



THE UNIVERSITY *of* EDINBURGH

Edinburgh Research Explorer

## Association Rules with Graph Patterns

**Citation for published version:**

Fan, W, Wang, X, Wu, Y & Xu, J 2015, 'Association Rules with Graph Patterns', *Proceedings of the VLDB Endowment (PVLDB)*, vol. 8, no. 12, pp. 1502-1513. <https://doi.org/10.14778/2824032.2824048>

**Digital Object Identifier (DOI):**

[10.14778/2824032.2824048](https://doi.org/10.14778/2824032.2824048)

**Link:**

[Link to publication record in Edinburgh Research Explorer](#)

**Document Version:**

Peer reviewed version

**Published In:**

Proceedings of the VLDB Endowment (PVLDB)

**General rights**

Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

**Take down policy**

The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact [openaccess@ed.ac.uk](mailto:openaccess@ed.ac.uk) providing details, and we will remove access to the work immediately and investigate your claim.



# Association Rules with Graph Patterns

Wenfei Fan<sup>1,2</sup>      Xin Wang<sup>3</sup>      Yinghui Wu<sup>4</sup>      Jingbo Xu<sup>1,2</sup>  
<sup>1</sup>Univ. of Edinburgh    <sup>2</sup>Beihang Univ.    <sup>3</sup>Southwest Jiaotong Univ.    <sup>4</sup>Washington State Univ.  
 {wenfei@inf, jingbo.xu@}.ed.ac.uk, xinwang@swjtu.cn, yinghui@eecs.wsu.edu

## ABSTRACT

We propose graph-pattern association rules (GPARs) for social media marketing. Extending association rules for itemsets, GPARs help us discover regularities between entities in social graphs, and identify potential customers by exploring social influence. We study the problem of discovering top- $k$  diversified GPARs. While this problem is NP-hard, we develop a parallel algorithm with accuracy bound. We also study the problem of identifying potential customers with GPARs. While it is also NP-hard, we provide a parallel scalable algorithm that guarantees a polynomial speedup over sequential algorithms with the increase of processors. Using real-life and synthetic graphs, we experimentally verify the scalability and effectiveness of the algorithms.

## 1. INTRODUCTION

Association rules have been well studied for discovering regularities between items in relational data, for promotional pricing and product placements [4, 45]. They have a traditional form  $X \Rightarrow Y$ , where  $X$  and  $Y$  are disjoint itemsets.

There have been recent interests in studying associations between entities in social graphs. Such associations are useful in social media marketing; indeed, “90% of customers trust peer recommendations versus 14% who trust advertising” [2], and “60% of users said Twitter plays an important role in their shopping” [43]. Nonetheless, association rules for social graphs are more involved than rules for itemsets.

**Example 1:** (1) Association rules for social graphs are defined on graphs rather on itemsets. Below is an example.

- If (a)  $x$  and  $x'$  are friends living in the same city  $c$ , (b) there are at least 3 French restaurants in  $c$  that  $x$  and  $x'$  both like, and if (c)  $x'$  visits a newly opened French restaurant  $y$  in  $c$ , then  $x$  may also visit  $y$ .

The antecedent of the rule can be represented as a graph pattern  $Q_1$  (with solid edges) shown in Fig. 1(a), and the consequent is indicated by a dotted edge  $\text{visit}(x, y)$ . A succinct presentation of  $Q_1$  associates integer 3 with “French

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-nd/3.0/>. Obtain permission prior to any use beyond those covered by the license. Contact copyright holder by emailing [info@vldb.org](mailto:info@vldb.org). Articles from this volume were invited to present their results at the 41st International Conference on Very Large Data Bases, August 31st - September 4th 2015, Kohala Coast, Hawaii.

*Proceedings of the VLDB Endowment*, Vol. 8, No. 12  
 Copyright 2015 VLDB Endowment 2150-8097/15/08.

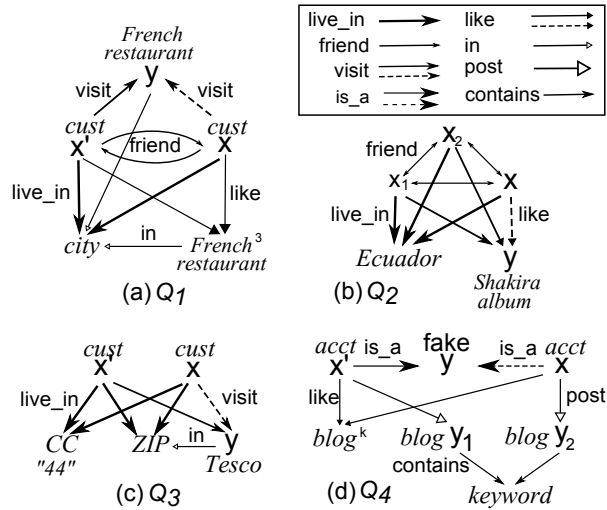


Figure 1: Associations as graph patterns

Restaurant” to indicate its 3 copies. As opposed to conventional association rules,  $Q_1$  specifies conditions as topological constraints: edges between customers (the friend relation), customers and restaurants (like, visit), city and restaurants (in), and between city and customers (live\_in).

In a social graph  $G$ , for  $x$  and  $y$  satisfying the antecedent  $Q_1$  via graph pattern matching, we can recommend  $y$  to  $x$ .

(2) As opposed to rules for itemsets, association rules for social graphs may target social groups with multiple entities:

- If (a)  $x$ ,  $x_1$  and  $x_2$  are friends, (b) they all live in Ecuador, and (c) if  $x_1$ ,  $x_2$  both like Shakira’s album  $y$  (a Colombian singer), then  $x$  may also like  $y$ .

This rule is depicted in Fig. 1(b), in which a graph pattern  $Q_2$  (excluding the dotted edge) specifies conditions for  $(x, y)$  as antecedent, and dotted edge  $\text{like}(x, y)$  indicates its consequent. We can use the rule to identify potential customers  $x$  of  $y$ , characterized by a social group of three members.

(3) Association rules with graph patterns conveniently extend data dependencies such as conditional functional dependencies (CFDs) [14] in the context of social networks.

- If the addresses of  $x$  and  $x'$  have the same country code “44” and same zip code, and if  $x'$  shops at a Tesco store  $y$  with the same zip, then  $x$  may also shop at  $y$ .

Such a rule (Fig. 1(c)) embeds a corresponding CFD in its pattern  $Q_3$ , stating that if  $x$  and  $x'$  live in the UK with the same zip code, then they live on the same street. The rule is valid in the UK where zip code determines street.

(4) The applications of association rules are not limited to marketing activities. They also help us detect scams. As an example, the rule below is used to identify fake accounts [9].

- If (a) account  $x'$  is confirmed fake, (b) both  $x$  and  $x'$  like blogs  $P_1, \dots, P_k$ , (c)  $x$  posts blog  $y_1$ , (d)  $x'$  posts  $y_2$ , and (e) if  $y_1$  and  $y_2$  contain the same particular content (keyword), then  $x$  is likely a fake account.

As depicted in Fig. 1(d), its antecedent is given by graph pattern  $Q_4$  (excluding the dotted edge), and its consequent is the dotted edge  $\text{is\_a}(x, \text{fake})$ . In a social graph  $G$ , the rule is to identify suspects for fake accounts, *i.e.*, accounts  $x$  that satisfy the structural constraints of pattern  $Q_4$ .  $\square$

The need for graph-pattern association rules (GPARs) is evident in social media marketing, community structure analysis, social recommendation, knowledge extraction and link prediction [33]. Such rules, however, depart from association rules for itemsets, and introduce several challenges. (1) Conventional support and confidence metrics no longer work for GPARs. (2) Mining algorithms for traditional rules and frequent graph patterns cannot be used to discover practical diversified GPARs. (3) A major application of GPARs is to identify potential customers in social graphs. This is costly: graph pattern matching by subgraph isomorphism is intractable. Worse still, real-life social graphs are often big, *e.g.*, Facebook has 13.1 billion nodes and 1 trillion links [21].

**Contributions.** This paper proposes GPARs, and provide effective algorithms for discovering and applying GPARs.

(1) We introduce graph-pattern association rules (GPARs) for social media marketing (Section 2). GPARs differ from conventional rules for itemsets in both syntax and semantics. A GPAR defines its antecedent as a graph pattern, which specifies associations between entities in a social graph, and explores social links, influence and recommendations. It enforces conditions via both value bindings (*e.g.*, “44”) and topological constraints by subgraph isomorphism.

(2) We define topological support and confidence metrics for GPARs (Section 3). Conventional support for itemsets is no longer anti-monotonic for GPARs. We define support in terms of distinct “potential customers” by revising a measure proposed by [7]. We propose a confidence measure for GPARs by revising Bayes Factor [31] to incorporate the local closed world assumption [11, 17]. This allows us to cope with (incomplete) social graphs, and to identify interesting GPARs with correlated antecedent and consequent.

