Beyond Frequencies: Graph Pattern Mining in Multi-weighted Graphs

Giulia Preti
University of Trento
gp@disi.unitn.eu

Matteo Lissandrini
University of Trento
ml@disi.unitn.eu

Davide Mottin
Hasso Plattner Institute
davide.mottin@hpi.de

Yannis Velegrakis
University of Trento
velgias@disi.unitn.eu

ABSTRACT

Graph pattern mining aims at identifying structures that appear frequently in large graphs, under the assumption that frequency signifies importance. Several measures of frequency have been proposed that respect the *apriori* property, essential for an efficient search of the patterns. This property states that the number of appearances of a pattern in a graph cannot be larger than the frequency of any of its sub-patterns. In real life, there are many graphs with weights on nodes and/or edges. For these graphs, it is fair that the importance (score) of a pattern is determined not only by the number of its appearances, but also by the weights on the nodes/edges of those appearances. Scoring functions based on the weights do not generally satisfy the apriori property, thus forcing many approaches to employ other, less efficient, pruning strategies to speed up the computation. The problem becomes even more challenging in the case of multiple weighting functions that assign different weights to the same nodes/edges. In this work, we provide efficient and effective techniques for mining patterns in multi-weight graphs. We devise both an exact and an approximate solution. The first is characterized by intelligent storage and computation of the pattern scores, while the second is based on the aggregation of similar weighting functions to allow scalability and avoid redundant computations. Both methods adopt a scoring function that respects the apriori property, and thus they can rely on effective pruning strategies. Extensive experiments under different parameter settings prove that the presence of edge weights and the choice of scoring function affect the patterns mined, and hence the quality of the results returned to the user. Finally, experiments on datasets of different sizes and increasing numbers of weighting functions show that, even when the performance of the exact algorithm degrades, the approximate algorithm performs well and with quite good quality.

1 INTRODUCTION

Pattern mining in large graphs has attracted considerable attention, since it finds applications in many real world scenarios like fraud detection [31], biological structures identification [16], anticipation of user intention [33], graph similarity search [20], traffic control [21], and query optimization [42]. It has been studied for graph collections [41], for attributed [35], probabilistic [26], or even generic large graphs [10]. The goal is to identify patterns that occur frequently, given that frequency indicates importance. An interesting property regarding frequency is that a pattern cannot be more frequent that any of its sub-patterns, known as the *apriori* property. This property enables efficient implementations [43], as it ensures that the frequency of a pattern decreases monotonically as the pattern grows in size, thus allowing 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.

In graph databases the frequency of a pattern has been effectively computed as the number of distinct graphs containing an appearance of the pattern. However, the same implementation cannot be used in single large graphs, as each pattern would have frequency either equal to 0 or to 1. Furthermore, if we simply define the frequency as the number of distinct occurrences of the pattern, we may break the apriori property [38]. In fact, this implementation counts every overlap that may occur among the occurrences, hence assigning larger frequencies to larger patterns, causing an unwanted (and unjustified) skew in the value of importance for some patterns. For this reason, alternative metrics have been considered in the literature [6, 11, 38], with the more prevalent one being the MNI support, as it enjoys high effectiveness [10].

Many real world scenarios are naturally modeled through weighted graphs, and in these cases, the importance of a pattern should be determined not only by the frequency, but also by the weights of its appearances. Examples include the discovery of metabolic pathways in genomic networks [23], where weights indicate strength between genomes [8], the identification of topics of interest in large knowledge graphs [29], where weights quantify the degree a piece of data is qualified as an answer to a user [39], or the detection of common problematic cases in computer networks, where weights indicate congestion [5]. Unfortunately, weighted graphs do not possess the apriori property, since the weights of the extra edges/nodes of a larger pattern may offset its lower frequency. As a consequence, some works that considered weighted graphs for pattern mining proposed solutions that are less efficient than those based on the apriori property [43].

A requirement in modern applications is to offer personalized products and services rather than generic preferences [34]. Such generic preferences suit the user on average but fail to deliver the right answer for each specific user. The same argument holds for graph patterns. For instance, social network systems record user interactions [22] and activities [4] and build graphs by modeling the relationships among users and web content to find frequent patterns of interactions [30]. Advertisers subsequently exploit such patterns to target the right customer for a certain product. Some patterns of interactions may be more important than others to an advertiser depending on the product or the specific business model; in such case, multiple weights are valuable. Other

Series ISSN: 2367-2005

10.5441/002/edbt.2018.16

examples include online retailers like Amazon, which build large graphs on product co-purchases and then exploit the discovered patterns to recommend future offers [36]. 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 [34]. Such examples highlight the need for a solution that accounts for the individual preferences expressed as a multi-weighted graph, as opposed to "one size fits all" solutions. However, the straightforward approach to multi-weighted pattern mining that runs the mining algorithm on each weighted graph separately, is clearly impractical due to the graph size and the large number of users.

In this paper, we propose a novel approach to mine patterns in weighted graphs that goes beyond frequencies, yet has performance not significantly different from the pattern mining in unweighted graphs. We achieve this by defining a family of scoring functions that are based on the MNI score [6], a metric that is widely used in graph mining due to its characteristic of respecting the apriori property, while being efficient to compute. The solution we have devised is modeled as a constraint satisfaction problem (CSP), as proposed also for unweighted pattern mining [10], and implements the pattern growth approach, as introduced by gSpan [41]. Furthermore, we extend the idea above for the case of graphs with multiple weights on their edges/nodes. To avoid running the algorithm one time for each different weighting function, we compute all the scores of each pattern at the moment we are visiting it, and keep the patterns that return a high score with respect to at least one weighting function.

In particular we make the following contributions:

(a) We extend the task of pattern mining in weighted large graphs for a novel family of scoring functions based on the MNI support [6] (Section 2).
(b) We introduce and formally define the problem of pattern mining in multi-weight graphs with different weighting functions.
(c) We devise two efficient and effective techniques for solving the pattern mining problem on weighted graphs (Section 3). The first one is an exact solution, called ReSuM, that is less time and space consuming than the (naive) brute force method. It avoids redundant revisits of the graph, by aggregating and performing once multiple computations on the same parts of the graph, and storing the relevant patterns in a compact way. The second one is a conservative approximate solution, called ReSuM *approximate*, that reduces the number of weighting functions to consider, by aggregating those having a high probability to generate similar results (Section 4) into a single representative function. In addition, we show that this method introduces only few false positives, while running considerably faster than the exact approach.
(d) We study four different scoring functions (all based on the MNI support) for devising the score of a pattern in an efficient way (Section 5).
(e) We evaluate our approach with an extensive set of experiments and discuss our findings (Section 7).

2 PROBLEM DEFINITION

We assume the existence of a countable set of labels Σ that includes the special symbol $\perp$, and a set $\mathcal{I}_0^1 = [0,1] \cup \{\perp\}$ of weights. The symbol $\perp$ in Σ and $\mathcal{I}_0^1$ is used to denote *no label* and *no weight*, respectively. A weighted graph is a structure that consists of a set of nodes, a set of edges between them, an assignment of labels to all the nodes and edges, and an assignment of weights to all the edges.

