IMT Institute for Advanced Studies, Lucca

Lucca, Italy

Graph and Network Data: Mining the Temporal Dimension

PhD Program in Computer Science and Engineering

XXI Cycle

By

Michele Berlingerio

2009

The dissertation of Michele Berlingerio is approved.

Program Coordinator: Prof. Ugo Montanari, IMT, Lucca

Supervisor: Dr. Fosca Giannotti, ISTI - CNR, Pisa, Italy

Supervisor: Dr. Francesco Bonchi, Yahoo! Research, Barcelona, Spain

Tutor: Prof. Paolo Ciancarini, IMT, Lucca

The dissertation of Michele Berlingerio has been reviewed by:

Dr. Aristides Gionis, Yahoo! Research, Barcelona, Spain

Prof. Dimitrios Gunopulos, University of Athens, Greece

Dr. Christophe Rigotti, INSA, Lyon, France

IMT Institute for Advanced Studies, Lucca

2009

To my family

Contents

List of Figures

List of Tables

Acknowledgements

There is a very long list of persons I would like to thank for helping me achieving me this result.

First, I would like to thank my supervisor Fosca Giannotti: her guide started during my Master thesis and continued during my Ph.D. During all these years she has helped me in finding interesting problems, guiding me to their solutions, always encouraging me to ask the best to myself.

Second, I am particularly grateful to Francesco Bonchi, whose advises and collaboration have always led to good results and achievements, from my Master thesis to this Ph.D thesis. My admiration for him started many years ago and still keeps being alive.

I would express also my best gratitude to all my co-authors with which I share part of the results of this thesis: Björn Bringmann, Michele Coscia, Aristides Gionis, Mirco Nanni and Fabio Pinelli.

My gratefulness goes also to the reviewers of the thesis for their useful and positive comments: Aristides Gionis, Dimitrios Gunopulos and Christophe Rigotti.

I would like to thank all my friends and the people I met so far: either in a direct or indirect way, all of them contributed to this result.

The achievement of this important result would not have been possible without the continuous support of all my family, which always encouraged me in any moment, and helped me whenever I needed it. This thesis is dedicated to them.

A warm thank to my girlfriend Maddalena, whose support during all these years was very precious, and with which I've shared all the pains and successes I've been living so far.

Vita

October 10, 1980	Born in Bari, Italy
August 2002-December 2002	Erasmus Student
	University of Uppsala, Sweden
December 2005	Degree in Informatica (Computer Science)
	Final mark: 110/110
	University of Pisa, Italy
Since February 2006	Phd Student
	IMT Lucca Institute for Advanced Studies, Italy
July 2006	Student
	Proteoms and Proteins Summer School
	Lipari Island, Italy
September 2006	Student
	KDUbiq Summer School
	Dortmund, Germany
June 2007	Student
	SADA Summer School
	Helsinki, Finland
Since 2008	Research Associate
	KDD-Lab, ISTI-CNR
	Pisa, Italy
May 2008 - February 2009	Internship
	Yahoo! Research
	Barcelona, Spain
September 2009	Visiting
	UBC, prof. Laks V.S. Lakshmanan
	Vancouver, Canada

Publications

1. Michele Berlingerio, Francesco Bonchi, Silvia Chelazzi, Michele Curcio, Fosca Giannotti, Fabrizio Scatena: "Mining HLA Patterns Explaining Liver Diseases." In Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems (CBMS 2006), 22-23 June 2006, Salt Lake City, Utah, USA. 702-707
2. Michele Berlingerio, Francesco Bonchi, Fosca Giannotti: "Towards Constraint-Based Graph Mining." In Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD 2007, 17-20 June 2007, Torre Canne, Fasano, BR, Italy. 274-281
3. Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, Franco Turini: "Time-annotated Sequences for Medical Data Mining." In Proceedings of The IEEE International Workshop of Data Mining in Medicine 2007 (DMMed '07 in conjunction with ICDM'07), 28 October 2007, Omaha, Nebraska. 133-138
4. Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, Franco Turini: "Mining Clinical Data with a Temporal Dimension: a Case Study." In Proceedings of The 1st IEEE International Conference on Bioinformatics and Biomedicine (BIBM 2007), 2-4 November 2007, San Jose, California. 429-436
5. Michele Berlingerio, Fosca Giannotti, Mirco Nanni, Fabio Pinelli: "Temporal analysis of process logs: a case study." In Proceedings of the Sixteenth Italian Symposium on Advanced Database Systems, SEBD 2008, 22-25 June 2008, Mondello, PA, Italy. 430-437
6. Michele Berlingerio, Michele Coscia, Fosca Giannotti: "Mining the Information Propagation in a Network." In Proceedings of the Seventeenth Italian Symposium on Advanced Database Systems, SEBD 2009, 21-24 June 2009, Camogli, GE, Italy. 333-340
7. Michele Berlingerio, Fabio Pinelli, Mirco Nanni, Fosca Giannotti: "Temporal mining for interactive workflow data analysis." In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, June 28 - July 1, 2009, Paris, France. 109-118
8. Michele Berlingerio, Francesco Bonchi, Michele Curcio, Fosca Giannotti, Franco Turini: "Mining Clinical, Immunological, and Genetic Data of Solid Organ Transplantation." Book Chapter In "Biomedical Data and Applications", Amandeep S. Sidhu and Tharam S.Dillon Ed., Studies in Computational Intelligence, Volume 224/2009, Springer, 2009.

9. Michele Berlingerio, Michele Coscia, Fosca Giannotti: "Mining the Temporal Dimension of the Information Propagation." In Proceedings of the 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France. 237-248

10. Michele Berlingerio, Francesco Bonchi, Bjoern Bringmann, Aristides Gionis: "Mining Graph Evolution Rules." In Proceedings of Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD 2009, Bled, Slovenia. 115-130

Abstract

In the last years, there have been many studies on analyzing network and graph data. A wide range of problems, such as studying the global and local properties of a graph, finding interesting structures, modeling particular characteristics, assessing the properties of some particular networks such as the Web or a co-authorship networks, have increased the attention of the scientific community, involved in finding efficient and powerful techniques to enable the achievement of the desired results. For example, with the aim of finding interesting and frequent substructures in graphs, algorithms such as AGM, FSG, gSpan, Gaston, FFSMY, ADI-Mine, HSIGRAM and VSIGRAM have been presented for improving scalability on mining subgraphs one after one.

However, only in the last few years the attention has moved to a particular aspect of graphs and networks: the temporal dimension. Thanks also to the larger availability of online social network services, the amount of data that allows for the analysis of the dynamics of complex networks has increased very fast in the last five years. This kind of data contains rich information about what happens to a network during time, and enables the analysts to model and discover interesting properties related to the temporal dimension, which are both meaningless and impossible in the static setting.

The temporal dimension can play a double role for a network. First, the underlying structure, namely the graph, can evolve over time, showing new users joining a community, new connections created among users, change of properties of a particular group of people, and so on. Second, given an established

network, users may perform actions during time, leading to flows of information circulating among the connections, sequences of tasks performed by a sequence of users, spread of influence among the network, and so on.

Despite the clear richness of the above setting, the current graph mining techniques are somehow too generic, and they do not explicitly take into consideration the time during their stages.

In order to overcome to this problem, in this thesis we study the current graph mining algorithms, we study the possibility of pushing constraints during the computation that would allow us to efficiently analyze the temporal dimension at mining stage, and we develop new techniques that can help in this kind of analysis.

In order to prove the effectiveness of our approach, we apply a pre-existent graph miner, a modified version of it specialized to deal with the temporal dimension, and another pre-existent tool of analysis, namely the Temporally Annotated Sequences framework, to real data, to show how we can deal with the above setting, with particular focus on problems such as mining the information propagation in a network, mining graph evolution rules, and mining the temporal dimension of process logs to derive the actual workflow diagram in a process.

Our results justify the need for this approach, and show that specialized techniques help in modeling and analyzing temporal graph and network data.

Chapter 1

Introduction

1.1 Context and Motivation

With the advent of automated data collection tools, our ability to generate and collect data have increased rapidly in the last few decades. This explosive growth in data has opened the possibility of extracting useful information and knowledge from the data. Data mining helps in discovering non-trivial patterns on a stored data and hence more research is being done in this field to extract useful information from large amounts of collected data. Most often the data of interest is very complex and it is common to model it with the help of graphs consisting of nodes and edges that are often labeled to store additional information. Graphs are hence useful to model networks of users, or entities, connected by any kind of relationship: two users can be friends, relative, colleagues, they can like the same kind of music or movies, they are geographically close to each other, the share the same customer profile, and so on.

This kind of rich data, moreover, can be analyzed considering another important dimension: the time. While, so far, much attention was devoted to the analysis of static graphs, such as identifying communities in a network of users, identifying frequent segments of proteins sharing a subset of functionalities, analyzing the Web Graph, and so on, only in the last few years the scientific community has realized that the dynamic as-

pects of graph data are of large interest, and allow for a richer and wider analysis, that can not be performed on static data. When considering, for example, an online social network, where people keep in touch with each other, interact exchanging information, commenting the actions of other users, sharing interests, joining communities, and so on, it is clear how a static analysis can not model the classical cause-and-effect schema properly, as the temporal information about the sequence of events is not definied and can not be pushed into the analysis stages.

Generally speaking, there can be two temporal settings when dealing with graph data:

- **Action.** In this setting, users perform actions along time: they may pass information among the network, they may perform tasks independently from each other, they may look at the actions of their neighbors and get influenced by performed similar actions, and so on.

- **Evolution.** Here the structure of the network may change over time: new users may join a community, they can break the connection with a set of friends or colleagues, they can quit a specific network, new components of the network may appear independently, they may form a single core after a specific amount of time, and so on.

Moreover, the two settings may coexist: this is exactly the case of real-world online social networks: a network of connections among users evolve over time, typically growing, and those users usually interact with their friends or with other users of the network. As a result of this, often in the literature we find the information about the social connections separated from the list of actions performed.

Furthermore, the temporal dimension of a network can also induce different *views* of the same data, or may suggest the presence of hidden layers of networks among the users. As a result of this, often the social connections themselves can be learned based on the observations of the actions performed by the users.

All the above considerations suggest questions such as:

- How does a network evolve over time?
- How will a specific network look like in 2 years?
- How fast the information can spread among users?
- What are the characteristics of the users that spread the information faster?
- Is there a relationship among the structure of a network, the types of actions performed by its users, the time needed by the network to evolve, and the time in which the information can reach all the users?

1.2 Thesis Contribution

In this thesis, we address the above questions, trying to catch the intrinsic relations among users, structure, and time, by means of *Graph Mining* and *Temporally Annotated Sequences* (TAS) techniques. The first aims at finding frequent subgraphs in graph data, and it is of particular help in finding communities, detecting properties of users in a community, detecting changes in the structure of a network, and so on. The second aims at discovering frequent sequences of events during time, for which the transition time between any consecutive pair of events was automatically found frequent in the data, making the temporal analysis of action logs and information flows possible.

However, while the second of the two techniques was explicitly intended to deal with the temporal dimension, and does not require any further adaptation to deal with most of the available data, the current available Graph Mining tools do not take into account the evolution, thus we need to find new ways of performing such analysis by extending the state of the art of this area with new more powerful tools.

For this purpose, in this thesis we study the main problems of *Frequent Subgraph Mining* and *Constraint-Based Frequent Subgraph Mining*. After formally defining the two problems, we give an extensive state of the art of the current approaches, we show an algorithm for pre-processing

graph data according to a conjunction of constraints, and we give a generic approach for solving the two problems. We then show how to combine Graph Mining and TAS Mining techniques in order to analyze the flows of information in a real-world network, we present a specialized version of a graph miner capable of dealing with the temporal dimension of graph and network data, then we show a possible application of TAS mining to workflow data, where the goal is to contruct the actual workflow diagram, annotated with transition times between tasks found frequent in the data.

The original contribution of this thesis hence includes: the study of the literature in analyzing graph and network data with the temporal dimension, the study of constraints on graph data with their main properties; the definition, implementation and experimental evaluation of a preprocessing algorithm for the constraint-based graph mining problem; the definition of an algorithm for pushing contraints deep into the computation; the application of this algorithm to real life problem such as mining the information propagation in a network, and mining frequent patterns in an evolving graph, for which the generic approach to graph mining has been specialized; the study of an application of the opposite problem, i.e. reconstruncting of a graph from local patterns, to a real-life problem such as workflow mining, the application of the existing TAS mining framework to networks showing a temporal dimension.

1.3 Thesis Organization

The remainder of the thesis is organized as follows: chapter 2 reviews the state of art on analyzing the temporal dimension of graph and network data; chapter 3 presents the graph mining technique, and gives original contribution to it; in chapter 4 we show how to combine graph mining technique to TAS mining technique for mining the information propagation in a network; in chapter 5 we deal with evolving graphs, for which we define a specialization of the approach proposed in chapter 3, and we define graph evolution rules; chapter 6 shows the problem of reconstructing a temporally-annotated graph from local patterns, in workflow

mining; finally, chapter 7 summarizes the conclusions of the thesis.

Chapter 2

Mining the Temporal Dimension of Graph and Network Data

In this Chapter we review some of the current analyses performed in the literature to deal with the temporal with the temporal dimension in graph and network data. Following the two settings defined in Chapter 1, we give an overview of the approaches in each of the two cases.

2.1 Action

When speaking about actions performed on a network, there are actually several possible scenarios. Users, in fact, can *interact*, by propagating information, exchanging emails, performing joint actions (such as playing a football match), commenting the same pictures on an online photo sharing service, and so on. Often we are not even provided with the social graph on which these kind of interactions rely, and one possibility is to infer this graph based on the observation of the actions performed: any jointly performed task, any direct pass of information, any simultaneous action, tell us something about the users involved. Based on our observation, we may want to consider these users connected, obtaining, at the

end, a possible hidden social graph among them.

In sections 2.1.1 and 2.1.2 we give examples of two possible settings. In the first, the focus is on analyzing the propagation of information in a network. In some cases the network connections are learned during the analysis of the flows of information. In the second, users perform actions independently from each other, there is no explicit connection among them, but the focus is on finding the connections among the tasks performed, as the setting under investigation is that of workflow mining.

2.1.1 Information Propagation

During the last years, several approaches have been proposed addressing the problem of analyzing how the information propagates in a network (2; 3; 12; 24; 40; 66; 87; 98).

In (87), Tyler et al. develop an automated method applying a betweenness centrality algorithm to rapidly identify communities, both formal and informal, within the network. In this work, the authors learn the communities from the email interaction, i.e. they simply consider the senders and the receivers of the emails as nodes, connecting them in the derived social graph if they exchanged emails with each other. This approach also enables the identification of leadership roles within the communities. The automated analysis was complemented by a qualitative evaluation of the results in the field.

In (98), Wu et al. analyze email to model the flows of information in social communities, taking into account the observation that an item relevant to one person is more likely to be of interest to individuals in the same social circle than those outside of it. This is due to the fact that the nodes that are close in the social network tend to have similar properties. An epidemic model on a scale-free network with this property has a finite threshold, implying that the spread of information is limited. These hypotheses were tested by measuring the diffusion of messages in an organization and also by experiments that take into consideration the organizational distance among individuals. Since social structure affects the flow of information, knowing the communities that exist within a net-

work can also be used for browsing a network when looking for specific individuals or specific topics.

Adamic and Adar do exactly this in (2), by simulating a small world experiment on the HP Labs email network. The small world experiment has been carried out a number of times in the last years, always demonstrating that individuals passing messages to their friends can form a short chain between two people separated by geography, profession, or other attributes. While the existence of these chains has been established, how people are able to navigate this kind of data without knowing the complete social networks residing under the model has remained an open question. Recently, models have been proposed to explain the phenomenon, and the work of Adamic and Adar is a first study to test the validity of these models on a social network.

In (3), Adamic et al. studied the propagation of information through blogspace, both to uncover general trends and to explain specific instances of URL transmission. They determine general categories of popularity, ranging from sustained interest to short lived events, through the use of cluster analysis. By analyzing the routes of individual URLs they built a tool able to visualize and explain how information travels. Both the availability and quantity of temporal information is unique in blog data. Using it they were able to not only infer link structure, but also to create a novel ranking algorithm, iRank, for ranking blogs. Whereas traditional ranking strategies rely primarily on explicit link structure, iRank successfully folds in implicit paths of propagation to find blogs that are at the source of information.

In (24), the model of timestamped graph and digraph are introduced in order to study the influence in a network. In particular, the authors studied the associated influence digraphs of time-stamped graphs, focusing on their realizability. Their motivating examples of collaboration graphs are undirected, as an edge represents a collaboration between two authors, and hence each is influenced by the other. In some situations, however, a collaboration between two persons may induce a one-way influence; for example a student may be influenced by but not influence her/his supervisor. In this case, the time-stamped graph may be a di-

rected graph, with parallel arcs allowed.

Leskovec et al. analyze in (66) the problem of Viral Marketing with several different statistical approaches. They present an analysis of a person-to-person recommendation network, consisting of 4 million people who made 16 million recommendations on half a million products. They observe the propagation of recommendations and the cascade sizes, which they explain by a simple stochastic model. They analyze how user behavior varies within user communities defined by a recommendation network. Despite the fact that it is frequently assumed that in epidemic models individuals have an equal probability of being infected every time they, the authors observe that the probability of infection decreases with repeated interaction. In the context of marketing, they also found that the probability of purchasing a product increases with the number of recommendations received but quickly saturates to a constant and relatively low probability. This means individuals are often impervious to the recommendations of their friends, and resist buying items that they do not want. They hence establish how the recommendation network grows over time and how effective it is from the viewpoint of the sender and receiver of the recommendations. While on average recommendations are not very effective at inducing purchases and do not spread very far, they present a model that successfully identifies communities, product, and pricing categories for which viral marketing seems to be very effective.

Kossinets et al., in (59), formulate a temporal notion of "distance" in the underlying social network by measuring the minimum time required for information to spread from one node to another. That is, they exploited the possibility of considering the connections as weighted, making thus explicit the *potential* for the information to flow. In this model, then, some of the direct connections in the network become much longer, due to low rates of communication, while other multi-step paths become much shorter, due to the rapidity with which information can flow along them. Moreover, they define the network backbone to be the subgraph consisting of edges on which information has the potential to flow the quickest. They find that the backbone is a sparse graph with a concentra-

tion of both highly embedded edges and long-range bridges.

In (67), the authors, rather than concentrating in the speed of communication, they put emphasis on how much a topic can spread. They find that rather than fanning out widely, reaching many people in very few steps according to "small-world" principles, the progress of chain letters proceeds in a narrow but very deep tree-like pattern, continuing for several hundred steps. This suggests a new and more complex picture for the spread of information through a social network. They describe a probabilistic model based on network clustering and asynchronous response times that produces trees with this characteristic structure on social-network data.

Goyal et al., in (40), introduce a novel frequent pattern mining approach to discover leaders and tribes in social networks. In their scenario, they assume an known underlying social graph, and an action table containing all the actions performed by the users in the graph. They assume that users act based on the feeds of other users, calling this effect *influence*. They define the two problems of finding users that influence a given number of users (leader discovery), and its tribe version, where the users influenced must by the same for each action (tribe leader discovery).

2.1.2 Workflow Mining

In the past few years, research has been performed on discovering a process model from a set of process instances. Most of the work done assume the existence of a process model underlying the given set of process instances. Workflow mining, or process mining, aims at the automatic discovery of a process diagram, also called workflow or process schema, from process logs. In this diagram nodes represent the performed tasks, while there is a (directed) edge between two tasks if the two appeared to be executed consecutively in a process log. Several different approaches have been proposed to solve such a problem (5; 6; 13; 27; 29; 43; 49; 81; 91; 95). The idea of applying process mining in the context of workflow management was first introduced in (5) by Agrawal et al. This work is based on workflow graphs, which are inspired by workflow products

such as IBM MQSeries workflow (formerly known as Flowmark) and InConcert. In this paper, two problems are defined. The first problem is to find a workflow graph generating events appearing in a given workflow log. The second problem is to find the definitions of edge conditions. A concrete algorithm is given for tackling the first problem. The approach is quite different from other approaches, as the authors claim that there is no need to identify the nature (AND or OR) of joins and splits, that are the possible dependencies among any two tasks. In (27), Cook et al. describe three different methods for process discovery: the one uses neural networks, the second purely algorithmic approach, and the third uses a Markovian approach. The authors consider the latter two the most promising approaches. The purely algorithmic approach builds a finite state machine (FSM) where states are fused if their futures (in terms of possible behavior in the next k steps) are identical. The Markovian approach uses a mixture of algorithmic and statistical methods and is able to deal with noise. In (6; 49) directed graphs are used, and in both papers, the researchers consider tasks that can be executed in parallel. These are actually the first papers in which *temporal dependencies* of different tasks are also modeled. However, in (49) the notion of parallelism between tasks is more sophisticated, i.e. they go beyond the simple temporal dependency between tasks that was treated in (6): they define *overlapping* and *disjointed* activities. Finite state machines are proposed in (29), and Petri-nets are used in (90) for representing process instances. Hwang et al., in (48), define the concept of *temporal graphs*, which help in modeling the dependencies among the performed tasks during specific instances of the processes. They propose three different algorithms that work with temporal graphs, itemsets or sequences. These algorithms solve the *temporal pattern discovery* problem, defined as the discovery of the maximal temporal graphs among all frequent temporal graphs. In this work, the authors consider the starting and ending time of each task to explicitly detect situations of parallelism or choice between pairs of tasks. However, they do not look for frequent transition times or execution times of the tasks, i.e. they use the temporal dimension only to detect the temporal dependencies between tasks. Greco et al. deal in (41) with the problem of

mining *unconnected* patterns in workflows, i.e. detecting sets of activities that are frequently executed together and do not exhibit explicit dependent relationships. They present two different algorithms for solving the problem. This paper uses the concept of *frequency* of a pattern.

2.2 Evolution

Several papers have focussed on the global evolution of networks by an exploratory point of view. Leskovec et al. (65) discovered the shrinking diameter phenomena on time-evolving networks. Backstrom et al. (8) study the evolution of communities in social networks. Still from an exploratory perspective, Leskovec et al. (64) study the evolution of networks but at a more local level. Using a methodology based on the maximum-likelihood principle, they investigate a wide variety of network formation strategies, and show that edge locality plays a critical role in evolution of networks.

Other recent papers, present algorithmic tools for the analysis of evolving networks. Tantipathananandh et al. (86) focus on assessing the community affiliation of users and how this changes over time. Sun et al. (84), apply the MDL principle to the discovery of communities in dynamic networks, developing a parameter-free framework. This is the main difference to previous work such as (4; 85). However, as in (86), the focus lies on identifying approximate clusters of users and their temporal change. No exact patterns are found, nor is time part of the results obtained with these approaches. Ferlez et al. (33) use the MDL principle for monitoring the evolution of a network.

While the aforementioned body of work studies the evolution in networks, it does not take a pattern mining approach as we do in this thesis. Desikan and Srivastava (31) study the problem of mining temporally evolving web graphs. Three levels of interest are defined: single node, subgraphs and whole graph analysis, each of them requiring different techniques. They study changes of properties on each of the three levels under investigation. Inokuchi and Washio (50) propose a fast method to mine frequent subsequences from graph sequence data defining a for-

malism to represent changes of subgraphs over time. However the time in which the changes take place is not specified in the patterns. Liu et al. (68) identify subgraphs changing over time by means of vertex-importance scores and vertex-closeness changes in subsequent snapshots of the graphs. The most relevant subgraphs are hence not the most frequent, but the most significant based on the two defined measures. Another interesting paper is (19),where Borgwardt et al. represent the history of an edge as a sequence of 0's and 1's representing the absence and presence of the edge respectively. Then conventional graph-mining techniques are applied to mine frequent patterns. However, there are several insights in their approach. First, the employed mining algorithm GREW is not complete, but heuristic. Further, the overlap-based support measure used requires solving an maximal independent set problem for which a greedy algorithm is used. Another computational issue with their approach (next to the overlap-graph and the subsequent MIS problem), is the extension of an edge in the so-called inter-asynchronous FDS case. Accordingly the size of the networks analyzed in the paper is rather small.

Chapter 3

Mining Graph Data

In this chapter we present the basic insights of Graph Mining. This technique is the most popular way to handle graph data, when looking for frequent patterns. It constitutes also the basis for the remainder of the thesis, where extensions of the current existing approaches, presented in this chapter, are used as underlying framework to mine the temporal dimension of graph data. In this chapter the problems of *Frequent Subgraph Mining* and *Constraint-Based Frequent Subgraph Mining* are formally defined, together with various classes of constraints with their properties. The chapter presents also an overview of several approaches to the frequent subgraph mining problem. The chapter is organized as follows: section 3.1 gives the preliminaries definitions, introducing the reader to the graph data we are interested in; section 3.2 presents the concept of constraint on graph data, together with a list of possible interesting constraints on graphs with their main properties; section 3.3 presents the two main problems we treat in this chapter; section 3.4 reviews the state of the art of the current approaches to Graph Mining; the two above problems are studied in section 3.5 in the transactional setting and in section 3.6 in the single graph setting.

3.1 Preliminary concepts

As most of the work on frequent subgraph mining, the focus is on undirected connected labeled graphs without multiple edges.

Definition 1 (Labeled Graph) *A labeled graph is a 4-tuple $G = (V, E, L, l)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, L is a set of labels, and $l : V \cup E \rightarrow L$ is a labeling function that assigns a label to an edge or a vertex.*

Definition 2 (Connected Graph) *A graph G is called* connected *if for any vertices $u, v \in V$, there exist vertices $w_1, \ldots, w_n \in V$ such that $\{(u, w_1), (w_1, w_2), \ldots, (w_{n-1}, w_n), (w_n, v)\} \subseteq E$.*

In the remainder of the thesis, we assume to work on a *dataset* of graph data. We discern between the *transactional* setting (also known as graph-transaction setting), i.e. where the dataset is a collection of usually small graphs, each one representing a transaction, and the single graph setting, i.e. where the dataset is a single usually large graph. Examples of the first setting are a dataset of proteins, or chemical compounds, while examples of the single graph case are large networks such as the Web, a Social Network, a graph of interactions between proteins, and so on.

One of the most important issues in Graph Mining is to find an occurrence of a pattern, i.e., to solve the Subgraph Isomorphism Problem.

Definition 3 (Subgraph isomorphism) *A subgraph isomorphism from G' to G is an injective function $f : V' \rightarrow V$ such that: (1) for any vertex $u \in V'$, $f(u) \in V$ and $l'(u) = l(f(u))$; and (2) for any edge $(u, v) \in E'$, $(f(u), f(v)) \in E$ and $l'(u, v) = l(f(u), f(v))$.*

Definition 4 (Subgraph) *A graph $G' = (V', E', L', l')$ is a subgraph of a graph $G = (V, E, L, l)$ (denoted as $G' \subseteq G$), if there exists a subgraph isomorphism from G' to G.*

Note that while the graph isomorphism problem (i.e. the problem of deciding wheter two graph have identical topological structure) has un unknown computational complexity (except for a few particular cases for which it is proven to be polynomial), the subgraph isomorphism problem is known to be NP-complete.

3.2 Constraints on Graph Data

This section introduces the kind of constraints that we encounter in the remainder of the thesis, and discusses their main properties (anti-monotonicity or monotonicity). Since the proofs of these properties are always straightforward, they are not provided. The variety of constraints presented is not meant to be exhaustive, since meaningful constraints will be suggested by the application at hand: the objective here is to show that most of the fundamental properties of graphs can be modelled as either monotone or anti-monotone constraints.

Definition 5 (Constraint) *Let $\mathbb{G}$ be the domain of all possible connected graphs, a constraint on graph patterns is a function $\mathcal{C} : \mathbb{G} \rightarrow \{true, false\}$. We denote $Th(\mathcal{C}) = \{G \in \mathbb{G} | \mathcal{C}(G) = true\}$ the set of all graphs satisfying a constraint $\mathcal{C}$.*

One of the most important constraint is the *frequency*. We give here a general definition of the frequency constraint, defined on the basis of a *support* (sup) function. As we see in the rest of the thesis, defining a convenient support function is a crucial step when mining graph data, and it turns out to be easy in the transactional setting, but not trivial in the single graph setting.

Definition 6 (Frequency constraint) *Given a minimum support threshold σ, the frequency constraint $\mathcal{C}_{freq[\mathcal{D},\sigma]}$ is satisfied by all the graphs G such that $\sup(G) \geq \sigma$.*

In the following there is a distinction between *topological constraints*, i.e., those constraints regarding only the topological structure of a graph pattern, and *labeled graph constraints*, i.e., those constraints that involve not only the structure of a graph pattern but also its labels. In particular, the focus is on the problem of pushing *monotone* constraints in the frequent pattern computation. A monotone constraint $\mathcal{C}_M$ is such that, if satisfied by a graph G, then it is satisfied by any connected supergraph of G. The frequency constraint $\mathcal{C}_{freq}$ instead behaves the opposite: if satisfied by a graph G, then it is satisfied by any connected subgraph of G. Constraints that behave as $\mathcal{C}_{freq}$ are said *anti-monotone* and denoted $\mathcal{C}_{AM}$.

Topological Constraints

Given a graph $G = (V, E, L, l)$, a topological constraint is such that, if satisfied or not by G can be decided on the basis of just the V and E components of G. The first simple example of topological constraint is the constraint on the size, in terms of number of vertices or edges, that a substructure should have in order to qualify as interesting pattern.

Definition 7 (Size Constraint) *The size constraint is denoted $\mathcal{C}_{size[S,\theta,\alpha]}$, where $S \in \{V, E\}$, $\theta \in \{\geq, \leq\}$, and $\alpha \in \mathbb{N}$. It is satisfied by a graph G iff $|S| \, \theta \, \alpha$. It is trivial to see that the size constraint $\mathcal{C}_{size[S,\leq,\alpha]}$ is anti-monotone, while $\mathcal{C}_{size[S,\geq,\alpha]}$ is monotone.*

Often in molecular biology one is not interested in generic substructures, but in rigid structures such as chains and rings. These can be expressed as topological constraints.

Definition 8 (Cycle Constraint) *Given $\alpha \in \mathbb{N}$, the cycle constraint $\mathcal{C}_{cycle[\alpha]}$ is satisfied by $G = (V, E, L, l)$ iff G contains a simple cycle of size $\geq \alpha$, i.e., there exist $n \geq \alpha$ distinct vertices $v_1, \ldots, v_n \in V$ such that $\{(v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, v_1)\} \subseteq E$. The cycle constraint $\mathcal{C}_{cycle[\alpha]}$ is monotone.*

One could be interested in mining frequent *acyclic* substructures. This topological constraint can be obtained by negating a cycle constraint, i.e., we can define $\mathcal{C}_{acyc} \equiv \neg \mathcal{C}_{cycle[2]}$. Since the negated of a monotone constraint is anti-monotone (and viceversa), $\mathcal{C}_{acyc}$ is anti-monotone. Since only connected substructures are considered, also the *chain constraint* $\mathcal{C}_{chain}$, which is satisfied only by graphs that are chains, is anti-monotone: any connected substructure of a chain is still a chain, or the other way around, if a graph is not a chain, all its supergraphs are not chains as well. Many other constraints, which are either monotone or anti-monotone, can be defined on the topological structure of the graph patterns we are mining: the point is to individuate the meaningful constraints for the given application. As an example, here the *girth* and the *circumference* constraints are provided.