(3) We study a new mining problem, referred to as the *diversified mining problem* and denoted by DMP (Section 4). It is a bi-criteria optimization problem to discover top- $k$  GPARs. While useful, DMP is NP-hard. Nonetheless, we develop a parallel approximation algorithm with a *constant accuracy bound*. We also provide optimization methods to filter redundant or non-promising rules as early as possible.

(4) We also study how to identify potential customers by applying GPARs, referred to as *the entity identification problem* and denoted by EIP (Section 5). Given a social graph  $G$  and a set  $\Sigma$  of GPARs pertaining to an event  $p(x, y)$ , we identify potential customers  $x$  of  $y$  in  $G$  with confidence above a given bound  $\eta$ , by using GPARs in  $\Sigma$ . We show that it is NP-hard even to decide whether such  $x$  exists.

Despite this, we develop a *parallel scalable* algorithm for EIP such that its response time is in  $O(t(|G|, |\Sigma|)/n)$ , a poly-

nomial reduction in the running time  $t(|G|, |\Sigma|)$  of *sequential algorithms*, by using  $n$  processors. Hence given a big graph, we can identify potential customers in it by increasing  $n$ .

(5) Using real-life and synthetic graphs, we experimentally verify the scalability and effectiveness of our algorithms (Section 6). We find the following. (a) Our algorithms for DMP and EIP scale well with the increase of processors ( $n$ ): they are on average 3.2 and 3.53 times faster on real-world social networks, respectively, when  $n$  increases from 4 to 20. (b) They work reasonably well on large graphs: the one for DMP takes less than 9 minutes (533.2 seconds) on graphs with 30 million nodes and edges, and the one for EIP takes 45 seconds on graphs with 150 million nodes and edges for 24 GPARs, with 20 processors. (c) The DMP algorithm finds interesting GPARs from real-life social graphs. (d) Our optimization methods are effective: they speed up DMP and EIP processing by 1.52 and 1.27 times, respectively, on real-life graphs. Hence, despite their complexity, applying and discovering GPARs are feasible in practice via parallelization.

**Related Work.** We categorize related work as follows.

Association rules. Introduced in [4], association rules are defined on relations of transaction data. Prior work on association rules for social networks [41] and RDF knowledge bases resorts to mining conventional rules and Horn rules (as conjunctive binary predicates) [17] over tuples with extracted attributes from social graphs, instead of exploiting graph patterns. While [6] studies time-dependent rules via graph patterns, it focuses on evolving graphs and hence adopts different semantics for support and confidence.

GPARs extend association rules from relations to graphs. (a) It demands topological support and confidence metrics. Moreover, incomplete information is common in social graphs [11, 17] and has to be incorporated into the metrics. (b) GPARs are interpreted with isomorphic functions and hence, cannot be expressed as conjunctive queries, which do not support negation or inequality needed for functions. (c) Applying GPARs becomes an intractable problem of multi-pattern-query processing in big graphs. (d) Mining (diversified) GPARs is beyond rule mining from itemsets [46].

Graph pattern mining. There have been algorithms for pattern mining in graph databases [22, 24] (see [25] for a survey). Large-scale mining techniques are also studied in a single graph [13], notably top- $k$  algorithms [16, 27, 42, 44]. To reduce the cost, scalable subgraph isomorphism algorithms, *e.g.*, [38], can be adopted to generate pattern candidates. Diversity of graph patterns is not studied there.

However, (a) pattern mining over graph databases [24, 27] cannot be used to mine GPARs, as their anti-monotonic property does not hold in a single graph [25]. (b) While mining single graphs is based only on isomorphic counting [13], DMP is bi-criteria optimization problem for confidence and diversity of GPARs, apart from [16, 44]. We are not aware of prior work on discovering diversified graph patterns.

Graph pattern matching. Several parallel algorithms have been developed for subgraph isomorphism, *e.g.*, [28, 37, 38], and for multi-pattern optimization, *e.g.*, [23, 32]. Our algorithms for EIP differ from the prior work in the following. (a) Instead of enumerating isomorphic matches, EIP identifies a potential customer *once* one match is found, and moreover, computes its associated confidence. That is, EIP is beyond conventional subgraph isomorphism. (b) We provide paral-

lel scalable algorithms for multi-pattern matching. To the best of our knowledge, these are among the first algorithms on big graphs that *guarantee a polynomial speedup over sequential algorithms* with the increase of processors [30]. (c) We propose optimization strategies that are not studied by previous work. This said, prior optimization techniques can be incorporated into GPAR-based entity identification; *e.g.*, the methods of [32] to extract common sub-patterns.

## 2. ASSOCIATION VIA GRAPH PATTERNS

In this section we define graph-pattern association rules.

### 2.1 Graphs, Patterns, and Pattern Matching

We start with notions of graphs and graph patterns.

**Graphs.** A *graph* is defined as  $G = (V, E, L)$ , where (1)  $V$  is a finite set of nodes; (2)  $E \subseteq V \times V$  is a set of edges, in which  $(v, v')$  denotes an edge from node  $v$  to  $v'$ ; (3) each node  $v$  in  $V$  (resp. edge  $e$ ) carries  $L(v)$  (resp.  $L(e)$ ), indicating its label or content *e.g.*, *cust*, *French restaurant*, 44 (resp. *post*, *like*), as found in social networks and property graphs.

**Example 2:** Two graphs  $G_1$  and  $G_2$  are shown in Fig. 2. (1) Graph  $G_1$  depicts a restaurant recommendation network. For instance, *cust*<sub>1</sub> and *cust*<sub>2</sub> (labeled *cust*) live in New York; they share common interests in 3 *French restaurants* (marked with superscript 3 for simplicity); and they both visit a newly opened French restaurant “Le Bernardin” in New York. (2) Graph  $G_2$  shows activities of social accounts. It contains (a) accounts *acct*<sub>1</sub>, ..., *acct*<sub>4</sub> (labeled *acct*), (b) *blogs*  $p_1, \dots, p_7$ ; and (c) edges from accounts to blogs. For example, edge *post*(*acct*<sub>1</sub>,  $p_1$ ) means that account *acct*<sub>1</sub> posts blog  $p_1$ , which contains keyword  $w_1$  “claim a prize”. □

**Patterns.** A *pattern query*  $Q$  is a graph  $(V_p, E_p, f, C)$ , in which  $V_p$  and  $E_p$  are the set of pattern nodes and edges, respectively; each node  $u_p$  in  $V_p$  (resp. edge  $e_p$  in  $E_p$ ) has a label  $f(u_p)$  (resp.  $f(e_p)$ ) specifying a search condition, *e.g.*, *city*, or “44” for *value binding* ( $Q_3$  of Example 1). For succinct representation, a node  $u_p$  can be labeled with an integer  $C(u_p) = k$ , indicating  $k$  copies of  $u_p$  with the same label and associated links in the common neighborhood.

**Graph pattern matching.** We first review two notions of subgraphs. (1) 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 edge  $e \in E'$ ,  $L'(e) = L(e)$ , and for each  $v \in V'$ ,  $L'(v) = L(v)$ . (2) We say that  $G'$  is a *subgraph induced* by a set  $V'$  of nodes if  $G' \subseteq G$  and  $E'$  consists of all those edges in  $G$  whose endpoints are both in  $V'$ .

We adopt subgraph isomorphism for pattern matching. A *match* of pattern  $Q$  in graph  $G$  is a *bijective function*  $h$  from the nodes of  $Q$  to the nodes of a subgraph  $G'$  of  $G$  such that (a) for each node  $u \in V_p$ ,  $f(u) = L(h(u))$ , and (b)  $(u, u')$  is an edge in  $Q$  if and only if  $(h(u), h(u'))$  is an edge in  $G'$ , and  $f(u, u') = L(h(u), h(u'))$ . We say that  $G'$  *matches*  $Q$ .

Note that *similarity predicates* can be used instead of equality “=” with no impact on our algorithms.

We denote by  $Q(G)$  the set of all matches of  $Q$  in  $G$ . For each pattern node  $u$ , we use  $Q(u, G)$  to denote the set of all *matches* of  $u$  in  $Q(G)$ , *i.e.*,  $Q(u, G)$  consists of nodes  $v$  in  $G$  such that there exists a function  $h$  under which a subgraph  $G' \in Q(G)$  is isomorphic to  $Q$ ,  $v \in G'$  and  $h(u) = v$ .