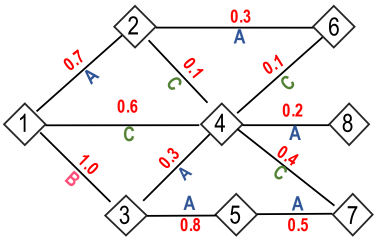

Figure 1: Example of a edge-labeled, weighted graph.

Definition 2.1. A **weighted labeled graph**, or simply a **graph**, is a tuple $\langle V, E, \ell, \omega \rangle$ where V is a set of vertices, $E \subseteq V \times V$ is a set of edges, $\ell : E \cup V \rightarrow \Sigma$ is a labeling function, and $\omega : E \rightarrow \mathcal{I}_0^1$ is a weighting function. The symbol $\mathcal{G}$ is used to denote the set of all the possible graphs.

Note that we assume weights on edges only, mainly for presentation purposes. Weights on nodes can also be considered with no need for any major modification. A graph $S{:}\langle V_S, E_S, \ell, \omega \rangle$ is said to be a **subgraph** of another graph $G{:}\langle V_G, E_G, \ell, \omega \rangle$, denoted as $S \sqsubseteq G$, if $V_S \subseteq V_G$ and $E_S \subseteq E_G$. Note that the two graphs have the same labeling and weighting function.

To express the fact that two graphs have the same topological structure, we use 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 2.2. 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: $\forall \langle u, v \rangle \in E : \langle \phi(u), \phi(v) \rangle \in E'$ and $\ell(\langle u, v \rangle) = \ell'(\langle \phi(u), \phi(v) \rangle)$.

A graph G may have multiple isomorphic graphs. To collectively represent those graphs, we introduce the concept of *pattern*. Intuitively, a pattern is a graph with no weights, serving as a representative of a set of isomorphic graphs and describing their common structure.

Definition 2.3. A **pattern** is a graph $\langle V, E, \ell, \omega \rangle$, such that $\forall e \in E : \omega(e) = \perp$. The symbol $\mathcal{P}$ denotes the set of all possible patterns. Given a graph G, and a pattern P, the *support set* of the pattern P is the set $S_G(P) = \{g | g \sqsubseteq G \wedge g \simeq P \wedge P \in \mathcal{P}\}$. Each element in $S_G(P)$ is referred to as an *appearance* (or matching) of P in G.

By definition, the support set of P is the set of all the subgraphs of G that are isomorphic to P. By abuse of notation we write $P \sqsubseteq G$ and call the pattern P a subgraph of G if its support set is non-empty. Then, We denote by ϕ_g^P the bijection that maps an isomorphic subgraph g of G to the pattern P.

Given a **scoring function** $f{:}\,\mathcal{P} \times \mathcal{G} \rightarrow \mathbb{R}$, we will refer to the value $f(P, G)$ as the **score** of P in G. Graph pattern mining is the task that aims at identifying those patterns that have score higher than a threshold τ, or the k patterns with the highest score [10]. A natural scoring function is the one that returns the cardinality of $S_G(P)$, i.e., the number of appearances of the pattern P in the graph G, and the patterns identified by this function are called *frequent patterns.* Nevertheless, it has been shown that this simple function violates the a-priory property, due to the presence of overlapping isomorphisms in G [6]. As an example, the frequency of the pattern $P_1 : [v_1]{-}B{-}[v_2]{-}A{-}[v_3]$ in the graph in Figure 1

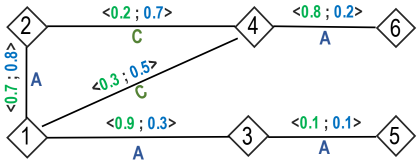

Figure 2: Graph with two weights $< \omega_1, \omega_2 >$ on each edge.

is 3, while the frequency of its sub-pattern $P_2 : [v_1] - B - [v_2]$ is 1. For this reason a number of works have investigated alternative scoring functions [11, 15, 25, 38]. Among them, the MNI support is highly effective and efficient to compute [6].

Definition 2.4. Given a graph $G : \langle V, E, \ell, \omega \rangle$, the **MNI support** of a pattern $P{:}\langle V_P, E_P, \ell_P, \omega_P \rangle$ in G is the number $MNI(P, G) = \min_{v' \in V_P} |\mathcal{N}(G, v')|$ where $\mathcal{N}(G, v') = \{v \mid v{\in}V \wedge \exists g{\in}S_G(P)$ such that $\phi_g^P(v){=}v'\}$.

Intuitively, the set $\mathcal{N}(G, v')$ contains all the nodes of G that are mapped to the pattern node v' by some isomorphism ϕ_g^P from g to P. Then, the MNI support is the minimum cardinality of this set across all the nodes of the pattern P. We can define similar sets also for the pattern edges, i.e., for each $e' \in E_P$, the set $\mathcal{E}(G, e') = \{e | e \in E \wedge \exists g \in S_G(P)$ such that $\phi_g^P(e) = e'\}$ contains all the edges of G that are mapped to the pattern edge e' by some isomorphism $\phi_g^P(e)$. Consider, for instance, the graph G in Figure 1 and the pattern $P : [v_1] - B - [v_2] - A - [v_3]$. Since $\mathcal{N}(G, v_1) = \{1, 3\}$, $\mathcal{N}(G, v_2) = \{1, 3\}$, and $\mathcal{N}(G, v_3) = \{2, 4, 5\}$, the MNI support of P is 2. On the other hand, the number of appearances of P in G is 3: $S_G(P) = \{[3] - B - [1] - A - [2], [1] - B - [3] - A - [4], [1] - B - [3] - A - [5]\}$.

In the presence of weights on edges, the score of a pattern cannot be based only on the frequency, but should strike a balance between frequency and weights, allowing also the weights to play a role in assessing the relevance of the pattern. Thus, there is a need for a different scoring function that looks beyond the structure of the subgraphs, by considering the importance of their edges as well. In this case, we talk about *weighted frequent patterns*, or *relevant patterns*. This alternative scoring function, however, has to be carefully selected to satisfy the apriori property [43].

Furthermore, if there are multiple weighting functions, i.e., several functions that assign weights on the graph edges/nodes, then the pattern mining task must be carried out for each individual function. An example of graph with multiple weights on the edges is illustrated in Figure 2. Such situation leads to the following specification of the mining task.

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

3 SCORE-BASED PATTERN MINING

Our solution to the pattern mining on weighted graphs problem consists of two steps. The first step is the identification of the frequent patterns and the elimination of the appearances that do not satisfy the constraints on the weights imposed by the scoring function used. In the second step, a score is computed for each pattern (and for each weighting function in the case of multi-weighted graphs) in terms of the appearances in its support set that were selected in the first step.

