Mining Frequent Patterns with Counting Quantifiers

Author list

No Institute Given

Abstract. Frequent Pattern Mining (FPM) is a classical graph mining task. It is typically defined as discovering all the graph patterns Q whose occurrence frequencies are above a user-specified threshold τ, in a single large graph G. In recent years, we have witnessed wide applications of FPM, such as social network analysis, fraud detection, etc. However, emerging applications keep calling for more expressive graph patterns to capture more complex structures in a large graph. In light of this, we investigate the problem of mining frequent patterns with counting quantifiers. We first introduce quantified graph patterns (QGPs), which are patterns incorporated with counting quantifiers and able to express richer semantics. We then develop an algorithm, denoted as QGPM, to mine QGPs in a single large graph, and provide effective optimization techniques to reduce space and computational costs. We also introduce quantified graph pattern association rules (QGPARs) and propose techniques to identify links with QGPARs. Using real graphs and synthetic graphs, we verified that (1) our QGPM algorithm is very efficient to discover QGPs and our optimization strategies are pretty effective in reducing space and computational costs; (2) compared with existing link prediction methods, QGPARs mined achieve even higher prediction accuracy.

1 Introduction

Graphs simulate complex connections between objects in a variety of applications such as chemistry, bioinformatics, social networks, and text retrieval, where entities are represented as nodes, and interactions between entities are represented as edges. The problem of FPM has been a core task in the field of graph mining, due to its wide applications. Driven by applications, existing work considers the problem in two different settings: transaction-based and single-graph-based. With the popularity of large scale networks, FPM under single-graph-based setting has received more attention [12]. Nevertheless, facing more complicated scenarios, more expressive patterns are required, especially those with counting quantifiers (CQs), which are represented as ordinary mathematical symbols in first-order logic. The introduction of CQs can not only extend the expressive ability of semantics but also capture complex structures in the underlying graphs [13].

Example 1. Graph patterns with CQs reflect the regularity of connections between entities in social networks. Patterns Q_1 and Q_2 of Fig. 1 show that:

∘ *if* (i) *person* X_0 *is in a music club, and* (ii) *among the people whom* X_0 *follows, at least* m *of them like Shakira's album, then* X_0 *may also like the album.*

∘ *if* (i) m *people* Z *whom* X_0 *follows,* (ii) Z *recommends Lenovo computer, then* X_0 *may buy a Lenovo computer.*

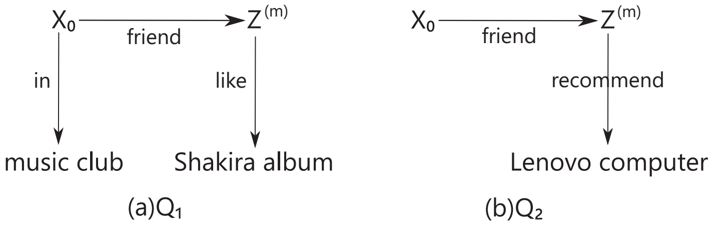


Fig. 1. Quantified graph patterns

Intuitively, the above graph patterns take quantifiers m, and achieve quantitative aggregation through m. Compared with the traditional graph pattern, the graph pattern with CQs has even richer semantics. Indeed, formulating a marketing strategy based on the above patterns can achieve better performance than traditional marketing methods. Since empirical studies suggest that "90% of customers believe in peer recommendations, while only 14% of customers trust advertisements [1]", and "the peer influence from one's friends leads to more than 50% increase in the probability of buying a product [6]". Despite the wide application, we are not aware any prior work for discovering such patterns.

To this end, this paper developed techniques to mine and use quantified graph patterns. Our contributions are as follows.

Contributions. Our contributions in this paper can be summarized as follows.

(1) We extend the traditional graph pattern and propose QGPs (Section 2) for capturing more complex structures.

(2) We propose QGPM, a framework for mining frequent QGPs in large single graphs. Based on QGPM, corresponding optimization strategies are proposed to improve the performance of the algorithm.

(3) As an application of QGP, we introduce Quantified Graph Pattern Association Rules (QGPARs; Section 4). QGPARs are able to predict relationship among entities in large networks.

(4) Using real-life and synthetic graphs, we experimentally verify the performance of our algorithms (Section 5) and find the following. (a) It is feasible to mine QGPs on large graphs. (b) QGPARs can predict missing relationships with an average accuracy of 84.3% on real-life graphs, and it can achieve higher prediction accuracy compared with other link prediction methods.

Related Work. We categorize related work as follows.

Graph pattern mining. Frequent pattern mining is a core problem in many graph analysis applications [27, 21]. In recent years, many frequent pattern mining algorithms for large-scale data sets have emerged. [11] presented GraMi, a novel framework for frequent subgraph mining in a single large graph. GraMi is based on a novel approach, which is not storing all the appearances of a subgraph in a large graph, but only stores the subgraph's templates. [18] proposed FSSG, an algorithm that uses graph invariant properties and symmetries present in a graph to generate candidate subgraphs. FSSG reduces the generation of a large number of candidate subgraphs, thereby reducing the complexity of candidate generation and frequency counting. [20] introduced the SOCMI algorithm, its core idea is to store the appearance of patterns with pathgraph, which makes it easier for SOCMI to extend patterns and calculate frequency during mining. In addition, some mining methods such as weighted graph and dynamic graph have also been widely studied. [5, 19] proposed to mine weighted frequent subgraphs in a weighted single large graph. StreamFSM [26] and IncGM+ [3] are mining algorithms based on dynamic graphs.

Pattern semantic extension. In practice, traditional graph patterns are not powerful enough for modeling complex problems, and the necessity of semantic extension of graph patterns is gradually revealed. SPARQLog extends SPARQL with first-order logic (FO) rules including existential and universal quantification of nodes in the graph [9]. For social networks, SocialScope [4] and SNQL [24] are algebraic languages with numerical aggregations on node and edge sets. Furthermore, social recommendation rules are studied in [22], and support counts are introduced as constraints.

Prior work show that existing extensions still suffer from limitations and in emerging applications, patterns with complex features, notably counting quantifiers (CQs), predicates and negation are required [23]. Among these features, CQs have received more attention [7, 14, 12, 23]. The QGraph [7] allows annotating edges of a graph pattern with CQs of the form [min, max] that can express different semantics (*e.g.,* negated ([0, 0]) and optional ([0, 1]) edges). From this, users can formulate query conditions consisting of nodes and edges. In [14], a quantified graph pattern has been proposed where CQs can express numeric and ratio aggregates of the forms: "=p(%)", "≥p(%)", and "=0" for negation. [14] also introduced a quantified matching model that can be used to identify potential customers in social media marketing. In [12], CQs are expressed in the following forms: "[min, max]", "≥min" and "≤max", which are used to annotate edges to extend the traditional graph pattern. [23] proposed the Conditional Graph Pattern (CGP), which extended the traditional graph pattern with CQs.