Definition 9 (Girth and Circ Constraints) *The girth of a graph is the length of a shortest simple cycle in the graph; while the circumference is the length of*

a longest simple cycle. Given $\theta \in \{\geq, \leq\}$ and $\alpha \in \mathbb{N}$, the girth constraint is denoted $\mathcal{C}_{girth[\theta,\alpha]}$, while the circumference constraint is denoted $\mathcal{C}_{circ[\theta,\alpha]}$. The constraint $\mathcal{C}_{girth[\leq,\alpha]}$ is monotone, while $\mathcal{C}_{girth[\geq,\alpha]}$ is anti-monotone; conversely, the constraint $\mathcal{C}_{circ[\leq,\alpha]}$ is anti-monotone while $\mathcal{C}_{circ[\geq,\alpha]}$ is monotone.

Labeled Graph Constraints

Given a graph $G = (V, E, L, l)$, a labeled graph constraint is such that, if satisfied or not by G can not be decided simply on the basis of its topology (the V and E components), but also the labels are needed. The first simple example of constraint on labels specifies which particular elements (both vertices or edges labels) must be contained or not in the patterns.

Definition 10 (Labels Constraint) *Let L' be a set of labels, and $G = (V, E, L, l)$ a graph. The following four constraints on vertices labels are defined:*

- $\mathcal{C}_{vrtx[L',\subseteq]}(G) \equiv \forall l \in L', \exists\, v \in V : l(v) = l.$
- $\mathcal{C}_{vrtx[L',\supseteq]}(G) \equiv \forall v \in V,\, l(v) \in L'.$
- $\mathcal{C}_{vrtx[L',\cap]}(G) \equiv \exists l \in L', \exists\, v \in V : l(v) = l.$
- $\mathcal{C}_{vrtx[L',\not\ni]}(G) \equiv \forall l \in L',\, \nexists\, v \in V : l(v) = l.$

Similarly, the same four constraints are defined for edges labels (using the notation $\mathcal{C}_{edge}$).

It is straightforward to see that: $\mathcal{C}_{vrtx[L',\subseteq]}(G)$ is monotone, $\mathcal{C}_{vrtx[L',\supseteq]}(G)$ is anti-monotone, $\mathcal{C}_{vrtx[L',\cap]}(G)$ is monotone, and $\mathcal{C}_{vrtx[L',\not\ni]}(G)$ is anti-monotone. The same properties hold for constraints on edges labels.

Example 1 *Suppose that from a database of molecules $\mathcal{D}$ we want to mine frequent ($\sigma = 30$) substructures, that are made only of vertices labeled with atom types* O, N *and* C*, that contain at least one double bond (i.e., an edge with label 1), and that contain at least a ring of size at least 4. This query can be expressed as:*

$$Th(\mathcal{C}_{freq[\mathcal{D},30]} \wedge \mathcal{C}_{vrtx[\{\mathrm{O,N,C}\},\supseteq]} \wedge \mathcal{C}_{edge[\{1\},\subseteq]} \wedge \mathcal{C}_{cycle[4]})$$

The first two constraints in the conjunction are anti-monotone, while the second two are monotone: this is the kind of query that we address here.

Given a graph G' one could be interested in mining only patterns that are supergraph (or subgraph) of G'.

Definition 11 (Sub/Super-graph Constraints) *Given a graph G',*

- *the constraint $\mathcal{C}_{subg[G']}(G) \equiv G \subseteq G'$ is anti-monotone*
- *the constraint $\mathcal{C}_{supg[G']}(G) \equiv G \supseteq G'$ is monotone.*

Also constraints based on aggregates of the labels (either of vertices or edges) can be defined. Such aggregates can be computed directly over the labels, or on some attributes of the labels. In the second case, such attributes must be in functional dependency with the labels. For instance, suppose that labels of vertices are atom types, then it is possible to define a constraint on the *sum of atomic weights*, where *atomic weight* is an attribute of the label atom type, in functional dependency (i.e., the same atom type must have a unique atomic weight all over the database).

Example 2 *Suppose that from a molecules database $\mathcal{D}$ the user wants to mine frequent ($\sigma = 30$) substructures, that contain at least 3 atoms of oxygen, and have sum of atomic weights larger than 120. This query can be expressed as:*

$$Th(\mathcal{C}_{freq[\mathcal{D},30]} \wedge \mathcal{C}_{count[V=\mathrm{O},\geq,3]} \wedge \mathcal{C}_{sum[V.weight,\geq,120]})$$

Definition 12 (Aggregate Constraints) *An aggregate constraint is denoted $\mathcal{C}_{aggr[\varphi,\theta,\alpha]}$, where:*

- $aggr \in \{count, min, max, sum\}$;
- $\varphi \in \{S, S.a, S\,\vartheta\,l, S.a\,\vartheta\,v\}$;
- *$S.a$ is an attribute of S and v one value;*
- $\vartheta \in \{=, \neq, \geq, \leq\}$;
- *l is a label of S;*
- $S \in \{V, E\}$, $\theta \in \{\geq, \leq\}$, *and* $\alpha \in \mathbb{R}$

Table 1 resumes all the constraints introduced in this section and their properties.

Proposition 1 *The constraint $\mathcal{C}_{sum[\varphi,\leq,\alpha]}$ is anti-monotone and $\mathcal{C}_{sum[\varphi,\geq,\alpha]}$ is monotone, if φ has non negative values. Otherwise they are neither anti-monotone nor monotone. All the other constraints listed in Table 1 are anti-monotone or monotone according to the Table.*

Constraint	Anti	Mono
$size[S, \leq, \alpha]$	✓	
$size[S, \geq, \alpha]$		✓
$cycle[\alpha]$		✓
$acyc$	✓	
$chain$	✓	
$girth[\leq, \alpha]$		✓
$girth[\geq, \alpha]$	✓	
$circ[\leq, \alpha]$	✓	
$circ[\geq, \alpha]$		✓
$vrtx[L, \subseteq]$		✓
$vrtx[L, \supseteq]$	✓	
$vrtx[L, \cap]$		✓
$vrtx[L, \not\cap]$	✓	
$edge[L, \subseteq]$		✓
$edge[L, \supseteq]$	✓	
$edge[L, \cap]$		✓
$edge[L, \not\cap]$	✓	
$subg[G]$	✓	
$supg[G]$		✓
$count[\varphi, \leq, \alpha]$	✓	
$count[\varphi, \geq, \alpha]$		✓
$min[\varphi, \leq, \alpha]$		✓
$min[\varphi, \geq, \alpha]$	✓	
$max[\varphi, \leq, \alpha]$	✓	
$max[\varphi, \geq, \alpha]$		✓
$sum[\varphi, \leq, \alpha]$	✓	
$sum[\varphi, \geq, \alpha]$		✓

Table 1: Constraints and their properties.

We have introduced a set of simple generic constraints, which is not meant to be complete. It is possible to define other constraints on graph data, and we expect the applications to suggest new interesting ones. Moreover, monotonicity and anti-monotonicity are not the only interesting properties of constraints: for the itemset, for example, other properties such as loose anti-monotonicity (16), succinctness (72) and convertibility (77; 78) have been defined. However, the intent in this chapter was to provide a general idea, while ad-hoc solutions can be studied and implemented when applying a specialized framework for a specific prob-

lem. Chapter 5, for example, show how to deal with the *evolution* of a graph, by defining a new constraint that requires a particular implementation.

3.3 Mining Frequent Subgraphs

At this point, we can define the two main problems under analysis in this thesis (we omit $\mathcal{D}$ and σ when reasonable).

Definition 13 (Frequent Subgraph Mining)

$$Th(\mathcal{C}_{freq})$$

As we see in section 3.4, many approaches have been proposed so far to tackle this challenging problem from many points of view.

If the substructures have to satisfy also a conjunction of other constraints $\mathcal{C}$, we obtain the *constraint-based frequent subgraph mining problem*:

Definition 14 (Constraint-based Frequent Subgraph Mining)

$$Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$$

As we have stated above, in order to be these two problems correctly defined, we still have to define an appropriate support function. This is done in sections 3.5 and 3.6, where we treat the transactional and the single graph settings, respectively, as they need different support functions.

3.4 State of the Art on Graph Mining

The earliest studies to find subgraph patterns characterized by some measures from massive graph data were conducted by Cook and Holder (25) and Yoshida and Motoda (GBI) (100) in the middle of the 1990's. Their approaches used greedy search to avoid high complexity of the graph isomorphism problem, which resulted in an incomplete set of characteristic subgraphs. In 1999, Dehaspe and Toivonen proposed an ILP-based algorithm, WARMR, enabling a complete search for frequent subgraphs

from graph data (30). Subsequent work done by Nijssen and Kok proposed a faster algorithm, FARMAR (73). In 2000, Inokuchi et al. proposed an approach called AGM to combine Apriori algorithm and mathematical graph theory (51). In 2001, De Raedt and Kramer proposed the version space based approach called MolFea to find characteristic paths from the graph data (60). Based on these pioneering studies, several other approaches were proposed to tackle the challenging problem of finding frequent subgraphs in graph data, or other interesting related problems such as graph similarity search, graph indexing, graph compression, problems related to social network analysis, and so on.

From the point of view of the search strategy, they can all be categorized into heuristic search methods and complete search methods in terms of the completeness of search. They can also be categorized into direct and indirect matching methods from the view point of the subgraph isomorphism matching problem. The indirect matching does not solve the subgraph isomorphism problem but subgraph similarity problem under some similarity measure.

From a more high-level point of view, all the approaches can be categorized into four groups: greedy search based approaches, inductive logic programming (ILP) based approaches, kernel function based approaches and apriori-like approaches. We now briefly introduce the first three categories, then we focus on the last one, as the rest of the thesis follows this kind of approach.

3.4.1 Greedy Algorithms

Two pioneering works appeared in around 1994, both of which were in the framework of greedy search based graph mining (25; 100). Interestingly both were originated to discover concepts from graph representations of some structure, e.g. a conceptual graph similar to semantic network and a physical system such as electric circuits. One is called SUBDUE (25). SUBDUE deals with conceptual graphs which belong to a class of connected graph. The vertex set $V(G)$ is $R \cup C$ where R and C are the sets of labeled vertices representing relations and concepts respec-

tively. The edge set $E(G)$ is U which is a set of labeled edges. Though the original SUBDUE targeted the discovery of repeatedly appearing connected subgraphs in this specific type of graph data, i.e., concept graph data, the principle can be applied to generic connected graphs. SUBDUE starts looking for a subgraph which can best compress an input graph G based on Minimum Description Length (MDL) principle (80). The found subgraph can be considered a concept. This algorithm is based on a computationally-constrained beam search. It begins with a subgraph comprising only a single vertex in the input graph G, and grows it incrementally expanding a node in it. At each expansion it evaluates the total description length (DL), $I(G_s) + I(G \mid G_s)$, of the input graph G which is defined as the sum of the two: DL of the subgraph, $I(G_s)$, and DL of the input graph, $I(G \mid G_s)$, in which all the instances of the subgraph are replaced by single nodes. It stops when the subgraph that minimizes the total description length is found. The search is completely greedy, and it never backtracks. Since the maximum width of the beam is predetermined, it may miss an optimum Gs. One of the good features of SUBDUE is that it can perform approximate matching to allow slight variations of subgraphs. It can also embed background knowledge in the form of predefined subgraphs. After the best substructure is found and the input graph is rewritten, the next iteration starts using the rewritten graph as a new input. This way, SUBDUE finds a more abstract concept at each round of iteration. As is clear, the algorithm can find only one substructure at each iteration. Furthermore, it does not maintain strictly the original input graph structure after compression because its aim is to facilitate the global understanding of the complex database by forming hierarchical concepts and using them to approximately describe the input data. In 2002, a new technique to induce a graph grammar has been developed by the same group (53). The other one is called Graph Based Induction (GBI) (100). GBI was originally intended to find interesting concepts from inference patterns by extracting frequently appearing patterns in the inference trace. GBI was formulated to derive a graph having a minimal size similarly to SUBDUE by replacing each found subgraph with one vertex that it repeatedly compress the graph. It used an empirical graph

size definition that reflected the sizes of extracted patterns as well as the size of compressed graph. This prevented the algorithm from continually compressing, which meant the graph never became a single vertex. GBI can handle both directed and undirected labeled graph with closed paths (including closed edges). An opportunistic beam search similar to genetic algorithm was used to arrive at local minimum solutions.

3.4.2 Inductive Logic Programming

The first system to try complete search for the wider class of frequent substructure in graphs named WARMR was proposed in 1999 (30). They combined ILP method with Apriori-like level wise search to a problem of carcinogenesis prediction of chemical compounds. Because this approach allows variables to be introduced in the arguments of the predicates, the class of structures which can be searched is more general than graphs. However, this approach easily faces the high computational complexity due to the equivalence checking under θ-subsumption (an NP-complete operation) on clauses and the generality of the problem class to be solved. To alleviate this difficulty, a new system called FARMAR has later been proposed (73). It also uses the level wise search, but applied less strict equivalence relation under substitution to reduced atom sets. FARMAR runs two orders of magnitudes faster. However, its result includes some propositions having different forms but equivalent in the sense of the θ-subsumption due to the weaker equivalence criterion. A major advantage of these two systems is that they can discover frequent structures in high level descriptions. However, finding first-order pattern is harder than finding propositional patterns, and it is unclear how fast such techniques will work on very large graph datasets.

3.4.3 Kernel Function Based Approaches

Graph invariants are the quantities to characterize the topological structure of a graph. If two graphs are topologically identical, i.e., isomorphic, they also have identical graph invariants, though the reverse property does not hold. Examples of graph invariants are the number of ver-

tices, the number of edges, the number of cyclic loops, and so on. From a graph G is possible to collect its invariants in a feature vector X_G. When the graph G is very complex, the dimension of X_G required to approximate the graph topology closely can be very large. This causes computational problems for many mining approaches. To alleviate these issues, it is possible to introduce a kernel function $K(X_{G_x}, X_{G_y})$ and a mapping $\phi : X_g \rightarrow H$ enabling the representation of K by the inner product $< \phi(X_{Gx}), \phi(X_{Gy}) >$. K represents a similarity between two graphs G_x and G_y, and H is usually the Hilbert space. For the application to graph-based data mining, the key issue is to find the good combinations of the feature vector X_G and the mapping $\phi : X_g \rightarrow H$ to define appropriate similarity. A recent study proposed a composition of a kernel function characterizing the similarity between two graphs G_x and G_y based on the feature vectors consisting of graph invariants of vertex labels and edge labels in the certain neighbor area of each vertex (44). This is used to classify the graphs into binary classes by SVM (Support Vector Machine). Given training data consisting of graphs having binary class, the SVM is trained to classify each graph. Though the similarity is not complete and sound in terms of the graph isomorphism, the graphs are classified properly based on the similarity defined by the kernel function. Another framework of kernel function related with graph structures is called *diffusion kernel* (58). Though this is not dedicated to graph-based data mining, each instance is assigned to a vertex in a graph structure, and the similarity between instances is evaluated under the diffusion process along the edges of the graph. Some experiments report that the similarity evaluation in the structure characterizing the relations among the instances provides better performance in classification and clustering tasks than the distance based similarity evaluation.

3.4.4 Apriori-like Algorithms

This set of algorithms mines a complete set of subgraphs under a support measure. The search strategy of these approaches can be BFS or DFS. The breadth-first algorithms have the same setup as the original Apriori algo-

rithm for mining frequent itemsets (7). They iterate a process of generating candidate subgraphs and determining their support in the database. Candidates are generated by joining two subgraphs that were previously found to be frequent. After the joined subgraph is obtained, it is checked whether all its subgraphs are frequent; if not, the candidate is pruned. The depth-first algorithms do not subdivide the search into strict candidate generation and candidate evaluation phases. Essentially, each of these algorithms scans all embeddings of a subgraph in the database, and collects from that scan the support of the refinements, i.e., the individual edges and nodes that can be connected to the subgraph. The search recurses immediately on each of the frequent refinements. Disadvantages of the first approach include a large memory consumption, as at any moment the complete search space at level k is in memory, while the DFS approach has the drawback that only an arbitrary part of the isomorphic subgraphs can be found when the search is stopped due to the search time constraints if the search space is very large. In sections 3.5 and 3.6 we show how to overcome these limitations by using a multi-level data structure that can be dropped on disk level by level, or making usage of a maximum size constraint that allows to stop the pattern expansion on the basis of an arbitrary user-defined level that is appropriate to the application under analysis.

The initial work in this set was AGM (51). Starting from frequent graphs where each graph is a single vertex, the frequent graphs having larger sizes are search in bottom up manner by generating candidates having an extra vertex. AGM represents the graphs with vertex-sorted adjacency matrices. Every sub-matrix is normalized and since for every candidate all the normal forms are calculated and stored at each step, this leads to an extreme redundancy. This means that, despite the ease of implementation and the good performances for small datasets, AGM does not perform well on large datasets.

After the proposal of AGM, a family of graph-based data mining based on similar principles has been proposed. A work is FSG (Frequent SubGraph discovery) system (62) which also takes similar definition of canonical labeling of graphs based on the adjacency matrix. To increase the ef-

ficiency of deriving the canonical labels, the approach uses some graph vertex invariants such as the degree of each vertex in the graph. FSG also increases the efficiency of the candidate generation of frequent subgraphs by introducing the transaction ID (TID) method. Furthermore, FSG limits the class of the frequent subgraphs to connected graphs. Under this limitation, FSG introduces an efficient search algorithm using *core* which is a shared part of the size $k-1$ in the two frequent subgraphs of the size k. FSG increases the joining efficiency by limiting the common part of the two frequent graphs to the core. Once the candidate set is obtained, their frequency counting is conducted by checking the cardinality of the intersection of both TID lists. FSG runs fast due to the introduction of many techniques, but it consumes much memory space to store TID lists for massive graph data.

More recently, DFS based canonical labeling approach called gSpan (graph-based Substructure pattern mining) has been proposed (99). This approach also uses the idea of canonical labeling which is derived from a coding scheme of a graph representation. The main difference of the cod-

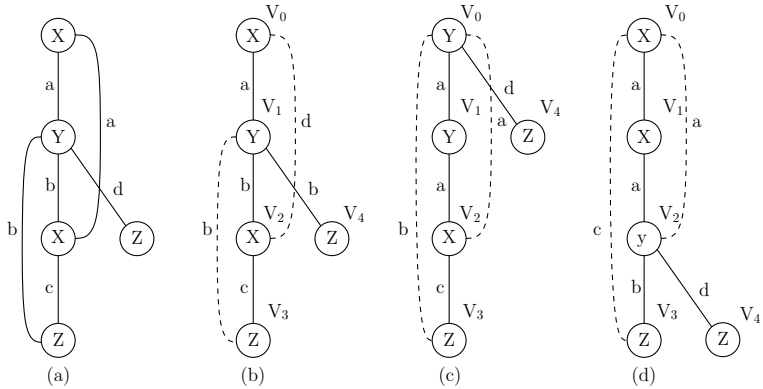


Figure 1: Different DFS coding for a graph

ing from the other approach is that it uses a tree representation of each graph instead of the adjacency matrix to define the code of the graph as depicted in Figure 1. Given a graph (a), various quasi-tree expressions

of the graph exist depending on the way to take a root vertex among the vertices. (b), (c) and (d) are the examples where the least number of edges to remove all cyclic paths are represented by dashed lines. Upon this representation, starting from the root vertex, the code is generated by following the lexicographical order of the labels for the combinations of vertices and the edge bound by the vertices. For example, the combinations of (v_0, v_1) with the label (v_0, v_1, X, a, Y) comes first in the code because this is younger than dashed (v_0, v_2) having (v_0, v_2, X, a, X). Next, starting from v1 which is the last vertex of the previous code element, the youngest code element (v_1, v_2, Y, b, X) is chosen. Then from the last v_2, the element (v_2, v_0, X, a, X) is chosen. When this trace returns to a vertex which is involved in the previous code element, the trace backtracks by one step, and the next younger element starting from v_2, i.e., (v_2, v_3, X, c, Z) is chosen. The subsequent elements (v_3, v_1, Z, b, Y) and (v_1, v_4, Y, d, Z) are traced in a recursive manner. The sequence of these elements is called a code of this quasi-tree expression. The other quasi-tree expressions including (c) and (d) have their own codes respectively. Table 2 shows the DFS codes for the graphs (b), (c) and (d) of Figure 1. Among the codes, the quasi-tree expression having the minimum code in

edge	(b)	(c)	(d)
0	(v_0, v_1, X, a, Y)	(v_0, v_1, Y, a, X)	(v_0, v_1, X, a, X)
1	(v_1, v_2, Y, b, X)	(v_1, v_2, X, a, X)	(v_1, v_2, X, a, Y)
2	(v_2, v_0, X, a, X)	(v_2, v_0, X, b, Y)	(v_2, v_0, Y, b, X)
3	(v_2, v_3, X, c, Z)	(v_2, v_3, X, c, Z)	(v_2, v_3, Y, b, Z)
4	(v_3, v_1, Z, b, Y)	(v_3, v_0, Z, b, Y)	(v_3, v_0, Z, c, X)
5	(v_1, v_4, Y, d, Z)	(v_0, v_4, Y, d, Z)	(v_2, v_4, Y, d, Z)

Table 2: The DFS codes for the graphs (b),(c) and (d) in Figure 1

terms of the lexicographical order is the canonical form, and the corresponding code is the canonical label. Because the code is derived in the DFS algorithm, this code is called DFS code. Every graph in a data set is represented by the multiple codes in this manner. Then all codes are sorted according to ascending lexicographical order, and the matching of the codes starting from the first elements among the codes are conducted

by using DFS in the sorted order. This means that the search trees of the DFS matching are ordered trees where the left branch is always younger than the right branch. Accordingly, when the branch representing a subgraph which is identical to the subgraph previously visited in the ordered tree search are found, the further DFS in the search tree can be pruned. By applying this DFS coding and DFS search, gSpan can derive complete set of frequent subgraphs over a given minsup in a very efficient manner in both computational time and memory consumption. In conclusion, gSpan ensures a lower spatial complexity w.r.t. its predecessor, thanks to the DFS search strategy, and performs globally much better than FSG.

MoFa (18) has been targeted towards molecular databases, but it can also be used for arbitrary graphs. MoFa performs a depth-first search and generates new fragments by extending smaller ones while keeping embedding lists to speed up the support computation. MoFa stores all embeddings (both nodes and edges). Extension is restricted to those fragments, that actually appear in the database. Isomorphism tests in the database can cheaply be done by testing whether an embedding can be refined in the same way. MoFa uses a fragment-local numbering scheme to reduce the number of refinements generated from a fragment: MoFa counts the nodes of a fragment according to the sequence in which they have been added. When a fragment is extended at node n, later refinements may only occur at n or at nodes bigger than n. Moreover, all extensions that grow from the same node n are ordered according to increasing node and edge labels. Although this local ordering helps, MoFa still generates many isomorphic fragments and then uses standard isomorphism testing to prune duplicates.

FFSM (46) represents graphs as triangle matrices (node labels on the diagonal, edge labels elsewhere). The matrix-code is the concatenation of all its entries, left to right and line by line. Based on lexicographic ordering, isomorphic graphs have the same canonical code (CAM - Canonical Adjacency Matrix). When FFSM joins two matrices of fragments to generate refinements, only at most two new structures result. FFSM also needs a restricted extension operation: a new edge-node pair may only be added to the last node of a CAM. After refinement generation, FFSM

permutes matrix lines to check whether a generated matrix is in canonical form. If not, it can be pruned. FFSM stores embeddings to avoid explicit subgraph isomorphism testing. However, FFSM only stores the matching nodes, edges are ignored. This helps speeding up the join and extension operations since the embedding lists of new fragments can be calculated by set operations on the nodes. FFSM performs generally better than gSpan, thanks to the fact that it does not need to solve the subgraphs isomorphism problem during the frequency counting, but still it needs for a simple but costly canonical labeling.

Gaston (74) stores all embeddings, to generate only refinements that actually appear and to achieve fast isomorphism testing. The main insight is that there are efficient ways to enumerate paths and (non-cyclic) trees. By considering fragments that are paths or trees first, and by only proceeding to general graphs with cycles at the end, a large fraction of the work can be done efficiently. Only in that last phase, Gaston faces the NP-completeness of the subgraph isomorphism problem. Gaston defines a global order on cycle-closing edges and only generates those cycles that are larger than the last one. Duplicate detection is done in two phases: hashing to pre-sort and a graph isomorphism test for final duplicate detection. Among the algorithms presented so far, Gaston is the one the performs better.

In ADI-Mine (93), a new data structure was proposed to deal with the problem of most the above approaches: the memory consumption. They are, in fact, also known as memory-based approaches, while ADI-Mine was introduced as the first disk-based approach to graph mining. The *ADI* structure, shown in section 3.5, is a three-level data structure containing all the information in the database needed for the mining stage. In the first level, it contains all the frequent edges in the database. In the second level, for each edge, the list of graphs containing the edge is present. In the last level, each graph is represented in a adjacency list, and for each occurrence of a particular edge in a particular graph, a pointer to it is set in this level. If the structure is unavailable, then the ADI-Mine algorithm scans the graph database and constructs the index. Otherwise, it just uses the *ADI* structure on the disk. The frequent edges can be ob-

tained from the edge table in the *ADI* structure. Each frequent edge is one of the smallest frequent graph patterns and thus should be output. Then, the frequent edges should be used as the "seeds" to grow larger frequent graph patterns, and the frequent adjacent edges of e should be used in the pattern-growth. An edge e' is a frequent adjacent edge of e if e' is an adjacent edge of e in at least min_sup graphs. The set of frequent adjacent edges can be retrieved efficiently from the *ADI* structure since the identities of the graphs containing e are indexed as a linked-list, and the adjacent edges are also indexed in the adjacency information part in the *ADI* structure. The pattern growth is implemented as calls to procedure subgraph-mine. Procedure subgraph-mine tries every frequent adjacent edge e (i.e., edges in set F_e) and checks whether e can be added into the current frequent graph pattern G to form a larger pattern G'. The DFS code is used to test the redundancy. Only the patterns G' whose DFS code is minimum is output and further grown. All other patterns G' are either found before or will be found later at other branches. The correctness of this step is guaranteed by the property of DFS code. Once a larger pattern G' is found, the set of adjacent edges of the current pattern should be updated, since the adjacent edges of the newly inserted edge should also be considered in the future growth from G'. This update operation can be implemented efficiently, since the identities of graphs that contain an edge e are linked together in the *ADI* structure, and the adjacency information is also indexed and linked according to the graph-ids.

CabGin (94) is a recent freamework developed for pushing various classes of constraints in the task of frequent subgraph mining. This algorithm is the first attempt to push certain classes of constraints, together with the frequency, when working on graph datasets. The CabGin framework was developed on the structure of ADI-Mine, modifying the latter for working with monotone and succint constraints. However, the monotone constraints are used only to minimize the computational cost of the check for antimonotonicity, without being really pushed in the computation to reduce the search space.

GPrune (101) is a more recent work that allows the user to push constraints into the mining stage. However, only a few constraints are pro-

posed, and the data pruning can be further enhanced to support the disconnected components of a graph. Moreover, this approach is still memory-based.

In section 3.5 we present an extension of the ADI structure, together with an enhanced set of pruning strategy. These concepts have been implemented in a graph preprocessor that can be coupled with any existing graph miner, and tested for efficiency and effectiveness. The results of this analysis are shown in section 3.5. In the same chapter, we propose a general algorithm for constraint-based graph mining that includes the strategies of the preprocessor into the mining stage.

Other algorithms have been proposed in the last years, especially to overcome the need for specialized approaches to particular problems that show peculiar characteristics. A few more algorithms are presented in section 3.6.2, as they deal with the problem of finding frequent subgraphs in a single large graph. In the very few last years, there has been increasing attention in mining pattern in *dynamic* graphs, to model problems such as Social Network Analysis, Information Propagation, Viral Marketing, and so on. A few of them were discussed in chapter 2. Other approaches solve a different problem, opposite to what we defined here: starting from local patterns, they reconstruct the original graph. Such approaches are of particular interest when performing *Workflow Mining*, and some of them were discussed in chapter 2.

3.5 The Transactional Setting

In section 3.3 we have defined the frequent subgraph mining problem and its constrained version. In this section, we analyze the latter, in the transactional setting, i.e. when the dataset is a collection of graphs. This setting fits problems in biology, chemistry, workflow mining, and whenever the data can be aggregated in graph-transactions. For sake of simplicity here we focus on the problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$, but since a conjunction of $\mathcal{C}_{AM}$ constraints is still a $\mathcal{C}_{AM}$ constraint, the proposed solutions work more generally for any conjunction of anti-monotone and monotone constraints: i.e., $Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$. We present $\mathcal{G}amp$ ($\mathcal{G}$raphs $a$nti-

monotone and $m$onotone $p$runing), a pre-processing algorithm, which reduces dramatically the input database, hence inducing a strong pruning of the search space. Furthermore, a novel data structure, named *EADI*, is introduced. The *EADI* is an extension of the *ADI* structure (93), and is the basis of efficient $\mathcal{G}amp$ implementation. Being a pre-processor, $\mathcal{G}amp$ can be coupled with any subgraph miner proposed so far. This is the first work showing how to effectively push monotone constraints in frequent subgraph mining, using them to reduce input data and to prune the search space.

This section is organized as follows: section 3.5.1 give the needed definition of support for the problem; section 3.5.2 explains how to push monotone and anti-monotone constraints into the computation; section 3.5.3 shows an extension of the mentioned ADI data structure; section 3.5.4 shows the $\mathcal{G}amp$ preprocessing algorithm; in section 3.5.5, the results of applying $\mathcal{G}amp$ to both synthetic and real datasets are presented; finally, in section 3.5.6, we show a generic algorithm for solving the Constraint-Based Frequent Subgraph Mining problem.

3.5.1 Support definition

As mentioned in the previous section, we have to define an appropriate and convenient support function in order to have a well defined mining problem. Fortunately, in the transactional setting, such a function is easy to define: we take as the support of a pattern the number of graphs in which the pattern can be found at least in one occurrence. More formally:

Definition 15 (Support)

$$sup_{\mathcal{D}}(G) = \sigma_{\mathcal{D}}(G) = \mid \{G' \in \mathcal{D} \mid G \subseteq G'\} \mid$$

It is easy to see that the support defined in this way is anti-monotone, and this property helps in the apriori-like approaches, as the expansion of a pattern can be stopped as soon as the frequency constraint is not satisfied anymore, since all the supergraphs of it will not satisfy it as well.

3.5.2 Pushing monotone constraints

The previous section has provided a large variety of constraints which are all either monotone or anti-monotone. Dealing with conjunctions of such kinds of constraints is very significant in many contexts. In fact, while the frequency constraint (which is anti-monotone) makes patterns emerge from the mere random component of the data, monotone constraints privilege larger, more structured patterns (e.g., with many vertices, containing a cycle, having sum of atomic weight larger than, etc.). The situation is graphically represented in Figure 2. On the left is shown how, conjoining another anti-monotone constraint to $\mathcal{C}_{freq}$, just results is strengthening the focus on small patterns, most of which are insignificant and redundant (just think about all frequent 1-edge graphs). But conjoining a monotone constraint to frequency (as in the right half of Figure 2), really helps in making few, interesting (i.e., frequent enough and structured enough) patterns emerge: while $\mathcal{C}_{freq}$ cuts away many large but infrequent patterns, $\mathcal{C}_M$ cuts away many small patterns.

