UNIVERSITY OF TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE

International Doctorate School
in Information and Communication Technologies

PhD Dissertation

# ON THE DISCOVERY OF RELEVANT STRUCTURES IN DYNAMIC AND HETEROGENEOUS DATA

Giulia PRETI

*Advisor:*
Prof. Yannis Velegrakis
University of Trento

ACADEMIC YEAR 2018-2019

# Abstract

We are witnessing an explosion of available data coming from a huge amount of sources and domains, which is leading to the creation of datasets larger and larger, as well as richer and richer. Understanding, processing, and extracting useful information from those datasets requires specialized algorithms that take into consideration both the dynamism and the heterogeneity of the data they contain. Although several pattern mining techniques have been proposed in the literature, most of them fall short in providing interesting structures when the data can be interpreted differently from user to user, when it can change from time to time, and when it has different representations. In this thesis, we propose novel approaches that go beyond the traditional pattern mining algorithms, and can effectively and efficiently discover relevant structures in dynamic and heterogeneous settings. In particular, we address the task of pattern mining in multi-weighted graphs, pattern mining in dynamic graphs, and pattern mining in heterogeneous temporal databases.

In pattern mining in multi-weighted graphs, we consider the problem of mining patterns for a new category of graphs called *multi-weighted graphs*. In these graphs, nodes and edges can carry multiple weights that represent, for example, the preferences of different users or applications, and that are used to assess the relevance of the patterns. We introduce a novel family of scoring functions that assign a score to each pattern based on both the weights of its appearances and their number, and that respect the anti-monotone property, pivotal for efficient implementations. We then propose a centralized and a distributed algorithm that solve the problem both exactly and approximately. The approximate solution has better scalability in terms of the number of edge weighting functions, while achieving good accuracy in the results found. An extensive experimental study shows the advantages and disadvantages of our strategies, and proves their effectiveness.

Then, in pattern mining in dynamic graphs, we focus on the particular task of discovering structures that are both well-connected and correlated over time, in graphs where nodes and edges can change over time. These structures represent edges that are topologically close and exhibit a similar behavior of appearance and disappearance in the snapshots of the graph. To this aim, we introduce two measures for computing the density of a subgraph whose edges change in time, and a measure to compute their correlation. The density measures are able to detect subgraphs that are silent in some periods of time but highly connected in the others, and thus they can detect events or anomalies happened in the network. The correlation measure can identify groups of edges that tend to co-appear together, as well as edges that are characterized by similar levels of activity. For both variants of density measure, we provide an effective solution that enumerates all the maximal subgraphs whose density and correlation exceed given minimum thresholds, but can also return a more compact subset of representative subgraphs that exhibit high levels of pairwise dissimilarity. Furthermore, we propose an approximate algorithm that scales well with the size of the network, while achieving a high accuracy. We evaluate our framework with an extensive set of experiments on both real and synthetic datasets, and compare its performance with the main competitor algorithm. The results confirm the correctness of the exact solution, the high accuracy of the approximate, and the superiority of our framework over the existing solutions. In addition, they demonstrate the scalability of the framework and its applicability to networks of different nature.

Finally, we address the problem of entity resolution in heterogeneous temporal databases, which are datasets that contain records that give different descriptions of the status of real-world entities at different periods of time, and thus are characterized by different sets of attributes that can change over time. Detecting records that refer to the same entity in such scenario requires a record similarity measure that takes into account the temporal information and that is aware of the absence of a common fixed schema between the records. However, existing record matching approaches either ignore the dynamism in the attribute values of the records, or assume that all the records share the same set of attributes throughout time. In this thesis, we propose a novel time-aware schema-agnostic similarity measure for temporal records to find pairs of matching records, and integrate it into an exact and an approximate algorithm. The exact algorithm can find all the maximal groups of pairwise similar records in the database. The approximate algorithm, on the other hand, can achieve higher scalability with the size of the dataset and the number of attributes, by relying on a technique called meta-blocking. This algorithm can find a good-quality approximation of the actual groups of similar records, by adopting an effective and efficient clustering algorithm.

## Keywords

[ Pattern Mining, Graph Mining, Weighted Pattern Mining, Dynamic Pattern Mining, Dense Pattern Mining, Entity Resolution, Multi-weighted Graphs, Heterogeneous Datasets, Dynamic Datasets ]

*To Whom It May Concern.*

# Contents

# Abbreviations

| | |
|---|---|
| ReSuM | **Re**levant **Su**bgraph **M**ining |
| multiW-SPM | multi-**W**eighted **S**core-based **P**attern **M**ining |
| DiCorDiS | **Di**verse **Cor**related **D**ense **S**ubgraphs |
| ExCoDE | **Ex**tract **Co**rrelated **D**ense **E**dges |
| HeTERe | **He**terogeneous **T**emporal **E**ntity **Re**solution |
| A-HeTERe | **A**pproximate **He**terogeneous **T**emporal **E**ntity **Re**solution |

# Chapter 1

# Introduction

With the advent of cheaper storage and the growth of the data collection capacity, many organizations, companies, and other autonomous sources in various domains have started collecting vast quantities of data every day. Similarly, the steady increase in social media usage has resulted in tons of new content posted on the Web daily, and in new social interactions between users. These phenomenons have led to the availability of huge, heterogeneous, and constantly changing datasets, which constitute the subject of analysis of data mining. Pattern mining has emerged in the data mining community as a way of extracting knowledge from these datasets, in the form of patterns satisfying desirable properties specified by the application.

In this thesis, we study three novel pattern mining tasks for datasets that display high levels of heterogeneity and dynamism, i.e., multi-weighted graphs, dynamic graphs, and heterogeneous temporal databases. We introduce each problem with a simple use case that serves as motivation for our work, and then discuss why existing approaches fall short in providing the desired results. Finally, we outline the specialized algorithms we designed to solve the three problems, which go beyond the traditional methods by incorporating user preferences, qualitative, and temporal information of the data into custom measures of relevance, for the first two problems, and of similarity, for the last problem.

## 1.1 Mining Multi-weighted Graphs

Graphs have become very popular recently, as they can model complex real-world relationships easily and represent data originating from a large variety of domains. In biology, for example, nodes can represent proteins and edges their interactions [PJZ05]; in sociology, nodes describe persons and edges their communications and relationships [Agg16]; and in market basket analysis, nodes model products and buyers, while edges connects products bought together, and users with the products they bought [VCR14].

Graph pattern mining finds applications in real-world scenarios such as fraud detection [NC03], event detection [ASKS12], community detection [CS10], identification of biological structures [HBW$^+$05, FNBB06], traffic control [JVB$^+$05], graph similarity search [JWYZ07], anticipation of user intention [PHMAZ00], query optimization [YYH04], and stock market analysis [DJD$^+$09], among others. In particular, it has proved its importance for both graph collections [YH02] and attributed [SMJZ12], probabilistic [LZG12], as well as generic, large graphs [EASK14].

A key primitive in such application domains has been finding structures that appear frequently in the graph, under the assumption that frequency signifies importance. In graph databases the frequency of a pattern has been effectively computed as the number of distinct graphs containing an appearance of the pattern. This definition satisfies the so-called *anti-monotone* property, which states that a pattern cannot be more frequent that any of its sub-patterns. This property is useful, because it enables the implementation of efficient frequent pattern mining algorithms [YSLD12]. In fact, by ensuring that the frequency of a pattern decreases monotonically as the pattern grows in size, it allows the mining process to start from small patterns and extend to larger ones only when the frequency of the pattern is above a certain frequency threshold.

Unfortunately, the same frequency measure cannot be used in single large graphs, as each pattern would have frequency either equal to 0 or to 1. On the other hand, if we measure it as the number of occurrences of the pattern in the graph, we would violate the anti-monotone property, because we could assign larger frequencies to larger patterns, due to the presence of overlapping occurrences [VSG06]. For this reason, alternative metrics have been considered in the literature [VSG06, FB07, BN08], with the more prevalent one being the MNI support, as it enjoys high effectiveness [EASK14].

Many real-world networks are naturally modeled through weighted graphs, which are graphs whose nodes and edges can be associated with a weight indicating, for example, their relevance or quality. In these graphs, the importance of a pattern can be better defined in terms of the weights of its appearances, in addition to its frequency, as there might be a discrepancy between patterns that are frequent in general, and patterns that are relevant according to the weights.

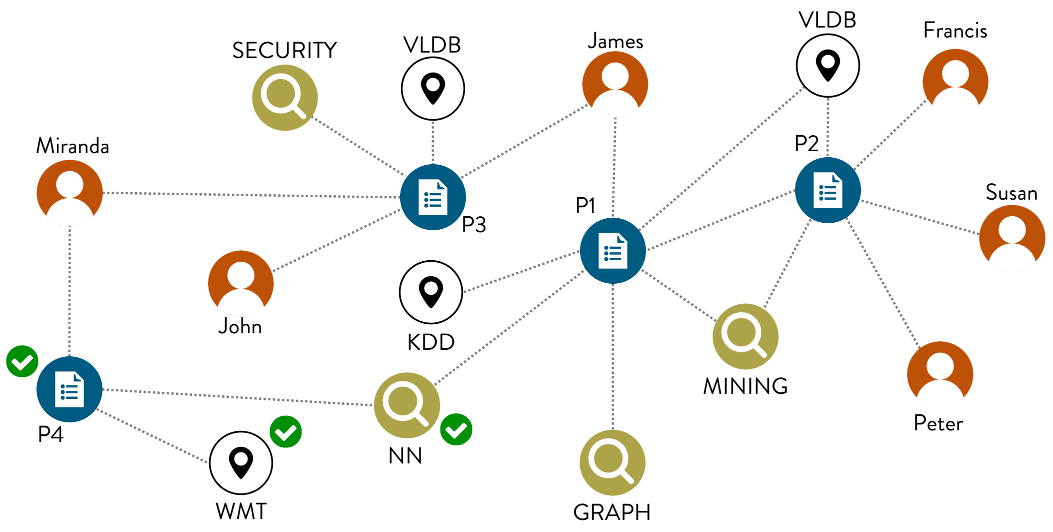


**Figure 1.1:** Portion of a citation network.

Examples include the discovery of metabolic pathways in genomic networks [KG00], where weights indicate strength between genomes [CDB$^+$09], the identification of topics of interest in large knowledge graphs [MLVP16], where weights quantify the degree a piece of data is qualified as an answer to a user [WA10], or the detection of common problematic cases in computer networks, where weights indicate congestion [BMS11].

Furthermore, many modern applications aim at offering personalized products and services to each individual user rather than proposing "one size fits all" solutions to everyone [SSTW01]. Indeed, such generic solutions would suit the user on average but would not be the best recommendations for her, as they are not tailored to her specific interests or preferences. These cases require multiple weights to model the diversity of user preferences, i.e., a multi-weighted graph, and a scoring function that assesses the relevance of a pattern for a user based on her own set of weights.

**Example 1.1.** *Consider a heterogeneous citation network whose nodes represent authors, papers, venues, and terms (keywords); and edges connect papers with their authors, the works they cite, the venue where they were presented, and with the keywords appearing in the title and in the keyword list. Figure 1.1 illustrates a portion of this network, where* Francis, James, John, Miranda, Peter, *and* Susan *indicate authors;* $P1$, $P2$, $P3$, *and* $P4$ *indicate papers;* KDD, VLDB, *and* WMT *indicate venues; and* GRAPH, MINING, NN, *and* SECURITY *indicate keywords. We can assign a weight to the nodes and edges of the network, according to specific user preferences that can be inferred, for example, from the papers the user published, the coauthors, or the keywords used and liked. Since from different users we can infer different preferences,* *each node and edge will be associated with multiple weights, one for each user. Let consider a researcher working in the field of machine translation, who wants to discover novel papers and emerging topics in this research area. From her past activities, we infer preferences for the Workshop on Statistical Machine Translation (WMT), Neural Networks (NN), and paper $P4$ (marked with a green check mark in the figure), and then we define edge weights accordingly.*

*Running a frequent pattern mining algorithm in this network will find patterns that mainly contain top venues and terms related to research fields with high engagement, because those labels appear very often in the graph, and thus are characterized by a larger support. In Figure 1.1, for example,* VLDB *is the most frequent venue because it appears twice, and therefore, a frequent pattern can be* $pattern_1 : [P_i] - [VLDB]$ *with* $P_i$ *indicating a* paper *node.*

*On the other hand, a relevant pattern mining algorithm can guide the user in the exploration of the literature most related to her own research interests, by suggesting the patterns that best fit her preferences. From this network, for example, the algorithm extracts the pattern* $pattern_2 : [WMT] - [P4] - [NN]$*, which is more relevant than* $pattern_1$ *to the field of study of this particular researcher.*

Other examples include on-line retailers like Amazon and social networks. In the first case, we can build large graphs of product co-purchases and exploit the patterns discovered to recommend future offers to the customers [SWD16]. Frequency, number of items, recency of the purchase, as well as the company's business intentions affect the importance of some co-purchases with respect to others [SSTW01]. In the second case, we can model the interactions between users and web content, as well as their activities recorded by the system [JWL$^+$11, BXL17], and then mine patterns of interactions [New04] that will help advertisers to target the desired audience. Since each advertiser has a specific business model, specific products to sell, and a specific target to reach, some patterns of interactions may be more important than others for her purposes. We thus need to create a different set of weights for each advertiser to encompass her needs.

All these examples highlight the need for a solution that is able to mine patterns based on their weights instead of limiting to their frequency, and that accounts for the individual preferences expressed as a multi-weighted graph, as opposed to common solutions where only a single set of weights, or none, is considered. In the literature, most works require the user herself to provide the structure of the desired patterns [PLM08, LHW10], and little effort has been dedicated to the problem of mining patterns that satisfy constraints on the weights [YSLD12]. Although these approaches can extract patterns characterized by large average edge weights and thus large relevance for a specific user, they cannot be applied efficiently to graphs with multiple weights on the edges. Indeed, the straightforward procedure to multi-weighted pattern mining would run these algorithms on each single-weighted graph separately, but would be clearly impractical when used for large graphs or large numbers of users.

Moreover, in contrast to frequent patterns, weighted patterns do not generally possess the apriori property because the weights of the extra edges (nodes) of a larger pattern may offset its lower frequency, and as a consequence existing works in weighted pattern mining proposed solutions that use pruning strategies less efficient than those developed for frequent pattern mining [YSLD12]. In fact, since this property ensures that the frequency of a pattern is always lower or equal to the frequency of its sub-patterns, the frequent pattern mining process can start from small patterns and extend to larger ones only when the frequency of the pattern is above a certain frequency threshold, hence pruning the search space considerably.

In contrast, we propose a novel approach to mine patterns in multi-weighted graphs that has performance comparable with the state-of-the-art in pattern mining in unweighted graphs. We achieve this by defining a novel family of scoring functions that are based on the MNI [BN08], a frequency measure widely used in the literature due to its characteristic of respecting the apriori property, while being efficient to compute [EASK14]. We model our solution as a constraint satisfaction problem (CSP), as proposed also for unweighted pattern mining [EASK14], and avoid redundant expensive computations by calculating the scores of each pattern at the moment we are visiting it, while keeping those that return a high score with respect to at least one set of weights. We present also a conservative approximate solution that reduces the number of weights to consider by aggregating those having a high probability to generate similar results into a single representative function, and show that this method introduces only few false positives, while running considerably faster than the exact approach. Finally, we propose a distributed version of our exact algorithm that runs on top of the distributed graph processing system Arabesque [TFS$^+$15] and is able to scale to large graphs.

We prove the effectiveness and efficiency of our approaches with an extensive set of experiments on both real and synthetic graphs, and discuss our findings with particular regard to the limitations of the exact solution when compared to the approximate solution, and to the settings where the distributed algorithm proves to be the best choice.

## 1.2 Mining Dynamic Graphs

Many real-world networks are dynamic by nature, meaning that their nodes and edges may appear or disappear over time. In social networks like Facebook, for instance, users make new friends and post new content; and similarly, in mobile networks users contact each others only at specific points of time. Introducing a temporal component in the graph, allows us to model the network more accurately, and moreover to discover interesting substructures that would not be uncovered otherwise. In Twitter, for example, a group of bloggers interested in Apple products interact more when a new iPhone is released and communicate with other friends otherwise. If we model this dynamic network as a static graph that contains all the interactions in the history of the network, the group may be hidden by the other external interactions.

In the context of dynamic networks, a lot of effort has been devoted to the problem of dense subgraph mining, with approaches devised either for single subgraph extraction or for subgraph enumeration. The former have the goal of finding the current densest subgraph [ELS15], the most lasting densest subgraph [SPTT16], or the heaviest subgraph [BMS11]; while the latter compute all the subgraphs that are dense in some time interval [YYW$^+$16, RTG17]. Finding dense subgraphs is important, for example, in event detection, as as people with the purpose of attending a common event gather at the same time and place, hence forming a dense group in an activity network.

In dynamic networks, nodes and edges experience structural and attribute changes, and in some cases, their series of changes can follow similar patterns or even be positive correlated. Detecting these correlations has many applications, especially when the nodes and edges involved are topologically close.

**Example 1.2.** *The Border Gateway Protocol (BGP) is the protocol used by the routers to establish how the packets are forwarded across the Internet. A challenge in managing the Internet is how to detect issues in the BGP routing topology and diagnose the human or natural disaster that caused each issue, hence allowing a faster recovery in the future, or even preventing them from happening again. We can model the BGP routing topology as a graph where nodes represent routers and edges represent routing paths. As these paths can change due to reconfigurations, bottlenecks, or faults, the graph changes over time. Figure 1.2 shows* 4 *snapshots of how a small portion of the BGP graph may look like.*

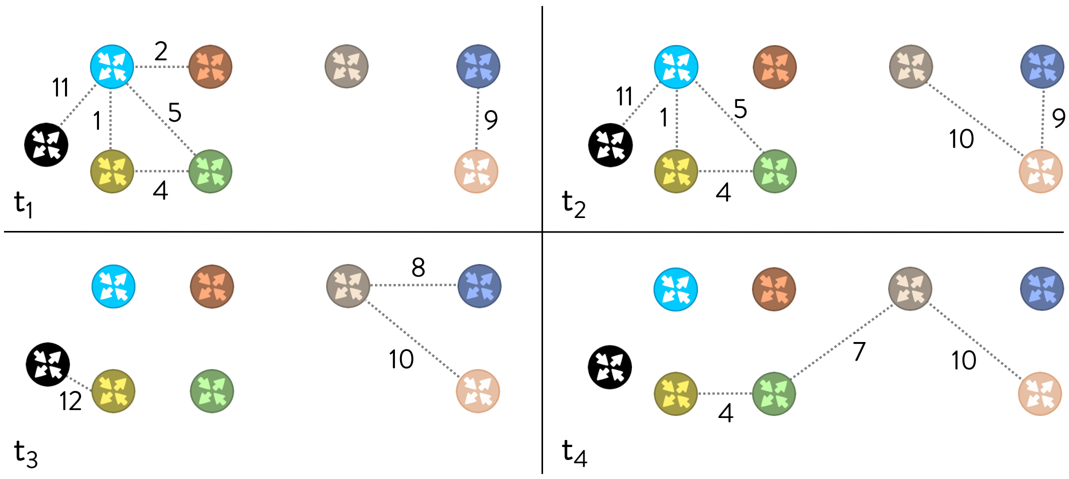


**Figure 1.2:** Snapshots of a dynamic network.

*A fault at some router can induce changes in other portions of the graph, due to the fact that all the paths that traverse the faulty router are affected and must be replaced to ensure the routing operations continue properly. As a consequence, changes in the same periods of time that involve a group of edges close in the graph are likely caused by a common cause. For instance, the snapshots in Figure 1.2 show that the routes* $1, 5, 9$ *and* $11$ *changed always at the same times, and thus they are correlated. Among them, routes* $1, 5, 11$ *are close together in the topology network, and thus, with higher probability, they were affected by the failure of the same router.*

*By focusing the attention on the whole dense group of temporally correlated routes, a network manager is able to isolate the root causes of the faults in the topology. However, in each snapshot of the entire BGP graph, there can be a significant number of elements experiencing a change that need to be analyzed by the manager. In addition, not every change is associated with an anomalous event. Thus, there is a need for an automatic tool that can simplify the detection of the issues by finding the regions in the graph whose edges present a similar pattern of appearance, so that the analyst need to focus on a small number of network elements.*

However, little research has addressed the task of finding subgraphs of correlated change in a dynamic network [CBL08, CBLH12], and to the best of our knowledge, no work performs a complete enumeration. In contrast, we consider the problem of enumerating all the dense groups of correlated edges in dynamic networks, and hence we propose two different measures to compute the density of a group of edges that change over time, and two measures to compute their temporal correlation. We propose an exact solution that extracts all the subgraph with high density and high correlation, as well as an approximate solution that scales better with the size of the network, while achieving high accuracy in the results found. In addition, as some types of dynamic networks may naturally contain a large number of dense groups of correlated edges, we study the problem of identifying a more compact subset of results that is representative of the whole set. Therefore, we introduce a threshold on the maximum pairwise Jaccard similarity between the edge groups returned and propose an approach that extracts a set of subgraphs with low pairwise overlap.

## 1.3 Mining Heterogeneous Temporal Databases

Entity resolution (ER) has been extensively studied in the literature [KSS06, EIV07], as it is a preliminary step for many Web applications [CES15] and social applications [BLGS06, ERSR$^+$15], as well as data mining tasks such as data cleaning, data integration, and Linked Data [MCD$^+$07, GM12, DS15]. The goal of ER is to determine whether two records in a database refer to the same real-world entity, and it typically achieved by comparing each pair of records [NH10] and retaining those pairs exhibiting high similarity in their attribute-value pairs. Given the inherently quadratic complexity of this process, ER becomes impractical for larger databases. Therefore, more recent works adopt blocking methods to split the records into blocks and then reduce the number of comparisons by considering only the pairs of records within the same block, though at the cost of missing some matches [Chr12]. These works generally select some distinctive attributes to create blocking keys that they use to produce cluster of the records displaying similar attribute values [MK06].

The openness of the Web and the success of projects like Linked Open Data have made available to the public a large amount of datasets collected from varied sources. However, due to the lack of standardization and hence the different choices of schema, these datasets are usually highly heterogeneous, meaning that they give a different description of the real-world entities they represent. For example, the record of a person in a hospital database contains the personal information, the contacts, and the medical data, whereas the record of the same person in the database of the ministry of employment includes the employment information, the curriculum vitae, as well as the personal information. As a consequence, ER algorithms must often deal with the challenge of reconciling records characterized by different sets of attributes [JHS$^+$10]. Many efforts have been devoted to schema-agnostic approaches that accommodate to noisy, loosely structured, heterogeneous, and growing collections of records [PIP$^+$12, PKPN14, PPPK16a,

| *database* | *id* | *attributes* | | *date* |
|---|---|---|---|---|
| **LinkedIn** | $r_1$ | **name:** Edward | **email:** edward@apple.com | 2018 |
| | | **birthday:** 21/02/1985 | **experience:** researcher | |
| | | **address:** Palo Alto, CA | | |
| | $r_1$ | **name:** Edward | **email:** ed@cs.stanford.edu | 2019 |
| | | **birthday:** 21/02/1985 | **website:** stanford.edu/edward | |
| | | **address:** Stanford, CA | **experience:** professor | |
| | $r_2$ | **name:** Elizabeth | **website:** elizabeth.me | 2019 |
| | | **address:** Los Angeles, CA | **experience:** consultant | |
| | | **email:** liz@accenture.com | | |
| **Twitter** | $r_3$ | **name:** Liz | **birth date:** 03/01/1973 | 2006 |
| | | **location:** Stanford, CA | **friends:** 20 | |
| | $r_3$ | **name:** Liza | **birth date:** 03/01/1973 | 2019 |
| | | **location:** Los Angeles, CA | **followers:** 122 | |
| **Facebook** | $r_4$ | **name:** Ed | **hometown:** Boston, MA | 2018 |
| | | **work:** researcher | **birth date:** 21/02 | |
| | | **college:** MIT, MA | **birth year:** 1985 | |
| | $r_4$ | **name:** Ed | **current city:** Stanford, CA | 2019 |
| | | **work:** Stanford, CA | **birth date:** 21/02 | |
| | | **college:** MIT, MA | **birth year:** 1985 | |
| | | **hometown:** Boston, MA | **relationship status:** married | |

**Table 1.1:** Records from three databases.

PPPK16b, EPP$^+$17]. These solutions typically rely on attribute-agnostic blocking mechanisms that group together those records having at least one attribute value in common, independently of the corresponding attribute name. Due to the high level of redundancy in the collection of blocks generated, these algorithms are generally less efficient than the schema-based approaches [PINF11].

Most of the existing approaches in ER work under the assumption that the data is static, and thus are inadequate for detecting records that describe the same real-world entity at different time instances [LDMS11]. In practice, many databases gather new data on a regular basis, and assign a timestamp to each record, indicating when the record was created or its validity period [S$^+$86]. In such cases, the identification of all the records, referred to as *temporal records*, that refer to the same real-world entity can be useful, for example, to trace the history of that entity and build a complete profile.

**Example 1.3.** *In an online recruitment system, such as Glassdoor*[1]*, companies post their job positions and users search and apply for new jobs. The system helps the users navigate the vast list of positions available by recommending the most relevant ones, and similarly, it helps*

[1]`https://www.glassdoor.com`

*the employers by suggesting potential employees. To achieve these goals, the system may need historical profiles of its users, which can be built by integrating temporal information collected from different data sources. Table 1.1 shows 7 temporal records extracted from 3 databases, i.e., LinkedIn, Twitter, and Facebook, where the attribute* date *indicates when the user identified by the id* id *updated his/her personal information on the corresponding Social Network. To build the profiles, the system must first detect which temporal records refer to the same real-world entity. This task is challenging mainly for two reasons.*

*The first reason is that each database uses a different schema, and therefore two records cannot be matched simply based on the similarity between pairs of attribute values. For example, record* $r_2$ *in* ***LinkedIn*** *has* $5$ *attributes, of which* address *indicates the current home address, while the records with* $id = r_3$ *in* ***Twitter*** *have* $4$ *attributes, of which* location *is the current city. Therefore, all these records contain the name of the city where the person lives, but this information is labeled differently in the two databases. When calculating the similarity between* $r_2$ *and* $r_3$*, the matching algorithm must be aware of these discrepancies. To further complicate matters, the same database may use different schemata in different periods of time. For example, in* $2006$ *the number of friends of an user in the Social Network* ***Twitter*** *was stored under the label* friends, *while in* $2019$ *under the label* followers.

*The second reason is that the attribute values of a record can change over time, and ignoring the temporal information may result in missing some matches. Let consider, for example, record* $r_1$ *with timestamp* $2019$ *in* ***LinkedIn****, and record* $r_4$ *with timestamp* $2018$ *in* ***Facebook****. A traditional entity matching algorithm does not match the two records, because they have somewhat similar names, but different addresses and jobs. However, since the timestamps of the two records are different, they might as well refer to the same real-world entity but at different points of time. Indeed, if we look at record* $r_4$ *with timestamp* $2019$ *in* FACEBOOK, *we can notice that it has the same address as the record* $r_1$ *with the same timestamp in* ***LinkedIn****. Considering the temporal information when calculating the similarity between records allows the matching algorithm to match* $r_1$ *and* $r_4$*.*

A handful of studies have been devoted to entity resolution in temporal databases [LDMS11, LLHT15, LLH17, CDN14a, CG13]. These works rely on a training dataset to learn a temporal model capturing the evolution of the attribute values of the entities over time, and then, when comparing two records, use this model to weight the similarity of each pair of attribute values. As a result, they all require both the same schema for all the records and the same schema through time. In addition, they can be affected by selection bias, because the collection of entities used to train the temporal model may not be representative of the whole domain of entities. For instance, a model trained with data collected from Western data sources may not be effective if used in Eastern databases, as the two populations have different traditions and evolutions.

In this thesis, we introduce the novel problem of entity resolution in heterogeneous temporal databases, discuss its main challenges, and present both an exact and an approximate solution to the problem. The exact solution compares each pair of records and then enumerates the maximal groups of pairwise similar records, while the approximate solution trades effectiveness for efficiency, by using a time-aware schema-agnostic meta-blocking algorithm. This algorithm proceeds in four steps. In the first step, it creates a collection of overlapping blocks of temporal records according to their attribute values, independently of the corresponding attribute names. In the second step, it builds a blocking graph where the nodes represent the records, the edges link records co-appearing in some block, and the edges weights are a function of the number of blocks the two records share and the number of times they share each block. This graph is used to transform the block collection into a new set of blocks that can be processed more efficiently. In the third step, the pairs of records within the same block are compared, and those with similarity greater than a minimum threshold are retained to create a similarity graph. In the last step, the algorithm extracts dense clusters of similar records from the similarity graph, each of which corresponds to a real-world entity.

## 1.4 Outline

The remainder of this thesis is organized as follows. In Chapter 2 we review the literature on the research fields most related to our work; Chapter 3 introduces the problem of mining relevant patterns in multi-weighted graphs; presents our exact, approximate, and distributed solution; and discusses their performance on real and synthetic graphs. Chapter 4 formalizes the task of finding dense groups of correlated edges in a dynamic network, and the related task of computing only a subset of diverse groups; shows our implementations and optimizations designed to increase the scalability of the mining process; and analyses the results of our experimental evaluation. In Chapter 5 we introduce the novel problem of entity resolution in heterogeneous temporal databases; discuss the inherent complexity of the task of matching records with different schemata and that can change over time; and illustrate the exact and approximate solutions we propose to address all the challenges of this problem. Finally, in Chapter 6 we discuss the limitations of our work, identify future directions for research, and draw our conclusions.

# Chapter 2

# State of the art

In this chapter we survey the major advances in the research areas related to our work, with the purpose of revealing the gaps in the literature that motivated our study and proving the significant contribution of this thesis to the field of data mining.

We first review how the problem of finding patterns in graph data has been tackled by existing approaches, which major extensions and novel formulations those approaches have proposed, and how our score-based pattern mining approach differs from them (Section 2.1). Then, we examine the literature of dense subgraph mining, focusing on the subfields of dynamic dense subgraph mining and anomaly detection in dynamic graphs, as they are the most related to the novel research topic we investigate in this thesis. We provide an overview of the existing works and illustrate why they are not suitable to address the task of dense correlated subgraph mining (Section 2.2). Finally, we move to the task of entity resolution and present the challenges of solving the problem in heterogeneous databases and temporal databases. We explain how existing techniques have tackled these challenges, and show the main differences with our approach to solve entity resolution in heterogeneous temporal databases (Section 2.3).

## 2.1 Graph Pattern Mining

The frequent pattern mining problem was originally studied in the context of market basket data analysis [AIS93], with the goal of finding sets of products bought together and whose frequency is higher than a frequency threshold. The advantage of this model is that it satisfies the downward closure property, which allows the design of algorithms that search for frequent patterns in a bottom-up fashion. Such property ensures that if an itemset is frequent, then its subsets are frequent as well, and thus the search can start from sets of single items and keep enlarging them until the are no longer frequent.

Subsequently, frequent pattern mining has attracted a lot of attention also in the context of graph data, as it finds applications in numerous domains such as detection of frauds and anomalies [NC03], characterization of protein structures [HBW+05, HYH+05], identification of web access patterns [PHMAZ00], fast-query processing [YYH04], and detection of traffic bottlenecks [JVB+05]. In this context, frequent patterns are subgraphs that appear often in the graph, and their discovery is an intrinsically intractable process because determining if two subgraphs represent the same pattern is a **NP**-hard problem known as *subgraph isomorphism problem* [Coo71, GJ79]. The frequency of a pattern is defined based on the type of graph data in input. When the input is a graph database, i.e., a collection of small graphs, it is the number of graphs in the collection containing the pattern [KK01], whereas if the input is a single graph, it is the number of appearances (a.k.a. *embeddings*) of the subgraph in the graph [KK05]. Because of this design, algorithms developed for graph databases fail in finding the correct results if used for single graphs, as every pattern would have frequency equal to $1$ or $0$.

### 2.1.1 Graph Databases

Similarly to the market basket data scenario, in the graph database setting the frequency measure satisfies the downward closure property (a.k.a. *apriori* property), meaning that any extension of an infrequent pattern is infrequent as well [AIS93]. Given the intractability of the graph pattern mining problem, this property has been extensively exploited in designing pattern-growth algorithms able to prune considerably the search space and thus scale better to larger graphs [YH03, NK04, HWPY04, PN07]. Rather than examining each possible combination of edges, these algorithms recursively expand frequent structures in larger subgraphs and stop the expansion when the subgraph generated is not frequent. Despite the massive pruning, they provide a complete and correct solution.

These algorithms can be divided into two categories, i.e., *apriori-based* and *pattern-growth*. The apriori-based approaches generate patterns of size $k$ by joining frequent patterns of size $k-1$. The join can either produce a pattern with an additional node [IWM00], or a pattern with an additional edge [KK01]. However, these approaches are impractical for mining long patterns, because when $k$ becomes larger, the joining procedure becomes expensive and generates a huge candidate set. On the other hand, the pattern-growth methods add an extra edge to frequent patterns of size $k$ to generate candidates of size $k + 1$. This technique avoids costly joining operations, but may generate the same candidate multiple times because the same set of edges is used to create and expand the candidates. To prevent the redundant examination of the same candidate pattern, *gSpan* [YH02] expands the patterns by adding the extra edge on their right-most path and performs the search in a depth-first manner. In addition, it uses a novel canonical labeling to accomplish the isomorphism tests, having proved that two subgraphs are isomorphic if they have the same canonical label. The canonical label of a pattern, called DFS code, is created by concatenating the ids of its edges in the order that gives the minimum lexicographic string.

In frequent pattern mining, the size of the result depends heavily on the choice of frequency threshold, meaning that at low values the number of patterns returned can be very large. As a consequence, many efforts have been devoted to develop algorithms that computed a reduced but representative set of patterns of great interest. *SPIN* [HWPY04] extracts the maximal frequent patterns from a graph database by computing the maximal frequent spanning tree and using them to build the maximal frequent subgraphs. Maximal frequent patterns are patterns that are not contained in any other frequent pattern, and therefore they constitute a lossless representation of the set of frequent patterns. Similarly, *CloseGraph* [YH03] extracts the set of closed frequent patterns, defined as the frequent patterns that are not contained in any larger pattern with the same frequency. *ORIGAMI* [AHCS$^+$07] introduces a novel concept of representative pattern called *$\alpha$-orthogonal pattern* and proposes an approximate algorithm to mine the set of $\alpha$-orthogonal $\beta$-representative patterns. A set of patterns is $\alpha$-orthogonal if the pairwise similarities between its elements are lower than $\alpha$, and it is $\beta$-representative if each pattern not in the set has similarity greater than $\beta$ with at least one pattern in the set. This set is constructed by randomly traversing the search space to find all the maximal patterns, and then retaining the subset of $\alpha$-orthogonal patterns that minimizes the set of unrepresented patterns, i.e., patterns having similarity lower than $\beta$ with any pattern in the subset retained.

On the other hand, *LEAP* [YCHY08] and *GraphSig* [RS09] focus on the discovery of the patterns with the highest statistical significance. LEAP adopts structural leap search to prune the search space horizontally and examine only a subset of promising subgraphs, and frequency descending mining to discover the significant patterns earlier in the search. GraphSig, instead, creates a set of feature vectors for each graph in the database, and use them to compute the significance of the patterns efficiently.

Despite closed and maximal frequent patterns are orders of magnitude fewer than the frequent patterns, the output of the mining algorithm can still be huge and contain redundant information. Moreover, the lack of user-defined thresholds leads to the discovery of many uninteresting patterns. Aiming at mining patterns that meet the user's demands, constraint-based algorithms have been proposed, which let the user specify the requirements that a pattern must meet in order to be part of the solution. *CabGin* [WZW$^+$05] identifies five types of constraints: element, size, super-graph, distance, and aggregate. The element constraints specify which nodes and edges should or should not be part of the patterns mined, the size constraints limit the size of the patterns, the super-graph constraints specify a group of valid subgraphs, the distance constraints regulate the length of the path distances in the pattern, and the aggregate constraints control the value of some aggregate of the nodes or edges in the pattern, where the aggregate function can be sum, avg, max, or min. These constraints are grouped into three categories, i.e., monotonic, anti-monotonic, and succinct, each of which is pushed at a different level of the mining process to speed up the extraction of the valid patterns.

On the other hand, *gPrune* [ZYHP07] defines two properties of the constraints, called *P-anti-monotonicity* and *D-antimonotonicity*, that are weaker than the anti-monotonicity and the monotonicity, yet allow a significant pruning of the pattern and the data space, respectively. Thank to these properties, the gPrune framework is able to mine all the valid patterns in a pattern-growth fashion, without enumerating all the frequent patterns. The framework considers three structural constraints, namely the degree, the density, and the connectivity. Following the minimum description length principle, *Forage* [PN07] selects from the set of frequent patterns, those maximizing a scoring function that depends on the size of the pattern and its frequency. The score of a pattern is interpreted as an estimate of the space saved when all its occurrences in the graph database are replaced by a single vertex, and thus it allows the selection of a very limited set of both discriminative and representative elements.

Skyline patterns have been introduced to mine interesting patterns in a threshold-free manner. Given a set of objectives, a pattern *dominates* another pattern if it has a higher score in at least one objective and has the same value in all the others, and skyline patterns are those patterns not dominated by any other. *MOSubdue* [SQC13] defines the dominance in terms of number of nodes, frequency, and density, but claims it can work with any custom objective that can be expressed in a simple way. Pareto dominance is applied to perform both the search in the multi-objective subgraph space and the evaluation of the subgraphs discovered. The search is a single-objective heuristic search based on a beam search method, which generates promising subgraph-seeds according to the MDL principle, and extend them by one node or edge to create larger candidates. The evaluation is achieved by associating a d-dimensional objective vector to each subgraph and then ranking those vectors in ascending order.

All these approaches treat the nodes and the edges in the graphs equally, but real-world scenarios show that they can be associated with different weights representing their relevance or some quantity measured by the system. In protein-protein interaction networks, for example, the weight of an edge represent the strength of the interaction, while in transportation networks it can represent the speed limit of the corresponding road segment. In weighted pattern mining, application-specific real numbers are assigned to the elements of the graphs, enabling the use of weight-based constraints in the mining process. Eichinger et al. [EHB10] investigate three non-anti-monotone constraints, namely information gain, Pearson correlation, and variance, and present an algorithm based on gSpan that returns the frequent patterns satisfying the weight-constraint. The solution is approximate, as it relies on the anti-monotonicity of the frequency to prune the search space, and hence may not discover an infrequent pattern that complies with the weight-constraint. *ATW-gSpan*, *AW-gSpan*, and *UBW-gSpan* [JCZ10] are other variations of gSpan that incorporate three different weighting mechanisms to mine weighted frequent patterns. The first mechanism is the average total weighting, which divides the sum of the average weights in the graphs containing the pattern by the sum of the average weights of all the graphs. The second is the affinity weighting, which calculates the ratio between the minimum and maximum weight among the occurrences of the pattern in the graphs, and the average Jaccard distance between the pattern and the graphs in which it appears. Finally, the third is the utility-based weighting, which computes the reciprocal of the sum of the Jaccard similarities between the supports sets of endpoints of the pattern edges, and the ratio between the total weight of the pattern and the total weight of all the graphs. The first two mechanisms take advantage of the anti-monotone property to prune the search space, while the third one exploits an alternative less effective pruning technique. Additionally, *WFSM-MR* [BJ16] further extends such approaches in a distributed manner on top of the MapReduce framework.

### 2.1.2 Single Graphs

As opposed to the graph database case, the frequency defined for the single-graph setting do not possess the anti-monotone property, due to the presence in the graph of occurrences, i.e., *embeddings*, of the same pattern that share some edges (i.e., they overlap) [FB07]. These embeddings count as a single contribution to the frequency of the overlapping edges, but as a separate contribution to the frequency of the larger pattern, meaning that the larger pattern achieves a higher frequency than its sub-pattern. As a consequence, alternative different measures have been introduced to allow an effective pruning of the search space [BN08].

*Subdue* [HCD+94] is the first pattern mining algorithm in single graphs, but instead of mining the frequent patterns, it adopts an approximate greedy strategy based on the Minimum Description Length to mine the subgraphs that can better compress and thus describe the original graph. In addition, it allows the user to provide background knowledge in the form of domain-dependent or domain-independent rules that will be used to evaluate the subgraphs and thus bias the search towards specific types of structures. The first apriori-compliant frequency metric proposed in the literature is the *maximum independent set* (MIS) support [VGS02], which counts each pair of overlapping embeddings at most once. This metric is used by *hSiGraM* and *vSiGraM* [KK05], two pattern-growth algorithms that find the frequent subgraphs in a graph by following a horizontal and a vertical approach, respectively. The horizontal approach discovers the frequent patterns in a breadth-first fashion, while the vertical approach traverses the search space in a depth-first fashion. They propose both an exact and an approximate discovery, for which they compute, respectively, the MIS supports exactly and approximately. The exact MIS supports are computed using a maximal clique algorithm, while a greedy algorithm is used to approximate their size. vSiGraM achieves better performances than hSiGraM, as it stores all the embeddings of the frequent patterns discovered in the previous iteration to speed up the isomorphism tests for the current candidate patterns. *GREW* [KK04], instead, calculates the MIS support of a pattern using a greedy maximal independent set algorithm that quickly finds a subset of the vertex-disjoint embeddings of the pattern, hence leading to an underestimation of the real support of the pattern. To increase the efficiency of the algorithm, the computation of the frequent patterns is iteratively performed on the augmented graph built by collapsing the embeddings of the frequent patterns into vertices and augmenting the edges incident to these vertices with information present in the original graph. At the beginning of each iteration, GREW identifies the edge-types that occur enough times in the augmented graphs and processes them in descending order of number of appearances. At the end of the iteration, if no edge types has MIS support greater than the minimum frequency threshold, the algorithm terminates; otherwise, it discards the infrequent edge-types and updates the augmented graph for the next iteration.

The second metric proposed is the *Harmful Overlap* (HO) support [FB07], which categorizes the pairs of overlapping embeddings into *harmful* and *harmless*, and then counts each harmful pair at most once, while the harmless pairs are considered separate contributions. However, calculating the MIS or the HO support is an intractable problem [GJ79], rendering them unsuitable in many practical scenarios.

In contrast, the *minimum number of node images* (MNI) support can be computed efficiently [FB07], and thus has been extensively exploited by more recent works such as *GraMi* [EASK14] and its parallel extension *ScaleMine* [AAK$^+$16]. These algorithms optimize the computation of the frequent patterns by modeling the search of the embeddings of a pattern as a *constraint satisfaction problem*, denoted as $CSP(X, D, C)$. According to the CSP model, an embedding is isomorphic to a pattern $P : (X, D, C)$ if it is a valid assignment of the variables in $X$ to variables in $D$, and it satisfies all the constraints in $C$, which are node and arc consistency constraints that define how the variables in $X$ are connected in the pattern $P$. To avoid the computation and the storage of all the embeddings of the patterns, the two algorithms search for valid assignments on-the-fly, terminating the search process as soon as the set of embeddings found pass the minimum frequency threshold.