2 Preliminaries

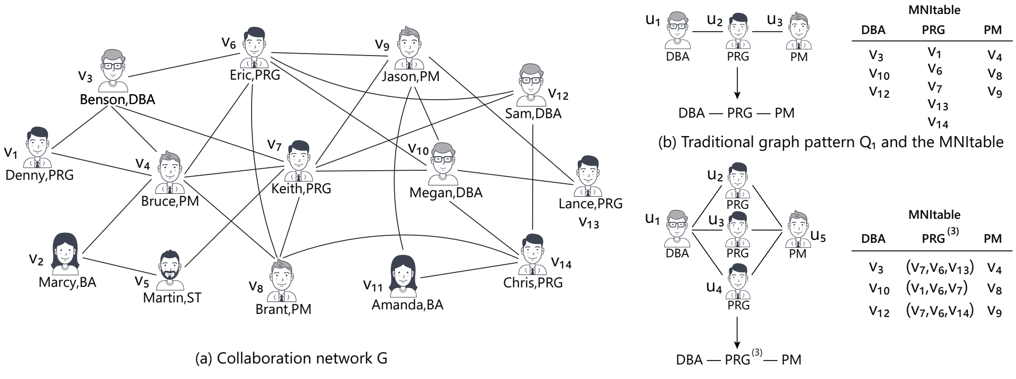
In this section, we first review traditional graph patterns; we then introduce related concepts of quantified graph patterns(QGPs).

2.1 Conventional Graph Pattern Mining

Graph. A *graph* is defined as $G = (V, E, L)$, where (1) V is a set of nodes; (2) $E \subseteq V \times V$ is a set of edges; and (3) each node v in V carries a tuple $L(v) = (A_1 = a_1, \cdots, A_n = a_n)$, where $A_i = a_i (i \in [1, n])$ represents that the node v has a value a_i for the attribute A_i, and is denoted as $v.A_i = a_i$, *e.g.,* v.name ="*Sam*", v.job_title ="*DBA*".

A graph $G' = (V', E', L')$ is a *subgraph* of $G = (V, E, L)$, denoted by $G' \subseteq G$, if $V' \subseteq V$, $E' \subseteq E$, and moreover, for each $v \in V'$, $L'(v) = L(v)$.

Pattern. A *pattern* Q is defined as a graph (V_p, E_p, L_v), where V_p and E_p are the set of nodes and edges, respectively; for each u in V_p, it is associated with a predicate $L_v(u)$ defined as a conjunction of atomic formulas of the form of '$A = a$' such that A



(a) Collaboration network G

(b) Traditional graph pattern Q_1 and the MNItable

(c) Quantified graph pattern Q_2 and the MNItable

Fig. 2. Graph & Traditional graph pattern & Quantified graph pattern(QGP)

denotes an attribute of the node u and a is a value of A. Intuitively, $L_v(u)$ specifies search conditions imposed by u.

Graph pattern matching. Consider graph $G = (V, E, L)$ and pattern $Q = (V_p, E_p, L_v)$, a node v in G satisfies the search conditions of a pattern node u in Q, denoted as $v \sim u$, if for each atomic formula '$A = a$' in $L_v(u)$, there exists an attribute A in $L(v)$ such that $v.A = a$.

Further, a match of pattern Q in graph G is a *bijective function* f from the nodes of Q to the nodes of a subgraph G, such that (1) for each node $u \in V_p$, $f(u) \sim u$, and (2) (u, u') is an edge in Q if and only if $(f(u), f(u'))$ is an edge in G. We denote by $Q(G)$ the set of matches of Q in G.

Example 2. Consider Q_1 (Fig. 2(b)) and G (Fig. 2(a)), $Q_1(G) = \{\{v_3, v_1, v_4\}, \{v_3, v_7, v_4\}, \{v_3, v_7, v_8\}, \{v_3, v_7, v_9\}, \{v_3, v_6, v_9\}, \{v_3, v_6, v_8\}, \{v_3, v_6, v_4\}, \{v_{10}, v_7, v_9\}, \{v_{10}, v_7, v_8\}, \{v_{10}, v_7, v_4\}, \{v_{10}, v_6, v_9\}, \{v_{10}, v_6, v_8\}, \{v_{10}, v_6, v_4\}, \{v_{10}, v_{13}, v_9\}, \{v_{10}, v_{14}, v_8\}, \{v_{12}, v_6, v_9\}, \{v_{12}, v_6, v_8\}, \{v_{12}, v_6, v_4\}, \{v_{12}, v_7, v_4\}, \{v_{12}, v_7, v_8\}, \{v_{12}, v_7, v_9\}, \{v_{12}, v_{14}, v_8\}\}$.

Support. The support of a pattern Q in a graph G, denoted by $sup(G, Q)$, indicates the occurrence frequency of Q in G.

Having an anti-monotone support metric is of crucial importance since it allows the development of methods to effectively prune the search space; several anti-monotone support metrics exist such as minimum image based (MNI) [8], harmful overlap (HO) [15], and maximum independent sets (MIS) [16]. In this paper, we adopt the MNI metric because it can be calculated more efficiently.

Let f be the set of isomorphic mappings of pattern $Q = (V_p, E_p, L_v)$ in G. The length of the set f is m, while $F(u) = \{f_1(u), f_2(u), \cdots, f_m(u)\}$ is the mapping function for each u on the set f after de-duplication, where $u \in V_p$. The minimum image based support (MNI) of Q in G, denoted by $sup(G, Q)$, is defined as $sup(G, Q) = \min\{t | t = |F(u)|, u \in V_p\}$.

Intuitively, MNI computes the support of a pattern Q as the minimum number of distinct graph vertices matching each $u_i \in V_p$. We use MNItable to represent the MNI field of pattern Q. An MNItable consists of a set of MNIcol; the MNI metric returns the length of the smallest MNIcol. An MNIcol(u_i) contains a list of distinct valid nodes, *i.e.,* nodes corresponding to pattern node u_i in the match list for Q in G.