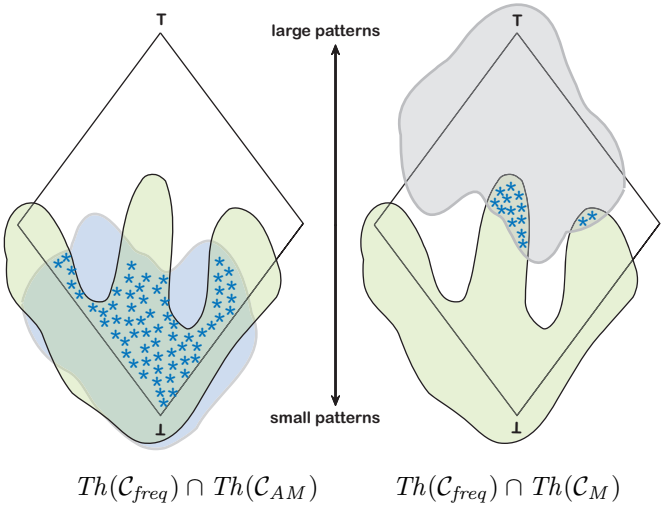


Figure 2: Different solution spaces.

However, while pushing anti-monotone constraints deep into the min-

ing algorithm is easy and effective, due to the fact that the computation is geared on $\mathcal{C}_{freq}$, the case is different for monotone constraints. Many authors (20; 22; 61) have attacked the challenging computational problem $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ (especially on the pattern class of itemsets) focusing on its search space and facing the intrinsic *tradeoff* between $\mathcal{C}_{AM}$ and $\mathcal{C}_M$ based pruning. Still on itemsets, (15) introduced a completely different approach which exploits monotone constraints by means of data-reduction. The *ExAnte Property* (15) is obtained by shifting attention from the pattern search space to the input data: indeed, the $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *tradeoff* exists only if we focus exclusively on the search space of the problem, while if exploited properly, monotone constraints can reduce dramatically the data in input, in turn strengthening the anti-monotonicity pruning power. With data reduction techniques it is possible to exploit the effectiveness of a $\mathcal{C}_{AM}$-$\mathcal{C}_M$ *synergy* (14; 17). In the following the same idea is applied and it is showed that, in the case of graph mining, the idea can be strengthen further thanks to the fundamental constraint of mining *only connected* subgraphs.

Data Reduction Properties

The ExAnte property states that a transaction (i.e., a graph in the database, in our case) which does not satisfy the given monotone constraint can be deleted from the input database since it will never contribute to the support of any solution substructure. Thanks to this property, we can define the *monotone pruning* of graph data as follows:

Proposition 2 (Monotone Pruning) *Given a conjunction of monotone constraints $\mathcal{C}_M$, it is possible to define the monotone pruning of a database of graphs $\mathcal{D}$ as the dataset resulting from removing the graphs that do not satisfy $\mathcal{C}_M$:*

$$\mathcal{M}p_{[\mathcal{C}_M]}(\mathcal{D}) = \{G \in \mathcal{D} \mid G \in Th(\mathcal{C}_M)\}.$$

It holds that this data reduction does not affect the support of solution patterns: $\forall G' \in Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$:

$$sup_{\mathcal{D}}(G') = sup_{\mathcal{M}p_{[\mathcal{C}_M]}(\mathcal{D})}(G').$$

A major consequence of reducing the input database in this way is that it implicitly reduces the support of a large amount of edges that do not satisfy $\mathcal{C}_M$ as well, resulting in a reduced number of patterns visited during the mining. Moreover, infrequent edges can not only be removed from the search space together with all their supergraphs, for the same anti-monotonicity property of $\mathcal{C}_{freq}$, they can be deleted also from all graphs in $\mathcal{D}$.

Subsequently, we can define also an *anti-monotone pruning* as follows:

Proposition 3 (Anti-monotone Pruning) *Given a conjunction of anti-monotone constraints $\mathcal{C}_{AM}$, the anti-monotone pruning of a graph $G = (V, E, L, l)$ is defined as the graph obtained removing the edges that do not satisfy $\mathcal{C}_{AM}$ from G: this pruning is denoted as $\mathcal{AM}p_{[\mathcal{C}_{AM}]}(G)$. Given a database of graphs $\mathcal{D}$ the anti-monotone pruning of $\mathcal{D}$ is defined as the anti-monotone pruning of all graphs in $\mathcal{D}$: $\mathcal{AM}p_{[\mathcal{C}_{AM}]}(\mathcal{D}) = \{G' \mid G \in \mathcal{D} \wedge G' = \mathcal{AM}p_{[\mathcal{C}_{AM}]}(G)\}$. It holds that this data reduction does not affect the support of solution patterns: $\forall G' \in Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$:*

$$sup_{\mathcal{D}}(G') = sup_{\mathcal{AM}p_{[\mathcal{C}_{AM}]}(\mathcal{D})}(G').$$

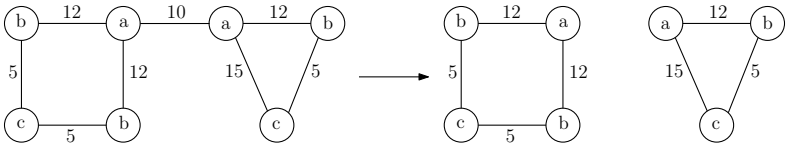


Figure 3: Example of $\mathcal{AM}p$ pruning.

Removing edges from graphs has got another positive effect: after the anti-monotone pruning, a growing number of graphs that do not satisfy $\mathcal{C}_M$ can be found in the input database. Therefore the anti-monotone pruning creates new opportunities for monotone pruning, and viceversa: in this way we enter a loop where two different kinds of pruning reduce the input data, strengthening each other step by step. Moreover, removing edges from the connected graphs in the input database, can disconnect these graphs. Since the focus is in mining frequent *connected* substructures, it is possible to strengthen the $\mathcal{M}p$ pruning to remove any

connected component, i.e., *maximal connected subgraph*, in the input database which does not satisfy $\mathcal{C}_M$.

Example 3 *Let G be the graph in Figure 3. Consider the query: $Th(\mathcal{C}_{freq[\mathcal{D},30]} \wedge \mathcal{C}_{sum[E,\geq,35]})$. We have that $\mathcal{C}_{sum[E,\geq,35]}(G) = true$, since the sum of edge labels of G is 76. Suppose now that the edge $(a, 10, a)$ results to be infrequent. $\mathcal{AM}p$ removes it from G, which results to be divided in two connected components. When the satisfaction of $\mathcal{C}_M$ is checked, the sum of edge labels of the whole G must not be considered, but instead the one of all its connected components in isolation. In this case none of the two connected components satisfies $\mathcal{C}_M$ and thus they will be both deleted by the subsequent $\mathcal{M}p$.*

On the basis of these considerations it is possible to make the monotone pruning more effective.

Proposition 4 (Connected Components Pruning) *Let $G \in \mathcal{D}$ be a graph non necessarily connected. Given a conjunction of monotone constraints $\mathcal{C}_M$, the monotone pruning $\mathcal{M}p_{[\mathcal{C}_M]}(G)$ is defined as the graph resulting by removing from G all its connected components that do not satisfy $\mathcal{C}_M$. The monotone pruning of the whole database is defined as the monotone pruning of all graphs in the database: $\mathcal{M}p_{[\mathcal{C}_M]}(\mathcal{D}) = \{G' \mid G \in \mathcal{D} \wedge G' = \mathcal{M}p_{[\mathcal{C}_M]}(G)\}$. It holds that this data reduction does not affect the support of solution patterns: $\forall G' \in Th(\mathcal{C}_{AM}) \cap Th(\mathcal{C}_M)$:*

$$sup_{\mathcal{D}}(G') = sup_{\mathcal{M}p_{[\mathcal{C}_M]}(\mathcal{D})}(G').$$

This section has described a loop where two different kinds of pruning ($\mathcal{AM}p$ and $\mathcal{M}p$) cooperate to reduce the search space and the input dataset, strengthening each other step by step until no more pruning is possible (a fix-point has been reached). In the end, the reduced dataset resulting from this fix-point computation is usually much smaller than the initial dataset, and it can feed any frequent substructure miner for a much smaller (but complete) computation. We call this method the $\mathcal{G}amp$ (Graph Anti-monotone Monotone Pruning) pre-processing, which can be further clarified by the following example.

Run-through Example

Let $\mathcal{D}$ in Figure 4 be the input database of graph and suppose that we want to mine patterns having support ≥ 2 and sum of edge labels ≥ 35,

i.e., $Th(\mathcal{C}_{freq[\mathcal{D},2]} \wedge \mathcal{C}_{sum[E,\geq,35]})$. If we start counting the support of edges, we discover that they are all frequent. Therefore it could seem that no pruning is possible at this time. But if we take in consideration the monotone constraint $\mathcal{C}_{sum[E,\geq,35]}$, we discover that graph G_2 does not satisfy it. Therefore G_2 can be removed from $\mathcal{D}$ (pruning $\mathcal{M}p$). This removal

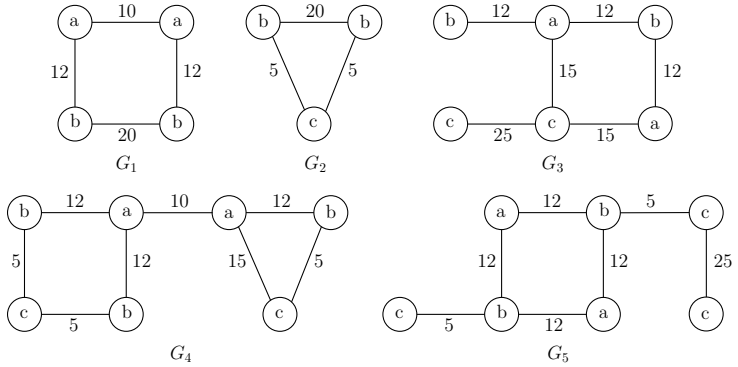


Figure 4: Run-through example: the input database $\mathcal{D}$.

makes the support of the edge $(b, 20, b)$ shrink to 1, and thus such edge which was initially frequent, is now infrequent and can be removed in all its occurrences from $\mathcal{D}$ (pruning $\mathcal{AM}p$). In particular, it is removed from graph G_1. The database after the first round of $\mathcal{M}p$ and $\mathcal{AM}p$ pruning is as in Figure 5. After the removal of $(b, 20, b)$, the graph G_1 has a sum of edges which does no longer satisfy $\mathcal{C}_M$, and thus it will be $\mathcal{M}p$runed during the second iteration. After the removal of G_1, the edge $(a, 10, a)$ turns out to be infrequent, and thus it is $\mathcal{AM}p$runed. This pruning, which corresponds to the one in Figure 3, has the effect of dividing G_4 in two connected components, that do not satisfy $\mathcal{C}_M$.

Therefore, both components can be $\mathcal{M}p$runed. At this point both the edges $(a, 15, c)$ (supported only by G_3) and $(b, 5, c)$ (supported only by G_5) turn out to be infrequent and they are $\mathcal{AM}p$runed. After this pruning the input database has been reduced as in Figure 6.

Both G_3 and G_5 are divided in two connected components. During the following $\mathcal{M}p$ the components not satisfying $\mathcal{C}_M$ are deleted. In this case,

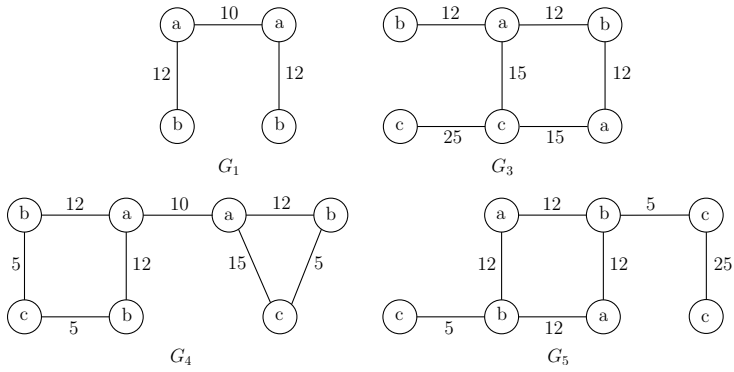


Figure 5: After the first $\mathcal{M}p$ and $\mathcal{AM}p$.

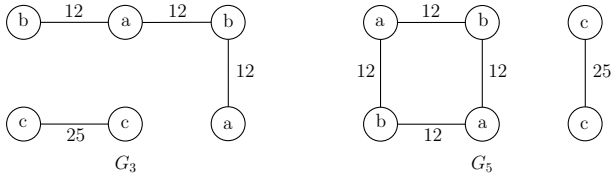


Figure 6: After the third $\mathcal{M}p$ and $\mathcal{AM}p$.

in both graphs, there is one connected component made only by the edge $(c, 25, c)$: since it has not a sum of edge labels larger than 35 it is removed by both graphs. Note that, even if the edge $(c, 25, c)$ is frequent, since it appears in two graphs, it is removed by $\mathcal{M}p$ (as a connected component) and not by $\mathcal{AM}p$ (as an edge). At this point the input database is as in Figure 7. The unique edge which is present in both graphs $(a, 12, b)$ is frequent, and thus no other pruning is possible. This is the final database, the result of our $\mathcal{G}amp$ pre-processing, which can feed any frequent substructure miner for a much smaller (but complete) computation. Note that on this toy-example there is no need to run a mining algorithm: the unique pattern solution to the query (i.e., a graph with 3 $(a, 12, b)$ edges appropriately connected, with support = 2 and sum of edge labels = 36) is evident in Figure 7.

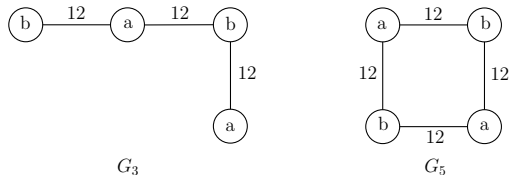


Figure 7: Run-through example: the final reduced database.

3.5.3 Extending the *ADI* structure

This section introduces an extension of the *ADI* structure (93), which is a key point for an efficient implementation of our $\mathcal{G}amp$ pre-processing algorithm. In Figure 8 the *ADI* index structure for graphs G_3 and G_5 from database $\mathcal{D}$ of Figure 4 is reported. Recall from section 3.4 that the ADI structure is composed by three levels, in which we have the frequent edges, the list of graphs in which they appear, and the pointers to every occurrence in a linked-list representation of the graphs in the dataset.

We decided to implement our $\mathcal{G}amp$ pre-processing on the basis of the *ADI* index, because $\mathcal{G}amp$ requires all the operations for which the *ADI* index was designed:
OP1: for each new loop, $\mathcal{G}amp$ requires to compute the support of all edges, because some of them could have become infrequent;
OP2: when an edge is found infrequent, $\mathcal{G}amp$ needs to remove all its appearances in the database ($\mathcal{AM}p$), thus it has to access all the graphs where it appears;
OP3: when an edge is $\mathcal{AM}p$runed from a graph, we must check if it was a *bridge* or not, i.e., if its removal disconnects the graph. An edge is a bridge if it does not lie on any simple graph (28): this can be easily checked using the adjacency information.

While all the fundamental operations for which the *ADI* index was designed are necessary for the $\mathcal{G}amp$ algorithm, they are not enough. In fact, when the disconnection of a graph in the database occurs, we must from that moment consider its connected components (Proposition 4). But we can not consider each connected component simply like a new graph,

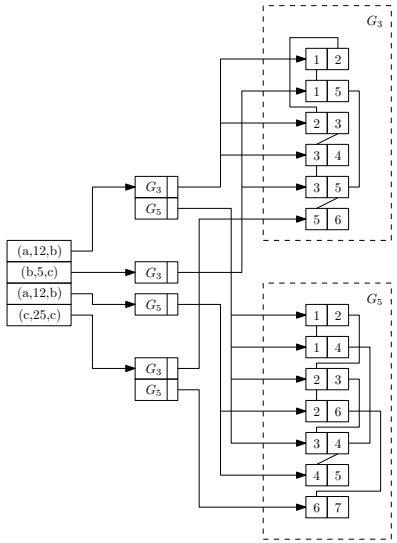


Figure 8: The *ADI* index structure for graphs G_3 and G_5 from database $\mathcal{D}$ of Figure 4.

otherwise we risk to count twice when checking the support for an edge appearing in two different subcomponents of the same graph. Thus we have two different conflicting requirements: on one hand the support of edges must be counted on the original graphs (by definition), on the other hand, monotone constraints must be checked on the connected components (for more effective pruning). To this purpose we extend the *ADI* index, adding a new level, named *connected components* level, in between the linked list of graphs and the adjacency information level. This new level contains, for each edge and for each graph in which the edge appears, a *linked list of connected component-ids*. In the adjacency information level we keep connected components information needed for checking constraints satisfaction: for instance, we keep the sum of edges if $\mathcal{C}_{sum[E,\geq,35]}$ is one of the constraints in the given query. Moreover, for each edge in the adjacency information level, we keep not only the vertices ids, but also the labels: this is needed to go back to the edge table, when we remove a con-

nected component, and we need to update the information for the edges it contains (see later line 6 of Algorithm 4).

This extended *ADI* index, named *EADI* (Extended-ADI, see an example in Figure 9), is suited for an efficient execution of the following operations needed by the $\mathcal{G}amp$ preprocessing:

OP4 Edge-host component checking: given an edge and a graph find the connected components of the graph in which the edge appears;

OP5 Monotone constraint checking: find which connected components of graphs in the input database do not satisfy a given $\mathcal{C}_M$ constraint.

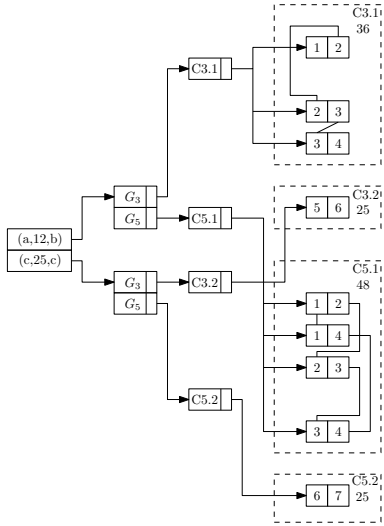


Figure 9: The extended *ADI* index structure (*EADI*) for graphs G_3 and G_5 after the third round of pruning as in Figure 6.

3.5.4 The $\mathcal{G}amp$ algorithm

This section describes how to implement the $\mathcal{G}amp$ pre-processing idea, described in section 3.5.2, on the extended *ADI* structure introduced in section 3.5.3.

Given the mining problem $Th(\mathcal{C}_{freq[\mathcal{D},\sigma]}) \cap Th(\mathcal{C}_M)$, the $\mathcal{G}amp$ pre-processing (Algorithm 1) outputs a reduced database $\mathcal{D}'$ obtained as follows. First the *EADI* structure is constructed using only the graphs in the input database $\mathcal{D}$ that satisfy $\mathcal{C}_M$. Then the $\mathcal{AM}p$ and $\mathcal{M}p$ pruning procedures are applied directly on the *EADI* structure in a loop until the number of frequent edges stops shrinking. At the end, the reduced *EADI* structure in written to a file $\mathcal{D}'$ (see line 8 of Algorithm 1), so that we it can be reused by some subgraph miner. Obviously this last step is not strictly necessary: if after the $\mathcal{G}amp$ pre-processing, the adopted mining algorithm works on the *ADI* structure, we can just transform the resulting *EADI* structure back to *ADI* and return it, so that the miner can start directly mining, thus saving the time for building the *ADI* structure.

Algorithm 1 $\mathcal{G}amp$

Input: $\mathcal{D}, \sigma, \mathcal{C}_M$
Output: A reduced database $\mathcal{D}'$.

1: $build\_EADI(\mathcal{D}, \mathcal{C}_M)$;
2: $f \leftarrow 0$;
3: **while** $f \neq EADI.number\_frequent\_edges$ **do**
4: $\quad f \leftarrow EADI.number\_frequent\_edges$;
5: $\quad \mathcal{AM}p(EADI, \sigma)$;
6: $\quad \mathcal{M}p(EADI, \mathcal{C}_M)$;
7: **end while**
8: $\mathcal{D}' \leftarrow eadi2db(EADI)$

The procedure for constructing the *EADI* index (Algorithm 2) works as follows. For each graph in the database the satisfaction of $\mathcal{C}_M$ is checked, and only if the graph satisfies the constraint it is inserted in the data structure, otherwise it is discarded. For each graph satisfying $\mathcal{C}_M$, we first build the adjacency information (4_{th} level of the *EADI* index, line 3 of Algorithm 2): in this step the edges are encoded according to the DFS-tree in the *minimum DFS code* (93; 99). Then each edge of the graph is added to the edge table (if not already present), the graph-id is inserted in the graph-id linked list of the edge (2_{nd} level of the *EADI* index, line 6 of Algorithm 2), and the connected components-id linked list is initial-

Algorithm 2 $build\_EADI$

Input: $\mathcal{D}, \mathcal{C}_M$
Output: An *EADI* structure.

```
1: for all G ∈ D do
2:   if C_M(G) then
3:     create G.adj_inf;
4:     for all edges e ∈ G do
5:       insert e in edge_table (if not present);
6:       insert G.id in e.graph_id;
7:       initialize e.G.component_id;
8:       link e.G to e.G.component_id;
9:       link e.G.component_id to G.adj_inf;
```

ized adding a unique connected component, corresponding to the graph G (3_{rd} level of the *EADI* index, line 5 of Algorithm 2). Finally the construction algorithm links the 2_{nd} to the 3_{rd} level of the index, and the 3_{rd} to the 4_{th} level appropriately. The $\mathcal{AM}p$ procedure (Algorithm 3), works directly on the *EADI* structure, removing infrequent edges. When an in-

Algorithm 3 $\mathcal{AM}p$

Input: *EADI*, σ
Output: A reduced *EADI*.

```
1: for all edges e ∈ EADI.edge_table do
2:   if |e.graph_id| < σ then
3:     for all G ∈ e.graph_id do
4:       for all C ∈ e.G.component_id do
5:         remove e from e.G.C.adj_inf;
6:         if bridge(e) then
7:           identify new connected components;
8:           update(EADI);
9:         end if
10:      end for
11:    end for
12:    set pruned(e);
13:  end if
14: end for
```

frequent edge is removed from a connected component, we must check if it was a bridge, and in the case, we must take the appropriate actions: identifying the new connected components, and update all the information in the *EADI* structure regarding the new components and the component from which they originates. At the end of the $\mathcal{AM}p$ procedure infrequent edges are not yet removed from the edge table: they are just marked as pruned. This is needed for the subsequent $\mathcal{M}p$ procedure (Algorithm 4) to directly access only those connected components that risk to no longer satisfy the monotone constraint, i.e., those components from which some edges have been removed by $\mathcal{AM}p$. For each edge which has been marked as pruned by the previous $\mathcal{AM}p$, the $\mathcal{M}p$ procedure checks $\mathcal{C}_M$ satisfaction for all the connected components that were containing the pruned edge. If a connected component is found no longer satisfying $\mathcal{C}_M$, it is removed from the *EADI* structure (lines from 5 to 13 of Algorithm 4). At the end of the $\mathcal{M}p$ procedure, also the edges which were marked from the previous $\mathcal{AM}p$ are definitively removed from the *EADI* structure (lines from 15 to 18 of Algorithm 4).

3.5.5 Experimental Results

This section reports the results of the experiments we conducted in order to asses the effectiveness of the method proposed. $\mathcal{G}amp$ was implemented in C++, and experimented on a IBM Thinkpad T43, equipped with a Pentium M 1.6Ghz, 1 GB Ram, running the Linux (Ubuntu 6.06) operating system. As subgraph miners to be run after $\mathcal{G}amp$ preprocessing we used a breadth-first apriori-like miner such as *FSG* (62), and depth-first miner such as *gSpan* (99). We experimented $\mathcal{G}amp$ on $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C}_M)$ with $\mathcal{C}_M$ being either $\mathcal{C}_{size[E,\geq,m]}$ or $\mathcal{C}_{cycle[m]}$. We measured the benefits of our approach in terms of data reduction, search space reduction, and run-time reduction.

Datasets

We tested $\mathcal{G}amp$ on both synthetic and real-world datasets. In particular, as real-world datasets we used the molecules dataset of the HIV essay

Algorithm 4 $\mathcal{M}p$

Input: *EADI*, $\mathcal{C}_M$
Output: A reduced *EADI*.

```
1: for all edges e ∈ EADI.edge_table do
2:   if pruned(e) then
3:     for all G ∈ e.graph_id do
4:       for all C ∈ e.G.component_id do
5:         if not C_M(C) then
6:           for all edges e' ∈ C do
7:             remove C from e'.G.C.component_id;
8:             if |e'.G.C.component_id| = 0 then
9:               remove G from e'.graph_id;
10:            end if
11:          end for
12:          destroy e.G.C.adj_inf;
13:        end if
14:      end for
15:      destroy e.G.component_id;
16:    end for
17:    destroy e.graph_id;
18:    remove e from edge_table;
19:  end if
20: end for
```

from 1999[1]: it contains 42,689 graphs, having in average 27 edges, the number of distinct labels is 58 for vertices and 2 for edges. The synthetic datasets were generated using the software described in (62). We have created two datasets: one with 100,000 graphs ($D100kE2V30$) and one with 250,000 graphs ($D250kE2V30$), both having 30 vertex labels, 2 edge labels, while all the other parameters are kept at the default values: i.e., average size of frequent graphs is 5, the number of potentially frequent graphs is 100, and average transaction size is 10.

[1] `http://dtp.nci.nih.gov/docs/aids/aids_data.html`

Data Reduction

In Table 3 we report the output of a typical $\mathcal{G}amp$ execution. In this example we used the dataset $D100kE2V30$ with quite a low minimum support threshold (0.6%) and the monotone constraint $\mathcal{C}_{size[E,\geq,12]}$. We

D100kE2V30

Iter	Edges	Graphs
0	642	63118
1	339	31997
2	258	21193
3	204	13424
4	130	6118
5	58	2538
6	16	1197
7	12	1197
8	12	1197

Table 3: $\mathcal{G}amp$ data reduction on $\mathcal{C}_{freq[\mathcal{D},600]} \wedge \mathcal{C}_{size[E,\geq,12]}$.

can see how, iteration by iteration the two different pruning cooperate to reduce the graphs in the input dataset and the frequent edges they contain. At the end both frequent edges and graphs are reduced of two orders of magnitude. Figure 10(a), (b) and (c) report data reduction experiments on the same dataset $D100kE2V30$, using the cycle constraint (i.e, $\mathcal{C}_M \equiv cycle[m]$): for three different support thresholds the figures report the reduction in terms of number of graphs, number of frequent edges and size of the reduced datasets. It is worth noting that the reduction on the number of frequent edges is an important measure, because even a small reduction of this number implies a strong pruning on the search space.

The dataset reduction in bytes is interesting because not only the number of graphs in the input dataset is reduced, but these graphs are also smaller than the original ones, thanks to edge reduction and to the removal of whole subgraphs.

The same constraint is also applied to the HIV dataset (Figure 10(d), (e) and (f)). We can note that the graphs in the HIV datasets are much

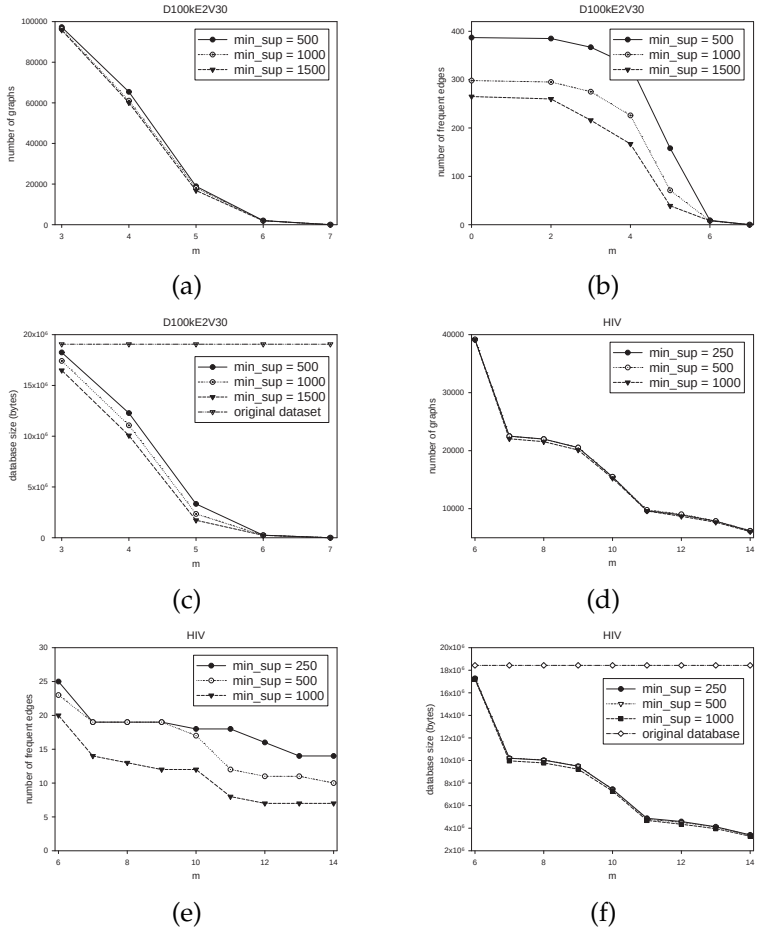
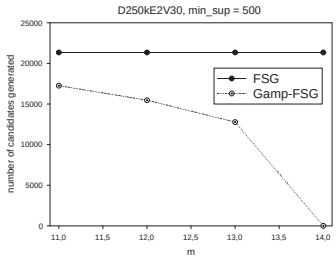


Figure 10: Experimental results: data reduction.

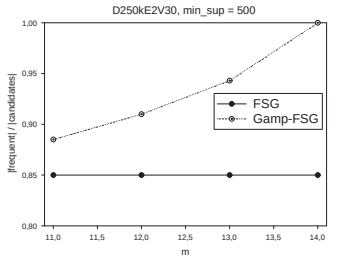
larger and more structured that those in the synthetic dataset: while a cycle threshold of 6 destroys almost all the data in the syntectic datasets, much larger thresholds are needed on the HIV dataset.

Search-space Reduction

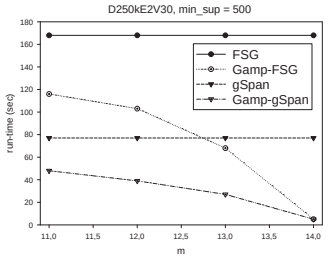
Figure 11(a) reports the number of candidates generated by *FSG* (which being an apriori-like algorithm, is based on candidate generation): this number can be seen as a measure of the search space explored by the algorithm. We can observe that as the monotone constraint becomes more selective, the number of candidates generated shrinks accordingly. The $\mathcal{G}amp$ preprocessing does not only reduce data and search space, it also increases the density of the patterns contained in the data. In 11(b) the ratio between the number of frequent patterns found and the number of candidates generated is reported: as the monotone constraint becomes more selective, this ratio increases.



(a): $\mathcal{C}_M \equiv size[E, \geq, m]$



(b): $\mathcal{C}_M \equiv size[E, \geq, m]$



(c): $\mathcal{C}_M \equiv size[E, \geq, m]$

Figure 11: Experimental results: search space and run-time reduction.

Run-time Reduction

Finally, Figure 11(c) reports a run-time comparison to better appreciate the benefits of $\mathcal{G}amp$ preprocessing. The comparison is between the execution time of *FSG*, *gSpan* and the one of $\mathcal{G}amp$-*FSG* and $\mathcal{G}amp$-*gSpan*, i.e., the pre-processor coupled with the two miners. We can note a run-time gain ranging from 40 to more than 100 seconds depending on the selectivity of the monotone constraint. It is worth noting that the time paid by $\mathcal{G}amp$ to build the *EADI* data structure, has not to be paid twice if we exploit a fine-grained tight coupling of $\mathcal{G}amp$ with some algorithm based on the *ADI* data structure such as *ADI-Mine*. Next section describes a constraint-based graph miner which encapsulate the data reductions and the *EADI* ideas introduced in this chapter, in a depth-first algorithm such as *ADI-Mine*, suitable for both memory-based and disk-based mining.