With the goal of defining a support measure that satisfies the downward closure property, as well as being intuitive and significant, Han et al. [HW13] introduces the concept of *pivot* and a new type of pattern called *frequent neighborhood pattern*. Having selected a vertex of each subgraph as pivot, they treat two subgraphs as identical if they are isomorphic and map the pivot to the same node of the graph. The support of a pattern, or *neighborhood*, is then defined as the number of distinct nodes in the graph to which its pivot is mapped, and the problem of finding patterns with large support is called frequent neighborhood pattern mining.

However, all the previous frequency-based approaches focus their attention solely on the structure and the popularity of the patterns, disregarding any other user-defined constraint or user preference. Little attention has been focused on the development of constraint-based algorithms that find patterns satisfying some notion of relevance. *SkyGraph* [PLM08] is the first contribution to the field, which expresses the relevance in terms of number of vertices and edge connectivity, and mines patterns maximizing these two objectives by recursive graph partitioning. At each iteration, SkyGraph uses a MIN-CUT algorithm to obtain the partitioning that maximize the edge connectivity within the partitions, and then performs the multi-objective evaluation of the subgraphs obtained in order to determine if they can be part of the solution. Finally, the set of current best patterns is updated and the next iteration starts. Since the search of the candidate subgraphs is guided by the edge connectivity objective, this algorithm is not designed to work with other user-defined constraints like preferences expressed in terms of weights on the elements of the graph.

The first work on weighted pattern mining in single graphs is *WIGM* [YSLD12], which quantifies the importance, or *weighted support*, of a pattern as the ratio between the sum of the edge weights of its appearances, and the number of edges in the pattern. Since this measure do not satisfy the anti-monotone property, WIGM bounds the search space using a novel but weaker pruning strategy called *1-extension property*. In addition, it maintains an index on the subgraphs in the graph, in order to speed up the computation of the weighted supports.

Uncertain graphs are graphs that include existence probabilities for their nodes and edges, and thus they can be seen, to some extent, as a special case of weighted graphs. Several works have been proposed to mine frequent patterns in those graphs [ZLGZ10, JKHB11, PIS11, CZLW15, WRS17, LZG12]; however, as opposed to weighted graph pattern mining algorithms, their support measures are defined in terms of the uncertainty of the edges and compute the support of a pattern as an expected value. As a consequence, they are not designed to work with general weighted graphs.

Differently from *WIGM* and all the frequency-based algorithms, we introduce a family of scoring functions that assess the relevance of a pattern in terms of the weights of its appearances, and yet satisfy the anti-monotone property, hence allowing efficient implementations. Moreover, we propose a more general framework that supports multi-weighted graphs, and thus is able to mine the relevant patterns for each set of weights, effectively and efficiently.

## 2.2 Dense Subgraph Mining

Density has been considered a measure of importance in many different kinds of networks, ranging from social networks, where dense groups of nodes and edges can represent communities of people that share similar interests [CS10, ASKS12]; to biological networks, where dense subgraphs can indicate biological modules [BH03, HYH$^+$05]; to the World Wide Web, where a spam page is characterized by a large number of incoming links [BXG$^+$13].

### 2.2.1 Single Graphs

The literature abounds with works aiming at extracting dense subgraphs from a single graph. These works differ in how they calculate the density of a subgraph, and how they select the dense subgraphs to return. The ideal form of dense component is represented by the clique, which is a fully-connected subgraph. However, determining if a graph contains a clique of size $k$ is a well-known $\mathbf{NP}$-complete problem [Kar72], and thus alternative notions of density have been proposed [ARS02] and approximate algorithms have been developed [GKT05]. Goldberg [Gol84], for example, defines the density of a subgraph as the average node degree and proposes an approach that finds the subgraph with maximum average degree by iteratively applying a max-flow/min-cut algorithm that decomposes the current node set into two partitions. On the other hand, Charikar [Cha00] reduces the problem to a linear programming task, and describes a greedy algorithm with 2-approximation guarantee that is linear to the number of edges and nodes in the graph. The algorithm iteratively removes the node with lowest degree, until the graph becomes dense or empty. Other approximate works have considered the problem of extracting the densest subgraph with at least k nodes (DALKS), the densest subgraph with at most k nodes (DAMKS), and the densest subgraph with k nodes (DKS), which are all $\mathbf{NP}$-complete problems [AC09]. DALKS is solved with a $1/3$-approximate algorithm based on core decomposition, that iteratively removes the node with minimum weighted degree and finally chooses the subgraph with maximal density among those discovered during the process. DKS is solved with a $1/4$-approximate algorithm Rozenshtein et al. [RAGT14], instead, take a weighted activity network in input, and detect the subgraph that maximizes the sum of the weights and minimizes the sum of the pairwise node distances. The two objectives are combined in a single objective function, which is optimized using a randomized double-greedy algorithm with $1/2$-approximation guarantee [BFSS15]. The paper proposes also a slower but more effective solution based on the MAXCUT problem [GW95]. Finally, Balalau et al. [BBC$^+$15] generalize the densest subgraph problem to the problem of detecting at most $k$ subgraphs such that the sum of the average degrees is maximized, and the maximum pairwise Jaccard similarity between their node sets is at most $\alpha$. They devise an exact algorithm based on Charikar's approach [Cha00], as well as a fast heuristic that allows the algorithm to scale to graphs with 100 million edges. The algorithm iteratively executes two subroutines called TRYREMOVE and TRYENHANCE, which produce minimal densest subgraph at each step. The former receives a node, and checks if it can be removed from the graph without decreasing its density; while the latter receives a node and a density value, and returns the subgraph that contains the node and has the given density. The algorithm terminates when $k$ subgraphs are found, or when the graph becomes empty.

### 2.2.2 Multiple Graphs

The problem of finding dense components has been extended also to datasets that consists of a set of graphs modeling a dynamic network, with applications like detecting congested locations in road networks [BCK$^+$03] and real-time story identification in social networks [ASKS12]. However, applying algorithms for the static case to this setting is generally inefficient [BMS11] and ineffective [YYG$^+$14], as the temporal dimension brings a new wave of challenges that require more specialized solutions.

Several approaches have been developed with the goal of extracting dense subgraphs from a dynamic network. Bogdanov et al. [BMS11] extract the highest-scoring temporal subgraph from a dynamic weighted network, which is defined as the subgraph with largest sum of edge weights in a sub-interval of graph snapshots. The paper proves that the problem is $\mathbf{NP}$-hard and proposes a filer-and-verify approach that effectively prunes the quadratic sub-interval space and efficiently verify the promising sub-intervals. Ma et al. [MHW$^+$17] propose a more scalable solution that employs hidden statistical characteristics of the time intervals to detect $k$ candidate sub-intervals and then extracts the dense subgraphs in each sub-interval by reducing the problem to the Prize Collecting Steiner Tree problem.

Yang et al. [YYG$^+$14] focus on capturing the most frequently changing structure in a series of graph snapshots, i.e., the subgraph that maximizes the density of the connectivity change between each pair of the subgraph nodes. The connectivity change between two nodes between two snapshots is measured using the maximum number of independent paths, and the algorithm proposed computes the cumulated connectivity change between two nodes in the entire sequence of snapshots by iterating over the snapshots and computing the edge connectivity using a max-flow algorithm twice at each step. Then, the algorithm constructs a static graph $\bar{G}$ by merging all the snapshots together and assigning a weight to each edge equal to the number of snapshot where it appears, and finally it extracts the minimum spanning tree from $\bar{G}$.

The problem of finding the densest temporally compact subgraph in the context of interaction networks is studied by Rozenshtein et al. [RTG17]. The temporal compactness of a dense group of nodes with respect to a set of time intervals, is ensured posing a constraint on the number of intervals in the set, and on their total length. The two algorithms proposed are based on Charikar's greedy algorithm [Cha00] and differ in how they try to optimize the two temporal constraints. One approach optimizes the two constraints in an alternating fashion with binary search, while the other approach aggregates the two constraints into a single gain/cost ratio optimized in a greedy fashion. The performance of two algorithms is improved by exploiting the concavity property of the objective functions and applying fractional programming.

Charikar's algorithm has been exploited also by Semertzidis et al. [SPTT18] to find the group of nodes most densely connected in all the snapshots of a dynamic graph. They measure the density of a group of nodes as either the minimum node degree or the average node degree, and then the density of a group over time as either the minimum or the average density across the snapshots, thus obtaining four variants of the problem, i.e., BFF-MM, BFF-AA, BFF-MA, and BFF-AM. The paper presents a generic greedy algorithm that solves BFF-MM and BFF-AA optimally in polynomial time, while no theoretical guarantees are proved for BFF-MA and BFF-AM. Furthermore, the paper solves the problem of finding the group of nodes and the set of $k$ snapshots such that the density of the group in these snapshots is maximized. The first solution is an iterative algorithm that starts with a set of snapshots and a set of nodes and tries to improve them at each step; while the second solution is an incremental algorithm that builds the solution incrementally starting with a set of two snapshots.

Although some of these works can be modified to retrieve multiple subgraphs (e.g., by iteratively applying the algorithm and removing the solution found), they would detect only non-overlapping subgraphs, and more importantly, they would not scale to very large graphs. The enumeration of dense structures has been studied in the context of frequent subgraph mining, community evolution, and temporal subgraph mining. In frequent dense subgraph mining in dynamic networks, the goal is to extract components that are dense and that occur frequently in the snapshots of the network. Yan et al. [YZH05] develop a pattern growth and a pattern reduction approach that exploit the minimum cut clustering criterion to compute all the frequent closed patterns that satisfy given connectivity constraints. Abdelhamid et al. [ACS$^+$17] propose an incremental approach to continuously report the frequent patterns, which employs three novel techniques to save space and time, among which, (i) it uses an efficient index to store a minimal number of occurrences of a selected subset of patterns, (ii) it supports batch processing, and (iii) it reorders the execution of the isomorphism tests needed for computing the support values based on information collected during past graph updates, with the goal of improving the efficiency of the next iterations. An high-quality approximation of the subset of frequent patterns with $k$

vertices is instead computed by Aslay et al. [ANDFMG18]. Their approach is based on two components, i.e., a reservoir of subgraph samples and an exploration procedure. The subgraph samples are used to capture the changes in $k$-subgraphs previously sampled using a scheme with guarantees on its accuracy, while the exploration procedure includes newly (dis)connected $k$-subgraphs into the sample.

Aggarwal et al. [ALYJ10] define a density measure in terms of node co-occurrence and edge density, and determine the frequent dense patterns in a dynamic network using a probabilistic algorithm. The algorithm consists in two step. In the first step, it detects the correlated node patterns, which are groups of nodes that frequently appear together in the same snapshots; while in the second step, it determines the subset of nodes that satisfy a given edge-density constraint. To speed up the detection of the correlated nodes, the algorithm maintains a set of min-hash values for each node that are updated as new snapshots of the network are processed. On the other hand, Quin et al. [QGY13] define six types of significant structures characterized by different pattern of appearance in the snapshots, and introduce a two-step algorithm that creates a summary graph, deletes the infrequent edges, and finally uses a density-based clustering algorithm to obtain the set of subgraphs with the desired properties. In the field of bioinformatics, Hu et al. [HYH$^+$05] find the coherent dense subgraphs that appear frequently in a sequence of relational graphs. The temporal component is encapsulated into a time-series-based edge similarity function, and the coherence of a subgraph is measured as the total pairwise edge similarity. The main steps of the algorithm proposed are the construction of a summary graph that contains the edges appearing in at least $k$ graphs, and the construction of a second-order graph that represents the edges of the original graph with similar binary series of appearance. Then, the algorithm extracts all the subgraphs that are simultaneously dense in both meta-graphs, by first computing the dense subgraphs in the summary graph, and secondly computing the dense subgraphs in the second-order graph of each subgraph found in the first step. The search for the dense subgraphs is carried out by recursively partitioning the graph following a normalized-cut and a min-cut approach.

Works in community evolution track the progress of dense groups of nodes over time. Kim et al. [KH09] describe a particle-and-density-based clustering algorithm able to extract communities of arbitrary forming and dissolving, addressing the challenges of how to cluster a graph in a variable number of communities, and how to connect the communities across time given that the number of communities is variable. The first challenge is solved by means of a density-based clustering algorithm that extracts smoothed clusters using an efficient cost embedding technique and optimal modularity. The second challenge is tackled via a mapping method that can identify the status of each community at each snapshot, i.e., evolving, forming, and dissolving. Greene et al. [GDC10], instead, identify the dynamic communities of a dynamic graph using an incremental dynamic clustering approach independent of the choice of the underlying static community detection algorithm. The approach starts with the detection of the communities in the first snapshot and the initialization of the dynamic communities as sets containing the corresponding static community. Then, it applies a community detection algorithm to each subsequent snapshot and uses a heuristic threshold-based method to perform a many-to-many mapping between the existing dynamic communities and the current static communities. Communities without a match will constitute a new dynamic communities, while the others are added to the corresponding dynamic community.

Finally, temporal subgraph mining focuses on the detection of subgraphs of a dynamic graphs that satisfy given temporal and structural constraints. TimeCrunch [SKZ$^+$15] searches for the coherent temporal patterns that best minimize the encoding cost of the dynamic graph. It defines 6 types of dense structures and 5 types of temporal signatures that model the connectivity and periodicity of the structures, and use them to encode each static subgraph in each snapshot, following the Minimum Description Length paradigm. Then, it stitches the static subgraphs with the same connectivity behavior together to obtain a set of temporal subgraphs, and finally uses the VANILLA, TOP-10, TOP-100, and STEPWISE heuristics to select the non-redundant temporal structures that best summarize the dynamic graph. Crochet [PJZ05], on the other hand, enumerates all the maximal groups of nodes that induce a dense subgraph of minimum size in all the input graphs. When the input is a dynamic network, these groups represent subgraphs that persist over time. They prove the problem is $\#\mathbf{P}$-complete, and describe an algorithm that enumerates the subsets of nodes using a set enumeration tree, and then conducts a depth-first search on the tree. At each step of the search, the algorithm uses effective strategies to prune futile subtrees and decide the order of exploration of the next children, hence speeding up the mining process.

Other approaches restrict their enumeration to the top $k$ subgraphs. Yang et al. [YYW$^+$16] find the $k$ diversified $\gamma$-quasi-cliques with minimum size, which are subgraph that are $\gamma$-dense in a time interval of minimum size. A divide-and-conquer algorithm starts with the entire dynamic graph and the entire set of time instances, recursively divides the current graph into two parts, applies effective pruning rules to remove futile vertices, and search for the maximal dense subgraphs induced by each part. During the whole mining process, the algorithm maintains the $k$

best subgraphs using a greedy procedure. Nasir et al. [NGMG17] detect the $k$ node-disjoint subgraphs with maximal total density in dynamic graphs modeled as streams of edges. Given the hardness of the problem, they propose a greedy algorithm with 2-approximation guarantee for the first subgraph extracted, and $2k$-approximation guarantee for the top-$k$ subgraphs. The algorithm incrementally stores the strongly connected subgraphs using a novel memory-efficient data structure called *snowball*, which handles all the graph update operations and allows the extraction of the top-$k$ disjoint subgraphs on-the-fly.

All these works, however, find subgraphs that satisfy only density and structural constraints, and therefore fail in enumerating all the dense correlated subgraphs in a dynamic network. In anomaly and fraud detection, other measures have been considered, together with the density, with the goal of discovering rare changes in the snapshots of a dynamic network, with applications in security [DKB$^+$12], and finance [MBA$^+$09] among others. Jiang et al. [JBC$^+$15] propose an axiomatization of metrics to measure the suspiciousness of dense blocks in tensors. Then, they develop an algorithm that detects dense regions of anomaly in dynamic networks modeled as tensors, assuming that the dense regions are randomly distributed according to a Poisson distribution, and using the negative log likelihood as suspiciousness measure. Similarly M-Zoom [SHF16] detects only the top-$k$ dense blocks. Spotlight [EFGM18], instead, spots anomalous graphs in dynamic bipartite graphs, using a sketch space to save space and time. A graph is anomalous if its density significantly increased or decreased compared to the previous snapshots of the network.

Other methods bound the search for anomalous patterns to a time window. They model the normal behaviour by means of the previous snapshots of the network, and compare it against the current snapshot to determine if it is anomalous. Netspot [MBR$^+$13], for example, computes the anomalousness of each edge in each snapshot of a weighted dynamic network as the statistical p-value according to the distribution of the edge weights over time. Then, it iteratively tries to optimize the time interval that gives the highest sum of anomaly scores for a given set of edges, and the set of edges that leads to the highest sum of scores in a given interval. The algorithm outputs subgraphs that are more anomalous than a given threshold. However, these groups are in general not correlated.

All these works, however, focus their search on the statistically significant structures, and thus the edges within the discovered subgraphs may not be correlated in time. A notion of correlation has been introduced by Guan et al. [GYK12] and Yu et al. [YAMW13]. The first work proposes a novel measure for assessing the structural correlation of two events in a graph. The main idea is to select uniformly a subset of *reference* nodes from the vicinity of all the nodes in the two events, and compute the average concordance of density changes of the two events between two reference nodes using the Kendall's $\tau$ rank correlation measure. High concordance scores indicate that the occurrence of one event tends to attract the occurrence of the other event, while low concordance scores indicate a repulsion. This approach, however, is designed for static graphs and assumes that the two events are given in input. On the contrary, the second work detects anomalous hot spots in a stream of edges that model a dynamic network. Hot spots are defined as localized regions of sudden activity or change, and are dynamically determined by a localized principal component analysis algorithm that computes a decay factor for each edge and time stamp, and calculates the correlation between two edges as the correlation between the decay factors. The mean and standard deviation of the magnitude change and the correlation change at each node are continuously maintained with the goal of extracting those nodes that unusually deviate during some time interval. An algorithm to search for graphs that have more statistical co-occurrence ratio in a graph database have been introduced by Samiullah et al. [SAF$^+$14]. This algorithm efficiently mines graphs that are both frequent and correlated, using a novel correlation measure called *gConfidence* defined as the ratio between the frequency of the graph and the maximum among the frequencies of its subgraphs. The mining process follows a pattern-growth strategy similar to the one used by gSpan [YH02] that exploits the anti-monotonicity of the frequency and the correlation measure, to prune the search space. Even though this algorithm can be effectively applied to dynamic graphs modeled as sequences of static graphs, the subgraphs extracted do not satisfy any density constraint. In contrast to these works, our goal is to extract groups of correlated edges that are dense.

In the field of correlated dense subgraph mining, Chan et al. propose the greedy algorithm CStag [CBL08] and its incremental and more scalable version ciForager [CBLH12], which can find regions of correlated temporal change in dynamic graphs. The temporal similarity between two edges is defined as the Euclidean similarity between the corresponding time series, while the spatial closeness is measured in terms of the shortest path distance. The proposed algorithms iterate over the snapshots of the graph using a window of fixed size, and for each window, they cluster the edges using first the temporal distance and then the spatial distance. To save computational time, they adopt heuristics to calculate the temporal distances, and exploits a data structure called union graph to compute the spatial distances. However, these approaches partition the edges into a possibly very large number of regions, i.e., each edge of the graph is part of the output. In contrast, we aim to enumerate only the subgraphs with large density and high pairwise edge correlation, and in addition, our approach is independent of the choice of the underlying spatial and temporal measure.

## 2.3 Entity Resolution

Entity resolution (ER) has been extensively studied in the context of data integration [SGV06], data cleaning [BG04], and information retrieval [KJDR08], among others, and for both structured and unstructured data, such as Web data [CES15], genealogical data [ERSR$^+$15], and social networks [BLGS06]. The exact solution to this problem computes the similarity between each pair of records in the dataset and then clusters the records, aiming to maximize the intra-cluster similarities and minimize the inter-cluster similarities. To improve on the quadratic complexity of this solution, ER algorithms usually reduce the number of candidate records to compare using a strategy called *blocking* [MK06, PIP$^+$12, EIV07, Chr12], which assigns a blocking key to each record based on its attribute values, and places the records into blocks according to some blocking criteria. Since the comparisons will be executed only between records in the same block, the effectiveness of a blocking criteria is measured as the number of missing matches, while its efficiency is given by the number of comparisons saved. The literature abounds with different strategies that can be classified into schema-based approaches and schema-agnostic approaches, and that create either overlapping or non-overlapping blocks, which are processed in a predefined order [HS95, GIJ$^+$01, MNU08, dVKCC09]. Whang et al. [WMK$^+$09] propose a more advanced technique called *iterative blocking*, which exploits the matches detected in previously processed blocks to save comparisons in the following blocks and to infer new matches. The main idea is to first run a core ER algorithm on each block, and then, whenever a pair of duplicate records is found, merge the records and replace their occurrences in all the blocks with the unified record. The already examined blocks that contain either of the duplicate records are re-processed to identify more duplicates, and the process is repeated until no more matching records can be found in the blocks. This technique is also used by Papadakis et al. [PINF11] in their schema-agnostic approach designed for records coming from heterogeneous sources. Assuming that two matching records have at least one value in common, the algorithm proposed groups the records considering only their values, defines a block processing order by comparing the cost for processing a block against the corresponding gain, and finally processes the ordered set of blocks. Oversized blocks and blocks whose cost exceeds an upper bound are discarded, and the duplicates detected are propagated via iterative blocking.

To reduce the likelihood of missed matches, overlapping blocking strategies place the records into multiple blocks, but this redundancy comes at the cost of lower efficiency, as the number of comparisons to perform is higher. Therefore, block processing techniques have been introduced with the goal of decreasing the number of redundant comparisons without any significant impact on the effectiveness. These techniques are applied after the block creation step, to restructure the initial block collection into a smaller but effective new collection. Meta-blocking [PKPN14], for example, is a block processing method that relies on a graph built from the block collection to remove the comparisons that are likely to link dissimilar records. A node is created for each record, and every pair of records that co-appear in some block is linked with an edge. The edges are then weighted using one of the five schema-agnostic schemes that trades between computational cost and gain of comparing the adjacent records. Those edges that do not satisfy a given weight constraint are removed, while each of the remaining edges is transformed into a new block containing its adjacent nodes. Papadakis et al. [PPPK16a] further improve the performance of meta-blocking [PKPN14] with three node-centric pruning algorithms, i.e, Redundancy Pruning, Graph Partitioning, and Reciprocal Pruning; and with a block filtering algorithm that removes every record from the blocks that are the least important for it. In addition, they study how to select the best pruning scheme a-priori, depending on the application and the resources.

To deal with applications that have limitations on the maximum response time or the available resources, Simonini et al. [SPPB18] propose a progressive schema-agnostic algorithm based on blocking, which generates on-line the most promising pairs of records in decreasing order of matching likelihood. In this way, it can identify matching records earlier, hence providing the best possible partial solution. This algorithm relies on two principles, i.e., it assumes that matching records have blocking keys that are closer in alphabetical order, and that the matching likelihood of two records is proportional to the number of shared blocks.

### 2.3.1 Temporal Data

When the records in the dataset are associated with temporal information, we talk about temporal entity resolution. In temporal entity resolution, the goal is to identify records that describe the same real-world entity over time, hence tracing the history of that entity. Since traditional ER algorithms are agnostic to the temporal dimension, they fail to identify entity changes over time [LDMS11]. In contrast, temporal matching approaches typically rely on a temporal model able to capture the evolution of the attribute values, hence allowing to detect which records can represent the same real-world entity at different times.

The pioneering work in this context is the paper of Li et al. [LDMS11], which compares the temporal records using a weighted similarity function that assigns penalties to value agreements and disagreements based on the time gap of the two records. This function favors value differences over a long time gap (*disagreement decay*) and penalizes value similarities over a long time gap (*agreement decay*). Decay values are learned for each attribute using a labeled training set. In addition, the paper proposes three temporal clustering methods that process the records in temporal order. Early binding merges a record with the most similar cluster among the existing ones; late binding compares a record with each existing cluster but makes the clustering decision at the end; and adjusted binding, used in addition to one of the previous methods, compares a record with clusters created at later times to adjust the clustering results.

Chiang et al. [CDN14a] improve the accuracy and the robustness to noise of [LDMS11] via a more complex model able to learn more detailed patterns of value change. The model is based on the notion of mutation, which gives the probability that an attribute of an entity has a value never appeared in that attribute in the history of that entity; and on the notion of recurrence, which gives the probability that an attribute value of an entity recurs after a time interval $\Delta_t$. The algorithm proposed cluster the records by processing them in increasing temporal order, and computing the temporal similarity between two records as the weighted average of the attribute value similarities based on the weights obtained from the mutation model. Low weights are assigned to attributes that tend to change over time, as considered less reliable. The temporal model introduced in [LDMS11] forms the basis of the two-step temporal clustering algorithm proposed by Chiang et al. [CDN14b]. The first step of the algorithm assumes the records are static and groups them according to the value similarities. The second step merges the clusters by deciding if it is possible for an entity to evolve from the state described in one cluster to the state described in another one.

On the other hand, Li et al. [LLHT15] present a temporal entity matching algorithm that use a value transition model and a source freshness model to track the evolution of the entities over time, hence building entity profiles. The transition model captures the probability that an entity change to a particular value for a particular attribute after some period of time, while the source freshness model captures the freshness of a data source on an attribute. The algorithm starts with clustering the records depending on the freshness of their source, placing stale records into multiple clusters if need be. Then, given an entity profile, it iteratively identifies the subset of clusters that match with the profile, and updates the profile by adding the records in those clusters. The matching decision depends on both the value transition probabilities and the support of the data sources. This work is extended in [LLH17] with an approach that considers not only the freshness of the data sources, but also their reliability. Reliability is measured in terms of accuracy and coverage, which indicate, respectively, the probability that an attribute value from a source conforms to the real world and the probability that a change of a value in the real world is captured by a source. They propose a source-aware matching algorithm that relies on a transition model, a freshness metric, and a reliability metric, to augment the entity profiles with correct values at the right times. The matching algorithm works in two phases, which are the clustering of the records into a set of clusters representing the status of an entity over some time period, and the matching of the clusters to the entity profiles to augment them with the best clusters. On the other hand, Ranbaduge et al. [RC18a, RC18b] address the task of temporal entity resolution between datasets containing sensitive information. They propose a privacy-preserving temporal record linkage (PPTRL) protocol able to link the entities in the temporal databases without sharing confidential information between their owners. This protocol uses a homomorphic-encryption-based technique to learn the decay values for each attribute, which are exploited to generate masking Bloom filters to adjust the similarity values between the records.

An adaptive approach is presented by Christen et al. [CG13], that is able to adjust the similarities between two records based on both their attribute values and their time difference. This algorithm uses the same weighted similarity function introduced in [LDMS11], but adopts an efficient technique to calculate the weights in an adaptive fashion, which incorporates the frequency distributions of the attribute values in the calculation of the agreement probabilities. Given a stream of new records, the matching algorithm processes them in order of arrival, and updates the weights after each matching decision. All these approaches assume the records have the same schema, whereas we focus on heterogeneous data. In addition, in contrast to these approaches, we offer a model-free solution that does not require a training dataset to learn how the entities evolve over time, and yet is effective in identifying a good-quality approximation of the set of matching records in a temporal database.

Finally, we note that our problem may resemble a time-series matching problem, where the goal is to retrieve, in a database with time-series, those series (or sub-series) that match a query (sub)series exactly or approximately. Existing works in this field define a time series as an ordered sequence of real numbers, where each number represents a value measured at a particular time instance, and typically transform it into a feature vector of lower dimension to handle also high-dimensional data [Lia05, EA12]. However, while structured records can be easily transformed into real-valued time series by deciding an ordering of the attributes and mapping the attribute values to integers, a consistent mapping can not be efficiently and effectively obtained for heterogeneous dynamic records, for two reasons. First, dynamic records can have different sets of attributes over time and new attributes can appear in successive updates to the database, and therefore a fixed ordering cannot be selected a-priori. Secondly, the values of the attributes of a dynamic record can change over time, and their concatenation results in a very large feature vector that is hard to project to a low-dimensional space.

# Chapter 3

# Score-based Graph Pattern Mining

In this chapter we present the first contribution of this thesis, which is the formulation of a novel graph mining task that focuses on the discovery of patterns that are relevant according to individual user's preferences; and an efficient and effective solution that assumes that those preferences can be expressed as multiple weights on the edges of the graph, and mines the patterns using a special scoring function that assess the relevance of a pattern in terms of the weights of its appearances in the graph, as well as its frequency. As a consequence, we are able to provide personalized results, rather than "one size fits all" solutions that satisfy the user on average.

## 3.1 Contributions

The contributions of this chapter can be summarized as follows:

- We extend the task of pattern mining in weighted graphs for a novel family of scoring functions that are based on the MNI support, which guarantees they are efficient to compute, and on the edge weights, meaning that they can identify the most relevant weighted patterns in the graph efficiently and effectively (Section 3.2). We call these functions *MNI-compatible* scoring functions.
- We formally introduce the problem of mining patterns in graphs with multiple weights on the edges, which has the goal of discovering the most relevant weighted patterns for each set of edge weights (Section 3.2).

- We devise two centralized algorithms for solving pattern mining in multi-weighted graphs (Section 3.4). RESUM is an exact solution less time and space consuming than the naive approach that extracts each set of patterns separately, as it avoids redundant revisits of the graph by aggregating and performing once multiple computations on the same parts of the graph, and it stores the relevant patterns in a compact way. On the other hand, RESUM *approximate* is a conservative approximate solution that reduces the number of edge-weight functions to consider in the mining process by aggregating those having a high probability to share similar results into a single representative function (Section 3.5).

- We develop a distributed version of RESUM, called RESUM *distributed*, which runs on top of the distributed graph processing system Arabesque [TFS+15] and is able to scale to larger and richer graphs (Section 3.6).

- We study four MNI-compatible scoring functions that can be effectively used to mine the relevant patterns in different applications and in graphs with weights drawn from different distributions or with different characteristics (Section 3.7).

- We evaluate our approaches with an extensive set of experiments on both real and synthetic graphs and discuss our findings. In particular, we showcase advantages and limitations of RESUM when compared to the approximate and the distributed algorithm; we show that RESUM *approximate* introduces only few false positives, while running considerably faster than the exact approach; and we discuss in which cases a distributed solution can be beneficial, and when a centralized solution is rather preferable (Section 3.8).

## 3.2 Problem Formulation

We first introduce the graph concepts and notation that we use in the formulation of our problem and in the further sections.

Let $\Sigma$ be a countable set of labels, and $\mathcal{I} = [0,1] \cup \{\perp\}$ the interval of numbers between 0 and 1, plus a special symbol $\perp$ denoting *no weight*. A weighted labeled graph is a structure that consists of a set of nodes, a set of edges between nodes, an assignment of labels to nodes and edges, and an assignment of weights to edges. For presentation purposes we assume that the graph has weights only on the edges, but weights on the nodes can be considered as well with no need for any major modification.

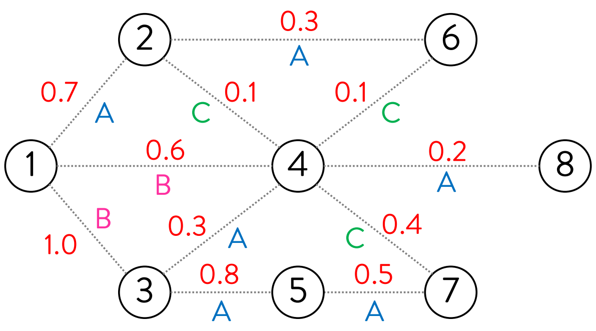


**Figure 3.1:** Example of a edge-labeled, weighted graph.

**Definition 3.1.** A **weighted labeled graph**, or simply a **graph**, is a tuple $G : \langle V, E, \ell, \omega \rangle$ where $V$ is a set of nodes, $E \subseteq V \times V$ is a set of edges, $\ell : E \cup V \rightarrow \Sigma$ is a labeling function, and $\omega : E \rightarrow \mathcal{I}$ is a weighting function.

Definition 3.1 assumes that the edge weights are in the range of $[0, 1]$. Nonetheless, this definition is not restrictive, as any positive or negative value can be scaled down to $[0, 1]$, while any categorical value can be mapped to some number according to its semantics. For example, if the values represent a range of preferences, such as *extremely needed*, *moderately needed*, and *not needed*, we can replace the top preference value with 1, the moderate value with 0.5 and the bottom value with 0. On the other hand, for the edges that have no weight, the 0 or $\perp$ value can be assumed.

A graph $S : \langle V_S, E_S, \ell_S, \omega_S \rangle$ is a **subgraph** of a graph $G : \langle V_G, E_G, \ell, \omega \rangle$ (denoted as $S \sqsubseteq G$), if $V_S \subseteq V_G$, $E_S \subseteq E_G$, $\forall n \in E_S \cup V_S$ , $\ell_S(n) = \ell(n)$, and $\forall e \in E_S$ , $\omega_S(e) = \omega(e)$, meaning that they assign the same labels and the same weights to the common nodes and edges.

To express the fact that two graphs share the same topological structure, we introduce the notion of *isomorphism*, which is a bijective mapping between the nodes of the two graphs such that the edges between the nodes, alongside their labels, are preserved through the mapping.

**Definition 3.2.** A graph $G : \langle V, E, \ell, \omega \rangle$ is **isomorphic** to a graph $G' : \langle V', E', \ell', \omega' \rangle$ (denoted as $G \simeq G'$) if there exists a bijective function $\phi : V \rightarrow V'$ such that $\forall (u, v) \in E$ , $(\phi(u), \phi(v)) \in E'$ and $\ell((u, v)) = \ell'((\phi(u), \phi(v)))$.

We collectively represent a set of isomorphic graphs with a special labeled graph called *pattern*, which has no weights on its edges but simply describes the common structure of these graphs.

**Definition 3.3.** A **pattern** is a graph $P : \langle V, E, \ell, \omega \rangle$ such that $\forall e \in E$ , $\omega(e) = \perp$. Given a graph $G$, and a pattern $P$, the **support set** of $P$ in $G$ is the set of subgraphs of $G$ that are isomorphic to $P$, i.e., $S_G(P) = \{g | g \sqsubseteq G \land g \simeq P\}$. The elements in $S_G(P)$ are called *appearances*, or *embeddings*, of $P$ in $G$.

We denote by $\phi_g^P$ the bijection that maps a subgraph $g$ of $G$ to its isomorphic pattern $P$, and we call $P$ a subgraph of $G$ (denoted $P \sqsubseteq G$) if $S_G(P)$ is non-empty.

Frequent pattern mining in graphs is the data mining task that aims at identifying all the patterns that appear frequently in the input graph, according to a minimum frequency threshold $\tau$. Intuitively, a pattern can be considered a frequent pattern if its support set have cardinality higher than $\tau$, since this set contains all the appearances of that pattern in the graph. However, this simple method for determining the frequency of the patterns is unpractical for mining frequent patterns in large graphs, since it does not satisfy the anti-monotone property and hence requires the exploration of the entire search space. The violation of this property is caused by the presence of overlapping embeddings of the same pattern in the graph [BN08]. As an example, note that the frequency $P_1 : [v_1] - B - [v_2] - A - [v_3]$ in the graph in Figure 3.1 is 3, while the frequency of its sub-pattern $P_2 : [v_1] - B - [v_2]$ is 1.

With the goal of implementing efficient techniques to mine frequent patterns in large graphs, a lot of effort has been dedicated to define alternative frequency measures [HCD+94, VSG06, KK05, FB07], with the MNI support being the most effective and efficient to compute [BN08].

**Definition 3.4.** Given a graph $G : \langle V, E, \ell, \omega \rangle$ and a pattern $P : \langle V_P, E_P, \ell_P, \omega_P \rangle$, the **MNI support** of $P$ in $G$ is $MNI(P, G) = \min_{v \in V_P} |\mathcal{N}(G, v)|$ where $\mathcal{N}(G, v) = \{n | n \in V \land \exists g \in S_G(P) \,.\, \phi_g^P(n) = v\}$.

The set $\mathcal{N}(G, v)$ contains all the nodes of $G$ that are mapped to the pattern node $v$ by the isomorphisms $\phi_g^P$ of $P$, meaning that the MNI support of a pattern is simply the minimum number of mappings found for any of its nodes. For example, the pattern $P : [v_1] - B - [v_2] - A - [v_3]$ in the graph in Figure 3.1 has MNI support equal to 2 because $\mathcal{N}(G, v_1) = \{1, 3\}$, $\mathcal{N}(G, v_2) = \{1, 3\}$, and $\mathcal{N}(G, v_3) = \{2, 4, 5\}$. In contrast, the number of appearances of $P$ in $G$ is 3: $S_G(P) = \{[3] - B - [1] - A - [2], [1] - B - [3] - A - [4], [1] - B - [3] - A - [5]\}$.

Similarly, we denote as $\mathcal{E}(G, e)$ the set of edges of $G$ that are mapped to the pattern edge $e$, i.e., $\mathcal{E}(G, e) = \{a | a \in E \land \exists g \in S_G(P) \,.\, \phi_g^P(a) = e\}$.

In the presence of weights on the edges of the graph, the importance, or *score*, of a pattern should not be based on the frequency only, but rather strike a balance between frequency and weights, hence allowing the latter to play a role in assessing the relevance of the pattern. In these cases we thus need to replace the frequency measure with an alternative scoring function able to adjust the contribution of a subgraph to the score of its isomorphic pattern, according to the weights of its edges. The patterns discovered using this kind of function are called *weighted frequent patterns*, or *relevant patterns*.

When different weights are assigned to the edges by multiple weighting functions, the scoring function may produce different sets of relevant patterns. Such situation leads to the following problem formulation, which we call MULTIW-SPM.

**Problem 1** (Score-based Pattern Mining)**.** *Given a threshold $\tau$, a scoring function $f$, and a graph $G : \langle V, E, \ell, W\rangle$ where $W$ is a finite set of weighting functions, discover, $\forall \omega_i \in W$, the set of patterns $R_i = \{P | G' = \langle V, E, \ell, \omega_i\rangle \wedge f(P, G') \geq \tau\}$.*

In the following sections we introduce a novel family of scoring functions that be efficiently implemented and effectively used in MULTIW-SPM; our approach, called RESUM, that adopt one of those functions to compute all the sets $R_i$ exactly; our approximate solution, called RESUM *approximate* that can scale to larger numbers of weighting functions; and a distributed version of our framework, called RESUM *distributed*. Finally we present and analyze the four scoring functions we used in our experiments, and discuss our findings.

## 3.3 MNI-compatible Scoring Functions

Different semantics of the edge weights and different applications may demand different ways of aggregating the weights of the embeddings, meaning that there is no scoring function that is consistently better than the others. For this reason, we propose a class of scoring functions, called *MNI-compatible scoring functions*, that can accommodate a wide range of applications.

Assuming that larger weights indicate higher importance, a scoring function $f$ is a MNI-compatible scoring function if it satisfies the following properties:

**(i)** the larger are the edge weights in the embeddings of a pattern $P$ in $G$, the larger is the score $f(P, G)$;

**(ii)** the higher is the number of embeddings of $P$ in $G$ with positive edge weights, the larger is the score $f(P, G)$;

**(iii)** for each pattern $P$ in $G$, $f(P, G) \geq \tau \implies MNI(P, G) \geq \tau$.

Property **(i)** states that between two patterns with the same number of embeddings, the pattern whose embeddings have largest edge weights receives the largest score. On the other hand, Property **(ii)** guarantees that when all the embeddings of two patterns have the same edge weights, the pattern with more embeddings obtains the largest score. We note that these two properties are a natural consequence of our assumption on the importance of the weights. Last but not least, Property **(iii)** allows efficient solutions to the score-based pattern mining problem, as it ensures that we can use the same pruning strategies adopted in the MNI-based pattern mining algorithms. Note that, according to Property **(iii)**, a high frequency is a condition necessary but not sufficient to achieve a high score, i.e., $MNI(P, G) \geq \tau$ does not guarantee that $f(P, G) \geq \tau$. In Section 3.7, we introduce and analyze four scoring functions that satisfy the aforementioned properties.

## 3.4 The ReSuM Framework

Our solution to score-based pattern mining in weighted (and multi-weighted) graphs is a one-step process that explores the pattern search space only once and simultaneously computes the score $f(P, G)$ (all the scores $f(P, G')$ when $G$ is multi-weighted) of $P$ when $P$ is discovered. The computation of the score requires the identification of all the isomorphisms of $P$ that satisfy the constraints on the weights specified by the scoring function $f$.

### 3.4.1 Mining Single-weighted Graphs

We model the pattern mining problem as a *constraint satisfaction problem* (CSP) [DRZ07]. An instance of CSP is a tuple $(X, D, C)$ where $X$ is a set of variables, $D$ is a set of domains corresponding to the variables in $X$, and $C$ is a set of constraints between the variables in $X$. A solution for an instance of CSP is an assignment from the candidates in $D$ to the variables in $X$ that satisfies all the constraints in $C$. The matching problem for a pattern $P \sqsubseteq G$ is translated into $CSP(P) = (X_P, D_P, C_P)$, so that any solution to $CSP(P)$ corresponds to a subgraph $g$ isomorphic to $P$. Specifically, each node $v \in V_P$ is mapped to a variable $x_v \in X_P$, each domain $D_v \in D_P$ is a subset of $V$ containing all the graph nodes isomorphic to $v$, and $C$ includes consistency constraints that enforce a topology isomorphic to that of $P$ [Mac77]. We

**Algorithm 1** scoreBasedPatternMining

```
Input: Graph $G : \langle V, E, \ell, \omega \rangle$, threshold $\tau$
Output: Set of relevant patterns $R$
 1: $R \leftarrow$ RelevantEdges($G$)
 2: $fE \leftarrow$ FrequentEdges($G$)
 3: while $fE \neq \emptyset$ do
 4:     $e \leftarrow fE.pop$
 5:     $R \leftarrow R \cup$ patternExtension($G, e, \tau, fE \cup \{e\}$)
 6: return $R$

 7: function patternExtension($G, g, \tau, fE$)
 8:     $Cand \leftarrow \emptyset$; $\mathfrak{S} \leftarrow \emptyset$
 9:     for all $e \in fE$ do
10:         $Cand \leftarrow Cand \cup \{g \diamond e\}$
11:     for all $c \in Cand$ do
12:         $(score, sup) \leftarrow$ examinePattern($G, c$)
13:         if $sup \geq \tau$ then
14:             $\mathfrak{S} \leftarrow \mathfrak{S} \cup$ patternExtension($G, c, \tau, fE$)
15:         if $score \geq \tau$ then
16:             $\mathfrak{S} \leftarrow \mathfrak{S} \cup \{c\}$
17:     return $\mathfrak{S}$
```

discover solutions to $CSP(P)$ through an iterative process that examines each candidate node $n \in D_v$ and search for valid assignments that maps $n$ to $v$. If no assignment is found, $n$ is removed from the domain $D_v$ and the topology constraints are checked again until no invalid candidate is found in the other domains. At the end of the process, the number of elements in the smallest domain, i.e., $\arg\min_{D_v \in D_P} |D_v|$, corresponds to the MNI support of $P$, meaning that, given a threshold $\tau$, $P$ is frequent if no variable in $X_P$ has less than $\tau$ distinct valid assignments. On the other hand, the computation of the score of $P$ requires an additional step where we discard the assignments corresponding to embeddings that do not satisfy the weight constraints imposed by the scoring function $f$, and we aggregate the remaining ones.