Example 3. Recall Example 2, it can be easily verified that MNIcol(u_1) = $\{v_3, v_{10}, v_{12}\}$, MNIcol(u_2) = $\{v_1, v_6, v_7, v_{13}, v_{14}\}$, MNIcol($u_3$) = $\{v_4, v_8, v_9\}$, thus $sup(G, Q_1)$ = 3. Fig. 2(b) shows the resulting MNItable.

2.2 Quantified Graph Patterns

Quantified Graph Patterns (QGPs). A QGP $Q^{(m)}_{(u_0)}$ is defined as (V_p, E_p, L_u, f_u), where (1) V_p and E_p are the set of pattern nodes and edges, respectively; (2) L_u is a function that assigns a label to a node u ($u \subseteq V_p$); and (3) f_u assigns a CQ m (a natural number) to the specified quantifier-constrained node u_0 such that $f_u(u_0) = m$.

Intuitively, the quantified graph pattern extends the traditional graph pattern by incorporating a CQ. It supports counting on a single node, *i.e.,* for two or more nodes with the same label, if they all connect to the same set of nodes, they can be shrunk into a single node with a CQ. *e.g.,* in Fig. 2(c), Q_2 is a quantified graph pattern and can be represented as $DBA - PRG^{(3)} - PM$, *i.e., PRG* is associated with a superscript "(3)" as a quantifier.

In particular, we only consider the case where a single node in the pattern has a CQ. For a QGP, when the quantifier constraints are not considered, the corresponding pattern is called a quantifier-free pattern, and is denoted as Q^w. *e.g.,* Q_1 in Fig. 2(b) is the quantifier-free pattern of Q_2 in Fig. 2(c).

Support of QGP. Inspired by MNI support. Since QGP carries CQs constraints, when calculating MNItable, we regard the nodes with CQ as a column, and then calculate their MNI support. *e.g.,* in Fig. 2(c), for Q_2, denote MNIcol($PRG^{(3)}$)=$\{(v_7, v_6, v_{13}), (v_1, v_6, v_7), (v_7, v_6, v_{14})\}$, MNIcol($PM$)=$\{v_4, v_8, v_9\}$, MNIcol($DBA$)=$\{v_3, v_{10}, v_{12}\}$, hence $sup(G, Q_2) = 3$.

Theorem 1. *Given a QGP* ${Q_1}^{(m)}_{(v)}$*, if there is another QGP* ${Q_2}^{(m')}_{(v)}$*, they have the same* Q^w*, but* $m > m'$*, then* ${Q_1}^{(m)}_{(v)}$ *implies* ${Q_2}^{(m')}_{(v)}$*, denoted as* ${Q_1}^{(m)}_{(v)} \Rightarrow {Q_2}^{(m')}_{(v)}$*. For this case, only need to consider the pattern with the largest CQ.*

Example 4. Consider two QGPs, $Q^{(2)}_{(B)} : A - B^{(2)} - C$ and $Q^{(3)}_{(B)} : A - B^{(3)} - C$, we will only focus on the latter, because (1) $Q^{(3)}_{(B)} \Rightarrow Q^{(2)}_{(B)}$. (2) According to the "anti-monotone" property, if $Q^{(3)}_{(B)}$ is frequent, $Q^{(2)}_{(B)}$ must also be frequent.

In view of this, in this paper, for frequent patterns with the same Q^w, we only discuss the case with the largest CQ.

1-itemset. *1-itemset* refers to the structure of $X - Y^{(m)} (m \geq 1)$ in the graph, where X and Y are quantifier-free node and quantifier-constrained node, respectively. It is frequent when the support of a *1-itemset* meets the threshold.

Forward & Backward extension. The forward extension on a pattern Q essentially introduces a new edge from one node in Q; while the backward extension includes a new edge from two existing nodes. Interested readers can refer to [29].

The notations of this paper are summarized in Table 1.

symbols	notations
CQ	counting quantifier
$Q^{(m)}_{(v)}$	QGP, v is the quantifier-constrained node and m is the value of CQ.
Q^w	A pattern for a QGP without considering CQ
$Q_1{}^{(m)}_{(v)} \Rightarrow Q_2{}^{(m')}_{(v)}$	$Q_1{}^{(m)}_{(v)}$ implies $Q_2{}^{(m')}_{(v)}$
QnodeLabel	quantifier-constrained node label
Qnum	value of CQ
SimpleNode	quantifier-free node
InstanceSet.size	number of pattern matches
MNItable.SimplenodeCol(u$_i$)	column corresponding to quantifier-free node u_i in MNItable
MNItable.QuantifiednodeCol	column of quantifier-constrained node in MNItable

Table 1. Notations used in the paper

3 Frequent QGPs mining

In this section, we first investigate the frequent QGP mining problem (Section 3.1); we then develop an algorithm for the problem (Section 3.2) and provide an optimization strategy to improve performance (Section 3.3).

3.1 Problem Statement

The *frequent QGP mining* (QPM) problem can be stated as follows.

- Input: A graph G, support threshold τ.
- Output: A set $\mathbb{S}$ of frequent QGPs Q of G such that $sup(G, Q) \geq \tau$ for any Q in $\mathbb{S}$.

According to Theorem 1, here $\mathbb{S}$ only contains all frequent QGPs with the largest CQ in the same Q^w structure.

Proposition 2: *The decision problem of QPM is NP-hard.*

To see Proposition 2, observe that the subgraph isomorphism (ISO) problem is embedded in QPM problem, thus QPM problem must be at least as hard as ISO problem [10]. Since ISO is an NP-complete problem, thus QPM problem must be NP-hard.

3.2 Frequent QGP mining

Before presenting our mining algorithm, we first introduce a key observations *w.r.t.* QGP. A QGP is constrained via its counting quantifier, under the same support threshold, the number of frequent QGP will be much less than patterns without quantifier. Based on the observation, Propositions 3 and 4 follows.

Proposition 3: *The expansion process adopts the method of quantifiers from high to low, which can ensure that the first frequent pattern found by the algorithm must be the pattern with the largest number of quantifiers in the same Q^w.*

Star-QGP. Star-QGP has the characteristics of star topology. The pattern structure takes a quantifier-constrained node as the center, and other quantifier-free nodes are connected with the center node.

Proposition 4: *The generation of Star-QGP starts with a 1-itemset, and then gradually adds adjacent 1-itemsets.*

Specifically, the Star-QGP is generated by merging nodes with the same quantifier-constrained node labels in both patterns, and then the CQ value of the new pattern takes the smaller CQ of the two patterns.

Remark. Unless otherwise stated, the patterns we mention below are QGP (quantifier-constrained node and CQ are not given).