3.5.6 An algorithm for constraint-based graph mining in transactional setting

During the years in which this thesis was developed, a new graph miner named gPrune (101) was published by Zhu et al. This is a memory-based miner that is able to push monotone and anti-monotone constraints deep into the computation. However, this miner has some limitations: first, the set of constraints that the authors claim to be able to handle is somehow restricted; second, it does not consider the possibility of pruning the connected components, which we indeed proved to be a key feature; finally, it does not take advantage of any sophisticated data structure such as ADI or EADI, thus remaining a memory-based approach. Please note that, despite the saving of memory and runtime guaranteed by the use of constraints, when reasonable threshold are specified thus making the pruning effective, the check for a large set of constraints requires somehow additional time and memory, so an efficient data structure is a crucial key feature and its usage should definitely taken into account.

In the next section, we propose our algorithm for solving the Constraint-Based Frequent Subgraph Mining problem that takes into account these peculiarities.

The $\mathcal{G}P^3$ algorithm

It is possible to integrate the data reduction technique used by $\mathcal{G}amp$ inside an gSpan-like algorithm to dramatically reduce the size of the DFS Code Tree, avoiding costly subgraphs generations and counting. Unlike CabGin, the $\mathcal{G}P^3$ ($\mathcal{G}$raph Pattern discovery Pruning and Pushing constraints) is able to push monotone and antimonotone constraints deep into the computation, and use them to reduce the input dataset and to prune the search space.

Algorithm 5 $\mathcal{G}P^3$

Input: $\mathcal{D}, \sigma, \mathcal{C}_M$
Output: $\{G\}$ $frequent\ and\ that\ satisfy\ \mathcal{C}_M$.

1: $build\_EADI(\mathcal{D}, \mathcal{C}_M)$;
2: $\mathcal{G}amp(\mathcal{D}, \sigma, \mathcal{C}_M)$
3: **for all** edges $e \in EADI.edgetable$ **do**
4: **if** $Cm(e)$ **then**
5: $output(e)$
6: $D' \leftarrow \{G \in \mathcal{D}\}\ that\ contains\ e$
7: $\mathcal{G}amp(D', \sigma, \mathcal{C}_{M_e})$
8: $F_e \leftarrow \{(frequent)\ extensions\ of\ e\}$
9: $call\ subgraphmining(\mathcal{D}', F_e)$

The Algorithm 5 shows the steps taken by the $\mathcal{G}P^3$ miner: in line 1, the *EADI* structure for the database $\mathcal{D}$ is built according to Algorithm 2 presented in section 3.5.4. Then the $\mathcal{G}amp$ preprocessor is performed on the input dataset to reduce the graphs according to user-specified monotone and antimonotone constraints. In lines 4-8 all the frequent edges are taken as minimal graph patterns: if they satisfy Cm then they are graph pattern to be output, then, in any case, the projected database D' is calculated on line 6. In line 7 $\mathcal{G}P^3$ calls the $\mathcal{G}amp$ routine for efficiently pruning the projected dataset. In line 8, the set F_e of all the frequent extensions for a minimal graph pattern is calculated. Then $\mathcal{G}P^3$ calls the procedure $subgraphmining$ on D' and F_e.

Algorithm 7 shows the procedure $subgraphmining$: F_e is a set of graph patterns G; for each of them, the procedure checks the minimality of the

Algorithm 6 $subgraphmining(D, F_e)$

Input: D, F_e

```
1: for all G ∈ F_e do
2:    if DFS(G) not minimal then
3:       return;
4:    if Cm(G) then
5:       output G
6:       D' ← {G' ∈ D} that contains G
7:       Gamp(D', σ, C_{M_e})
8:       newF_e ← {(frequent) extensions of e}
9:       call subgraphmining(D', newF_e)
10: return;
```

DFS code in line 2. If the pattern satisfies the constraints, then it is output in line 5. In 6 the procedure calculates the projected dataset for the pattern G. Again, in line 7, the set $newF_e$ of all the frequent exstensions of G is calculated. Then the procedure is recursively called in line 8.

$\mathcal{G}P^3$ shows how, integrating $\mathcal{G}amp$ into a dept-first algorithm, it is possible to obtain a mining algorithm capable to push deep into the computation monotone constraints and to keep advantage of the synergy between monotone and antimonotone pruning for efficiently reduce both the originary input database and the search space.

3.6 The Single Graph Setting

In this section we study the problem of finding frequent subgraph patterns in the single graph setting, where the database consists in only one single graph, which is potentially large and unconnected.

As we have seen in section 3.5, in recent years, a number of efficient and scalable algorithms have been developed to find patterns in the transactional setting. These algorithms are complete in the sense that they are guaranteed to discover all frequent subgraphs and were shown to scale to very large graph datasets. However, algorithms that are capable of

finding patterns in the single graph setting have received much less attention, despite the fact that this problem setting is more generic and applicable to a wider range of datasets and application domains than the other. Recently, in fact, there arose a large number of graphs with massive sizes and complex structures in many new applications, such as biological networks, social networks, and the Web, demanding powerful Data Mining methods. Examples of such data are collaborative networks (DBLP[2], arXiv[3]), social networks (Flickr[4]), Y!360[5], the Web, protein interaction databases (UniProt[6]), and so on.

Because of their complex topological and semantical characteristics, we are now interested in patterns that frequently appear at many different places of a single network.

The remainder of this section is organized as follows: section 3.6.1 gives a possible definition of support in the single graph setting; section 3.6.2 briefly summarizes some of the approaches to the problem of mining frequent subgraphs in a single large graph; in section 3.6.3 we show what are the issues in pushing contraints in this setting and finally, in section 3.6.4 we give a possible algorithm for solving the Constraint-Based Frequent Subgraph Mining in the single graph setting.

3.6.1 Support Definition

Recalling from section 3.5, our problem is defined as mining $Th(\mathcal{C}_{freq}) \cap Th(\mathcal{C})$. The difference with the transactional setting is the computation of $Th(\mathcal{C}_{freq})$. While for the graph-transaction setting, the frequency of a pattern is determined by the number of graph transactions that the pattern occurs in, irrespective of how many times a pattern occurs in a particular transaction, in the single graph setting, the frequency of a pattern should be based on the number of its occurrences (i.e., embeddings) in the single graph. Otherwise, every pattern would have a support of either 0 or 1,

[2] http://www.informatik.uni-trier.de/~ley/db/
[3] http://arxiv.org/
[4] http://www.flickr.com/
[5] http://360.yahoo.com/
[6] http://www.ebi.ac.uk/uniprot/

depending if it is possible to find an occurrence of it into the graph.

Defining a concept of support for the single graph setting is a non-trivial task, which has received attention recently (21; 23; 34; 63). Let $\mathcal{G}_\Sigma$ be the set of all graphs over an alphabet Σ. We define support as a function $\sigma : \mathcal{G}_\Sigma \times \mathcal{G}_\Sigma \rightarrow \mathbb{N}$. Given a host-graph G and a pattern P, the value of $\sigma(P, G)$ reflects the support of the pattern in the host-graph. The most important property that a definition of support must satisfy is anti-monotonicity, that is, for all graphs G, P and P', where P is a subgraph of P', it must hold that $\sigma(P, G) \geq \sigma(P', G)$. This property is exploited by pattern miners to prune the search space.

However, there are some issues related to this problem: naive definitions of support have the problem that they are not anti-monotonic. For example, consider Figure 12. If we count the number of occurrences, what

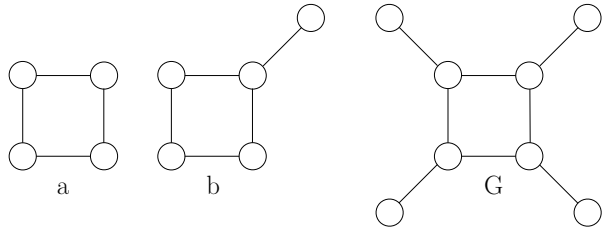


Figure 12: Example of non anti-monotoniticy of the support

would be the support of pattern a in the graph dataset G? The answer is simple, it is 1. Then, what would be the support of pattern b? Simple again, 4. This simple example shows that definitions based on the number of occurrences of the pattern in the graph do not work well, as they are not anti-monotone, To address this problem, Kuramochi and Karypis (63) as well as Fiedler and Borgelt (35) studied anti-monotonic support measures based on computing maximum independent sets (*MIS*) in overlap graphs, which is NP-complete.

Anti-monotonicity is then not the only requirement for efficient frequent pattern mining. It is also important that the frequency measure can be evaluated efficiently. We argue that the computation of overlap-based

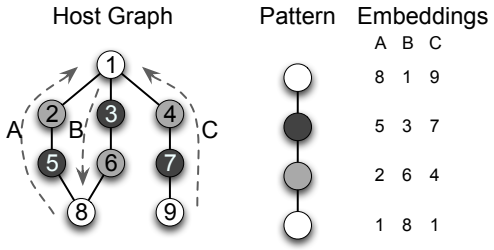


Figure 13: A graph with three different occurrences of a pattern evaluates to $\sigma = 2$.

support measures is not feasible in many graph databases, and that more scalable support measures are needed to enable the use of frequent graph mining algorithms on network data. Bringmann and Nijssen (21) propose a new support measure, and provide practical and theoretical evidence that this measure is more scalable, more general and more widely applicable than the support measures mentioned earlier. This measure is based on the number of unique nodes in the graph $G = (V_G, E_G)$ that a node of the pattern $P = (V_P, E_P)$ is mapped to, and defined as follows:

Definition 16 (Support)

$$\sigma(P, G) = \min_{v \in V_P} |\{\varphi_i(v) : \varphi_i \textit{ is an occurrence of } P \textit{ in } G\}|$$

By taking the node in P as reference which is mapped to the least number of unique nodes in G, the anti-monotonicity of the measure is ensured.

An example of minimum image based support is given in Fig. 13(a). Even if the pattern has 3 occurrences in the host graph, it has support $\sigma = 2$. In fact the lower white node of the pattern can only be mapped to nodes 1 and 8 in the host graph.

The advantage of this definition over other definitions introduced (34; 63) is twofold. From a practical point of view it is computationally easier to calculate since it does not require the computation of all possible occurrences of a pattern-graph in the host-graph. Additionally it does not require to solve a maximal independent set problem for each candidate pattern. From a theoretical perspective we know that this definition is an

upper bound for the according overlap based definitions (21; 34). Hence the support according to this definition is closer to the real number of occurrences in the graph.

In the remainder of the thesis, and in particular in chapters 4 and 5, when we are in the single graph setting, we assume the use of this support measure.

3.6.2 State of the art

This section extends section 3.4 with a brief overview of some approaches to the problem of finding frequent subgraphs in a single large graph.

The most well-known algorithm for finding recurring subgraphs in a single large graph is the SUBDUE system, originally developed in 1994, and improved over the years (25; 26). We have presented SUBDUE in Section 3.4. Ghazizadeh and Chawathe developed an algorithm called SEuS (36) that uses a data structure called summary to construct a lossy compressed representation of the input graph. This summary is obtained by collapsing together all the vertices of the input graph that have the same label and is used to quickly prune infrequent candidates. As the authors indicate, this summary data-structure is useful only when the input graph contains a relatively small number of frequent subgraphs with high frequency, and is not effective if there are a large number of frequent subgraphs with low frequency. Vanetik et al. (92) presented an algorithm for finding all frequently occurring subgraphs from a single labeled undirected graph using the maximum number of edge-disjoint embeddings of a graph as a measure of its frequency. Each subgraph is represented by its minimum number of edge-disjoint paths (path number) and use a level-by-level approach to grow the patterns based on their path-number. Their emphasis is on efficient candidate generation and no special attention is paid for frequency counting. In 2005, Kuramochi and Karypis (63) presented two computationally efficient algorithms that can find subgraphs which are frequently embedded within a large sparse graph. The first algorithm, called HSIGRAM, follows a horizontal approach and finds the frequent subgraphs in a breadth-first fashion, whereas the second

algorithm, called VSIGRAM, follows a vertical approach and finds the frequent subgraphs in a depth-first fashion. These algorithms incorporate efficient algorithms for candidate generation and frequency counting that allow them to scale to graphs containing over 120,000 vertices and find patterns with relatively low occurrence frequency. The experimental evaluation on eight real graphs shows that both HSIGRAM and VSIGRAM achieve reasonably good performance, scale to large graphs, and substantially outperform previously developed approaches for solving similar or simpler versions of the problem. In 2008, Bringmann and Nijssen (21), together with the aforementioned definition of support, developed a gSpan-based algorithm for mining frequent substructures in a single graph. Basically, they simply replaced the gSpan definition of support, intented so solve the problem in the transactional setting, with their definition, in order to deal with the single graph case. This algorithm can be further expanded to deal with constraints, as we see in section 3.6.4, and it is used in chapter 4 for mining the information propagation in a network, and in chapter 5, in a modified version, in order to solve the problem of mining patterns in an *evolving* graph.

3.6.3 Pushing constraints

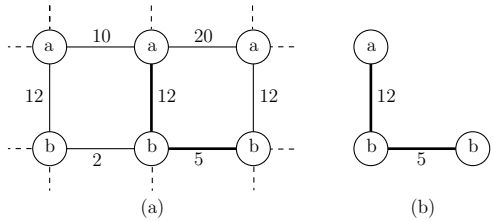


Figure 14: A portion of a single graph and a possible pattern.

In section 3.5 we have shown how to push monotone and anti-monotone constraints in graph mining in the transactional setting, both at preprocessing stage and at mining stage. Unfortunately, dealing with constraints in the single graph setting is not as easy as in the transactional.

Recall from Proposition 2 in section 3.5, that a transaction (i.e., a graph of the database, in that setting) which does not satisfy the given monotone constraint can be deleted from the input database since it will never contribute to the support of any solution substructure. Applying this proposition to the single graph case in not straightforward, as we should first define what a transaction is in this setting. Basically, we can keep saying that each graph in our database represents a transaction. In the single graph setting, this translates in having only one transaction. With this view, it is clear that Proposition 2 still holds: if the sole transaction we have does not satisfy the monotone constraints we can just delete it from our input. However, this approach turns out not to be really helpful, and the intuition suggest that we can do more than this.

Thus, what exactly can be pruned in a single graph? For example, consider Figure 14, showing in (a) a portion of an input graph, and in (b) a subgraph pattern (drawn in bold in (a)). Consider the monotone constraint $sum(E, \geq, 20)$, i.e., we want the sum of the edge labels in the patterns to be greater than or equal to 20. It is clear that we can not simply remove the edges in bold in Figure (a) since considering larger patterns by including, for example, the edge labeled with "20", would result in a pattern satisfying the constraint. However, if there is a connected component of the graph not satisfying the monotone constraint, it can be safely pruned according to the Proposition 4: in fact, this proposition remains still valid as is in the single graph setting, where the only difference with respect to it is that the database contains only one graph.

Now consider Proposition 3, Figure 14 and the anti-monotone constraint $sum(E, \leq, 15)$. It is clear that the edge $(a, 20, a)$ does not satisfy the constraint, and no pattern that contains it will satisfy it. Thus, exactly as in the transactional setting, it can be safely removed from the data. As one can see, this operation is in fact possible for all the anti-monotone constraints. That is, Proposition 3 holds in the single graph setting as well as the other two.

To summarize, we have just shown that all the three propositions that allow for data reduction are still valid in the single graph case, even though Proposition 2 is not as helpful as in the transactional case, as we

have only one graph that can be pruned here, and that Proposition 4 can be applied in the only graph we have (which can actually consist of many connected components).

3.6.4 An Algorithm for Constraint-Based Graph Mining in Single Graph Setting

At this point, defining a generic miner for solving the Constraint-Based Frequent Subgraph Mining problem in the single graph setting is straigthforward. Recall algorithms 5 and 7. Here, the only changes we need to apply are in the two lines 8. In these two lines, in fact, the frequency computation of the patterns are performed. These lines need to implement one of the possible support functions (for example, the minimum image based support function aforementioned). Moreover, the algorithms call the $\mathcal{G}amp$ loop, that in turns calls Algorithm 3 to perform antimonotone pruning. Line 2 of this algorithm should be changed according to the chosen support definition.

In chapters 4 and 5, we show the results obtained by pushing a few constraints into the computation, for dealing with two specific applications: mining the information propagation in a network, and mining frequent patterns of evolution in a single graph. In these two chapters, instead of showing the performances of the generic algorithm, we show two specific applications, the constraints we needed in those analysis, and the results of applying the miner to real-life datasets.

Chapter 4

Mining the Information Propagation in a Network

In the previous chapters we have introduced the general setting we are analyzing in the thesis, namely graph and network data showing a temporal dimension, and the basic technique we could use to discover interesting patterns in this setting, namely Graph Mining. In this chapter, we present possible applications of constraint-based Graph Mining in real life data. While in he previous chapter the focus was on a theoretical, general approach to the problem of finding frequent subgraphs in graph data with constraints, in this chapter and in chapter 5 we show how to implement and make use of generic and specialized constraint-based graph mining in real world applications. That is, the focus is on showing that starting from the general view of the previous chapter, specific applications may suggest and need for specific ad-hoc constraints, that need to be implemented in specialized ways, thus focusing in a particular set of results. In detail, we show the results obtained performing an analysis aimed at mining the information propagation in a network. For such analysis, we make use of three constraints:

- $vrtx[L, \supseteq]$: we want the labels of the vertices in the patterns to be in a specific set. Thanks to this constraint, we can achieve this goal by checking the node labels in the patterns. Moreover, for some of the

analyses performed, the set was totally ordered, which translated in constraining the labels to be between a minimum and a maximum label.

- $edge[L, \supseteq]$: similarly to the above constraint, we want the labels of the edges to be in a specific set. This can be easily achieved by checking this constraint on the edge labels.

- $size[S, \leq, \alpha]$: we want to restrict the size of the patterns (in number of edges) to be less than a maximum size specified as parameter. Please note that this constraint is both helpful and meaningful: being the search strategy of the miner a DFS, constraining the size of the patterns means to block the expansion of unwanted candidates, and it helps in assuring to achieve in feasible time all the results smaller than a specific size. On the other hand, it avoids the search for too large, non meaningful patterns, where the local properties are lost due to the large size.

In this chapter, in order to tackle the problem of finding frequent patterns of information propagation together with their causes in a network, Graph Mining is used in conjunction with TAS Mining, another mining technique already existing in the literature, presented in section 4.3.

4.1 On Mining the Information Propagation

In the last decade, the interest in Social Network Analysis topics from researchers in the Data Mining area has increased very fast. Much effort has been devoted, for example, in the Community Discovery, Leader Detection and Network Evolution problems (9; 19; 31; 84; 86). Another topic that has attracted much interest recently is how the information propagates over a network (3; 47; 54; 59; 66; 79). This problem has been studied from several points of view: statistics, modeling, mining are few of the approaches that have been applied so far in this direction. However, only a few answers to the questions "How does the information propagates over a network, why and how fast?" have been discovered so far. On

the other hand, these answers are of large interest, since they help in the tasks of finding experts in a network, assessing viral marketing strategies, identifying fast or slow paths of the information inside a collaborative network, and so on. In this chapter we study the problem of finding frequent patterns in a network focusing in two aspects:

- The temporal dimension intrinsically contained in the flow of information: why certain topics are spread faster than others? What is the distribution of the temporal intervals among the "hops" that the information passes through?
- The causes of the information propagation: why certain discussions are passed over while others stop in two hops? What are the characteristics of the nodes that pass the information?

As one can notice, the two dimensions of our focus are orthogonal to each other: certain nodes with certain characteristics may let a particular kind of information spread faster or slower than other nodes, or compared to information with other characteristics. The combination of the two aspects finds several possible application in real life. Among all of them, we believe that Viral Marketing can be powerfully enhanced by such kind of analysis. Companies willing to advertise a new product in their network of users may discover that giving a certain kind of information or special offers to a particular set of selected nodes may result in a cheaper or more effective advertisement campaign.

In this chapter, we study the above problem on the well known Enron email dataset (57), and the 20 Newsgroup dataset (52; 70), and with the help of two different techniques: TAS (Temporally Annotated Sequences) mining, which is a paradigm aimed at extracting sequential patterns where each transition between two events is annotated with a typical transition time that emerges from input data, and Graph Mining, which is helpful for locally analyzing the nodes of the networks with their properties.

The contribution of this chapter can be summarized as follows: we show how to extract useful information from a network in order to mine the information propagation, in the format of a graph where nodes are

users and edges are words used as email subjects, and a set of timestamped sequences of emails grouped by threads; we show how to apply the two techniques above to a real-life dataset with the aim of mining a particular aspect of the temporal dimension in graph and network data, namely the information propagation; we present the preliminary results obtained by applying the two algorithms on graphs extracted from the datasets, and on the sequences of exchanged email, showing a general methodology that can be applied in any sort of network where an exchange of information is present.

The remainder of the chapter is organized as follows: section 4.2 defines the problem under investigation and which kind of data we want to analyze; section 4.3 reviews the TAS mining paradigm together with some scenarios of application; section 4.4 shows the preliminary results obtained during our analysis of the datasets; section 4.5 briefly summarizes the results of our work and some possible future work.

4.2 Problem definition

We are given a dataset $\mathcal{D}$ of activities in a network, from which we can extract both a network of users $\mathcal{U}$ as a graph $\mathcal{G}$ and a flow of any kind of information (emails, documents, comments, instant messages, etc.) as a set of timestamped sequences $\mathcal{S}$. Examples of such datasets can be a set of emails exchanged among people, the logs of an instant messaging service, the logs of a social networking system, the content of a social bookmarking site, and so on. In this dataset we are interested in finding frequent patterns of information propagation, and we want to let the causes of such patterns emerge from the data. We then want to compare these rich patterns with the local patterns found in the graph $\mathcal{G}$, to see how the characteristics of the nodes interact both with the information spread and with the interactions of the nodes with their local communities in the network.
We assume $\mathcal{D}$ to contain at least the information about:

- a set of users with their characteristics (such as: gender, country,

age, typical discussed topics, degree, betweenness and closeness centrality computed over the network, and so on)

- a timestamped set of sequences of actions performed by the above users that involve the propagation of a certain kind of information (such as: exchange of emails, posts in a forum, instant messages, comments in a blog, and so on)

From the first, we can build several kinds of graphs that can be analyzed with classical graph mining techniques. In order to mine and analyze the local communities of the nodes with the focus on the spread of information, we want to build such graphs on the basis of the information exchanged among the nodes. As an example, the nodes of the graph can be the users of a mailserver, while there is an edge between two nodes if the nodes exchanged an email. The edge can be then labeled with the typical words used in the communications, that may be also grouped semantically or by statistical properties. Depending on the characteristics of the users and the way we consider them connected among each other, we are able to perform Social Network Analysis of the original network from several different points of view. For example, we may want to use as vertex labels the gender, the country and the age if we are analyzing a so called web social network, while we may want to use structural properties such as the degree, the closeness centrality, the betweenness or the clustering coefficient, if we are analyzing a network of a company. Each different combination of properties would result in a different kind of analysis.

From the second, we can derive a set of timestamped sequences to use as an input of the TAS mining paradigm (see Section 4.4), in order to be able to extract sequences of itemsets (i.e. characteristics of the users) that are found frequent in the data, together with frequent temporal annotations for them.

The entire analysis will be an interactive and iterative loop of the following steps:

1. Building a graph $\mathcal{G}$ of users in $\mathcal{U}$, connected by edges representing typical words or topics discussed by or among them

2. Assigning labels to the users in $\mathcal{U}$ according to their semantical (such as age, gender, newsgroup of major activity, preferred topic, etc.) and statistical (computed in $\mathcal{G}$, such as betweenness, closeness centrality, etc.) characteristics, collecting them in a set $\mathcal{L}$

3. Assigning labels to the edges in $\mathcal{G}$ according to their semantical (such as semantical cluster, etc.) or statistical (such as frequency of the stemmed word or topic in the subjects, etc.) characteristics, collecting them in a set $\mathcal{W}$

4. Extracting the flows of information in $\mathcal{D}$, grouped by any property to use as transaction identifier (thread, email subject, conversation ID, ..), and building a set of temporally annotated sequences $\mathcal{S}$, containing both the information on the users involved in each flow (represented as itemsets of labels in $\mathcal{L}$), and the temporal information about the flow (usually found as timestamps in seconds since the Epoch)

5. Extracting frequent Temporally Annotated Sequences $\mathcal{T}$ from $\mathcal{S}$, representing the frequent flows of information, and containing both the temporal dimension of the patterns, and the characteristics of the users involved

6. Extracting frequent subgraphs from $\mathcal{G}$ with the help of classical Graph Mining, that represent the local communities of nodes together with their characteristics and typical words or topics used

7. Analyzing the results produced in 4 in order to find frequent items (users' caracteristics) associated with typical fast or slow transition times, then analyze the patterns produced in 6 in order to find patterns containing nodes with the same characteristics as labels: these patterns will tell if the users with these characteristics are the best ones in spreading fast the type of information described by the graph patterns

Steps 1, 2, and 3, are clearly crucial and may vary the analysis that will be performed. By setting different labelings for the edges in $\mathcal{G}$ and

including or excluding different characteristics as vertex labels in 6, the analyst may drive the search for frequent patterns in different directions.

4.3 The TAS mining paradigm

In this section we briefly summarize the main aspects of the $\mathcal{TAS}$ mining paradigm, developed and widely used in the last few years (10; 11; 37; 39). This technique is used both in this chapter and in chapter 6.

$\mathcal{TAS}$ are sequential patterns where each transition between two events is annotated with a typical transition time that is found frequently in the data. In principle, this form of pattern is useful in several contexts: for instance:

- in web log analysis, different categories of users (experienced vs. novice, interested vs. uninterested, robots vs. humans) might react in similar ways to some pages - i.e., they follow similar sequences of web access - but with different reaction times
- in medicine, the relationship in time between the onset of patients' symptoms, drug consumption, and response to treatments
- in workflow logs, the typical data is a sequence of operations performed with specific moments, and from this, it could be interesting to extract frequent sequences containing frequent temporal annotations.

$\mathcal{TAS}$ patterns have been also used as building blocks for a truly spatio-temporal trajectory pattern mining framework (38). In all these cases, enforcing fixed time constraints on the mined sequences is not a solution. It is desirable that typical transition times, when they exist, emerge from the input data.

Time in FSP (Frequent Sequence Patterns) is mainly considered:

- for the sequentiality that it imposes on events
- as a basis for user-specified constraints to the purpose of either preprocessing the input data into ordered sequences of (sets of) events

- as a pruning mechanism to shrink the pattern search space and make computation more efficient.

In all of these cases, time is not explicitly returned in the ouput as timestamps or timestamped intervals, although in some cases interval precedence and overlap is expressed (55; 56; 71; 75; 76; 82; 97). For these reason, the $\mathcal{TAS}$, a form of sequential patterns annotated with temporal information representing typical transition times between the events in a frequent sequence, was introduced in (39).

Definition 17 ($\mathcal{TAS}$) *Given a set of items $\mathcal{I}$, a* temporally-annotated sequence *of length $n > 0$, called n-$\mathcal{TAS}$ or simply $\mathcal{TAS}$, is a couple $T = (s, \alpha)$, where $s = \langle s_0, ..., s_n \rangle$,*
$\forall_{0 \leq i \leq n} s_i \in 2^{\mathcal{I}}$ is called the sequence, *and $\alpha = \langle \alpha_1, ..., \alpha_n \rangle \in \mathbb{R}_+^n$ is called the* (temporal) annotation. *$\mathcal{TAS}$ will also be represented as follows:*

$$T = (s, \alpha) = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} ... \xrightarrow{\alpha_n} s_n$$

Example 4 *In a temporally-annotated sequence referring to process mining, shown above, the operations are represented as items, and the typical transition time between two consecutive operations are indicated as annotations. E.g.:*

$$Download(foo) \xrightarrow{1} Update(foo) \xrightarrow{5} Upload(foo) \xrightarrow{35} Delete(foo)$$

represents a sequence of workflow operations of the object 'foo,' which is downloaded at a certain time, updated after 1 second, then uploaded after 5 seconds, and finally deleted after 35 seconds.

Similar to traditional sequence pattern mining, it is possible to define a containment relation between annotated sequences:

Definition 18 (τ-containment ($\preceq_\tau$)) *Given a n-$\mathcal{TAS}$ $T_1 = (s_1, \alpha_1)$ and a m-$\mathcal{TAS}$ $T_2 = (s_2, \alpha_2)$ with $n \leq m$, and a time threshold τ, we say that T_1 is τ-contained in T_2, denoted as $T_1 \preceq_\tau T_2$, if and only if there exists a sequence of integers $0 \leq i_0 < ... < i_n \leq m$ such that:*

1. $\forall_{0 \leq k \leq n}. \; s_{1,k} \subseteq s_{2,i_k}$
2. $\forall_{1 \leq k \leq n}. \mid \alpha_{1,k} - \alpha_{*,k} \mid \leq \tau$

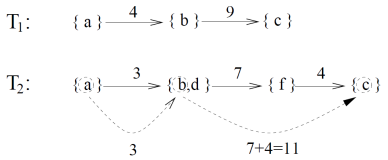


Figure 15: Example of τ-containment computation

where $\forall_{1 \leq k \leq n}.\ \alpha_{*,k} = \sum_{i_{k-1} < j \leq i_k} \alpha_{2,j}$. *As special cases, when condition 2 holds with the strict inequality we say that* T_1 *is* strictly τ-contained in T_2, *denoted with* $T_1 \prec_\tau T_2$, *and when* $T_1 \preceq_\tau T_2$ *with* $\tau = 0$ *we say that* T_1 *is* exactly contained *in* T_2. *Finally, given a set of TAS* D, *we say that* T_1 *is* τ*-contained in* D *(*$T_1 \preceq_\tau D$*) if* $T_1 \preceq_\tau T_2$ *for some* $T_2 \in D$.

Essentially, a *TAS* T_1 is τ-contained into another, *TAS* T_2 if the former is a subsequence of the latter and its transition times do not differ too much from those of the corresponding itemsets in T_2. In particular, each itemset in T_1 can be mapped to an itemset in T_2. When two itemsets are consecutive in T_1, but their mappings are not consecutive in T_2, the transition time for the latter couple of itemsets is computed by summing up the times of all the transitions between them, which is exactly the definition of the annotations α_*. The following example describes a sample computation of τ-containment between two *TAS*:

Example 5 *Consider two TAS:*

$$T_1 = (\langle \{a\}, \{b\}, \{c\} \rangle, \langle 4, 9 \rangle)$$
$$T_2 = (\langle \{a\}, \{b, d\}, \{f\}, \{c\} \rangle, \langle 3, 7, 4 \rangle)$$

also depicted in Figure 15, and let $\tau = 3$. *Then, in order to check if* $T_1 \preceq_\tau T_2$, *we verify that:*

- $s_1 \subset s_2$*:the first and the last itemsets of* T_1 *are equal to the first and the last itemsets of* T_2 *respectively, while the second itemset of* T_1 *(*$\{b\}$*) is strictly contained in the second one of* T_2 *(*$\{b,d\}$*).*
- *The transition times between* T_1 *and its corresponding subsequence in* T_2 *are similar: the first two itemsets of* T_1 *are mapped to contiguous itemsets in* T_2, *so we can directly take their transition time in* T_2, *which is equal*

to $\alpha_{*,1} = 3$ (from $\{a\} \xrightarrow{3} \{b,d\}$ in T_2). The second and third itemsets in T_1 are instead mapped to non-consecutive itemsets in T_2, and so the transition time for their mappings must be computed by summing up all the transition times between them, i.e.: $\alpha_{*,2} = 7+4 = 11$ (from $\{b,d\} \xrightarrow{7} \{f\}$ and $\{f\} \xrightarrow{4} \{c\}$ in T_2). Then, we see that $\mid \alpha_{1,1} - \alpha_{*,1} \mid = \mid 4-3 \mid < \tau$ and $\mid \alpha_{1,2} - \alpha_{*,2} \mid = \mid 9-11 \mid < \tau$.