Algorithm 1 outlines the ReSuM framework. The first step is determining the relevant and the frequent edges (Lines 1-2). Then, each subgraph is recursively extended following the pattern-growth approach introduced by gSpan [YH02] (Line 5), until no further extension is possible. Each extension is a candidate relevant pattern, whose MNI support is computed alongside its *score* by the examinePattern procedure (Algorithm 2). This procedure first initializes the candidate domain $D_v$ of each pattern node $v \in V_P$ with all the nodes in $G$ with the same label as $v$ (Lines 1-3), and the support set $sup_v$ of each node $v \in V_P$ with the empty set. Then, the

**Algorithm 2** EXAMINEPATTERN

```
Input: Graph $G{:}\langle V, E, \ell, \omega\rangle$, pattern $P$, threshold $\tau$
Output: Score and MNI support of $P$
 1: for all $v \in V_P$ do
 2:     $sup_v \leftarrow \emptyset$
 3:     $D_v \leftarrow \{v'{\in}V | \ell(v') = \ell(v)\}$
 4: $\mathcal{A} \leftarrow$ automorphisms of $P$
 5: STRUCTURALCONSISTENCY($\{D_v | v \in V_P\}, P$)
 6: for all $v \in V_P$ do
 7:     if $\exists\, w = \mathcal{A}(v)\ s.t.\ D_w$ already computed then
 8:         $D_v \leftarrow D_w$
 9:         continue
10:     STRUCTURALCONSISTENCY($\{D_v | v \in V_P\}, P$)
11:     if $\exists\, D_u\ s.t.\ |D_u| < \tau$ then return $(-1, -1)$
12:     for all $n \in D_v$ do
13:         search for $g$ s.t. $g \simeq P \ \wedge\ n \in V_g \ \wedge\ n \mapsto v$
14:         if $g \neq Nil$ then
15:             Valid $\leftarrow$ ISVALID($g, \omega$)
16:             for all $n' \in V_g$, $v' \in V_P$ s.t. $n' \mapsto v'$ do
17:                 mark $n'$ in $D_{v'}$
18:                 if Valid then
19:                     $sup_{v'} \leftarrow sup_{v'} \cup \{n'\}$
20:         else
21:             remove $n$ from $D_v$
22: $score \leftarrow$ RELEVANCESCORE($\{sup_v | v \in V_P\}$)
23: $mni \leftarrow min_{v \in V_P} |D_v|$
24: return $(score, mni)$
```

algorithm computes the automorphisms of the pattern (Line 4). Automorphisms are isomorphisms of a graph to itself and can be used to compute the valid assignments more efficiently (Lines 7-8), since each assignment valid for a pattern node $v$ is valid for each automorphic node $w$ too. Finally the algorithm iterates over each candidate node $n \in D_v$ to determine if it belongs to some subgraph $g$ isomorphic to $P$ (Lines 12-13). As soon as such subgraph is found, all the domains are updated (Lines 16-17) and the subgraph is checked for validity (Line 15). In particular, the ISVALID procedure compares the edge weights in $g$ against the constraints specified by the scoring function $f$, and if $g$ satisfies the condition, the nodes of the subgraph are stored in the corresponding support sets (Line 19). These nodes will contribute to the *score* of $P$.

On the other hand, if $n$ does not participate in any isomorphism, it is removed from $D_v$. As a consequence, in the following iteration, the structural constraints like the minimum degree of a node mapped to a $v \in V_P$ are enforced, to remove candidates that can no longer participate to any isomorphism of $P$ (Line 10). The algorithm terminates either when all the pattern nodes

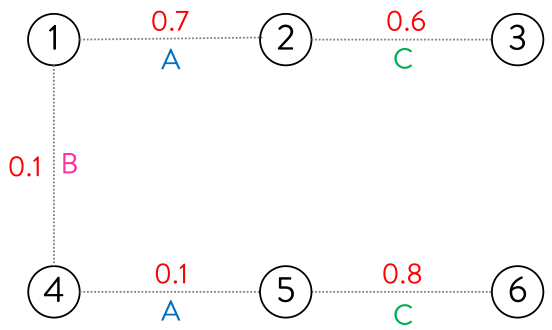


**Figure 3.2:** A weighted graph with patterns $P_1 : [v_1]{-}A{-}[v_2]$ and $P_2 : [v_1]{-}A{-}[v_2]{-}C{-}[v_3]$.

have been examined, or when the size of some domain becomes lower than $\tau$, as in this case $P$ can be neither relevant nor frequent (Line 11). In the first case, instead, the MNI support and the *score* of $P$ are calculated and returned. We refer to Section 3.7 for a discussion about desirable MNI-compatible scoring functions that we implemented in Procedure ISVALID.

Finally in Lines 13-17 of Algorithm 1, all the frequent patterns are further extended, while all the relevant patterns are included in the final set of relevant patterns $R$.

Note that, unlike the MNI support, the scoring function $f$ is not anti-monotonic. As an example consider the graph $G$ in Figure 3.2 and a scoring function $f$ that counts the number of appearances of $P$ with a large average edge weight. Using the relevance threshold $\alpha = 0.4$, the score of $P_1 : [v_1] - A - [v_2]$ is 1 because only the appearance $[1] - A - [2]$ has average edge weight above $0.4$ (i.e., $0.7$). On the other hand, the score of its extension $P_2 : [v_1] - A - [v_2] - C - [v_3]$ is 2 because both the appearances $[1] - A - [2] - C - [3]$ and $[4] - A - [5] - C - [6]$ have a large average edge weight (0.65 and 0.45 respectively).

As a consequence, also the patterns with score below $\tau$ must be expanded in order to obtain a complete solution to MULTIW-SPM. Nonetheless, Property **iii** guarantees that the score of a pattern is upper bounded by its MNI support, and therefore we can safely call Procedure PATTERNEXTENSION only for the frequent patterns (Lines 13-14).

#### 3.4.1.1 Complexity

Even though the computation of the automorphisms ($\mathcal{O}(|V_P|^{|V_P|})$) and the pruning strategy improve the expected performance of the algorithm, in the worst case it takes $C = \mathcal{O}(2^{|V|^2} \cdot |V|^{|V_P|})$ time, which is exponential in the number of nodes and the size of the patterns. In

**Algorithm 3** EXAMINESUBGRAPHMULTI

```
Input: Graph $G$:$\langle V, E, \ell, W\rangle$, pattern $P$, frequency threshold $\tau$
Output: Scores and MNI support of $P$
 1: for all $v \in V_P$ do
 2:     $D_v \leftarrow \{v' \in V | \ell(v') = \ell(v)\}$
 3:     for all $i \in 1, \dots, |W|$ do
 4:         $SUP_v[i] \leftarrow \emptyset$
 5: $\mathcal{A} \leftarrow$ automorphisms of $P$
 6: STRUCTURALCONSISTENCY($\{D_v | v \in V_P\}, P$)
 7: for all $v \in V_P$ do
 8:     if $\exists\, w = \mathcal{A}(v)\; s.t.\; D_w$ already computed then
 9:         $D_v \leftarrow D_w$
10:         continue
11:     STRUCTURALCONSISTENCY($\{D_v | v \in V_P\}, P$)
12:     if $\exists D_u\; s.t.\; |D_u| < \tau$ then return $(\{-1, \dots, -1\}, -1)$
13:     for all $n \in D_v$ do
14:         search for $g$ s.t. $g \simeq P \wedge n \in V_g \wedge n \mapsto v$
15:         if $g \neq Nil$ then
16:             $VAL \leftarrow$ ISVALID($g, W$)
17:             for all $n' \in V_g$, $v' \in V_P$ s.t. $n' \mapsto v'$ do
18:                 mark $n'$ in $D_{v'}$
19:                 for all $i \in 1, \dots, |W|$ do
20:                     if $VAL[i]$ then
21:                         $SUP_{v'}[i] \leftarrow SUP_{v'}[i] \cup \{n'\}$
22:         else
23:             remove $n$ from $D_v$
24: $\mathcal{S} \leftarrow$ RELEVANCESCORES($\{SUP_v | v \in V_P\}$)
25: $mni \leftarrow min_{v \in V_P} |D_v|$
26: return $(\mathcal{S}, mni)$
```

particular, $\mathcal{O}(2^{|V|^2})$ is the time required to compute all the patterns in $G$, and $\mathcal{O}(|V|^{|V_P|})$ is that needed to find all the isomorphisms of a pattern $P$.

### 3.4.2 Mining Multi-weighted Graphs

In the case of multiple edge weighting functions $W = \{\omega_1, \dots, \omega_m\}$, the naïve approach for solving MULTIW-SPM runs Algorithm 1 $|W|$ times, once for each set of weights. Each run requires the computation of the isomorphisms of the patterns, which is an operation computationally intense. As a consequence, this approach becomes impractical for large $m$.

The naïve approach recomputes the same patterns multiple times, incurring in a significant time overhead that can be avoided by running the algorithm only once, while keeping track of the relevant patterns for each weighting function. This strategy replaces Line 12 in Algorithm 1 with Algorithm 3, which searches for the isomorphisms of the pattern $P$, while checking their validity with respect to each $\omega_i \in W$, at the same time. Similarly to the single-weight case, we initialize each candidate domain and all the support sets for each weighting function (Lines 1-4). When an isomorphic subgraph is found, procedure ISVALID checks *in parallel* each set of edge weights against the constraints set by the scoring function and stores the results in the auxiliary array *VAL*. If the weights of $\omega_i$ satisfy the constraints, the nodes of the subgraph are stored in the corresponding sets $SUP_v[i]$ (Line 21).

Finally, all the scores of the candidate pattern $c$ are evaluated in Line 16 of Algorithm 1, and $c$ is added to the final set $R$ if only if at least one of its scores is larger than $\tau$. As a further optimization, instead of storing in memory the sets of relevant patterns for each function $\omega_i$, we maintain a binary vector of size $m$ for each relevant pattern $P$, where position $i$ is set to 1 if $P$ is relevant for $\omega_i$.

#### 3.4.2.1 Complexity

The automorphisms of each pattern $P$ are computed once ($\mathcal{O}(|V|^{|V_P|})$), while the $m$ scores are computed incrementally as new subgraphs isomorphic to $P$ are found ($\mathcal{O}(|V|^{|V_P|} \cdot |V_P| \cdot m)$). Even though the search of isomorphisms stops as soon as all the $m$ scores exceed the threshold $\tau$, in the worst case we must find all of them. The complexity is therefore $\mathcal{O}(2^{|V|^2} \cdot (|V|^{|V_P|} + m \cdot |V_P| \cdot |V|^{|V_P|})$, which can be approximated to $\mathcal{O}(2^{|V|^2} \cdot m \cdot |V_P| \cdot |V|^{|V_P|})$. If we can assume that the size of the pattern $|V_P|$ is negligible, the complexity becomes $\mathcal{O}(2^{|V|^2} \cdot m \cdot |V|^{|V_P|})$.

## 3.5 ReSuM Approximate

Our exact algorithm ReSuM incurs a significant memory overhead when the number of weighting functions is in the order of thousands, which, for example, is the case for recommender systems for big retailers (e.g., Amazon). For such applications, we devise a more conservative approximate solution, called ReSuM *approximate*, that significantly reduces the memory consumption by taking advantage of the similarities between the weighting functions $\omega_1, \ldots, \omega_m \in W$.

The ReSuM *approximate* algorithm generates $k \ll m$ representative functions $\omega_j^*$, by clustering and aggregating the weighting functions $\omega_i$. Then, it runs Algorithm 3 to compute $k$ sets of

**Algorithm 4** GENERATEREPRESENTATIVEFUNCTIONS

```
Input: Graph $G : \langle V, E, \ell, W \rangle$, number of buckets $b$, number of clusters $k$
Output: Set of representative functions $W^*$
1: $\mathcal{F} \leftarrow$ CREATEFEATUREVECTORS$(E, W, b)$
2: $\mathcal{C} \leftarrow$ COMPUTECLUSTERING$(\mathcal{F}, k)$
3: $W^* \leftarrow$ GENERATEMAXWEIGHTVECTORS$(\mathcal{C}, W)$
4: return $W^*$
```

relevant patterns $R_1^*, \ldots, R_k^*$, which are used to build $m$ sets of relevant patterns $A_1, \ldots, A_m$, one for each weighting function $\omega_i$. Different clustering can produce different sets $R_1^*, \ldots, R_k^*$, and therefore the sets $A_1, \ldots, A_m$ are in general an approximation of the real sets $R_1, \ldots, R_m$. The quality of the approximation depends on how the representative functions $\omega_i^*$ are generated. With our implementation, we aim at minimizing the difference between the approximate and the exact sets of patterns.

### 3.5.1 Generation of the Representative Functions

The generation of the representative functions is shown in Algorithm 4 and consists of three steps. First, we create a feature vector for each weighting function $\omega_i \in W$ (Line 1). Secondly, we cluster the feature vectors into $k$ groups according to their similarity (Line 2). Finally, the set of $k$ representative functions $W^* = \{\omega_1^*, \ldots, \omega_k^*\}$ is returned (Lines 3-4).

#### 3.5.1.1 Creation of the Feature Vectors

In the first step, we construct a feature vector $r_i$ for each $\omega_i$, which is used in the second step to determine the similarities between the weighting functions. Since our final goal is to compute a set of patterns $A_i$ for each $\omega_i$ that is as close as possible to the exact set $R_i$, a natural choice is to use the edge weights as features. Given an ordering of the edges, the feature vector $\mathbf{r}_i$ of the function $\omega_i$ is a vector of size $|E|$, where $\mathbf{r}_i[x]$ is the weight assigned by $\omega_i$ to the edge in the $x^{th}$ position. We call this approach *full-vector strategy*.

Although similar edge weights lead, with high probability, to similar sets of relevant patterns, the effectiveness and the efficacy of the full-vector strategy decrease as the size of the graph increases. In fact, the high dimensionality of the vectors complicates the detection of functions with similar properties, as a consequence of the curse of dimensionality phenomenon [SEK04].

**Algorithm 5** CREATEBUCKETFEATUREVECTORS

```
1: function CREATEBUCKETFEATUREVECTORS($E, W, b$)
2:     for all $l \in \Sigma_E$ do
3:         $BucketList_l \leftarrow$ COMPUTEBUCKETLIMITS($E_l, W, b$)
4:         for all $\omega_i \in W$ do
5:             $\mathbf{r}_i^l \leftarrow$ FILLBUCKETS($E_l, \omega_i, BucketList_l$)
6:     for all $\omega_i \in W$ do
7:         $\mathbf{r}_i \leftarrow$ CONCATE($\{\mathbf{r}_i^l | l \in \Sigma_E\}$)
8:     return $\{\mathbf{r}_1, \ldots, \mathbf{r}_{|W|}\}$
```

Thus, we propose also a more efficient approach called *bucket-based strategy*, which overcomes the problem of high dimensionality by considering the edge labels as features, rather than the edges. The underlying idea is that, in real scenarios, a preference for an edge is highly correlated with the preference for the label of that edge. This strategy is implemented in Procedure CREATEBUCKETFEATUREVECTORS (Algorithm 5), which takes the set of weighting functions $W$ and the number of buckets $b$, and generates a set of feature vectors $\mathbf{r}_1, \ldots, \mathbf{r}_m$ each of size $|\Sigma_E| \cdot b$, where $\Sigma_E$ indicates the set of distinct edge labels. Each vector $\mathbf{r}_i$ is the concatenation of $|\Sigma_E|$ *summaries* of the weights of $\omega_i$, one for each edge label, and $b$ is the resolution of each summary.

The summary for a label $l$ is obtained by splitting the range of weights $[0, 1]$ into $b$ sub-ranges (buckets) (Line 3), e.g., $[0, x_1)$, $[x_1, x_2)$, and $[x_2, 1.0]$ for $b = 3$. Then Procedure FILLBUCKETS (Line 5) counts, for each sub-range, how many times $\omega_i$ assigns a weight within that sub-range to an edge with label $l$. Note that, in the degenerate case of $b = 1$, the vector $\mathbf{r}_i$ simply keeps, for each label, the number of edges with that label and whose weight is greater than 0.

The *bucketization* of a label $l$ is performed by Procedure COMPUTEBUCKETLIMITS (Line 3) following the equi-depth paradigm [GK01], which assigns the input values to buckets, while trying to balance the number of elements in each bucket. Thus, we consider all the weights of all the weighting functions to edges with label $l$, and split the range $[0, 1]$ into $b$ depth-balanced intervals.

For example, given $b = 2$, the label ordering $A \,|\, C$, and the two weighting functions $\omega_1$ and $\omega_2$ in Figure 3.3, we obtain the vectors $\mathbf{r}_1 = [1, 3, 2, 0]$ and $\mathbf{r}_2 = [3, 1, 0, 2]$. As such, the buckets of $A$ are the ranges of values $[0, 0.3]$ and $[0.3, 1]$, and those of $C$ the ranges $[0, 0.5]$ and $[0.5, 1]$.

Note that the bucket-based strategy allows us to decide the size of the feature vectors apriori, and tune the parameter $b$ to improve the accuracy of the clustering.

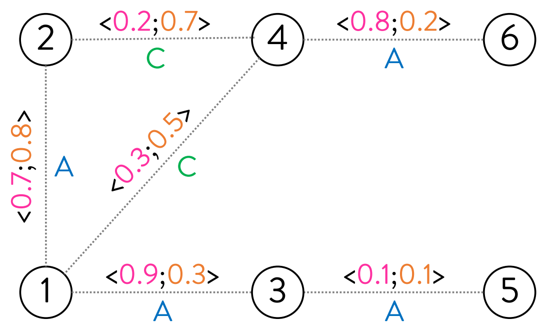


**Figure 3.3:** Graph with two weights $< \omega_1, \omega_2 >$ on each edge.

#### 3.5.1.2 Identification of Similar Functions

Procedure COMPUTECLUSTERING (Algorithm 4, Line 2) implements the Lloyd's clustering algorithm [Llo82], which identifies groups of similar $\omega_i$ by computing the pairwise cosine similarity between the feature vectors $\mathbf{r}_1, \ldots, \mathbf{r}_m \in \mathcal{F}$.

The algorithm can be initialized either providing $k$ random seeds among all the vectors in $\mathcal{F}$, or by selecting the $k$ most diverse feature vectors. Note that finding the most diverse vectors may increase the running time of the algorithm, but also allows the discovery of better separated clusters. In addition, the algorithm can be executed either until convergence or in iterative steps. In the first case it finds $k$ clusters, while in the second case it runs multiple times with $k$ ranging from 2 to a maximum value $k_{max}$, and then returns the clustering with largest silhouette coefficient.

#### 3.5.1.3 Aggregation of Similar Functions

Given the set of clusters $\mathcal{C}$, Procedure GENERATEMAXWEIGHTVECTORS (Algorithm 4, Line 3) generates a representative function $\omega_j^*$ for each cluster $C_j$. Different choices of $\omega_j^*$ can lead to different sets of patterns $R_j^*$, which can contain patterns not relevant for some $\omega_i \in C_j$, as well as missing out patterns relevant for some other $\omega_l \in C_j$. As stated in the following proposition, we resort to take the *maximum* among the weights to prevent missing any relevant pattern:

**Proposition 3.5.** *Given a cluster $C_i$, and a MNI-compatible scoring function $f$, a* complete *set of relevant patterns for $C_i$ can be mined using the representative function $\omega_i^*$ defined as $\forall e \in E\,,\, \omega_i^*(e) = max_{\omega_j \in C_i} \omega_j(e)$.*

*Proof.* By definition, only the subgraphs that satisfy the constraints on the weights imposed by the scoring function $f$ can contribute to the score of the corresponding pattern. The larger the weights of a subgraph, the higher is the chance that it fulfills those constraints. Since the function $\omega_i^*$ assigns to each edge $e \in E$ the largest weight among those of the weighting functions in the cluster $C_i$, i.e., $\forall \omega_j \in C_i\,, \omega_i^*(e) \geq \omega_j(e)$, if a subgraph satisfies the constraints with respect to some $\omega_j \in C_i$, it must satisfy them with respect to $\omega_i^*$. It follows that $\forall \omega_j \in C_i\,,\, f(P, \omega_i^*) \geq f(P, \omega_j)$, and therefore if a pattern is relevant for some $\omega_j \in C_i$, it is also relevant for $\omega_i^*$. Thus, the set of patterns mined is complete. □

Given the sets of relevant patterns $R_1^*, \ldots, R_k^*$ discovered by Algorithm 1 using the representative functions $\omega_1^*, \ldots, \omega_k^*$, we create a pattern set $A_i$ for each function $\omega_i$ using the patterns in the set $R_j^*$ for $j \leq k\,.\,\omega_i \in C_j$, i.e., each function $\omega_i$ receives the set of relevant patterns of the cluster to which it belongs.

### 3.5.2 Complexity

The generation of the $k$ representative functions for the $m$ weighting functions requires the creation of the feature vectors ($\mathcal{O}(m \cdot |E|)$), the identification of similar functions ($\mathcal{O}(I \cdot k \cdot m \cdot b \cdot |\Sigma_E|)$, where $I$ is the number of iteration of $k$-means, and $b \cdot |\Sigma_E|$ is the size of the feature vectors), and the computation of the maximal weights for each edge and for each cluster of functions ($\mathcal{O}(m \cdot |E|)$). Then, the $k$ sets of relevant patterns are found running the algorithm described in Section 3.4 ($\mathcal{O}(2^{|V|^2} \cdot k \cdot |V|^{|V_P|})$). Since $k$, $I$ and $b$ are negligible, and $|\Sigma_E| = |E|$ in the worst case, the complexity of ReSuM *approximate* reduces to $\mathcal{O}(m \cdot |E| + 2^{|V|^2} \cdot k \cdot |V|^{|V_P|})$.

### 3.5.3 Quality Measures for ReSuM Approximate

ReSuM *approximate* reduces the problem of pattern mining in graphs with $m$ weights on each edge to finding $k$ sets of relevant patterns $R_j^*$, with $k \ll m$. The quality $Q$ of the solution can be measured in different ways, according to the requirements of the user or the application. In the literature, the most common quality measure is the accuracy, which is defined in terms of precision and recall with respect to a given ground-truth. In our case, the sets $R_i$ constitute the ground-truth, and the accuracy of the sets $A_i$ with respect to the sets $R_i$ is measured only in terms of the precision, since Proposition 3.5 ensures a total recall. Specifically, the quality of the solution $\{A_1, \ldots A_m\}$ is the average Jaccard similarity with the corresponding set in $\{R_1, \ldots, R_m\}$:

$$Q = \frac{1}{m} \sum_{i=1}^{m} |R_i \cap A_i| / |R_i| \tag{3.1}$$

The quality $Q$ can be measured also in terms of the average distance between the patterns in the sets $R_i$ and those in the sets $A_i$. We calculate the distance between two patterns $P_1$ and $P_2$ as their normalized Levenshtein distance $\text{NORMLEV}(P_1, P_2)$, i.e., the minimum number of edges that should be added or removed from the first to transform it into the second, normalized to obtain a value in the range $[0, 1]$. Then, we define the distance between a pattern $P \in A_i$ and the set of patterns $R_i$, as the normalized Levenshtein distance between $P$ and the closest pattern $P_j \in R_i$, i.e., $\text{NORMLEV}(P, R_i) = \min_{P_j \in R_i} \text{NORMLEV}(P, P_j)$, and the distance $\Delta$ between the two sets $A_i$ and $R_i$ as the average normalized Levenshtein distance, i.e. $\Delta(A_i, R_i) = \frac{1}{|A_i|} \sum_{P \in A_i} \text{NORMLEV}(P, R_i)$. Finally, the quality of the solution $\{A_1, \ldots A_m\}$ is the inverse of the average of the distances $\Delta(A_1, R_1), \ldots, \Delta(A_m, R_m)$:

$$Q = 1 - \frac{1}{m} \sum_{i=1}^{m} \Delta(A_i, R_i) \tag{3.2}$$

The average of the distances $\Delta(A_1, R_1), \ldots, \Delta(A_m, R_m)$ measures the average number of operations required to transform a pattern in $A_i$ to a pattern in $R_i$, and thus, according to Equation 3.2, $A_i$ is a good solution for $\omega_i$ if the patterns in $A_i$ have structure and labels similar to the patterns in $R_i$. We recall that our method is complete, and therefore no relevant pattern is missing. However, RESUM *approximate* may return spurious patterns, which are patterns not relevant for any function in the cluster. Computing the distance between the two pattern sets allows us to understand how much a spurious pattern, on average, differs from the patterns that are actually relevant for some weighting function in the cluster.

## 3.6 Distributed Score-based Pattern Mining

Distributed graph processing systems have been introduced to overcome the challenges of dealing with very large graphs [TFS+15, MAB+10]. Those systems scale by distributing the computation among multiple machines communicating with each other. Moreover, they are usually designed such that all the details related to the distribution, the message-passing, and the synchronization, are hidden behind simple API that allow non-expert users to implement efficient and scalable algorithms [TFS+15, MAB+10].

In the following, we show to apply our score-based pattern mining framework in the distributed settings by designing a distributed version of ReSuM. We implemented our algorithms on top of Arabesque [TFS+15], a framework for distributed graph mining that differs from other existing platforms (e.g., Pregel [MAB+10]) in the programming paradigm adopted. In fact, Arabesque follows the Bulk Synchronous Parallel model [Val90], but centers the computation around the task of searching for embeddings, meaning that each worker is delegated to examine and expand a different set of embeddings in the graph. This programming model is specifically designed for the implementation of graph pattern mining algorithms, instead of generic vertex-centric computations (like those supported by Pregel [MAB+10]).

### 3.6.1 ReSuM Distributed

Generic distributed pattern mining frameworks like Arabesque base the examination of the pattern search space only on the frequency of the patterns, and therefore two important extensions are required to implement score-based pattern mining in such framework: an appropriate data-structure for the storage of the embeddings that can keep track of their weights (especially for the case of multiple weighting functions), and the implementation of aggregation functions for the computation of the MNI-compatible scoring functions. In particular, in the case of multiple weights, it is important to aggregate the support sets associated to each weighting function.

The computation proceeds via a sequence of supersteps in the Bulk Synchronous Parallel model, where a master coordinates and collects the results from a cluster of workers. Given an initial set of embeddings in the graph, the task of the workers is to identify all the possible expansions of each of them, i.e., embeddings with an additional edge, which will be used to compute the frequency of the corresponding patterns. In the first step of the computation, the initial set contains only a special *undefined* embedding, whose set of expansions is the edge set of the graph. This set is collected by the master as input for the next step of computation. In each of the following supersteps, the master broadcasts the set of embeddings received in the previous superstep, while the workers expand those corresponding to frequent patterns and give back the new expanded embeddings to the master. The computation halts when the new set of embeddings is empty.

**Algorithm 6** DISTRIBUTEDRELEVANTPATTERNMINING

```
Input: Set of initial embeddings $I$, threshold $\tau$
Output: Set of expanded embeddings $F$
Output: Set of relevant patterns $R$
1: for all $e \in I$ do
2:     if AGGREGATIONFILTER($e$) then
3:         $Cand \leftarrow$ EMBEDDINGEXPANSION($e$)
4:         for all $e' \in Cand$ do
5:             if ISCANONICAL($e'$) then
6:                 PROCESS($e'$)
7:                 $F \leftarrow F \cup \{e'\}$
8: PATTERNAGGREGATION($F$)

9: function AGGREGATIONFILTER($e$)
10:     $D_{v_1}, \ldots, D_{v_n} \leftarrow$ GETDOMAINS($e$)
11:     $sup_{v_1}, \ldots, sup_{v_n} \leftarrow$ GETSUPPORTSETS($e$)
12:     $mni \leftarrow min_{v_i}|D_{v_i}|$
13:     $score \leftarrow$ RELEVANCESCORE($\{sup_v | v \in V_{P_e}\}$)
14:     if $score \geq \tau$ then
15:         $R \leftarrow R \cup \{P_e\}$
16:     return $mni \geq \tau$

17: function PROCESS($e$)
18:     MAP($P_e$, $\{D_{v_i}\}_{v_i \in V_{P_e}}$, $\{sup_{v_i}\}_{v_i \in V_{P_e}}$)
19:     REDUCE($P$, $\{D^1_{v_i}, \ldots, D^m_{v_i}\}_{v_i \in V_P}$, $\{sup^1_{v_i}, \ldots, sup^m_{v_i}\}_{v_i \in V_P}$)
```

Upon receiving the embeddings, the workers use Round Robin on large blocks of embeddings to partition them. A different subset of embeddings is thus assigned to each worker to be filtered and processed. Since the number of embeddings in a graph increases exponentially with the graph and the pattern size, the workers use a special data structure called Overapproximating Directed Acyclic Graph (ODAG) to store them in a compact way. ODAGs trade space for time by over-approximating the set of embeddings they want to store, hence entailing additional work to extract only the actual embeddings from them and avoid the generation of spurious patterns.

Once restored the valid embeddings $I$, the workers run the procedures shown in Algorithm 6. When processing an embedding $e$, the worker must first determine if it corresponds to a frequent pattern, since embeddings of infrequent patterns will not be expanded. The frequency values are computed via a MapReduce job (Line 8) where the mappers send the ODAGs of the same pattern to the reducer responsible for that pattern, and the reducers aggregate the domains and the support sets contained in the ODAGs received. The aggregation of the domains (support sets) consists in computing the union of the domains $D_v$ (support sets $sup_v$) of each vertex $v$ of the pattern $P$. As described in Section 3.4, the domains are used to compute the MNI support of $P_e$, while the support sets to compute its *score*. In the initialization of the support sets of $P_e$, the mapper runs procedure ISVALID to check whether the weights of $e$ satisfy the constraints specified by the scoring function or not. If they pass the validity test, the nodes of $e$ are stored in the support sets; otherwise the sets are left empty.

At the end of the aggregation, if all the domains have size greater than $\tau$ (Line 16), the pattern is frequent and thus its embeddings are further processed (Line 2). Similarly, if all the size of all the support sets exceeds $\tau$, the pattern is inserted in the relevant pattern set $R$ that will be output to the underlying distributed file system (Line 14). We recall that the MNI support $mni$ is the minimum among the sizes of the domains (Line 12), while the evaluation of the *score* depends on the scoring function chosen (Line 13). To speed up the computation, the reducers actually stop merging the values in the domains/ support sets that have already exceeded the threshold, hence terminating the ODAG aggregation when all the domains/ support sets contain enough values.

All the embeddings retained are expanded by Procedure EMBEDDINGEXPANSION(Line 3), which adds one additional edge in all the possible positions. Since the workers have access to a local copy of the graph, they do not need to communicate and exchange information in this phase. Nonetheless, the same embedding can be generated by multiple workers as a result of processing the same set of edges in different orders. To avoid duplicate embeddings, one of the ordering is elected as canonical (Line 5), so that all the others can be safely pruned. All the extensions of canonical embeddings are stored in a set $F$ that will be sent to the master at the end of the superstep (Line 7).

To reduce the amount of messages that will be sent through the network, the workers perform a MapReduce job locally (Line 6) to aggregate the extensions related to the same pattern, hence building an ODAG for that pattern. In the Map phase, the domains $D_v$ and the support sets $sup_v$ of the pattern $P_e$ of $e$ are initialized with the ids of the nodes of $e$ (Line 18). In the reduce phase, the domains and the support sets related to the same pattern $P$ are aggregated in the same way as when running Procedure PATTERNAGGREGATION. To identify the canonical pattern $P_e$ of $e$, the workers use a technique called two-level pattern aggregation. In the first level, they create a so-called *quick pattern* by scanning all the edges of the embedding and extracting the corresponding labels. In the second level, they compute the canonical pattern of each quick pattern and reorder the list of domains and support sets of the quick pattern according to the canonical vertex ordering. When computing the canonical patterns, the workers also search for automorphisms that will be exploited to insert all the elements in a domain $D_v$ to the domains of the symmetric counterparts of $v$. Note that the two-level pattern aggregation technique reduces the complexity of Procedure PROCESS, as the graph isomorphism tests required to aggregate the embeddings are performed for the smaller number of quick patterns rather than the larger number of embeddings.

### 3.6.2 Time Complexity

At the beginning of each superstep, each worker must first extract the valid embeddings from the ODAGs, then generate the canonical extensions of the embeddings associated to frequent patterns, and finally produce the ODAGs for the next step. The cost of the first operation is upper bounded by the number of paths contained in the ODAGs, i.e., $\mathcal{O}(|V|^{|V_P|})$. The cost of the second operation is the sum of the cost of computing the frequency of the pattern associated to each ODAG ($\mathcal{O}(|V_P| \cdot |V|^2) = \mathcal{O}(|V|^2)$) and that of creating the canonical extensions of each embedding retained ($\mathcal{O}((|V_P| + 1)^{(|V_P|+1)} \cdot |V|^{(|V_P|+1)})$). The last operation consists in performing the two-level pattern aggregation to generate a single ODAG per canonical pattern ($\mathcal{O}((|V_P| + 1) \cdot |V|^{(|V_P|+1)})$ to generate the quick patterns, and $\mathcal{O}(|V|^{(|V_P|+1)})$ to generate the canonical patterns). The total time required to perform each superstep is therefore $\mathcal{O}(|V_P|^{(|V_P|+1)} \cdot |V|^{(|V_P|+1)})$.

### 3.6.3 Space Complexity

Each worker has access to a copy of the input graph. In addition, at the beginning of each superstep, it receives every ODAG produced in the previous step, and thus must keep in memory $|V_P|$ vectors of integers. The maximum number of integers to store for all the ODAGs is $|V_P| \cdot |V|^2$.

### 3.6.4 Machine-to-machine Communications

At the beginning of each superstep, the master sends all the ODAGs of the previous step to all the workers. In the worst case, every embedding is associated to a different pattern, and therefore the number of these messages is upper bounded by $num\_workers \cdot |V|^{|V_P|}$. At the end of each superstep, a map-reduce job is executed to aggregate the ODAGs associated to the same canonical pattern, that is $\mathcal{O}(|V|^{(|V_P|+1)})$, as all the ODAGs of the same patterns must be sent to the reducer responsible for that pattern and then all the aggregated ODAGs are sent to the master. The total communication cost is $\mathcal{O}(|V|^{(|V_P|+1)})$.

## 3.7 Computation of the Pattern Scores

We demonstrate the flexibility of our framework by proposing four different MNI-compatible scoring functions that can be implemented in Procedure ISVALID (Algorithm 2) and in Procedure RELEVANCESCORE (Algorithm 3 and Algorithm 6). They are called *ALL*, *ANY*, *SUM* and *AVG*. We defined these functions because of their intuitive semantics and their suitability for various scenarios that may pose different requirements or provide a different interpretation of the edge weights. Moreover, since they are *MNI-compatible*, they are anti-monotonic, hence allowing efficient implementations.

The *ALL*, *ANY*, *SUM* and *AVG scores* differ in the choice of which subgraphs they retain in the support set of the patterns, and in how they aggregate the edge weights of such subgraphs. In particular, *ALL*, *ANY*, and *SUM* rely on an additional system-dependent parameter $\alpha$ that is used to select the subgraphs that contribute to the *score*, while *AVG* is parameter-free. In the following we provide a formal definition and analysis of those functions.

### 3.7.1 The ALL Score

The *ALL score* considers only the subgraphs whose edge weights are larger than the relevance threshold $\alpha$ as valid appearances of a pattern $P$. Specifically, the *ALL score* of $P$ is its MNI support computed over the restricted set of appearances $S'_G(P) = \{g \mid g = \langle V_g, E_g, \ell, \omega\rangle \wedge g \in S_G(P) \wedge \forall e \in E_g \, , \, \omega(e) > \alpha\}$, that is, $f_{ALL}(P, G) = min_{v_P \in V_P} \left|\mathcal{N}(G, v_P)\restriction_{S'_G(P)}\right|$, where $\mathcal{N}(G, v_P)\restriction_{S'_G(P)} = \{v | v \in V \wedge \exists g \in S'_G(P) \, . \, \phi^P_g(v) = v_P\}$ is the restriction of $\mathcal{N}(G, v_P)$ to the subset $S'_G(P) \subseteq S_G(P)$.

In graphs like protein-to-protein interaction networks, this *score* retrieves patterns characterized by an overall confidence greater than a certain value.

### 3.7.2 The ANY Score

The *ANY score* takes into account only the appearances of a pattern having at least one edge with weight above the relevance threshold $\alpha$. Hence, the *ANY score* of $P$ is the MNI support of $P$ over the set of appearances $S'_G(P) = \{g \,|\, g = \langle V_g, E_g, \ell, \omega\rangle \wedge g \in S_G(P) \wedge \exists e \in E_g \,.\, \omega(e) > \alpha\}$, i.e., $f_{ANY}(P, G) = min_{v_P \in V_P} \left|\mathcal{N}(G, v_P)\restriction_{S'_G(P)}\right|$.

This *score* is suitable especially for the cases in which only partial weights are available (e.g., product reviews for some product), to find patterns that are overall interesting (e.g., the entire transaction comprising the product), as well as super-patterns around relevant core structures.

By definition, the *ANY score* of $P$ is always equal or larger than its *ALL score*, as any appearance of $P$ considered by $f_{ALL}$ is considered also by $f_{ANY}$, while in general, the opposite is not true. For example, given the graph in Figure 3.3 and the relevance threshold $\alpha = 0.4$, the subgraph $g$ : [1]–A–[2]–C–[4] does not contribute to the *ALL score* of $P$ : [$v_1$]–A–[$v_2$]–C–[$v_3$], but contributes to its *ANY score*.

### 3.7.3 The SUM Score

For the *SUM score* of $P$, a subgraph $g$ contributes if the sum of its weights is larger than the relevance threshold $\alpha$. The restricted support set obtained in this way is $S'_G(P) = \{g \,|\, g = \langle V_g, E_g, \ell, \omega\rangle \wedge g \in S_G(P) \wedge \sum_{e \in E_g} \omega(e) > \alpha\}$. The MNI support over this set is the *SUM score* of $P$: $f_{SUM}(P, G) = min_{v_P \in V_P} \left|\mathcal{N}(G, v_P)\restriction_{S'_G(P)}\right|$.

This *score* accounts for the overall pattern weight in scenarios like money transactions, where it is beneficial to sum each single contribution in order to judge the complete value of a structure.

Note that if an appearance of $P$ has some weight greater than $\alpha$, then the sum of all its weights is at least $\alpha$, and therefore $f_{SUM}(P, G) \geq f_{ANY}(P, G)$. For example, all the appearances considered by *ANY* in computing the score of $P$ : [$v_1$]–A–[$v_2$]–A–[$v_3$] for $\alpha$=0.4 in Figure 3.3 are considered also by *SUM*, whereas the subgraph $g$ : [3]–A–[4]–A–[8] contributes to the *SUM score* only.

### 3.7.4 The AVG Score

In contrast to the previous scoring functions, the *AVG score* is not defined in terms of the minimum cardinality among the node sets of the pattern, but in terms of the relative weights of its appearances. Previous works have defined the *weighted support* ($\mathit{WSUP}$) of a pattern $P$ as the sum of the weights of the embeddings of $P$. For example, WIGM [YSLD12] proposes a measure called *normalized weighted support* ($\mathit{NWSUP}$), which is the weighted support of $P$ divided by its size $|E_P|$, i.e., $\mathit{NWSUP}(G, P) = \mathit{WSUP}(G, P)/|E_P|$. Nevertheless, this scoring function is not MNI-compatible and our focus in on MNI-compatible scoring functions. To guarantee the anti-monotonicity and be consistent with the other MNI-compatible scoring functions, we compute $\mathit{WSUP}(G, P)$ by first retaining, for each edge set $\mathcal{E}(\mathcal{G}, e_P)$ with $e_P \in E_P$, the set $\mathcal{E}(\mathcal{G}, e_P)\restriction_\mu$ of $\mu$ edges with largest weight, and then summing up those weights, i.e., $\mathit{WSUP}(G, P) = \sum_{e_P \in E_P} \sum_{e \in \mathcal{E}(\mathcal{G}, e_P)\restriction_\mu} \omega(e)$. By setting $\mu$ to be the MNI support of $P$ we guarantee that the *AVG score* is bounded by the MNI support, as stated in the following proposition:

**Proposition 3.6.** *Given a graph $G{:}\langle V, E, \ell, \omega\rangle$, a pattern $P$, and an edge $e \in E$, it holds that $f_{\mathit{AVG}}(P \diamond e, G) \leq \mathit{MNI}(P, G)$, where $P \diamond e$ is an extension of $P$ with $E_{P\diamond e} = E_P \cup \{e\}$.*

*Proof.* Since the MNI support is anti-monotonic [BN08], it holds that $\mathit{MNI}(P \diamond e, G) \leq \mathit{MNI}(P, G)$. By definition, the pattern $P \diamond e$ has the maximum normalized weight $f^*_{\mathit{AVG}}(P \diamond e, G)$ when all the edges in $\mathcal{E}(\mathcal{G}, e)\restriction_\mu$ have weight 1, and hence each subgraph contributes with a total weight of $(|E_P|+1)$. In this case, $f^*_{\mathit{AVG}}(P\diamond e, G) = \mathit{MNI}(P\diamond e, G)\cdot(|E_P|+1)/(|E_P|+1)$, and thus $f_{\mathit{AVG}}(P \diamond e, G) \leq f^*_{\mathit{AVG}}(P \diamond e, G) = \mathit{MNI}(P \diamond e, G) \leq \mathit{MNI}(P, G)$. □

According to Proposition 3.6, although *AVG* is not anti-monotonic, the *AVG score* of a pattern is at least bounded by the MNI support of its sub-patterns, making it MNI-compatible and allowing an early pruning of the pattern search space when the MNI support of a pattern $P$ is lower than $\tau$. On the other hand, $f_{\mathit{AVG}}(P\diamond e, G)$ can be higher than $f_{\mathit{AVG}}(P, G)$ even though the frequency of $P \diamond e$ is lower, because the weights of the edges in $\mathcal{E}(\mathcal{G}, e)\restriction_\mu$ can compensate for the lower frequency. For example, the *AVG score* of $P$ : [$v_1$]–C–[$v_2$] in the graph $G$ in Figure 3.3 is 0.6, because $MNI(P, G) = 1$ and $\mathcal{E}(\mathcal{G}, C)\restriction_1 = \{(1, 4)\}$. Instead, the *AVG score* of $P$ : [$v_1$]–C–[$v_2$]–B–[$v_3$]–A–[$v_4$] is 0.8, because $\mathcal{E}(\mathcal{G}, C)\restriction_1 = \{(1, 4)\}$, $\mathcal{E}(\mathcal{G}, B)\restriction_1 = \{(1, 3)\}$, and $\mathcal{E}(\mathcal{G}, A)\restriction_1 = \{(3, 5)\}$.

| | | | | *degree* | *label frequency* | | |
|---|---|---|---|---|---|---|---|
| *dataset* | $\lvert V\rvert$ | $\lvert E\rvert$ | $\lvert\Sigma\rvert$ | min/avg/max | min/med/avg/max | $\tau$ | $\alpha$ |
| FREEBASE-T | 7.2k | 10k | 40 | 1/2.8/504 | 3/70/251.3/2.8k | 90 | .05 |
| FREEBASE-C | 16.7k | 26k | 77 | 1/3.2/1082 | 1/66/348.5/4.8k | 155 | .05 |
| AMAZON | 163k | 296k | 4<br>1710 | 1/3.6/1072 | 2k/12k/30k/113k<br>1/1/95/142k | 130 | .0001 |
| CITESEER | 2.1k | 3.6k | 21 | 1/3.5/99 | 15/55/174.7/988 | 95 | .05 |
| FREEBASE-O | 1.9M | 2.4M | 19294 | 1/2.4/46k | 1/1/103/237k | 6000 | .05 |
| SHOP-S | 11k | 12k | 80<br>24 | 1/3/35 | 1/60/161/3k<br>1/100/467/2.8k | 76 | .05 |
| SHOP-M | 163k | 296k | 81<br>24 | 1/3/129 | 3/606/1.6k/30k<br>6/1k/4.6k/28k | 759 | .05 |
| SHOP-L | 1.1M | 1.2M | 81<br>24 | 1/3/583 | 5/5.9k/16k/305k<br>60/10k/46k/280k | 7580 | .05 |
| SHOP-XL | 11M | 13M | 81<br>24 | 1/3/3868 | 115/60k/160k/3M<br>600/100k/467k/2.8M | 76124 | .05 |

**Table 3.1:** Real (top) and synthetic (bottom) datasets with default $\tau, \alpha$ parameters.

### 3.7.5 Implementation in the ReSuM Framework

The *ALL*, *ANY*, and *SUM* scores are implemented in Procedure isValid by checking the edge weights of the assignment $g$ against the constraint in the corresponding definition of $S'_G(P)$, and returning **true** if the constraint is fulfilled. In this way, each support set $sup_v$ (and $SUP_v$ for the multi-weighted case) for $v \in V_P$ will be equal to $\mathcal{N}(G, v)\restriction_{S'_G(P)}$. Therefore, Procedure relevanceScore actually computes the MNI support over the support set $S'_G(P)$.