Algorithm. We next provide an algorithm QGPM for mining QGPs.

Initialization. QGPM first initializes a set of parameters: set *result* for keeping track of frequent QGPs (line 1), sets *fEdge* and *fItemSet* are used to maintain frequent single-edge patterns and frequent *1-itemsets*, respectively(lines 2-3). Note that the *1-itemset* here is the case with the largest CQ in the same Q^w.

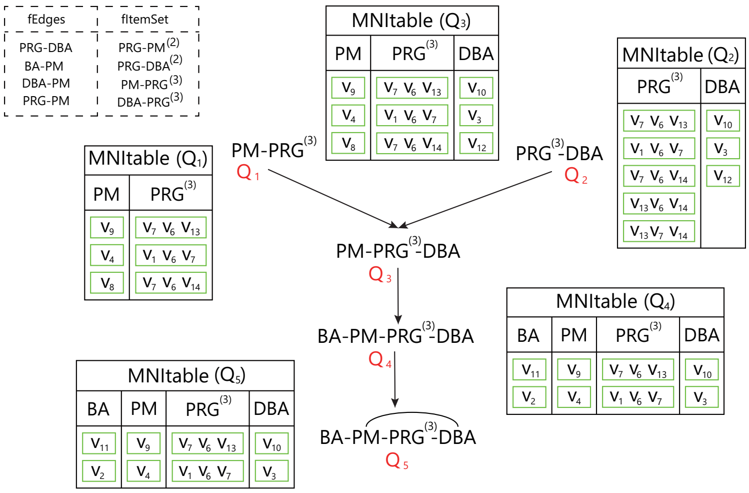
Frequent QGPs mining. Based on *fEdge* and *fItemSet*, QGPM is divided into different stages to obtain results. In the first stage, for each *1-itemset* (from *fItemSet*), QGPM applies the procedure *QuanExt* to generate a set of frequent Star-QGPs (lines 4-6); the second is forward expansion, QGPM calls the procedure *FWExt* to extend the Star-QGP produced in the first stage by adding new edges (from *fEdges*) (lines 7-8); finally, QGPM employs the procedure *BWExt* for backward extension (lines 9-10). In addition, to exclude the already generated extensions, we adopt the VF2-based graph isomorphism detection algorithm. Next, we describe the details of each stage.

Algorithm 1: *QGPM*

```
   Input: graph $G$, support threshold $\tau$.
   Output: All QGPs $Q$ of $G$ such that $\sup(G, Q) \geq \tau$.
 1 $result \leftarrow \emptyset$;
 2 Let $fEdges$ be the set of frequent single edge patterns in $G$;
 3 Let $fItemSet$ be the set of frequent 1-itemset in $G$;
 4 for each $item \in fItemSet$ do
 5     $result \leftarrow result \cup$ QuanExt($item, G, \tau, fItemSet$);
 6     Remove $item$ from $fItemSet$;
 7 for each $fwitem \in result$ do
 8     $result \leftarrow result \cup$ FWExt($fwitem, G, \tau, fEdges$);
 9 for each $bwitem \in result$ do
10     $result \leftarrow result \cup$ BWExt($bwitem, G, \tau, fEdges$);
11 return $result$;
```

(I) Star-QGP generation. QGPM employs procedure *QuanExt* to generate Star-QGPs. More specifically, *QuanExt* takes a QGP S as input and tries to extend it with the *1-itemset* of *fItemSet* (lines 2-5). That is, when S and the current *1-itemset* have the same quantifier-constrained node label, it means that they can be merged to generate a new candidate pattern. All applicable extensions that have not been previously considered are stored in *candidateSet* (line 6). Then, *QuanExt*(lines 7-11) evaluates all candidate patterns generated. If the candidate is frequent, the program continues to extend and evaluates it recursively. Otherwise, decrement the CQ for the current pattern, and check whether the pattern after the decrement is frequent(line 11, details of *CQdec* will be given shortly).

```
Algorithm 2: QuanExt
Input: QGP $S$, graph $G$, support threshold $\tau$, frequent itemset
       $fItemSet$.
Output: frequent Star-QGPs.
$result \leftarrow S$, $candidateSet \leftarrow \emptyset$;
for each $item \in fItemSet$ do
    if $item.QnodeLabel$ equals $S.QnodeLabel$ then
        Let $G'_s$ be the extension of $S$ with $item$;
        if $G'_s$ was not generated then
            $candidateSet \leftarrow candidateSet \cup \{G'_s\}$;
for each $c \in candidateSet$ do
    if ISFrequent$(c, G, \tau)$ then
        $result \leftarrow result \cup$ QuanExt$(c, G, \tau, fItemSet)$;
    else
        $candidateSet \leftarrow candidateSet \cup$ CQdec$(c, G, \tau)$;
return $result$;
```



fEdges	fItemSet
PRG-DBA	PRG-PM$^{(2)}$
BA-PM	PRG-DBA$^{(2)}$
DBA-PM	PM-PRG$^{(3)}$
PRG-PM	DBA-PRG$^{(3)}$

MNItable (Q_1)

PM	PRG$^{(3)}$
V_9	V_7 V_6 V_{13}
V_4	V_1 V_6 V_7
V_8	V_7 V_6 V_{14}

MNItable (Q_3)

PM	PRG$^{(3)}$	DBA
V_9	V_7 V_6 V_{13}	V_{10}
V_4	V_1 V_6 V_7	V_3
V_8	V_7 V_6 V_{14}	V_{12}

MNItable (Q_2)

PRG$^{(3)}$	DBA
V_7 V_6 V_{13}	V_{10}
V_1 V_6 V_7	V_3
V_7 V_6 V_{14}	V_{12}
V_{13} V_6 V_{14}	
V_{13} V_7 V_{14}	

MNItable (Q_4)

BA	PM	PRG$^{(3)}$	DBA
V_{11}	V_9	V_7 V_6 V_{13}	V_{10}
V_2	V_4	V_1 V_6 V_7	V_3

MNItable (Q_5)

BA	PM	PRG$^{(3)}$	DBA
V_{11}	V_9	V_7 V_6 V_{13}	V_{10}
V_2	V_4	V_1 V_6 V_7	V_3

Fig. 3. Example of pattern extension process

Example 5. Recall Fig. 3 with τ=2 and G (Fig. 2(a)). QGPM first initializes the sets *fEdge* and *fItemSet* (shown in the upper left corner of Fig. 3), as their support is greater than 2, and then QGPM runs in stages. For example, considering Q_1 and Q_2(Fig. 3), QGPM applies procedure *QuanExt* to extend them, and generates candidate Star-QGP Q_3 ($sup(G, Q_3) = 3$).