Therefore, we have that $T_1 \preceq \tau T_2$. Moreover, since all inequalities hold strictly, we also have $T_1 \prec_\tau T_2$.

Now, frequent sequential patterns can be easily extended to the notion of frequent $\mathcal{TAS}$:

Definition 19 (Frequent $\mathcal{TAS}$) *Given a set $\mathcal{D}$ of $\mathcal{TAS}$, a time threshold τ and a minimum support threshold σ, we define the τ-*support *of a $\mathcal{TAS}$ T as $supp_{[\tau,\mathcal{D}]}(T) = \mid \{T^* \in \mathcal{D} \mid T \preceq_\tau T^*\} \mid$ and say that T is* frequent *in $\mathcal{D}$, given a minimum support threshold σ if $supp_{[\tau,\mathcal{D}]}(T) \geq \sigma$.*

It should be noted that a frequent sequence s may not correspond to any frequent $\mathcal{TAS}$ $T = (s, \alpha)$: indeed, its occurrences in the database could have highly dispersed annotations, thus not allowing any single annotation $\alpha \in \mathbf{R}^n_+$ to be close (i.e. similar) enough to a sufficient number of them. That essentially means s has no *typical* transition times.

At this stage, introducing time in sequential patterns gives rise to a novel issue: intuitively, for any frequent $\mathcal{TAS}$ $T = (s, \alpha)$, we can usually find a vector ϵ of small, strictly positive values such that $T' = (s, \alpha + \epsilon)$ is frequent as well, since they are approximatively contained in the same $\mathcal{TAS}$ in the dataset, and then have very similar τ-support. Since any vector with smaller values than ϵ (e.g., a fraction ϵ/n of it) would yield the same effect, we have that, in general, the raw set of all frequent $\mathcal{TAS}$ is highly redundant (and also not finite, mathematically speaking), due to the existence of several very similar - and then practically equivalent - frequent annotations for the same sequence.

Example 6 *Given the following toy database of $\mathcal{TAS}$:*

$$a \xrightarrow{1} b \xrightarrow{2.1} c \quad a \xrightarrow{1.1} b \xrightarrow{1.9} c$$
$$a \xrightarrow{1.2} b \xrightarrow{2} c \quad a \xrightarrow{0.9} b \xrightarrow{1.9} c$$

if $\tau = 0.2$ *and* $s_{min} = 0.8$ *we see that* $T = a \xrightarrow{1} b \xrightarrow{2} c$. *In General, we can see that any* $a \xrightarrow{\alpha_1} b \xrightarrow{\alpha_2} c$ *is frequent whenever* $\alpha_1 \in [1, 1.1]$ *and* $\alpha_2 \in [1.9, 2.1]$.

This problem is solved by the $\mathcal{TAS}$ mining software developed in (39), by finding *"dense regions"* of annotations, and thus grouping together $\mathcal{TAS}$ patterns accordingly. The output of this process is a set of $\mathcal{TAS}$ patterns where the annotations are no longer points in $\mathbb{R}^n_+$, but instead are intervals: e.g.,

$$a \xrightarrow{[1,1.1]} b \xrightarrow{[1.9,2.1]} c$$

We say that this resulting pattern can be read as: the sequence *a, b, c* is frequently executed, with typical transition times of $t_1 \in [1, 1.1]$ for the first transition and $t_2 \in [1.9, 2.1]$ for the second one.
The MiSTA software gives the user a special parameter *strict*: by means of this parameter it is possible to look for sequences made of *strictly consecutive* tasks. For example, having a sequence $a \to b \to c \to d$, it is possible to obtain, say, $a \to b \to c$ and $c \to d$, but not $a \to d$. This parameter is useful when looking for tasks which are performed consecutively without gaps among them. In addition to this, it has a positive side effect in terms of performance: since every sequence that does not match this constraint is removed from the search space at runtime, the problem of extracting $\mathcal{TAS}$ becomes more feasible. For more details see (37; 39).

4.3.1 $\mathcal{TAS}$-based Mining

In (10; 11) it is shown how it is possible to apply the $\mathcal{TAS}$ mining paradigm to medical data when its structure is a sequence of clinical observations taken at different times. In this context the temporal dimension of the data is a variable that should be taken in account in the mining process and returned as part of the extracted knowledge. In these papers a real-world medical case study was reported in which the $\mathcal{TAS}$ mining paradigm was applied to such a data.

In (38), the authors introduce a novel spatio-temporal pattern that formalizes the idea of aggregate movement behaviour. In their approach a trajectory pattern is a sequence of spatial regions that, on the basis of the

source trajectory data, emerge as frequently visited in the order specified by the sequence; in addition, the transition between two consecutive regions in this sequence is annotated with typical travel times that emerge from the input trajectories.

4.4 Case study

4.4.1 Dataset

We used for our experiments two e-mail datasets. The first one is the Enron email dataset(57). This dataset contains 619,446 email messages complete with senders, recipients, cc, bcc, and text sent and received from 158 Enron's employees. It was published in 2004 in response to inquiries regarding the well known fraudulent bankruptcy. This dataset is characterized by an exceptional wealth of information, and it enables the tracking of chains of communication, together with their associated subjects and the complete data regarding the exchange of information.

We took from the entire dataset the "from", "to", "cc", "bcc", "subject" and "date" fields in each email in the "sent" folder of every employee. We took only the emails that were sent to other Enron employees, removing the outgoing emails. We also performed basic cleaning by removing emails with empty subjects, noise, and so on. After the cleaning stage, the number of remaining emails was about 12,000. We refer to it as the "Enron" dataset.

The second dataset consists of Usenet articles collected from 20 different newsgroups about general discussions on politics and religion, technical discussions on computers and hardware, general discussions on hobbies and sports, general discussion on sciences, and a newsgroup for items on sale, and was first used in (52; 70). Over a period of time, 1000 articles were taken from each of the newsgroups, which makes an overall number of 20,000 documents in this collection. Except for a small fraction of the articles, each document belongs to exactly one newsgroup. We took from each sent email the "from", "to" and "date" field. After a cleaning stage, the number of remaining emails was about 18,000. We refer to it as

the "Newsgroup" dataset.

4.4.2 Tools

For our analysis, we used the MiSTA software (37; 39), which extracts frequent Temporally Annotated Sequences from a dataset of timestamped sequences; we also used a single graph miner in order to find frequent subgraphs of a large graph, implementing a Minimum Image Support function as described in (21).
All the experiments were conducted on a machine equipped with 4 processors at 3.4GHz, 8GB of RAM, running the Ubuntu 8.04 Server Edition, and took from seconds to minutes for the TAS mining, and from minutes to hours for the Graph Mining.

4.4.3 Steps of Analysis

We then followed the steps described in section 4.2 in order to perform our analysis. In the following steps, the subscripts E and N indicate wether the sets refer to the Enron or Newsgroup datasets, respectively.

As step 1, we built the graph $\mathcal{G}_\mathcal{E}$ for the Enron dataset by taking the users as nodes and connecting two nodes with edges representing the subjects of emails exchanged between them. For the Newsgroup dataset, we built $\mathcal{G}_\mathcal{N}$ by taking the users as nodes and connecting two nodes with edges representing the subjects for which both users posted a message to the newsgroups.

As step 2, we labeled the users $\mathcal{U}_\mathcal{E}$ and $\mathcal{U}_\mathcal{N}$ following five different possible labeling, according to their structural characteristics in the graphs $\mathcal{G}_\mathcal{E}$ and $\mathcal{G}_\mathcal{N}$: the degree (the number of ties to other nodes in the network, referred as "DEG"), the closeness centrality (i.e., the inverse of the distance in number of edges of the node from all other nodes in the network, referred as "CL"), the betweenness centrality (i.e., the number of geodesic paths that pass through the node, referred as "BET") and two different clustering annotations (following two different clustering strategies, referred as "CC1" and "CC2"). Table 4.4.3 shows the labeling according to the real values of these variables. For users in $\mathcal{U}_\mathcal{N}$ we also performed a

labeling according to the newsgroup in which the user was most active, assigning thus 20 possible labels for each node.

As step 3, the edges in $\mathcal{G}_{\mathcal{E}}$ and $\mathcal{G}_{\mathcal{N}}$ have been assigned a label according to various criteria. Both for the Enron and the Newsgroup datasets, the most frequent words in the subjects were manually clustered by their semantic in 5 different clusters per dataset. Each edge was then labeled with the most frequent cluster among its words (ignoring the words not belonging to any cluster). The edges corresponding to subjects for which none of the contained words was frequent or was not belonging to any of the clusters were removed from the graphs. We refer to the graphs created in this way as "Enron S" and "Newsgroup S". For the Newsgroup dataset we performed also a different labeling: all the words were divided in three frequency classes and the edges were then labeled accordingly. We refer to this graph as "Newsgroup F". Finally, in each graph, multiple edges between two nodes have been collapsed into a single edge labeled with its more frequent label. Table 5 shows some statistical properties of the graphs generated, in which: n is the number of nodes, e the number of edges, $\bar{k}$ the average degree, #*Components* the number of components, *GiantComponent* the size of the largest component of the graph (percentage of the total number of nodes), $\bar{C}$ the average clustering coefficient of the graph (betweem 0 and 1), ℓ the average length of the shortest paths in the graph and *Diameter* the length of the longest shortest path in the graph.

For the step 4, in order to build our $\mathcal{S}_{\mathcal{E}}$ and $\mathcal{S}_{\mathcal{N}}$ for the TAS mining paradigm, we grouped all the emails by subject, keeping the timestamp given by the mailserver to every email. Figure 21 is a graphical representation of the flow of emails in Enron with initial subject "2002 capital plan". In order to give this in input to the software, we processed each of these flow by splitting it in all the possible sequences of emails passed from an user to the others, following the natural temporal ordering. This last step was not necessary for Newsgroup, as the emails were sent only to one recipient, namely the newsgroup. The complete set of these timestamped sequences constituted then our $\mathcal{S}_{\mathcal{E}}$ and $\mathcal{S}_{\mathcal{N}}$.

Steps 5 and 6 produced the results in the following paragraph.

Enron					
Label	Degree	Closeness	Betweenness	CC1	CC2
1	$[0, 5]$	$[0, 0.21[$	$[0, 0.0015[$	0	0
2	$[6, 15]$	$[0.21, 0.2329[$	$[0.0015, 0.0046[$	$]0, 0.2[$	$]0, 35e^{-6}[$
3	$[16, 33]$	$[0.2329, 0.2513[$	$[0.0046, 0.013[$	$[0.2, 0.34[$	$[35e^{-6}, 14e^{-5}[$
4	$[34, 75]$	$[0.2513, 0.267[$	$[0.013, 0.034[$	$[0.34, 0.67[$	$[14e^{-5}, 61e^{-5}[$
5	$[76, +\infty[$	$[0.267, 1]$	$[0.034, 1]$	$[0.67, 1]$	$[61e^{-5}, 1[$

Newsgroup					
Label	Degree	Closeness	Betweenness	CC1	CC2
1	$[0, 15]$	0	0	0	0
2	$[16, 39]$	$]0, 0.12[$	$]0, 0.0002[$	$]0, 0.42[$	$]0, 0.00015[$
3	$[40, 84]$	$[0.12, 0.145[$	$[0.0002, 0.001[$	$[0.42, 0.61[$	$[0.00015, 0.00085[$
4	$[85, 154]$	$[0.0002, 0.001[$	$[0.001, 0.002[$	$[0.61, 1[$	$[0.00085, 0.005[$
5	$[155, +\infty[$	$[0.1632, 1]$	$[0.002, 1]$	1	$[0.005, 1[$

Table 4: The labels assigned to the users in the datasets

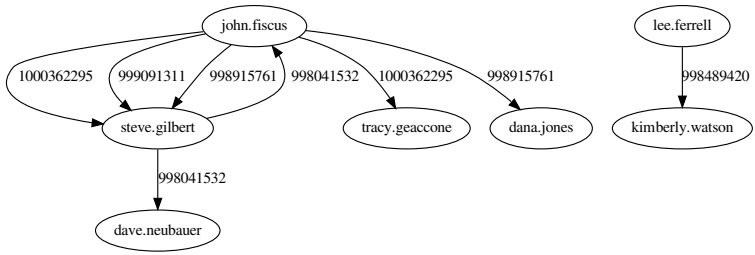


Figure 16: Example of mail flow for the subject "2002 capital plan"

4.4.4 Results

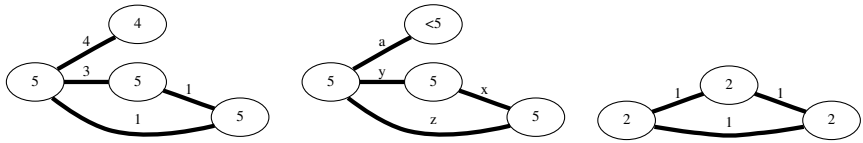
Although the focus in this chapter is only to show the power of the combination of the two techniques we used in our analysis to graph data showing a temporal dimension, we can make some interpretation of some of the resulting patterns, that clearly show the differences between the two datasets.

Graph Mining.

The Enron graph represents interactions in the working environment of a company, from which we can infer particular considerations regard-

Graph	n	e	$\bar{k}$	Comps	Giant Component	$\bar{C}$	ℓ	Diameter
Enron S	3731	9543	5.11	30	98.01%	0.17	4.52199	15
Newsgroup S	1457	12560	17.24	151	64.51%	0.78	4.02730	11
Newsgroup F	3923	31632	16.12	249	82.41%	0.73	4.42142	17

Table 5: The dataset statistics.



(a) (b) (c)

Figure 17: Subgraphs found in Enron dataset

ing possible stages of the workflow followed by the employees. Contacts between employees are direct (not thus as in the newsgroup case), and are very often of the one-to-many type (many cc in an email).

The first pattern extracted, Figure 17a, represents an exchange of emails. The labels on the nodes represent the level of Clustering Coefficient 2, i.e. the tendency of an employee to create a working group around him or she. It can be notice that employees with high CC2 have a frequent exchange of emails with several subjects. At a certain moment, one of these high CC2 employees has a contact with a low CC2 employee (a node outside the central part of the graph beeing maybe a specific member of a work group) with a different subject. This pattern may represent the mechanism by which members acting as "bridges" between groups detect, with a mutual exchange of knowledge, who can solve a problem.

The pattern is even more interesting because there are many instances of the abstract pattern like the in Figure 17b, where the label "<5" means any label lower than 5.

Another interesting pattern in the Enron graph is represented in Figure 17c, where labels are assigned to nodes according to the first defini-

(a) (b)

Figure 18: Subgraphs found in newsgroups dataset

tion of clustering coefficient (CC1). You can see that nodes with a low clustering coefficient (often synonymous of high degree, so central nodes of the graph) tend to behave in constrast to their typical feature, as the nodes are found in a frequent clique. This happens when the semantics of the label of their connection is the cluster number 1, that is typical of managers and directors.

We finish the discussion on frequent graph patterns noting that, using labels such as the frequencies of the words of the topics of discussions for the edges and CC2 for nodes, users who create groups around themselves tend to use with people like them the same classes of words, but do not use the same words with users of other CC2 categories. This may be a typical mechanism of communication inter and intra-groups: within an entertainment community are often used "slang" terms less common (edge label = 1) that those that come from groups outside do not understand or use (and use more common and frequent words, edge label = 2). Examples of this behavior are patterns in Figure 18a and 18b.

TAS Mining.

We now present some considerations derived from the analysis of the most frequent temporal sequences extracted from the two datasets. Figure 19 is a graphical representation of a possible extracted pattern.

Consider graphs in Figure 20. The 20a and 20b graphs were generated by analyzing the average response time of the most frequent sequences (i.e. the most representative) according to different characteristics (De-

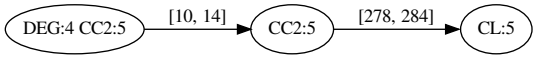


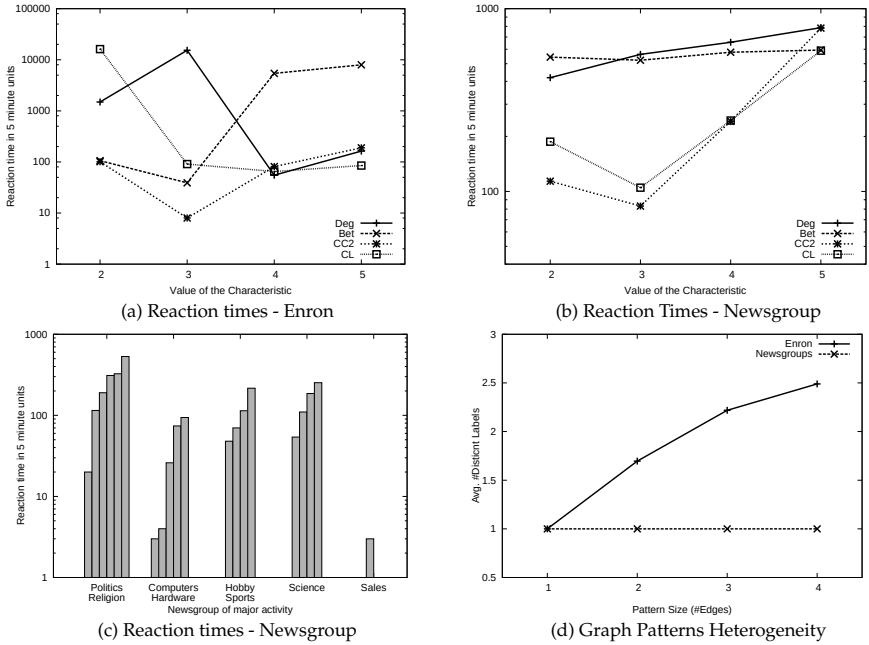
Figure 19: An example of TAS found

gree, Closeness, Betweenness and CC2) of the sender.

First, we can notice the difference of scale: in the Enron dataset there are higher average response times. This can be explained considering the different nature of the exchange of knowledge in a working-like communication: there are not (frequent) immediate answers, since, after an email, usually several stages of documentation, meetings and brainstorming, follow, enlarging the time needed for providing a response. On the opposite side, users within a social community usually only need to read all the messages before answering, leading to usual short response time.

Regarding the Enron dataset, Figure 20a reveals an important information regarding the response time of the employees with an high degree of betweenness centrality: having an high betweenness centrality for an employee means that many shortest paths in internal communications pass through that employee. The TAS mining revealed that this tends to result in much higher response times, due to the additional working burden that employees of this type have to face. The knowledge that can be extracted from this analysis of mining is to avoid, if possible, the structural hubs when there is the need to speed up a communication. This pattern of reaction times in relation to betweenness is completely absent in the Newsgroup dataset.

Regarding the Newsgroup dataset, Figure 20b, shows another difference of behavior from the Enron dataset: the higher regularity of growth of the reaction time for users with higher degree. The degree grows as the user follows many different discussions and, especially, when these discussions involve an increasing number of users. The mining stage showed that the typical response times go up because these discussions



(a) Reaction times - Enron

(b) Reaction Times - Newsgroup

(c) Reaction times - Newsgroup

(d) Graph Patterns Heterogeneity

Figure 20: Quantitative Analysis of the Results

are probably the most controversial and interesting ones, and in order to follow them, much time and attention have to be spent. These considerations are not necessarily true in a business context: an employee with a high degree (many different contacts) often respond very quickly.

Another consideration can be done w.r.t to the "newsgroup" labeling. Consider Figure 20(c). In the x axis we have the newsgroup of major activity of the users (i.e., a possible value of the "newsgroup" label for the nodes), clustered by main topics (x labels), while in the y the reaction times as found in the frequent TAS. Each of the 20 bars represents one particular newsgroup. As we can notice, there are differences in the reaction times according to the main topic of the newsgroups. While politics and religion seem to be general "relaxed" discussion topics, technical discussions in computers and hardware find more reactive answers. The

most reactive is the newsgroup where people put items for sale: the first offer in generally set after 5-10 minutes, as we can see from the figure.

Based on the above consideration, we can give a "draft" of what could be done in order to perform step 7 of the general approach described in section 4.2: once found that the "Sale" label could be a characteristic related to the speed of the users, it is possible to go back on the results of the graph mining and see if that label was found frequent, possibly in the center of a large subgraph pattern where other nodes have different labels. If the frequency of this pattern is found high, one can argue that passing the information to "Sale" nodes would result in a faster and effective spread of information. In this case study, the meaning of the "Sale" label does not really suggest anything special, but the focus here is to give an idea of the potentialities of the general approach followed in this Section.

Finally, consider Figure 20d that shows a direct comparison between the two datasets. It shows the degree of heterogeneity of the communication (i.e. the number of different semantic labels associated with the edges of the frequent patterns) compared with the volume of communication (number of edges in the pattern). From the graph it can be inferred that the business environment shows a greater heterogeneity in the communications: while in the Enron dataset employee tend to speak about different topics with their neighbors, in the Newsgroup dataset close users speak about the same topics. This seems quite easy to explain: employees usually manage more than one different situation, while users in newsgroups tend to be clustered by newsgroup, and hence by topics.

4.5 Discussion

In this chapter, we have shown a general methodology to mine the information propagation in a network where users exchange information. We have described how to extract useful information from such a network in order to be able to use a combination of two powerful techniques, namely TAS mining and Graph Mining, in order to find frequent patterns of propagation of information that involve also the possible causes of these propagation. We have shown how this combination can help

in finding frequent temporal behaviors in the network together with the characteristics of the users, and what are the roles of these users in the network. We have presented preliminary results of a case study on real-life datasets and we have provided a possible interpretation of some of these results.

While in this chapter the focus was on datasets of exchanged emails, one can easily see that this approach is easily applicable to any kind of network of users where flows of any kind of information are detectable. In particular, other possible uses of this approach include:

- the study of the temporal behavior of a network of users, with *a priori* knowledge of the social connections among them (that in this chapter were, on the other hand, learnt from the flows of emails themselves)
- the study of the influence that a user can have in a network, and the time needed for a user with certain characteristics to influence a certain kind of users
- the study of possible redundancy or bottlenecks in a network where information is spread
- the identification of particular kinds of users that could be replaced or moved from a position in the network to another, in order to optimize the information flow
- the study of spread of viruses

and many other related problems.

Chapter 5

Mining Graph Evolution Rules

While in the previous chapter we have focused in the analysis of *action*, in this chapter we show how to apply constraint-based graph mining in the context of *evolution*. In particular, here we deal with the evolution of real world social and bibliographic networks, topic that is increasing attention during the last few years.

For this kind of analysis, besides the $size[S, \leq, \alpha]$ constraint mentioned in the previous chapter, we wanted the pattern to express some evolution of the network. Thanks to the edge labeling, according to which the label represents the time in which the edge first appeared in the graph, this can be done by enforcing the expansion of patterns with at least two different edge labels. As we show later, this is not easily implementable as a simple constraint check as all the examples we showed so far in this thesis, but it requires a modification of the canonical form. Moreover, in this analysis, we modified also the matching operator of a gSpan-based graph miner that implements the subgraph isomorphism check, in order to be able to mine the evolution both with relative and with absolute timestamps. Again, this is much different than simply checking for a constraint during the pattern expansion.

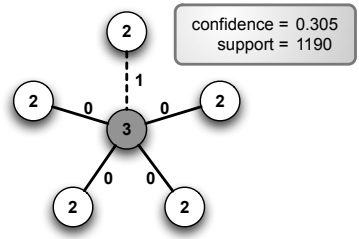


Figure 21: A Graph Evolution Rule extracted from the DBLP co-authorship network.

5.1 On Mining the Evolution of a Network

With the increasing availability of large social-network data, the study of the temporal evolution of graphs is receiving a growing attention. While most research so far has been devoted to analyze the change of global properties of evolving networks, such as the diameter or the clustering coefficient, not much work has been done to study graph evolution at a microscopic level. In this chapter, we consider the problem of searching for patterns that indicate local, structural changes in dynamic graphs. Mining for such local patterns is a computationally challenging task that can provide further insight into the increasing amount of evolving network data.

Following a frequent pattern-mining approach, we introduce the problem of extracting *Graph Evolution Rules* (*GER*), which are rules that satisfy given constraints of minimum support and confidence in evolving graphs. An example of a real *GER* extracted form the DBLP co-authorship network is given in Fig. 21: nodes are authors, with an edge between two nodes if they co-authored a paper.

In this specific example, the node labels represent a class of degree of the node: the higher the label the higher the degree of the node. It is important to note that the label refers to the degree of the node in the input graph, not in the rule. In particular the label 3 indicates a node with degree > 50. In general, node labels may represent any property of a node. The labels on the edges instead are more important as they represent the (relative) year in which the first collaboration between two

authors was established. Intuitively (later we will provide all the needed definitions) the rule might be read as a sort of local evidence of *preferential attachment*, as it shows a researcher with a large degree (label 3) that at time t is connected to four medium degree researchers (labels 2), and that at time $t+1$ will be connected to another medium degree researcher. The definition, extraction and subsequent empirical analysis of such *Graph Evolution Rules* (*GER*) constitute the main body of our work.

The remainder of the chapter is organized as follows: section 5.2 describes the problem under investigation and defines the novel kind of pattern we are interested in. In section 5.3 we describe the details of our algorithm. We report on our experimental results in section 5.4. Finally, in section 5.5 we discuss possible future research directions and in section 5.6 we discuss our approach.

5.2 Patterns of graph evolution

5.2.1 Time-evolving graphs

We start by describing how we *conceptually* represent an evolving graph, and subsequently discuss how to *actually* represent the graph in a more compact format. As usual the terminology $G = (V, E, \lambda)$ is used to denote a graph G over a set of nodes V and edges $E \subseteq V \times V$, with a labeling function $\lambda : V \cup E \rightarrow \Sigma$, assigning to nodes and edges labels from an alphabet Σ. These labels represent properties, and for simplicity we assume that they do not change with time. As an example, in a social network where nodes model its members, node properties may be $gender$, $country$, $college$, etc., while an edge property can be the kind of connection between two users. The evolution of the graph over time is conceptually represented by a series of undirected graphs $G_1, \ldots, G_T$, so that $G_t = (V_t, E_t)$ represents the graph at time t. Since $G_1, \ldots, G_T$ represent different snapshots of the same graph, we have $V_t \subseteq V$ and $E_t \subseteq E$. For simplicity of presentation, we assume that as the graph evolves, nodes and edges are only added and never deleted: i.e., $V_1 \subseteq V_2 \subseteq \ldots V_T$ and $E_1 \subseteq E_2 \subseteq \ldots E_T$.

It is worth noting that the number of edge deletions in social networks is so small to be negligible when analyzing the temporal evolution of networks. However, in our framework we can handle also deletions by slightly changing the matching operator as described in section 5.5.

Our mining algorithm represents the dataset by simply collapsing all the snapshots $G_1, \ldots, G_T$ in one undirected graph G, in which edges are *time-stamped* with their first appearance. Thus, we have $G = (V, E)$ with $V = \bigcup_{t=1}^{T} V_t = V_T$ and $E = \bigcup_{t=1}^{T} E_t = E_T$. To each edge $e = (u, v)$ a time-stamp $t(e) = \arg\min_j \{E_j \mid e \in E_j\}$ is assigned. Note that time-stamps on the nodes may be ignored as a node always comes with its first edge and hence this information is implicitly kept in edge time-stamps. Overall, a time-evolving graph is described as $G = (V, E, t, \lambda)$, with t assigning time-stamps to the set of edges E.

5.2.2 Patterns

Consider a time-evolving graph G, as defined above. Intuitively a pattern P of G is a subgraph of G that in addition to matching edges of G also matches their time-stamps, and if present, the properties on the nodes and edges of G.

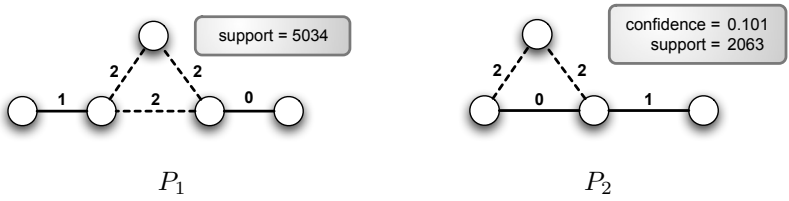
Definition 20 (Absolute-time pattern)
Let $G = (V, E, t, \lambda)$ and $P = (V_P, E_P, t_P, \lambda_P)$ be graphs, where G is the time-evolving dataset and P a pattern. We assume that P is connected. An occurrence *of P in G is a function $\varphi : V_P \to V$ mapping the nodes of P to the nodes of G such that for all $u, v \in V_P$:*

(i) $(u, v) \in E_P$ *it is* $(\varphi(u), \varphi(v)) \in E$,

(ii) $(u, v) \in E_P$ *it is* $t(\varphi(u), \varphi(v)) = t(u, v)$, *and*

(iii) $\lambda_P(v) = \lambda(\varphi(v)) \wedge \lambda_P((u, v)) = \lambda((\varphi(u), \varphi(v)))$

In case no labels are present for edges or nodes, the last condition (iii) is ignored. Two examples of patterns from the DBLP co-authorship network are shown in Fig. 22. Those examples motivate us to make two important decisions. First, since our goal in this chapter is to study patterns of evolution we naturally focus on patterns that refer to more than



P_1 P_2

Figure 22: Relative time patterns extracted from two different samples of the DBLP co-authorship network: respectively 1992-2002 for (P_1), and 2005-2007 (P_2). Dataset details are given in Sec. 5.4.1.

one snapshots such as the examples in Fig. 22. In other terms we are not interested in patterns where all edges have the same time-stamp. The second decision is based on the following observation. Consider pattern P_1: arguably, the essence of the pattern is the fact that two distinct pairs of connected authors, one collaboration created at time 0, and one at time 1, are later (at time 2) connected by a collaboration involving one author from each pair, plus a third author. We would like to account for an occurrence of that event even if it was taking place at times, say, 16, 17 and 18. To capture this intuition we define *relative-time patterns*.

Definition 21 (Relative-time Pattern) *Let G and P be a graph and pattern as in Definition 20. We say that P* occurs in G at relative time *if there exists a $\Delta \in \mathbb{R}$ and a function $\varphi : V_P \to V$ mapping the nodes of P to the nodes in G such that $\forall u, v \in V_P$*

(i) $(u, v) \in E_P$ *it is* $(\varphi(u), \varphi(v)) \in E$,

(ii) $(u, v) \in E_P$ *it is* $t(\varphi(u), \varphi(v)) = t(u, v) + \Delta$, *and*

(iii) $\lambda_P(v) = \lambda(\varphi(v)) \wedge \lambda_P((u, v)) = \lambda((\varphi(u), \varphi(v)))$

The difference between Definitions 20 and 21 is only in the second condition. As a result of Definition 21, we obtain naturally forming equivalence classes of structurally isomorphic relative time patterns that differ only by a constant on their edge time-stamps. To avoid the resulting redundancies in the search space of all relative time patterns we only

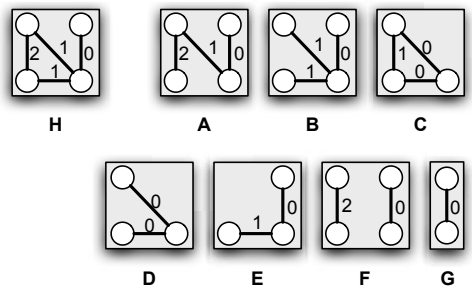


Figure 23: A graph H with relative edge labels and all possible relative subgraphs A, B, C, D, E, F, G.

pick one representative pattern for each equivalence class, namely *the one where the lowest time-stamp is zero.*

In the remainder of this chapter we focus on relative time patterns, as they subsume the absolute time case: they are both more interesting and more challenging to mine, as obviously more patterns are found to be frequent.