On the other hand, to implement the *AVG score*, Procedure isValid returns always **true**, while Procedure relevanceScore calculates the normalized sum of the top-$k$ edge weights of every pattern edge, where $k = min_{v\in V_P}|D_v|$.

## 3.8 Experimental Evaluation

We first compare the scalability of our exact algorithm with the performance of our approximate algorithm. The results demonstrate that ReSuM *approximate* allows faster response time, yet retaining good accuracy in terms of the patterns returned. We then study the behavior of ReSuM *distributed* under different settings to identify in which cases we can benefit more from the distribution, as well as understanding when the overhead of a distributed system may lead to performances worse than those of single-machine algorithms [MIM15].

### 3.8.1 Datasets

We run experiments on both real and synthetic datasets of different sizes, and in particular, we used five real networks and four randomly generated graphs. All the datasets are listed in Table 3.1 together with their characteristics, i.e., the number of vertices $|N|$, edges $|E|$, and labels $|\Sigma|$; the minimum, average, and maximum node degree; and the minimum, median, average, and maximum edge label frequency.

For the AMAZON and the synthetic datasets we report statistics for both edge (top) and node labels (bottom). We also report the default frequency ($\tau$) and relevance ($\alpha$) values used in the experiments (unless otherwise stated). Experiments on the quality of RESUM and RESUM *approximate*, and on their scalability, were conducted on the first four real datasets; the scalability of RESUM and RESUM *distributed* was tested on the last two real datasets and the synthetic datasets.

• FREEBASE-T and FREEBASE-C are directed subgraphs extracted from the knowledge graph FreeBase [1], which is a database collecting structured information about real-world entities like people, places and things for various topics. We obtained the two samples by restricting the graph to the topic *travel* and *computer* respectively, and then we kept only the largest weakly connected component in the restriction.

• AMAZON[2] [HM16] is a directed graph representing items, purchases, and user ratings. We considered the subgraph of electronic products, in which every node represents a product, a category, or a brand, and a link represents items bought together, bought in subsequent transactions, or viewed on the website one after the other. Weights represent individual user review scores (from 1 to 5), and we considered only users with more than 100 reviews. Given the sparsity of the weights, we used Personalized PageRank to spread the user preferences to products other than those they rated, as it is a standard technique for recommendations [Agg16]. In this way we obtained weights not only for the items reviewed, but also for the most related items. Each edge weight is actually computed as the average between the PageRank value of its endpoint nodes.

[1] `developers.google.com/Freebase/data`
[2] `jmcauley.ucsd.edu/data/amazon/`

• CITESEER [EASK14], is a graph representing Computer Science publications and citations between them. The labels on the edges indicate the area in which the two papers were published (e.g., a database conference).

• FREEBASE-O is a undirected, node-labeled subgraph extracted from FreeBase and restricted to the topics *organizations*, *business*, *finance*, and *government*. The node labels refer to the types of nodes, obtained by following "*instance of*" edges.

• SHOP-S, SHOP-M, SHOP-L, and SHOP-XL are synthetic graphs generated using the *gMark* framework [BBC$^+$17], which creates labeled graphs with a user-defined schema that specifies constraints on the number of nodes and labels, the proportions of nodes and edges per label, and the degree distribution. We used the *shop.xml* schema provided by the framework, which encodes an online shop network consisting of sellers, users and products, according to the specifications in the WatDiv default schema [AHÖD14]. This schema contains 24 node labels, 82 edge labels, default probabilities for each label, and specifies a different degree distribution (Uniform, Gaussian, or Zipfian) for each combination of node and edge labels allowed.

### 3.8.2 Experimental Setup

RESUM is implemented in Java 1.8 on top of the constraint satisfaction problem presented in GRAMI [EASK14] whose code was kindly provided by the authors[3]. The code of our implementation and all the datasets we used are publicly available[4]. We also compare with a frequent pattern mining approach (FREQ) based on GRAMI, which is also implemented in Java 1.8. All the single-machine experiments were run on a 24 Cores (2.40GHz) Intel Xeon $E5 - 2440$ with 188Gb RAM with Linux 3.13.

RESUM *distributed* is implemented in Java 1.8 on top of the Arabesque framework, which is in turn built as a layer on top of Apache Spark [ZXW$^+$16] (v. 2.0.0). All the multi-machine experiments were run on a cluster of 7 machines: the master is a 8 Cores machine with 80Gb RAM and Linux 14.04, while the workers are 16 Cores machines with 30Gb RAM and Linux 14.04.

[3]`github.com/ehab-abdelhamid/GraMi`
[4]`https://github.com/lady-bluecopper/ReSuM`

### 3.8.3 Generation of the Weights

Since we had real weights only for the AMAZON graph, to test the scalability of our method with a larger number of weighting functions, for the other datasets we created synthetic weights based on the results of a user study we conducted on the Crowdflower[5] platform. We extracted a sample from the FreeBase knowledge base, restricting the domain of the edge labels to five topics (Music, Books, Celebrities, Movies, and Sport). Then we asked the users to rate each graph edge (i.e., fact) according to their preferences, using a relevance value between $1$ and $5$. Once collected the relevance values from $123$ users, we modeled the distribution of the edge weights with respect to the number of facts. We found that the edge weights, after normalization, are distributed as a Gaussian with mean $0.452$ and variance $0.02$. In addition, we noted that, on average, a user rated above $0$ between $10\%$ and $20\%$ of the labels, and thus we concluded that real graph weights are usually quite sparse. Therefore, we uniformly subset edge labels according to our findings and generated weights normally distributed in $[0, 1]$.

Furthermore, in order to evaluate the performance of RESUM and RESUM *approximate* with different weight distributions, we generated sets of synthetic edge weights, varying a *focus* parameter representing the ratio of weighted edges for each edge label. The edge weights were sampled from a normal distribution $\mathcal{N}(\mu, \sigma^2)$ and a $Beta(\alpha, \beta)$ distribution, hence allowing us to prove the effectiveness of our algorithms under normally distributed weights and exponentially distributed weights. We set $\mu = 0.5$ and $\sigma = 0.25$ for the normal distribution and $\alpha = 0.7, \beta = 5$ and $\beta = 0.7, \alpha = 5$, for the $Beta$ distribution. The two choices of the parameters for the $Beta$ distribution represent two extreme of an exponential behavior: the former concentrates the probability mass on low weights, the latter on large weights. The *focus* parameter takes values in the range $\{0.5, 0.8\}$ for the normal distribution and in the range $\{0.25, 0.5, 0.75, 1\}$ for the $Beta$ distribution.

### 3.8.4 Comparison with Frequent Pattern Mining

We compared the patterns returned by a frequent pattern mining algorithm (FREQ) and our algorithm RESUM to validate our claim that frequent pattern mining returns a large number of low-weight patterns, which, instead, are correctly discarded in score-based pattern mining. Unless otherwise stated, we report the average of 10 different randomly sampled weighting

[5] `www.crowdflower.com`

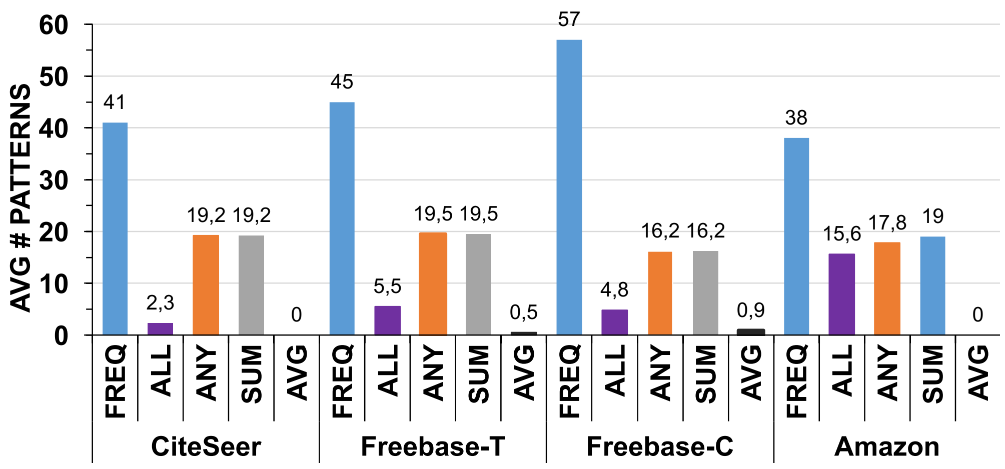


**Figure 3.4:** Average number of patterns found in each dataset, using different *scores* and default parameters.

functions. In particular, these weights were sampled from a normal distribution using *focus* $0.5$, as previously described.

Figure 3.4 reports the average number of patterns found using different scoring functions on the four datasets, with default parameters, as shown in Table 3.1. We observe that FREQ returns patterns, at least half of which are irrelevant with respect to any of the four scoring functions. As expected, in all the datasets, *ANY* and *SUM* return more patterns than *ALL* and *AVG*, due to the less restrictive conditions on the weights. On the other hand, *AVG* returns a low number of patterns, mainly because more than 50% of the edges have low or zero weight. Therefore, *AVG* is particularly suited in graphs where weights are uniformly distributed in the entire graph, e.g., biological or chemical datasets.

We now discuss quality (Table 3.2), number of patterns, and running time of RESUM compared to FREQ, when varying relevance ($\alpha$) and frequency ($\tau$) threshold (Figure 3.5 and 3.6). Here we report results for two datasets (FREEBASE-C and FREEBASE-T), however we observe similar results also on the other datasets. In particular, as an example, within the top-5 *frequent* patterns in the AMAZON graph, we found that the most frequently bought products are Sony appliances, but some *relevant* patterns actually involve Nikon products. This result shows that Sony products are popular but not interesting for all the users.

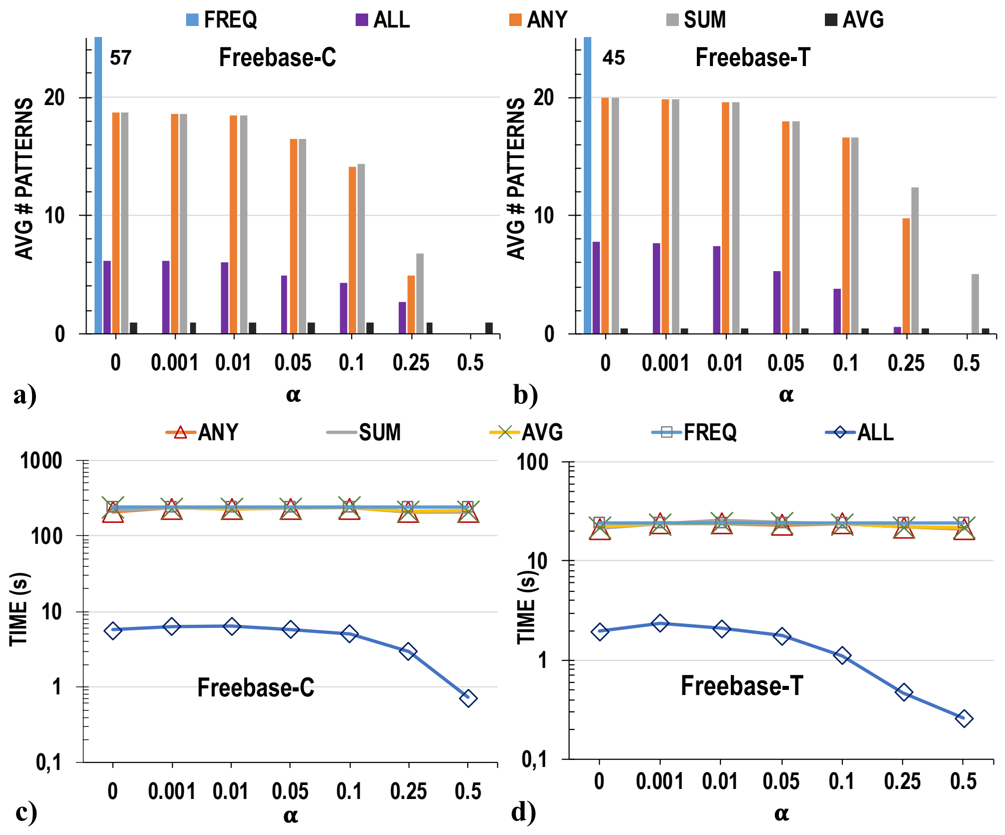


**Figure 3.5:** Varying $\alpha$: average number of patterns (top) and running time (bottom) in FREEBASE-C (a,c) and FREEBASE-T (b,d).

| | FREEBASE-C | | | | FREEBASE-T | | | |
|---|---|---|---|---|---|---|---|---|
| top-$k$ | *ALL* | *ANY* | *SUM* | *AVG* | *ALL* | *ANY* | *SUM* | *AVG* |
| 1 | 0.6 | 0.6 | 0.6 | 0.86 | 0.5 | 0.5 | 0.5 | 1 |
| 3 | 0.43 | 0.43 | 0.43 | 1 | 0.45 | 0.33 | 0.33 | 1 |
| 10 | 0.44 | 0.49 | 0.49 | 1 | 0.8 | 0.66 | 0.66 | 1 |

**Table 3.2:** Quality of FREQ vs RESUM on the top-$k$ patterns.

#### 3.8.4.1 Quality of FREQ vs RESUM

Table 3.2 shows the quality of the patterns discovered by FREQ, measured on the $k$ most frequent patterns. We selected 10 random weighting functions and mined the patterns relevant for each of them. The quality of FREQ is measured as the average Jaccard similarity between the top-$k$ frequent patterns and the top-$k$ relevant patterns. As expected, frequency is a bad predictor of relevance, since most of the relevant patterns are not in top-$k$ frequent patterns. Notably, for *AVG* the quality is higher mostly due to the small or null number of patterns returned, as reported in Figure 3.4.

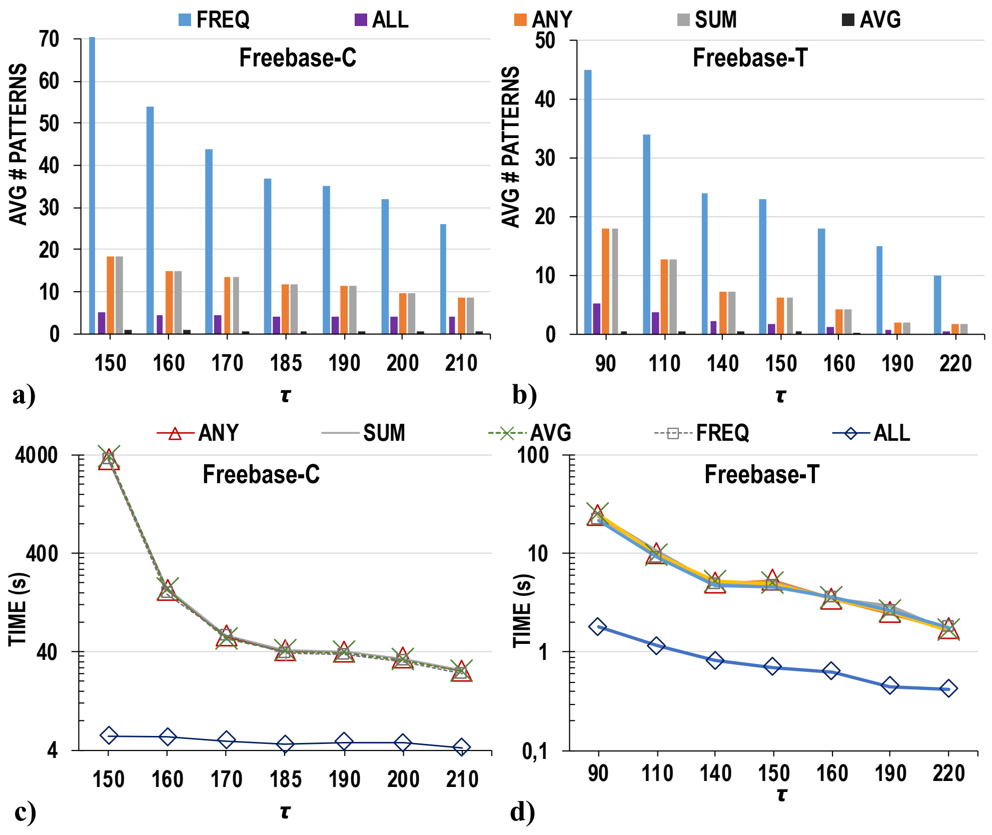


**Figure 3.6:** Varying $\tau$: average number of patterns (top) and running time (bottom) in FREEBASE-C (a,c) and FREEBASE-T (b,d).

### 3.8.4.2 Analysis of the Relevance Threshold

The relevance threshold $\alpha$ is a system-dependent parameter set only for *ALL*, *ANY*, and *SUM*. It can be easily tuned on demand and strongly affects the number of patterns (Figure 3.5 (a) and Figure 3.5 (b)), because the larger the value of $\alpha$, the smaller is the number of appearances that are considered in the computation of the score of the patterns, and thus the smaller is the total number of relevant patterns mined. We observe that with $\alpha$>0 the number of relevant patterns is less than half of the number of the frequent ones. This behavior reflects the characteristics of the weights in the datasets, as half of the edges have zero weight. Moreover, for FREEBASE-T, *SUM*, being the most lenient scoring function, returns patterns even in the restrictive cases when $\alpha > 0.5$ (Figure 3.5 (b)). Finally, since *AVG* does not depend on $\alpha$, it always returns the same patterns.

Figure 3.5 (c) and Figure 3.5 (d) show that the threshold $\alpha$ affects the running time of RESUM mostly when *ALL* is used, as this function can prune the irrelevant patterns earlier in the process.

In fact, an occurrence of a pattern is discarded and not included in the support set of any extension of the pattern, as soon as one edge weight is found to be below $\alpha$. On the other hand, for all the other scoring functions, the extension of an invalid occurrence of a pattern can be valid for some super-pattern, and therefore cannot be discarded until all its edge weights have been examined. As a consequence, the running time of the algorithm is almost unaffected by $\alpha$.

#### 3.8.4.3 Analysis of the Frequency Threshold

Figure 3.6 reports the behavior of RESUM and FREQ when varying the frequency threshold $\tau$. We performed preliminary tests to decide a reasonable range of values $[\tau_{min}, \tau_{max}]$ for each dataset. In particular, the $\tau_{min}$ corresponds to the smallest value that allowed FREQ to terminate the computation within 48 hours, and $\tau_{max}$ is the maximum value returning a non-empty set of frequent patterns. The choice of different ranges for each dataset is consistent with previous researches [EASK14] and reflects the observation that pattern frequency is dataset-dependent, while relevance is user-dependent.

As we can see in Figure 3.6 (a) and Figure 3.6 (b), the number of frequent patterns decreases almost linearly with $\tau$, and consequently the number of relevant patterns decreases as well. Regarding the performance, as opposed to the score threshold, the frequency threshold always alters the computation time, since higher values lead to an early pruning of many patterns, and thus the algorithm terminates earlier. Moreover, Figure 3.6 (c) and Figure 3.6 (d) show that when $\tau$ takes low values (i.e. between 150 and 180), RESUM runs up to two orders of magnitude faster in both the datasets. Finally, as previously noted, *ALL* performs significantly better than the other scoring functions.

### 3.8.5 The Case of Multiple Weights

We tested the scalability of RESUM in the case of multiple weighting functions, varying their number between 50 and 50.000 in the real graphs, and between 1 and 1000 in the synthetic graphs. Similarly, we also measured time and quality of RESUM *approximate*. In the following we do not further discuss and report the number of patterns retrieved for each weighting function and each scoring function, since these results are consistent with what reported in the single weight case.

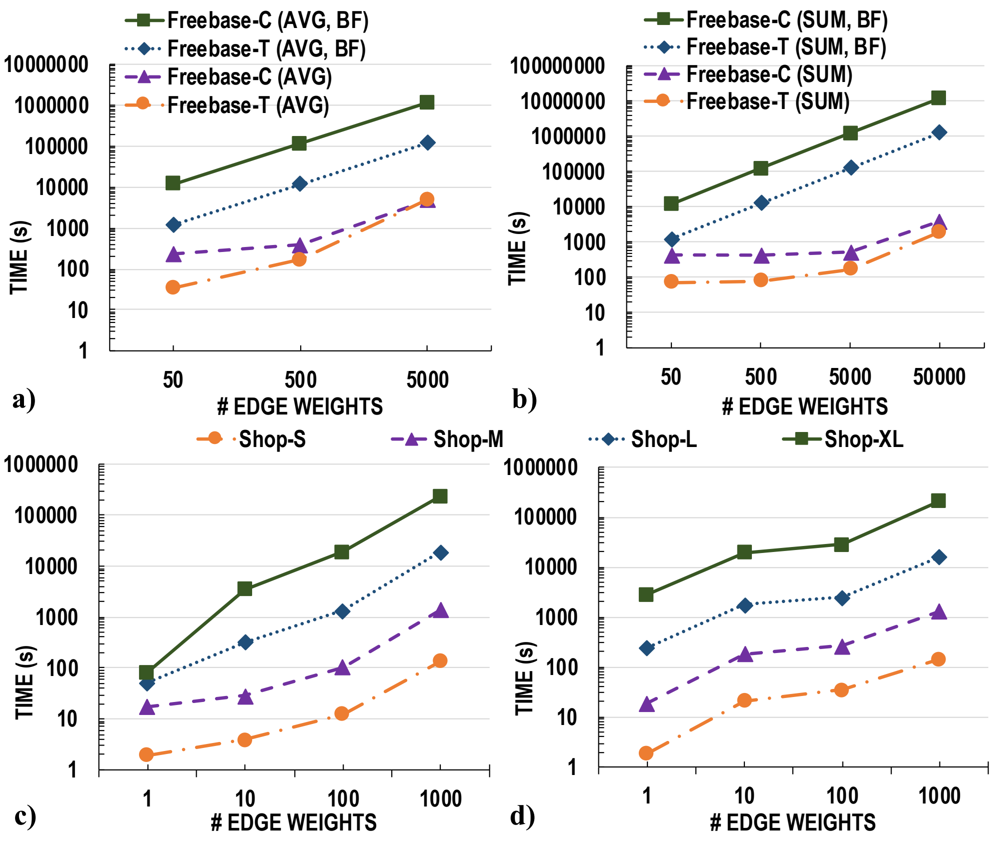


**Figure 3.7:** Scalability of ReSuM: running time in Freebase-C and Freebase-T compared with the brute-force approach (BF), varying number of edge weights, using *AVG* (a) and *SUM* (b); and running time in Shop-S, Shop-M, Shop-L, and Shop-XL, varying number of edge weights, using *AVG* (c) and *SUM* (d).

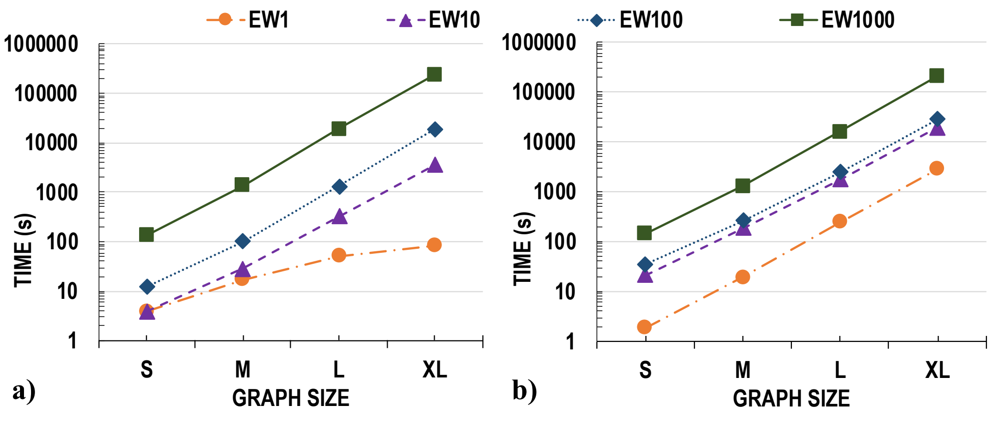


**Figure 3.8:** Scalability of ReSuM: running time in Shop-S (S), Shop-M (M), Shop-L (L), and Shop-XL (XL), using *AVG* (a) and *SUM* (b), and using 1, 10, 100, and 1000 edge weights (EW1, EW10, EW100, EW1000).

#### 3.8.5.1 Scalability in Real Graphs

Figure 3.7 shows the impact of the number of weighting functions on the running time. We report the performance obtained with weights sampled from a normal distribution with *focus* 0.5.

Figure 3.7 (a,b) presents the comparison between RESUM and the brute-force (BF) approach, which computes the patterns for each weighting function separately. While BF scales linearly with the number of weighting functions, the running time of RESUM is nearly constant with 5000 functions, and slowly increases as the number of edge weights approaches 50000. As a pitfall, the memory requirement grows linearly with the number of weights for both algorithms. Note that RESUM keeps all the edge weights and the scores in main memory to speed up the score computation and the pattern evaluation respectively, and thus the number of weighting functions it can handle heavily depends on the available memory. On our machine, we were able to process up to 5000 functions when using the *AVG* score (Figure 3.7 (a)), while we were able to scale larger than 5000 when using the *SUM* score (Figure 3.7 (b)).

#### 3.8.5.2 Scalability in Synthetic Graphs

The synthetic graphs were generated using the same degree distribution, and assigning the node and edge labels proportionally to their size. As a consequence, they display relatively similar characteristics and can thus be effectively used to test the scalability of our approaches in terms of the input size only. Figure 3.7(c,d) shows the performances of RESUM in both the single edge weight and the multi-weighted edge setting, when using the *AVG* (c) and the *SUM* (d) scoring functions. The weights were generated using a Beta distribution with parameters $\alpha = 0.7$, $\beta = 5$, and *focus* $0.75$. Figure 3.8 (a,b) shows that adding one order of magnitude to the size of the graph causes a performance degradation by up to one order of magnitude for all the edge weight settings (EW), and this one order of magnitude difference is maintained when increasing the number of weights per edge (Figure 3.7(c,d)). On the other hand, an increase in the number of weights do not lead to an equally steep increase in the running time.

Finally, we note that the performance with *AVG* is comparable to that of *SUM*, even though *AVG* requires the algorithm to find all the embeddings of the pattern, while *SUM* terminates the algorithm as soon as enough embeddings are found.

In Figure 3.9 (a, b) instead, we compare RESUM and RESUM *approximate*. For these set of experiments, we generated the representative functions by first clustering the weighting functions using the bucket-based strategy. The clustering phase is performed as a preprocessing and not reported, since it is agnostic to the choice of the various thresholds and depends solely on the clustering algorithm (e.g., k-means, hierarchical, or spectral). In particular, we tried numbers of buckets $b$ of different orders of magnitude and proportional to the frequency of the edge labels in

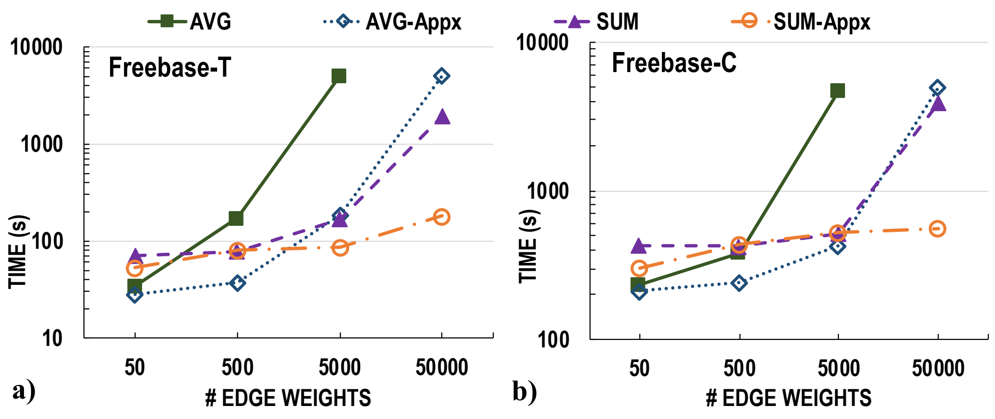


**Figure 3.9:** Scalability of ReSuM and ReSuM *approximate*: running time in Freebase-T (a) and Freebase-C (b) using *AVG* and *SUM*, varying number of edge weights.

the graph. Then, we run $k$-means using different $k$ to study the impact of the number of clusters on the quality and the running time of ReSuM *approximate*. Finally, we set the default value of $b$ of each dataset to the number of buckets that allowed the algorithm to use at least one order of magnitude less memory than those consumed using the full-vector strategy, i.e., 12 buckets for Freebase-T, 16 for Freebase-C, and 10 for CiteSeer.

We observe that ReSuM becomes impractical as the number of weighting functions increases. As a matter of fact, when *AVG* is used, ReSuM exhausts the available memory, hence returning no patterns. This behavior reflects the characteristics of *AVG*, which requires the algorithm to exhaustively search for all the occurrences of a pattern before computing its score. In contrast, ReSuM *approximate* terminates the computation. On the other hand, when *SUM* is used, ReSuM is able to return the relevant patterns; however, ReSuM *approximate* outperforms the exact algorithm again, taking nearly constant time to terminate. In conclusion, in all the cases of large numbers of weighting functions, ReSuM *approximate* performs better than ReSuM by at least one order of magnitude.

### 3.8.6 Case Study

Figure 3.10 reports the most frequent pattern ($P_1$), the most relevant pattern for two randomly selected users $u_1$ and $u_2$ ($P_2$), and a pattern in the top-5 relevant patterns for $u_1$ and $u_2$ ($P_3$ and $P_4$ respectively), found in the real network Amazon using the *ALL score* and the default settings in Table 3.1.

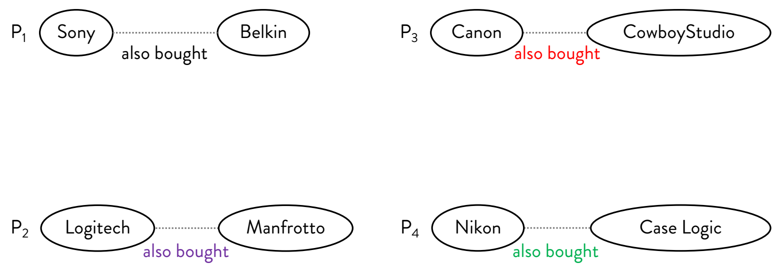


**Figure 3.10:** Case study: the most frequent pattern (above left), the most relevant pattern (above right), and two of the top 5 patterns (below left and right) found in AMAZON.

Pattern $P_1$ shows that users who bought Sony products, frequently bought also Belkin products in other transactions. This result makes sense, considering that Sony sells electronics and Belkin accessories for computers, smartphones, and cameras. On the other hand, pattern $P_2$ confirms our claim that frequent patterns are not necessarily the most interesting patterns for every user, as it contains other less popular node labels, which indicate a more professional user. The high relevance score of $P_2$ follows from the high ratings that users $u_1$ and $u_2$ gave to their Canon and Cowboy Studio purchases. Note that Cowboy Studio is a US retailer selling camera accessories such as tripods, lens, batteries, and flashes, and thus, the appearance of these two node labels in the same pattern is realistic. Thanks to pattern $P_2$ we know that user $u_1$ prefers the more professional Cowboy Studio accessories, and thus, if she buys a new camera, we can recommend her a portrait umbrella rather than a cover from Belkin.

Finally, patterns $P_3$ and $P_4$ prove that different users like different products, and in particular, $u_1$ expressed her preference for Logitech and Manfrotto, while $u_2$ liked Case Logic accessories for her Nikon purchases. As a consequence, we attest that our algorithms can effectively help the design of personalized recommending systems.

### 3.8.7 Quality of ReSuM *approximate*

As discussed in Section 3.5, we measure the quality of ReSuM *approximate* in terms of the average distance between the patterns it returns (sets $A_i$) and those returned by ReSuM (sets $R_i$). Table 3.3 (left) reports the quality values obtained using the four scoring functions in FREEBASE-T. Higher values indicate higher quality, with 1 representing the maximum quality and 0 representing the minimum quality. As we can see, *ANY* and *SUM* exhibit the best quality; *ALL* performs reasonably good, despite being more restrictive and therefore more sensitive to the approximation based on the maximum edge weights. On the other hand, when *AVG* is used,

| | *Q* | | | |
|---|---|---|---|---|
| $\lvert W\rvert$ | *ALL* | *ANY* | *SUM* | *AVG* |
| 50 | 0.805 | 0.931 | 0.931 | 0.373 |
| 500 | 0.808 | 0.938 | 0.938 | 0.382 |
| 5000 | 0.797 | 0.947 | 0.947 | 0.391 |
| 50000 | 0.796 | 0.948 | 0.949 | - |

| | *Q* | | | |
|---|---|---|---|---|
| | *ALL* | *ANY* | *SUM* | *AVG* |
| *clustering* | FREEBASE-T | | | |
| A-POST | 0.80 | 0.93 | 0.93 | 0.30 |
| BUCK | 0.80 | 0.94 | 0.94 | 0.38 |
| *clustering* | FREEBASE-C | | | |
| A-POST | 0.72 | 0.93 | 0.93 | 0.55 |
| BUCK | 0.73 | 0.93 | 0.93 | 0.61 |

**Table 3.3:** Quality of RESUM *approximate* varying the number of edge weights $|W|$ in FREEBASE-T (left), and using BUCK and A-POST clustering in FREEBASE-T and FREEBASE-C (right), measured using Equation 3.2.

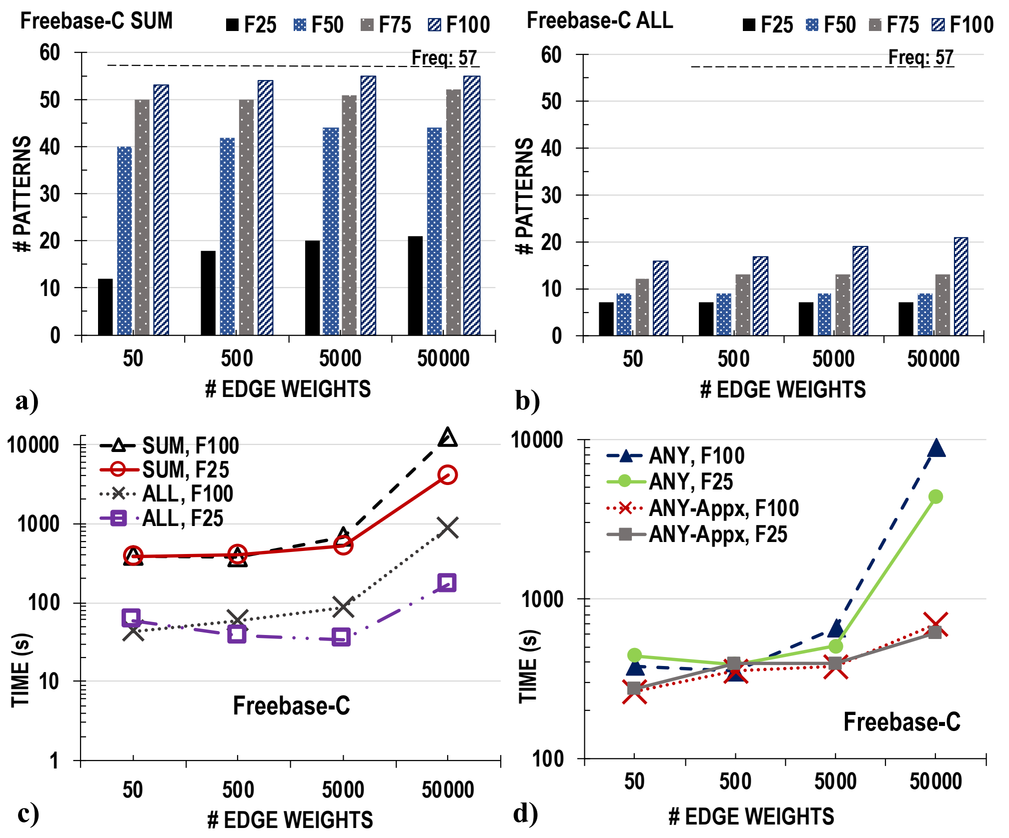


**Figure 3.11:** Varying *focus* in FREEBASE-C: number of patterns using *SUM* (a) and *ALL* (b) with *focus* between 0.25 (F25) and 1 (F100); and running time of RESUM and RESUM *approximate* with $Beta(0.7, 5)$ weights with *focus* 0.25 (F25) and 1 (F100), using *SUM*, *ALL* (c), and *ANY* (d).

the quality of the answer is quite poor. Nevertheless, this behavior is due to the extremely low number of patterns this scoring function considers interesting, which skews the computation of the pattern set distance. Note that, we do no report any value for the case of 50000 weighting functions with *AVG*, since the algorithm exhausted all the available memory and did not terminate. We conclude that, the additional patterns returned by RESUM *approximate* are indeed closely related to the relevant patterns of each individual weighting function.

Finally, we tested the capability of our bucket-based clustering (BUCK in short) to correctly identify groups of similar weighting functions. To this end, we compared the quality of the results mined using BUCK in the creation of the feature vectors of the weighting functions, with the quality measured using a *ground-truth* clustering (A-POST in short). The A-POST clustering was created using the sets of relevant patterns $R_1, \dots, R_m$ as feature vectors of $\omega_1, \dots, \omega_m$, and then running a $k$-medoid algorithm. We regard it as a ground-truth clustering, because it is obtained knowing what makes two weighting functions really similar, i.e. their relevant patterns, and maximizing the intra-cluster similarity. Table 3.3 (right) reports the comparison between A-POST and BUCK on FREEBASE-C and FREEBASE-T. We recall that lower values mean higher quality, as they indicate distances. We can see that we experience a quality comparable with that obtained using A-POST, and thus we can conclude that our clustering technique is indeed effective.

### 3.8.8 Impact of the Weight Distribution

For the experiments presented above, we weighted the Amazon graph using real weights, whereas for the FREEBASE-T, FREEBASE-C, and CITESEER graphs we used synthetic weights generated according to the results of our user study. The common feature of these two kinds of weights is that they are highly sparse. It is worth studying whether weights following other distributions or that are denser, affect the performance of our algorithms. To this end, we performed an additional set of experiments using weighting functions generated following a $Beta(5, 0.7)$, a $Beta(0.7, 5)$ and a normal distribution with different densities (*focus*), as described at the beginning of Section 3.8.

One would expect that, with higher densities, the cost of the computation would be higher too. Although these expectations are reasonable, in the following we show that the behavior of RESUM and RESUM *approximate* is consistent with what observed in the case of sparse weights. Figure 3.11 (a) and Figure 3.11 (b) report the average number of patterns found using *SUM* and *ALL*, with weights generated using a $Beta(0.7, 5)$ distribution with *focus* varying between 0.25 and 1 (i.e., all edges have weight $> 0$). Comparing these results with those in Figure 3.4 when *SUM* is used, we can see that the number of relevant patterns is largely affected by the presence of more (or all) edges with non-null weight, meaning that the patterns mined are actually many more. On the other hand, when *ALL* is used, RESUM still finds a larger number of relevant patterns, but the increment is not as large as in the *SUM* case.

| | $Q$ | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *ALL* | | | | *SUM* | | | | *ALL* | | *SUM* | |
| | 0.25 | 0.5 | 0.75 | 1 | 0.25 | 0.5 | 0.75 | 1 | 0.5 | 0.8 | 0.5 | 0.8 |
| $\lvert W\rvert$ | $Beta(0.7, 5)$ | | | | | | | | $\mathcal{N}(0.5, 0.25)$ | | | |
| 50 | 0.22 | 0.20 | 0.21 | 0.26 | 0.17 | 0.53 | 0.74 | 0.91 | 0.15 | 0.18 | 0.36 | 0.54 |
| 500 | 0.21 | 0.21 | 0.24 | 0.27 | 0.18 | 0.53 | 0.74 | 0.91 | 0.18 | 0.20 | 0.44 | 0.57 |
| 5000 | 0.21 | 0.22 | 0.24 | 0.28 | 0.19 | 0.54 | 0.75 | 0.91 | 0.22 | 0.20 | 0.49 | 0.59 |
| 50000 | 0.21 | 0.22 | 0.25 | 0.28 | 0.19 | 0.54 | 0.75 | 0.91 | 0.22 | 0.21 | 0.51 | 0.59 |
| $\lvert W\rvert$ | $Beta(5, 0.7)$ | | | | | | | | | | | |
| 50 | 0.19 | 0.23 | 0.42 | 1 | 0.34 | 0.73 | 0.94 | 1 | | | | |
| 500 | 0.21 | 0.24 | 0.42 | 1 | 0.36 | 0.73 | 0.94 | 1 | | | | |
| 5000 | 0.21 | 0.25 | 0.43 | 1 | 0.36 | 0.74 | 0.95 | 1 | | | | |
| 50000 | 0.21 | 0.25 | 0.44 | 0.99 | 0.36 | 0.74 | 0.95 | 1 | | | | |

**Table 3.4:** Quality of ReSum *approximate* with *ALL* and *SUM* on Freebase-C, with $Beta(\alpha, \beta)$ and normal $\mathcal{N}(\mu, \sigma^2)$ weights generated using *focus* values in $\{0.25, 0.5, 0.75, 1\}$ and $\{0.5, 0.8\}$ respectively, measured using Equation 3.1.