Algorithm 1 RELEVANTPATTERNMINING

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

function patternExtension($G, g, \tau, fE$)
    $Cand \leftarrow \emptyset$; $\mathfrak{S} \leftarrow \emptyset$
    for each $e \in fE$ do
        $Cand \leftarrow Cand \cup \{g \diamond e\}$
    for each $c \in Cand$ do
        $(score, sup) \leftarrow$ examinePattern($G, c$)
        if $sup \geq \tau$ then
            $\mathfrak{S} \leftarrow \mathfrak{S} \cup$ patternExtension($G, c, \tau, fE$)
        if $score \geq \tau$ then
            $\mathfrak{S} \leftarrow \mathfrak{S} \cup \{c\}$
    return $\mathfrak{S}$
```

3.1 Assessing the relevance of a pattern

A scoring function can have many different properties, which may be desirable for certain applications but not for others. As a consequence there may be no scoring function that is consistently better than others in all the applications. Therefore, in this work we do not advocate a single one-size-fits-all scoring function, but we propose a framework that can accommodate a wide range of functions.

Assuming w.l.o.g. that larger weights signify higher importance, the function f must satisfy the following key properties:

P1: the score $f(P, G)$ monotonically increases with the weights of its appearances.

P2: the score $f(P, G)$ monotonically increases with the number of appearances with large weights.

P3: f is *MNI-compatible*, i.e., $f(P, G){\geq}\tau \implies MNI(P, G){\geq}\tau$.

Properties **P1** and **P2** are a natural consequence of our assumption on the importance of the weights, while **P3** ensures the time practicality of the solution.

3.2 Mining weighted graphs

Finding the frequent patterns on weighted graphs requires the computation of the frequency and the score of each pattern. To this end, we propose ReSuM, an efficient and effective general algorithm for *any* MNI-compatible score that exploits the pruning power of the anti-monotonicity property of the MNI support.

We model the frequent subgraph mining as a *constraint satisfaction problem* (CSP) [9]. An instance of the CSP problem 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 then translated into $CSP(P) = (X_P, D_P, C_P)$, so that any solution for $CSP(P)$ 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

Algorithm 2 EXAMINEPATTERN

```
Input: Graph $G$:$\langle V, E, \ell, \omega\rangle$, pattern $P$, score threshold $\tau$
Output: Score and MNI support of $P$
 1: for each $v \in V_P$ 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 each $v \in V_P$ 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 each $n \in D_v$ 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 each $n' \in V_g, v' \in V_P$ 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 ($score$, $mni$)
```

nodes isomorphic to v, and C includes consistency constraints that enforce a topology isomorphic to that of P [28]. Then, for each candidate node $n \in D_v$ we search for a valid assignment 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., $\text{argmin}_{D_v \in D_P} |D_v|$, corresponds to the MNI support of P, as defined in Definition 2.4. Therefore, given a score threshold τ, P is frequent if each variable in X_P has at least τ distinct valid assignments. This means that if the size of some domain D_u is lower than τ, P cannot be frequent. Notice that in general not all the matching subgraphs of a pattern satisfy the constraints on the weights forced by the scoring function used, and thus we must additionally check each of them to determine if it contributes to the score of the pattern. The aggregated score is then computed considering only the matches not discarded.

Algorithm 1 outlines the ReSuM framework. First, the relevant and the frequent edges are found (Lines 1-2). Then, each subgraph is recursively extended following the pattern-growth approach introduced by gSpan [41] (Line 5), until no other 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 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 relevance *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 subsequent iteration, 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 have been examined, or when the size of some domain becomes lower than τ, as in this case P can be neither relevant nor frequent (Line 11). In the first case, instead, the MNI support and the relevance *score* of P are calculated and returned. We refer to Section 5 for a discussion about suitable scoring functions that can be 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. Since we enforce the use of MNI-compatible scoring functions, the MNI support of P is an upper-bound of its *score*, and thus the pruning strategy ensures that all the relevant patterns are returned.

Complexity. Even though the computation of the automorphisms ($O(|V_P|^{|V_P|})$) and the pruning strategy improve the expected performance of the algorithm, in the worst case it takes $C = O(2^{|V|^2}|V|^{|V_P|})$ time, which is exponential in the number of nodes and the size of the patterns. In particular, $O(2^{|V|^2})$ is the time required to compute all the patterns in G, and $O(|V|^{|V_P|})$ is that needed to find all the isomorphisms of a pattern P.

3.3 Mining in multi-weighted graphs

In the case of multiple edge weights assigned by m weighting functions $W = \{\omega_1, \ldots, \omega_m\}$, the naïve approach for solving the Pattern Mining in Multi-Weighted Graphs problem runs Algorithm 1 $|W|$ times, once for each function. Evidently, this approach becomes impractical for large m, as the process of mining the patterns is computationally intense. In fact, this process would take $O(C^m)$ to terminate.

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 and 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 assigned by ω_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 only if at least one of its scores is larger than τ. As a further optimization,

Algorithm 3 examineSubgraphMulti

```
Input: Graph $G{:}\langle V, E, \ell, W\rangle$, pattern $P$, score threshold $\tau$
Output: Scores and MNI support of $P$
for each $v \in V_P$ do
    $D_v \leftarrow \{v' {\in} V | \ell(v') = \ell(v)\}$
    for each $i \in 1, \dots, |W|$ do
        $SUP_v[i] \leftarrow \emptyset$
$\mathcal{A} \leftarrow$ automorphisms of $P$
structuralConsistency($\{D_v | v \in V_P\}, P$)
for each $v \in V_P$ do
    if $\exists\, w = \mathcal{A}(v)\, s.t.\, D_w$ already computed then
        $D_v \leftarrow D_w$
        continue
    structuralConsistency($\{D_v | v \in V_P\}, P$)
    if $\exists D_u\, s.t.\, |D_u| < \tau$ then return $(\{-1, \dots, -1\}, -1)$
    for each $n \in D_v$ do
        search for $g$ s.t. $g \simeq P \wedge n \in V_g \wedge n \mapsto v$
        if $g \neq Nil$ then
            $VAL \leftarrow$ isValid($g, W$)
            for each $n' \in V_g, v' \in V_P$ s.t. $n' \mapsto v'$ do
                mark $n'$ in $D_{v'}$
                for each $i \in 1, \dots, |W|$ do
                    if $VAL[i]$ then
                        $SUP_{v'}[i] \leftarrow SUP_{v'}[i] \cup \{n'\}$
        else
            remove $n$ from $D_v$
$\mathcal{S} \leftarrow$ relevanceScores($\{SUP_v | v \in V_P\}$)
$mni \leftarrow min_{v \in V_P} |D_v|$
return ($\mathcal{S}$, $mni$)
```

instead of storing in memory the sets of relevant patterns for each function ω_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 ω_i.

4 APPROXIMATE ALGORITHM

The exact algorithm introduced in Section 3 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 $\omega_1, \dots, \omega_m \in W$.

The ReSuM *approximate* algorithm first generates $k \ll m$ representative functions ω_j^*, by clustering and aggregating the original functions ω_i. Then, it runs Algorithm 3 to compute k sets of relevant patterns $R_1^*, \dots, R_k^*$, which are used to build m approximate sets of relevant patterns $A_1, \dots, A_m$, returned in place of the exact sets $R_1, \dots, R_m$. Clearly, the quality of the approximate result depends on the way the representative functions are generated. With our implementation, we aim at returning a set A_j for each ω_i that resembles the exact set R_i as much as possible.

4.1 Generation of the representative functions

The generation of the representative functions is shown in Algorithm 4 and consists of three steps. First, each weighting function $\omega_i \in W$ is transformed into a feature vector (Line 1). Secondly, the weighting functions are clustered into k groups of similar functions (Line 2). Thirdly, the set of k representative functions $W^* = \{\omega_1^*, \dots, \omega_k^*\}$ is returned (Lines 3-4).

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 $W^*$
```