**Example 3:** For  $Q_1$  of Fig. 1 and  $G_1$  of Fig. 2, a match in  $Q_1(G)$  is  $x \mapsto \text{cust}_1$ ,  $x' \mapsto \text{cust}_2$ ,  $city \mapsto \text{New York}$ ,  $y \mapsto$

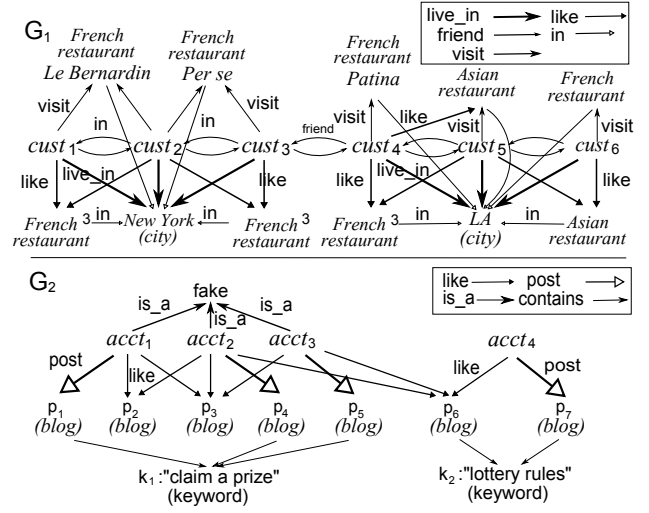


Figure 2: Labeled social graphs

Le Bernardin, and French restaurant<sup>3</sup> to 3 French restaurants. Here  $Q_1(x, G_1)$  includes *cust*<sub>1</sub>–*cust*<sub>3</sub> and *cust*<sub>5</sub>. □

A pattern  $Q' = (V'_p, E'_p, f', C')$  is *subsumed* by another pattern  $Q = (V_p, E_p, f, C)$ , denoted by  $Q' \sqsubseteq Q$ , if  $(V'_p, E'_p)$  is a subgraph of  $(V_p, E_p)$ , and functions  $f'$  and  $C'$  are restrictions of  $f$  and  $C$  in  $V$ , respectively. Observe that if  $Q' \sqsubseteq Q$ , then for any graph  $G'$  that matches  $Q$ , there exists a subgraph  $G''$  of  $G'$  such that  $G''$  matches  $Q'$ .

We will use the following notations. (1) For a pattern  $Q$  and a node  $x$  in  $Q$ , the *radius* of  $Q$  at  $x$ , denoted by  $r(Q, x)$ , is the longest distance from  $x$  to all nodes in  $Q$  when  $Q$  is treated as an *undirected* graph. (2) Pattern  $Q$  is *connected* if for each pair of nodes in  $Q$ , there exists an undirected path in  $Q$  between them. (3) For a node  $v_x$  in a graph  $G$  and a positive integer  $r$ ,  $N_r(v_x)$  denotes the set of all nodes in  $G$  within radius  $r$  of  $v_x$ . (4) The *size*  $|G|$  of  $G$  is  $|V| + |E|$ , the number of nodes and edges in  $G$ . (5) Node  $v'$  is a *descendant* of  $v$  if there is a directed path from  $v$  to  $v'$  in  $G$ .

### 2.2 Graph Pattern Association Rules

We now define graph-pattern association rules.

**GPARs.** A *graph-pattern association rule* (GPAR)  $R(x, y)$  is defined as  $Q(x, y) \Rightarrow q(x, y)$ , where  $Q(x, y)$  is a graph pattern in which  $x$  and  $y$  are two *designated nodes*, and  $q(x, y)$  is an edge labeled  $q$  from  $x$  to  $y$ , on which the same search conditions as in  $Q$  are imposed. We refer to  $Q$  and  $q$  as the *antecedent* and *consequent* of  $R$ , respectively.

The rule states that *for all nodes*  $v_x$  and  $v_y$  in a (social) graph  $G$ , if *there exists* a match  $h \in Q(G)$  such that  $h(x) = v_x$  and  $h(y) = v_y$ , *i.e.*,  $v_x$  and  $v_y$  match the designated nodes  $x$  and  $y$  in  $Q$ , respectively, then the consequent  $q(v_x, v_y)$  will likely hold. Intuitively,  $v_x$  is a potential customer of  $v_y$ .

We model  $R(x, y)$  as a *graph pattern*  $P_R$ , by extending  $Q$  with a (dotted) edge  $q(x, y)$ . We refer to pattern  $P_R$  as  $R$  when it is clear from the context. We treat  $q(x, y)$  as pattern  $P_q$ , and  $q(x, G)$  as the set of matches of  $x$  in  $G$  by  $P_q$ .

We consider practical and nontrivial GPARs by requiring that (1)  $P_R$  is connected; (2)  $Q$  is *nonempty*, *i.e.*, it has at least one edge; and (3)  $q(x, y)$  does not appear in  $Q$ .

**Example 4:** Recall the first association rule described in Example 1. It can be expressed as a GPAR  $R_1(x, y)$ :

$Q_1(x, y) \Rightarrow \text{visit}(x, y)$ , where its antecedent is the pattern  $Q_1$  given in Example 1, and its consequent is  $\text{visit}(x, y)$ . The GPAR can be depicted as the graph pattern of Fig. 1(a), by extending  $Q_1(x, y)$  with a dotted edge for  $\text{visit}(x, y)$ .

The last rule of Example 1 is written as  $R_4(x, y): Q_4(x, y) \Rightarrow \text{is\_a}(x, y)$ , where in  $Q_4$ ,  $y = \text{fake}$  is a value binding. The GPAR is depicted as the pattern of Fig. 1(d). In  $\text{is\_a}(x, y)$ , the same search condition  $y = \text{fake}$  is imposed.  $\square$

**Remark.** (1) To simplify the discussion, we define the consequent of GPAR with a single predicate  $q(x, y)$  following [4]. However, a consequent can be readily extended to *multiple* predicates and even to a *graph pattern*. (2) Conventional association rules [4] and a range of predication and classification rules [39] are a special case of GPARs, since their antecedents can be modeled as a graph pattern in which nodes denote items. Conditional functional dependencies [14] can also be represented by GPARs (see  $Q_3$  of Fig. 1(c)).

### 3. SUPPORT AND CONFIDENCE

We next define support and confidence for GPARs.

**Support.** The support of a graph pattern  $Q$  in a graph  $G$ , denoted by  $\text{supp}(Q, G)$ , indicates how often  $Q$  is applicable. As for association rules for itemsets, the support measure should be *anti-monotonic*, *i.e.*, for patterns  $Q$  and  $Q'$ , if  $Q' \sqsubseteq Q$ , then in any graph  $G$ ,  $\text{supp}(Q', G) \geq \text{supp}(Q, G)$ .

One may want to define  $\text{supp}(Q, G)$  as *the number*  $\|Q(G)\|$  of matches of  $Q$  in  $G$ , following its counterpart for itemsets [46]. However, as observed in [7, 13, 25], this conventional notion is *not* anti-monotonic. For example, consider pattern  $Q'$  with a single node labeled *cust*, and  $Q$  with a single edge like(*cust*, *French restaurant*). When posed on  $G_1$ ,  $\|Q(G)\| = 18 > \|Q'(G)\| = 6$  (since *French restaurant*<sup>3</sup> denotes 3 nodes labeled *French restaurant*), although  $Q' \sqsubseteq Q$ .

To cope with this, we revise the support measure proposed in [7]. We define the support of the designated node  $x$  of  $Q$  as  $\|Q(x, G)\|$ , *i.e.*, the number of distinct matches of  $x$  in  $Q(G)$ . We define *the support of  $Q$  in  $G$*  as

$$\text{supp}(Q, G) = \|Q(x, G)\|.$$

One can verify that this support measure is *anti-monotonic*.

For a GPAR  $R(x, y): Q(x, y) \Rightarrow q(x, y)$ , we define

$$\text{supp}(R, G) = \|P_R(x, G)\|,$$

by treating  $R$  as pattern  $P_R(x, y)$  with designated nodes  $x, y$ .

**Example 5:** For GPAR  $R_1(x, y): Q_1(x, y) \Rightarrow \text{visit}(x, y)$  of Example 4 and graph  $G_1$  of Fig 2, (1)  $\|Q_1(x, G_1)\| = 4$  (see Example 3); hence  $\text{supp}(Q_1, G_1)$  is 4; and (2)  $\text{supp}(R_1, G_1) = \|P_{R_1}(x, G_1)\| = 3$ , where  $x$  has 3 matches *cust*<sub>1</sub>–*cust*<sub>3</sub>.

Similarly, consider  $R_4(x, y): Q_4(x, y) \Rightarrow \text{is\_a}(x, y)$  of Example 4 and graph  $G_2$  in Fig 2, where  $y = \text{fake}$ . When  $k=2$ ,  $\text{supp}(R_4, G_2) = \text{supp}(Q_4, G_2) = \|Q_4(x, G_2)\| = 3$ , with matches *acct*<sub>1</sub>–*acct*<sub>3</sub> for the designated node  $x$  in  $Q_4$ .  $\square$

**Confidence.** To find how likely  $q(x, y)$  holds when  $x$  and  $y$  satisfy the constraints of  $Q(x, y)$ , we study the *confidence* of  $R(x, y)$  in  $G$ , denoted as  $\text{conf}(R, G)$ . One may want to adopt the conventional confidence for association rules, and define  $\text{conf}(R, G)$  as  $\frac{\text{supp}(R, G)}{\text{supp}(Q, G)}$ . That is, every match  $x$  in  $Q$  but not in  $R$  is considered as negative example for  $R$ . However, as observed in [11, 17], the standard confidence is blind to the distinction between “negative” and “unknown”. This is particularly an overkill when  $G$  is incomplete [11, 34].