Regarding the running time, Figure 3.11 (c) and Figure 3.11 (d) show that the two algorithms behave accordingly to what already seen in the previous experiments, meaning that the fact there more patterns are mined do not downgrade the performance heavily.

Finally, Table 3.4 displays the quality of ReSum in terms of average precision, as defined in Equation 3.1. As we can see, our approximate algorithm achieves similar quality values no matter which weight distribution is chosen. In addition, the denser the weights in the graph, the higher is the average precision of the pattern sets mined. Intuitively, this is due to the fact knowing a larger number of positive weights allows the clustering algorithm to better detect which weighting functions are similar.

### 3.8.9 Comparison with Distributed Pattern Mining

We first investigate in which cases the distributed algorithm (noted as Dist) offers an advantage over the centralized one. When running both algorithms on the CiteSeer graph, Figure 3.12 (a,c) shows that the distributed version is always one order of magnitude slower. We note that CiteSeer is a small and relatively dense graph with few labels, meaning that the graph contains a small number of candidate relevant patterns with large sets of embeddings. In this particular type of graph, the centralized algorithm can stop the search of new assignments and early terminate the computation for a pattern as soon as the embeddings found are sufficient to verify that the pattern has a high relevance score. On the other hand, in the distributed algorithm each worker looks for embeddings separately, and the aggregation of the embeddings takes place

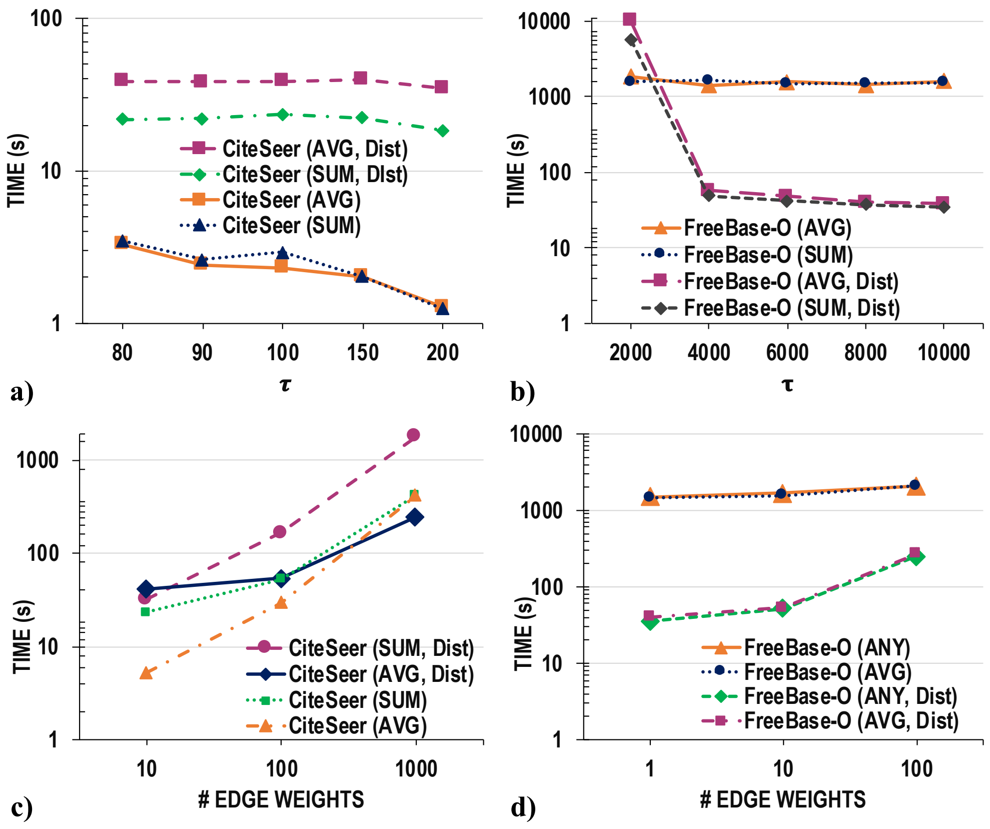
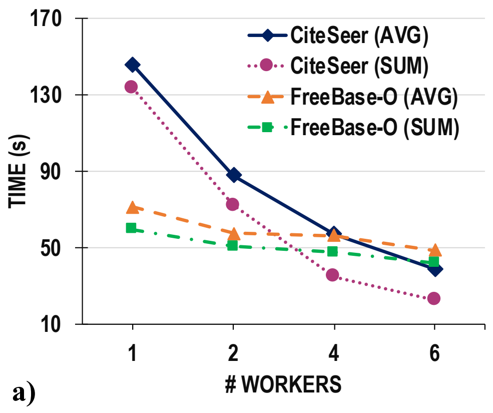


**Figure 3.12:** ReSum vs ReSum *distributed*: running time in CiteSeer (a) and Freebase-O (b) varying $\tau$, using *SUM* and *AVG*; and running time in CiteSeer (c) and Freebase-O (d) varying number of edge weights, using *SUM* and *AVG*.

only at the end of the computation step. For this reason, the algorithm cannot exploit the early termination condition, and hence may compute far more embeddings than necessary.

ReSum *distributed*, instead, provides a clear advantage in the larger and richer Freebase-O graph. This graph has a higher number of labels, which, paired with the larger size of the graph, allows for workers to share effectively the workload and provide the answer in more than one tenth of the time required by the centralized version (Figure 3.12 (b)). The striking difference between the performance of the two algorithms in the two datasets, suggests that the distributed version has to be preferred for larger and richer graphs, when many different patterns can be retrieved. The same behavior is observed when changing the number of users (Figure 3.12 (d)), proving that our strategy for the multi-weight pattern mining is still effective in the distributed environment.



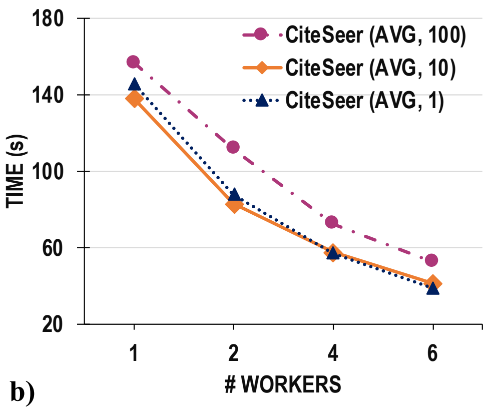


**Figure 3.13:** Varying number of workers: running time of RESUM *distributed* in CITESEER and FREEBASE-O, using *SUM* and *AVG* with single edge weights (a) and using *AVG* varying number of edge weights (b).

#### 3.8.9.1 Varying the Number of Workers

In order to better understand which dimensions affect the most the performance of the distributed algorithm, we compared its running times over both the CITESEER and the FREEBASE-O graph, when varying the number of workers[6] Figure 3.13 (a) shows that, with one or two workers, the RESUM *distributed* is sensibly slower on the much smaller citation network than on the larger knowledge graph. Note that for FREEBASE-O the algorithm returned around 30 relevant patterns, while for CiteSeer it retrieved 66 patterns. Only when using 6 machines we were able to run faster on the smaller graph.

In addition, Figure 3.13 (b) shows that the effects of the distribution remain the same when varying the number of weighting functions we consider. This substantiates our previous finding, namely, when the graph contains few very frequent patterns the distribution strategy provided by Arabesque is not optimal as it is dominated by the time required to compute an unnecessary large amounts of embeddings. As a consequence, an embedding-based distribution cannot be generally recommended for score-based pattern mining, although it provides higher load balance and less worker communication than the most popular alternative distributed graph processing systems [TFS+15].

[6]For this experiment we kept a single edge-weighting function, and parameter $\alpha = 0.05$, with $\tau = 6000$ for FREEBASE-O, and $\tau = 100$ for CITESEER.

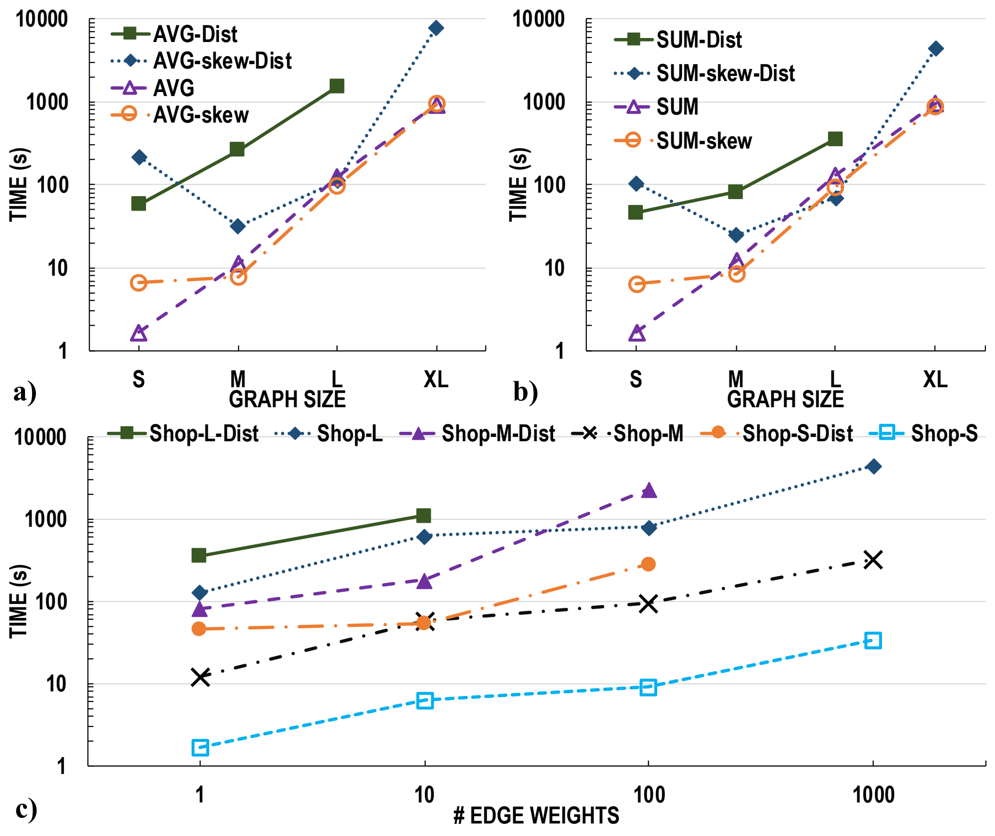


**Figure 3.14:** Comparison between ReSuM and ReSuM *distributed*: running time varying the size of the graph, using *AVG* (a) and *SUM* (b), with the edge weights generated by a Beta distribution $(0.7, 5)$ and a skewed distribution (*skew*); and running time in Shop-S, Shop-M, Shop-L, and Shop-XL, varying the number of edge weights, using *SUM* (c).

### 3.8.9.2 Scalability

We additionally compared ReSuM and ReSuM *distributed* on the four synthetic graphs using default score threshold and frequency threshold 90, 900, 9000, 90000, respectively. At these frequencies, the graphs contain roughly the same number of frequent patterns (47, 43, 44, and 45 respectively), hence allowing us to analyze how the increasing number of embeddings per pattern affects the performances of the two algorithms. For these experiments, we also sampled the edge weights from two different distributions: a Beta distribution with parameters $(0.7, 5)$ and *focus* $0.75$, and a skewed distribution that simulates a user interested in a category of products available in the online shop, and thus assigns a large positive weights to the corresponding edges and zero to the others.

In Figure 3.14(a, b) we can see that ReSuM *distributed* works better when the weights are sparse, with *AVG* achieving performances comparable to *SUM*, similarly to what observed for ReSuM. In particular, the algorithm succeeded in extracting the relevant patterns from the largest graphs with skewed weights, but was able to finish the computation up to the graph of size L when the weights were drawn from the Beta distribution. In the case of sparse weights, the number of embeddings satisfying the condition posed by the scoring function is lower, and thus the workers must process and send a lower number of embeddings through the network. The communication between the machines is thus faster. In contrast, the running time of ReSuM is not significantly affected by the weight distribution, as it does not generate and keep in memory all the embeddings in the graph, as opposed to the distributed framework. In addition, we note that when the graph is small, the overhead of setting up the distributed environment outweighs the cost of mining the relevant patterns, and thus ReSuM *distributed* takes more time than ReSuM to complete the computation. On the other hand, when the size of graph is large, so is the number of embeddings in the graph, and thus the algorithm can suffer from delays in machine communication and increasing cost of embedding generation.

Finally, Figure 3.14 (c) reports the running times of ReSuM and ReSuM *distributed* in the four synthetic graphs, varying the number of weights per edge. The performance is consistent with the single-weight setting, thus demonstrating the superiority of the centralized algorithm and the complexity of developing practical and scalable distributed solutions to graph mining problems.

# Chapter 4

# Dense Correlated Subgraph Mining

In this chapter we focus on the second contribution of this thesis, which is the identification of dense and correlated groups of edges in dynamic networks. In dynamic networks, nodes and edges can undergo both structural and attribute changes, and there are cases in which some of those nodes and edges evolve in a convergent manner, meaning that they display a positive correlation on their behavior. These groups of correlated elements, especially when they involve nodes and edges that are topologically close, can represent regions of interest in the network. In the following, we present two different measures to compute the density of a group of edges in a dynamic settings, a measure to compute their temporal correlation, and the solution we propose to the problem of enumerating all the maximal dense groups of correlated edges that are pairwise dissimilar. We introduce a constraint on the similarity of the groups returned, as many real-world networks naturally contain a large number of dense groups of correlated edges, and thus an user may rather prefer a more compact but yet representative subset of results.

## 4.1 Contributions

The contributions of this chapter can be summarized as follows:

- We introduce and formally define the generic problem of detecting a set of dense and correlated subgraphs in dynamic networks (Section 4.2).

- We propose two different measures to compute the density of a group of edges that change over time, which are based on the average-degree density [Cha00, Gol84], and a measure to compute their correlation, which is based on the Pearson correlation (Section 4.2).
- We develop an exact solution, called EXCODE, for enumerating all the subgraphs that satisfy given density and correlation thresholds. We also propose an approximate solution that scales well with the size of the network, and at the same time achieves high accuracy (Section 4.3).
- As some networks naturally contain a large number of dense groups of correlated edges, we study the problem of identifying a more compact and diverse subset of results that is representative of the whole answer set. To this aim, we introduce a threshold on the maximum pairwise Jaccard similarity allowed between the edge groups in the result set, and extend our framework with an approach to extract a set of subgraphs whose pairwise overlap is less than the threshold (Section 4.3).
- We evaluate our framework through an extensive set of experiments on both real and synthetic datasets, confirming the correctness of the exact solution, the high accuracy of the approximate one, the scalability of the framework, and the applicability of the solution on networks of different nature (Section 4.4).
- We showcase EXCODE-VIZ, a powerful tool that can help the user to configure the EXCODE framework easily and effectively, visualize the results of the mining process, and finally interact with them (Section 4.5).

## 4.2 Problem Formulation

We start by presenting our notation and defining the problem we study in this chapter. The main actor in our setting is a *dynamic network*, which is a graph that models data that change over time, and is represented as a sequence of consecutive static graphs, also called the *snapshots* of the network.

**Definition 4.1** (Dynamic Network)**.** Let $T \subseteq \mathcal{T}$ be a set of time instances over a time domain $\mathcal{T}$. A *dynamic network* $D = (V, E)$ is a sequence of graphs $G_i = (V, E_i, \omega_i)$ with $i \in T$, referred to as snapshots of the network, where $V$ is a set of vertices, $E_i \subseteq V \times V$ is a set of edges between these vertices, and $\omega_i : E_i \mapsto \mathbb{R}$ is an edge weight function. We denote with $E$ the union of the edges of the snapshots, i.e., $E{=}\cup_{i\in T}E_i$.

Note that we consider dynamic networks where all the snapshots share the same set of nodes. The case of disappearing nodes can be easily mapped to an instance of our problem by inserting, in each snapshot, all the nodes that are not present in this snapshot but exist in other snapshots. Such nodes will be isolated in that snapshot. We slightly abuse notation, and whenever an edge $e$ does not appear in a graph $G_i$, i.e., $e \notin E_i$, then we assume that $\omega_i(e) = 0$. In the case of unweighted graphs, the edge weight functions $\omega_i$ take values in $\{0, 1\}$.

The focus of our work is on identifying dense correlated subgraphs in dynamic networks. We recall that, given a graph $G = (V, E)$, a *subgraph* $H$ of $G$ is a graph $H = (V_H, E_H)$, such that $V_H \subseteq V$ and $E_H \subseteq E$. We adopt the traditional notion of density used for static graphs [Cha00, KS09] to measure the density of the subgraph.

**Definition 4.2** (Density)**.** The *density* of a (static) graph $G = (V, E)$ is the average degree of its nodes, i.e., $\rho(G) = 2|E|/|V|$.

In the case of a dynamic network $D$, the edges of a subgraph $H$ may not exist in all its snapshots, meaning that this density measure would give a different value in each snapshot. Therefore, we propose two alternative ways to aggregate those values. Let $G_i(H) = (V_H, E_H \cap E_i)$ denote the subgraph induced by $H$ in the snapshot $i$. The first approach, called *minimum density* and denoted as $\rho_m$, computes the density of $H$ as the minimum density of any subgraph induced by $H$ across the snapshots of $D$; while the second approach, called *average density* and denoted as $\rho_a$, computes the average density among these induced subgraphs. In particular

$$\rho_m(H) = \min_{i\in T} \rho(G_i(H)) \tag{4.1}$$

and

$$\rho_a(H) = \frac{1}{|T|}\sum_{i\in T} \rho(G_i(H)). \tag{4.2}$$

Given a density threshold $\delta$, a subgraph $H$ is called $\delta$-dense if $\rho_m(G) \geq \delta$ or $\rho_a(G) \geq \delta$, respectively.

Consider a subgraph $H$ that is highly dense in some snapshots, while there are other snapshots in which it does not appear. For these particular subgraphs, $\rho_m(H)$ will be 0 even if there is just one snapshot in which the edges of $H$ do not appear. Similarly, $\rho_a(H)$ will take low values, even though $H$ has a large average degree in the snapshot in which it appears. These definitions are thus too strict for most practical situations: in many applications edges appear and disappear and an interesting event (or an anomaly [RSK+15]) may take place in the dynamic network and exhibit itself only in a small number of snapshots. To account for such situations, we introduce the notion of *activity* and say that a subgraph $H$ is active at time $t$ if at least $k$ edges of $H$ exist in $t$, i.e., $|E_t \cap E_H| \geq k$. Then, we relax our density definitions and compute the minimum and average density of $H$ by aggregating only the values computed in the snapshots where $H$ is active. Let $T_H^k$ denote the subset of snapshots $H$ is active, i.e., $T_H^k = \{t \mid t \in T \text{ and } |E_t \cap E_H| \geq k\}$. We redefine Equation 4.1 and Equation 4.2 as follows:

$$\rho_m^k(H) = \min_{i \in T_H^k} \rho(G_i(H)) \tag{4.3}$$

and

$$\rho_a^k(H) = \frac{1}{|T_H^k|} \sum_{i \in T_H^k} \rho(G_i(H)). \tag{4.4}$$

If $T_H^k$ is empty then both $\rho_m^k(H)$ and $\rho_a^k(H)$ are defined to be 0. Also, we use the notation $\rho^k$ to collectively refer at $\rho_m^k$ and $\rho_a^k$.

Since the dynamic part of a network consists in its edges (recall that all the snapshots of the network have the same set of nodes), we say that a subgraph is correlated if its edges are pairwise correlated. Intuitively, two edges are correlated if they demonstrate a similar behavior of presence/absence across the different snapshots of the network. We therefore represent every edge as a time series over the snapshots of the network, and measure the correlation between two edges as the Pearson correlation between the corresponding time series. We consider this measure because it has been widely used to detect associations between time series [DBD94], but our framework can work with any other correlation measure. Let $\mathbf{t}(e)$ denote the time series of the edge $e$, where each coordinate $t_i(e)$ is set to $t_i(e) = \omega_i(e)$, and thus $t_i(e) = 0$ if $e$ does not appear in the snapshot $i$.

**Definition 4.3** (Edge Correlation)**.** Given a dynamic network $D{=}(V{,}E)$ and two edges $e_1, e_2 \in E$ with respective time series $\mathbf{t}(e_1) = \{t_1(e_1), \ldots, t_T(e_1)\}$ and $\mathbf{t}(e_2) = \{t_1(e_2), \ldots, t_T(e_2)\}$, the correlation between $e_1$ and $e_2$, denoted as $c(e_1, e_2)$, is the Pearson correlation between $t(e_1)$

and $t(e_2)$, i.e.,

$$c(e_1, e_2) = \frac{\sum_{i=1}^{T}(t_i(e_1) - \bar{t}(e_1))(t_i(e_2) - \bar{t}(e_2))}{\sqrt{\sum_{i=1}^{T}(t_i(e_1) - \bar{t}(e_1))^2}\sqrt{\sum_{i=1}^{T}(t_i(e_2) - \bar{t}(e_2))^2}},$$

where $\bar{t}(e) = \frac{1}{|T|}\sum_{i=1}^{T} t_i(e)$.

Given a correlation threshold $\sigma$, the edges $e_1$ and $e_2$ are considered correlated if $c(e_1, e_2) \geq \sigma$.

Definition 4.3 refers to pairs of edges. We compute the correlation of a subgraph $H$ as the minimum pairwise correlation between its edges, i.e.,

$$c_m(H) = \min_{e_i \neq e_j \in E_H} c(e_i, e_j) \tag{4.5}$$

and say that $H$ is $\sigma$-correlated if $c_m(H) \geq \sigma$.

Our goal is to identify all the dense and correlated subgraphs in a dynamic network. However, since a dense correlated subgraph may contain smaller dense correlated structures due to the nature of the density and correlation measures used, we restrict our attention to the *maximal* subgraphs. Given some properties of interest, we say that a subgraph satisfying the properties is maximal, if it is not a strict subset of another subgraph that satisfies them. Thus, given a dynamic network $D$, a density threshold $\delta$, and a correlation threshold $\sigma$, we want to find all the *maximal* subgraphs $H$ that are $\delta$-dense and $\sigma$-correlated.

As it is often the case with problems that enumerate a complete set of solutions that satisfy given constraints, the answer set could potentially be very large and contain solutions with a large degree of overlap between each other. To counter this effect, we further focus on a modified version of our problem that reports only *diverse* subgraphs, which are subgraphs that differ from one another and are representative of the whole answer set. To measure the similarity between subgraphs, we compute the Jaccard similarity between their edge sets, i.e., the Jaccard similarity between the graph $G'$=($V'$, $E'$) and $G''$=($V''$, $E''$), denoted as $J(G', G'')$, is $J(G', G'')$=$|E' \cap E''|/|E' \cup E''|$. Then, we require that the pairwise similarities between subgraphs in the answer set are lower than a given similarity threshold $\epsilon$. This is in line with previous work that has aimed at finding a diverse collection of dense subgraphs [BBC+15, GGT16].

**Problem 2** (Diverse Dense Correlated Subgraphs)**.** *Given a dynamic network $D$, a density threshold $\delta$, a correlation threshold $\sigma$, and a similarity threshold $\epsilon$, find a collection $\mathcal{S}$ of maximal and diverse subgraphs such that for each $H \in \mathcal{S}$, $H$ is $\delta$-dense and $\sigma$-correlated, and for each distinct $H, H' \in \mathcal{S}$, $J(H, H') \leq \epsilon$.*

We refer to Problem 2 through the shorthand DiCorDiS. Note that the density can be measured by using either $\rho_m^k$ or $\rho_a^k$. Note also that the smaller is the value of $\epsilon$, the smaller becomes the set $\mathcal{S}$, and the more diverse turn out to be its members.

DiCorDiS is an enumeration problem, since the goal is to find all the subgraphs that satisfy certain desirable properties. Its decision problem, i.e., determining whether there exists at least one subgraph satisfying the desirable properties, can be shown to be $\mathbf{NP}$-hard, independently of which density function is used ($\rho_m^k$ or $\rho_a^k$).

**Proposition 4.4.** *Consider a dynamic network $D$, and correlated and density thresholds $\sigma$ and $\delta$, respectively. Finding whether there is an edge set $X$ with induced subgraph $H$ for which $c_m(H) \geq \sigma$ and $\rho^k(H) \geq \delta$ is $\mathbf{NP}$-hard for both $\rho_m^k$ and $\rho_a^k$.*

*Proof.* We can reduce the Dal$k$S problem (densest at-least-$k$ subgraph) [KS09] to our case. An instance of the Dal$k$S problem consists of an undirected graph $G = (V, E)$ and parameters $k'$ and $\delta'$. The task is to decide whether there exists a subgraph $H \subseteq G$ having at least $k'$ nodes and density at least $\delta'$. Given an instance of the Dal$k$S problem we construct an instance of the decision version of DiCorDiS as follows. We construct a network $D$ consisting of two snapshots $G_0$ and $G_1$, where $G_0 = G$ and $G_1 = (V, \emptyset)$. We set $\sigma = 1$, $\delta = \delta'$, and $k = \delta' k'/2$. For a subgraph $H$ of $G$, we define $H' = (V_{H'}, E_{H'})$ to be the corresponding subgraph in $G_0$.

We claim that if there exists a solution $H \subseteq G$ for Dal$k$S, the edge set of the corresponding subgraph $H' \subseteq G_0$ is a solution for DiCorDiS. Indeed, it holds that $|E_{H'}| \geq \delta' |V_{H'}| \geq \delta' k'$, and thus $H'$ and $H$ have the same density, which is at least $\delta$. Furthermore, every pair of edges in $E_{H'}$ satisfies the correlation constraint.

Inversely, if $E_{H'}$ is a solution to DiCorDiS, then $|V_{H'}| \geq 2|E_{H'}|/\delta \geq k'$. The density of the corresponding $H \subseteq G$ is equal to the density of $H'$, which is at least $\delta'$, thus, $H$ is a solution for Dal$k$S as well.

**Algorithm 7** ExCoDE

```
Input: Dynamic network $D = (V, E)$, Density function $\rho^k$
Input: Thresholds: Correlation $\sigma$, Density $\delta$, Size $s_M$
Input: Thresholds: Edges-per-snapshot $k$, Similarity $\epsilon$
Output: Diverse dense correlated maximal subgraphs $\mathcal{S}$
1: $\mathcal{G} \leftarrow$ CreateCorrelationGraph$(G, \sigma)$
2: $\mathcal{C} \leftarrow$ FindMaximalCliques$(\mathcal{G})$
3: $\mathcal{S} \leftarrow$ FindDiverseDenseEdges$(D, \mathcal{C}, \rho^k, \delta, k, s_M, \epsilon)$
4: return $\mathcal{S}$
```

Note that since the computation of $\rho^k$ requires the aggregation of the density values over the snapshots that contain at least $k$ edges, we aggregate only the density value in first snapshot $G_0$. Thus, the proof holds for both versions of $\rho^k$, i.e., $\rho^k_m$ and $\rho^k_a$. $\square$

## 4.3 The ExCoDE framework

To solve the DiCorDiS problem, we propose a two-step approach, called ExCoDE (**Ex**tract **Co**rrelated **D**ense **E**dges). This approach first identifies maximal sets of correlated edges, and then extract subsets of these edges that form a dense subgraph according the density measure selected between $\rho^k_m$ and $\rho^k_a$. The correlation of a set of edges is instead computed using Equation (4.5). Given the dynamic network $D = (V, E)$ we create a *correlation graph* $\mathcal{G} = (E, \mathcal{E})$, such that the vertex set of $\mathcal{G}$ is the edge set $E$ of $D$, and the edges of $\mathcal{G}$ are the pairs $(e_1, e_2) \in E \times E$ that have correlation value $c(e_1, e_2) \geq \sigma$. Using this construction, a *maximal clique* in the correlation graph $\mathcal{G}$ corresponds to a maximal set of correlated edges in $D$. In fact, a set of nodes forms a clique in the correlation graph $\mathcal{G}$ if and only if they are all connected, and thus, the corresponding edges in the dynamic network $D$ have pairwise correlation that is greater than $\sigma$, meaning that they are a set of correlated edges. Furthermore, by the maximality of the clique in $\mathcal{G}$, the corresponding edge set is maximal in $D$.

The flow of ExCoDE is illustrated in Algorithm 7. Starting from the dynamic network $D = (V, E)$ the algorithm first creates the correlation graph $\mathcal{G}$ (line 1) by adding a meta-edge between two edges of $D$ if their correlation is greater than $\sigma$. Then it enumerates all the maximal cliques in $\mathcal{G}$ (line 2). This collection of maximal cliques in $\mathcal{G}$ corresponds to a collection $\mathcal{C}$ of maximal correlated edge sets in $D$. Finally, Procedure FindDiverseDenseEdges (line 3) examines each connected component in $\mathcal{C}$ (by using either the density measure $\rho^k_m$ or $\rho^k_a$) to identify those

**Algorithm 8** CREATECORRELATIONGRAPH (approximate)

```
Input: Dynamic network $D = (V, E)$
Input: Threshold: Correlation $\sigma$
Output: Correlation graph $\mathcal{G} = (E, \mathcal{E})$
 1: $cand \leftarrow \emptyset$; $\mathcal{E} \leftarrow \emptyset$
 2: for all $e \in E$; $i \in [0, h\,r]$ do
 3:     $H[e][i] \leftarrow min\{h_i(t) \mid \omega(e,t) = 1\}$
 4: for all $i \in [0, r]$ do
 5:     $B[i] \leftarrow \emptyset$
 6:     for all $e \in E$ do
 7:         $code \leftarrow H[e][i\,h : (i+1)\,h - 1]$
 8:         $B[i][code] \leftarrow B[i][code] \cup \{e\}$
 9: for all $i \in [0, r]$; $b \in B[i]$ do
10:     for all $e_1, e_2 \in B[i][b]$ do
11:         $cand \leftarrow cand \cup \{(e_1, e_2)\}$
12: for all $(e_1, e_2) \in cand$ such that $c(e_1, e_2) \geq \sigma$ do
13:     $\mathcal{E} \leftarrow \mathcal{E} \cup \{(e_1, e_2)\}$
14: return $\mathcal{G} \leftarrow$ CREATEGRAPH$(E, \mathcal{E})$
```

constituting dense subgraphs in $D$, retaining only a subset of pairwise dissimilar subgraphs according to the similarity threshold $\epsilon$.

### 4.3.1 Creation of the Correlation Graph

The correlation graph $\mathcal{G}$ can be built exactly, by computing the correlation $c(e_1, e_2)$ between each pair of edges $e_1, e_2 \in E$ and retaining those pairs satisfying $c(e_1, e_2) \geq \sigma$. However, when $D$ is large, comparing each pair of edges is prohibitively expensive, and thus we propose also an approximate solution based on *min-wise hashing* [BCFM00] which is described in Algorithm 8. Here we assume that a strong correlation between two edges implies a high Jaccard similarity, and thus, we use min-wise hashing to identify sets of candidate correlated edges. Specifically, we use a variant of the TAPER algorithm [ZF06], which repeats a min-wise hashing procedure $r$ times, each time using $h$ independent hash functions $h_i : T \to \mathbb{N}$. In each run, the algorithm computes $h$ hash values for each edge and concatenates them to create a *hash code* for the edge (line 7). For each edge $e$ and each hash function $h_i$, the hash value $H[e][i]$ is the minimum among the values $h_i(t)$, for timestamps $t \in T$ where $e$ exists, i.e., $\omega_t(e) = 1$. Note that, for efficiency purposes, the $r\,h$ hash values are computed all together by traversing once the set of edges of $D$ (lines 2–3). Edges with the same *hash code* are inserted into the same bucket (line 8) and the Pearson correlation is calculated for each pair of edges in the same bucket (line 12). If

**Algorithm 9** FINDMAXIMALCLIQUES

```
Input: Graph $\mathcal{G} = (E, \mathcal{E})$
Output: Set of maximal cliques $\mathcal{C}$
 1: $anc \leftarrow \emptyset$; $not \leftarrow \emptyset$; $cand \leftarrow E$
 2: $\mathcal{C} \leftarrow$ ENUMCLIQUES$(anc, cand, not)$
 3: return $\mathcal{C}$

 4: function ENUMCLIQUES$(anc, cand, not)$
 5:     $\mathcal{C} \leftarrow \emptyset$
 6:     if ISACLIQUE$(cand)$ then
 7:         return $\{anc \cup cand\}$
 8:     repeat
 9:         $v \leftarrow$ vertex with smallest degree in $cand$
10:         $nxAnc \leftarrow anc \cup \{v\}$
11:         $nxCand \leftarrow cand \cap adj[v]$
12:         $nxNot \leftarrow not \cap adj[v]$
13:         if $\nexists u \in nxNot$ s.t. $\forall w \in nxCand$ , $u \in adj[w]$ then
14:             $\mathcal{C} \leftarrow \mathcal{C} \cup$ ENUMCLIQUES$(nxAnc, nxCand, nxNot)$
15:         $cand \leftarrow cand \setminus \{v\}$
16:         $not \leftarrow not \cup \{v\}$
17:     until ISACLIQUE$(cand)$
18:     if $\nexists u \in not$ s.t. $\forall w \in cand$ , $u \in adj[w]$ then
19:         $\mathcal{C} \leftarrow \mathcal{C} \cup \{anc \cup cand\}$
20:     return $\mathcal{C}$
```

the correlation is greater than the correlation threshold $\sigma$, the pair is inserted in the edge set of the correlation graph $\mathcal{G}$ (line 13).

The algorithm requires two parameters that specify the number of runs $r$ to execute and the number of hash functions $h$ to use. Larger $r$ means less false negatives, and larger $h$ means more effective pruning.

### 4.3.2 Enumeration of the Maximal Cliques

After the creation of the correlation graph $\mathcal{G}$, the maximal groups of correlated edges are enumerated by identifying the maximal cliques in $\mathcal{G}$. To perform this enumeration we use our implementation of the *GP* algorithm of Wang et al. [WCH+17].

The algorithm proceeds by recursively partitioning the graph into two disjoint parts and examining each one independently using Procedure ENUMCLIQUES. In each step, the algorithm maintains three sets of vertices, $anchor$, $cand$, and $not$. The set $anchor$, initially empty, is recursively extended by adding a new vertex such that, at every step, the vertices in $anchor$ are

**Algorithm 10** FINDDIVERSEDENSEEDGES

```
Input: Dynamic Network $D = (V, E)$
Input: Set of maximal cliques $\mathcal{C}$
Input: Density function $\rho^k$
Input: Thresholds: Density $\delta$, Size $s_{\max}$
Input: Thresholds: Edges-per-snapshot $k$, Similarity $\epsilon$
Output: Set of diverse dense maximal subgraphs $\mathcal{S}$
 1: $\mathcal{S} \leftarrow \emptyset$; $\mathcal{P} \leftarrow \emptyset$
 2: $CC \leftarrow$ EXTRACTCC($\mathcal{C}$)
 3: for all $X \in CC$ do
 4:     if $X.size < s_{\max}$ and ISMAXIMAL($X$, $\mathcal{S} \cup \mathcal{P}$) and ISDIVERSE($X$, $\mathcal{S}$) then
 5:         $(flag, R) \leftarrow$ ISDENSE($D, X, k, \rho^k, \delta$)
 6:         if $flag = 1$ then
 7:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{X\}$
 8:         else if $flag = 0$ then
 9:             $\mathcal{P} \leftarrow \mathcal{P} \cup R$
10: for all $X \in \mathcal{P}$ do
11:     if ISMAXIMAL($X, \mathcal{S}$) and ISDIVERSE(X, $\mathcal{S}$) then
12:         $\mathcal{S} \leftarrow \mathcal{S} \cup \{X\}$
13: return $\mathcal{S}$

14: function ISDIVERSE($X, S$)
15:     if $S = \emptyset$ then return true
16:     for all $C_j \in S$ do
17:         if $|C_j \cap X| / |C_j \cup X| > \epsilon$ then return false
18:     return true
```

all connected in the input graph $\mathcal{G}$. The set $cand$, initially set to $E$, contains the vertices that can still be used to extend $anchor$, i.e., vertices that do not belong in $anchor$ and are connected to every vertex in $anchor$. Finally, the set $not$, initially empty, contains vertices already used as extensions for $anchor$ in the previous steps.

When examining the set $cand$ in a graph, the algorithm first checks if the vertices in $cand$ form a clique, and hence returns the maximal clique $cand \cup anchor$. Otherwise, the algorithm recursively takes the vertex with smallest degree in $cand$ (line 9), creates a new set $nxCand$ of all the vertices in $cand$ adjacent to $v$ (line 11), and updates $cand$ removing $v$ (line 15). Set $not$ is updated adding vertex $v$, as it cannot be used as a partitioning anchor anymore (line 16), and a new set *nxNot* is initialized adding all the vertices in $not$ adjacent to $v$ (line 12). If some vertex in *nxNot* is connected to all the vertices in $nxCand$, the recursive function is not called (line 13), because this means that the vertices $nxCand$ cannot generate a maximal clique. When $cand$ becomes a clique, the recursion stops (line 17) and the algorithm checks if the clique is maximal before inserting it into the output set (line 19).

**Algorithm 11** isDense

```
Input: Dynamic Network $D = (V, E)$
Input: A set of edges $X$
Input: Density function $\rho^k$
Input: Thresholds: Density $\delta$, Edges-per-snapshot $k$
Output: $(1, \emptyset)$ if $X$ is dense; $(0, R)$ if $X$ contains the dense subsets $R$; $(-1, \emptyset)$ otherwise
 1: $K \leftarrow$ kEdgeSnapshots$(X, k)$
 2: if $\rho^k(X, K, \delta)$ then return $(1, \emptyset)$
 3: if containsDense$(X, K, k) = \emptyset$ then
 4:     return $(-1, \emptyset)$
 5: return $(0,$ extractDense$(X, K, k))$

 6: function containsDense$(X, K, k)$
 7:     while $\rho^k(X, K, \delta/2) =$ false do
 8:         if $X = \emptyset$ or $K = \emptyset$ then return false
 9:         $max \leftarrow$ getMaxDeg$(X)$
10:         $n \leftarrow$ getMinDegNode$(X)$
11:         if $max < \delta/2$ then return false
12:         $X \leftarrow X \setminus adj(n)$
13:         $K \leftarrow$ kEdgeSnapshots$(X, k)$
14:     return true

15: function extractDense$(X, K, k)$
16:     $R \leftarrow \emptyset$; $Q \leftarrow \{(X, K)\}$
17:     while $Q \neq \emptyset$ do
18:         $(Y, K') \leftarrow Q.pop$
19:         if $\rho^k(Y, K', \delta)$ then
20:             $R \leftarrow R \cup \{Y\}$
21:         else if $K' \neq \emptyset$ then
22:             $max \leftarrow$ getMaxDeg$(Y)$
23:             $N \leftarrow$ getMinDegNodes$(Y)$
24:             if $max < \delta$ then
25:                 continue
26:             for all $n \in N$ do
27:                 $Y \leftarrow Y \setminus adj(n)$
28:                 $K' \leftarrow$ kEdgeSnapshots$(Y, k)$
29:                 if $Y \neq \emptyset$ then
30:                     $Q \leftarrow Q \cup \{(Y, K')\}$
31:     return $R$
```

### 4.3.3 Discovery of the Dense Subgraphs

The goal of this step is to find connected groups of edges that form a dense subgraph, using either $\rho^k_m$ or $\rho^k_a$ as density function $\rho^k$. Algorithm 10 receives as input a set of maximal cliques $\mathcal{C}$, each of which represents a maximal group of correlated edges. Since some of the edges in a clique may not be connected in the network $D$, the algorithm extracts all the distinct connected

**Algorithm 12** Density Functions

```
1: function isAvgDense($X, K, \delta$)
2:     if $K = \emptyset$ then return false
3:     let $H$ be the subgraph induced by $X$
4:     $d \leftarrow 1/K \sum_{t \in K} \rho(G_t(H))$
5:     return $d \geq \delta$

6: function isMinDense($X, K, \delta$)
7:     if $K = \emptyset$ then return false
8:     let $H$ be the subgraph induced by $X$
9:     for all $t \in K$ do
10:        if $\rho(G_t(H)) < \delta$ then return false
11:    return true
```

components from the cliques (line 2), before computing the density values. To allow a faster discovery of the maximal groups of dense edges, the connected components are sorted in descending order of their size and processed iteratively. If no larger or similar dense set of the current candidate component $X$ has been discovered yet (line 4), and if the size of $X$ does not exceed the threshold $s_M$, the density of $X$ is computed invoking Algorithm 12 (line 5). We recall that the similarity between two sets is calculated using the Jaccard similarity, and two sets are considered dissimilar if the similarity is below the threshold $\epsilon$. On the other hand, the threshold on the maximum size controls the size of the subgraphs in the result set $\mathcal{S}$, as well as the complexity of Algorithm 10.

Algorithm 11 describes the steps for determining if a set of edges $X$ is dense or if at least contains some dense subset. This algorithm uses either Procedure isMinDense or Procedure isAvgDense in Algorithm 12 to compute the density of $X$ and thus assessing if $X$ is dense. Procedure isAvgDense applies Equation (4.4) to determine if the average density of $X$ is above the threshold $\delta$. The average density of $X$ is computed as the average among the average node degrees of the subgraphs $H$ induced by $X$ in all the snapshots where at least $k$ edges of $X$ are present.