Algorithm 5 createBucketFeatureVectors

```
1: function createBucketFeatureVectors($E, W, b$)
2:     for each $l \in \Sigma_E$ do
3:         $BucketList_l \leftarrow$ computeBucketLimits($E_l, W, b$)
4:         for each $\omega_i \in W$ do
5:             $\mathbf{r}_i^l \leftarrow$ fillBuckets($E_l, \omega_i, BucketList_l$)
6:     for each $\omega_i \in W$ do
7:         $\mathbf{r}_i \leftarrow$ concate($\{\mathbf{r}_i^l | l \in \Sigma_E\}$)
8:     return $\{\mathbf{r}_1, \dots, \mathbf{r}_{|W|}\}$
```

Creation of the feature vectors.

In the first step, we construct a feature vector r_i for each ω_i, which is used in the second step to determine the similarities between the functions. Since our final goal is to assign a set of patterns A_i to each ω_i that is as close as possible to the exact set R_i, a straightforward choice is to use the edge weights as features. We call this approach *full-vector strategy*. According to this strategy, Procedure 1 decides an ordering of the graph edges and creates m vectors $\mathbf{r}_1, \dots, \mathbf{r}_m$ of size $|E|$, where $\mathbf{r}_i[x]$ is the weight assigned by ω_i to the edge in the x^{th} position.

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 [37].

Thus, we propose also a more efficient approach called *bucket-based strategy*, which overcomes the problem of high dimensionality by considering the edge labels in place of the graph edges, as features to build the vectors. 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 $\mathbf{r}_1, \dots, \mathbf{r}_m$ each of size $|\Sigma_E| \cdot b$, where Σ_E indicates the set of distinct edge labels. In particular, each vector $\mathbf{r}_i$ is the concatenation of $|\Sigma_E|$ summaries of the edge-weights of ω_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 admissible weights $[0, 1]$ into b of 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 the function ω_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 [13], which assigns the input values to buckets, while trying to balance the number of elements in each bucket. Thus, we consider all the weights assigned by 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 $A \mid C$, and the two weighting functions ω_1 and ω_2 in Figure 1, 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.

Identification of similar functions. Procedure computeClustering (Algorithm 4, Line 2) implements the Lloyd's clustering algorithm [27], which identifies groups of similar ω_i by comparing the feature vectors $\mathbf{r}_1, \ldots, \mathbf{r}_m \in \mathcal{F}$ using the cosine similarity.

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 this strategy allows the discovery of better separated clusters. Moreover, the algorithm can either be executed until convergence or can be run 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 some maximum value k_{max}, and then returns the clustering with largest silhouette coefficient.

Generation of the representative functions. Given the set of clusters C, Procedure generateMaxWeightVectors (Algorithm 4, Line 3) generates a representative function ω_j^* for each cluster C_j. Different choices of ω_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 $\omega_l \in C_j$. However, as stated in the following theorem, we resort to take the *maximum* among the weights to prevent missing any relevant pattern:

Theorem 4.1. *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 ω_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 through the scoring function f can contribute to the score of a pattern. Moreover, the larger the weights of a subgraph, the higher the chances that such subgraph fulfill those constraints. Since the function ω_i^* assigns to each edge $e \in E$ the largest weight among those of the weighting functions in the cluster C_i, i.e., $\forall \omega_j \in C_i$, $\omega_i^*(e) \geq \omega_j(e)$, the chances that a matching subgraph contributes to the score of a pattern is higher for ω_i^* than for any $\omega_j \in C_i$. It follows that $\forall \omega_j \in C_i\ f(P, \omega_i^*) \geq f(P, \omega_j)$, so if a pattern is relevant for some $\omega_j \in C_i$, it is also relevant for ω_i^*. Thus, the set of mined patterns 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 ω_i using the patterns in the set R_j^* for $j \leq k$. $\omega_i \in C_j$, i.e., each function ω_i receives the set of relevant patterns of the cluster to which it belongs.

4.2 Quality of ReSuM *approximate*

The ReSuM *approximate* algorithm reduces the problem of mining patterns 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. The most common quality measure used in the literature is the accuracy, which is defined in terms of precision and recall. In our case, since Theorem 4.1 ensures a total recall, we consider the average precision of the sets A_i with respect to the exact sets R_i:

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

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. As shown in Section 7, the distance between two patterns can be calculated using the normalized Levenshtein distance, and the distance between two pattern sets as the average normalized Levenshtein distance among the pairs of closest patterns in the two sets. According to this measure, A_i is a good solution for ω_i if the patterns in A_i have structure and labels similar to the patterns in R_i.

5 PATTERN EVALUATION

A number of scoring function satisfying properties **P1**, **P2**, and **P3** can be proposed and implemented in Procedure isValid and relevanceScore in Algorithm 2 and 3. Nevertheless, to demonstrate the flexibility of our framework, we propose here four different scoring functions that can be used to assess the relevance of a pattern in a weighted graph. They are called *ALL*, *ANY*, *SUM* and *AVG*. We chose 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, as they are defined by the MNI support of the pattern over a specific restriction of its support set, they are *MNI-compatible* by definition, and thus they preserve the apriori property.

ALL, *ANY*, *SUM* and *AVG* differ in the choice of which subgraphs they include in the support sets of the patterns P and in how they aggregate the edge weights of such subgraphs. In particular, *ALL*, *ANY*, and *SUM* rely on an additional system-dependent parameter, called relevance threshold α, 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 of the four scoring functions.

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

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

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 2 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*.

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

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 α, then the sum of all its weights is at least α, 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 α=0.4 in Figure 2 are considered also by *SUM*, whereas the subgraph g : [3]–A–[4]–A–[8] contributes to the *SUM score* only.