**Example 6:** Consider pattern  $Q_2$  in Fig. 1(b). Let  $Q_2(x, G)$  contain three matches  $v_1, v_2, v_3$  of  $x_1, x_2, x_3$  in a social graph  $G$ , all living in Ecuador, where (1)  $v_1$  has an edge like to *Shakira album*, (2)  $v_2$  has only a single edge like to *MJ’s album*, and (3)  $v_3$  has no edge of type like. Conventional confidence treats  $v_2$  and  $v_3$  both as negative examples, with  $\text{conf}(R_2, G) = \frac{1}{3}$ . However,  $G$  may be incomplete:  $v_3$  has not entered any albums she likes. Thus we should treat  $v_3$  as “unknown”, not as a counterexample to  $R_2$ .  $\square$

Indeed, closed world assumption may not hold for social network [34]. To distinguish “unknown” cases from true negative for GPAR mining in incomplete social networks, we adopt the *local closed world assumption* [11, 17], as commonly used in mining incomplete knowledge bases.

*Local closed world assumption (LCWA).* Given a predicate  $q(x, y)$ , we introduce the following notations.

- (1)  $\text{supp}(q, G) = \|P_q(x, G)\|$ , the number of matches of  $x$ ;
- (2)  $\text{supp}(\bar{q}, G)$ , the number of nodes  $u$  in  $G$  that (a) have the same label as  $x$ , (b) have at least one edge of type  $q$ , but (c)  $u \notin P_q(x, G)$ ; and
- (3)  $\text{supp}(Q\bar{q}, G)$ , the number of nodes that satisfy conditions (a) to (c) of (2), and are also in  $Q(x, G)$ .

Given an (incomplete) social network  $G$  and a predicate  $q(x, y)$ , the local closed world assumption (LCWA) distinguishes the following three cases for a node  $u$ .

- (1) “positive” case, if  $u \in P_q(x, G)$ ;
- (2) “negative” case, for every  $u$  counted in  $\text{supp}(\bar{q}, G)$ ; and
- (3) “unknown” case, for every  $u$  that satisfies the search condition of  $x$  but has *no* edge labeled as  $q$ .

That is,  $G$  is assumed “locally complete”: it either gives all correct local information of  $u$  in connection with predicate  $q$ , or knows nothing about  $q$  at node  $u$  (hence unknown cases).

Based on LCWA, we define  $\text{conf}(R, G)$  by revising Bayes Factor (BF) of association rules [31] as follows:

$$\text{conf}(R, G) = \frac{\text{supp}(R, G) * \text{supp}(\bar{q}, G)}{\text{supp}(Q\bar{q}, G) * \text{supp}(q, G)}.$$

Intuitively,  $\text{conf}(R, G)$  measures the product of *completeness* and *discriminant*. A GPAR  $R(x, y)$  has a better completeness if it holds on more matches  $x$  of  $Q(x, y)$ , and is more discriminant if it is less likely to hold on more nodes from  $Q\bar{q}$ . In addition, BF-based  $\text{conf}(R, G)$  is better justified than conventional confidence. As verified in [26, 31], BF satisfies a set of principles for reasonable interestingness measures, including fixed under independence ( $\text{conf}(R, G) = 1$  if  $Q$  and  $q$  are statistically independent), fixed under incompatibility ( $\text{conf}(R, G) = 0$  if  $\text{supp}(R, G) = 0$ ), and monotonicity (increases monotonically with  $\text{supp}(R, G)$  when  $\text{supp}(\bar{q}, G)$ ,  $\text{supp}(Q, G)$  and  $\text{supp}(q, G)$  are fixed). Hence we adapt BF by incorporating LCWA and topological support.

**Example 7:** Consider GPAR  $R_2$  and  $Q_2(x, G)$  described in Example 6. Under the LCWA, match  $v_1$  accounts for “positive” for  $R_2$ , while  $v_2$  and  $v_3$  are “negative” and “unknown”, respectively. Indeed, assuming that  $G$  provides complete local information for  $v_2$ , then  $v_2$  is a counter-example to people who live in Ecuador but do not like *Shakira album*; in contrast,  $G$  knows nothing about what albums  $v_3$  likes.

One can see that  $\text{supp}(R_2, G) = 1$  (match  $v_1$ ),  $\text{supp}(\bar{q}, G) = 1$  (match  $v_2$ ),  $\text{supp}(Q\bar{q}, G) = 1$  (match  $v_2$ ), and  $\text{supp}(q, G) = 1$  (match  $v_1$ ). The BF-based confidence  $\text{conf}(R_2, G)$  is 1, larger than its conventional counterpart ( $\frac{1}{3}$ ) as the LCWA removes the impact of the unknown case  $v_3$ .  $\square$

symbols	notations
$Q(x, G)$	the set of distinct nodes that match $x$ in $Q(G)$
$R(x, y)$	GPAR $Q(x, y) \Rightarrow q(x, y)$ , represented as pattern $P_R$
$r(Q, x)$	the radius of $Q$ at node $x$
$N_r(v_x)$	the set of nodes within radius $r$ of $v_x$
$\text{supp}(Q, G)$	the number $\ Q(x, G)\ $ of distinct matches of $x$ in $Q(G)$
$\text{conf}(Q, G)$	$(\text{supp}(R, G) * \text{supp}(q, G)) / (\text{supp}(Q\bar{q}, G) * \text{supp}(q, G))$
$\Sigma(x, G, \eta)$	$\{v_x \mid v_x \in Q(x, G), Q \Rightarrow q \in \Sigma, \text{conf}(R, G) \geq \eta\}$

Table 1: Notations: graphs, queries and rules

There are other alternatives to define support and confidence for GPARs. (1) Following minimum image-based support [7],  $\text{supp}(R, G)$  can be defined as the the maximum number of matches for  $x$  in non-overlap matches (*i.e.*, no shared nodes and edges) of  $R$ . However, this excludes potential customers from matches that share even a single node (*e.g.*, only one of the three matches  $\text{cust}_1\text{-cust}_3$  of Fig. 2 is counted), and thus underestimates the significance. (2) Similar to PCA confidence [17],  $\text{conf}(R, G)$  can be computed as  $\frac{\text{supp}(R, G)}{\text{supp}(Q\bar{q}, G)}$  under LCWA. However, this only considers the “coverage” of  $R$  instead of its interestingness in terms of completeness and discriminant [26, 31] (see Section 6).

**Remark.** We identify the following two “trivial” cases when  $\text{conf}(R, G) = \infty$ : (1)  $\text{supp}(Q\bar{q}, G)$  is 0, which interprets  $R$  as a logic rule that holds on the entire  $G$ , *i.e.*, “if  $v$  is in  $Q(x, G)$  then  $v$  is a match in  $P_q(x, G)$  (hence  $P_R(x, G)$ ”); and (2)  $\text{supp}(q, G) = 0$ , which means that  $q(x, y)$  in  $R$  specifies no user in  $G$ ; hence  $R$  should be discarded as uninteresting case. These two cases can be easily detected and distinguished in the GPAR discovery process (see Section 4).

The notations of this paper are summarized in Table 1.

## 4. DIVERSIFIED RULE DISCOVERY

We now study how to discover useful GPARs.

### 4.1 The Diversified Mining Problem

We are interested in GPARs for a particular event  $q(x, y)$ . However, this often generates an excessive number of rules, which often pertain to the same or similar people [5, 44].

This motivates us to study a diversified mining problem, to discover GPARs that are *both* interesting and diverse.

**Objective function.** To formalize the problem, we first define a function  $\text{diff}(\cdot)$  to measure the difference of GPARs. Given two GPARs  $R_1$  and  $R_2$ ,  $\text{diff}(R_1, R_2)$  is defined as

$$\text{diff}(R_1, R_2) = 1 - \frac{|P_{R_1}(x, G) \cap P_{R_2}(x, G)|}{|P_{R_1}(x, G) \cup P_{R_2}(x, G)|}$$

in terms of the Jaccard distance of their match set (as social groups). Such diversification has been adopted to battle against over-concentration in social recommender systems when the items recommended are too “homogeneous” [5].