On the other hand, Procedure isMinDense computes Equation (4.3), which expresses the density of a group of edges as the minimum between the average degrees of the induced subgraph in all the snapshots where at least $k$ edges of the group are present, and then checks if its values is above $\delta$. The minimum average degree is computed by iterating over the set $K$ of snapshots at least $k$ edges of $X$ present (line 9). Nonetheless, we can stop the iteration as soon as we find a snapshot $t$ where the subgraph $H$ induced by $X$ in $t$ has average degree below the density threshold $\delta$ (line 10), because the minimum average degree is now guaranteed to be lower than

$\delta$. Thanks to the optimizations described in the next paragraph, the implementation of Procedure ISAVGDENSE is more efficient than that of Procedure ISMINDENSE, and thus we call the latter only when the former returns **true**, given that the average average degree is an upper bound on the minimum average degree.

When the density $\rho_a^k(H)$ ($\rho_m^k(H)$ respectively) of the subgraph $H$ induced by $X$ is above the threshold $\delta$, $X$ is inserted in the result set $\mathcal{S}$ (Algorithm 10 line 7). When the density is below the threshold $\delta$, the set $X$ is not dense; though some subset $X' \subseteq X$ may satisfy the condition $\rho_a^k(H') \geq \delta$. Since examining all the possible subsets of $X$ is a costly operation especially when the size of $X$ is large, Algorithm 11 uses Procedure CONTAINSDENSE, which is based on a 2-approximation algorithm for the densest subgraph problem [Cha00], to prune the search space. In details, Procedure CONTAINSDENSE iteratively removes the vertex with lowest degree from the induced subgraph $H$, until it becomes empty or its density is greater than $\delta/2$. Every time a vertex is removed, its outgoing edges are removed as well (line 12), and thus the set of valid snapshots $K$ must be updated (line 13). If $K$ becomes empty, any subset of $X$ will have zero density, and thus the algorithm returns **false** (line 8). If the maximum value of density calculated during the execution of this algorithm is below the threshold $\delta/2$, it holds that $X$ cannot contain a subset $X'$ with density above $\delta$ [Cha00], and thus CONTAINSDENSE returns **false**. Therefore, Procedure EXTRACTDENSE, which extracts all the dense subsets in the set $X$, is invoked (line 5) only when Procedure CONTAINSDENSE returns **true**.

When CONTAINSDENSE returns **true**, Procedure EXTRACTDENSE iteratively searches for all the dense subsets in $X$. At each iteration, a subset of edges $Y$ is extracted from the queue of candidates $Q$ and its density is checked. If $Y$ is not dense but the set of valid snapshots is not empty (line 21), a new candidate is created for each vertex $n$ with lowest degree in the subgraph induced by $Y$. These candidates are then inserted into $Q$. On the other hand, when $Y$ is dense, it is inserted into the result set $R$ returned at the end of the procedure. At the end of Algorithm 10, the maximal subsets in the set $\mathcal{P}$, which contains the elements of all the $R$ sets computed during the search, are checked for similarity with the subsets already in $\mathcal{S}$. Those with Jaccard similarity below $\epsilon$ with any subsets in $\mathcal{S}$ are finally added to $\mathcal{S}$ (lines 10–12).

### 4.3.4 An Efficient Way for Computing the Average Density

The average density $\rho_a(H)$ of a subnetwork $H = (V_H, E_H)$ in a dynamic network $D$ can be computed via the *summary graph* of $D$ defined as the *static graph* $\mathcal{R} = (V, E, \sigma)$ where $V$ is the set of vertices of $D$, $E$ is the union of the edges $E_i$ of all the snapshots of $D$, and $\sigma : E \mapsto \mathbb{R}$ is a weighting function that assigns, to each edge $e \in E$, a value equal to its average weight over all the snapshots of $D$, i.e., $\sigma(e) = 1/|T| \sum_{t \in T} \omega_t(e)$. The following proposition ensures that $\rho_a(H)$ is equivalent to the weighted density of $H$ in the summary graph $\mathcal{R}$, which is defined as $w\rho(H) = 2 \sum_{e \in E_H} \sigma(e)/|V_H|$.

**Proposition 4.5.** *Given a dynamic network $D$, its summary graph $\mathcal{R}$, and a subnetwork $H$, it holds that $\rho_a(H) = w\rho(H)$.*

*Proof.*

$$\rho_a(H) = \frac{1}{|T|} \sum_{t \in T} \rho(G_t(H)) = \frac{1}{|T|} \sum_{i \in T} \left( \frac{2|E_H \cap E_t|}{|V_H|} \right)$$

$$= \frac{2}{|V_H|} \frac{1}{|T|} \sum_{i \in T} \sum_{e \in E_H} \omega_t(e) = \frac{2}{|V_H|} \sum_{e \in E_H} \sigma(e) = w\rho(H)$$

□

The weighted density of $H$ in the summary graph $\mathcal{R}$ can be calculated significantly faster than its average density in the dynamic network $D$, since the former is obtained by summing the weights of the edges of $H$ defined by $\sigma$, while the latter is obtained by (i) constructing the subgraph induced by $E_H$ in each snapshot, (ii) computing the average node degree of each induced subgraph, and (iii) taking the average among those values. As a consequence, Proposition 4.5 allows us to improve the efficiency of our algorithm in solving Problem 2 in the case of the $\rho_a$ density function.

### 4.3.5 Complexity

The exact construction of $\mathcal{G}$ (Algorithm 8) takes $\mathcal{O}(|E|^2)$, as it requires the computation of all the pairwise edge correlations. On the other hand, the approximate solution creates $h \cdot r$ hash values for the edges in $\mathcal{O}(h \cdot r \cdot |E|)$ and compares only the edges that share at least one hash code. Even though the worst-case time complexity is still $\mathcal{O}(|E|^2)$ (every pair of edges share some hash code), the experiments show that the actual number of edge comparisons is much smaller than $|E|^2$. The time complexity of the maximal clique enumeration (Algorithm 9) is $\mathcal{O}\left(|E| \cdot \kappa(\mathcal{G})\right)$, where $\kappa(\mathcal{G})$ is the number of cliques in $\mathcal{G}$. The computation of the connected components in the maximal cliques takes again $\mathcal{O}(|E| \cdot \kappa(\mathcal{G}))$, as it requires a visit of the network $D$ for each clique. In the worst case, each edge of the network belongs to a different connected component, and thus Algorithm 10 must iterate $|E|$ times. At each iteration, it calls Procedure ISDENSE to compute the density of the current set of edges $X$ if its size is lower than $s_M$. Procedure ISDENSE calculates the average node degree of each subgraph induced by $X$ in all the snapshots where at least $k$ edges of $X$ are present (at most $|T|$), and thus its running time is bounded by $\mathcal{O}(s_M \cdot |T|)$. When $X$ is not dense, the algorithm further calls Procedure CONTAINSDENSE and Procedure EXTRACTDENSE. The former runs in $s_M$, since it removes at least one edge from $X$ at each iteration; while the latter must process all the subsets of $X$ in the worst case ($2^{s_M}$). The complexity of Algorithm 10 is therefore $\mathcal{O}(\kappa(\mathcal{G}) \cdot |E| + |E|(s_M \cdot |T| + s_M + 2^{s_M})) = \mathcal{O}(|E|(\kappa(\mathcal{G}) + s_M|T| + 2^{s_M}))$, which is also the complexity of Algorithm 7.

## 4.4 Experiments

We evaluate the performance of our exact and approximate solutions in terms of accuracy and execution time. The accuracy of the exact algorithm is tested on synthetic datasets, where ground-truth is known, while the approximate algorithm is compared against the exact. The scalability of the two approaches is evaluated using datasets of increasing size. Furthermore, we study the impact of all the parameters of the system on both the running time and the size of the output, hence steering non-expert users towards the right configuration for obtaining the desired output.

### 4.4.1 Datasets

We run experiments on both real and synthetic datasets of different sizes. In particular, we used 3 real networks and 12 randomly-generated networks. The characteristics of the synthetic datasets are listed in Table 4.2, while the characteristics of the real datasets are listed in Table 4.1. The table contains the number of vertices $|V|$, edges $|E|$, and snapshots $|\mathcal{T}|$; the minimum, average, and maximum node degree $deg(G)$; the minimum, average, and maximum node degree per snapshot $deg(G_i)$; and the minimum, average, and maximum number of appearances of an edge in the snapshots $count(e)$.

• HAGGLE [CHC+07] is a human-contact network where nodes represent persons and an edge between two persons indicate that there was a contact between them. The contacts were recorded by portable wireless devices that logged the MAC address of all the visible devices every 120 seconds, and then their timestamps were aggregated to form 90 network snapshots. This network was used to study the frequency and the duration of the human contacts with the goal of developing opportunistic forwarding algorithms.

• TWITTER [LGG18] is a hashtag co-occurrence network created using tweets collected from 2011 to 2016 related to the topics of gun control, abortion, and Obamacare. The tweets were restricted to users who tweeted about all three topics in this time span. We pruned hashtags by retaining only those with tf-idf value above $0.8$ and created an edge for each pair of unpruned hashtags appearing together in at least one tweet. By using a one-day granularity level we ended up with 2716 snapshots. Finally we extracted four samples of increasing size, denoted as TWITTER-S, TWITTER-M, and TWITTER-L, by limiting the set of nodes to the first 2, 8, and 12, highest-degree nodes, respectively, and keeping all the out-going edges and their destination nodes.

• MOBILE [Ita15] is a telecommunication network created using Call Detail Records collected from 2013-12-30 to 2013-12-31 and indicating calls made between Telecom Italia mobile users within a two-day period. The calls were spatially aggregated using a regular grid overlaid on the territory, to obtain a set of different areas, which constitute the nodes of our network. The edges indicate interactions between the areas. Each edge is associated with a timestamp indicating an interval start time, and the number of calls recorded in that interval. We built our dynamic networks MOBILE-S, MOBILE-M, and MOBILE-L by aggregating the edges by hour, computing the average strength for each pair of connected areas, retaining the edges with strength above the average, and finally extracting a connected component of size 42 K, 80 K, and 118 K respectively. The correlation between two edges is then computed by our algorithm using the vectors of strength values.

• GAUSSIAN-X-Y-Z are synthetic networks generated in Python using the *gaussian random partition graph* generator [BGW03] in the NetworkX library [1]. Given parameters $n$, $s$, $v$, $p_{in}$, and $p_{out}$, a Gaussian random partition graph is a connected graph obtained by partitioning the set of $n$ nodes into $k$ groups each of size drawn from a normal distribution $\mathcal{N}(s, s/v)$, and then adding intra-cluster edges with probability $p_{in}$ and inter-cluster edges with probability $p_{out}$. We created two sets of dynamic networks by setting $s = 10$, $v = 20$; and $n = |V|$, $p_{in}$, and $p_{out}$ as indicated in Table 4.2. For the first set of networks (top part of Table 4.2) we used a high

[1] https://networkx.github.io/documentation/stable/reference/generators.html

| | | | | $deg(G)$ | $deg(G_i)$ | $count(e)$ |
|---|---|---|---|---|---|---|
| *dataset* | $\lvert V\rvert$ | $\lvert E\rvert$ | $\lvert\mathcal{T}\rvert$ | min/avg/max | min/avg/max | min/avg/max |
| HAGGLE | 274 | 2K | 90 | 1/15.5/101 | 1/5.2/19.7 | 1/5.4/51 |
| TWITTER-S | 767 | 2K | 2K | 1/6.2/627 | 1/3/5.6 | 26/121.1/1853 |
| TWITTER-M | 1.2K | 7K | 2K | 1/12.1/641 | 1/3.2/7.4 | 26/86.4/1853 |
| TWITTER-L | 1.3K | 10K | 2K | 1/15.2/776 | 1/3.3/7.9 | 19/68.6/1853 |
| MOBILE-S | 5K | 42K | 48 | 1/15.3/4415 | 1.8/4.9/7.1 | 1/3.8/25 |
| MOBILE-M | 5K | 80K | 48 | 1/28.6/4418 | 1.7/6.4/10.6 | 1/3.6/25 |
| MOBILE-L | 5K | 118K | 48 | 1/41.4/4421 | 1.8/7.5/12.9 | 1/3.6/25 |

**Table 4.1:** Real datasets.

| | | | | | | $deg(G)$ | $deg(G_i)$ | $count(e)$ |
|---|---|---|---|---|---|---|---|---|
| *dataset* | $\lvert V\rvert$ | $\lvert E\rvert$ | $\lvert\mathcal{T}\rvert$ | $p_{in}$ | $p_{out}$ | min/avg/max | min/avg/max | min/avg/max |
| GAUSSIAN-1-7-1 | 100 | 1059 | 100 | 0.7 | 0.1 | 9/21.1/35 | 9.7/10.6/11.2 | 35/50/67 |
| | | | | | | | 3.37/10.32/17.14 | 38/48.2/64 |
| GAUSSIAN-2-7-1 | 200 | 3029 | 100 | 0.7 | 0.1 | 20/30.2/45 | 14.5/15.1/16 | 33/50.1/66 |
| | | | | | | | 9.8/15.1/19.6 | 33/50.1/68 |
| GAUSSIAN-3-7-1 | 300 | 6070 | 100 | 0.7 | 0.1 | 26/40.4/54 | 19.5/20.2/20.8 | 30/50/69 |
| | | | | | | | 16.3/20.3/23.4 | 30/50.3/68 |
| GAUSSIAN-1-7-3 | 100 | 1825 | 100 | 0.7 | 0.3 | 24/36.5/46 | 17.4/18.2/19.1 | 33/50/67 |
| | | | | | | | 12.5/18.2/23.1 | 34/49.9/66 |
| GAUSSIAN-2-7-3 | 200 | 6828 | 100 | 0.7 | 0.3 | 52/68.2/84 | 33.1/34/35.1 | 32/49.9/71 |
| | | | | | | | 28.4/33.8/39.7 | 30/49.6/68 |
| GAUSSIAN-3-7-3 | 300 | 14723 | 100 | 0.7 | 0.3 | 77/98.1/124 | 48/49/50 | 29/49.9/69 |
| | | | | | | | 44.4/48.9/53.1 | 31/49.8/69 |
| GAUSSIAN-1-3-2 | 100 | 1113 | 100 | 0.3 | 0.2 | 10/22.2/37 | 8.18/11/14.24 | 33/49.76/63 |
| GAUSSIAN-5-3-2 | 500 | 25450 | 100 | 0.3 | 0.2 | 74/101.8/132 | 49/50.8/52.9 | 31/49.9/69 |
| GAUSSIAN-10-3-2 | 1000 | 100700 | 100 | 0.3 | 0.2 | 160/201.4/238 | 99.3/100.6/102 | 28/49.9/74 |

**Table 4.2:** Synthetic datasets.

intra-cluster probability, with the goal of obtaining densely connected groups of edges that could serve as ground-truth; and two inter-cluster probabilities, with the goal of testing the accuracy of the algorithm under different levels of noise in the data. Moreover, we generated two different lists of existence strings for their edges, such that the first list associates strings highly correlated to edges in the same dense group, while the second list assigns mutually independent strings. The reason behind these two lists is that we wanted to test the correctness of our algorithm in discovering all the dense groups in the setting where their edges are correlated, and in returning no group when the edges are not correlated, even though it is highly dense. For each of these dynamic networks, Table 4.2 reports the statistics $deg(G_i)$ and $count(e)$ for both the list of correlated existence strings (top rule) and the list of independent strings (bottom rule). On the other hand, the purpose of the second set of networks (bottom part of Table 4.2) is to test the scalability of our framework in larger and sparser networks.

| | CIFORAGER | | | | | EXCODE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| *dataset* | $F_m$ | $F_a$ | | *time (m)* | | $F_m$ | $F_a$ | *time (s)* | |
| GAUSSIAN-1-7-1 | 0 | (0.073) | 0.026 | (18.6) | 0.22 | (0.98) 1 | (0.99) 1 | (1.14) | 0.71 |
| GAUSSIAN-2-7-1 | 0 | (0.033) | 0.012 | (365) | 3.94 | (0.00) 1 | (0.95) 1 | (2.22) | 1.57 |
| GAUSSIAN-3-7-1 | 0 | (-) | 0.007 | (-) | 44.00 | (0.98) 1 | (0.99) 1 | (8.03) | 4.47 |
| GAUSSIAN-1-7-3 | 0 | (0.021) | 0.009 | (78.3) | 0.88 | (0.98) 1 | (0.99) 1 | (1.18) | 0.91 |
| GAUSSIAN-2-7-3 | 0 | (-) | 0.003 | (-) | 332.00 | (0.97) 1 | (0.99) 1 | (10.1) | 5.52 |
| GAUSSIAN-3-7-3 | - | (-) | - | (-) | - | (0.98) 1 | (0.99) 1 | (50.9) | 23.40 |

**Table 4.3:** $F_m$, $F_a$, and running time of CIFORAGER and EXCODE in GAUSSIAN-X-7-Y using default parameters. Worst scores and corresponding running times are in brackets.

### 4.4.2 Experimental Setup

We implemented our algorithms in Java 1.8, and run the experiments on a 24 Cores (2.40 GHz) Intel Xeon E5-2440 with 188Gb RAM with Linux 3.13, limiting the amount of memory available to 150Gb. In addition, we implemented a Java version of CIFORAGER, which is the approach most related to our work, though it computes the edge similarities using the Euclidean distance instead of a correlation measure. This approach divides the sequence of snapshots into a sequence of overlapping windows of size $w_l$ and overlap $w_i$, and then performs a multi-objective clustering of the graph for each window. The multi-objective clustering is implemented by first clustering the edges using the edge similarity measure, and then clustering the groups found in the first step using a spatial distance. Finally, the regions found in the second step are merged to obtain regions that span multiple windows.

### 4.4.3 Effectiveness of the Exact Solution

We tested the effectiveness of our exact algorithm in detecting the actual dense groups of correlated edges in the synthetic networks GAUSSIAN-X-7-1 and GAUSSIAN-X-7-3, when their edges are correlated. To this aim, we set the correlation threshold $\sigma$ to the high value $0.8$; the density threshold $\delta$ equal to the minimum average degree among the actual dense groups, namely 2; and the maximum size $s_M$ to $\infty$ to ensure we do not miss any dense group. We measured the accuracy of the algorithm for a network $N$ in terms of the Jaccard similarity between the groups of edges discovered $\mathcal{S}$ and the dense groups in $N$, which constitute our ground-truth $\mathcal{G}$. In particular, for each group $H \in \mathcal{S}$, we computed the Jaccard similarity with its closest dense group in $\mathcal{G}$, and aggregate those values to obtain the average and minimum precision $P_a$ and $P_m$:

$$P_a = \frac{1}{|\mathcal{S}|} \sum_{H \in \mathcal{S}} \max_{J \in \mathcal{G}} \text{JACCARD}(H, J) \qquad P_m = \min_{H \in \mathcal{S}} \max_{J \in \mathcal{G}} \text{JACCARD}(H, J)$$

Then, for each dense group $J \in \mathcal{G}$, we computed the Jaccard similarity with its closest group in $\mathcal{S}$, and aggregated those values to obtain the average and minimum recall $R_a$ and $R_m$:

$$R_a = \frac{1}{|\mathcal{G}|} \sum_{J \in \mathcal{G}} \max_{H \in \mathcal{S}} \text{JACCARD}(H, J) \qquad R_m = \min_{J \in \mathcal{G}} \max_{H \in \mathcal{S}} \text{JACCARD}(H, J)$$

Finally, we calculated the average and minimum balanced F-score, as $F_a = 2(P_a \cdot R_a)/(P_a + R_a)$ and $F_m = 2(P_m \cdot R_m)(P_m + R_m)$. As shown in Table 4.3, for each synthetic network EXCODE obtained both $F_a = 1$ and $F_m = 1$, meaning that the algorithm correctly identified all the dense groups despite the extra edges added between the groups in the networks. We achieved lower scores only when using correlation thresholds $\sigma \leq 0.2$ for the smallest network, and $\sigma \leq 0.3$ for the others. In these particular cases, since the existence strings of the edges were generated at random, it is more likely that some inter-group edges have correlation greater than $\sigma$ with the edges in the group to which they are attached, and therefore the algorithm discovers sets of edges that are supersets of the actual dense groups. Nonetheless, the $F_a$ score is always greater than 0.94, while the $F_m$ score is lower than 0.97 only for network GAUSSIAN-2-7-1.

In the experiments where we associated mutually independent existence strings to the edges, we correctly returned none of the edge groups present in the networks, even though we used a threshold $\delta$ lower than the minimum density among those groups. Given the nature of these strings, it is unlikely that all the pairs of edges within a group are highly correlated, meaning that none of the dense groups of edges form a correlated group of edges.

Finally, we tested also the ability of CIFORAGER to detect the dense regions in the synthetic networks. We run the code with the default parameters indicated in the paper [CBLH12], i.e., $w_l = 10$, $w_i = 1$, clustering similarity threshold 0.25, and region similarity threshold 0.2. The goal of these experiments is to compare the results of CIFORAGER with those of EXCODE, and hence prove that our work is more effective in identifying the actual dense regions of correlation. Table 4.3 shows the minimum and average $F$ score achieved by the two approaches, as well as their running times. For EXCODE, it reports both the best and worst (in brackets) score obtained using the settings explained above. For CIFORAGER, it illustrates the scores achieved using $w_l = 10$ and $w_l = 100$ (in brackets). The symbol $-$ indicates that the algorithm was

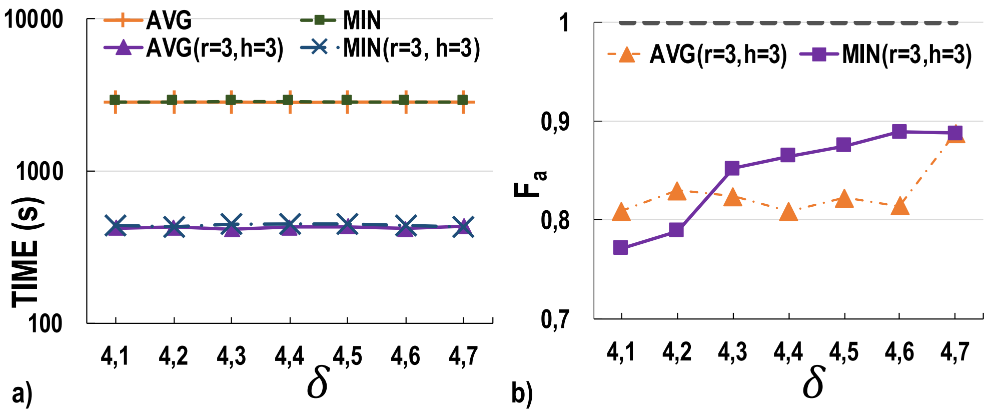


**Figure 4.1:** Running time of EXCODE and EXCODE *approximate* in MOBILE-M (a) and $F_a$ score (b), varying $\delta$, and using $r = 3$, $h = 3$, $\sigma = 0.9$, $\rho_a^k$ (AVG) and $\rho_m^k$ (MIN).

not able to terminate within 2 days. Since the output of CIFORAGER contains also edges that are not part of any dense group, the average and minimum precisions $P_a$ and $P_m$ are always low, and, as a consequence, the $F_a$ and $F_m$ are always lower than those of EXCODE. We note that, since the spatial distances are computed in the summary graph created by collapsing the snapshots of the window, the $F$ scores are worse when using $w_l = 100$. We recall that CIFORAGER creates a partitioning of the input graph, meaning that, especially when the edges have low temporal similarity, the size of its output can be very large and many of the regions discovered are very small. This situation happens also when using larger clustering similarity thresholds, as the algorithm aggregates a lower number of edges in the same cluster, and hence creates a larger number of small-sized clusters of edges. As an example, in GAUSSIAN-1-7-1 using $w_l = 10$, it found 372 regions of size 1, 123 of size 2, while the average and maximum size of a region are 122.8 and 216, respectively. Similarly, in GAUSSIAN-1-7-3, 1260 of the 2699 regions discovered are size-1 regions. Similar results were found using $w_l = 100$.

Regarding the performance, the most expensive task is the computation of the temporal distances, and since this operation is performed for almost each pair of edges and for each window, the running time increases with both the network and the window size. With window size $w_l = 10$, the algorithm terminated in 18 minutes in GAUSSIAN-1-7-1, and in 1.3 hours in GAUSSIAN-1-7-3. The time required to process each window was roughly the same, and hence, with $w_l = 100$ it terminated in 13.7 and 53.31 seconds, respectively. In contrast, our algorithm was able to terminate in less than a minute with every configuration and network tested.

Since these experiments prove that CIFORAGER is not able to solve DICORDIS, we do not present further experiments on its performance in the following paragraphs.

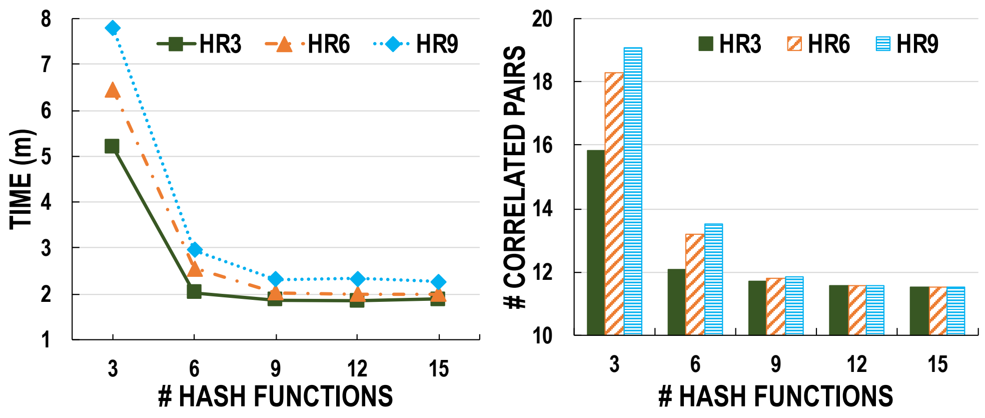


**Figure 4.2:** Tuning of the min-wise hashing parameters $r$ (HR) and $h$ (HASH FUNCTIONS) running EXCODE *approximate* in TWITTER-L.

### 4.4.4 Effectiveness of the Approximate Solution

Our approximate algorithm trades accuracy for performance by approximating the set of $\sigma$-correlated edges. As described in Section 4.3, the approximate set is obtained by the min-wise hashing technique with parameters $r$ and $h$. The parameter $r$ indicates the number of repetitions, so larger values of $r$ increase the quality of the result, at the cost of additional computation. For the number of hash functions $h$, larger numbers generate more informative hash codes, meaning that the algorithm will cluster the edges into smaller groups. As the number of comparisons decreases, the running time decreases as well.

We tested different combinations of $(r, h)$, starting from $h = r = 3$ and increasing their value up to $h = r = 15$, and counted the number of $\sigma$-correlated edges discovered. Figure 4.2 shows that after a period of decrease, the number of correlated pairs reached a plateau at $h = 9$, thus prompting us to use combinations of small values for both $r$ and $h$.

### 4.4.5 Efficiency of the Approximate Solution

Figure 4.1 shows the performance and running time of EXCODE *approximate* to find the $(0.9)$-correlated $\delta$-dense subgraphs in the MOBILE-M network, using both $\rho_a^k$ and $\rho_m^k$, and varying $\delta$. As we can see, the approximate solution is one order of magnitude faster than the exact algorithm, and yet achieves a $F_a$ score of at least $0.8$ for the $\rho_a^k$ case, and $0.77$ for the $\rho_m^k$. We observed a similar behavior also in the other networks. As an example, in the TWITTER samples we obtained the highest $F_a$ score at high density values, while a minimum of $0.63$ at low density values.

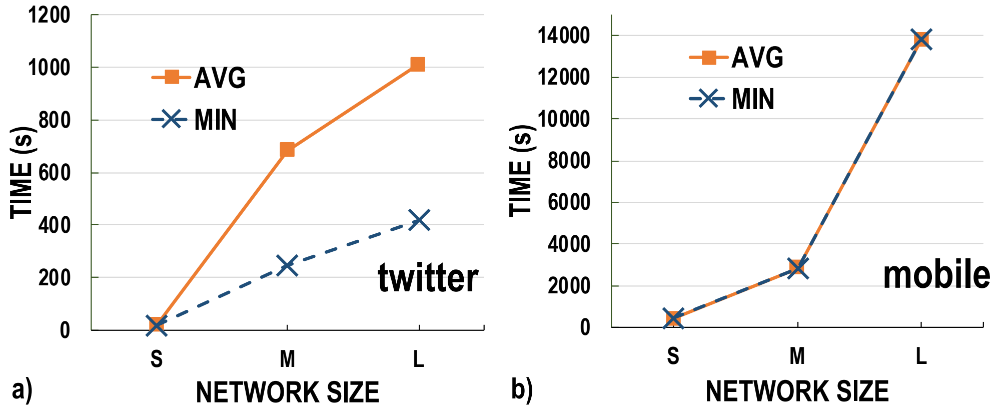


**Figure 4.3:** Scalability of EXCODE in TWITTER with $\sigma = 0.7$ and $\delta = 2.6$ (a), and in MOBILE with $\sigma = 0.9$ and $\delta = 4.1$ (b), using both $\rho_a^k$ and $\rho_m^k$.

### 4.4.6 Scalability

We tested the scalability of EXCODE using samples of increasing size, extracted from both the TWITTER and the MOBILE network. In the TWITTER-X samples we set $\sigma = 0.7$ and $\delta = 2.6$, while in the MOBILE-X samples we used $\sigma = 0.9$ and $\delta = 4.1$. Figure 4.3 shows that the running time increases exponentially for both $\rho_a^k$ and $\rho_m^k$ in the MOBILE-X samples (b), while it increases slightly slower for $\rho_m^k$ in the TWITTER-X samples. This exponential growth is due to the $\mathbf{NP}$-complete nature of the candidate generation task, which requires the enumeration of the maximal cliques in the correlation graph. Nonetheless, these cliques can be stored and used for all the experiments where $\sigma$ is kept fixed, hence allowing a significant speed up in the performance when the user is interested in examining different combinations of the other parameters. We also note that the different behavior of $\rho_m^k$ in the TWITTER-X samples is mainly due to the sparsity of the snapshots in TWITTER and the early pruning strategy in Algorithm 11 line 10 that discard a candidate group as soon as it finds a snapshot in which it is not dense. Finally, we mention that, while EXCODE terminated in under 12 minutes in the TWITTER-M sample, CIFORAGER took 2.6 minutes per window (for a total of 89h using $w_l = 10$) to produce 12296 results.

### 4.4.7 System Parameters

The system parameters are the density threshold $\delta$, the correlation threshold $\sigma$, and the edges-per-snapshot threshold $k$. Setting their value can be a hard task, especially if the user is exploring an unfamiliar dataset and thus does not know what she can expect from it. A suitable value for

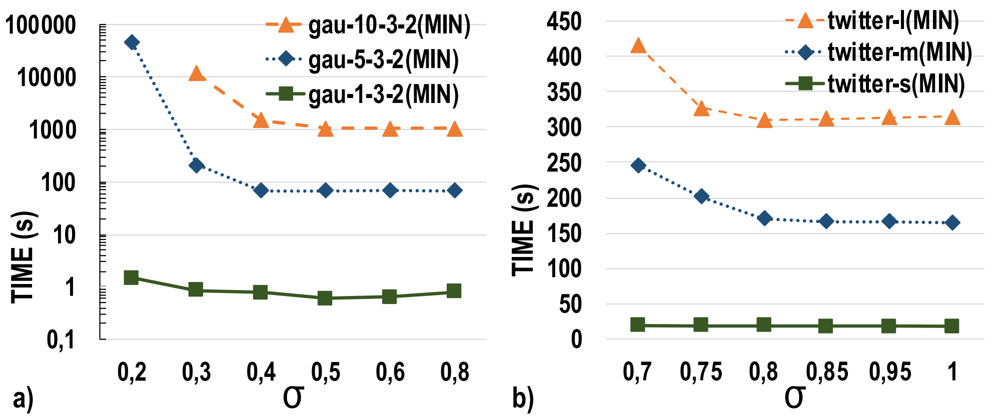


**Figure 4.4:** Running time of ExCoDE in the synthetic networks GAUSSIAN-X-3-2 with $\delta = 2.5$ and in the TWITTER-X samples with $\delta = 2.6$, using $\rho_m^k$, varying $\sigma$.

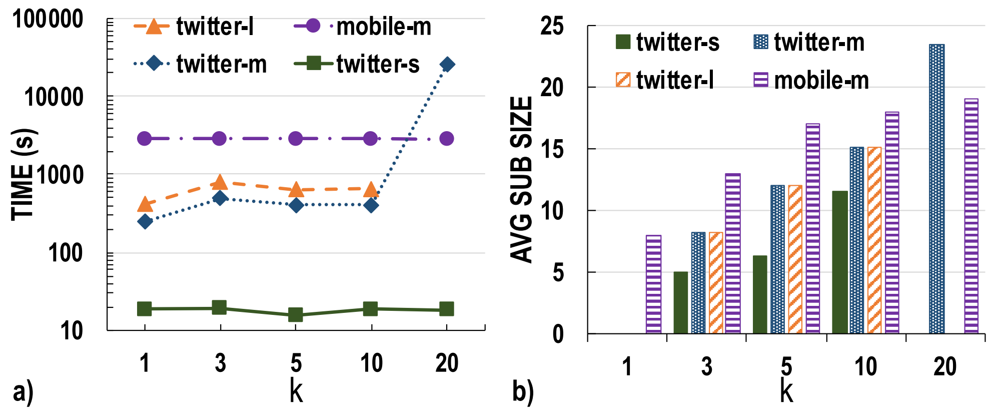


**Figure 4.5:** Running time of ExCoDE (a) and average subgraph size (b) with $\delta = 3$, $\sigma = 0.7$ in the TWITTER-X samples, and $\delta = 4.7$, $\sigma = 0.9$ in MOBILE-M, using $\rho_m^k$, varying $k$.

$\delta$ can be discovered by initializing its value to the maximum degree over the snapshots of the network and decreasing it until the size of the result and/or the size of the subgraphs in the result set meet the user's demands. Indeed, the minimum degree in a snapshot is an upper-bound on the minimum degree of a subgraph in the snapshot, and thus on its $\rho_m$ and $\rho_a$ density. The user can benefit from this strategy as it allows to discover the tightest groups first, which can be considered the most interesting. Regarding the performance of ExCoDE at different values of $\delta$, note that thanks to the size threshold $s_M$ that limits the candidates to explore, $\delta$ affects significantly the running time only at very low values close to 2, and in particular, in the case of sparse snapshots. In fact, in these cases Procedure EXTRACTDENSE must generate a larger number of candidate subsets before finding one that is dense or reaching the terminating condition.

Setting a value for $\sigma$ can be much harder, as the probability of finding $\sigma$-correlated groups of edges in a graph depends on the edge distribution over the snapshots, which is usually unknown.

Our strategy consists in starting with a high value and decreasing it until the result set is not empty. The main advantage of such a technique is that the first subgraphs found are surely the most significant. In addition, since the number of correlated pairs is monotonically non-decreasing with the decrease of $\sigma$, the running time at lower thresholds is longer than that at higher thresholds. Therefore, starting the search at higher thresholds allows to set a lower bound on the complexity of the first task. As an example, Figure 4.4 shows that for large networks, the running time increases exponentially with the decrease of $\sigma$, as the number of candidate edge pairs is $\mathcal{O}(|E|^2)$ and thus the size of the correlation graph is $\mathcal{O}(|E|^2)$. We note that the algorithm was not able to terminate within 24 hours, when running on the largest synthetic dataset with $\sigma = 0.2$, as the number of candidate edge pairs was in the order of 10 millions. We obtained similar results also in the case of $\rho_a^k$.

Finally, the parameter $k$ affects the computation of the average and minimum density of a subgraph $H$, as only the average degree values computed in snapshots where at least $k$ edges of $H$ are present, are considered in $\rho_a^k(H)$ and $\rho_m^k(H)$. In the snapshots where less than $k$ edges exist, the average degree is lower than that in the other snapshots, and therefore higher values of $k$ leads to higher values of $\rho_a^k$ and $\rho_m^k$. However, setting the value of $k$ too high can be too restrictive in some cases, because the algorithm will discard all the candidates whose edges do not appear in groups of $k$, which, however, can be dense correlated subgraphs for some values of $\delta$ and $\sigma$. To avoid missing any highly dense but rare subgraph, in our experiments we always used a threshold greater than 0, and to avoid discarding any interesting subgraph, we always started our exploration with $k = 1$, and then increased its value if the algorithm was not able to find any result. This was the case, as illustrated in Figure 4.5, when we used $\rho_m^k$ in the TWITTER samples. The TWITTER network is characterized by highly sparse snapshots, and thus the minimum density over all the snapshots is 0 for most of the candidates. By increasing $k$, we were able to discover an increasing number of results for all the networks. As an example, with $k = 1$, $\delta = 3$, and $\sigma = 0.7$, we discovered no dense correlated group of edges, while with $k = 5$, in TWITTER-L we found as much as 50 groups. We note that, due to the reasons explained above, also the average size of the results increased with $k$, with the only exception of TWITTER-S, which is too small to contain large dense groups of correlated edges.

| | $\rho_a^k$ | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $Red(\mathcal{S})$ | | | $Rps(\mathcal{S},\mathcal{C})$ | | | $\|\mathcal{S}\|/\|\mathcal{C}\|$ | | |
| $\delta$ | 0.25 | 0.5 | 0.75 | 0.25 | 0.5 | 0.75 | 0.25 | 0.5 | 0.75 |
| 3.7 | 0.019 | 0.214 | 0.444 | 0.693 | 0.831 | 0.893 | 0.0005 | 0.0015 | 0.0110 |
| 3.9 | 0.020 | 0.192 | 0.432 | 0.690 | 0.802 | 0.894 | 0.0059 | 0.0138 | 0.0909 |
| 4.1 | 0.000 | 0.218 | 0.370 | 0.750 | 0.925 | 0.962 | 0.0168 | 0.0420 | 0.1764 |

**Table 4.4:** Redundancy, representativeness, and size ratio of the result set in HAGGLE, varying $\epsilon$ and $\delta$, using $\rho_a^k$ and $\sigma = 0.6$.

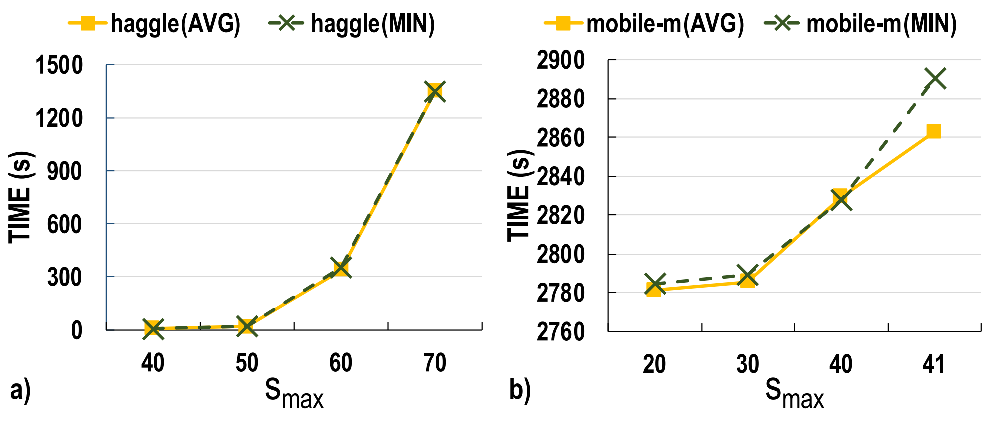


**Figure 4.6:** Running time of ExCoDE in HAGGLE with $\sigma = 0.6$ and $\delta = 3.7$ (a), and in MOBILE-M (b) with $\sigma = 0.9$ and$\delta = 4.1$, varying $s_max$, and using both $\rho_a^k$ and $\rho_m^k$.

### 4.4.8 Application-specific Parameters

A major challenge in enumeration problems is the possible large set of solutions that satisfy the desired specifications. To avoid overwhelming the end user with too many results to analyze, we introduced in our problem formulation the parameter $\epsilon$, which controls the level of redundancy and representativeness of the result set $\mathcal{S}$ computed by ExCoDE. We calculate the redundancy of $\mathcal{S}$ ($Red(\mathcal{S})$) in terms of the average pairwise Jaccard similarity between its elements, and the representativeness of $\mathcal{S}$ with respect to the complete set of solutions $\mathcal{C}$ ($Rps(\mathcal{S},\mathcal{C})$) as the percentage of edges in $\mathcal{C}$ that are present in $\mathcal{S}$. Table 4.4 shows the values calculated for the network HAGGLE using $\rho_a^k$, and varying both $\epsilon$ and $\delta$. We do not report the values for the density measure $\rho_m^k$, as they are consistent to those for $\rho_a^k$. It is interesting to notice that even using large values of $\epsilon$, the size of $\mathcal{S}$ is less than one tenth the size of the complete set, which is $5060$ at its highest value, and yet the representativeness of $\mathcal{S}$ almost reaches the best score. In addition, since an increase in the density threshold naturally leads to a decrease in the number of qualified subgraphs, the redundancy score and the size of the complete set decrease as well, meaning that the relative redundancy and the relative size of $\mathcal{S}$ get higher values. Finally, we note that the redundancy score does not follow a clear pattern, taking low values both at lower densities and at higher density. The main reason is that higher density thresholds generate less solutions that tend to be far apart in the network, while at lower thresholds, the union of small dense groups of edges close in the network is likely to form a dense subgraphs, which is the only solution returned due to the maximality constraint.

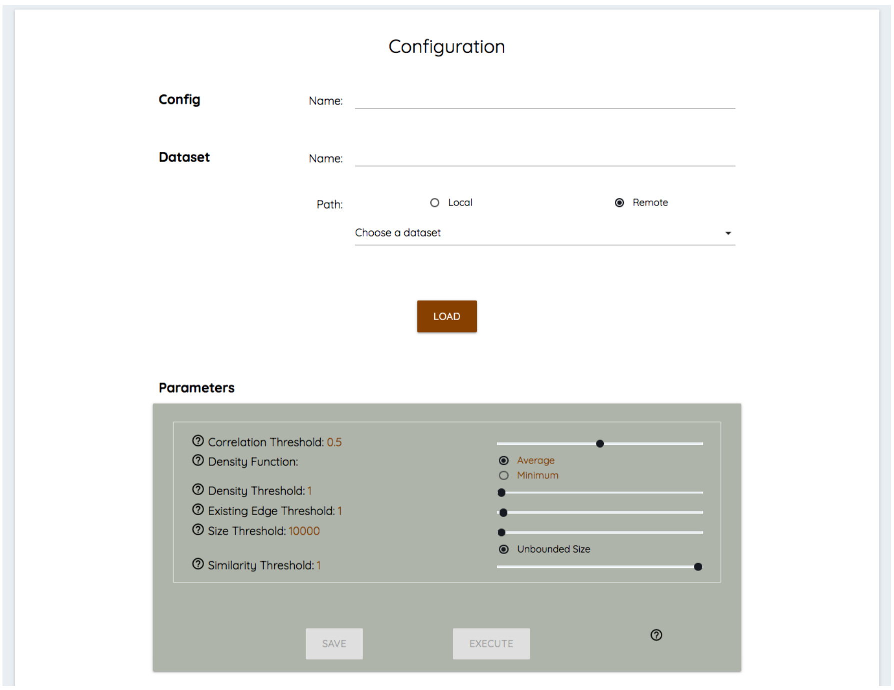
Another application-specific parameter that can be set by the users is $s_M$, which limits the candidate groups of edges that are processed by EXCODE to those having size smaller than $s_M$. As a consequence, low values of $s_M$ can significantly prune the subgraph search space, hence boosting the performance of the algorithm. Figure 4.6 shows that the running time increases almost exponentially for both the smaller network HAGGLE (a) and the larger MOBILE-M (b), and for both density measures $\rho_a^k$ (AVG) and $\rho_m^k$ (MIN). This result proves the importance of a proper tuning of this parameter to avoid incurring in a degradation of performance.