AVG. In contrast to the previous scoring functions, the *AVG score* is not defined in terms of the minimum cardinality among some node sets of the pattern, but in terms of the relative weights of its appearances. In general, the score of a pattern P can be a function of the sum of the weights of the subgraphs in its support set, and this is called the *weighted support* (*WSUP*) of P. In particular, WIGM [43] proposes a measure called *normalized weighted support* (*NWSUP*), which is the weighted support of P divided by its size $|E_P|$, i.e., *NWSUP*(G, P) = *WSUP*$(G, P)/|E_P|$. Nevertheless, this scoring function is not MNI-compatible. In order to guarantee the apriori property and be consistent with the other MNI-compatible scoring functions, we compute *WSUP*(G, P) by first retaining, for each edge set $\mathcal{E}(\mathcal{G}, e_P)$ with $e_P \in E_P$, the set $\mathcal{E}(\mathcal{G}, e_P) \upharpoonright_\mu$ of μ edges with largest weight, and then summing up all those weights, i.e., *WSUP*$(G, P) = \sum_{e_P \in E_P} \sum_{e \in \mathcal{E}(\mathcal{G}, e_P) \upharpoonright_\mu} \omega(e)$. Setting μ to be the MNI support of P we guarantee that the *AVG score* is bounded by the MNI support, as stated in the following theorem:

Theorem 5.1. *Given a graph* G:$\langle V, E, \ell, \omega \rangle$, *a pattern* P, *and an edge* $e \in E$, *it holds that* $f_{AVG}(P \diamond e, G) \leq$ *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 has the apriori property [6], *MNI*$(P \diamond e, G) \leq$ *MNI*(P, G). By definition, the pattern $P \diamond e$ has the maximum normalized weight $f^*_{AVG}(P \diamond e, G)$ when all the edges in $\mathcal{E}(\mathcal{G}, e) \upharpoonright_\mu$ have weight 1, and hence each subgraph contributes with a total weight of $(|E_P| + 1)$. In this case, $f^*_{AVG}(P \diamond e, G) =$ *MNI*$(P \diamond e, G) \cdot (|E_P| + 1)/(|E_P| + 1)$, and thus $f_{AVG}(P \diamond e, G) \leq f^*_{AVG}(P \diamond e, G) =$ *MNI*$(P \diamond e, G) \leq$ *MNI*(P, G). □

According to this theorem, although *AVG* does not have the apriori property, the *AVG score* of a pattern is at least bounded by the frequency of its sub-patterns, making it MNI-compatible and allowing early pruning during the pattern search. In fact, if the MNI support of P is lower than τ, then all its super-patterns can be discarded. On the other hand, $f_{AVG}(P \diamond e, G)$ can be higher than $f_{AVG}(P, G)$ even though the frequency of $P \diamond e$ is lower, because the weights of the edges in $\mathcal{E}(\mathcal{G}, e) \upharpoonright_\mu$ can be so large that they compensate for the lower frequency. For example, the *AVG score* of P : $[v_1]$–C–$[v_2]$ in the graph G in Figure 2 is 0.6, because $MNI(P, G) = 1$ and $\mathcal{E}(\mathcal{G}, C) \upharpoonright_1 = \{(1, 4)\}$. Instead, the *AVG score* of P : $[v_1]$–C–$[v_2]$–B–$[v_3]$–A–$[v_4]$ is 0.8, because $\mathcal{E}(\mathcal{G}, C) \upharpoonright_1 = \{(1, 4)\}$, $\mathcal{E}(\mathcal{G}, B) \upharpoonright_1 = \{(1, 3)\}$, and $\mathcal{E}(\mathcal{G}, A) \upharpoonright_1 = \{(3, 5)\}$.

Implementation. To implement *ALL*, *ANY*, and *SUM* in our framework, function isValid checks every match g of P in its support set, by comparing its edge weights against the relevance threshold α, according to the corresponding definition of $S'_G(P)$. Then, Procedure relevanceScore computes the MNI support over the support set $S'_G(P)$. On the other hand, for 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 $k = min_{v \in V_g} |D_v|$.

6 RELATED WORK

We survey the main solutions for pattern mining in *graph databases*, *single graphs*, and *probabilistic graphs*. While previous work has tackled the problem of pattern mining in weighted graphs to a certain extent, no solution has been proposed for pattern mining in multi-weighted graphs.

Graph databases. Graph databases are collections of graphs such as chemical compounds, transactions, and workflows. Two main approaches have been proposed for pattern mining in *unweighted* collections of graphs: apriori-based methods, and pattern-growth methods. The *apriori-based* approaches generate frequent structures incrementally, by merging smaller frequent patterns [24]. Pattern-growth methods, on the other hand, generate one structure at a time, expanding each pattern in a depth-first fashion [17, 41].

In *weighted* graphs, a few pattern-growth methods have been recently introduced [19] to embody weights into the support measure. Additionally, WFSM-MR [3] further extends such approaches in a distributed manner on top of the MapReduce framework.

Nevertheless, frequent pattern mining in graph databases employs a support measure, i.e., the number of graphs containing a specific pattern, that cannot be used to mine patterns in large graphs, as each pattern would have a support equal to 1 or 0.

Single Large Graphs. Pattern mining in large graphs requires the support measure to be adjusted to account for edges shared by multiple subgraphs [6]. To this end, alternative support measures satisfying the apriori property have been proposed, alongside efficient algorithms using such measures. SUBDUE [15] is the first pattern mining algorithm in single graphs and adopts an approximate greedy strategy based on the Minimum Description Length (MDL). Other support measures include the maximum number of edge-disjoint matchings [38], the Maximum Independent Set (MIS) support [25], and the Harmful Overlap (HO) [11] support.

Nonetheless, the latter two measures require **NP**-complete problems to be solved, rendering them unsuitable in many practical scenarios. In contrast, the Minimum Image-based (MNI) support can be computed efficiently [11]. This measure is used by GraMi [10] and its parallel extension ScaleMine[1], which optimize the computation of the frequent patterns via a constraint satisfaction problem approach. Yet, as opposed to the problem we tackle in this work, GraMi and all the support-based approaches disregard weights on the edges of the graph and do not generalize to the case of multi-weights.

The first work on *weighted* large graphs is WTMaxMiner [12]. However, WTMaxMiner restricts the problem to mining *path* patterns, which can be efficiently discovered as opposed to subgraphs. To the best of our knowledge, WIGM [43] is the only work that deals with weighted pattern mining in large graphs, defining the importance of a pattern as the average weight over its appearances. Although weighted patterns do not naturally possess the apriori property, WIGM adopts a weaker pruning strategy based on the so-called *1-extension property*. Differently from WIGM, our solution, ReSuM is scalable and efficient since it uses measures (a.k.a. scoring functions) that satisfy the apriori property and are based on the MNI support. Additionally, ReSuM is a more general framework that supports multi-weighted graphs, as well as a broad family of scoring functions, showcasing the WIGM support measure as one example (see Section 5).

Uncertain graphs. Uncertain graphs include existence probabilities for edges or nodes of the graph. To some extent, uncertain graphs can be seen as a special case of weighted graphs in which probabilities arises, for instance, from random walk approaches, and represent the likelihood that an edge exists between two nodes. Few works have been proposed to mine frequent patterns in uncertain graphs [7, 18, 26, 32, 40, 44]. As opposed to weighted graphs, support measures for uncertain graphs must consider the uncertainty in the edges and compute the support as an expected value. Moreover, the time complexity of mining in such graphs is exponential in the worst case, since any edge can either exists or not, and hence all the possible combinations must be considered.

7 EXPERIMENTS

We experimentally show how the patterns found with ReSuM differ from those returned by frequency-based methods, thus proving the importance of our approach in pruning irrelevant patterns that are merely frequent. We also 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.