(II) Forward expansion. QGPM applies *FWExt* for forward expansion, which is aimed at the quantifier-free nodes in QGPs. *FWExt* takes QGP S as input, and first obtains the quantifier-free nodes in S (line 2), then extends S by adding new edges (from *fEdges*) to generate candidate patterns (lines 3-6). After that, *FWExt* evaluates the support of candidate patterns and eliminates the members of the candidate set that do not satisfy the support threshold τ. Finally, *FWExt* is recursively executed to further expand the frequent subgraphs (lines 9-10). For the contents of line 12, please refer to *QuanExt*.

Algorithm 3: *FWExt*

```
Input: QGP $S$, graph $G$, support threshold $\tau$, frequent edgeset
       $fEdges$.
Output: frequent QGPs after forward expansion.
1  $result \leftarrow S$, $candidateSet \leftarrow \emptyset$;
2  $SimpleNodeSet \leftarrow SimpleNode$ of $S$;
3  for each $e \in fEdges$ and $u \in SimpleNodeSet$ do
4      if $e$ can be used to extend $u$ then
5          Let $G'_s$ be the extension of $S$ with $e$;
6          if $G'_s$ was not generated then
7              $candidateSet \leftarrow candidateSet \cup \{G'_s\}$;
8  for each $c \in candidateSet$ do
9      if ISFrequent$(c, G, \tau)$ then
10         $result \leftarrow result \cup \textbf{FWExt}(c, G, \tau, fEdges)$;
11     else
12         $candidateSet \leftarrow candidateSet \cup \textbf{CQdec}(c, G, \tau)$;
13 return $result$;
```

Algorithm 4: *BWExt*

```
Input: QGP $S$, graph $G$, support threshold $\tau$, frequent edgeset
       $fEdges$.
Output: frequent QGPs after backward expansion.
$result \leftarrow S$, $candidateSet \leftarrow \emptyset$;
$SimpleNodeSet \leftarrow SimpleNode$ of $S$;
for each $u \in SimpleNodeSet$ do
    for each $v \in SimpleNodeSet$ do
        if $S$ does not contain $Edge(u, v)$ and $fEdges$ contains
         $Edge(u, v)$ then
            Let $G'_s$ be the extension of $S$ with $Edge(u, v)$;
            if $G'_s$ is not already generated then
                $candidateSet \leftarrow candidateSet \cup \{G'_s\}$;
for each $c \in candidateSet$ do
    if ISFrequent$(c, G, \tau)$ then
        $result \leftarrow result \cup \textbf{BWExt}(c, G, \tau, fEdges)$;
    else
        $candidateSet \leftarrow candidateSet \cup \textbf{CQdec}(c, G, \tau)$;
return $result$;
```

(III) Backward expansion. QGPM applies *BWExt* for backward expansion. Similar to *FWExt*, *BWExt* takes QGP S as input and then extends S by adding new edges (from *fEdges*), but the difference is that *BWExt* does not introduce new nodes (i.e. two nodes of the extended edge already exist in pattern S).

Example 6. Continuing with Example 5, *FWExt* generates candidate pattern Q_4 by enlarging Q_3 with frequent single-edge pattern $PM - BA$. Then Q_5 is generated by extending Q_4 through *BWExt*.

Algorithm 5: *CQdec*

```
Input: QGP $c$, graph $G$, support threshold $\tau$.
Output: prePattern: the input pattern after the quantifier is
        decremented
$Qnum := c.Qnum$;
$prePattern \leftarrow c$;
while $Qnum > 2$ do
    $prePattern.Qnum := --Qnum$;
    if ISFrequent($prePattern, G, \tau$) then
        return $prePattern$;
        break;
return;
```

In addition, during the different extension phases of the pattern, QGPM uses the procedure *CQdec* to reduce the CQ value of the pattern. Specifically, if a QGP is infrequent but its CQ $m > 2$, we reduce m and then continue to determine if the pattern is frequent.

Algorithm 6: *ISFrequent*

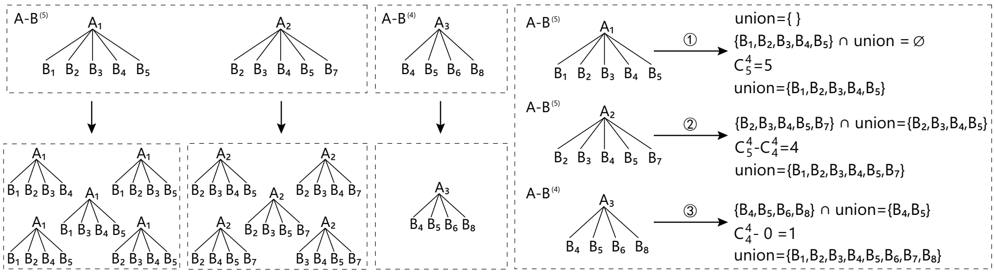
```
Input: QGP $S$, graph $G$, support threshold $\tau$.
Output: true if $S$ is a frequent QGP of $G$, false otherwise
$InstanceSet \leftarrow$ all instance of $S$;
if $InstanceSet.size < \tau$ then
    return $false$;
for each $instance \in InstanceSet$ do
    for each $v \in instance.SimpleNode$ do
        $MNItable.SimplenodeCol(u_i).add(v)$;
    $MNItable.QuantifiednodeCol.add(instance.QuantifiedNode)$;
    $minlength = MinLength(MNItable)$;
    if $minlength \geq \tau$ then
        return $true$;
return $false$;
```

Support evaluation. Recall Section 2, a QGP S is frequent in G (*i.e., sup*$(G, Q) \geq \tau$) if there exist at least τ nodes in each domain (MNIcol(v_1), $\cdots$, MNIcol(v_n)) that are valid variable assignments (*i.e.,* are part of a match) for the corresponding variables

$(v_1, \cdots, v_n)$. To evaluate support, a procedure *ISFrequent* is employed by QGPM. It returns *true* iff S is a frequent QGP in G or *false* otherwise. In a nutshell, if QGP has fewer matches (*i.e.,* instances) than τ, the procedure directly returns *false* (line 3). Following, *ISFrequent* considers each match and places nodes assigned to variables in the corresponding domains (lines 5-7). If all domains have at least τ nodes, then S is frequent in G. Otherwise, *ISFrequent* continues to consider other matches.