Given a set  $L_k$  of  $k$  GPARs that pertain to the same predicate  $q(x, y)$ , we define the objective function  $F(L_k)$  again by following the practice of social recommender systems [19]:

$$(1 - \lambda) \sum_{R_i \in S} \frac{\text{conf}(R_i)}{N} + \frac{2\lambda}{k-1} \sum_{R_i, R_j \in S, i < j} \text{diff}(R_i, R_j).$$

This, known as *max-sum diversification*, aims to strike a balance between interestingness (measured by revised Bayes Factor) and diversity (by distance  $\text{diff}(\cdot)$ ) with a parameter  $\lambda$  controlled by users. We consider nontrivial GPARs (Section 3) with  $\text{conf}(R, G) \in [0, \text{supp}(R, G) * \text{supp}(q, G)]$ , and normalize (1) the confidence metric with  $N = \text{supp}(q, G) *$

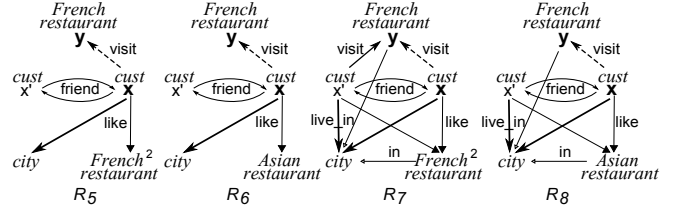


Figure 3: Diversified GPARs

$\text{supp}(q, G)$  (a constant for fixed  $q(x, y)$ ), and (2) the diversity metric with  $\frac{2\lambda}{k-1}$ , since there are  $\frac{k(k-1)}{2}$  numbers for the difference sum, while only  $k$  numbers for the confidence sum.

**Example 8:** Consider GPARs  $R_1$  of Fig. 1, and  $R_7$  and  $R_8$  shown in Fig. 3, all pertaining to visits( $x$ , French restaurant). Then in graph  $G_1$  (Fig. 2), (1)  $\text{supp}(q, G_1) = 5$  ( $\text{cust}_1\text{-cust}_4, \text{cust}_6$ ),  $\text{supp}(q, G_1) = 1$  ( $\text{cust}_5$ ); (2)  $R_1(x, G_1) = R_7(x, G_1) = \{\text{cust}_1, \text{cust}_2, \text{cust}_3\}$ ,  $R_8(x, G_1) = \{\text{cust}_6\}$ ; (3)  $\text{conf}(R_1, G_1) = \text{conf}(R_7, G_1) = 0.6$ ,  $\text{conf}(R_8, G_1) = 0.2$ ; and (4)  $\text{diff}(R_1, R_7) = 0$ ,  $\text{diff}(R_1, R_8) = \text{diff}(R_7, R_8) = 1$ .

For  $\lambda = 0.5$ , a top-2 diversified set of these GPARs is  $\{R_7, R_8\}$  with  $F(R_7, R_8) = 0.5 * \frac{0.8}{5} + 1 * 1 = 1.08$  (similarly for  $\{R_1, R_8\}$ ). Indeed,  $R_7$  and  $R_8$  find two disjoint customer groups sharing interests in French restaurant and Asian restaurant, respectively, with their friends.  $\square$

**Problem.** Based on the objective function, the *diversified GPAR mining problem* (DMP) is stated as follows.

- *Input:* A graph  $G$ , a predicate  $q(x, y)$ , a support bound  $\sigma$  and positive integers  $k$  and  $d$ .
- *Output:* A set  $L_k$  of  $k$  nontrivial GPARs pertaining to  $q(x, y)$  such that (a)  $F(L_k)$  is maximized; and (b) for each GPAR  $R \in L_k$ ,  $\text{supp}(R, G) \geq \sigma$  and  $r(P_R, x) \leq d$ .

DMP is a bi-criteria optimization problem to discover GPARs for a particular event  $q(x, y)$  with high support, bounded radius, and a balanced confidence and diversity. In practice, users can freely specify  $q(x, y)$  of interests, while proper parameters (*e.g.*, support, confidence, diversity) can be estimated from query logs or recommended by domain experts.

The problem is nontrivial. Consider its decision problem to decide whether there exists a set  $L_k$  of  $k$  GPARs with  $F(L_k) \geq B$  for a given bound  $B$ . One can show the following by reduction from the dispersion problem (cf. [19]).

**Proposition 1:** *The DMP decision problem is NP-hard.*  $\square$

### 4.2 Discovery Algorithm

One might want to follow a “discover and diversify” approach that (1) first finds all GPARs pertaining to  $q(x, y)$  by frequent graph pattern mining [35], and then (2) selects top- $k$  GPARs via result diversification [19]. However, this is costly: (a) an excessive number of GPARs are generated; and (b) for all GPARs  $R$  generated, it has to compute  $\text{conf}(R, G)$  and their pairwise distances, and moreover, pick a top- $k$  set based on  $F(\cdot)$ ; the latter is an intractable process itself.

One can do it more efficiently, with accuracy guarantees.

**Theorem 2:** *There exists a parallel algorithm for DMP that finds a set  $L_k$  of top- $k$  diversified GPARs such that (a)  $L_k$  has approximation ratio 2, and (b)  $L_k$  is discovered in  $d$  rounds by using  $n$  processors, and each round takes at most  $t(|G|/n, k, |\Sigma|)$  time, where  $\Sigma$  is the set of GPARs  $R(x, y)$  such that  $\text{supp}(R, G) \geq \sigma$  and  $r(P_R, x) \leq d$ .*  $\square$

Here  $t(|G|/n, k, |\Sigma|)$  is a function that takes  $|G|/n$ ,  $k$  and  $|\Sigma|$  as parameters, *rather than* the size  $|G|$  of the entire  $G$ .

As a proof, we give such an algorithm, denoted as **DMine** and shown in Fig. 4. It designates one processor as *coordinator*  $S_c$  and the rest as *workers*  $S_i$ . It works as follows.

(1) It divides  $G$  into  $n-1$  fragments  $(F_1, \dots, F_{n-1})$  such that (a) for each “candidate”  $v_x$  that satisfies the search condition on  $x$  in  $q(x, y)$ , its  $d$ -neighbor  $G_d(v_x)$ , *i.e.*, the subgraph of  $G$  induced by  $N_d(v_x)$ , is in some fragment; and (b) the fragments have roughly even size. These are possible since 98% of real-life patterns have radius 1, 1.8% have radius 2 [18], and the average node degree is 14.3 in social graphs [8]; thus  $G_d(v_x)$  is typically small compared with fragment size.

Fragment  $F_i$  is stored at worker  $S_i$ , for  $i \in [1, n-1]$ .

(2) **DMine** discovers GPARs *in parallel* by following bulk synchronous processing, in  $d$  rounds. The coordinator  $S_c$  maintains a list  $L_k$  of diversified top- $k$  GPARs, initially empty. In each round, (a)  $S_c$  posts a set  $M$  of GPARs to all workers, initially  $q(x, y)$  only; (b) each worker  $S_i$  generates GPARs *locally* at  $F_i$  in parallel, by extending those in  $M$  with new edges if possible; (c) these GPARs are collected and assembled by  $S_c$  in the barrier synchronization phase; moreover,  $S_c$  *incrementally* updates  $L_k$ : it filters GPARs that have low support or cannot make top- $k$  as *early as possible*, and prepares a set  $M$  of GPARs for expansion in the next round.

As opposed to the “discover and diversify” method, **DMine** (a) combines diversifying into discovering to *terminate* the expansion of non-promising rules *early*, rather than to conduct diversifying after discovering; and (b) it *incrementally* computes top- $k$  diversified matches, rather than recomputing the diversification function  $F()$  starting from scratch.

We next present the details of algorithm **DMine**.

**Auxiliary structures.** Algorithm **DMine** maintains the following: (a) at the coordinator  $S_c$ , a set  $L_k$  to store top  $k$  GPARs, and a set  $\Sigma$  to keep track of generated GPARs; and (b) at each worker  $S_i$ , a set  $C_i$  of candidates  $v_x$  for  $x$  at  $F_i$ .

**Messages.** In each round, coordinator  $S_c$  and workers  $S_i$  communicate via messages. (1) Each worker  $S_i$  generates a set  $M_i$  of messages. Each message is a triple  $\langle R, \text{conf}, \text{flag} \rangle$ , where (a)  $R$  is a GPAR generated at  $S_i$ , (b)  $\text{conf}$  includes, *e.g.*,  $\text{supp}(R(x, y), F_i)$  and  $\text{supp}(Q\bar{q}(x, y), F_i)$ , and (c) a Boolean  $\text{flag}$  to indicate whether  $R$  can be extended at  $S_i$ . (2) After receiving  $M_i$ ,  $S_c$  generates a set  $M$  of messages, which are GPARs to be extended in the next round.

**Algorithm.** **DMine** initializes  $L_k$  and  $\Sigma$  as empty, and  $M$  as  $\{q(x, y)\}$  (line 1). For  $r$  from 1 to  $d$ , it improves  $L_k$  by incorporating GPARs of radius  $r$  (lines 2-11), following a *levelwise* approach. In each round, it invokes **localMine** with  $M$  at all workers (line 4). Below we present the details.

