neural Networks, 1995. Proceedings., IEEE International Conference on", volume 2, pages 1018 –1022 vol.2 (1995).

Geng, L. and Hamilton, H.J. "Interestingness measures for data mining: A survey". In *ACM Comput. Surv.*, 38 (2006). ISSN 0360-0300.

Guillame-Bert, M. "Symbolic time sequence dataset". In (2011). "available at _urlhttp://www-prima.imag.fr/guillame-bert/?page=database".

Guillame-Bert, M. and Crowley, J.L. "New approach on temporal data mining for symbolic time sequences: Temporal tree associate rules". In "Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence", ICTAI '11, pages 748–752. IEEE Computer Society, Washington, DC, USA (2011). ISBN 978-0-7695-4596-7.

Harms, S.K., Deogun, J., Saquer, J. and Tadesse, T. "Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints". In "In Proceedings of the 2001 IEEE International Conference on Data Mining", pages 603–606. IEEE Press (2001).

Hernandez-Leal, P., Sucar, L.E. and Gonzalez, J.A. "Learning temporal nodes bayesian networks". In "FLAIRS Conference", (2011).

Hirate, Y. and Yamana, H. "Generalized sequential pattern mining with item intervals". In *JCP*, pages 51–60 (2006).

Kamruzzaman, J., Sarker, R.A. and Ahmad, I. "Svm based models for predicting foreign currency exchange rates". In *Data Mining, IEEE International Conference on*, 0:557 (2003).

Kautz, H.A., Selman, B. and Hoffmann, J. "SatPlan: Planning as satisfiability". In "Abstracts of the 5th International Planning Competition", (2006).

Lehman, J.F., Laird, J.E. and Rosenbloom, P.E. "A gentle Introduction to Soar: 2006 update" (2006).

Mannila, H. and Toivonen, H. "Discovering generalized episodes using minimal occurrences". In "In Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining", pages 146–151. AAAI Press (1996).

Mannila, H., Toivonen, H. and Verkamo, A.I. "Discovery of frequent episodes in event sequences". In *DATA MINING AND KNOWLEDGE DISCOVERY*, 1:259–289 (1997).

Méger, N. and Rigotti, C. "Constraint-based mining of episode rules and optimal window sizes". In "Proceedings of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases", PKDD '04, pages 313–324. Springer-Verlag New York, Inc., New York, NY, USA (2004). ISBN 3-540-23108-0.

Mörchen, F. "Unsupervised pattern mining from symbolic temporal data". In *SIGKDD Explor. Newsl.*, 9:41–55 (2007). ISSN 1931-0145.

Myszkowski, P. and Bicz, A. "Evolutionary algorithm in forex trade strategy generation". In "Computer Science and Information Technology (IMCSIT), Proceedings of the 2010 International Multiconference on", pages 81 –88 (2010). ISSN 2157-5525.

Oates, T., Jensen, D. and Cohen, P.R. "Automatically acquiring rules for event correlation from event logs". Technical report (1997).

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1988). ISBN 0-934613-73-7.

Pei, J., Wang, H., Liu, J., Wang, K., Wang, J. and Yu, P. "Discovering frequent closed partial orders from strings". In *Knowledge and Data Engineering, IEEE Transactions on*, 18(11):1467 –1481 (2006). ISSN 1041-4347.

Pei, J., Han, J., Mortazavi-asl, B., Pinto, H., Chen, Q., Dayal, U. and chun Hsu, M. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth" (2001).

Quinlan, J.R. "Induction of decision trees". In "Machine Learning", pages 81–106 (1986).

Raïssi, C., Calders, T. and Poncelet, P. "Mining conjunctive sequential patterns". In W. Daelemans, B. Goethals and K. Morik, editors, "Machine Learning and Knowledge Discovery in Databases", volume 5211 of *Lecture Notes in Computer Science*, pages 19–19. Springer Berlin / Heidelberg (2008). ISBN 978-3-540-87478-2.

Schwartz, P.J. and Pollack, M.E. "Planning with disjunctive temporal constraints". In *ICAPS04 Workshop on Integrating Planning into Scheduling* (2000).

Srikant, R. and Agrawal, R. "Mining sequential patterns: Generalizations and performance improvements". pages 3–17 (1996).

van Kasteren, T., Noulas, A., Englebienne, G. and Kröse, B. "Accurate activity recognition in a home setting" (2008a).

van Kasteren, T., Noulas, A., Englebienne, G. and Kröse, B. "Accurate activity recognition in a home setting". In "UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing", pages 1–9. ACM, New York, NY, USA (2008b). ISBN 978-1-60558-136-1.

Vere, S.A. "Planning in time: Windows and durations for activities and goals". In *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-5(3):246 –267 (1983). ISSN 0162-8828.

Vidal, T. and Fargier, H. "Handling contingency in temporal constraint networks: from consistency to controllabilities". In *Journal of Experimental and Theoretical Artificial Intelligence* (1999).

Vilalta, R. and Ma, S. "Predicting rare events in temporal domains". In "Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on", pages 474 – 481 (2002).

Wren, C., Ivanov, Y., Wren, C.R., Ivanov, Y.A., Leigh, D., Leigh, D., Westhues, J. and Westhues, J. "The merl motion detector dataset". Technical report, 2007 Workshop on Massive Datasets. Mitsubishi Electric Research Laboratories (2007).

Yao, J. and Tan, C.L. "A case study on using neural networks to perform technical forecasting of forex" (2000).

Zaki, M.J. "Efficient enumeration of frequent sequences". In "CIKM '98: Proceedings of the seventh international conference on Information and knowledge management", ACM (1998).

Özden, B., Ramaswamy, S., Sridhar, O. and Silberschatz, A. "Cyclic association rules". In "In Proc. 14th International Conference on Data Engineering (ICDE", (1998).