Datasets. The experiments were performed on four real datasets of different sizes. Table 1 shows their characteristics, reporting the number of vertices $|V|$, edges $|E|$, and labels $|\mathcal{L}|$; the minimum, average, and maximum node degree; and the minimum, median, average, and maximum edge label frequency. For the Amazon dataset we report statistics for both edge (top) and node labels (bottom). We also report the default frequency (τ) and relevance (α) values used in the experiments (unless otherwise stated).

• CiteSeer [10], 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).

				degree	*label frequency*		
dataset	$\lvert\mathcal{V}\rvert$	$\lvert E\rvert$	$\lvert\Sigma\rvert$	min/avg/max	min/med/avg/max	τ	α
CiteSeer	2.1k	3.6k	21	1/3.5/99	15/55/174.7/988	95	.05
Freebase-T	7.2k	10k	40	1/2.8/504	3/70/251.3/2886	90	.05
Freebase-C	16.7k	26k	77	1/3.2/1082	1/66/348.5/4861	155	.05
Amazon	163k	296k	4 1710	1/3.6/1072	2k/12k/30k/113k 1/1/95/142k	130	.05

Table 1: Datasets and default τ, α parameters.

• 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 taking the largest weakly connected component in the restriction.

• Amazon[2] [14] 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 [2]. 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.

Experimental setup. ReSuM is implemented in Java 1.8 on top of the constraint satisfaction problem presented in GraMi [10] 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 experiments were run on a 24 Cores (2.40GHz) Intel Xeon $E5-2440$ with 188Gb RAM with Linux 3.13.

Generating 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

[1] developers.google.com/Freebase/data
[2] jmcauley.ucsd.edu/data/amazon/
[3] github.com/ehab-abdelhamid/GraMi
[4] https://github.com/lady-bluecopper/ReSuM
[5] www.crowdflower.com

	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 2: Quality of Freq vs ReSuM on the top-k patterns.

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 $\{0.5, 0.8\}$ for the normal distribution and in $\{0.25, 0.5, 0.75, 1\}$ for the *Beta* distribution.

7.1 Frequent vs Weighted 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 relevant pattern mining. Unless otherwise stated, we report the average of 10 different randomly sampled weighting functions. In particular, these weights were sampled from a normal distribution using focus 0.5, as previously described.

Figure 5 reports the average number of patterns found using different scoring functions on the four datasets, with default parameters, as shown in Table 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 biological or chemical datasets, where weights are uniformly distributed in the entire graph.

We now discuss quality (Table 2), number of patterns, and running time of ReSuM compared to Freq, when varying relevance (α) and frequency (τ) threshold (Figure 3 and 4). Due to space limits, 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.

Quality of Freq vs ReSuM. Table 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 relevant patterns 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 5.

Relevance threshold (α). Recall that the relevance threshold α 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(a) and Figure 3(b)), because the larger the value of α, the smaller is the number of appearances that are considered valid, and thus the smaller is the total number of relevant patterns mined. We observe that with α>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(b)). Finally, since *AVG* does not depend on α, it always returns the same patterns.

Figure 3(c) and Figure 3(d) show that the threshold α 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 α. 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 α.

Frequency threshold (τ). Figure 4 reports the behavior of ReSuM and Freq when varying the frequency threshold τ. We performed preliminary tests to decide a reasonable range of values $[\tau_{min}, \tau_{max}]$ for each dataset. In particular, the τ_{min} corresponds to the smallest value that allowed Freq to terminate the computation within 48 hours, and τ_{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 [10] and reflects the observation that pattern frequency is dataset-dependent, while relevance is user-dependent.

As we can see in Figure 4(a) and Figure 4(b), the number of frequent patterns decreases almost linearly with τ, and consequently the number of relevant patterns decreases as well. Regarding the performance, as opposed to the relevance 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 4(c) and Figure 4(d) show that when τ 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.

7.2 Multiple Weighting Functions

We tested the scalability of ReSuM in the case of multiple weighting functions, varying their number between 50 and 50.000. Similarly, we also measured time and quality of ReSuM *approximate*. Nevertheless, 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 edge weight case.

Time. Figure 6 shows how the number of weighting functions affects the running time. Here we report the performance obtained when the weights were generated following a normal distribution with focus 0.5. In Figure 6(a) we present the comparison between ReSuM and the brute-force (BF) approach, which computes the patterns for each weighting function separately. While BF scales

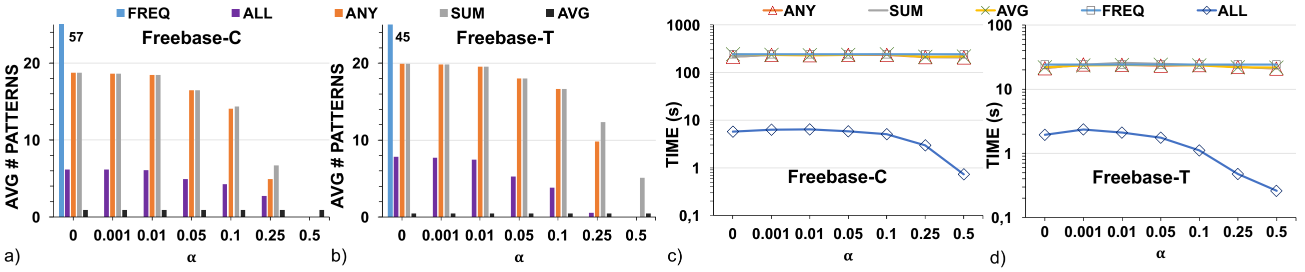

Figure 3: Varying α: number of patterns (left) and running time (right) in Freebase-C (a,c) and Freebase-T (b,d).

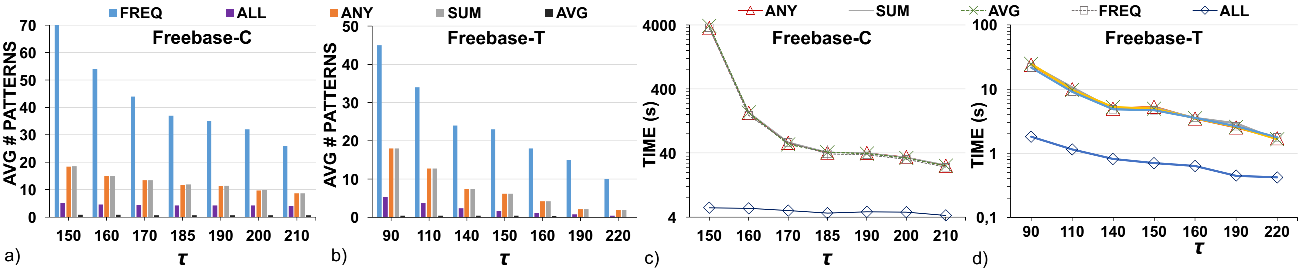

Figure 4: Varying τ: number of patterns (left) and running time (right) in Freebase-C (a,c) and Freebase-T (b,d).