**Parallel GPARs generation** (line 13). In the first round, procedure **localMine** receives  $q(x, y)$  from  $S_c$ , and computes the following: (a) three sets:  $C_i$ , nodes  $v_x$  that satisfy the search condition of  $x$  in discovered GPARs,  $P_q(x, F_i)$ , matches of  $x$  in  $q(x, y)$ , and  $\bar{q}(x, F_i)$ , nodes  $v$  in  $F_i$  that account for  $\text{supp}(\bar{q}, F_i)$  (Section 2.2); and (b)  $\text{supp}(q, F_i) = \|P_q(x, F_i)\|$ ,  $\text{supp}(\bar{q}, F_i) = \|P_{\bar{q}}(x, F_i)\|$ . Note that  $\text{supp}(q, F_i)$  and  $\text{supp}(\bar{q}, F_i)$  never change and hence are derived *once for all*. Each match  $v_x \in q(x, F_i)$  is referred to as a *center node*.

In round  $r$ , upon receiving  $M$  from  $S_c$ , **localMine** does the following. For each GPAR  $R(x, y) : Q(x, y) \Rightarrow q(x, y)$  in  $M$ ,

---

#### Algorithm **DMine**

*Input:* A graph  $G$ ,  $q(x, y)$ , bound  $\sigma$ , and positive integers  $k$  and  $d$ .

*Output:* A set  $L_k$  of top- $k$  diversified GPARs.

```

/* executed at coordinator */
1.  $L_k := \emptyset; \Sigma := \emptyset; r := 1; M := \{q(x, y)\};$ 
2. while  $r \leq d$  do
3.    $r := r + 1;$ 
4.   post  $M$  to all workers and invoke localMine ( $M$ ) in parallel;
5.   collect in  $\Delta E$  candidate GPARs in  $M_i$  from all workers;
6.   check automorphism and assemble confidence for these GPARs;
7.    $\Delta E$  includes  $R$  with  $\text{supp}(R, G) \geq \sigma; \Sigma := \Sigma \cup \Delta E; M := \emptyset;$ 
8.   for each GPAR  $R \in \Delta E$  do
9.     incDiv ( $L_k, R, \Sigma$ ); /* incrementally update  $L_k$ , prune  $\Sigma, \Delta E$  */
10.    if  $R$  is “extendable”
11.      then  $M := M \cup \{R\};$  /* next round */
12. return  $L_k;$ 

/* executed at each worker  $S_i$  in parallel, upon receiving  $M$  */
13.  $\Sigma_i := \text{localMine}$  ( $M$ );
14. construct message set  $M_i$  from  $\Sigma_i;$ 
15. send  $M_i$  to the coordinator;

```

---

**Figure 4: Algorithm **DMine****

and each center node  $v_x$ , it expands  $Q$  by including *at least* one *new edge* that is at hop  $r$  from  $v_x$ , for all such edges.

**Message construction** (lines 14–15). For each GPAR  $R(x, y) : Q(x, y) \Rightarrow q(x, y)$ , its *local confidence conf* is computed: (1)  $\text{supp}(R, F_i)$  and  $\text{supp}(Q, F_i)$  count nodes in  $P_q(x, F_i)$  and  $C_i$  that match  $x$  in  $R(x, y)$  and  $Q(x, y)$ , respectively; and (2)  $\text{supp}(Q\bar{q}, F_i) = \|Q(x, F_i) \cap P_{\bar{q}}(x, F_i)\|$ . Then  $\text{conf}$  contains  $\text{supp}(R, F_i)$ ,  $\text{supp}(Q\bar{q}, F_i)$ ,  $\text{supp}(q, F_i)$  and  $\text{supp}(\bar{q}(x, F_i))$ ; where  $\text{supp}(q, F_i)$  and  $\text{supp}(\bar{q}, F_i)$  values are from the first round. A Boolean  $\text{flag}$  is also set to indicate whether  $R$  can be extended by checking whether there exists a center node  $v_x$  that has edges at  $r+1$  hops from  $v_x$ . Message  $M_i$  includes  $\langle R, \text{conf}, \text{flag} \rangle$  for each  $R$ , and is sent to  $S_c$ .

**Message assembling** (lines 4-7). Upon receiving  $M_i$  from each  $S_i$ , coordinator  $S_c$  does the following. (1) It groups automorphic GPARs from all  $M_i$ . (2) For each group of  $m_i = \langle R, \text{conf}_i, \text{flag}_i \rangle$  that refers to the same (automorphic)  $R$ , it assembles  $\text{conf}(R)$  into a single  $m = \langle R, \text{conf}(R, G), \text{flag} \rangle$ , where (a)  $\text{conf}(R, G) = \frac{\sum \text{supp}(R, F_i) \sum \text{supp}(\bar{q}, F_i)}{\sum \text{supp}(Q\bar{q}, F_i) \sum \text{supp}(q, F_i)}$ ; and (b)  $\text{flag}$  is the disjunction of all  $\text{flag}_i$ , for  $i \in [1, n-1]$ . This suffices since by the partitioning of graph  $G$ , nodes accounted for local support in  $F_i$  are disjoint from those in  $F_j$  if  $i \neq j$ ; hence  $\text{conf}(R)$  can be directly assembled from local  $\text{conf}$  from  $F_i$ . Similarly,  $\text{supp}(R, G) = \sum_{i \in [1, n-1]} \text{supp}(R, F_i)$ . For each GPAR  $R$ , if  $\text{supp}(R, G) \geq \sigma$ , it is added to  $\Delta E$  and  $\Sigma$ .

**Incremental diversification** (lines 8-9). Next, **DMine** incrementally updates  $L_k$  by invoking procedure **incDiv**. It uses a max priority Queue of size  $\lceil \frac{k}{2} \rceil$ , where (1) each element in Queue is a pair of GPARs, and (2) all GPAR pairs in Queue are pairwise disjoint. In round  $r$ , starting from Queue of top- $k$  diversified GPARs with radius at most  $r-1$ , **DMine** improves Queue by incorporating pairs of GPARs from  $\Delta E$ , with radius  $r$ . (1) If Queue contains less than  $\lceil \frac{k}{2} \rceil$  GPARs pairs, **incDiv** iteratively selects two distinct GPARs  $R$  and  $R'$  from  $\Delta E$  that maximize a revised diversification function:

$$F'(R, R') = \frac{1 - \lambda}{N(k-1)} (\text{conf}(R) + \text{conf}(R')) + \frac{2\lambda}{k-1} \text{diff}(R, R').$$

and insert  $(R, R')$  into Queue, until  $|\text{Queue}| = \lceil \frac{k}{2} \rceil$ . It book-keeps each pair  $(R, R')$  and  $F'(R, R')$ . (2) If  $|\text{Queue}| = \lceil \frac{k}{2} \rceil$ , for each new GPAR  $R \in \Delta E$  (not in any pair of Queue)

and  $R' \in \Sigma$ , it incrementally computes and adds a new pair  $(R, R') \in \Delta E \times \Sigma$  that maximizes  $F'(R, R')$  to Queue. This ensures that a pair  $(R_1, R_2)$  with minimum  $F'(R_1, R_2)$  is replaced by  $(R, R')$ , if  $F'(R_1, R_2) < F'(R, R')$ .

After all GPAR pairs are processed, incDiv inserts  $R$  and  $R'$  into  $L_k$ , for each GPARs pairs  $(R, R') \in \text{Queue}$ .

*Message generation at  $S_c$*  (lines 10-11). DMine next selects promising GPARs for further parallel extension at the workers. These include  $R \in \Delta E$  that satisfy two conditions: (1)  $\text{supp}(R, G) \geq \sigma$ , since by the anti-monotonic property of support, if  $\text{supp}(R, G) < \sigma$ , then any extension of  $R$  cannot have support no less than  $\sigma$ ; and (2)  $R$  is “Extendable”, i.e.,  $\text{flag} = \text{true}$  in  $\langle R, \text{conf}, \text{flag} \rangle$ . It includes such  $R$  in  $M$ , and posts  $M$  to all workers in the next round.

**Example 9:** Suppose that graph  $G_1$  in Fig. 2 is distributed to two workers  $S_1$  and  $S_2$ , where  $S_1$  (resp.  $S_2$ ) contains sub-graphs induced by  $\text{cust}_1\text{-cust}_3$  (resp.  $\text{cust}_4\text{-cust}_6$ ) and their 2-hop neighborhoods in  $G_1$ . Let predicate  $q$  be  $\text{visits}(x, \text{French restaurant})$ ,  $\lambda=0.5$ ,  $d=2$  and  $k=2$ . We demonstrate algorithm DMine using example GPARs  $R_5\text{-}R_8$  (Fig. 3).