5.2.3 Rules and Confidence Measure

The support of a pattern (recall from section 3.6.1 that we are using a minimum image based support function) can provide insight into how often such an event may happen compared to other specific changes, but not how likely is a certain sequence of steps. To acquire this information we need to decompose a pattern into the particular steps and subsequently determine the *confidence* for each transition. Each step can be considered as a rule *body*→*head* with both *body* and *head* being patterns as defined in the previous section. Unfortunately, this does not yet solve our problem, but rather introduces two important questions:

1. How to decompose a pattern into *head* and *body*?

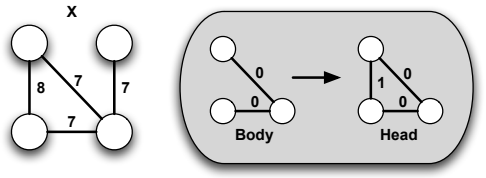
2. What are reasonable definitions of confidence?

Regarding the decomposition consider pattern H in Fig. 23. An occurrence of H implies an occurrence of all its sub-patterns $A-G$. Similarly to the definition of association rules all $A-G$ can be considered candidate-*body* in order to form a graph evolution rule with pattern H as *head*. Fortunately, most of those possibilities can be discarded immediately. First, we are interested in evolution and hence only care about rules describing edges emerging in the future. This allows us to discard bodies A, C, D, E, and F thus only leaving B and G. Furthermore, the step should be as small as possible to allow for a high granularity wherefore we would drop candidate-body G in the example, leaving B as body for the head H. Following the same reasoning, G would be the only choice as body for B as head. Similar the other rules in the example are $E \rightarrow A, D \rightarrow C, G \rightarrow E$. The natural body thus would be the head discarding all the edges from the last time-step of the target-pattern. More formally:

Definition 22 (Graph Evolution Rule) *Given a pattern head P_H the body P_B is defined as: $E_B = \{e \in E_H \mid t(e) < \max_{e^* \in E_H}(t(e^*))\}$ and $V_B = \{v \in V_H \mid deg(v, E_B) > 0\}$, where $deg(v, E_B)$ denotes the degree of v with respect to the edges in E_B. Moreover we constrain P_B to be* connected. *Finally, the support of a graph evolution rule is the support of its head.*

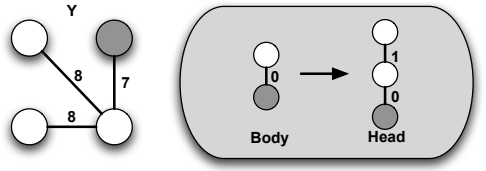
This definition yields a unique body for each head and therefore a unique confidence value for each head. This allows us to represent the rules by the head only. Note that the definition disallows disconnected graphs as body due to the lack of a support-definition for disconnected graphs. As a consequence *not all frequent patterns can be decomposed into graph evolution rules*.

Consider for instance pattern P_1 in Fig. 22: after removing all edges with the highest time-stamp, and discarding disconnected nodes, the graph that remains still contains two disconnected components (the one-edge component with label 1, and the one with label 0). Since the support is not defined for such disconnected pattern, P_1 can not be decomposed to be a *GER*. On the other hand, P_2 can be decomposed: in fact after removing all edges with the maximum time-stamp, and subsequently the disconnected node, we obtain a connected graph that will become the body of the rule for which P_2 is the head. Note that a *GER* can be represented



measure	head	body	confidence
minimum image support	1	1	1.00
number of occurrences	2	6	0.33

(a)



measure	head	body	confidence
minimum image support	1	1	1.00
number of occurrences	2	1	2.00

(b)

Figure 24: Two example host-graphs X and Y illustrating different problems with support and confidence notions.

in two different ways: either explicitly as two patterns (*body→head*), or implicitly by representing only the head as P_2 in Fig. 22. This is possible since there is a unique body for each head.

Finally, we have to choose a reasonable definition of *confidence* of a rule. Following the classic association rules framework, a first choice is to adopt the ratio of head and body supports as confidence. With the support being anti-monotonic this yields a confidence value which is guaranteed to be between zero and one. However, Fig. 24(a) shows that this definition may in some cases lack a reasonable semantic interpretation. In the upper host-graph X we find three possible ways to close a triangle given the edges from time-stamp 7. The confidence of 1 suggests that all of these will close to form triangles, while the graph shows that only one actually does. To overcome this counterintuitive result, we investigated if the ratio of number of occurrences of head and body can be employed to

solve this issue. While this definition of confidence allows for more reasonable semantics for the case in Fig. 24(a), it has the clear disadvantage that, due to the lack of anti-monotonicity, it may yield confidence values larger than 1, as in Fig. 24(b). In our experiments we compare the two alternative definitions showing that the minimum-image-based support is an effective and useful concept, while the occurrence-based definition has unpredictable behavior. Moreover, while the support is already available as it is computed for extracting the frequent patterns, the occurrence based confidence needs a separate and costly computation.

5.3 Mining graph evolution rules

GERM is an adaptation of the algorithm in (21), which was devised to prove the feasibility of the minimum image based support measure, and which in turn, was an adaptation of gSpan (99). Thus, *GERM* inherits the main characteristics from those algorithms. In particular, *GERM* is based on a DFS traversal of the search space, which leads to very low memory requirements. Indeed, in all the experiments that we performed the memory consumption was negligible.

Algorithm 7 $SubgraphMining(\mathbb{GS}, \mathbb{S}, s)$

1: **if** $s \neq min(s)$ **then** return *// using our canonical form*
2: $\mathbb{S} \leftarrow \mathbb{S} \cup s$
3: generate all s' potential children with one edge growth
4: Enumerate(s)
5: **for all** c, c is s' child **do**
6: *// using definition 16 based on definition 20 or definition 21*
7: **if** support(c)$\geq$ minSupp **then**
8: $s \leftarrow c$
9: $SubgraphMining(\mathbb{GS}, \mathbb{S}, s)$

We next describe in detail how to adapt gSpan to *GERM* whereas the main changes are in the *SubgraphMining* method shown as Algorithm 7. The first key point is that we mine patterns in large single graphs, while gSpan was developed to extract patterns from sets of graphs. The part that is most involved in adapting gSpan is the *support computation* in line

7. Thus we start from the implementation of (21), where gSpan support calculation is replaced by the minimum image based support computation, without the need for changing the core of the algorithm.

One of the key elements in gSpan is the use of the *minimum DFS code*, which is a canonical form introduced to avoid multiple generations of the same pattern.

We need to change this canonical form in order to enable *GERM* to mine patterns with relative time-stamps (cf. line 1). As explained after Definition 21, we only want one representative pattern per equivalence class; namely the one with the lowest time-stamp being zero. This is achieved by modifying the canonical form such that the first edge in the canonical form is always the one with the lowest time-stamp, as compared to gSpan where the highest label is used as a starting node for the canonical form. Any pattern grown from such a pattern by extending the canonical form will have the same lowest time-stamp, which we set to zero by a simple constraint on the first edge. Hence we guarantee to extract only one pattern per equivalence class which dramatically increases performance and eliminates redundancy in the output.

Note that when matching a pattern to the host-graph we implicitly fix a value of Δ, representing the time gap between the pattern and the host graph. In order to complete the match all remaining edges must adhere to this value of Δ. If all the edges match with the Δ set when matching the first edge, the pattern is discovered to match the host-graph with that value of Δ.

Another important issue is to be able to deal with large real-world graphs, in which several nodes have high degree (the degree distribution in our datasets follows a power law). In typical applications of frequent-subgraph mining in the transactional setting, such as biology and chemistry, the graphs are typically of small size and they are not high-degree nodes. Dealing with large graphs and high degrees give rise to increased computational complexity of the search. In particular, having nodes with large degree increases the possible combinations that have to be evaluated for each subgraph-isomorphism test. We thus equip *GERM* with a user-defined constraint specifying the maximum number of edges in a

pattern. This constraint allows to deal more efficiently with the DFS strategy by reducing the search space. Our experiments confirm that the total running time is much more influenced by the maximum-edge constraint than by the minimum support threshold.

5.4 Experimental Results

In this section, we report our experimental analysis. The *GERM* algorithm is implemented in C++. All the experiments are conducted on a Linux cluster equipped with 8 Intel Xeon processors at 1.8Ghz, 16Gb of RAM.

5.4.1 Datasets

We conduct experiments on four real-world datasets: two social networks (Flickr and Y!360) and two bibliographic networks (DBLP and arXiv). Table 6 reports statistics of the resulting graphs.

Flickr (`http://www.flickr.com/`): Flickr is a popular photo-sharing portal. We sample a set of Flickr users with edges representing mutual friendship and edge time-stamp the moment when the bidirectional contact is established. We generate one graph with monthly and one with weekly granularity.

Y!360 (`http://360.yahoo.com/`): Yahoo! 360° is an online service for blogging. Again we sample a set of users and proceed as in the Fickr dataset. In this case the monthly and weekly datasets contain exactly the same time period.

DBLP (`http://www.informatik.uni-trier.de/~ley/db/`): This dataset is based on a recent snapshot of the DBLP which has a yearly time-granularity. Authors are represented by vertices with a connecting edge if they are co-authors. The assigned time-stamp on an edge represents the year of the first co-authorship. Three different samples are extracted each containing the edges created in the corresponding years. These three samples allow us to analyze and compare long and short term trends.

arXiv(`http://arxiv.org/`): Similar to the DBLP dataset a co-authorship

							LCC			Growth Rates	
Dataset	**Date**	**|V|**	**|E|**	**avg deg**	**T**	**#CC**	**|V|**	**|E|**	**avg deg**	**total**	**avg**
flickr-month	03-05	147463	241391	3.27	24	16357	74792	182417	4.86	60347.8	2.83296
flickr-week	02-05	149863	246331	3.29	76	16661	76058	186504	4.90	246331	0.241055
y!360-month	04-05	177278	205412	2.32	10	17926	110627	155089	2.80	68470.7	5.15042
y!360-week	04-05	177278	205412	2.32	41	17926	110627	155089	2.80	68470.7	0.83434
arxiv92-01	92-01	70951	289226	8.15	10	6563	49008	260938	10.64	803.41	1.69114
dblp92-02	92-02	129073	277081	4.29	11	13444	83606	220098	5.26	25.52	0.408188
dblp03-05	03-05	109044	233961	4.29	3	14500	53370	153797	5.76	3.47	0.871401
dblp05-07	05-07	135116	290363	4.30	3	16333	72882	201468	5.52	3	0.749355

Table 6: Dataset statistics: Number of nodes and edges and resulting average degree for the total graph as well as for the largest connected component (LCC) out of all connected components (CC). Further the growth rate in terms of edges: total growth as ratio between the graph size at the final and the initial time-stamps, and average growth rate per time-stamp.

graph from a sample of the arXiv repository considering only physics publications is extracted. The obtained graph *arxiv92-01* contains the co-authorships which emerged in the years 1992 to 2001 with a time granularity of years.

As discussed in section 5.2, beside the time-stamp associated to each edge, our framework allows to have labels on both nodes and edges representing additional information. We experiment with node labels that are based on two graph-theoretic measures: the *degree* and the *closeness centrality*. These measures change as the graph evolves. To obtain static labels the measures are computed once on the whole graph, corresponding to the last time-stamp and then they are discretized in 5 bins.

Encoding additional information. As discussed in section 5.2, beside the time-stamp associated to each edge, our framework allows to have labels on both nodes and edges representing additional information. We experiment with node labels that are based on two graph-theoretic measures: the *degree* and the *closeness centrality*. These measures change as the graph evolves. To obtain static labels the measures are computed once on the whole graph, corresponding to the last time-stamp.

The *closeness centrality* (denoted cc) of a node v is defined as the average shortest path from v to all other nodes in the graph. Computing the closeness centrality for all nodes in a graph is an expensive task, so in order to scale our labeling process we resort to computing centralities approximately. Our approximation works by selecting a sample of random seed nodes, performing a breadth-first search from each of those seed nodes, and recording the distance of each node to the seed nodes. Since the seeds are selected uniformly at random and assuming that graph distances are bounded by a small number (which is true since real graphs typically have small diameter), we can use the Hoeffding inequality (45) to show that we can obtain an arbitrarily good approximation to centrality by sampling a constant number of seeds. Once the closeness centrality value for each node is computed, the nodes are divided in four classes: a *low centrality* class, containing nodes with a centrality value below the average centrality minus one standard deviation (labeled 3), a *medium centrality* class with values in the standard deviation of the average centrality

(labeled 2), a *high centrality* class with values above the average plus one standard deviation (labeled 1), and finally a class *miscellaneous* containing all nodes not sampled, or sampled too few times, possibly belonging to small connected components (labeled 0).

For the degree, nodes were discretized into 5 bins with ranges fixed in advance with the following mapping: Nodes v with $deg(v) = 1$ receive label 0, $deg(v) = 2 : 1$, $deg(v) \in [3, 9] : 2$, $deg(v) \in [10, 49] : 3$, and nodes with $deg(v) \geq 50$ are assigned label 5. Table 7 reports the distribution of the node labels.

Dataset	0 (%)	1 (%)	2 (%)	3 (%)	4 (%)
arxiv92-01 : deg	20.0	19.4	41.3	16.4	2.9
arxiv92-01 : cc	30.9	10.2	50.7	8.2	
dblp92-02 : deg	23.5	23.2	44.5	8.6	0.2
dblp92-02 : cc	35.2	9.1	47.4	8.3	
flickr-weeks : deg	72.2	8.0	13.7	5.4	0.7
flickr-weeks : cc	49.2	7.4	36.2	7.2	
y!360-weeks : deg	72.1	10.6	13.3	3.8	0.2
y!360-weeks : cc	37.6	8.4	46.5	7.5	

Table 7: Distribution of vertex labels

5.4.2 Results

We analyze the experimental results with regard to the following questions:

Q1 Do the extracted patterns and rules characterize the studied network?

Q2 Do different time granularities influence the confidence of the rules?

Q3 How do the different confidence definitions compare?

Q4 How do the parameters and the type of dataset influence the number of derivable rules, the number of patterns obtained, and the run time?

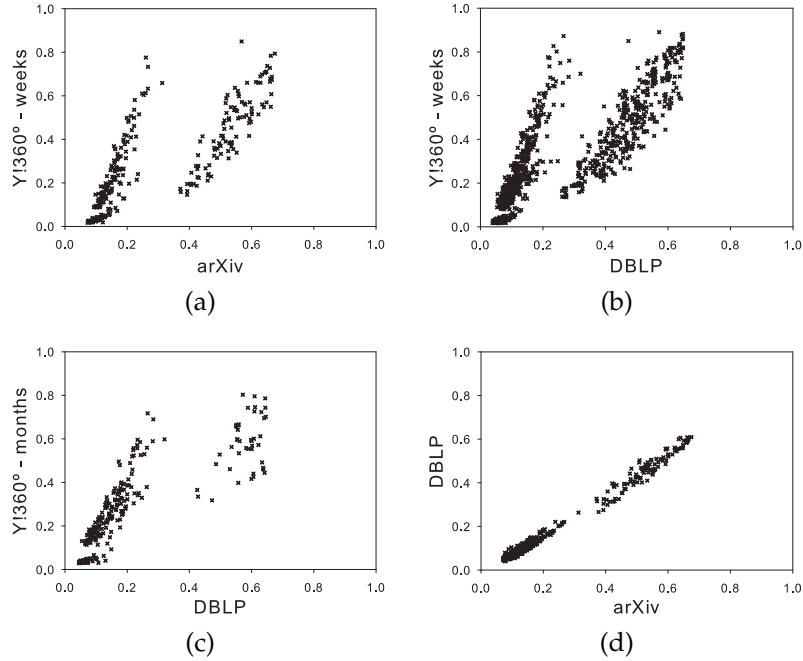


Figure 25: (a)–(d): comparison of confidence of graph evolution rules in the two bibliografic networks

Q1: Discriminative analysis. The first question is if the extracted patterns carry information that characterizes the analyzed network. In order to address this question graph evolution rules from the first six datasets in Table 6 were extracted with a minimum support threshold of 5000 for all but the "weeks" datasets where a minimum threshold of 3000 was used. Then we compared all pairs of datasets with respect to the rules confidences found in both datasets. We show the pair-wise comparison results in Fig. 25 and Fig. 26. The plots allow for several interesting observations. First, we see that the comparison between a co-authorship network (arXiv or DBLP) and a social network (Y!360) as in Fig. 25(a),(b) and (c) show different confidence values of the rules for each dataset (using Flickr instead of Y!360 gives the same results).

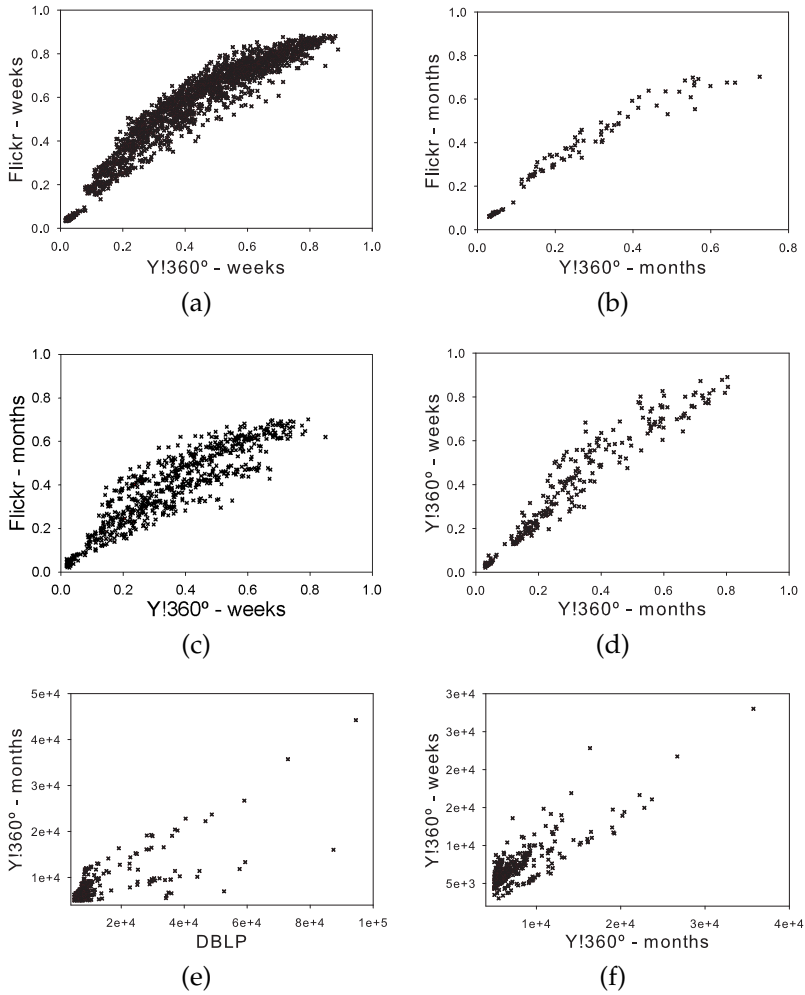


Figure 26: (a)–(d): comparison of confidence of graph evolution rules in the two social networks. (e),(f): comparison of support of patterns in different networks.

In contrast, the comparison of two co-authorship networks (arXiv and DBLP, in Fig. 25(d)) or two social networks (Flickr and Y!360, in Fig. 26(a) to (d)) reveals that all rules are in the proximity of the bisector, meaning

that each rule has very similar confidence values in the both datasets. This observation confirms our claim: *graph evolution rules indeed characterize the different types of networks*.

Fig. 26 (e) and (f) compare the same two datasets as in Fig. 25(c) and Fig. 26(d) respectively. However, in Fig. 25(e) and (f) we plot the rules according to their support instead of their confidence. Contrary to Fig. 25(c) and Fig. 26(d), both plots Fig. 26(e) and (f) show similar results, indicating that the support of a rule can not be used to characterize different types of networks.

Q2: Different granularity analysis. Fig. 27(a) similarly to Fig. 26(e), focuses on the difference of confidence from rules originating in the same network but with different time granularity. We observe that confidences for the weekly granularity are larger than the corresponding monthly confidences. The colors/shapes in the plot correspond to the difference between maximum time-stamp on an edge in the head (MTH) and maximum time-stamp on an edge in the body (MTB) of the rule. This figure very clearly reveals the cause for the specific structure of the plot. First, the difference between the maximum time-stamp in head and body indicated by the shape perfectly models the confidence differences between monthly and weekly granularity: the rules form three clear clusters (with the corresponding regression lines reported in the plot).

The second observation is that a larger difference between the time-stamps corresponds to a higher difference in confidence towards weekly granularity. This is quite natural if we think about confidence trough the lenses of prediction: the difference between the time-stamps in head and body can be thought as the temporal gap that must be bridged by a prediction task, and clearly predicting further in the future is more difficult (i.e., lower confidence). Hence clusters with higher time-difference have higher confidences in the weekly setting simply because three weeks are shorter than three months.

Finally, it is worth noting that only one rule in this plot has a difference of 4 between the maximum time-stamp in head and body: as expected it is in the left bottom corner, and is closer to the weekly axis than to the monthly axis.

Q3: Confidence and rules. Fig. 27(b) shows that the two confidence measures disagree. A more thorough investigation shows that all the rules with an occurrence-based confidence exceeding 200 have the most simple body: one single edge. Furthermore, all those rules span 3 or 4 time-steps from body to head. Given that they all share the same simplistic body, which can be matched anywhere, a correct prediction, especially 3 or 4 time-steps into the future is doomed to fail. The support-based confidence however, nicely assigns a confidence below 0.2 to all rules with the simplistic body, equivalent to declaring them almost meaningless, thus proving itself one more time fruitful being investigated and worthy be-

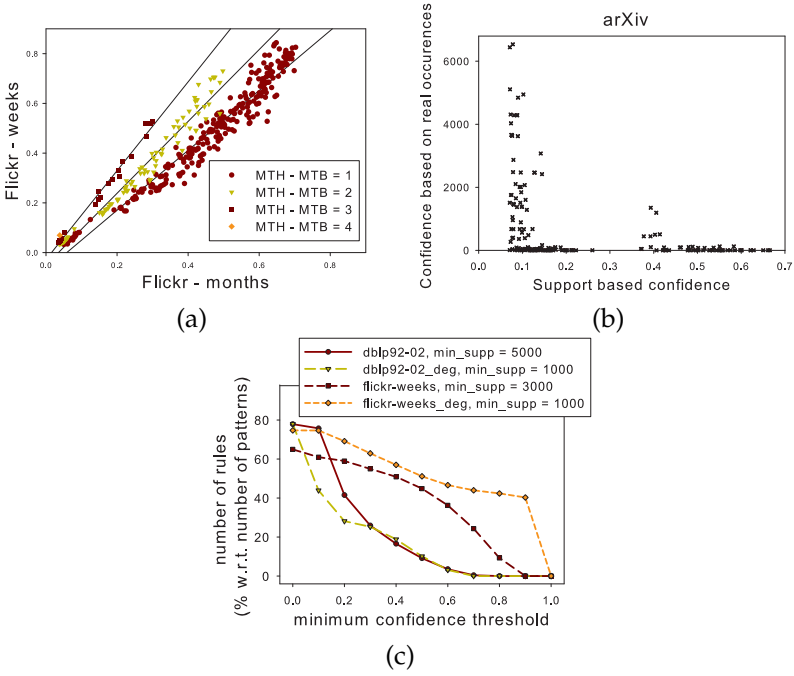


Figure 27: (a): confidence comparison between monthly and weekly granularity. (b): scatter plot comparing the two different definitions of confidence discussed in section 5.2.3. (c) number of valid rules as percentage of the number of frequent patterns, for varying confidence.

ing used.

Q4: Influence of parameters and dataset. Further important insights can be gained from an analysis of the number of rules and patterns extracted as well as the run-times. To understand how many of the extracted relative time patterns are decomposable and thus can be interpreted as rules we calculated the ratio of valid rules over all extracted patterns. Fig. 27(c) reports the number of valid graph evolution rules as percentage of the number of frequent patterns found, for various minimum confidence thresholds. This is done on one bibliographic and one social network; each with and without node labels. In all cases, the number of rules is close to 80% of the number of frequent patterns. Besides the fact that a lower minimum confidence thresholds yields more rules, the results nicely reaffirm the observation from Fig. 27(a). Indeed, rules extracted from a dataset with weekly granularity enjoy a much higher confidence than rules extracted from a dataset with yearly granularity.

As intended, the size of the result of the mining task depends on the maximum edge and the minimum support constraint. With a higher number of edges exponentially more graphs are possible, thus the exponentially increased number of extracted patterns for larger number of edges in Fig. 28(e) comes at no surprise. For lowering the minimal support Fig. 28(d) shows the typical result. Lowering the support threshold allows for more complex patterns which contain more edges and thus for the same reason as above the growth is exponential.

As Fig 28(a)-(b) show, the run time is affected much more by the maximum edge (max-edge) than the minimum support constraint (min-sup). While the increase is almost linear with decreasing minimum support, the run time grows exponentially with an increasing maximum edge size. Note the increase of two orders in magnitude in Fig. 28(b) from four to five and five to six edges.

A more interesting observation can be made from the Fig 28(c) and (f). The underlying graph structure is the same in both datasets with the only difference being the time-granularity of the edge time-stamps. The weekly graph with 41 edge labels is more *diverse* than the monthly graph with only 10. While the runtime between both datasets varies highly

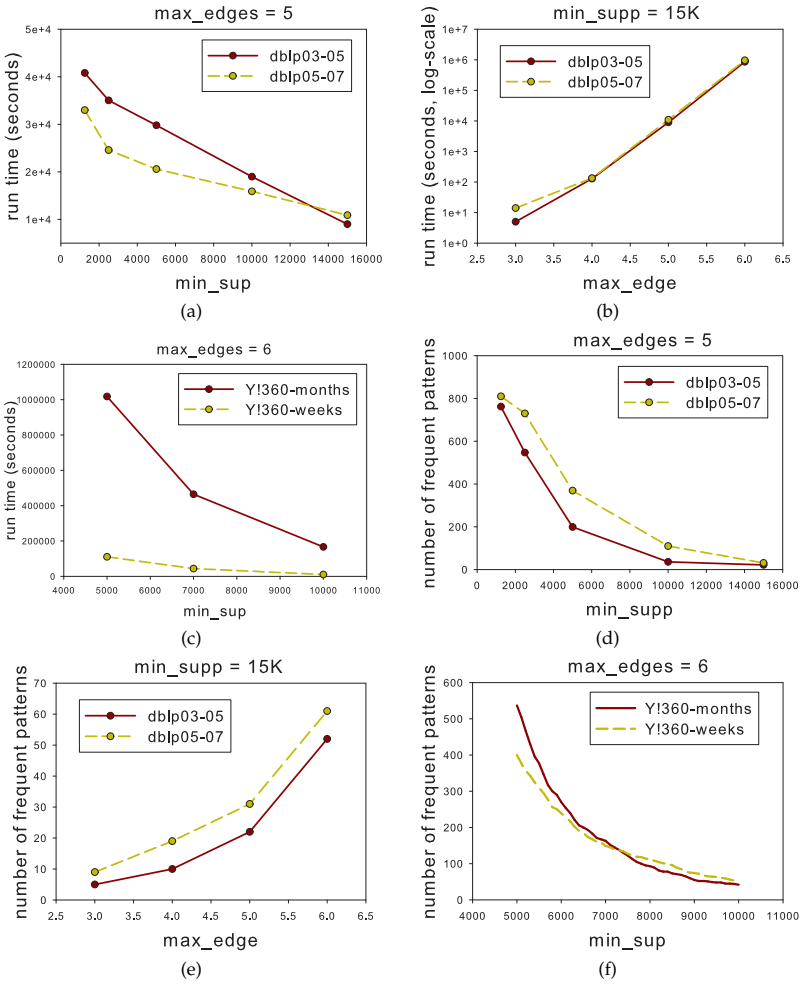


Figure 28: (a)–(f): run time and number of patterns found with varying *min. support* and *max. edge* thresholds.

with changing minimum support, the number of patterns extracted is almost the same for each minimum support. With regard to the number of patterns a higher label-diversity allows for more different patterns (i.e., more possible combinations for a fixed number of edges) *if* the sup-

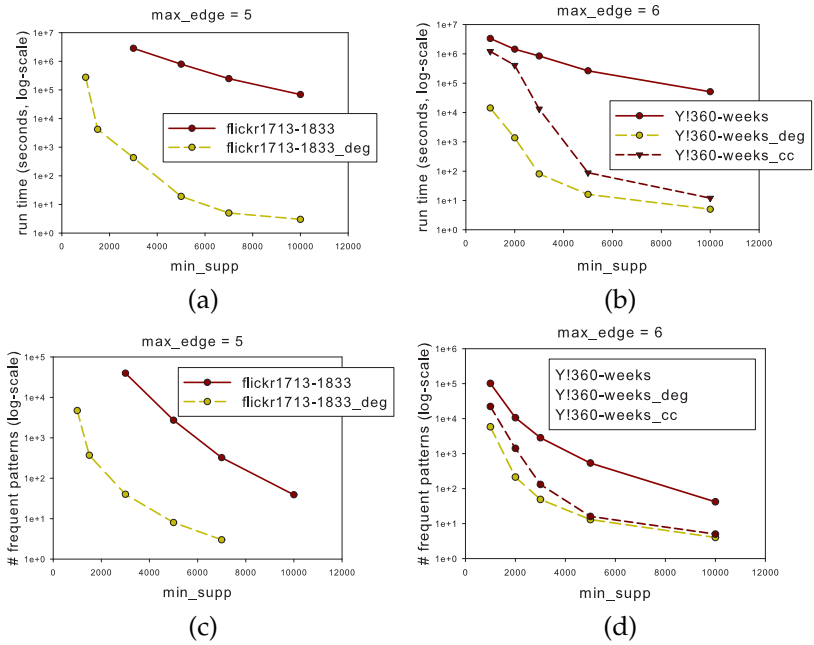


Figure 29: Run time and number of patterns found on networks with labelled nodes with varying min_sup.

port is low enough for these to be considered frequent. However, a lower label-diversity means that patterns can be found more repeatedly since the host-graph is more homogeneous, but the amount of different patterns is limited. Thus for a fixed number of edges there are more patterns with a high support in the more homogeneous graph and more patterns with low support in the graph with a higher label diversity as confirmed in Fig. 28(f). Regarding the varying run times between the datasets, in the more diversified data more patterns of smaller size can be found. The subgraph-isomorphism for these patterns is easier to calculate simply because they are smaller. Furthermore, it is easier to find a non-matching edge in the more diversified graph earlier, thus being able to terminate a search-branch for the subgraph-isomorphism check earlier. These two reasons explain the much lower runtime on more diverse graphs.