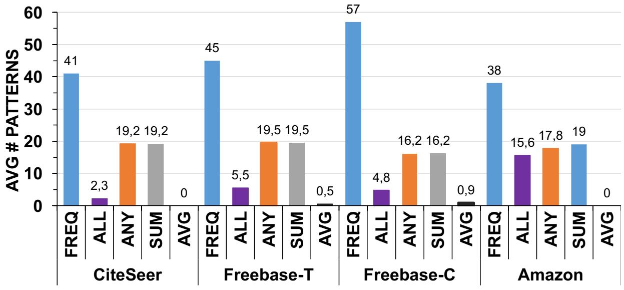

Figure 5: Number of patterns found in each dataset, using different *scores* and default parameters.

	average pattern edit distance			
$\|W\|$	*ALL*	*ANY*	*SUM*	*AVG*
50	0.195	0.069	0.069	0.627
500	0.192	0.062	0.062	0.618
5000	0.203	0.053	0.053	0.609
50000	0.204	0.052	0.051	–

Table 3: Quality of ReSuM *approximate* in Freebase-T.

	average pattern edit distance							
	Freebase-C				Freebase-T			
clustering	*ALL*	*ANY*	*SUM*	*AVG*	*ALL*	*ANY*	*SUM*	*AVG*
A-Post	0.28	0.07	0.07	0.45	0.2	0.07	0.07	0.7
Buck	0.27	0.07	0.07	0.39	0.2	0.06	0.06	0.62

Table 4: Quality of ReSuM *approximate* using Buck and A-Post clustering in Freebase-T and Freebase-C.

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 drawback, the memory requirement grows linearly with the number of weights for both algorithms.

In Figure 6(b) and Figure 6(c) 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 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 *ANY* 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.

Quality of ReSuM *approximate*. As mentioned in Section 4, 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). We define the distance between two patterns as the minimum number of edges that should be added or removed from the first to transform it into the second. Thus, the average distance between the two sets of patterns $\{A_1, \ldots, A_m\}$ and $\{R_1, \ldots, R_m\}$ measures the average number of operations required to transform a pattern in A_i to a pattern 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

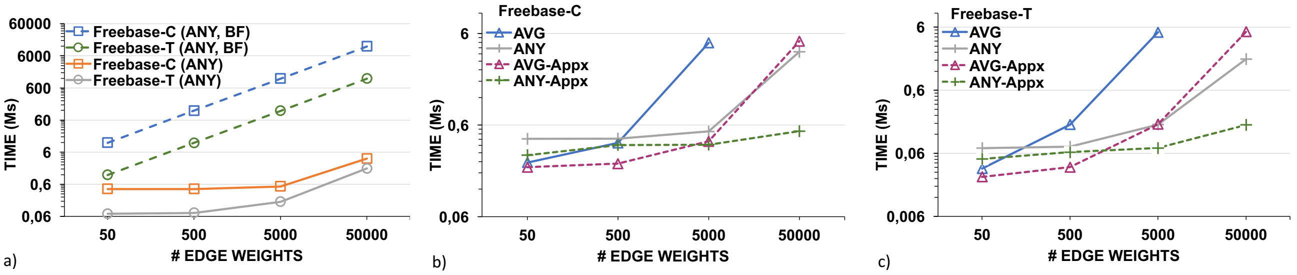

Figure 6: Varying number of edge weights in Freebase-C and Freebase-T: running time of ReSuM and brute-force approach (BF) with *ANY* (a); and running time of ReSuM and ReSuM *approximate* with *ANY* and *AVG* (b, c).

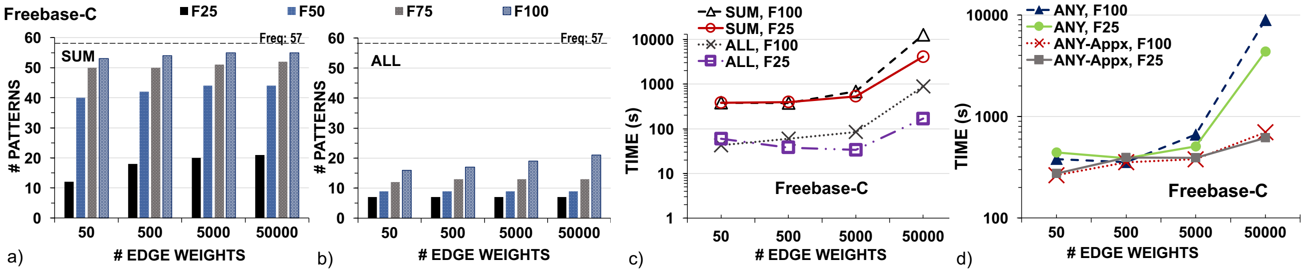

Figure 7: Varying *focus* in Freebase-C: number of patterns using *SUM* (a) and *ALL* (b) with focus between 0.25 (F25) and 1 (F100); and running time of ReSuM and ReSuM *approximate* with $Beta(0.7, 5)$ weights with focus 0.25 (F25) and 1 (F100), using *SUM*, *ALL* (c), and *ANY* (d).

average precision

	$Beta(0.7, 5)$								$Beta(5, 0.7)$								$\mathcal{N}(0.5, 0.25)$			
	ALL				*SUM*				*ALL*				*SUM*				*ALL*		*SUM*	
$\lvert W \rvert$	0.25	0.5	0.75	1	0.25	0.5	0.75	1	0.25	0.5	0.75	1	0.25	0.5	0.75	1	0.5	0.8	0.5	0.8
50	0.22	0.20	0.21	0.26	0.17	0.53	0.74	0.91	0.19	0.23	0.42	1	0.34	0.73	0.94	1	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	0.21	0.24	0.42	1	0.36	0.73	0.94	1	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	0.21	0.25	0.43	1	0.36	0.74	0.95	1	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	0.21	0.25	0.44	0.99	0.36	0.74	0.95	1	0.22	0.21	0.51	0.59

Table 5: 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.

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. Table 3 reports the distances obtained using the four scoring functions in Freebase-T. Here, *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, 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 $R_1, \ldots, R_m$ as feature vectors of $\omega_1, \ldots, \omega_m$, 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 4 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.

Impact of the weights. For the experiments presented above, we weighted the Amazon graph using real weights, and the Freebase-T, Freebase-C, and CiteSeer graphs with 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 Section7.

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 7(a) and Figure 7(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 5 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.

Regarding the running time, Figure 7(c) and Figure 7(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 5 displays the quality of ReSuM in terms of average precision, as defined in Equation 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.

8 CONCLUSIONS

In this paper we considered the problem of mining relevant patterns in weighted graphs. As opposed to the previous graph pattern mining approaches, which are solely based on the frequency of the patterns, our solution assesses the importance of a pattern also in terms of the weights on the edges of its appearances. Then, we proposed four different scoring functions that balance between frequency and weights, while retaining the apriori property, which is a powerful mean to an effective and early pruning of the search space. As a natural extension, we considered the complementary problem of mining patterns in graphs with multiple weights associated to the edges. We devised exact and approximate solutions and proved the effectiveness and efficiency of the algorithms on real datasets. As a future work, we plan to study the theoretical bounds on the clustering quality, and automatic approaches for parameter selection.