## 4.5 Displaying the Regions of Correlation

In this section, we present a visualization tool, called EXCODE-VIZ, that runs on top of the EXCODE framework and provides an easy-to-understand user interface. Its aim is twofold. First, it gives the users the possibility of characterizing, by means of a detailed configuration panel, the dense correlated subgraphs to extract from the input network, so that they better match their expectations. Secondly, it allows the users to visualize, by means of an interactive panel, where those subgraphs are located, how strong they are, and how they are related with each other. In particular, it facilitates the analysis of the network and the events happening in it, by giving the users the possibility of investigating how the regions behave over time, how their density changes, and in which snapshots they are active.

### 4.5.1 Applications

This tool is directed at any data practitioner who needs a tool to visualize and interact with large dynamic networks, and would like to understand how the network evolves, or more specifically, identify unexpected and significant events happening in the network. It is also interesting for any data engineer who needs an effective tool to detect correlated anomalies in the network under her supervision, and any data researcher who wants to understand the challenges behind understanding how a given dynamic network behaves over time, and how its changes are related to each other.



Configuration

**Config** Name:

**Dataset** Name:

Path: Local [✓ Remote]

Choose a dataset

LOAD

**Parameters**

Correlation Threshold: 0.5

Density Function: [✓ Average] Minimum

Density Threshold: 1

Existing Edge Threshold: 1

Size Threshold: 10000 [✓ Unbounded Size]

Similarity Threshold: 1

SAVE EXECUTE

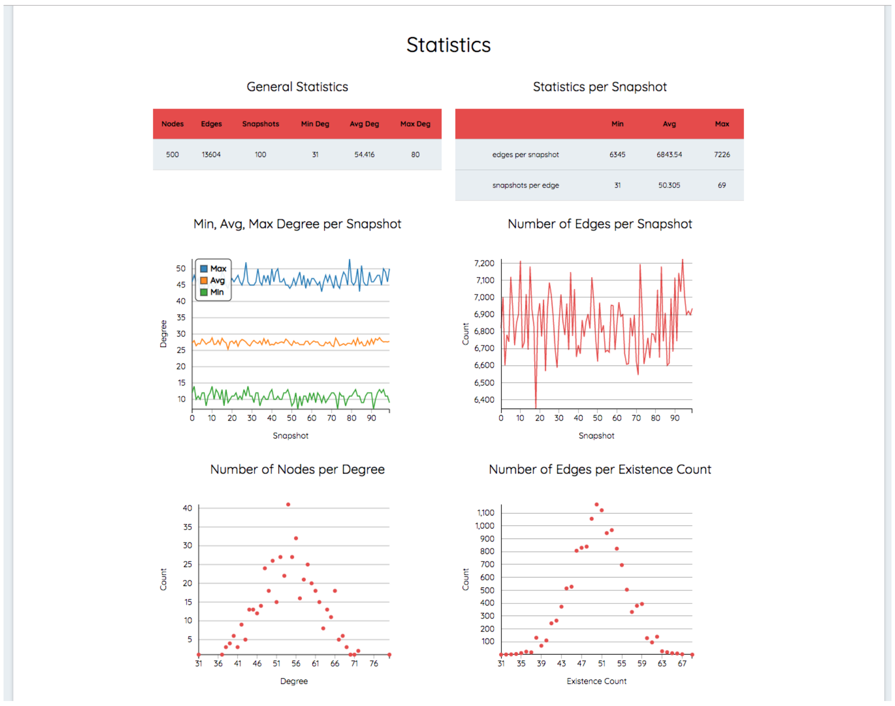
**Figure 4.7:** Dataset selection.

### 4.5.2 Scenario

To showcase the unique features and capabilities of ExCoDE-Viz, we simulate a practical scenario. The simulation starts at the configuration screen (Figure 4.7), where the user can select the dataset and configure the parameters. The first step is thus to load the dataset and the mapping file containing the node labels. For this scenario, we created a dynamic network from the Border Gateway Protocol (BGP) Internet Routing topology, modeling the Autonomous Systems (ASs) on the Internet as nodes, and the routing paths in the *PATH* entries of the AS routing tables as edges between ASs. We considered the routing tables collected by the RouteViews project[2] from August 29 to August 31. In these days, Hurricane Katrina hit Florida and Louisiana, causing major connectivity failures, and hence changes in the BGP topology network.

By clicking the *LOAD* button, the dataset is uploaded to the server, and a visualization of its main characteristics is presented to the users (Figure 4.8). Two tables show the minimum, average, and maximum node degree in the *union graph* created by joining the snapshots of the network; and the minimum, average, and maximum node degree in a snapshot. On the other hand, the four charts display the number of edges per snapshot, the degree distribution, and the edge distribution.

[2] `http://www.routeviews.org`



## Statistics

General Statistics

| Nodes | Edges | Snapshots | Min Deg | Avg Deg | Max Deg |
|---|---|---|---|---|---|
| 500 | 13604 | 100 | 31 | 54.416 | 80 |

Statistics per Snapshot

| | Min | Avg | Max |
|---|---|---|---|
| edges per snapshot | 6345 | 6843.54 | 7226 |
| snapshots per edge | 31 | 50.305 | 69 |

Min, Avg, Max Degree per Snapshot
Max
Avg
Min
Degree
Snapshot
Number of Edges per Snapshot
Count
Snapshot
Number of Nodes per Degree
Count
Degree
Number of Edges per Existence Count
Count
Existence Count

**Figure 4.8:** Dataset statistics.

The insights provided by the charts, together with the information included in the configuration card shown in Figure 4.7, can guide the users towards configuring the parameters of the system according to their desires. These parameters involve desired levels of density, correlation, size, activity, and redundancy in the results. The question marks on the left of the parameter names explain the role of the parameters and how their values affect the output, while the sliders on the right indicate what values the parameters can take, and hence they let the users select appropriate values only.

Once the parameters are configured, we can save the configuration for later use by clicking the *SAVE* button, and then, we can execute the mining algorithm by clicking the *EXECUTE* button. Once the results are available, an interactive panel appears in the result screen (Figure 4.9). This panel shows the union graph with the dense groups of correlated edges highlighted using different colors. As indicated in the legend in the bottom right corner, denser subgraphs are colored in darker colors, while nodes belonging to multiple subgraphs are indicated in black, and nodes that are not part of any dense subgraph are in white. The edges in each group are routing paths that changed in a similar way over time, and that were topologically close in the snapshots where the group was active, and therefore they are likely to represent an instability region originated from the failure of the same router during the time of the landfall of Hurricane Katrina.

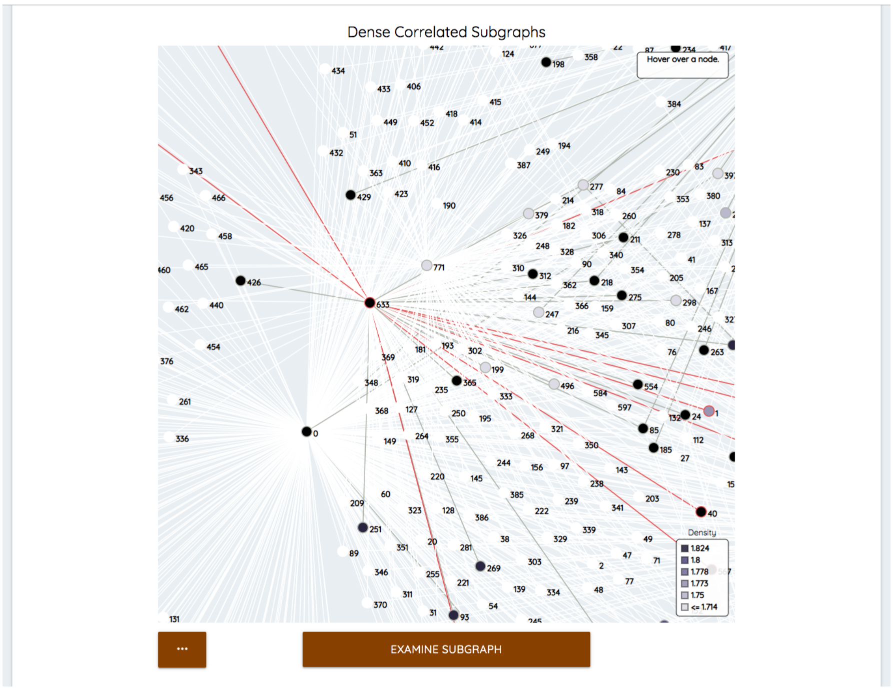
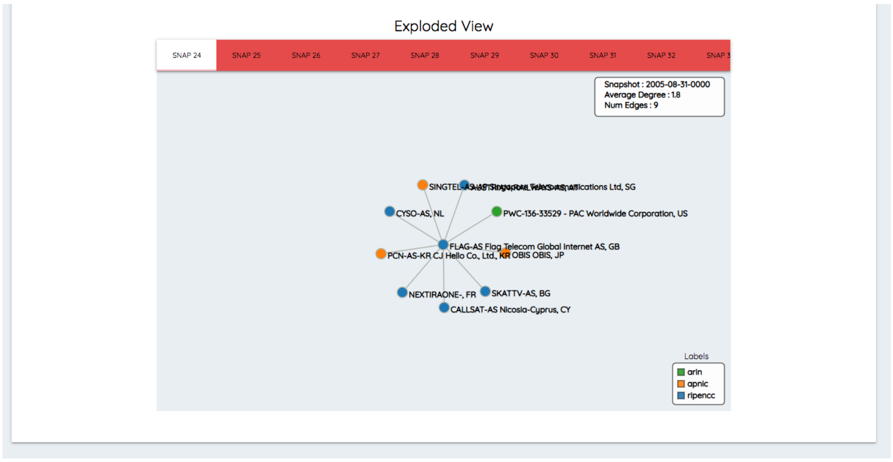


**Figure 4.9:** Dense correlated subgraphs.


Exploded View
SNAP 24
SNAP 25
SNAP 26
SNAP 27
SNAP 28
SNAP 29
SNAP 30
SNAP 31
SNAP 32
SINGTEL-AS Singapore Telecommunications Ltd, SG
CYSO-AS, NL
PWC-136-33529 - PAC Worldwide Corporation, US
FLAG-AS Flag Telecom Global Internet AS, GB
PCN-AS-KR CJ Hello Co., Ltd., KR
OBIS OBIS, JP
NEXTIRAONE-, FR
SKATTV-AS, BG
CALLSAT-AS Nicosia-Cyprus, CY
Labels
🟩 arin
🟧 apnic
🟦 ripencc

Snapshot : 2005-08-31-0000
Average Degree : 1.8
Num Edges : 9

**Figure 4.10:** Exploration of a dense subgraph.

The users can interact with the graph to better understand its structure. For example, by hovering over a node, they can see to which subgraphs the node belongs, and by clicking on it they can select that or those subgraphs. In addition, they can drag the nodes, and zoom in and out of the graph. Further information on the dense subgraphs is provided in a separate hidden panel that can be opened by clicking the dotted button. This panel includes a list of all the dense subgraphs, with the ids of all the nodes and edges in each of them, so that the users can understand where the regions of correlation are located in the network, as well as how they are related with each other.

Finally, the users can explore and analyze a dense subgraph in isolation, by selecting it in the main panel and clicking the *EXAMINE SUBGRAPH* button. A separate panel (Figure 4.10) shows how the subgraph changes over time, how many edges appears in each snapshot where the subgraph is active, and the average degree. Different colors are used to highlight the different types of nodes, according to the mapping file uploaded with the dataset. For our example, the colors indicate the regional Internet address registries for the five geographical areas: Asia-pacific, North America, South America, Europe, and Africa. Thanks to this tool, a network analyst can see which countries were affected by the disaster and which issues were caused by the same root cause, sparing him the trouble of examining each node in the network.

Chapter 5

# Entity Resolution in Temporal Databases

In this chapter we introduce the novel problem of detecting which records in a heterogeneous temporal database model the same real-world entity. To this aim, we define a similarity measure between heterogeneous temporal records, which can detect matching records under the assumption that two records are similar if they are characterized by similar attribute values over time. Then, we present an exact and an approximate algorithm that adopt this measure to first detect pairs of matching records, and then generate maximal groups of matching records.

## 5.1 Contributions

The contributions of this chapter can be summarized as follows:

- We formally define the notion of *temporal record* in the context of heterogeneous data, and introduce the novel problem of entity resolution in heterogeneous temporal databases, as the task of detecting which temporal records in the database refer to the same real-world entity (Section 5.2).
- We propose a time-aware schema-agnostic similarity measure to assess the similarity between temporal records characterized by different sets of attributes (Section 5.3). To overcome such heterogeneity, this measure compares the attribute values of each instance of the two temporal records, independently of the corresponding attribute names.
- We present an exact solution to cluster the records into groups with intra-class similarity greater than a minimum threshold, which compares each pair of records and extracts all

the maximal cliques from the similarity graph built using the pairs of similar records (Section 5.4).

- Given the hardness of the problem, we propose an approximate solution based on a time-aware schema-agnostic meta-blocking algorithm, which gains efficiency by reducing the number comparisons, yet it can provide good-quality results (Section 5.5).

## 5.2 Problem Formulation

Let consider a finite set of real-world entities $\mathcal{E}$ not known a-priori, where each entity can be described by a set of attributes taken from a finite set of attributes $\mathcal{A}$ and values taken from a finite domain of values $\Sigma$. We consider the setting where both the set of attributes and their values can change over time, and in particular, we consider a collection of temporal records describing an entity in $\mathcal{E}$ over time. Each temporal record is identified by an unique id, and the status of the record at a given time is called *record instance*.

**Definition 5.1** (Record Instance)**.** Let $\Sigma$ be a finite domain of values, $\mathcal{A}$ a finite domain of attributes $\mathcal{A}$, $\mathcal{O}$ a finite domain of record ids, $\mathcal{T}$ a finite domain of time instances, and $\phi$ : $\mathcal{A} \times \mathcal{O} \times \mathcal{T} \to \Sigma \cup \{\perp\}$ a function assigning a value in $\Sigma$ to attributes in $\mathcal{A}$, for records with id in $\mathcal{O}$, at time instances in $\mathcal{T}$. The *instance* of record $i \in \mathcal{O}$ at time $t \in \mathcal{T}$ is a tuple $r_i^t = \langle i, t, \Phi_i^t \rangle$ with $\Phi_i^t = \{(a, \phi(a, i, t)) \,|\, a \in \mathcal{A} \wedge \phi(a, i, t) \neq \perp\}$.

Given that $\phi(a, i, t) = \perp$ indicates that record $i$ does not possess attribute $a$ at time $t$, $\Phi_i^t$ is the set of all the attribute-value pairs that record $i$ has at time $t$. If record $i$ does not exist at time $t$, we set $\Phi_i^t = \emptyset$. In addition, we denote as $\Phi_i^t \lceil v$ the set of attribute values of record $i$ at time $t$.

Then, we define a *temporal record* as an ordered sequence of record instances with the same id, i.e., $r_i = \langle r_i^{t_{i_1}}, \ldots, r_i^{t_{i_n}} \rangle$, such that $j < k \to t_j < t_k$ and $\forall r_i^{t_j}$ , $\Phi_i^{t_j} \neq \emptyset$. When there is no ambiguity, we call a temporal record a *record* for simplicity. We call *temporal database* a set of temporal records:

**Definition 5.2** (Temporal Database)**.** A *temporal database* $\mathcal{D}$ is a set of temporal records, i.e., $\mathcal{D} = \{r_i \,|\, i \in \mathcal{O}\}$. We call *snapshot* of $\mathcal{D}$ at time $t$, the set of instances of all the records that exist at time $t$, i.e., $D^t = \{r_i^t \,|\, \exists r_i \in \mathcal{D} \,.\, r_i^t \in r_i\}$.

In this work, we consider the task of entity resolution in heterogeneous temporal databases, which is the problem of detecting which records in a heterogeneous temporal database $\mathcal{D}$ refer to the same real-world entity in $\mathcal{E}$. Let indicate the relationship between a record $r$ and the entity $e$ it models by $r \mapsto e$, i.e., $\forall r \in \mathcal{D}\,,\, \exists e \in \mathcal{E}\,.\, r \mapsto e$. Then, we formally define our problem as follows.

**Problem 3** (Temporal ER)**.** *Given a finite set of entities $\mathcal{E}$ and a temporal database $\mathcal{D}$, group the records in $\mathcal{D}$ into a set of clusters $\{E_1, \ldots, E_k\}$, such that every pair of records $r_i, r_j$ in the same cluster model the same entity, and each pair of records in different clusters model different entities, i.e.,*

- $\forall l \in [1, k]\,,\, \forall r_i, r_j \in E_l\,,\, \exists e \in \mathcal{E}\,.\, r_i \mapsto e \wedge r_j \mapsto e$*;*
- $\forall l \neq s \in [1, k]\,,\, \forall r_i \in E_l, r_j \in E_s\,,\, \exists e, e' \in \mathcal{E}\,.\, r_i \mapsto e \wedge r_j \mapsto e' \wedge e \neq e'$*.*

We assume that records modeling the same real-world entity are characterized by similar attributes values, and hence, given a similarity measure for temporal records $sim$ and a temporal similarity threshold $\tau$, we conclude that two temporal records $r_i$ and $r_j$ model the same entity $e \in \mathcal{E}$ if $sim\,(r_i, r_j) \geq \tau$. As a result, Problem 3 can be restated as the problem of finding a set of clusters $\{E_1, \ldots, E_k\}$ such that $\forall l \in [1, k]\,,\, \forall r_i, r_j \in E_l\,,\, sim\,(r_i, r_j) \geq \tau$.

In the next section we introduce a novel function to measure the similarity between temporal records, which can be effectively used to solve Problem 3.

## 5.3 Similarity Between Temporal Records

We design our similarity measure based on the intuition that two temporal records are more likely to refer to the same real-world entity if they frequently display similar attribute values, meaning that a sufficient number of instances of the two records match. Given an attribute similarity threshold $\epsilon$, we say that two record instances $r_i^t$ and $r_j^t$ match, if the Jaccard similarity $JS$ between the corresponding sets of attribute-value pairs is greater than $\epsilon$, i.e., $JS\left(\Phi_i^t, \Phi_j^t\right) \geq \epsilon$. We recall that the temporal records in $\mathcal{D}$ come from heterogeneous data sources, and therefore are characterized by different sets of attributes, as well as attributes with different names. To overcome such heterogeneity, existing works adopt either a schema matching technique to map the records to a unified schema [MZ98], or compare the values in the records without regard to the attribute to which they refer. The schema-based approaches achieve better performance at the cost of information loss [LWLG19], while the schema-agnostic approaches are less efficient

**Algorithm 13** HETERE

```
Input: Temporal database $\mathcal{D}$
Input: Set of time instances $\mathcal{T}$
Input: Similarity thresholds: Attribute $\epsilon$, Temporal $\tau$
Output: Set of clusters $\mathcal{C}$
1: $\mathcal{P} \leftarrow$ EXTRACTMATCHES($\mathcal{D}, \mathcal{T}, \epsilon, \tau$)
2: $G \leftarrow$ BUILDSIMILARITYGRAPH($\mathcal{P}$)
3: return FINDMAXIMALCLIQUES($G$)
```

but more effective. We follow the second line of work, and hence compute $JS$ by considering only the attribute values. Furthermore, to account for misspelling and abbreviations, instead of comparing the attribute values of the two records, we transform each value into a set of tokens of size $n$, and compare the two collections of distinct tokens. Let $TKS_i^t$ be the set of all the distinct tokens extracted from the attribute values in the set $\Phi_i^t\restriction_v$ of the instance $r_i^t$. Then, $JS\left(\Phi_i^t, \Phi_j^t\right)$ is defined as the number of tokens that $r_i^t$ and $r_j^t$ have in common, divided by the total number of tokens, i.e.,

$$JS\left(\Phi_i^t, \Phi_j^t\right) = \frac{\left|TKS_i^t \cap TKS_j^t\right|}{\left|TKS_i^t \cup TKS_j^t\right|}$$

If both $TKS_i^t$ and $TKS_j^t$ are empty, $JS\left(\Phi_i^t, \Phi_j^t\right)$ is set to be $1$.

We define the similarity $sim\left(r_i, r_j\right)$ between two temporal records $r_i$ and $r_j$ as the fraction of times the corresponding instances match. Let $\mathcal{T}_{i,j}^{\epsilon} = \left\{t \middle| t \in \mathcal{T} \wedge JS\left(\Phi_i^t, \Phi_j^t\right) \geq \epsilon\right\}$ be the set of time instances in which $r_i$ and $r_j$ match. Then,

$$sim\left(r_i, r_j\right) = \left|\mathcal{T}_{i,j}^{\epsilon}\right| / \left|\mathcal{T}\right| \tag{5.1}$$

Given a temporal similarity threshold $\tau$, we say that $r_i$ and $r_j$ *match*, if $sim\left(r_i, r_j\right) \geq \tau$.

Equation 5.1 takes values in the range $[0, 1]$, where $1$ indicates that the two records match in all the snapshots, and 0 that they never match. The more is the number of times in which the two records match, the higher becomes their similarity value, even though the attribute values that match are not the same in all the time instances.

## 5.4 Exact Solution

We first present our algorithm to solve Problem 3 exactly, called HETERE (**He**terogeneous **T**emporal **E**ntity **Re**solution), and discuss the main challenges that an exact solution entails.

**Algorithm 14** EXTRACTMATCHES

```
Input: Set of records $\mathcal{R}$
Input: Set of time instances $\mathcal{T}$
Input: Similarity thresholds: Attribute $\epsilon$, Temporal $\tau$
Output: Set of matching records $\mathcal{P}$
 1: $\mathcal{P} \leftarrow \emptyset$
 2: for all $r_i \in \mathcal{R}$ do
 3:     for all $r_j \in \mathcal{R}$ do
 4:         $evidence \leftarrow 0$
 5:         for all $t \in \mathcal{T}$ do
 6:             if $JS\left(\Phi_i^t, \Phi_j^t\right) \geq \epsilon$ then
 7:                 $evidence \leftarrow evidence + 1$
 8:         if $evidence / |\mathcal{T}| \geq \tau$ then
 9:             $\mathcal{P} \leftarrow \mathcal{P} \cup \{(r_i, r_j)\}$
10: return $\mathcal{P}$
```

In the next section, we present an approximate algorithm that can scale to larger collections of temporal records, while achieving good-quality results. Both algorithms rely on Equation 5.1 to assess the similarity between temporal records.

The exact solution computes Equation 5.1 for each pair of temporal records $r_i, r_j \in \mathcal{D}$, and then clusters the records into groups such that the pairwise similarities between records in the same group are all greater than the temporal similarity threshold $\tau$. The main steps of HETERE are outlined in Algorithm 13.

### 5.4.1 Find the Matching Pairs

In the first step, Algorithm 13 calls Algorithm 14 to extract the pairs of matching records from $\mathcal{D}$. This procedure iterates over all the pairs of records in $\mathcal{D}$ and incrementally calculates the value $\left|\mathcal{T}_{i,j}^{\epsilon}\right|$ using the auxiliary variable *evidence*. This variable is set to 0 at the beginning of the calculation, and is incremented by 1 every $t \in \mathcal{T}$ in which the record instances $r_i^t$ and $r_j^t$ match (lines 6-7). Once all the $t \in \mathcal{T}$ have been processed, variable $evidence$ is divided by the total number of time instances to obtain $sim\left(r_i, r_j\right)$, and if $r_i$ and $r_j$ are found to be matching, the pair $(r_i, r_j)$ is inserted into the set $\mathcal{P}$ (lines 8-9).

#### 5.4.1.1 Early Termination

We optimize the record matching process executed by Algorithm 14, by introducing an early termination criterion based on the following proposition.

**Proposition 5.3.** *Let $t_0, \ldots, t_{|\mathcal{T}|}$ be the sequence of time instances in $\mathcal{T}$, $\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k}$ be the restriction of $\mathcal{T}^{\epsilon}_{i,j}$ to the time instances before $t_k$, i.e., $\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k} = \left\{t_l | t_l \in \mathcal{T} \wedge l \leq k \wedge JS\left(\Phi^t_i, \Phi^t_j\right) \geq \epsilon\right\}$, and $s(i,j,k) = \left|\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k}\right|$. It holds that $s(i,j,k) + (|\mathcal{T}| - k) < \tau\,|\mathcal{T}| \Rightarrow sim\,(r_i, r_j) < \tau$.*

*Proof.* By definition $\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq |\mathcal{T}|} = \mathcal{T}^{\epsilon}_{i,j}$, and each set $\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k}$ is a subset of the set $\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k+1}$, with the inclusion being strict only when $JS\left(\Phi^{t_{k+1}}_i, \Phi^{t_{k+1}}_j\right) \geq \epsilon$. In particular, $JS\left(\Phi^{t_{k+1}}_i, \Phi^{t_{k+1}}_j\right) \geq \epsilon \Rightarrow \mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k+1} = \mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k} \cup \{t_{k+1}\}$. Therefore, the size function $s(i,j,k) = \left|\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k}\right|$ is monotonically non-decreasing with $k$., i.e., $s(i,j,k) \leq s(i,j,k+1)$, and satisfies the inequality $s(i,j,k+1) \leq s(i,j,k)+1$. It follows that $s(i,j,k+1) \leq s(i,j,|\mathcal{T}|) \leq s(i,j,k)+(|\mathcal{T}|-k)$, and hence $s(i,j,k) + (|\mathcal{T}| - k) < \tau\,|\mathcal{T}| \Rightarrow s(i,j,|\mathcal{T}|) = \left|\mathcal{T}^{\epsilon}_{i,j}\right| < \tau\,|\mathcal{T}|$. □

Let $t_0, \ldots, t_{|\mathcal{T}|}$ be the order in which the time instances in $\mathcal{T}$ are examined by Algorithm 14 for each pair of records $(r_i, r_j)$. We observe that, at any time $t_k$, the variable *evidence* holds the value $\left|\mathcal{T}^{\epsilon}_{i,j}\restriction_{\leq k}\right|$. Thanks to Proposition 5.3, the algorithm can thus stop processing new time instances and discard the pair $(r_i, r_j)$ when the condition $evidence < \tau\,|\mathcal{T}| - (|\mathcal{T}| - k) = |\mathcal{T}|\,(\tau - 1) - k$ is met.

Similarly, we can terminate the execution of the for loop and retain the pair $(r_i, r_j)$ as soon as $evidence/\,|\mathcal{T}| \geq \tau$. Thus, the loop over the time instances can be modified as follows:

```
1: for all $t \in \mathcal{T}$ do
2:     if $JS\left(\Phi^t_i, \Phi^t_j\right) \geq \epsilon$ then
3:         $evidence \leftarrow evidence + 1$
4:     if $evidence < |\mathcal{T}|\,(\tau - 1) - k$ or $evidence \geq \tau\,|\mathcal{T}|$ then
5:         break
```

### 5.4.2 Find the Matching Groups

To solve Problem 3, the algorithm needs to identify the groups of temporal records such that the similarity between each pair of records in the same group is greater than $\tau$. When the similarity relationship is transitive, these groups can be easily obtained by merging all the pairs in $\mathcal{P}$ that have one element in common, because $sim(r_1, r_2) \geq \tau \wedge sim(r_2, r_3) \geq \tau \Rightarrow sim(r_1, r_3) \geq \tau$. However, using our measure $sim$, the transitive property is not satisfied. Consider, for example, the set of time instances $\mathcal{T} = \{t_0, \ldots, t_5\}$, the attribute similarity threshold $\epsilon = 1$, the temporal similarity threshold $\tau = 1/2$, and the three records $r_1$, $r_2$, and $r_3$ in Figure 5.1. The red lines in records $r_1$ and $r_2$ indicate that $r_1$ and $r_2$ are similar in the time instances $t_0$, $t_1$, and $t_2$, i.e.,

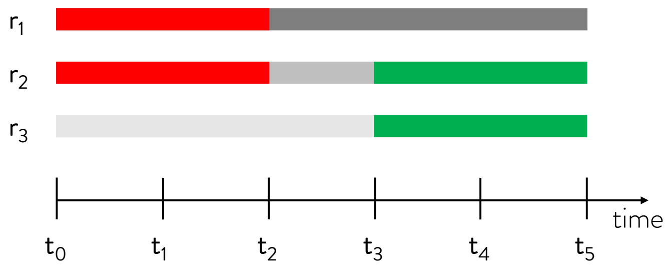


**Figure 5.1:** Three temporal records $r_1$, $r_2$, and $r_3$, where the matching instances of $r_1$ and $r_2$ are in red, and the matching instances of $r_2$ and $r_3$ are in green.

$\forall t \in [t_0, t_2]\,, JS\left(\Phi_1^t, \Phi_2^t\right) \geq \epsilon$. Similarly, the green lines in records $r_2$ and $r_3$ indicate that $r_2$ and $r_3$ are similar in the time instances $t_3$, $t_4$, and $t_5$, i.e., $\forall t \in [t_3, t_5]\,, JS\left(\Phi_2^t, \Phi_3^t\right) \geq \epsilon$. Therefore, $sim(r_1, r_2) = 3/6 \geq 1/2$ and $sim(r_2, r_3) = 3/6 \geq 1/2$. We can see that $r_3$ is not equal to $r_2$ in $t_0$, $t_1$, and $t_2$, and hence $r_3$ does not match $r_1$ in those time instances. Similarly, $r_1$ is not equal to $r_2$ in $t_3$, $t_4$, and $t_5$, and hence $r_1$ does not match $r_3$ in those time instances. As a consequence $sim(r_1, r_3) = 0 \leq 1/2$.

Our strategy to identify the groups of matching records consists of two phases: (i) we build a similarity graph $G$ by creating a node for each record and an edge for each pair of records in $\mathcal{P}$ (Procedure BUILDSIMILARITYGRAPH), and (ii) we extract the maximal cliques from $G$ (Procedure FINDMAXIMALCLIQUES). A clique is a fully-connected set of nodes, and therefore corresponds to a set of records with pairwise similarity greater than $\tau$, which in turn is a group of records describing the same real-world entity. To enumerate the maximal cliques in $G$ we use our implementation of the *GP* algorithm of Wang et al. [WCH+17], described in details and illustrated in Algorithm 9 in Section 4.3.

### 5.4.3 Complexity

The computation of the similarity values between each pair of records takes $\mathcal{O}(|\mathcal{D}|^2 \cdot |\mathcal{T}| \cdot |\mathcal{A}|^2)$, as it requires to compare the attribute values of the records in each time instance in $\mathcal{T}$ and, in the worst case, the records have all the attributes in the domain $\mathcal{A}$. To construct the similarity graph $G$, the algorithm needs to create an edge for each pair of records in $\mathcal{P}$. In the worst case, $\mathcal{P}$ contains every pair of records in $\mathcal{D}$, and thus the complexity of this phase is $\mathcal{O}(|\mathcal{D}|^2)$. Finally, the time complexity of the enumeration of the maximal cliques in $G$ is $\mathcal{O}\left(|\mathcal{P}| \cdot \kappa(G)\right)$, where $\kappa(G)$ is the number of cliques in $G$ and $|\mathcal{P}| = |\mathcal{D}|^2$ in the worst case. The complexity of Algorithm 13 is therefore $\mathcal{O}(|\mathcal{D}|^2\left(|\mathcal{T}| \cdot |\mathcal{A}|^2 + \kappa(G)\right)$.

**Algorithm 15** A-HETERE

```
Input: Temporal database $\mathcal{D}$
Input: Set of time instances $\mathcal{T}$
Input: Thresholds: Attribute $\epsilon$, Temporal $\tau$
Input: Threshold: Size $s_{max}$, Convergence $c$
Output: Set of clusters $\mathcal{S}$
 1: $\mathcal{B} \leftarrow \emptyset$
 2: $TKS \leftarrow$ EXTRACTTOKENS$(\mathcal{D})$
 3: for all $tk \in TKS$ do
 4:     $B_{tk} \leftarrow \emptyset$
 5: for all $r_i \in \mathcal{D}$ do
 6:     for all $tk \in TKS_i$ do
 7:         $B_{tk} \leftarrow B_{tk} \cup \{r_i\}$
 8: for all $tk \in TKS$ do
 9:     if $|B_{tk}| \leq s_{max}$ then
10:         $\mathcal{B} \leftarrow \mathcal{B} \cup \{B_{tk}\}$
11: $\mathcal{B}' \leftarrow$ RECONSTRUCTBLOCKS$(\mathcal{B})$
12: for all $B \in \mathcal{B}'$ do
13:     $\mathcal{P} \leftarrow \mathcal{P} \cup$ EXTRACTMATCHES$(B, \mathcal{T}, \epsilon, \tau)$
14: $G \leftarrow$ BUILDSIMILARITYGRAPH$(\mathcal{P})$
15: return GRAPHCLUSTERING$(G, c)$
```

## 5.5 Approximate Solution

Given the quadratic complexity of the record matching step and the hardness of the enumeration of the maximal cliques [WCH+17], Algorithm 13 becomes impractical for large dataset. We therefore propose also an approximate solution, called A-HETERE, which improves the efficiency of HETERE, without sacrificing much accuracy. This algorithm reduces the complexity of the matching step by adopting a schema-agnostic time-aware approach based on *meta-blocking* [PKPN14], which divides the records into blocks and compares only the records within the same block. Then, it replaces the enumeration of the maximal cliques in the similarity graph $G$ with the easier task of detecting dense groups of similar nodes, which is accomplished by applying a graph clustering algorithm for weighted graphs.

The pseudocode of A-HETERE is shown in Algorithm 15 and described in the following paragraphs.

### 5.5.1 Schema-agnostic time-aware meta-blocking

To improve the time complexity of Algorithm 14, algorithm A-HETERE reduces the number of record comparisons by pruning pairs of records unlikely to be similar. To achieve this goal, the algorithm exploits the attribute values of the records to generate blocks of records such that records in the same block are more likely to be similar than those in different blocks.

The first step is the segmentation of the attribute values of all the records in $\mathcal{D}$ to create a set of distinct tokens $TKS$ (line 2). The algorithm applies a transformation function $\psi$ to each $r_i \in \mathcal{D}$ to transform it into a collection of pairs, where the first element is a token of an attribute value appearing in some set $\Phi_i^t\lceil_v$ of $r_i$, and the second element is the set of time instances in which that token is present. The function $\psi$ is thus defined as $\psi(r_i) = \big\{\, (tk, \Psi(i, tk)) \,\big| tk \in TKS_i \wedge \Psi(i, tk) = \big\{t | tk \in TKS_i^t\big\}\,\big\}$, where $TKS_i$ is the set of all the distinct tokens extracted from the attribute values in the sets $\Phi_i^{t_{i1}}\lceil_v, \ldots, \Phi_i^{t_{in}}\lceil_v$ of $r_i$.

The next step is the creation of a block $B_{tk}$ for each distinct token $tk \in TKS$, which contains all the records $r_i$ such that $tk \in TKS_i$ (lines 5-7). Based on the observation that blocks containing a large percentage of records are likely associated to stopwords or other non-discriminative attributes (e.g., the hometown in the Census database of a particular city), we keep and insert in the collection $\mathcal{B}$ only the blocks with size smaller than the size threshold $s_{max}$ (lines 9-10).

The blocks retained may have a high degree of overlap, because the same record is placed in multiple blocks, one for each of its tokens. Despite this redundancy has the benefit of reducing the likelihood of missed matches, it comes at the cost of a larger number of record comparisons. To gain efficiency with limited impact on the effectiveness, we reconstruct the collection of blocks $\mathcal{B}$ to obtain a new set $\mathcal{B}'$ involving fewer records per block. The reconstruction of $\mathcal{B}$ is performed by Procedure RECONSTRUCTBLOCKS illustrated in Algorithm 16, and is divided into two stages, i.e., the construction of a blocking graph, and the removal of some of its edges. The goal of the first stage is to remove all the redundant record comparisons, while the second stage aims to remove the less promising comparisons.

#### 5.5.1.1 Blocking Graph

To construct the blocking graph, the algorithm creates a node $v_i$ for each record $r_i$ appearing in $\mathcal{B}$ (lines 6-12), and an edge $(v_i, v_j)$ for each pair of records $r_i$ and $r_j$ co-appearing in a block

**Algorithm 16** RECONSTRUCTBLOCKS

**Input:** A block collection $\mathcal{B}$

**Output:** A restructured block collection $\mathcal{B}'$

```
1:  $V \leftarrow \emptyset; E \leftarrow \emptyset; \mathcal{B}' \leftarrow \emptyset$
2:  $U \leftarrow \emptyset; I \leftarrow \emptyset$
3:  for all $B_{tk} \in \mathcal{B}$ do
4:      for all $r_i \in B_{tk}$ do
5:          if $v_i \notin V$ then
6:              $V \leftarrow V \cup \{v_i\}$
7:              $U[i] \leftarrow 1$
8:          else
9:              $U[i] \leftarrow U[i] + 1$
10:         for all $r_j \in B_{tk}$ do
11:             if $v_j \notin V$ then
12:                 $V \leftarrow V \cup \{v_j\}$
13:                 $U[j] \leftarrow 1$
14:             else
15:                 $U[j] \leftarrow U[j] + 1$
16:             if $(v_i, v_j) \notin E$ then
17:                 $E \leftarrow E \cup \{(v_i, v_j)\}$
18:                 $I[(i,j)] \leftarrow 1$
19:                 $(v_i, v_j).w \leftarrow \frac{|\Psi(i,tk) \cap \Psi(j,tk)|}{|\Psi(i,tk) \cup \Psi(j,tk)|}$
20:             else
21:                 $I[(i,j)] \leftarrow I[(i,j)] + 1$
22:                 $(v_i, v_j).w \leftarrow (v_i, v_j).w + \frac{|\Psi(i,tk) \cap \Psi(j,tk)|}{|\Psi(i,tk) \cup \Psi(j,tk)|}$
23: for all $(v_i, v_j) \in E$ do
24:     $(v_i, v_j).w \leftarrow \frac{(v_i,v_j).w}{(U[i]+U[j]-I[(i,j)])} \log \frac{|E|}{|N(v_i)|} \log \frac{|E|}{|N(v_j)|}$
25: for all $v_i \in V$ do
26:     $avgW \leftarrow$ COMPUTELOCALTHRESHOLD$(v_i)$
27:     for all $v_j \in N(v_i)$ do
28:         if $(v_i, v_j).w < avgW$ then
29:             $E \leftarrow E \setminus \{(v_i, v_j)\}$
30: for all $(v_i, v_j) \in E$ do
31:     $\mathcal{B}' \leftarrow \mathcal{B}' \cup \{(v_i, v_j)\}$
32: return $\mathcal{B}'$
```

(line 17). Since each pair of records is connected by a single edge, all the redundant comparisons between the two records are easily eliminated. Then, a weight is assigned to each edge, using a novel weighting function $\omega : E \mapsto \mathbb{R}$ inspired by the EJS weighting scheme [PKPN14]. Let $TKS_{i \cap j}$ indicate the set of tokens that $r_i$ and $r_j$ have in common, and $TKS_{i \cup j}$ the set of tokens appearing in either $r_i$ or $r_j$. We define the weight $\omega((v_i, v_j))$ as the weighted Jaccard similarity between the sets of tokens of $r_i$ and $r_j$, multiplied by a term inspired from the IDF metric of Information Retrieval, i.e.,

$$\omega((v_i, v_j)) = \frac{\sum_{tk \in TKS_{i \cap j}} \text{JACCARD}(tk, i, j)}{|\,TKS_{i \cup j}|} \log \frac{|E|}{|N(v_i)|} \log \frac{|E|}{|N(v_j)|}$$

with $\text{JACCARD}(tk, i, j) = |\Psi(i, tk) \cap \Psi(j, tk)| \,/\, |\Psi(i, tk) \cup \Psi(j, tk)|$, and $N(v_i)$ denoting the neighborhood of node $v_i$.

The peculiarity of the weighting function $\omega$ is that it assigns larger weights to (i) pairs of records that share more blocks, (ii) pairs of records that share tokens in more snapshots, and (iii) pairs of records with fewer outgoing edges.

Each edge weight $(v_i, v_j).w$ is incrementally calculated during the construction of the graph (lines 16-24), with the help of the two auxiliary data structures $U$ and $I$, which store the number of appearances of each record in $\mathcal{B}$ and the number of co-appearances of each pair of records in $\mathcal{B}$, respectively. In fact $|\,TKS_{i \cup j}| = (U[i] + U[j] - I[(i, j)])$.

#### 5.5.1.2 Edge Pruning

Assuming that a larger weight is an indicator of a higher similarity between the corresponding records, we eliminate the less promising comparisons by adopting a node-centric pruning strategy that retains, for each node $v_i$, the edges with weight greater than the average weight in the node neighborhood $N(v_i)$. Thus, for each node $v_i$ in the blocking graph, we compute the average weight in the node neighborhood $avgW$ (line 26), and then iterate over all its adjacent nodes $v_j$, removing all the edges $(v_i, v_j)$ with weight below the threshold $avgW$ (line 29).

The output of Algorithm 16 is a new collection of blocks $\mathcal{B}'$ containing a block $B'$ for each retained edge (line 31). These edges are the candidate pairs of records that need to be further compared by Procedure EXTRACTMATCHES (lines 12-13). All the pairs $(r_i, r_j)$ with similarity $sim(r_i, r_j) \geq \tau$ are inserted into $\mathcal{P}$ (line 13), and then used to create the edges of the similarity graph $G$ (line 14), each of which is weighted with the corresponding similarity value $sim(r_i, r_j)$.