A similar reasoning holds for graphs with labelled nodes (Fig. 29). Also in this case the diversity introduced by the node labels reduced the number of patterns found and the run time.

	y!360 weeks				
min_supp	$e2$	$e3$	$e4$	$e5$	$e6$
1000	16	241	1478	7253	34904
1100	16	215	1179	5337	23828
1200	16	196	963	4075	16777
1300	16	173	791	3119	12028
1400	16	155	665	2435	8814
1500	15	141	550	1933	6584
1600	15	130	487	1581	5011
1700	15	115	413	1272	3870
1800	15	106	358	1051	3004
1900	14	95	308	884	2405
2000	14	87	269	717	1903
2500	12	59	153	337	708
3000	11	43	99	180	315
3500	9	34	64	103	149
4000	8	26	43	61	76
5000	6	18	26	29	29
6000	5	12	15	16	16
7000	5	11	12	12	12
10000	3	5	5	5	5

Table 8: Number of patterns of different size at various minimum support (ei denotes a pattern with $\leq i$ edges).

Finally, Table 8 shows the number of patterns of different size at various minimum support, for the y!360 weeks dataset.

5.5 Extensions

In this section we discuss briefly how to relax some of the assumptions of our problem definition.

Consider first the pattern H in Fig. 23. Imagine that in the data it is the case that when there is a star of size 3 an edge between two peripheral nodes appear. Pattern H captures partly this phenomenon, but is also too

"specific" as it emphasizes that the star was formed in particular time instances before the appearance of the last edge. A more general pattern would be to replace the time-stamp of the last edge with T, and the time-stamp of all the edges in the star with the constraint "$< T$", which will have to be satisfied when tested with the time-stamps of the host graph.

For sake of presentation, in section 5.2 we assumed that graphs can only grow in time. However, our approach can be easily extended to handle edge-deletions if an edge can appear and disappear at most once. The extension would consider two time-stamps t_I (time of insertion) and t_D (time of deletion) on each edge instead of the single time t. By modifying definitions 1 and 2 condition (ii) to $\forall (u, v) \in E_P$ it is $t_I(\varphi(u), \varphi(v)) = t_I(u, v) + \Delta$ and $t_D(\varphi(u), \varphi(v)) = t_D(u, v) + \Delta$.

We did no implement the above matching since two out of four datasets (arXiv and DBLP) are naturally only growing (thus, no deletions) and deletions are rare in the other two.

In our approach, we have not considered node or edge relabelling so far. Considering node and edge relabeling is very interesting, as in many graphs, such as social networks, the properties of nodes and edges change over time. For example, in social-network analysis it would be interesting to study the change of leadership in communities and its effects.

5.6 Discussion

Following a frequent pattern mining approach, we defined relative time patterns and introduced the problem of extracting *Graph Evolution Rules*, satisfying given constraints of minimum support and confidence, from an evolving input graph. While providing the problem definition we discussed alternative definitions of support and confidence, their merits and limits. We implemented *GERM* an effective solution to mine Graph Evolution Rules, and we extensively test it on four large real-world networks (i.e., two social networks, and two co-authorship networks from bibliographic data), using different time granularities. Our experiments confirmed the feasibility and the utility of our framework and allowed for interesting insights. In particular we showed that Graph Evolution Rules

with their associated concept of confidence, indeed characterize the different types of networks.

Besides all the above extensions, one possible straightforward application of our framework is to take advantage of the just defined rules and confidence: such a paradigm enables us to put the basis for defining a framework that will allow us to *predict* graph evolution, and that, together with *GERM*, will provide helpful tools for social-network analysis and other fields of research where dynamic graphs are a good data representation.

Availability. The executable code of the GERM software is freely available at: `http://www-kdd.isti.cnr.it/GERM`.

Chapter 6

From Local Patterns to Graphs

Like in chapter 4, here we want to analyze the setting of *action* in a network. However, some details are in contrast with the rest of the thesis: instead of looking for frequent local patterns in graph data, we start from them and we build the original graph. In this chapter, the general approach to graph mining followed so far is lost, as we start from *traces* in a dataset of process logs, and constraints are not used to drive the search to an efficient and focused direction, but they are part of the model and represent properties of the local patterns. Another important detail that we do not assume here is the network of users, neither implicit (as in chapter 4) nor implicit (as in chapter 5): in this chapter, we do have a set of users performing actions during time, but the network we want to build is among the different tasks that the users perform. In fact, the context is that of Workflow Mining.

6.1 On Workflow Mining

In the past few years, many organizations have started to use information systems to support the execution of their business processes (69). With the increasing number of these available systems, the volume of the available

collected processes logs is growing rapidly. These logs are very useful in several fields: in design and production processes, it could be important to detect the actual state of the process, how many items have been produced and in how much time; in logistics, the optimization of times is crucial; every step should be made strictly on time, and if there are anomalies or problems, the entire logistic solution should be redesigned. For such reasons, the interest in analysing process logs has been increasing rapidly in the last years (32; 83; 89). However, such logs are hard to analyse from different points of view because there is too much data, the original process diagram is too complex, and there are too many users to observe. Several techniques, such as workflow mining, have been proposed to automatically derive the workflow models originating from the process logs (42; 88; 96). Their focus is to derive the process model that was actually followed, and this can be different from the original one in several ways, e.g., certain tasks from the original process were never performed or were performed too many times, or the tasks performed were not in the original diagram. In addition, these techniques answer questions such as:

- Given the logged traces, what is the workflow network?
- Is the mined workflow network equivalent to the original design? (Delta Analysis)
- Is the mined workflow network better than the original design? (Performance Analysis)

However, current approaches mainly use the temporal information contained in the logs just for keeping track of the temporal order of the performed tasks.

Indeed, the temporal information associated with logs in the form of timestamps conceals knowledge that allows to distinguish among different temporal behaviors.

For example, suppose we have to execute tasks A, B and C and that the transition time from A to B is usually 1 minute, and from B to C it is 9 minutes. If we have a transition time of 9 minutes from A to B and

1 minute from B to C, we are in the presence of an anomaly during the process, even if the sequence of the performed tasks follows the process workflow. In this case, the usual workflow mining techniques do not detect the anomaly and therefore treat the abnormal execution as normal. In addition to anomaly detection, it could be useful to highlight situations in which some users are faster (or slower) than others in performing certain tasks, or situations in which some resources take too much time to be performed. In this sense, the model returned by the analysis process might be even richer than the original model, since temporal features of the tasks are often kept out of the design phase, or at least they are not explicitly specified in the model.

The contributions of this chapter can be summarized in 3 points:

- a mining method that extensively takes into consideration the time information, i.e., the extracted patterns representing a group of executions of a given process with similar execution times
- extracted patterns are summarized by taking into account the semantics of the possible executions, namely parallelism or mutual exclusion
- users can interact with the extracted and summarized patterns and explore alternative cases proposed by the system.

The first point is based on Temporally-Annotated Sequences ($\mathcal{TAS}$) mining, introduced in section 4.3.

In summary we propose a methodology for helping the domain expert in the analysis of process logs. This methodology aims at understanding which possible models might have generated such logs, and whether such models might also contain temporal constraints. The methodology can be broken down in 4 main pieces described later in section 6.2.

This framework has been applied to a real-world system: a manufacturing company. We collected the logs of 3 million transactions on 9 tasks for a total of about 1 million performed tasks processing the access to the design of various mechanic components to be put into production. This factory is located worldwide and therefore the tasks are executed

by different users at different locations. The results are encouraging, and indeed unexpected behaviours emerge.

The remainder of the chapter is organized as follows: section 6.2 is the core of the paper that presents the original contributions of our work. It describes the overall methodology: the formal definition of the *factorization* operators, the exploration graph $\mathcal{TAG}$, and the algorithm for the interactive workflow analysis. Section 6.3 presents a case study in which we applied the framework to a real dataset of process logs. Section 6.4 discusses the contributions and the results of this chapter.

6.2 A $\mathcal{TAS}$-based workflow mining approach

In this section we introduce a methodology for helping the domain expert in the analysis of process logs, aimed at understanding which possible models might have generated such logs, and whether such models might also contain temporal constraints. The methodology is composed of the following elements:

- a $\mathcal{TAS}$-based representation of the original log traces, that filters out noisy behaviours and detects temporal regularities. Such representation consists of a set of frequent $\mathcal{TAS}$;
- a set of operators for recognizing and *factorizing* two standard components of workflow models – i.e., parallelism and choice – from the $\mathcal{TAS}$, keeping trace of the temporal component;
- a graph summarization of a database of $\mathcal{TAS}$, to provide the user with an easy-to-grasp view of the data;
- an iterative and interactive procedure for exploring different and alternative *factorizations* of the same database of $\mathcal{TAS}$, potentially corresponding to different interpretations of the original input traces.

Performing these operations manually, by analyzing large quantities of information (such as 1 million of tasks performed as in our case study in section 6.3) is unfeasible and may not guarantee to discover what the domain expert or the workflow designer was looking for.

In the following, we start the discussion by summarizing the ultimate objective of this work, i.e., an interactive workflow analysis system. Then, for ease of presentation, we first describe the kind of data our analysis starts from (i.e., workflow traces) and define the above mentioned *factorization* operators over such data type. After that, the $\mathcal{TAS}$-based representation of the input data is briefly described, extending the factorization operators to the case of $\mathcal{TAS}$ and defining a graph summarization of sets of $\mathcal{TAS}$. Both the factorization operators and the graph representation will be the building blocks of the final analysis system, which is then described in more detail.

6.2.1 Problem setting: workflow analysis

One of the most important objectives in workflow analysis consists in reconstructing (part of) the workflow model that has generated a given dataset of process execution traces. This sort of *reverse engineering* operation is often very useful for comparing the model derived from the traces with the original model that generated them. This kind of comparison might highlight some design mistakes, useless or redundant parts of the model or, in general, a usage of the model that differs from the intentions of its designer (e.g., containing the systematic adoption of actions that were originally meant to be exceptional measures).

Reconstructing the model underlying a set of process traces usually requires to make some guesses about the scheduled order of operations in the model, or whether some sets of actions were executed in parallel (*parallelism*) or they were executed as mutually exclusive alternatives (*choice*). The method proposed in this work tries to perform such a reconstruction in a step-by-step fashion, selecting (with the aid of the user) and isolating at each stage a single relation between actions, and iterating the process till all significant relations were caught. The whole process can be sketched as follows:

1: Represent the input set of process traces through a set of frequent $\mathcal{TAS}$;
2: **while** user does not stop the execution **do**

3: Compute a graph-based summary of the actual set of $\mathcal{TAS}$;
4: Detect the potential cases of *parallelism* and *choice* between pairs of actions within the actual set of $\mathcal{TAS}$;
5: Ask the user to choose a single case of *parallelism* or *choice* to factorize, or to backtrack;
6: **if** backtrack
then Return to the set of $\mathcal{TAS}$ preceding the last factorization step;
else Factorize the chosen relation between two actions (*parallelism* or *choice*), and update the set of $\mathcal{TAS}$ accordingly;

As we can see, the approach requires the interaction with the user, for choosing, among the several possible alternatives available at each step, the *factorization* that looks more promising. Performing such choice automatically would require to have a function that correctly evaluates the quality or utility of any alternative (i.e., any case of *parallelism* or *choice*) and selects the best one. To the best of our knowledge, the state-of-art of the field is still far from defining any function of this kind having a sufficiently wide applicability, therefore our solution demands this heavily domain-dependent evaluation to the user. The interaction with the user is facilitated by means of a graphical, graph-based, summarization of the set of $\mathcal{TAS}$ at hand, which provides a complementary, more readable view of the same data, that can help in choosing the next most interesting factorization step to perform, among those listed by the system. These aspects are detailed in sections 6.2.6 and following ones.

6.2.2 The process workflow context

The digital traces collected during the re-iterated execution of a workflow process essentially have a sequential nature, and describe the ordered list of actions executed in each run, together with the agents who performed them and the date/time of execution:

Definition 23 (Workflow trace, Workflow log) *Let $\mathcal{A}$ be a finite set of actions and $\mathcal{U}$ a finite set of users. Then $\sigma = \langle (a_1, u_1, t_1)(a_2, u_2, t_2) \dots (a_n, u_n, t_n) \rangle$, where $a_i \in \mathcal{A}, u_i \in \mathcal{U}$ and t_i is a timestamp describing when the user u_i atomi-*

cally performed a_i, is a Workflow trace *or* Process instance. *A set $\mathcal{L}$ of workflow traces is a* Workflow log.

Therefore, a workflow log describes several runs (i.e., instances) of the same workflow process, each run being represented as a sequence of operations. An example of such a data can be found in Table 9 in section 6.2.8, where two workflow traces (identified by the column "Inst.ID") are represented, each containing 4 actions (tasks) performed by a unique user at different times.

Basic applications of workflow log analysis focus on the sequences of actions performed in each trace, therefore disregarding the user identity and the temporal information, and representing each trace essentially as a sequence of items. For instance, the sample workflow log in Table 9 could be reduced to a set of two sequences: $\{\ x \rightarrow a \rightarrow b \rightarrow c, \quad x \rightarrow b \rightarrow a \rightarrow c\ \}$.

6.2.3 Detecting parallelism and choice

As mentioned above, a typical workflow model can schedule the actions in several ways, including sequential execution (action a must be executed before b), parallel execution (a and b are launched together), and choice (only one between a and b is executed). A simple way to infer the presence of a parallelism or of a choice looking at a set of process instances, then, consists in locating possible evidences (or just clues) of such relations in the traces. On one hand, two actions invoked in parallel can appear in the traces in any order; on the other hand, two actions that form a choice can never appear one after the other. Following these basic ideas we define two relations between actions, that hold when the workflow traces suggest that a pair of actions might be executed in parallel or as a choice:

Definition 24 (Items relationships) *Let I be a set of items, and S be a set of sequences of items. Then, given $a, b, x \in I$, we define the relations $a \parallel_x b$ (read "a is parallel to b w.r.t. x") and $a \%_x b$ (read "a is in choice with b w.r.t. x") as follows:*

- $a \parallel_x b \quad \Leftrightarrow \quad \exists\, s, s' \in S$ *such that:*
  $(x \rightarrow a \rightarrow b \sqsubseteq s) \wedge (x \rightarrow b \rightarrow a \sqsubseteq s')$;

- $a \%_x b \quad \Leftrightarrow \quad \exists\, s, s' \in S$ *such that:*
  $(x \rightarrow a \sqsubseteq s) \wedge\ (x \rightarrow b \sqsubseteq s')$, *and*
  $\nexists\, s'' \in S : (a \rightarrow b \sqsubseteq s'') \vee (b \rightarrow a \sqsubseteq s'')$;

where $\sqsubseteq$ is the substring relation, i.e., $s \sqsubseteq s''$ iff all items of s appear in s' in the same order and in contiguous positions.

In the above definition, the relation between two items takes into consideration not only their relative positions in the input sequences, but also a limited form of *context*: both items (a and b) are preceded by a common item (x). This is a trade-off between more conventional relations defined in literature (e.g., (69)), mostly focused only on the items involved, and a more general approach that takes into consideration a larger number of items in the *past context* and a number of items also in the *future context*, i.e., situations like $x_1 \rightarrow \cdots \rightarrow x_N \rightarrow a \rightarrow b \rightarrow y_1 \cdots \rightarrow y_M$. In our case, essentially, we are considering $N = 1$ and $M = 0$.

Example 7 ($\|_x$) *If we have the sequences: $x \rightarrow a \rightarrow b,\ x \rightarrow b \rightarrow a$, then, according to Definition 24, we can write: $a \parallel_x b$. On the contrary, in the case of sequences: $x \rightarrow a \rightarrow b,\ y \rightarrow b \rightarrow a$ there is no parallelism under our definition, since each context (resp. x and y) leads to a distinct and coherent order of a and b. More standard definitions of parallelism do not consider the* provenance *of subsequences $a \rightarrow b$ and $b \rightarrow a$, therefore they are mixed up and interpreted as a unique evidence of a parallelism.*

Example 8 ($\%_x$) *If we have the sequences: $x \rightarrow a \rightarrow b,\ x \rightarrow b \rightarrow d$, then, according to Definition 24, we can write: $a\%_x b$. If we add the sequence $x \rightarrow b \rightarrow a$ to this example, $a\%_x b$ does not hold anymore, while now it holds $a \parallel_x b$.*

After defining which pairs of items/actions might potentially be in relation, we provide a function that lists all such relations, divided in parallelisms and choices:

Definition 25 (Parallelism detector) *We define an unary operator $\mathcal{P}(S)$ that associates to a set of sequences S the collection of relations of parallelism contained in S, i.e., $\mathcal{P}(S) = \{(x, a, b) \mid a \parallel_x b \text{ in } S\}$.*

Definition 26 (Choice detector) *We define an unary operator $\mathcal{C}(S)$ that associates to a set of sequences S the collection of relations of choice contained in S, i.e., $\mathcal{C}(S) = \{(x, a, b) \mid a\%_x b \text{ in } S\}$*

Example 9 (Detectors) *Given a set of sequences* $S = \{x \to a \to b \to c,\ x \to b \to a\}$*, the following holds:*

- $P(S) = \{(x, a, b)\}$
- $C(S) = \{(b, a, c)\}$

The approach proposed in this work consists in iteratively selecting one of the possible relations between items, and then *factorizing* it in the traces, i.e., locating the occurrences of such relation and replacing the items involved with a new element that represents the pair of items and the relation that connects them. That yields a new set of traces, where the selected relation between items has been isolated and emphasized.

Definition 27 (Factorize$_{\|}$) *Let* S *be a set of sequences. Given* $(x, a, b) \in \mathcal{P}(S)$*, we define the operator* $Factorize_{\|}((x, a, b), S) = S'$*, where every subsequence* $x \to a \to b$ *or* $x \to b \to a$ *of* $s \in S$ *is replaced with* $x \to a \parallel b$*, where* $a \parallel b$ *is a new item.*

Definition 28 (Factorize$_{\%}$) *Let* S *be a set of sequences. Given* $(x, a, b) \in \mathcal{C}(S)$*, we define the operator* $Factorize_{\%}((x, a, b), S) = S'$*, where every subsequence* $x \to a$ *or* $x \to b$ *of* $s \in S$ *is replaced with* $x \to a\%b$*, where* $a\%b$ *is a new item.*

On the new set of traces obtained by applying one of the factorization operators above, the same kind of analysis (detection of relations) and transformation (factorization) can be applied, iteratively.

Example 10 (Factorization) *Given* S*,* $P(S)$ *and* $C(S)$ *as in Example 9, we can apply the factorization operators in the following way:*

- $Factorize_{\|}((x, a, b), S) = S' =$
  $= \{\ x \to a \parallel b \to c\ ,\ x \to a \parallel b\ \}$
- $Factorize_{\%}((b, a, c), S) = S'' =$
  $= \{\ x \to a \to b \to a\%c\ ,\ x \to b \to a\%c\ \}$

6.2.4 A $\mathcal{TAS}$-based representation of traces

Applying the operators described above to the raw workflow traces has some drawbacks, mainly due to the possible presence or errors (missing actions, or actions registered by mistake) or very rare behaviours that we might want to exclude from the analysis.

Our approach provides that the analysis is carried out not on the original traces but on a set of $\mathcal{TAS}$ that represent the frequent behaviours (w.r.t. a given frequency threshold) and their temporal characteristics. That yields two results:

- first, errors and spurious behaviours are eliminated, since they are expected to appear with very low frequency, and therefore cannot appear in frequent patterns;

- second, the temporal information carried by the $\mathcal{TAS}$ can be used to better understand the behaviours appearing in the original traces, since different times in performing the same sequence of actions might reveal different usages of the same resources.

An example of $\mathcal{TAS}$ obtained from an input dataset of workflow traces is given in Table 10. Each $\mathcal{TAS}$ represents a sequence of actions (e.g., $x \rightarrow a$ in the first $\mathcal{TAS}$ listed) together with the set of typical transition times taken to move from one action to the next one (e.g., any time $t \in [18, 20]$, for the first $\mathcal{TAS}$).

The set of $\mathcal{TAS}$ used to represent the original traces can be selected following different criteria. Beside adopting different parameters and thresholds for the $\mathcal{TAS}$ mining phase, we could choose to use all the $\mathcal{TAS}$ extracted, or just the maximal ones, or those that satisfy other constraints, for instance temporal (e.g., take only patterns having duration longer than 5 minutes) or structural constraints (e.g., exclude patterns where the same action appears twice, thus evidencing the presence of a loop). For simplicity, in this paper we will adopt the first option, thus using all the $\mathcal{TAS}$ extracted. However, the whole analysis process can be equally applied with different selection criteria.

In the following we extend the operators described above in order to treat $\mathcal{TAS}$, instead of simple sequences.

6.2.5 Parallelism and choice over $\mathcal{TAS}$

All the definitions given for workflow traces do not take into account the temporal dimension contained in the data we work with. In order to add the time to our model, we redefine them for the case where the input sequences are a set of $\mathcal{TAS}$, as follows.

From now on, we assume to have a set of $\mathcal{TAS}$ T, each $\mathcal{TAS}$ being represented as a pair $t = (s, \alpha)$, where s is a sequence of items and α is a sequence of transition times. We also define as S_T the set of sequences that appear in T, without times, i.e., $S_T = \{s | (s, \alpha) \in T\}$. Then, definitions 24, 25 and 26 can be applied to S_T, essentially defining and locating parallelisms and choices only on the sequence component of our $\mathcal{TAS}$.

However, since when we solve a parallelism or choice instance we have to perform some operations to the temporal annotations on the corresponding sequences, we should redefine the factorization operators as follows.

Definition 29 (Factorize$_{\|}$) *Let T be a set of $\mathcal{TAS}$. Given $(x, a, b) \in \mathcal{P}(S_T)$, we define the operator*
$Factorize_{\|}((x, a, b), T) = T'$, where every temporally annotated substring $x \xrightarrow{\alpha_0} a \xrightarrow{\alpha_1} b$ of $(s, \alpha) \in T$ is replaced by $x \xrightarrow{\alpha_0} a \parallel b$, and every temporally annotated substring $x \xrightarrow{\alpha'_0} b \xrightarrow{\alpha'_1} a$ of $(s', \alpha') \in T$ is replaced by $x \xrightarrow{\alpha'_0} a \parallel b$, where $a \parallel b$ is a new item.

Definition 30 (Factorize$_{\%}$) *Let T be a set of $\mathcal{TAS}$. Given $(x, a, b) \in \mathcal{C}(S_T)$, we define the operator*
$Factorize_{\%}((x, a, b), T) = T'$, where every temporally annotated substring $x \xrightarrow{\alpha_0} a$ of $(s, \alpha) \in T$ is replaced by $x \xrightarrow{\alpha_0} a\%b$, and every temporally annotated substring $x \xrightarrow{\alpha'_0} b$ of $(s, \alpha) \in T$ is replaced by $x \xrightarrow{\alpha'_0} a\%b$, where $a\%b$ is a new item.

Example 11 (Factorization) *Given a set of frequent $\mathcal{TAS}$ $T = \{x \xrightarrow{[18,20]} a \xrightarrow{[3,4]} b \xrightarrow{[7,10]} c,\ x \xrightarrow{[19,22]} b \xrightarrow{[2,4]} a\}$, its corresponding set of sequences*

is $S_T = \{x \to a \to b \to c,\ x \to b \to a\}$*. Then, we can apply the factorization operators in the following way:*

- $Factorize_{\|}((x, a, b), T) = T' = \{x \xrightarrow{[18,20]} a \parallel b \xrightarrow{[7,10]} c,\ x \xrightarrow{[19,22]} a \parallel b\}$
- $Factorize_{\%}((b, a, c), T) = T'' = \{x \xrightarrow{[18,20]} a \xrightarrow{[3,4]} b \xrightarrow{[7,10]} a\%c,\ x \xrightarrow{[19,22]} b \xrightarrow{[2,4]} a\%c\}$

6.2.6 A graph summarization of $\mathcal{TAS}$

The set of $\mathcal{TAS}$ extracted from a database of workflow traces can be rather large, though usually much less than the original data. That makes it difficult for a human expert to obtain an overall picture of the sequences of tasks described by the data by simply sifting through them. For this reason, in this work we define a graph data structure that provides a complementary, lossy yet easy-to-read view of the set of $\mathcal{TAS}$ under analysis.

Definition 31 (Temporally-Annotated Graph)
Given a set T *of frequent* $\mathcal{TAS}$*, we define the* temporally-annotated graph ($\mathcal{TAG}$) for T *as a labeled graph* $G(T) = \langle V, E, l \rangle$*, whose nodes represent the actions appearing in* T*, the edges represent pairs of actions performed consecutively in at least one* $\mathcal{TAS}$ *of* T*, and the label of each edge is a set containing all the transition times that occurred in any* $\mathcal{TAS}$ *between the two corresponding consecutive actions. More formally:*

$$\begin{aligned} V = &\ \{a \mid a \sqsubseteq s, s \in S_T\} \\ E = &\ \{(a, b) \mid a \to b \sqsubseteq s, s \in S_T\} \\ l((a, b)) = &\ \{\alpha \mid a \xrightarrow{\alpha} b \sqsubseteq t, t \in T\} \end{aligned}$$

Figure 30 shows the Temporally-Annotated Graph corresponding to the starting set of $\mathcal{TAS}$ in Example 11. As we can see, all actions, all transitions between consecutive actions and all transition times contained in the $\mathcal{TAS}$ are depicted in the graph. On one hand, the graph loses some information, since all sequences longer than 2 in the $\mathcal{TAS}$ are virtually broken into pieces of length 2, not allowing to understand whether there is a loop in the starting sequences ($a \to b \to a$) or whether $b \to c$ is preceded by a

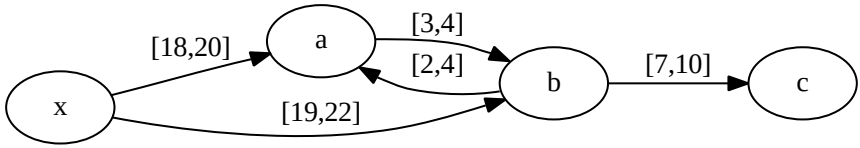


Figure 30: $\mathcal{TAG}$ for $\mathcal{TAS}$ T in Example 11

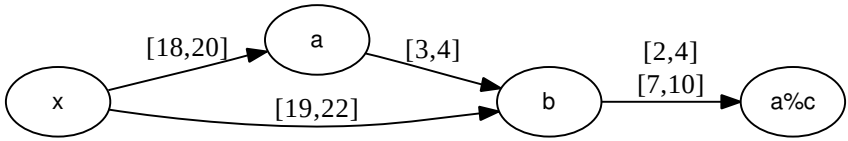


Figure 31: $\mathcal{TAG}$ after choice factorization in Examp. 11

in any sequence or, on the contrary, any sequence that passes through a terminates at b. On the other hand, the graph allows to understand at first sight some useful properties, for instance the fact that x plays the role of a source node, and c that of a terminal node, while between a and b there is not a strict order, which might be due to a loop or a case of parallelism. For comparison, in Figure 31 it is reported the $\mathcal{TAG}$ corresponding to the previous set of $\mathcal{TAS}$ after the factorization of a choice case. Notice that: (i) factorizing the choice case has as a side effect the disappearance of the parallelism located in the original set of $\mathcal{TAS}$, due to the fact that the two relations were in conflict and therefore the user must give priority to only one of them and disregard the other; (ii) the transition times for the rightmost edge ($b \rightarrow a\%c$) are obtained as union of those of $b \rightarrow a$ and $b \rightarrow c$, which is a direct effect of the way the labels of edges are defined in Definition 31.

Notice that our definitions of parallelism and choice involve a notion of *context*, that leads, in the case of parallelism, to check relations between actions in sequences of length 3 (which might become longer, if we extend the definitions to consider a longer context). That means that such relations cannot be clearly identified from the graph alone, and therefore

the factorization analysis must be performed directly on the $\mathcal{TAS}$, as done in section 6.2.5.

6.2.7 Interactive Workflow Analysis

The operators defined in the previous section allow to detect particular situations present in the dataset (parallelisms and choices), and to transform the latter in order to group the items involved.

We remark that the order of application of the operations is relevant, since after applying an operator the conditions for applying another operator could be not valid anymore (e.g., the result of a factorization for parallelism could destroy the subsequences that created a situation of choice), or simply the result could affect a different part of the dataset. In order to take into consideration all the possible sequences of operators applicable, we define a graph that represents the partially ordered set (*poset*) of all datasets that can be obtained from the original one (T), through a sequence of factorizations.

Definition 32 (Poset graph) *Given a dataset T of TAS, we represent the poset of transformations of T through a* poset graph $PG(T) = (V, E)$*, where:*

$$\begin{aligned} V = &\quad PC^* \uparrow^{\omega} (\{T\}) \\ E = &\quad \{(a,b) \in V \times V | b \in PC(\{a\})\} \end{aligned}$$

such that

$$\begin{aligned} PC(Ts) = &\quad \{Factorize_{\parallel}((x,a,b),T) | \\ &\qquad T \in Ts \wedge (x,a,b) \in \mathcal{P}(S_T)\} \\ \cup &\quad \{Factorize_{\%}((x,a,b),T) | \\ &\qquad T \in Ts \wedge (x,a,b) \in \mathcal{C}(S_T)\} \\ PC^*(Ts) = &\quad Ts \cup PC(Ts) \end{aligned}$$

i.e., V is the fix-point of operator PC^* applied to the original dataset, which yields the set of datasets obtained through a sequence of factorizations, and E connects each dataset with the dataset it was obtained from.

If the original dataset of $\mathcal{TAS}$ is complex and contains critical situations, such that items involved in several parallelisms or choices, loops, etc., the

set of transformed datasets can be very large. Therefore, it could be impractical for the end-user to simply fetch the whole graph of transformations. In Algorithm 6.2.7, we sketch an interactive procedure that extracts only a subset of the possible transformations, by asking the user which branch of the graph to explore, possibly backtracking to previous nodes of the graph: Figure 32 shows an example of a complete poset graph for

Algorithm 8 Interactive Poset Graph Navigation

Input: dataset of process logs L
Output: a set T of (factorized) $\mathcal{TAS}$

1: Extract the set $T = T_0$ of frequent $\mathcal{TAS}$ from L;
2: **while** execution not stopped by the user **do**
3: Compute the $\mathcal{TAG}$ on T and display it;
4: Compute the set $S = \mathcal{P}(S_T) \cup \mathcal{C}(S_T)$;
5: Present S to the user and ask him/her to select an operation op from $S \cup \{backtrack\}$;
6: **if** $op = backtrack \wedge T \neq T_0$ **then** $T = T' \; s.t. \; (T', T) \in E$, $PG(T_0) = (V, E)$;
7: **else** $T =$ factorization of T through op;
8: **return** T;

a small dataset. The topmost $\mathcal{TAG}$ represents the (graph representation of the) set of $\mathcal{TAS}$ exctracted from the input workflow log, as described in steps 1–3 of Algorithm 8. Then, each arrow represents a possible factorization operation for a given set of $\mathcal{TAS}$ (see step 4), and each time the user chooses one of such operations (step 5) the algorithm factorizes the actual set of $\mathcal{TAS}$ accordingly, and re-iterates the computation focusing on the resulting set of $\mathcal{TAS}$.

6.2.8 Run-through example

In this section we present a run-through example on a toy dataset of only 2 days of logs, where each day represents a transaction. For each transaction we have a sequence of performed tasks, together with their timestamps. Table 9 shows the data under investigation. On this data we apply

Inst.ID	Task	User	Timestamp
1	x	Administrator	Oct, 09, 1980, 12:00:00
1	a	Administrator	Oct, 09, 1980, 12:00:19
1	b	Administrator	Oct, 09, 1980, 12:00:29
1	c	Administrator	Oct, 09, 1980, 12:00:31
2	x	User1	Oct, 10, 1980, 17:10:12
2	b	User1	Oct, 10, 1980, 17:10:13
2	a	User1	Oct, 10, 1980, 17:10:51
2	c	User1	Oct, 10, 1980, 17:10:54

Table 9: Example of Process Logs

the $\mathcal{TAS}$ mining paradigm, in order to extract sequences that are executed frequently with typical transition times. Table 10 shows the $\mathcal{TAS}$ mined with minimum support $\sigma = 10\%$ and temporal tolerance $\tau = 1$.