REFERENCES

[1] Ehab Abdelhamid, Ibrahim Abdelaziz, Panos Kalnis, Zuhair Khayyat, and Fuad Jamour. 2016. ScaleMine: Scalable parallel frequent subgraph mining in a single large graph. In *International Conference for High Performance Computing, Networking, Storage and Analysis*. 716–727.

[2] Charu C Aggarwal. 2016. *Recommender Systems*. Springer.

[3] Nisha Babu and Ansamma John. 2016. A distributed approach to weighted frequent Subgraph mining. In *International Conference on Emerging Technological Trends*. 1–7.

[4] Dorna Bandari, Shuo Xiang, and Jure Leskovec. 2017. Categorizing User Sessions at Pinterest. *arXiv preprint arXiv:1703.09662* (2017).

[5] Petko Bogdanov, Misael Mongiovì, and Ambuj K Singh. 2011. Mining heavy subgraphs in time-evolving networks. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*. IEEE, 81–90.

[6] Bjorn Bringmann and Siegfried Nijssen. 2008. What is frequent in a single graph?. In *PAKDD*. 858–863.

[7] Yifan Chen, Xiang Zhao, Xuemin Lin, and Yang Wang. 2015. Towards frequent subgraph mining on single large uncertain graphs. In *2015 IEEE International Conference on Data Mining*. 41–50.

[8] James C Costello, Mehmet M Dalkilic, Scott M Beason, Jeff R Gehlhausen, Rupali Patwardhan, Sumit Middha, Brian D Eads, and Justen R Andrews. 2009. Gene networks in Drosophila melanogaster: integrating experimental data to predict gene function. *Genome biology* 10, 9 (2009), R97.

[9] L. De Raedt and A. Zimmermann. 2007. Constraint-Based Pattern Set Mining. In *SDM*. 237–248.

[10] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. 2014. Grami: Frequent subgraph and pattern mining in a single large graph. *PVLDB* 7, 7 (2014), 517–528.

[11] M. Fiedler and C. Borgelt. 2007. Subgraph support in a single large graph. In *ICDM Workshops*. 399–404.

[12] R. Geng, X. Dong, P. Zhang, and W. Xu. 2008. WTMaxMiner: Efficient Mining of Maximal Frequent Patterns Based on Weighted Directed Graph Traversals. In *CCIS*. 1081–1086.

[13] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. In *ACM SIGMOD Record*, Vol. 30. 58–66.

[14] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*. 507–517.

[15] Lawrence B Holder, Diane J Cook, Surnjani Djoko, and others. 1994. Substucture Discovery in the SUBDUE System.. In *KDD Workshop*. 169–180.

[16] J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. 2005. Comparing Graph Representations of Protein Structure for Mining Family-specific Residue-based Packing Motifs. *J. Comput Biol.* 12, 6 (2005), 657–671.

[17] J. Huan, W. Wang, J. Prins, and J. Yang. 2004. Spin: mining maximal frequent subgraphs from graph databases. In *SIGKDD*. 581–586.

[18] Shawana Jamil, Azam Khan, Zahid Halim, and A Rauf Baig. 2011. Weighted muse for frequent sub-graph pattern finding in uncertain dblp data. In *2011 International Conference on Internet Technology and Applications*. 1–6.

[19] C. Jiang, F. Coenen, and M. Zito. 2010. Frequent sub-graph mining on edge weighted graphs. In *DAWAK*. 77–88.

[20] H. Jiang, H. Wang, P. S. Yu, and S. Zhou. 2007. GString: A Novel Approach for Efficient Search in Graph Databases. In *ICDE*. 566–575.

[21] W. Jiang, J. Vaidya, Z. Balaporia, C. Clifton, and B. Banich. 2005. Knowledge discovery from transportation network data. In *ICDE*. 1061–1072.

[22] Xin Jin, Chi Wang, Jiebo Luo, Xiao Yu, and Jiawei Han. 2011. LikeMiner: a system for mining the power of'like'in social media networks. In *KDD*. 753–756.

[23] Minoru Kanehisa and Susumu Goto. 2000. KEGG: kyoto encyclopedia of genes and genomes. *Nucleic acids research* 28, 1 (2000), 27–30.

[24] M. Kuramochi and G. Karypis. 2001. Frequent subgraph discovery. In *ICDM*. 313–320.

[25] M. Kuramochi and G. Karypis. 2005. Finding frequent patterns in a large sparse graph. *DMKD* 11, 3 (2005), 243–271.

[26] Jianzhong Li, Zhaonian Zou, and Hong Gao. 2012. Mining frequent subgraphs over uncertain graph databases under probabilistic semantics. *VLDBJ* 21, 6 (2012), 753–777.

[27] S. P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–137.

[28] A. K. Mackworth. 1977. Consistency in networks of relations. *Artificial Intelligence* 8, 1 (1977), 99–118.

[29] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. 2016. Exemplar queries: a new way of searching. *VLDB J.* (2016), 1–25.

[30] Mark EJ Newman. 2004. Analysis of weighted networks. *Physical review E* 70, 5 (2004).

[31] C. C. Noble and D. J. Cook. 2003. Graph-based Anomaly Detection. In *SIGKDD*. 631–636.

[32] Odysseas Papapetrou, Ekaterini Ioannou, and Dimitrios Skoutas. 2011. Efficient discovery of frequent subgraph patterns in uncertain graph databases. In *Proceedings of the 14th International Conference on Extending Database Technology*. 355–366.

[33] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. 2000. Mining Access Patterns Efficiently from Web Logs. In *PAKDD*. 396–407.

[34] Michael J Shaw, Chandrasekar Subramaniam, Gek Woo Tan, and Michael E Welge. 2001. Knowledge management and data mining for marketing. *Decision support systems* 31 (2001), 127–137.

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

[36] Q. Song, Y. Wu, and X. L. Dong. 2016. Mining Summaries for Knowledge Graph Search. In *ICDM*. 1215–1220.

[37] Michael Steinbach, Levent Ertöz, and Vipin Kumar. 2004. The challenges of clustering high dimensional data. In *New Directions in Statistical Physics*. 273–309.

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

[39] Haixun Wang and Charu C Aggarwal. 2010. A survey of algorithms for keyword search on graph data. In *Managing and Mining Graph Data*. 249–273.

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

[41] X. Yan and J. Han. 2002. gspan: Graph-based substructure pattern mining. In *ICDM*. 721–724.

[42] X. Yan, P. S. Yu, and J. Han. 2004. Graph Indexing: A Frequent Structure-based Approach. In *SIGMOD*. 335–346.

[43] J. Yang, W. Su, S. Li, and M. M. Dalkilic. 2012. WIGM: Discovery of Subgraph Patterns in a Large Weighted Graph. In *SDM*. 1083–1094.

[44] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. 2010. Mining frequent subgraph patterns from uncertain graph data. *IEEE Transactions on Knowledge and Data Engineering* 22, 9 (2010), 1203–1218.