- (1) Coordinator  $S_c$  sends  $q$  to all workers, and computes  $\text{supp}(q, G_1) = 5$  ( $\text{cust}_1\text{-cust}_4, \text{cust}_6$ ),  $\text{supp}(\bar{q}, G_1) = 1$  ( $\text{cust}_5$ ).
- (2) In round 1,  $R_5$  (among others) is generated at  $S_1$  from 1-hop neighbors of  $\text{cust}_1\text{-cust}_3$ , which are matches in  $q(x, G_1)$  (Fig. 3). At  $S_2$ ,  $R_5$  and  $R_6$  are generated by expanding  $\text{cust}_4$  and  $\text{cust}_6$ . Local messages  $M_i$  from  $S_i$  include the following:

site	message	GPAR	$R(x, G_1)$	$Q\bar{q}(x, y)$	flag
$S_1$	$M_1$	$R_5$	$\text{cust}_1\text{-cust}_3$	$\emptyset$	T
		$R_5$	$\text{cust}_4$	$\text{cust}_5$	T
$S_2$	$M_2$	$R_6$	$\text{cust}_4, \text{cust}_6$	$\text{cust}_5$	T
		$R_5$	$\text{cust}_1\text{-cust}_4$	$\text{cust}_5$	T
$S_c$	$M$	$R_5$	$\text{cust}_1\text{-cust}_4$	$\text{cust}_5$	T
	$M$	$R_6$	$\text{cust}_4, \text{cust}_6$	$\text{cust}_5$	T

- (3) Coordinator  $S_c$  assembles  $M_1$  and  $M_2$ , and builds  $\Delta E$  including  $\{R_5, R_6\}$ . It computes  $\text{conf}(R_5) = 0.8$ ,  $\text{conf}(R_6) = 0.4$ ,  $\text{diff}(R_5, R_6) = 0.8$ . It updates  $L_k = \{R_5, R_6\}$ , with  $F'(R_5, R_6) = 0.5 * \frac{1.2}{5} + 1 * 0.8 = 0.92$ . It includes  $R_5$  and  $R_6$  in message  $M$  (the table above), and posts it to  $S_1$  and  $S_2$ .

- (4) In round 2,  $R_5$  is extended to  $R_7$  and  $R_1$  at  $S_1$  and  $S_2$ , and  $R_6$  to  $R_8$  at  $S_2$  (Fig. 3); the messages include:

site	message	GPAR	$R(x, G_1)$	$Q\bar{q}(x, y)$	flag
$S_1$	$M_1$	$R_7, R_1$	$\text{cust}_1\text{-cust}_3$	$\emptyset$	F
$S_2$	$M_2$	$R_7$	$\emptyset$	$\text{cust}_5$	F
		$R_8$	$\text{cust}_6$	$\text{cust}_5$	F

- (5) Given these, coordinator  $S_c$  assembles the messages and computes  $\text{conf}(R_7)=0.6$ ,  $\text{conf}(R_8)=0.2$  and  $\text{diff}(R_7, R_8)=1$ . DMine computes  $F'(R_7, R_8) = 0.5 * \frac{0.8}{5} + 1 * 1 = 1.08 > F'(R_5, R_6)=0.92$ . Hence, it replaces  $(R_5, R_6)$  with  $(R_7, R_8)$  and updates  $L_k$  to be  $\{R_7, R_8\}$ . As  $R_7$  and  $R_8$  are marked as “not extendable” at radius 2 (since  $d=2$ ), DMine returns  $\{R_7, R_8\}$  as top-2 diversified GPARs, in total 2 rounds.  $\square$

**Message reduction.** By maintaining additional information, DMine reduces the sizes of  $\Sigma$ ,  $M$  and  $M_i$ . The idea is to test whether an upper bound of marginal benefit for any GPAR pairs can improve the minimum  $F'$ -value of  $L_k$ .

In each round  $r$ , incDiv filters non-promising GPARs from  $\Sigma$  and  $\Delta E$  that cannot make top- $k$  even after new GPARs are discovered. It keeps track of (1) a value  $F'_m = \min F'(R_1, R_2)$  for all pairs  $(R_1, R_2)$  in  $L_k$ , (2) for each GPAR  $R_j$  in  $\Delta E$ , an estimated maximum confidence  $\text{Uconf}^+(R_j, G)$  for all the possible GPARs extended from  $R_j$ , and (3)  $\text{conf}(R, G)$  for

each GPAR  $R$  in  $\Sigma$ . Here  $\text{Uconf}^+(R_j, G)$  is estimated as follows. (a) Each  $S_i$  computes  $\text{Usupp}_i(R_j, F_i)$  as the number of matches of  $x$  in  $R_j(x, F_i)$  that connect to a center node in  $F_i$  at hop  $r + 1$  ( $r \leq d - 1$ ). (b) Then  $\text{Uconf}^+(R_j)$  is assembled at  $S_c$  as  $\frac{\sum \text{Usupp}_i(R_j, F_i) \text{supp}(\bar{q}, G)}{1 * \text{supp}(q, G)}$ . Denote the maximum  $\text{Uconf}^+(R_j, G)$  for  $R_j \in \Delta E$  as  $\max \text{Uconf}^+(\Delta E)$ , and the maximum  $\text{conf}(R, G)$  for  $R \in \Sigma$  as  $\max \text{conf}(\Sigma)$ . Then incDiv reduces  $\Sigma$  and  $M$  based on the reduction rules below.

**Lemma 3: [Reduction rules]:** (1) A GPAR  $R \in \Sigma$  cannot contribute to  $L_k$  if  $\frac{1-\lambda}{N^{(k-1)}}(\text{conf}(R, G) + \max \text{Uconf}^+(\Delta E)) + \frac{2\lambda}{k-1} \leq F'_m$ . (2) Extending a GPAR  $R_j \in \Delta E$  does not contribute to  $L_k$  if either (a)  $R_j$  is not extendable, or (b)  $\frac{1-\lambda}{N^{(k-1)}}(\text{Uconf}^+(R_j, G) + \max \text{conf}(\Sigma)) + \frac{2\lambda}{k-1} \leq F'_m$ .  $\square$

For the correctness of the rules, observe the following. (1) For each  $R \in \Sigma$ ,  $\text{conf}(R) + \max \text{Uconf}^+(\Delta E) + 1$  is an upper bound for its maximum possible increment to the  $F'$ -value of  $L_k$ ; similarly for any  $R_j$  from  $\Delta E$ . (2) If GPAR  $R$  does not contribute to  $L_k$ , then any GPARs extended from  $R$  do not contribute to  $L_k$ . Indeed, (a) upper bounds  $\text{Uconf}(R)$ ,  $\text{Usupp}_i(R)$ , and  $\text{Uconf}^+(R)$  are *anti-monotonic* with any  $R'$  expanded of  $R$ , and (b)  $\max \text{Uconf}^+(\Delta E)$  and  $\max \text{conf}(\Sigma)$  are *monotonically decreasing*, while  $F'_m$  is *monotonically increasing* with the increase of rounds. Hence  $R$  can be safely removed from  $\Sigma$ ,  $\Delta E$  or  $M_i$ . Note that the removal of GPARs from  $\Sigma$  benefit the reduction of  $\Delta E$  with smaller  $\max \text{conf}(\Sigma)$ , and vice versa. DMine repeatedly applies the rules until no GPARs can be reduced from  $\Sigma$  and  $\Delta E$ .

**Automorphism checking.** To reduce redundant GPARs, DMine checks whether GPARs in  $\Delta E$  are automorphic at coordinator  $S_c$  (line 6) and locally at each  $S_i$  (localMine). It is costly to conduct pairwise automorphism tests on all GPARs in  $\Delta E$ , since it is equivalent to graph isomorphism.

To reduce the cost, we use *bisimulation* [12]. A graph pattern  $P_{R_1}$  is *bisimilar* to  $P_{R_2}$  if there exists a binary relation  $O_b$  on nodes of  $P_{R_1}$  and  $P_{R_2}$  such that (a) for all nodes  $u_1$  in  $P_{R_1}$ , there exists a node  $u_2$  in  $P_{R_2}$  with the same label such that  $(u_1, u_2) \in O_b$ , and vice versa for all nodes in  $P_{R_2}$ ; and (b) for all edges  $(u_1, u'_1)$  in  $P_{R_1}$ , there exists an edge  $(u_2, u'_2)$  in  $P_{R_2}$  with the same label such that  $(u'_1, u'_2) \in O_b$ ; and vice versa for all edges in  $P_{R_2}$ . The connection between bisimulation and automorphism is stated as follows.

**Lemma 4:** If graph pattern  $P_{R_1}$  is not bisimilar to  $P_{R_2}$ , then  $R_1$  is not an automorphism of  $R_2$ .  $\square$

Hence, for a pair  $R_1$  and  $R_2$  of GPARs, DMine first checks whether  $P_{R_1}$  is bisimilar to  $P_{R_2}$ . It checks automorphism between  $R_1$  and  $R_2$  *only if* so. It takes  $O(|\Delta E|^2)$  time to check pairwise bisimilarity  $O_b$  for all GPARs in  $\Delta E$  [12]. Moreover,  $O_b$  can be incrementally maintained when new GPARs are added [40]. These allow us to use efficient (incremental) bisimulation tests instead of automorphism tests.