3.3 Optimization

To further improve the performance of QGPM, we propose the following optimization methods called storage reduction and computation optimization from the perspective of intermediate result storage and support computation.



(a)Storage reduction (b)Computational optimization

Fig. 4. Optimization

Storage reduction. To see Theorem 1, if $Q1_{(v)}^{(m)} \Rightarrow Q2_{(v)}^{(m')}$, it is not difficult to conclude that each match of $Q1_{(v)}^{(m)}$ must contain a certain number of matches of $Q2_{(v)}^{(m')}$. As shown in Fig. 4(a), denote $A - B^{(5)}$ as $Q_{(B)}^{(5)}$, and denote $A - B^{(4)}$ as $Q_{(B)}^{(4)}$. It is observed that two matches of $Q_{(B)}^{(5)}$ each contain five matches of $Q_{(B)}^{(4)}$(listed in the Fig. 4(a)). In light of this, when m is large, the matching of $Q1_{(v)}^{(m)}$ must contain a huge number of matching of $Q2_{(v)}^{(m')}$. If these split matches are stored, the space overhead is undoubtedly huge. To this end, we propose a non-split storage method, that is, the matching of patterns with large quantifiers is directly assigned to patterns with small quantifiers without splitting. This alleviates the memory overhead to some extent.

Computational optimization. Since the verification of the support does not need to obtain the exact result, it only needs to know whether the number of valid matches corresponding to all the nodes of the current candidate QGP meets the threshold τ. Therefore, according to the above storage method, for quantifier nodes, we use estimation to predict their matches without enumerating them.

Fig. 4(b) shows the specific process, an empty set *union*(recording the matches that have already appeared) is initialized first, and then the existing matches are traversed in turn. For the first match, there are 5 matches of $Q_{(B)}^{(4)}$, and then update the *union*. For the second match, the intersection with the union is $\{B_2, B_3, B_4, B_5\}$, indicating that a match formed by the intersection has been generated before, and if such a match occurs again, it must be subtracted, so the number of matches $Q^{(4)}_{(B)}$ formed by the second match is 4, and the *union* is then updated as well. For the third match, its intersection with *union* is $\{B_4, B_5\}$, and the cardinality of the intersection is less than the number of quantifiers, indicating that the previous match will not be generated, so there is 1 match. The final match count is the sum of the matches obtained each time, *i.e.,* the matches of $Q^{(4)}_{(B)}$ is 10.

4 QGPARs generation

As an application of QGPs, we introduce Quantified Graph Pattern Association Rules (QGPARs), to identify regularity between entities in a graph.

Graph Pattern Association Rule(GPAR) [29]. A GPAR R is defined as $Q_l \to Q_r$, where Q_l and Q_r (1) are different connected graph patterns, (2) share nodes but no common edges. We refer to Q_l and Q_r are the antecedent and consequent of R, respectively.

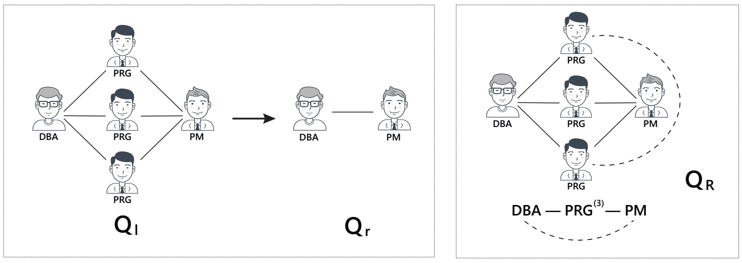


Fig. 5. QGPAR $R : Q_l \to Q_r$

Quantified GPARs (QGPAR). Analogous to GPARs, a QGPAR R is also defined as $Q_l \to Q_r$, with exception that Q_l is a quantified pattern rather than a traditional one. Then, the rule R states that in any graph G, if there is an isomorphism mapping h_l from Q_l to the subgraph G_1, there likely exists another mapping h_r from Q_r to the subgraph G_2, such that for each node $u \in V_l \cap V_r$, if it is mapped to the node v in G_1 by h_l, then it is also mapped to the same node v in G_2 by h_r. For a QGPAR R, we model R as a QGP Q_R by extending Q_l with the edge set of Q_r.

Example 7. Figure 5 shows a QGPAR R found from Fig. 2(a), expressed as $Q_l \to Q_r$, with Q_l and Q_r as its antecedent and consequent, respectively. The QGPAR R can be modeled as a graph pattern Q_R by extending Q_l with the edge set of Q_r.

Support & Confidence. By considering QGPAR R as pattern Q_R, the support of R in G is defined as:

$$Sup(G, R) = Sup(G, Q_R) \tag{1}$$

For the confidence metric, it is defined the same as that of GPARs, *i.e.,*

$$Conf(G, R) = \frac{Sup(G, Q_R)}{Sup(G, Q_l)} \tag{2}$$

QGPARs generation. For any frequent QGP $Q^m_{(v)}$, we enumerate its non-empty subgraph Q_l (including quantifier-constrained node v), and $Q_r = Q^m_{(v)} \backslash Q_l$ (excluding v), then we can generate rules R: $Q_l \rightarrow Q_r$. All rules that meet the support and confidence are valid.

5 Experimental Study

Using real-life and synthetic data, we conducted two sets of experiments to evaluate (1) The efficiency of our algorithm QGPM and the effectiveness of optimization techniques for QGPM; (2) the effectiveness of for identifying correlated entities in large real-world graphs.

DataSets	$\lvert V\rvert$	$\lvert L\rvert$	$\lvert E\rvert$
Citeseer	3312	6	4519
Lasftm_asia	7624	12	27806
Mico	100000	29	1080298
DBLP	425957	25	1049866
Amazon	334863	80	925872
AstroPh	18772	50	396160

Table 2. Description of graphs ($|L|$ indicates no. of node labels).