### 5.5.2 Graph Clustering

The final step of A-HETERE is the detection of dense groups of similar records via graph clustering. We adopt the Markov Clustering (MCL) algorithm [EVDO02], thanks to its accuracy and scalability. MCL finds clusters of nodes connected by large weights, by simulating random

**Algorithm 17** GRAPHCLUSTERING

```
Input: Graph $G$
Input: Convergence threshold $c$
1: $M_1 \leftarrow$ CREATESTOCHASTICMATRIX($G$)
2: $d \leftarrow 2c$
3: while $d > c$ do
4:     $M_2 \leftarrow M_1 * M_1$
5:     $M_1 \leftarrow \Gamma(M_2)$
6:     $d \leftarrow \Delta(M_1, M_2)$
7: return EXTRACTCONNECTEDCOMPONENTS($M_1$)
```

walks through the graph. This simulation is performed by alternating an expansion and an inflation phase, as illustrated in Algorithm 17. Expansion corresponds to performing random walks of higher length, and is implemented by computing the power of the stochastic matrix associated to the graph (line 4), which is obtained by normalizing the adjacency matrix of $G$ such that the sum of the elements in each column is 1 (line 1). Inflation, on the other hand, has the effect of boosting the probabilities of intra-cluster walks while demoting inter-cluster walks, and is achieved by computing the Hadamard power $\Gamma$ of the matrix combined with a diagonal scaling (line 5).

Iterating these two phases will eventually result in the separation of the graph represented by the matrix $M_1$ into several components. When this happens, further expansion or inflation phases will not change $M_1$ too much, and thus the difference between $M_1$ and $M_2$ is smaller than the convergence threshold $c$. Since an equilibrium is reached, the algorithm terminates (line 3). Each connected component in the graph associated with the final matrix $M_1$ is interpreted as a cluster, and thus is part of the final output of Algorithm 17, and hence of Algorithm 15.

### 5.5.3 Complexity

Algorithm 16 takes $\mathcal{O}(|V||E|) = \mathcal{O}(|\mathcal{D}|\,|E|)$ to restructure the blocks, while the complexity of MCL is $\mathcal{O}(|\mathcal{D}|^3)$. The total complexity of the approximate solution is therefore $\mathcal{O}(|\mathcal{D}|^3)$. Nonetheless, experimental results show that, in general, MCL reaches an equilibrium after 3-10 iterations, and hence is actually much faster [EVDO02].

### 5.5.4 Quality of the Approximate Solution

We measure the quality of the block collection $\mathcal{B}'$ created by Algorithm 16 in terms of three well-established measures [PKPN14], i.e., Pair Quality ($PQ$), Pair Completeness ($PC$), and Reduction Ratio ($RR$). $PQ$ indicates the percentage of records in the blocks in $\mathcal{B}'$ that are duplicates, and is defined as $PQ(\mathcal{B}') = |Dup(\mathcal{B}')| \,/\, ||\mathcal{B}'||$, where $|Dup(\mathcal{B}')|$ denotes the number of duplicates in $\mathcal{B}'$ and $||\mathcal{B}'||$ is the total number of comparisons entailed in the blocks in $\mathcal{B}'$; $PC$ is the percentage of duplicates in $\mathcal{D}$ that co-appear in a block in $\mathcal{B}'$, and is defined as $PC(\mathcal{B}') = |Dup(\mathcal{B}')| \,/\, |Dup(\mathcal{D})|$; and $RR$ is the percentage of comparisons saved by comparing only the records within the blocks in $\mathcal{B}'$ with respect to the comparisons entailed in a baseline block collection $\mathcal{B}_{bs}$, and is defined as $RR(\mathcal{B}') = 1 - ||\mathcal{B}'|| \,/\, ||\mathcal{B}_{bs}||$. All the three measures take value in the range $[0, 1]$, with higher values of $PC$ and indicating higher effectiveness, and higher values of $PQ$ and $RR$ indicating higher efficiency.

On the other hand, we measure the quality of the set of clusters $\mathcal{S} = \{S_1, \ldots, S_m\}$ created by Algorithm 17 in terms of precision and recall with the respect to the ground-truth, which is, in our case, the set of maximal cliques $\mathcal{C} = \{C_1, \ldots, C_k\}$ in $G$. Let $TP(\mathcal{S})$ be the number of records correctly placed in the same cluster, i.e.,

$$TP(\mathcal{S}) = \sum_{i=1}^{k} argmax_{j \in [1,m]} |C_i \cap S_j|$$

and $FP(\mathcal{S})$ be the number of records wrongly placed in the same cluster, i.e.,

$$FP(\mathcal{S}) = \sum_{j=1}^{m} argmin_{i \in [1,k]} |S_j \setminus C_i|$$

The *precision* of $\mathcal{S}$ is calculated as $Pr(\mathcal{S}) = TP(\mathcal{S}) \,/\, (TP(\mathcal{S}) + FP(\mathcal{S}))$, while its *recall* is $Re(\mathcal{S}) = TP(\mathcal{S}) / |\mathcal{D}|$. Then, the quality of $\mathcal{S}$ is defined as the balanced F-score: $Q(\mathcal{S}) = 2(Pr(\mathcal{S}) \cdot Re(\mathcal{S}))/(Pr(\mathcal{S}) + Re(\mathcal{S}))$. The measure $Q$ takes value in the range $[0, 1]$, with 1 denoting the best quality, and 0 the worst.

Chapter 6

# Conclusions

In this thesis we motivated and described three novel pattern mining problems in highly heterogeneous and dynamic datasets, and presented efficient and effective solutions.

First, we considered the problem of mining relevant patterns in multi-weighted graphs. As opposed to the previous approaches, which measure the importance of a pattern solely based on its frequency in the graph, we recognized the need for a measure that takes into consideration also the user preferences. We thus proposed a novel family of scoring functions, called MNI-compatible functions, that assess the relevance of a pattern in terms of both the weights on the edges of its appearances and its frequency. These functions allow the retrieval of personalized patterns, while retaining the advantages offered by the anti-monotone property, which is a powerful mean to an effective and early pruning of the search space. Then, we analyzed the capabilities and the limitations of four different functions from this family, by integrating them into novel exact and approximate algorithms. Finally, we proved the effectiveness and efficiency of the algorithms on real and synthetic datasets, and compared the performance of the centralized and the distributed version of our approach using graphs with different sizes and characteristics. We described in which cases we can benefit from distributing the computation, and when a centralized solution is still to be preferred, proving that distributing graph pattern mining algorithms in an efficient way is not straightforward.

Secondly, we tackled the problem of finding maximal dense correlated subgraphs in dynamic networks. We proposed two measures to compute the density of a subgraph that changes over time, and one measure to assess the temporal correlation of the subgraph edges. We developed a generic framework that uses those measures to identify all the subgraphs satisfying given density and correlation thresholds. We extended this framework to implement the retrieval of those subgraphs that have little redundant information, specified by low Jaccard similarity. We exhibited a number of experimental results that illustrate the effectiveness of our approach in detecting all maximal dense groups of correlated edges, compare its performance with its nearest competitor, and prove its applicability to large real networks with different characteristics. Last but not least, we provided an approximate solution that can run one order of magnitude faster, and yet achieves good precision and recall.

Finally, we formally defined the notion of temporal record in the context of heterogeneous data, and the task of detecting maximal groups of matching records in heterogeneous temporal databases. In contrast to existing solutions that either discard the temporal information of the data, or assume a fixed set of attributes for each record, we designed a time-aware schema-agnostic measure that computes the similarity between two heterogeneous temporal records in terms of the similarity between their attribute values over time, independently of the corresponding attribute names. We showed how this measure can be integrated into an exact algorithm that enumerates all the maximal groups of pairwise matching records, under the assumption that two records match if they display a sufficient level of similarity over time. Furthermore, we presented an approximate algorithm that overcomes the two main limitations of the exact solution, by adopting a meta-blocking and a graph clustering algorithm, respectively. The optimizations proposed can allow this algorithm to scale to datasets of larger size, without sacrificing much accuracy.

## 6.1 Extensions and Open Problems

The discussions on the challenges involved in solving the pattern mining problems introduced in this work, and the experimental results obtained in the evaluation of the approaches developed, unveiled some possible directions for future studies.

**Memory-efficient Distributed Score-based Pattern Mining.** The main limitations of our distributed algorithm to solve score-based pattern mining in multi-weighted graphs are the high memory requirement and large network cost inherent in the Arabesque framework and caused by the huge number of pattern embeddings that the workers need to materialize and shuffle in each superstep. For these reasons, at low frequencies the algorithm is likely to exceed the available resources and fail to provide any result.

Recently, several efforts have been devoted to the development of general-purpose distributed graph mining systems that require less storage and machine-to-machine communications, and thus are able to scale up to graphs of billions or even trillions of nodes and edges [PJR17, JKS18, QZL+18, BR18, DTG+19]. Fractal [DTG+19], for example, is a flexible framework that supports various graph pattern mining applications via intuitive and expressive APIs. Thanks to effective strategies to bound the memory consumption, balance the load, and speed up the enumeration of the pattern embeddings, this framework outperforms the existing distributed solutions in almost every computation.

We will investigate if it is possible to use Fractal's APIs to distribute our score-based pattern mining algorithm in a more efficient way, hence enabling us to test it on graphs with much larger amounts of edge weights, and opening the way for other interesting applications, such as product recommendation using the entire Amazon co-purchase network.

**Parallel Dense Correlated Subgraph Mining.** Our exact solution to the dense correlated subgraph mining problem entail three costly operations, namely the enumeration of the pairs of correlated edges, the enumeration of the maximal cliques of correlated edges, and the generation of the candidate subsets of dense edges. The first operation incurs a quadratic cost in the number of edges of the graph, as it requires to test each pair of edges. We addressed this challenge in our approximate solution, which uses a min-hash strategy to retain and then compare only the most promising pairs of edges.

The complexity of the second operation is bounded by the number of cliques in the graph created using the pairs of correlated edges. Since at low values of the correlation threshold the size of this graph tends to the size of the input graph, this operation can be prohibitive for some applications or datasets. Therefore, our next goal is to tackle this challenge, and hence identify optimal strategies to detect maximal groups of pairwise correlated edges. To this aim, the first step will be analyzing existing solutions in pseudo-clique enumeration [ZHOT16, CFM+17] and in parallel clique enumeration [WCH+17]. The second step will be investigating other possible ways to combine the first two operations and solve them more efficiently and effectively.

Finally, in this thesis we optimized the third operation by adopting effective strategies to prune the search space, and early termination conditions to stop the examination of a branch of candidates if no solution could be found among them. Since each branch of candidates can be processed independently, this operation can greatly benefit from parallelization. Each worker would need only a copy of the candidate subset of edges and their corresponding vectors of edge weights, and all the dense correlated subgraphs would be collected by the master at the end of the computation of the workers. Then, the maximal groups would be extracted from this collection. Therefore, another direction of future work will be studying how to distribute our algorithm, while guaranteeing load-balancing and correctness of the results produced.

**Evaluating Scalability and Quality of HeTERe and A-HeTERe.** Chapter 5 describes the exact and approximate algorithm that we propose to address the task of entity resolution in heterogeneous temporal databases. These algorithms adopt a similarity measure for temporal records that satisfies essential properties for any effective temporal entity resolution technique. In particular, this measure can match also those records that display a small but persistent percentage of common attributes. Moreover, our approximate solution can achieve high scalability, as it relies on meta-blocking, a technique commonly adopted in the literature to speed up entity matching processes.

In this work, however, we have not presented an experimental evaluation of the performance and capabilities of our solutions. Testing our algorithms on datasets of different sizes, for example, is pivotal to assess their level of scalability. Similarly, considering datasets of different temporal granularities and characteristics, we can determine how to configure the parameters of the system to maximize its accuracy in different entity matching tasks.

Therefore, as future work, we plan to evaluate the efficiency and effectiveness our algorithms using both real and synthetic datasets. Regarding the real datasets, we will consider two datasets used in previous works in temporal entity resolution [LDMS11, CDN14a, LLHT15], namely, a benchmark of European patent data[1] and the DBLP data set[2]. The former contains records about 359 inventors from French patents, while the latter contains records about authors and their papers. In addition, we will create a real-world profile dataset by integrating public profiles crawled by social networks like LinkedIn and Twitter, using the information provided in the *link* section of the profiles to derive the ground-truth.

[1]`http://www.esf-ape-inv.eu/`
[2]`http://www.informatik.uni-trier.de/ley/db/`

# List of Figures

# List of Tables

# Bibliography

[AAK$^+$16] E. Abdelhamid, I. Abdelaziz, P. Kalnis, Z. Khayyat, and F. Jamour. Scalemine: Scalable parallel frequent subgraph mining in a single large graph. In *SC*, pages 716–727, 2016. (Cited on page 19.)

[AC09] Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 25–37, 2009. (Cited on page 21.)

[ACS$^+$17] Ehab Abdelhamid, Mustafa Canim, Mohammad Sadoghi, Bishwaranjan Bhattacharjee, Yuan-Chi Chang, and Panos Kalnis. Incremental frequent subgraph mining on large evolving graphs. *TKDE*, 29(12):2710–2723, 2017. (Cited on page 23.)

[Agg16] Charu C Aggarwal. *Recommender Systems*. 2016. (Cited on pages 2 and 57.)

[AHCS$^+$07] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, Jeremy Besson, and Mohammed J Zaki. Origami: Mining representative orthogonal graph patterns. In *ICDM*, pages 153–162, 2007. (Cited on page 15.)

[AHÖD14] Güneş Aluç, Olaf Hartig, M Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of rdf data management systems. In *ISWC*, pages 197–212, 2014. (Cited on page 58.)

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD Records*, volume 22, pages 207–216, 1993. (Cited on pages 13 and 14.)

[ALYJ10] C. C. Aggarwal, Y. Li, P. S. Yu, and R. Jin. On dense pattern mining in graph streams. In *PVLDB*, 2010. (Cited on page 24.)

[ANDFMG18] C. Aslay, M. A. U. Nasir, G. De Francisci Morales, and A. Gionis. Mining frequent patterns in evolving graphs. In *CIKM*, pages 923–932, 2018. (Cited on page 24.)

[ARS02] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *LATIN*, pages 598–612, 2002. (Cited on page 21.)

[ASKS12] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *PVLDB*, 5(6):574–585, 2012. (Cited on pages 2, 20, and 22.)

[BBC$^+$15] Oana Denisa Balalau, Francesco Bonchi, TH Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In *WSDM*, pages 379–388, 2015. (Cited on pages 21 and 79.)

[BBC$^+$17] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. gMark: Schema-driven generation of graphs and queries. *TKDE*, 29(4):856–869, 2017. (Cited on page 58.)

[BCFM00] Andrei Z Broder, Moses Charikar, Alan M Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60(3):630–659, 2000. (Cited on page 82.)

[BCK$^+$03] Peter Bickel, Chao Chen, Jaimyoung Kwon, John Rice, Pravin Varaiya, and Erik van Zwet. Traffic flow on a freeway network. In *Nonlinear Estimation and Classification*, pages 63–81. 2003. (Cited on page 22.)

[BFSS15] Niv Buchbinder, Moran Feldman, Joseph Seffi, and Roy Schwartz. A tight linear time (1/2)-approximation for unconstrained submodular maximization. *SIAM Journal on Computing*, 44(5):1384–1402, 2015. (Cited on page 21.)

[BG04] Indrajit Bhattacharya and Lise Getoor. Iterative record linkage for cleaning and integration. In *SIGMOD*, pages 11–18, 2004. (Cited on page 28.)

[BGW03] Ulrik Brandes, Marco Gaertler, and Dorothea Wagner. Experiments on graph clustering algorithms. In *European Symposium on Algorithms*, 2003. (Cited on page 90.)

[BH03] Gary D Bader and Christopher WV Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC bioinformatics*, 4(1):2, 2003. (Cited on page 20.)

[BJ16] Nisha Babu and Ansamma John. A distributed approach to weighted frequent subgraph mining. In *International Conference on Emerging Technological Trends*, pages 1–7, 2016. (Cited on page 17.)

[BLGS06] Mustafa Bilgic, Louis Licamele, Lise Getoor, and Ben Shneiderman. D-dupe: An interactive tool for entity resolution in social networks. In *VAST*, pages 43–50, 2006. (Cited on pages 8 and 28.)

[BMS11] Petko Bogdanov, Misael Mongiovì, and Ambuj K Singh. Mining heavy subgraphs in time-evolving networks. In *ICDM*, pages 81–90, 2011. (Cited on pages 3, 6, and 22.)

[BN08] Bjorn Bringmann and Siegfried Nijssen. What is frequent in a single graph? In *PAKDD*, pages 858–863, 2008. (Cited on pages 2, 5, 18, 36, and 55.)

[BR18] Vandana Bhatia and Rinkle Rani. Ap-fsm: A parallel algorithm for approximate frequent subgraph mining using pregel. *Expert Systems with Applications*, 106:217–232, 2018. (Cited on page 121.)

[BXG$^+$13] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *WWW*, pages 119–130, 2013. (Cited on page 20.)

[BXL17] Dorna Bandari, Shuo Xiang, and Jure Leskovec. Categorizing user sessions at pinterest. *arXiv preprint arXiv:1703.09662*, 2017. (Cited on page 4.)

[CBL08] Jeffrey Chan, James Bailey, and Christopher Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16(1):53–96, 2008. (Cited on pages 7 and 27.)

[CBLH12] Jeffrey Chan, James Bailey, Christopher Leckie, and Michael Houle. ciforager: Incrementally discovering regions of correlated change in evolving graphs. *TKDD*, 6(3):11, 2012. (Cited on pages 7, 27, and 93.)

[CDB$^+$09] James C Costello, Mehmet M Dalkilic, Scott M Beason, Jeff R Gehlhausen, Rupali Patwardhan, Sumit Middha, Brian D Eads, and Justen R Andrews. Gene networks in drosophila melanogaster: integrating experimental data to predict gene function. *Genome biology*, 10(9):R97, 2009. (Cited on page 3.)

[CDN14a] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F Naughton. Modeling entity evolution for temporal record matching. In *SIGMOD*, pages 1175–1186, 2014. (Cited on pages 10, 30, and 122.)

[CDN14b] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F Naughton. Tracking entities in the dynamic world: A fast algorithm for matching temporal records. *PVLDB*, 7(6):469–480, 2014. (Cited on page 30.)

[CES15] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. Entity resolution in the web of data. *Synthesis Lectures on the Semantic Web*, 5(3):1–122, 2015. (Cited on pages 8 and 28.)

[CFM$^+$17] Alessio Conte, Donatella Firmani, Caterina Mordente, Maurizio Patrignani, and Riccardo Torlone. Fast enumeration of large k-plexes. In *KDD*, pages 115–124, 2017. (Cited on page 121.)

[CG13] Peter Christen and Ross W Gayler. Adaptive temporal entity resolution on dynamic databases. In *PAKDD*, pages 558–569, 2013. (Cited on pages 10 and 31.)

[Cha00] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95, 2000. (Cited on pages 21, 23, 76, 77, and 87.)

[CHC$^+$07] Augustin Chaintreau, Pan Hui, Jon Crowcroft, Christophe Diot, Richard Gass, and James Scott. Impact of human mobility on opportunistic forwarding algorithms. *IEEE Trans. on Mobile Comput.*, 6(6):606–620, 2007. (Cited on page 89.)

[Chr12] P. Christen. A survey of indexing techniques for scalable record linkage and deduplication. *TKDE*, 24(9):1537–1555, 2012. (Cited on pages 8 and 28.)

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971. (Cited on page 14.)

[CS10] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *TKDE*, 24(7):1216–1230, 2010. (Cited on pages 2 and 20.)

[CZLW15] Yifan Chen, Xiang Zhao, Xuemin Lin, and Yang Wang. Towards frequent subgraph mining on single large uncertain graphs. In *ICDM*, pages 41–50, 2015. (Cited on page 20.)

[DBD94] Timothy R Derrick, Barry T Bates, and Janet S Dufek. Evaluation of time-series data sets using the pearson product-moment correlation coefficient. *Medicine and science in sports and exercise*, 26(7), 1994. (Cited on page 78.)

[DJD$^+$09] Xiaoxi Du, Ruoming Jin, Liang Ding, Victor E Lee, and John H Thornton Jr. Migration motif: a spatial-temporal pattern mining approach for financial markets. In *KDD*, pages 1135–1144, 2009. (Cited on page 2.)

[DKB$^+$12] Qi Ding, Natallia Katenka, Paul Barford, Eric Kolaczyk, and Mark Crovella. Intrusion as (anti) social communication: characterization and detection. In *KDD*, pages 886–894, 2012. (Cited on page 26.)

[DRZ07] L. De Raedt and A. Zimmermann. Constraint-based pattern set mining. In *SDM*, pages 237–248, 2007. (Cited on page 38.)

[DS15] X. L. Dong and D. Srivastava. *Big Data Integration*. 2015. (Cited on page 8.)

[DTG$^+$19] Vinicius Dias, Carlos HC Teixeira, Dorgival Guedes, Wagner Meira, and Srinivasan Parthasarathy. Fractal: A general-purpose graph pattern mining system. In *SIGMOD*, pages 1357–1374, 2019. (Cited on page 121.)

[dVKCC09] T. de Vries, H. Ke, S. Chawla, and P. Christen. Robust record linkage blocking using suffix arrays. In *CIKM*, pages 305–314, 2009. (Cited on page 28.)

[EA12] Philippe Esling and Carlos Agon. Time-series data mining. *CSUR*, 45(1):12, 2012. (Cited on page 32.)

[EASK14] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *PVLDB*, 7(7):517–528, 2014. (Cited on pages 2, 5, 19, 58, and 63.)

[EFGM18] Dhivya Eswaran, Christos Faloutsos, Sudipto Guha, and Nina Mishra. Spotlight: Detecting anomalies in streaming graphs. In *KDD*, 2018. (Cited on page 26.)

[EHB10] Frank Eichinger, Matthias Huber, and Klemens Böhm. On the usefulness of weight-based constraints in frequent subgraph mining. In *SGAI*, pages 65–78, 2010. (Cited on page 17.)

[EIV07] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection. *TKDE*, 19(1):1–16, 2007. (Cited on pages 8 and 28.)

[ELS15] Alessandro Epasto, Silvio Lattanzi, and Mauro Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, pages 300–310, 2015. (Cited on page 6.)

[EPP$^+$17] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, 2017. (Cited on page 9.)

[ERSR$^+$15] Julia Efremova, Bijan Ranjbar-Sahraei, Hossein Rahmani, Frans A Oliehoek, Toon Calders, Karl Tuyls, and Gerhard Weiss. Multi-source entity resolution for genealogical data. In *Population reconstruction*, pages 129–154. 2015. (Cited on pages 8 and 28.)

[EVDO02] Anton J Enright, Stijn Van Dongen, and Christos A Ouzounis. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research*, 30(7):1575–1584, 2002. (Cited on pages 115 and 116.)

[FB07] Mathias Fiedler and Christian Borgelt. Subgraph support in a single large graph. In *ICDM*, pages 399–404, 2007. (Cited on pages 2, 17, 18, 19, and 36.)

[FNBB06] Eugene Fratkin, Brian T Naughton, Douglas L Brutlag, and Serafim Batzoglou. Motifcut: regulatory motifs finding with maximum density subgraphs. *Bioinformatics*, 22(14):e150–e157, 2006. (Cited on page 2.)

[GDC10] Derek Greene, Donal Doyle, and Padraig Cunningham. Tracking the evolution of communities in dynamic social networks. In *ASONAM*, 2010. (Cited on page 25.)

[GGT16] Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Top-$k$ overlapping densest subgraphs. *Data Mining and Knowledge Discovery*, 30(5):1134–1165, 2016. (Cited on page 79.)

[GIJ$^+$01] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, pages 491–500, 2001. (Cited on page 28.)

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. 1979. (Cited on pages 14 and 19.)

[GK01] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD Record*, volume 30, pages 58–66, 2001. (Cited on page 45.)

[GKT05] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *PVLDB*, pages 721–732, 2005. (Cited on page 21.)

[GM12] L. Getoor and A. Machanavajjhala. Entity resolution: Theory, practice and open challenges. *PVLDB*, 5(12):2018–2019, 2012. (Cited on page 8.)

[Gol84] Andrew V Goldberg. *Finding a maximum density subgraph*. University of California Berkeley, CA, 1984. (Cited on pages 21 and 76.)

[GW95] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *JACM*, 42(6):1115–1145, 1995. (Cited on page 21.)

[GYK12] Ziyu Guan, Xifeng Yan, and Lance M Kaplan. Measuring two-event structural correlations on graphs. *PVLDB*, 5(11):1400–1411, 2012. (Cited on page 26.)

[HBW$^+$05] J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *Journal of Computational Biology*, 12(6):657–671, 2005. (Cited on pages 2 and 14.)

[HCD$^+$94] Lawrence B Holder, Diane J Cook, Surnjani Djoko, et al. Substucture discovery in the subdue system. In *KDD*, pages 169–180, 1994. (Cited on pages 18 and 36.)

[HM16] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016. (Cited on page 57.)

[HS95] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD*, pages 127–138, 1995. (Cited on page 28.)

[HW13] Jialong Han and Ji-Rong Wen. Mining frequent neighborhood patterns in a large labeled graph. In *CIKM*, 2013. (Cited on page 19.)

[HWPY04] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: mining maximal frequent subgraphs from graph databases. In *KDD*, pages 581–586, 2004. (Cited on pages 14 and 15.)

[HYH$^+$05] Haiyan Hu, Xifeng Yan, Yu Huang, Jiawei Han, and Xianghong Jasmine Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21:i213–i221, 2005. (Cited on pages 14, 20, and 24.)

[Ita15] Telecom Italia. Telecommunications - tn to tn. `http://doi.org/10.7910/DVN/KCRS61`, 2015. (Cited on page 90.)

[IWM00] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *PKDD*, pages 13–23, 2000. (Cited on page 14.)

[JBC$^+$15] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A general suspiciousness metric for dense blocks in multimodal data. In *ICDM*, 2015. (Cited on page 26.)

[JCZ10] C. Jiang, F. Coenen, and M. Zito. Frequent sub-graph mining on edge weighted graphs. In *DAWAK*, pages 77–88. 2010. (Cited on page 17.)

[JHS$^+$10] Prateek Jain, Pascal Hitzler, Amit P Sheth, Kunal Verma, and Peter Z Yeh. Ontology alignment for linked open data. In *International semantic web conference*, pages 402–417, 2010. (Cited on page 8.)

[JKHB11] Shawana Jamil, Azam Khan, Zahid Halim, and A Rauf Baig. Weighted muse for frequent sub-graph pattern finding in uncertain dblp data. In *ICITA*, pages 1–6, 2011. (Cited on page 20.)

[JKS18] Bismita Jena, Cynthia Khan, and Rajshekhar Sunderraman. Sparkfsm: A highly scalable frequent subgraph mining approach using apache spark. In *ICDMW*, pages 990–997, 2018. (Cited on page 121.)

[JVB+05] W. Jiang, J. Vaidya, Z. Balaporia, C. Clifton, and B. Banich. Knowledge discovery from transportation network data. In *ICDE*, pages 1061–1072, 2005. (Cited on pages 2 and 14.)

[JWL+11] Xin Jin, Chi Wang, Jiebo Luo, Xiao Yu, and Jiawei Han. Likeminer: a system for mining the power of'like'in social media networks. In *KDD*, pages 753–756, 2011. (Cited on page 4.)

[JWYZ07] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. Gstring: A novel approach for efficient search in graph databases. In *ICDE*, pages 566–575, 2007. (Cited on page 2.)

[Kar72] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. 1972. (Cited on page 21.)

[KG00] Minoru Kanehisa and Susumu Goto. Kegg: kyoto encyclopedia of genes and genomes. *Nucleic acids research*, 28(1):27–30, 2000. (Cited on page 3.)

[KH09] Min-Soo Kim and Jiawei Han. A particle-and-density based evolutionary clustering method for dynamic networks. *PVLDB*, 2(1), 2009. (Cited on page 24.)

[KJDR08] Mahboob Alam Khalid, Valentin Jijkoun, and Maarten De Rijke. The impact of named entity normalization on information retrieval for question answering. In *European Conference on Information Retrieval*, pages 705–710, 2008. (Cited on page 28.)

[KK01] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM*, pages 313–320, 2001. (Cited on page 14.)

[KK04] Michihiro Kuramochi and George Karypis. Grew: A scalable frequent subgraph discovery algorithm. In *ICDM*, 2004. (Cited on page 18.)

[KK05] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *DMKD*, 11(3):243–271, 2005. (Cited on pages 14, 18, and 36.)

[KS09] Samir Khuller and Barna Saha. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*, pages 597–608, 2009. (Cited on pages 77 and 80.)

[KSS06] N. Koudas, S. Sarawagi, and D. Srivastava. Record linkage: similarity measures and algorithms. In *SIGMOD*, pages 802–803, 2006. (Cited on page 8.)

[LDMS11] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. Linking temporal records. *PVLDB*, 4(11):956–967, 2011. (Cited on pages 9, 10, 30, 31, and 122.)

[LGG18] Preethi Lahoti, Kiran Garimella, and Aristides Gionis. Joint non-negative matrix factorization for learning ideological leaning on twitter. In *WSDM*, 2018. (Cited on page 90.)

[LHW10] W. Y. Lin, K. W. Huang, and C. A. Wu. Mcfptree: An fp-tree-based algorithm for multi-constraint patterns discovery. *IJBIDM*, 5(3):231–246, 2010. (Cited on page 4.)

[Lia05] T Warren Liao. Clustering of time series data—a survey. *Pattern recognition*, 38(11):1857–1874, 2005. (Cited on page 32.)

[LLH17] Furong Li, Mong Li Lee, and Wynne Hsu. Profiling entities over time in the presence of unreliable sources. *TKDE*, 29(7):1522–1535, 2017. (Cited on pages 10 and 31.)

[LLHT15] Furong Li, Mong Li Lee, Wynne Hsu, and Wang-Chiew Tan. Linking temporal records for profiling entities. In *SIGMOD*, pages 593–605, 2015. (Cited on pages 10, 30, and 122.)

[Llo82] S. P. Lloyd. Least squares quantization in pcm. *Trans. Inf. Theory*, 28(2):129–137, 1982. (Cited on page 46.)

[LWLG19] Yiming Lin, Hongzhi Wang, Jianzhong Li, and Hong Gao. Efficient entity resolution on heterogeneous records. *TKDE*, 2019. (Cited on page 107.)

[LZG12] Jianzhong Li, Zhaonian Zou, and Hong Gao. Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *VLDBJ*, 21(6):753–777, 2012. (Cited on pages 2 and 20.)

[MAB$^+$10] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *SIGMOD*, pages 135–146, 2010. (Cited on pages 48 and 49.)

[Mac77] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. (Cited on page 38.)

[MBA$^+$09] Mary McGlohon, Stephen Bay, Markus G Anderle, David M Steier, and Christos Faloutsos. Snare: a link analytic system for graph labeling and risk detection. In *KDD*, pages 1265–1274, 2009. (Cited on page 26.)

[MBR$^+$13] Misael Mongiovi, Petko Bogdanov, Razvan Ranca, Evangelos E Papalexakis, Christos Faloutsos, and Ambuj K Singh. Netspot: Spotting significant anomalous regions on dynamic networks. In *ICDM*, 2013. (Cited on page 26.)

[MCD$^+$07] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, pages 342–350, 2007. (Cited on page 8.)

[MHW$^+$17] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jinpeng Huai. Fast computation of dense temporal subgraphs. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 361–372, 2017. (Cited on page 22.)

[MIM15] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? In *HotOS*, volume 15, pages 14–14, 2015. (Cited on page 56.)

[MK06] Matthew Michelson and Craig A Knoblock. Learning blocking schemes for record linkage. In *AAAI*, volume 6, pages 440–445, 2006. (Cited on pages 8 and 28.)

[MLVP16] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: a new way of searching. *VLDBJ*, pages 1–25, 2016. (Cited on page 3.)

[MNU08] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *KDD*, pages 169–178, 2008. (Cited on page 28.)

[MZ98] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *PVLDB*, volume 98, pages 24–27, 1998. (Cited on page 107.)

[NC03] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003. (Cited on pages 2 and 14.)

[New04] Mark EJ Newman. Analysis of weighted networks. *Physical review E*, 70(5), 2004. (Cited on page 4.)

[NGMG17] Muhammad Anis Uddin Nasir, Aristides Gionis, Gianmarco De Francisci Morales, and Sarunas Girdzijauskas. Fully dynamic algorithm for top-k densest subgraphs. In *CIKM*, pages 1817–1826, 2017. (Cited on page 26.)

[NH10] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. 2010. (Cited on page 8.)

[NK04] S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *KDD*, pages 647–652, 2004. (Cited on page 14.)

[PHMAZ00] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *PAKDD*, pages 396–407, 2000. (Cited on pages 2 and 14.)

[PINF11] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *WSDM*, 2011. (Cited on pages 9 and 28.)

[PIP$^+$12] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederee, and Wolfgang Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *TKDE*, 25(12):2665–2682, 2012. (Cited on pages 9 and 28.)

[PIS11] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *ICDT*, pages 355–366, 2011. (Cited on page 20.)

[PJR17] André Petermann, Martin Junghanns, and Erhard Rahm. Dimspan: Transactional frequent subgraph mining with distributed in-memory dataflow systems. In *BDCAT*, pages 237–246, 2017. (Cited on page 121.)

[PJZ05] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005. (Cited on pages 2 and 25.)

[PKPN14] G. Papadakis, G. Koutrika, T. Palpanas, and W. Nejdl. Meta-blocking: Taking entity resolution to the next level. *TKDE*, 26(8):1946–1960, 2014. (Cited on pages 9, 29, 112, 114, and 117.)

[PLM08] A. N. Papadopoulos, A. Lyritsis, and Y. Manolopoulos. Skygraph: an algorithm for important subgraph discovery in relational graphs. *DMKD*, 17(1):57–76, 2008. (Cited on pages 4 and 19.)

[PN07] F. Pennerath and A. Napoli. Mining frequent most informative subgraphs. In *Mining And Learning With Graphs*, 2007. (Cited on pages 14 and 16.)

[PPPK16a] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Boosting the efficiency of large-scale entity resolution with enhanced meta-blocking. *Big Data Research*, 6:43–63, 2016. (Cited on pages 9 and 29.)

[PPPK16b] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016. (Cited on page 9.)

[QGY13] Guimin Qin, Lin Gao, and Jianye Yang. Significant substructure discovery in dynamic networks. *Current Bioinformatics*, 8(1), 2013. (Cited on page 24.)

[QZL$^+$18] Fengcai Qiao, Xin Zhang, Pei Li, Zhaoyun Ding, Shanshan Jia, and Hui Wang. A parallel approach for frequent subgraph mining in a single large graph using spark. *Applied Sciences*, 8(2):230, 2018. (Cited on page 121.)

[RAGT14] Polina Rozenshtein, Aris Anagnostopoulos, Aristides Gionis, and Nikolaj Tatti. Event detection in activity networks. In *KDD*, 2014. (Cited on page 21.)

[RC18a] Thilina Ranbaduge and Peter Christen. Privacy-preserving temporal record linkage. In *ICDM*, pages 377–386, 2018. (Cited on page 31.)

[RC18b] Thilina Ranbaduge and Peter Christen. A scalable privacy-preserving framework for temporal record linkage. *Knowledge and Information Systems*, pages 1–34, 2018. (Cited on page 31.)

[RS09] S. Ranu and A. K. Singh. Graphsig: A scalable approach to mining significant subgraphs in large graph databases. In *ICDE*, pages 844–855, 2009. (Cited on page 15.)

[RSK$^+$15] Stephen Ranshous, Shitian Shen, Danai Koutra, Steve Harenberg, Christos Faloutsos, and Nagiza F Samatova. Anomaly detection in dynamic networks: a survey. *Wiley Interdisciplinary Reviews: Computational Statistics*, 7(3):223–247, 2015. (Cited on page 78.)

[RTG17] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. Finding dynamic dense subgraphs. *TKDD*, 11(3):27, 2017. (Cited on pages 6 and 22.)

[S$^+$86] Richard Snodgrass et al. Temporal databases. *Computer*, (9):35–42, 1986. (Cited on page 9.)

[SAF$^+$14] Md Samiullah, Chowdhury Farhan Ahmed, Anna Fariha, Md Rafiqul Islam, and Nicolas Lachiche. Mining frequent correlated graphs with a new measure. *Expert Systems with Applications*, 41(4):1847–1863, 2014. (Cited on page 27.)

[SEK04] Michael Steinbach, Levent Ertöz, and Vipin Kumar. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics*, pages 273–309. 2004. (Cited on page 44.)

[SGV06] Vivek Sehgal, Lise Getoor, and Peter D Viechnicki. Entity resolution in geospatial data integration. In *GIS*, pages 83–90, 2006. (Cited on page 28.)

[SHF16] Kijung Shin, Bryan Hooi, and Christos Faloutsos. M-zoom: Fast dense-block detection in tensors with quality guarantees. In *ECML PKDD*, 2016. (Cited on page 26.)

[SKZ$^+$15] Neil Shah, Danai Koutra, Tianmin Zou, Brian Gallagher, and Christos Faloutsos. Timecrunch: Interpretable dynamic graph summarization. In *KDD*, 2015. (Cited on page 25.)

[SMJZ12] Arlei Silva, Wagner Meira Jr, and Mohammed J Zaki. Mining attribute-structure correlated patterns in large attributed graphs. *PVLDB*, 5(5):466–477, 2012. (Cited on page 2.)

[SPPB18] G. Simonini, G. Papadakis, T. Palpanas, and S. Bergamaschi. Schema-agnostic progressive entity resolution. In *ICDE*, 2018. (Cited on page 29.)

[SPTT16] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. Best friends forever (bff): Finding lasting dense subgraphs. *arXiv preprint arXiv:1612.05440*, 2016. (Cited on page 6.)

[SPTT18] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. Finding lasting dense subgraphs. *Data Mining and Knowledge Discovery*, pages 1–29, 2018. (Cited on page 23.)

[SQC13] Prakash Shelokar, Arnaud Quirin, and Oscar Cordón. Mosubdue: a pareto dominance-based multiobjective subdue algorithm for frequent subgraph mining. *Knowledge and information systems*, 34(1):75–108, 2013. (Cited on page 16.)

[SSTW01] Michael J Shaw, Chandrasekar Subramaniam, Gek Woo Tan, and Michael E Welge. Knowledge management and data mining for marketing. *Decision Support Systems*, 31:127–137, 2001. (Cited on pages 3 and 4.)

[SWD16] Q. Song, Y. Wu, and X. L. Dong. Mining summaries for knowledge graph search. In *ICDM*, pages 1215–1220, 2016. (Cited on page 4.)

[TFS$^+$15] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. Arabesque: a system for distributed graph mining. In *SOSP*, pages 425–440, 2015. (Cited on pages 5, 34, 48, 49, and 72.)

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *ACM Communications*, 33(8):103–111, 1990. (Cited on page 49.)

[VCR14] Ivan F Videla-Cavieres and Sebastian A Rios. Extending market basket analysis with graph mining techniques: A real case. *Expert Systems with Applications*, 41(4):1928–1936, 2014. (Cited on page 2.)

[VGS02] Natalia Vanetik, Ehud Gudes, and Solomon Eyal Shimony. Computing frequent graph patterns from semistructured data. In *ICDM*, pages 458–465, 2002. (Cited on page 18.)

[VSG06] N. Vanetik, S. E. Shimony, and E. Gudes. Support measures for graph data. *Data Min. Knowl. Discov.*, 13(2):243–260, 2006. (Cited on pages 2 and 36.)

[WA10] Haixun Wang and Charu C Aggarwal. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*, pages 249–273. 2010. (Cited on page 3.)

[WCH$^+$17] Zhuo Wang, Qun Chen, Boyi Hou, Bo Suo, Zhanhuai Li, Wei Pan, and Zachary G Ives. Parallelizing maximal clique and k-plex enumeration over graph data. *Journal of Parallel and Distributed Computing*, 106:79–91, 2017. (Cited on pages 83, 111, 112, and 121.)

[WMK$^+$09] Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. Entity resolution with iterative blocking. In *SIGMOD*, pages 219–232, 2009. (Cited on page 28.)

[WRS17] Di Wu, Jiadong Ren, and Long Sheng. Uncertain maximal frequent subgraph mining algorithm based on adjacency matrix and weight. *International Journal of Machine Learning and Cybernetics*, pages 1–11, 2017. (Cited on page 20.)

[WZW$^+$05] Chen Wang, Yongtai Zhu, Tianyi Wu, Wei Wang, and Baile Shi. Constraint-based graph mining in large database. In *APWeb*, pages 133–144, 2005. (Cited on page 16.)

[YAMW13] Weiren Yu, Charu C Aggarwal, Shuai Ma, and Haixun Wang. On anomalous hotspot discovery in graph streams. In *ICDM*, 2013. (Cited on page 26.)

[YCHY08] Xifeng Yan, Hong Cheng, Jiawei Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *SIGMOD*, 2008. (Cited on page 15.)

[YH02] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724, 2002. (Cited on pages 2, 15, 27, and 39.)

[YH03] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. In *KDD*, pages 286–295, 2003. (Cited on pages 14 and 15.)

[YSLD12] J. Yang, W. Su, S. Li, and M. M. Dalkilic. Wigm: Discovery of subgraph patterns in a large weighted graph. In *SDM*, pages 1083–1094, 2012. (Cited on pages 2, 4, 5, 20, and 55.)

[YYG$^+$14] Yajun Yang, Jeffrey Xu Yu, Hong Gao, Jian Pei, and Jianzhong Li. Mining most frequently changing component in evolving graphs. *WWW*, 17(3), 2014. (Cited on page 22.)

[YYH04] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004. (Cited on pages 2 and 14.)

[YYW$^+$16] Yi Yang, Da Yan, Huanhuan Wu, James Cheng, Shuigeng Zhou, and John Lui. Diversified temporal subgraph pattern mining. In *KDD*, pages 1965–1974, 2016. (Cited on pages 6 and 25.)

[YZH05] X. Yan, X. J. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *KDD*, 2005. (Cited on page 23.)

[ZF06] Jian Zhang and Joan Feigenbaum. Finding highly correlated pairs efficiently with powerful pruning. In *CIKM*, pages 152–161, 2006. (Cited on page 82.)

[ZHOT16] Hongjie Zhai, Makoto Haraguchi, Yoshiaki Okubo, and Etsuji Tomita. A fast and complete algorithm for enumerating pseudo-cliques in large graphs. *IJDSA*, 2(3-4):145–158, 2016. (Cited on page 121.)

[ZLGZ10] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Mining frequent subgraph patterns from uncertain graph data. *TKDE*, 22(9):1203–1218, 2010. (Cited on page 20.)

[ZXW$^+$16] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016. (Cited on page 58.)

[ZYHP07] Feida Zhu, Xifeng Yan, Jiawei Han, and S Yu Philip. gprune: a constraint pushing framework for graph pattern mining. In *PAKDD*, pages 388–400, 2007. (Cited on page 16.)