$\mathcal{TAS}$ ID	$\mathcal{TAS}$	$\mathcal{TAS}$ ID	$\mathcal{TAS}$
1	$x \xrightarrow{[18,20]} a$	7	$x \xrightarrow{[18,20]} a \xrightarrow{[9,11]} b$
2	$x \xrightarrow{[0,2]} b$	8	$x \xrightarrow{[0,2]} b \xrightarrow{[37,39]} a$
3	$a \xrightarrow{[9,11]} b$	9	$a \xrightarrow{[9,11]} b \xrightarrow{[1,3]} c$
4	$a \xrightarrow{[2,4]} c$	10	$b \xrightarrow{[37,39]} a \xrightarrow{[2,4]} c$
5	$b \xrightarrow{[37,39]} a$	11	$x \xrightarrow{[18,20]} a \xrightarrow{[9,11]} b \xrightarrow{[1,3]} c$
6	$b \xrightarrow{[1,3]} c$	12	$x \xrightarrow{[0,2]} b \xrightarrow{[37,39]} a \xrightarrow{[2,4]} c$

Table 10: The corresponding mined $\mathcal{TAS}$

Figure 32 shows the poset graph of $\mathcal{TAG}$ that can be obtained starting from the $\mathcal{TAG}$ $G1$, which is the root of the graph. As we can see, we can have several possibilities at a certain level, for example after we generate graph $G2$. Each of them corresponds to having chosen to solve a particular parallelism or choice, first by enumerating all the possibilities by using one of the two *detector* operators defined in section 6.2, then by applying the corresponding factorization operator. Choosing which parallelism or choice to solve will correspond to choose a path of $\mathcal{TAG}$ along the graph. In this way we can navigate through all the possible actions that we can perform on the original mined workflow $\mathcal{TAG}$.

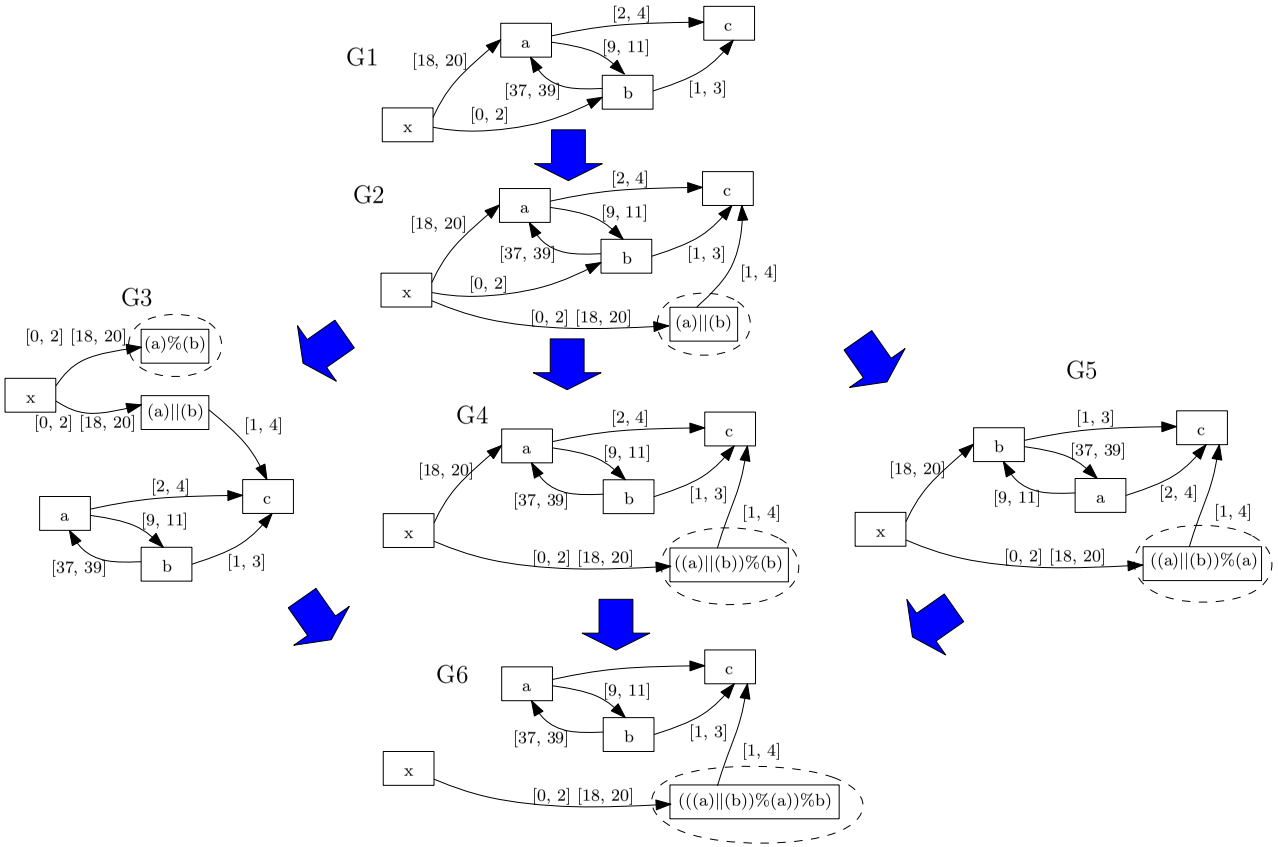


Figure 32: The poset of derived $\mathcal{TAG}s$ (dashed ellipses indicate the new items introduced by factorizations)

6.3 Case Study

In this section we present the work done as a case study on real-life data. The dataset comes from the usage of a real-world system developed by Think3(1), which is an object repository managing system, that allows the users to operate on the same objects from different locations. The timestamps contained on the logs represent the exact moment in which the event occurred. In particular, we did not have the starting and ending time of an operation, so we assumed that they are instantaneous and that the timestamps generally refer to the pair (execution time, transition time).

The dataset contains about 300000 transactions on 9 tasks, for a total of about 1 million of performed tasks. The logs span along 6 months of executions. For our analysis, we used a quite low support threshold of 0.5%, coupled with a τ of 1000 (seconds). Surprisingly, even these thresholds were enough to cut away two tasks from the frequently obtained annotated sequences. This proves that by manipulating the σ and τ parameters one can perform different grained analysis, even focusing on a frequently performed subprocess. Figure 33 shows a graph derived from the sequences of the original dataset of process logs in input, obtained with a procedure identical to the construction of $\mathcal{TAG}$, but without dealing with the temporal information.

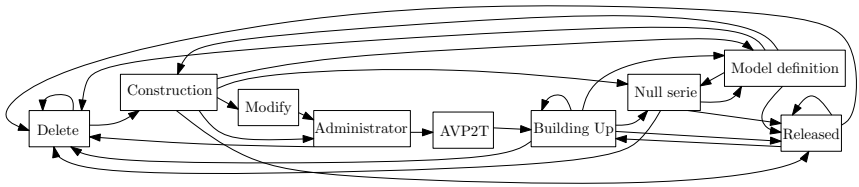


Figure 33: The graph derived from the original input data

Figure 34 shows the $\mathcal{TAG}$ resulting from the initial mining step, before looking for any dependency among the activities. As we can see, the σ and τ parameter played already an interesting role in this first step: sev-

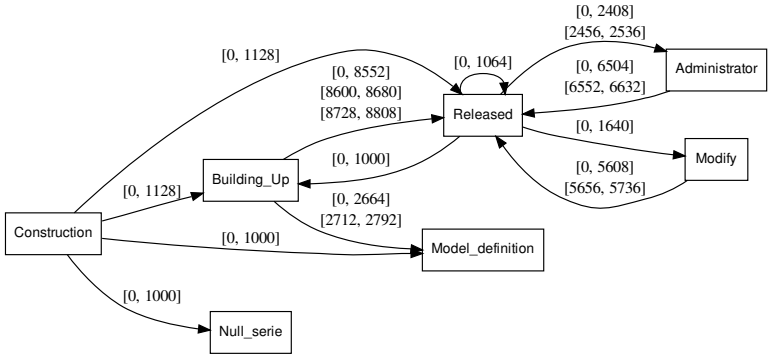


Figure 34: The initial mined $\mathcal{TAG}$

eral paths in the graph have disappeared, making 2 out of 9 tasks disappear as well. Of course, using a lower minimum support and/or a higher τ would have resulted in a more selective mining, making several other paths and tasks disappear from the graph.

We then followed the steps we have described on the previous section: after running the $\mathcal{TAS}$ mining software, we applied all the operators we have in our framework, looking for interesting dependency situations. After one step of analysis, we found one parallelism and several choices. We followed the parallelism, obtaining the $\mathcal{TAS}$ graphically depicted in Figure 35.

If we go one step forward, solving the choice between *(Administrator)* and *(Modify)*, we can note an interesting event: due to the particular handling of the temporal annotations and to the definition of the choice splitter, the annotations of the *(Administrator)* task became split between the choice node and what was left to the old *(Administrator)* node. Thanks to this particular feature, it was possible to detect frequent temporal behaviours that can be actually divided in two sub-behaviours. This situation is depicted in Figure 36. As we can see, this framework is particu-

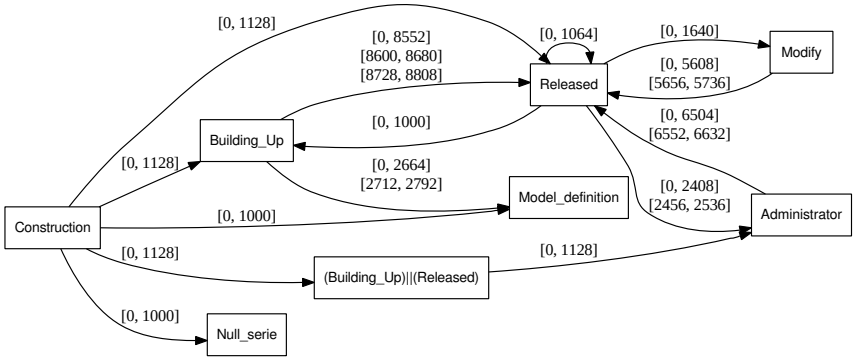


Figure 35: The $\mathcal{TAG}$ after one factorization step

larly suitable for any kind of temporal analysis of process logs. Thanks to the temporal annotations, it is easily possible to find bottlenecks on the process, unexpected behaviours, separate useless or redundant temporal information while performing business process analysis and so on. The $\mathcal{TAS}$ mining paradigm gives also the possibility, by a proper use of the minimum support parameter (σ), to look at the executed task with different granularity, looking for the most followed paths. The frame-

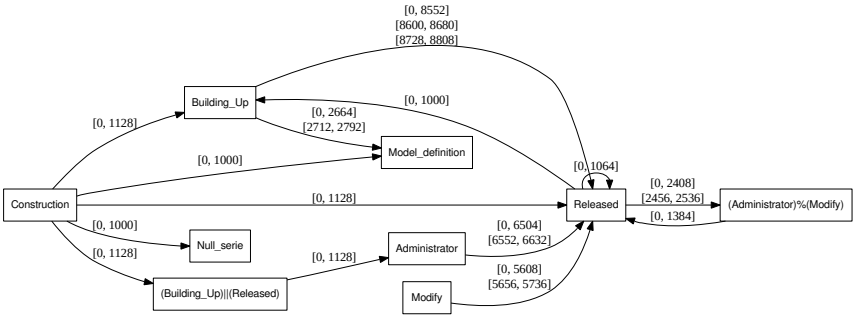


Figure 36: The $\mathcal{TAG}$ after two factorization steps

work hence results particularly suitable for performing Delta Analysis and Performance Analysis. Analysts, in fact, can take advantage of our methodology in two ways: by using iteratively and interactively the two operators described in the paper, they can detect situations of choice and parallelism performed by the users (either as their free choice or because it was an intrinsic requirement of the corresponding tasks) that were not designed, and discover a workflow diagram different from the designed one (Delta Analysis); or they can take advantage of the temporal information contained in the $\mathcal{TAG}$ to discover bottlenecks or to optimize the execution of (part of) the process, by looking at the expected (possibly designed) time needed to perform particular (sequences of) tasks (Performance Analysis).

6.4 Discussion

In this chapter we have introduced a novel framework for mining workflow graphs from process logs that enables the user to perform a temporal analysis by means of a $\mathcal{TAS}$-based mining paradigm. We have in fact shown that Graph Mining is not the only possible choice when trying to analyze graph data in the temporal dimension, and that meaningful graphs that express such a dimension can be built starting from sequence data by means of techniques that involve the analysis of this kind of data.

We have presented a methodology for helping the domain expert in the analysis of process logs, aimed at understanding which possible models might have generated such logs, and whether such models might also contain frequent temporal behaviours.

After a run-through example, we have presented a case study in which our model and framework have been used to perform visual temporal analysis on a real-life process log dataset. Based on our work, we have thus showed how the framework results suitable for performing Delta Analysis and Performance Analysis involving also the temporal dimension contained in the data. The results in these directions are encouraging, and indeed let emerge unexpected behaviours in our case study.

The presented technique opens the way for the develop of a com-

plete software for performing such an analysis, which will guide the user through an iterative and interactive navigation of the poset of the possible workflow diagrams that the data can support.

More sophisticated analysis is also possible, for example considering the possibility of extending the management of the transition times, in order to handle non-instantaneously executed tasks, which will enable an even more sophisticated temporal analysis of the data.

A possible research direction would be also to take the original designed workflow diagram as input, considering it during the mining step to better analyze the process logs.

Chapter 7

Conclusions

In this thesis we have studied the problem of mining the temporal dimension in graph and network data.

We have identified two main settings in which time can play a role in such data: *action* and *evolution*. We have seen how, in the former, users (or entities) of a network perform any kind of actions during time: they can diffuse information, they can influence other users, they can perform actions together with other users (like playing a football match), they can perform tasks independently from other users, and so on.

In the other setting, we have seen how graphs and networks can see a change in their structure over time: new nodes can appear, as well as users can quit a community, or they can densify the network by creating new connections with other users, and so on.

From this kind of data, which is the data we have analyzed in this thesis, we have seen how to extract useful knowledge about the temporal dimension. In order to do so, we have investigate possible techniques of analysis. We have shown how Graph Mining and TAS Mining can help in this direction.

In particular, we have made an extensive study of the Graph Mining literature, and we have studied its insights. We also presented a rich study of the constraints that can be pushed in the computation, and we have presented algorithms for pre-processing or mining graph data under

a conjunction of constraints, effectively extending the current literature.

We then have applied constraint-based Graph Mining to our main purpose, and we have showed how to extract useful information in real life datasets, both in the action and in the evolving settings. We have proven that the general Graph Mining approach can help in this direction, and that specific constraints can play an important role.

As applications of our approach, we have first shown a general methodology to mine the information propagation in a network where users exchange information. We have described how to extract useful information from such a network in order to be able to use a combination of two powerful techniques, namely TAS mining and Graph Mining, in order to find frequent patterns of propagation of information that involve also the possible causes of these propagation. We have shown how this combination can help in finding frequent temporal behaviors in the network together with the characteristics of the users, and what are the roles of these users in the network. We have presented results of a case study on real-life datasets and we have provided a possible interpretation of some of these results. While in these results the focus was on datasets of exchanged emails, one can easily see that this approach is easily applicable to any kind of network of users where flows of any kind of information are detectable. In particular, other possible uses of this approach include the study of influence propagation, spread of viruses, the identification of bottlenecks in a communication network, and so on.

Following a frequent pattern mining approach, we have also introduced the problem of extracting graph evolution rules satisfying given constraints of minimum support and confidence, from an evolving input graph. While providing the problem definition we have discussed alternative definitions of support and confidence, their merits and limits. We have provided an effective solution to mine Graph Evolution Rules, and we have shown extensive tests on four large real-world networks (i.e., two social networks, and two co-authorship networks from bibliographic data), using different time granularities. Our experiments confirmed the feasibility and the utility of our framework and allowed for interesting insights. In particular we have shown that Graph Evolution Rules with

their associated concept of confidence, indeed characterize the different types of networks.

At last, we have introduced a novel framework for mining workflow graphs from process logs that enables the user to perform a temporal analysis by means of a $\mathcal{TAS}$-based mining paradigm. We have in fact shown that Graph Mining is not the only possible choice when trying to analyze graph data in the temporal dimension, and that meaningful graphs that express such a dimension can be built starting from sequence data by means of techniques that involve the analysis of this kind of data.

We believe that the results of this thesis constitute a strong background knowledge for the analysts willing to study graph and network data where the time plays an important role. Thanks to our studies, we have proven that our methods are applicable to a wide range of problems, and that they are capable of extracting useful information about the explicit or implicit temporal behaviors detectable in a network.

References

[1] The think3 company. http://www.think3.com. 122

[2] Lada A. Adamic and Eytan Adar. How to search a social network, November 2004. 7, 8

[3] Li Zhang Lada A. Adamic Rajan M. Lukose Eytan Adar. Implicit structure and the dynamics of blogspace. *Communications of the ACM : CACMa publ. of the Association for Computing Machinery*, 47(12):35–39, 2004. 7, 8, 61

[4] C. C. Aggarwal and P. S. Yu. Online analysis of community evolution in data streams. In *Proceedings of SIAM International Data Mining Conference (SDM 2005)*, 2005. 12

[5] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs, 1998. 10

[6] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. volume 1377-469+ of *LNCS*. Springer, '98. 10, 11

[7] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In Peter Buneman and Sushil Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, pages 207–216. ACM Press, 1993. 26

[8] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 44–54, New York, NY, USA, 2006. ACM Press. 12

[9] Michele Berlingerio, Francesco Bonchi, Björn Bringmann, and Aristides Gionis. Mining graph evolution rules. In *ECML/PKDD (1)*, pages 115–130, 2009. 61

[10] Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, and Franco Turini. Mining clinical data with a temporal dimension: a case study. In *Proc. of The 1st Intern.Conf. on Bioinf. and Biomed.*, '07. 66, 70

[11] Michele Berlingerio, Francesco Bonchi, Fosca Giannotti, and Franco Turini. Time-annotated sequences for medical data mining. In *Proc. of The Intern. Workshop of Data Min. in Medicine*, 2007. 66, 70

[12] Michele Berlingerio, Michele Coscia, and Fosca Giannotti. Mining the temporal dimension of the information propagation. In *IDA*, pages 237–248, 2009. 7

[13] Michele Berlingerio, Fabio Pinelli, Mirco Nanni, and Fosca Giannotti. Temporal mining for interactive workflow data analysis. In *KDD*, pages 109–118, 2009. 10

[14] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAMiner: Optimized level-wise frequent pattern mining with monotone constraints. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, USA, 2003. 35

[15] F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. ExAnte: Anticipated data reduction in constrained pattern mining. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'03)*, Cavtat-Dubrovnik, Croatia, 2003. 35

[16] F. Bonchi and C. Lucchese. Pushing tougher constraints in frequent pattern mining. In *Proceedings of the 9th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*, Hanoi, Vietnam, 2005. 20

[17] Francesco Bonchi and Bart Goethals. FP-Bonsai: the art of growing and pruning small fp-trees. In *Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, Sydney, Australia, 2004. 35

[18] Christian Borgelt and Michael R. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the 2nd IEEE International Conference on Data Mining (ICDM'02)*, Maebashi City, Japan, 2002. 29

[19] Karsten M. Borgwardt, Hans-Peter Kriegel, and Peter Wackersreuther. Pattern mining in frequent dynamic subgraphs. *Data Mining, IEEE International Conference on*, 0:818–822, 2006. 13, 61

[20] J.F. Boulicaut and B. Jeudy. Using constraints during set mining: Should we prune or not. In *Proceedings of BDA'00, Blois, F, 2000. INRIA.*, 2000. 35

[21] Björn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *PAKDD*, pages 858–863, 2008. 54, 55, 56, 57, 72, 89, 90

[22] Cristian Bucila, Johannes Gehrke, Daniel Kifer, and Walker White. DualMiner: A dual-pruning algorithm for itemsets with constraints. In *Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'02)*, Edmonton, Alberta, Canada, 2002. 35

[23] T. Calders, J. Ramon, and D. Van Dyck. Anti-monotonic overlap-graph support measures. In *International Conference on Data Mining (ICDM)*. IEEE, 2008. 54

[24] Eddie Cheng, Jerrold W. Grossman, and Marc J. Lipman. Time-stamped graphs and their associated influence digraphs. *Discrete Appl. Math.*, 128(2-3):317–335, 2003. 7, 8

[25] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994. 21, 22, 56

[26] Diane J. Cook and Lawrence B. Holder. Graph-based data mining. *IEEE Intelligent Systems*, 15:32–41, 2000. 56

[27] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7:215–249, 1998. 10, 11

[28] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill Book Company, 2002. 40

[29] A. Datta. Automating the discovery of AS-IS business process models: probabilistic and algorithmic approaches. *Inf. Sys. Res.*, 9(3):275–301, '98. 10, 11

[30] Luc Dehaspe and Hannu Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999. 22, 24

[31] Prasanna Desikan and Jaideep Srivastava. Mining temporally changing web usage graphs. In *WebKDD*, pages 1–17, 2004. 12, 61

[32] P. Lawrence (ed). *Workflow Handbook 1997, Workflow Management Coalition.* J. Wiley and S., NY, 1997. 106

[33] Jure Ferlez, Christos Faloutsos, Jure Leskovec, Dunja Mladenic, and Marko Grobelnik. Monitoring network evolution using mdl. In *ICDE*, 2008. 12

[34] M. Fiedler and C. Borgelt. Subgraph support in a single graph. In *Proc. IEEE Int. Workshop on Mining Graphs and Complex Data (MGCS 2007 at ICDM 2007)*, 2007. 54, 55, 56

[35] Mathias Fiedler and Christian Borgelt. Subgraph support in a single large graph. *Data Mining Workshops, International Conference on*, 0:399–404, 2007. 54

[36] Shayan Ghazizadeh and Sudarshan S. Chawathe. Seus: Structure extraction using summaries. In *In Proc. of the 5th International Conference on Discovery Science*, pages 71–85. Springer, 2002. 56

[37] Fosca Giannotti, Mirco Nanni, and Dino Pedreschi. Efficient mining of temporally annotated sequences. In *Proc. of the 6th SIAM Intern. Conf. on Data Min.*, 2006. 66, 70, 72

[38] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, and Fabio Pinelli. Trajectory patter mining. In *The 30th KDD Int.Conf. on Knowl.Disc. and Data Min.*, '07. 66, 70

[39] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, and Fabio Pinelli. Mining sequences with temporal annotations. In *Proc. of the 2006 ACM Symp. on Applied Comp. (SAC)*, pages 593–597, 2006. 66, 67, 70, 72

[40] Amit Goyal, Francesco Bonchi, and Laks V. S. Lakshmanan. Discovering leaders from community actions. In *CIKM*, pages 499–508, 2008. 7, 10

[41] Gianluigi Greco, Antonella Guzzo, Giuseppe Manco, and Domenico Saccí. Mining unconnected patterns in workflows. *Inf. Syst.*, 32(5):685–712, 2007. 11

[42] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006. 106

[43] Joachim Herbst. A machine learning approach to workflow management. In *In Proceedings 11th European Conference on Machine Learning*, pages 183–194. Springer-Verlag, 2000. 10

[44] H.Kashima and A.Inokuchi. Kernels for graph classification. ICDM Workshop on Active Mining 2002, 2002. 25

[45] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. 93

[46] Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*, Melbourne, Florida, USA, 2003. 29

[47] Bernardo A. Huberman and Lada A. Adamic. Information dynamics in the networked world, Oct 2003. 61

[48] San-Yih Hwang, Chih-Ping Wei, and Wan-Shiou Yang. Discovery of temporal patterns from process instances. *Comput. Ind.*, 53(3):345–364, 2004. 11

[49] San-Yih Hwang and Wan-Shiou Yang. On the discovery of process models from their instances. *Decis. Support Syst.*, 34(1):41–57, 2002. 10, 11

[50] Akihiro Inokuchi and Takashi Washio. A fast method to mine frequent subsequences from graph sequence data. In *Proceedings of the 8th International Conference on Data Mining - ICDM08*, 2008. 12

[51] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'00)*, 2000. 22, 26

[52] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In Douglas H. Fisher, editor, *Proceedings of ICML-97, 14th International Conference on Machine Learning*, pages 143–151, Nashville, US, 1997. Morgan Kaufmann Publishers, San Francisco, US. 62, 71

[53] Istwan Jonyer, Lawrence B. Holder, and Diane J. Cook. Concept formation using graph grammars. In *Proceedings of the KDD Workshop on Multi-Relational Data Mining*, 2002. 23

[54] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146, New York, NY, USA, 2003. ACM. 61

[55] Steffen Kempe and Jochen Hipp. Mining sequences of temporal intervals. In *PKDD*, pages 569–576, 2006. 67

[56] Frank Klawonn. Finding informative rules in interval sequences. In *Intelligent Data Analysis*, pages 123–132. Springer, 2001. 67

[57] Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In *In ECML*, pages 217–226, 2004. 62, 71

[58] Risi Imre Kondor and John Lafferty. Diffusion kernels on graphs and other discrete structures. In *In Proceedings of the ICML*, pages 315–322, 2002. 25

[59] Gueorgi Kossinets, Jon Kleinberg, and Duncan Watts. The structure of information pathways in a social communication network, Jun 2008. 9, 61

[60] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 136–143, New York, NY, USA, 2001. ACM Press. 22

[61] Stefan Kramer, Luc De Raedt, and Christoph Helma. Molecular feature mining in hiv data. In *Proceedings of the 7th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'01)*, San Francisco, CA, USA, 2001. 35

[62] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the First IEEE International Conference on Data Mining (ICDM)*, pages 313–320. IEEE Press, 2001. 26, 45, 46

[63] Michihiro Kuramochi and George Karypis. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.*, 11(3):243–271, 2005. 54, 55, 56

[64] Jure Leskovec, Lars Backstrom, Ravi Kumar, and Andrew Tomkins. Microscopic evolution of social networks. In *KDD*, 2008. 12

[65] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *KDD*, 2005. 12

[66] Jurij Leskovec, Lada A. Adamic, and Bernardo A. Huberman. The dynamics of viral marketing, Sep 2005. 7, 9, 61

[67] David Liben-Nowell and Jon Kleinberg. Tracing information flow on a global scale using Internet chain-letter data. *Proceedings of the National Academy of Sciences*, 105(12):4633–4638, 2008. 10

[68] Zheng Liu, Jeffrey Xu Yu, Yiping Ke, Xuemin Lin, and Lei Chen. Spotting significant changing subgraphs in evolving graphs. In *Proceedings of the 8th International Conference on Data Mining - ICDM08*, 2008. 13

[69] Hongyan Ma. Process-aware information systems: Bridging people and software through process technology: Book reviews. *J. Am. Soc. Inf. Sci. Technol.*, 58(3):455–456, 2007. 105, 112

[70] T. Mitchell. *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997. 62, 71

[71] Fabian Moerchen. Algorithms for time series knowledge mining. In *Proc. of the 12th SIGKDD int.conf. on Knowl.disc. and data min.*, 2006. 67

[72] Raymond T. Ng, Laks V. S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the ACM International Conference on Management of Data (SIGMOD'98)*, New York, USA, 1998. 20

[73] Siegfried Nijssen and Joost N. Kok. Faster association rules for multiple relations. In Bernhard Nebel, editor, *IJCAI*, pages 891–896. Morgan Kaufmann, 2001. 22, 24

[74] Siegfried Nijssen and Joost N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'04)*, 2004. 30

[75] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. Discovering frequent arrangements of temporal intervals. In *ICDM*, 2005. 67

[76] Dhaval Patel, Wynne Hsu, and Mong Li Lee. Mining relationships among interval-based events for classification. In *Proc. of the 2008 int.conf. on Manag.of data*, pages 393–404, 2008. 67

[77] Jian Pei and Jiawei Han. Can we push more constraints into frequent pattern mining? In *Proceedings of the 6th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'00)*, Boston, MA, USA, 2000. 20

[78] Jian Pei, Jiawei Han, and Laks V. S. Lakshmanan. Mining frequent item sets with convertible constraints. In *17th IEEE International Conference on Data Engineering (ICDE'01)*, 2001. 20

[79] Matthew Richardson and Pedro Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–70, New York, NY, USA, 2002. ACM Press. 61

[80] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978. 23

[81] Guido Schimm. Process miner - a tool for mining process schemes from event-based data. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 525–528, London, UK, 2002. Springer-Verlag. 10

[82] Po shan Kam and Ada Wai chee Fu. Discovering temporal patterns for interval-based events. In *Proc. of the 2nd DaWaK*, pages 317–326. Springer, 2000. 67

[83] S.Jablonski and C.Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation*. Intern. Thomson Comp. Press, 1996. 106

[84] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S. Yu. Graphscope: parameter-free mining of large time-evolving graphs. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 687–696, New York, NY, USA, 2007. ACM. 12, 61

[85] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 374–383, New York, NY, USA, 2006. ACM Press. 12

[86] Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726, New York, NY, USA, 2007. ACM Press. 12, 61

[87] Joshua R. Tyler, Dennis M. Wilkinson, and Bernardo A. Huberman. Email as spectroscopy: Automated discovery of community structure within organizations. pages 81–96. Kluwer, 2003. 7

[88] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003. 106

[89] Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors. *Business Process Management, Models, Techniques, and Empirical Studies*, volume 1806 of *LNCS*. Springer, 2000. 106

[90] Wil M. P. van der Aalst and Kees M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002. 11

[91] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Which processes can be rediscovered? In *Eindhoven University of Technology*, pages 1–25, 2002. 10

[92] N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. *Data Mining, IEEE International Conference on*, 0:458, 2002. 56

[93] Chen Wang, Wei Wang, Jian Pei, Yongtai Zhu, and Baile Shi. Scalable mining of large disk-based graph databases. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD'04)*, 2004. 30, 33, 40, 43

[94] Chen Wang, Yongtai Zhu, Tianyi Wu, Wei Wang, and Baile Shi. Constraint-based graph mining in large database. In *Proceedings of the 7th Asia-Pacific Web Conference, (APWeb'05)*, Shanghai, China, 2005. 31

[95] A.J.M.M. Weijters and W.M.P van der Aalst. Process mining discovering workflow models from event-based data. In *Proceedings of the ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 283–290, 2001. 10

[96] T. Weijters and W. M. P. van der Aalst. Process mining: Discovering workflow models from event-based data., 2001. 106

[97] Edi Winarko and John F. Roddick. Discovering richer temporal association rules from interval-based data. In A. Min Tjoa and J. Trujillo, editors, *7th DaWaK*, volume 3589 of *LNCS*, pages 315–325. Springer, '05. 67

[98] Fang Wu, Bernardo A. Huberman, Lada A. Adamic, and Joshua R. Tyler. Information flow in social groups. *Physica A: Statistical and Theoretical Physics*, 337(1-2):327–335, June 2004. 7

[99] Xifeng Yan and Jiawei Han. gSpan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002. 27, 43, 45, 89

[100] Kenichi Yoshida, Hiroshi Motoda, and Nitin Indurkhya. Graph-based induction as a unified learning framework. *Appl. Intell.*, 4(3):297–316, 1994. 21, 22, 23

[101] Feida Zhu, Xifeng Yan, Jiawei Han, and Philip S. Yu. gprune: A constraint pushing framework for graph pattern mining. In *PAKDD*, pages 388–400, 2007. 31, 50