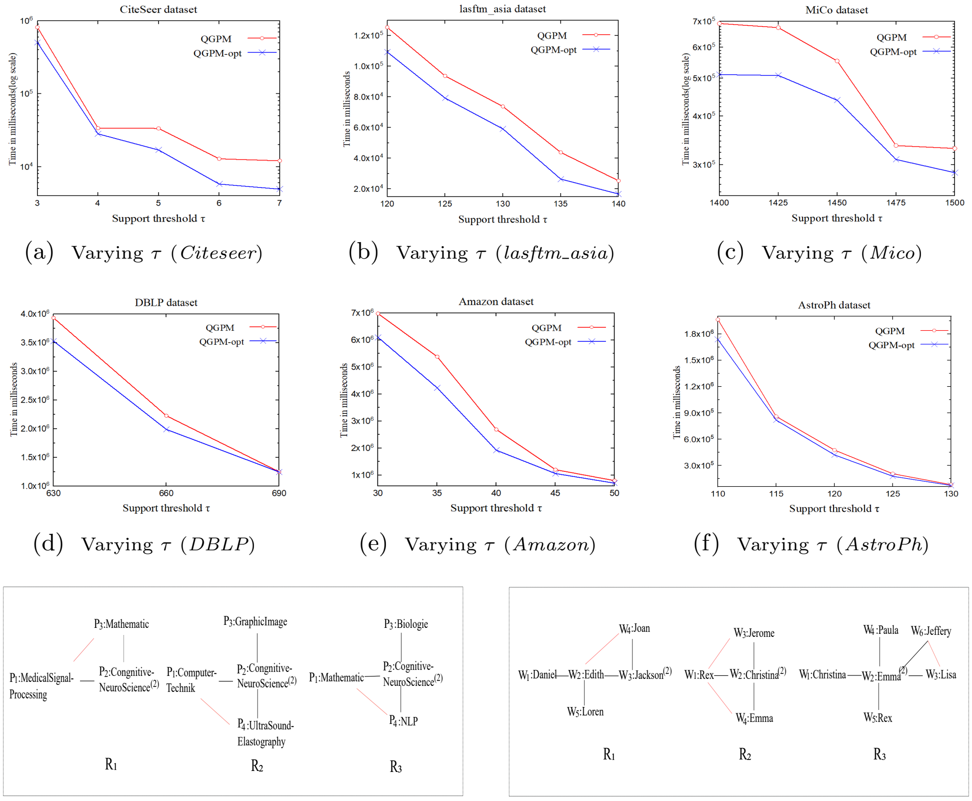
Experimental setting. We used six graphs, (a) CiteSeer [11] represents a graph consisting of publications (nodes) and citations between them (edges). (b) lasftm_asia [2], a social network consisting of users (nodes) and following relationships (edges) between them. (c) Mico [11] models the Microsoft co-authorship information. (d) DBLP [2] represents a collaborative network consisting of authors (nodes) and partnerships (edges) between them. (e) Amazon [2], is a product co-purchasing network. (f) AstroPh [2] covers scientific collaboration between author papers submitted to the Astrophysics category. Table 2 shows the statistics of each graph.

Experimental results. We next report our findings.

Exp-1: Performance of QGPM & QGPM-opt. In this set of experiments, we evaluated the performance of algorithm QGPM compared with QGPM-opt.

The results presented in Figure 6(a)-6(f) show the results of the algorithm on CiteSeer, lasftm_asia, MiCo, DBLP, Amazon, and AstroPh, respectively, and demonstrate the following.(1) All algorithms take longer with a small τ because more candidate patterns and their matches need to be verified. (2) QGPM-opt performs better than QGPM, as unnecessary computations are avoided owing to the strategy QGPM-opt applied.

Frequency threshold. By observing the results, the support threshold τ is a key evaluation metric, as it determines whether a pattern is frequent. As the threshold decreases, there is an exponential number of candidate pattern that leads to exponential runtime. For a given time budget, an efficient algorithm should be able to solve mining problems with low τ values. When τ is given, the efficiency is determined by the execution time.



(a) Varying τ (*Citeseer*)

(b) Varying τ (*lasftm_asia*)

(c) Varying τ (*Mico*)

(d) Varying τ (*DBLP*)

(e) Varying τ (*Amazon*)

(f) Varying τ (*AstroPh*)

(g) Real-life QGPs (*DBLP*)

(h) Real-life QGPs (*AstroPh*)

Fig. 6. Performance of QGPM & QGPM-opt on real-life graphs

Case study. We also manually examined QGPs mined from DBLP and AstroPh. *e.g.,* Fig. 6(g) shows three QGPs discovered from DBLP. Fig. 6(h) shows three QGPs discovered from AstroPh.

Exp-2: Prediction accuracy. In this set of tests, we evaluated the prediction accuracy of QGPARs vs. Jaccard [25] and SimRank [25]. The settings of the evaluation are as follows.

The prediction accuracy of QGPARs is tested via cross validation. That is, given a graph G, we partition it into two fragments F_1 and F_2, mine and select the top-k (k ranges from 10 to 50 in 10 increments) QGPARs from F_1, and evaluate their prediction accuracy, which is defined as $Acc(R) = \frac{Sup(F_2,Q_R)}{Sup(F_2,Q_l)}$, on F_2. We used Gain [17] and Interest [28] to rank the QGPARs, where θ is set as 0.4 for Gain.

We compared our rule-based method vs. Jaccard and SimRank, on three real-life graphs. The tests of Jaccard and SimRank are along the same line as above but with 4-fold cross-validation. Specifically, (1) we extracted a small subgraph G_F of size (10000,30729), (5000,88822), and (10000,5678) from Amazon, AstroPh, and

		Amazon					AstroPh					DBLP				
		top-k(Acc(R),%)					top-k(Acc(R),%)					top-k(Acc(R),%)				
		10	20	30	40	50	10	20	30	40	50	10	20	30	40	50
QGPARs	Gain	83.4	81.1	78.6	75.9	73.7	92.9	93.3	93.4	92.2	83.0	90.9	90.1	88.9	86.9	83.8
	Interest	79.1	76.9	76.8	73.8	71.7	92.7	91.9	90.8	90.0	89.7	91.5	86.1	86.1	85.1	82.7
		top-L(Acc,%)					top-L(Acc,%)					top-L(Acc,%)				
		50	100	150	200	250	50	100	150	200	250	50	100	150	200	250
Jaccard		43.5	42.5	43.6	43.4	42.0	56.5	43.0	43.0	33.5	27.0	47.0	49.7	48.7	48.4	48.1
Simrank		18.0	20.2	22.1	25.2	22.4	13.0	14.0	13.8	17.7	21.1	73.5	44.2	31.0	23.6	19.6

Table 3. Prediction accuracy of QGPARs vs. Jaccard and SimRank

DBLP respectively, since SimRank is too costly to run on the entire graph; (2) we run Jaccard and SimRank on the training set, and obtain a list of node pairs; (3) we then sort node pairs according to their similarity and next select top-L node pairs as the predicted edges. Here, the prediction accuracy is defined as $Acc = \frac{L_r}{L}$, where L ranges from 50 to 250 in 50 increments and L_r is the number of edges that are correctly predicted. Note that the parameter C used by SimRank is set as 0.8, and the number of iterations is fixed as 5.

Table 3 shows the prediction accuracy of QGPARs, compared with Jaccard and SimRank. We find that 1) using the top-50 QGPARs can predict missing relationships with an average accuracy of up to 84.3% on three real graphs, and the higher k is, the lower $Acc(R)$ is, as expected; 2) our rule-based method is superior to Jaccard and SimRank in terms of prediction accuracy. Taking Amazon as an example, when top-50 interest QGPARs are used, the prediction accuracy (*i.e.,* 71.7%) is 170.71% and 320.0% of that of Jaccard and SimRank (L=250) respectively; 3) two different metrics *i.e.,* Gain and Interest leads to different while close accuracy.

6 Conclusion

Incorporating counting quantifiers into traditional graph patterns can capture more complex structural relationships in graph data. Accordingly, we have studied the problem of mining Quantified Graph Pattern (QGP) in a single large graph and propose QGPM to identify QGPs in the graph that satisfy threshold conditions. As an application of QGP, we also proposed Quantified Graph Pattern Association Rules (QGPAR), which can model more complex associations between social entities. Our experimental study validates the efficiency and effectiveness of the algorithm. Therefore, we think our method is promising in social network analysis.

The study of QGP is still in its infancy. One topic for future work is to extend quantifiers to multiple nodes to make graph patterns more diverse. Another topic involves the mining and application of quantified graph patterns in distributed environment.

References

1. *Nielsen global online consumer survey.* http://www.nielsen.com/content/dam/ corporate/us/en/newswire/uploads/2009/07/pr global-study 07709.pdf.

2. *social network.* http://snap.stanford.edu/data/.
3. E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y. Chang, and P. Kalnis. Incremental frequent subgraph mining on large evolving graphs. *IEEE Trans. Knowl. Data Eng.*, 29(12):2710–2723, 2017.
4. S. Amer-Yahia, L. V. S. Lakshmanan, and C. Yu. Socialscope: Enabling information discovery on social content sites. *CoRR*, abs/0909.2058, 2009.
5. N. Ashraf, R. R. Haque, M. A. Islam, C. F. Ahmed, C. K. Leung, J. J. Mai, and B. H. Wodi. Wefres: weighted frequent subgraph mining in a single large graph. In *ICDM 2019*, pages 201–215. ibai Publishing, 2019.
6. R. Bapna and A. Umyarov. Do your online friends make you pay? A randomized field experiment on peer influence in online social networks. *Manag. Sci.*, 61(8):1902–1920, 2015.
7. H. Blau, N. Immerman, and D. Jensen. A visual language for querying and updating graphs. *University of Massachusetts Amherst Computer Science Technical Report*, 37:2002, 2002.
8. B. Bringmann and S. Nijssen. What is frequent in a single graph? In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 5012 of *Lecture Notes in Computer Science*, pages 858–863. Springer, 2008.
9. F. Bry, T. Furche, B. Marnette, C. Ley, B. Linse, and O. Poppe. Sparqlog: SPARQL with rules and quantification. In *Semantic Web Information Management - A Model-Based Perspective*, pages 341–370. Springer, 2009.
10. L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *TPAMI*, 26(10):1367–1372, 2004.
11. M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. GRAMI: frequent subgraph and pattern mining in a single large graph. *Proc. VLDB Endow.*, 7(7):517–528, 2014.
12. W. Fan. Graph pattern matching revised for social network analysis. In *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*, pages 8–21. ACM, 2012.
13. W. Fan, X. Wang, Y. Wu, and J. Xu. Association rules with graph patterns. *Proc. VLDB Endow.*, 8(12):1502–1513, 2015.
14. W. Fan, Y. Wu, and J. Xu. Adding counting quantifiers to graph patterns. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1215–1230. ACM, 2016.
15. M. Fiedler and C. Borgelt. Subgraph support in a single large graph. In *Workshops Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007)*, pages 399–404. IEEE Computer Society, 2007.
16. E. Gudes, S. E. Shimony, and N. Vanetik. Discovering frequent graph patterns using disjoint paths. *IEEE Trans. Knowl. Data Eng.*, 18(11):1441–1456, 2006.
17. R. J. B. Jr. and R. Agrawal. Mining the most interesting rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154. ACM, 1999.
18. D. Kavitha, D. Haritha, and Y. Padma. Optimized candidate generation for frequent subgraph mining in a single graph. In *Proceedings of International Conference on Computational Intelligence and Data Engineering*, pages 259–272. Springer, 2021.
19. N. Le, B. Vo, L. B. Q. Nguyen, H. Fujita, and B. Le. Mining weighted subgraphs in a single large graph. *Inf. Sci.*, 514:149–165, 2020.
20. L. Li, P. Ding, H. Chen, and X. Wu. Frequent pattern mining in big social graphs. *IEEE Transactions on Emerging Topics in Computational Intelligence*, pages 1–11, 2021.
21. M. Liaqat, S. Khan, M. S. Younis, M. Majid, and K. Rajpoot. Applying uncertain frequent pattern mining to improve ranking of retrieved images. *Appl. Intell.*, 49(8):2982–3001, 2019.

22. W. Lin, S. A. Alvarez, and C. Ruiz. Collaborative recommendation via adaptive association rule mining. *Data Mining and Knowledge Discovery*, 6(1):83–105, 2000.
23. H. Mahfoud. Expressive top-k matching for conditional graph patterns. *Neural Computing and Applications*, pages 1–17, 2021.
24. M. S. Martín, C. Gutiérrez, and P. T. Wood. SNQL: A social networks query and transformation language. volume 749 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2011.
25. V. Martínez, F. Berzal, and J. C. C. Talavera. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4):69:1–69:33, 2017.
26. A. Ray, L. Holder, and S. Choudhury. Frequent subgraph discovery in large attributed streaming graphs. In *Proceedings of the 3rd international workshop on big data*, pages 166–181. PMLR, 2014.
27. Q. Song, Y. Wu, P. Lin, X. Dong, and H. Sun. Mining summaries for knowledge graph search. *IEEE Trans. Knowl. Data Eng.*, 30(10):1887–1900, 2018.
28. B. Vo and B. Le. Interestingness measures for association rules: Combination between lattice and hash tables. *Expert Syst. Appl.*, 38(9):11630–11640, 2011.
29. X. Wang, Y. Xu, and H. Zhan. Extending association rules with graph patterns. *Expert Syst. Appl.*, 141, 2020.