**Trivial GPARs.** DMine detects trivial GPARs  $R(x, y)$ :  $Q(x, y) \Rightarrow q(x, y)$  at  $S_c$  as follows: (1) if  $\text{supp}(q, G)$  is 0, it returns  $\emptyset$  to indicate that no interesting GPARs exist; and (2) if an extension leads to  $\text{supp}(Q\bar{q}) = 0$ , i.e., no match in  $Q(x, G)$  violates  $q(x, y)$ ,  $S_c$  removes  $R$  from  $\Delta E$  and  $\Sigma$ .

**Analyses.** DMine returns a set  $L_k$  of  $k$  diversified GPARs with approximation ratio 2 (line 12), for the following reasons. (1) Parallel generation of GPARs finds all candidate GPARs within radius  $d$ . This is due to the *data locality* of



subgraph isomorphism: for any node  $v_x$  in  $G$ ,  $v_x \in P_R(x, G)$  iff  $v_x \in P_R(x, G_d(v_x))$  for any GPAR  $R$  of radius at most  $d$  at  $x$ . That is, we can decide whether  $v_x$  matches  $x$  via  $R$  by checking the  $d$ -neighbor of  $v_x$  locally at a fragment  $F_i$ . (2) Procedure `incDiv` updates  $L_k$  following the greedy strategy of [19], with approximation ratio 2. This is verified by approximation-preserving reduction to the max-sum dispersion problem, which maximizes the sum of pairwise distance for a set of data points and has approximation ratio 2 [19]. The reduction maps each GPAR to a data point, and sets the distance between two GPARs  $R$  and  $R'$  as  $F'(R, R')$ .

For time complexity, observe that in each round, the cost consists of (a) local parallel generation time  $T_1$  of candidate GPARs, determined by  $|F_i|$ ,  $M$  and  $M_i$ ; and (b) total assembling and incremental maintenance cost  $T_2$  of  $L_k$  at  $S_c$ , dominated by  $|\Sigma|$ ,  $k$  and  $|M_i|$ . The cost of message reduction (by applying Lemma 3) takes in total  $O(d|\Sigma|)$  time, where in each round, it takes a linear scan of  $\Delta E$  and  $\Sigma$  to identify redundant GPARs. Note that  $\sum_{i \in [1, n-1]} |M_i| \leq |\Delta E| \leq |\Sigma|$ ,  $|M| \leq |\Sigma|$ , and  $|F_i|$  is roughly  $|G|/n$  by our partitioning strategy. Hence  $T_1$  and  $T_2$  are functions of  $|G|/n$ ,  $k$  and  $|\Sigma|$ . This completes the proof of Theorem 2.

**Remarks.** Algorithm `DMine` can be easily adapted to the following two cases. (1) When a set of predicates instead of a single  $q(x, y)$  is given, it groups the predicates and iteratively mines GPARs for each distinct  $q(x, y)$ . (2) When no specific  $q(x, y)$  is given, it first collects a set of predicates of interests (e.g., most frequent edges, or with user specified label  $q$ ), and then mines GPARs for the predicate set as in (1).

## 5. IDENTIFYING CUSTOMERS

We study how to identify potential customers with GPARs.

### 5.1 The Entity Identification Problem

Consider a set  $\Sigma$  of GPARs *pertaining to the same*  $q(x, y)$ , i.e., their consequents are the same event  $q(x, y)$ . We define *the set of entities* identified by  $\Sigma$  in a (social) graph  $G$  with confidence  $\eta$ , denoted by  $\Sigma(x, G, \eta)$ , as follows:

$$\{v_x \mid v_x \in Q(x, G), Q(x, y) \Rightarrow q(x, y) \in \Sigma, \text{conf}(R, G) \geq \eta\}$$

**Problem.** We study the *entity identification problem* (EIP):

- *Input:* A set  $\Sigma$  of GPARs pertaining to the same  $q(x, y)$ , a confidence bound  $\eta > 0$ , and a graph  $G$ .
- *Output:*  $\Sigma(x, G, \eta)$ .

It is to find potential customers  $x$  of  $y$  in  $G$  identified by at least one GPAR in  $\Sigma$ , with confidence of at least  $\eta$ .

**Intractability.** The decision problem of EIP is to determine, given  $\Sigma$ ,  $G$  and  $\eta$ , whether  $\Sigma(x, G, \eta) \neq \emptyset$ . It is equivalent to decide whether there exists a GPAR  $R \in \Sigma$  such that  $\text{conf}(R, G) \geq \eta$ . The problem is nontrivial, as it embeds the subgraph isomorphism problem, which is NP-hard.

**Proposition 5:** *The decision problem for EIP is NP-hard, even when  $\Sigma$  consists of a single GPAR.*  $\square$

A naive way to compute  $\Sigma(x, G, \eta)$  is as follows. For each  $R(x, y) : Q(x, y) \Rightarrow q(x, y)$  in  $\Sigma$ , (a) enumerate all matches of  $Q\bar{q}$  and  $P_R$  in  $G$  by using an algorithm for subgraph isomorphism, e.g., VF2 [10]; (b) compute  $\text{supp}(q, G)$  and  $\text{supp}(\bar{q}, G)$  once in  $G$ ; then based on the findings, (c) identify those  $R$  with  $\text{conf}(R, G) \geq \eta$ , and return matches of  $x$  by these GPARs. This is cost-prohibitive (e.g., takes

$O(|G|!|G|^{|\Sigma|})$  time using VF2 [10]) in real-life social graphs  $G$ , which often have billions of nodes and edges [21]. It is thus not practical to simply apply graph pattern matching algorithms to EIP over large  $G$ .

One might think that parallelization would solve the problem. However, parallelization is *not always effective*.

**Parallel scalability.** To characterize the effectiveness of parallelization, we formalize parallel scalability following [30]. Consider a problem  $A$  posed on a graph  $G$ . We denote by  $t(|A|, |G|)$  the worst-case running time of a *sequential algorithm* for solving  $A$  on  $G$ . For a parallel algorithm, we denote by  $T(|A|, |G|, n)$  the time taken by the algorithm for solving  $A$  on  $G$  by using  $n$  processors. Here we assume  $n \ll |G|$ , i.e., the number of processors does not exceed the size of the graph; this typically holds in practice since  $G$  has billions of nodes and edges, much larger than  $n$ .

We say that the algorithm is *parallel scalable* if

$$T(|A|, |G|, n) = O(t(|A|, |G|)/n) + (n|A|)^{O(1)}.$$

That is, the parallel algorithm achieves a polynomial reduction in sequential running time, plus a “bookkeeping” cost  $O((n|A|)^l)$  for a constant  $l$  that is *independent* of  $|G|$ .

Obviously, if the algorithm is parallel scalable, then for a given  $G$ , it *guarantees* that the more processors are used, the less time it takes to solve  $A$  on  $G$ . It allows us to process big graphs by adding processors when needed. If an algorithm is not parallel scalable, we may not get reasonable response time *no matter how many* processors are used.

We say that problem  $A$  is *parallel scalable* if there exists a parallel scalable algorithm for it. Unfortunately, parallel scalability is *not* warranted for all problems, e.g., it is beyond reach for graph simulation [15]. The good news is as follows.

**Theorem 6:** *EIP is parallel scalable.*  $\square$

As a proof, we outline a parallel algorithm for EIP, denoted by `Matchc`. Given  $\Sigma$ ,  $G = (V, E, L)$ ,  $\eta$  and a positive integer  $n$ , it computes  $\Sigma(x, G, \eta)$  by using  $n$  processors. Note that `Matchc` is *exact*: it computes precisely  $\Sigma(x, G, \eta)$ .

To present `Matchc`, we use the following notations. (a) We use  $d$  to denote *the maximum radius* of  $R(x, y)$  at node  $x$ , for all GPARs  $R$  in  $\Sigma$ . (b) For a node  $v_x \in V$ ,  $G_d(v_x)$  is *the  $d$ -neighbor* of  $v_x$  in  $G$  (see Section 4.2). (c) We denote by  $L$  the set of all *candidates*  $v_x$  of  $x$ , i.e., nodes in  $G$  that satisfy the search condition of  $x$  in  $q(x, y)$ .

**Algorithm.** `Matchc` capitalizes on the data locality of subgraph isomorphism (see Section 4.2). It works as follows.

(1) *Partitioning.* It divides  $G$  into  $n$  fragments  $\mathcal{F} = (F_1, \dots, F_n)$  in the same way as algorithm `DMine` (Section 4.2), such that  $F_i$ 's have roughly even size, and  $G_d(v_x)$  is contained in one  $F_i$  for each  $v_x \in L$ . This is done in parallel. In particular,  $G_d(v_x)$  can be constructed in parallel by revising BFS (breadth-first search), within  $d$  hops from  $v_x$ . Each fragment  $F_i$  is assigned to a processor  $S_i$  for  $i \in [1, n]$ .

(2) *Matching.* All processors  $S_i$  compute local matches in  $F_i$  *in parallel*. For each candidate  $v_x \in L$  that resides in  $F_i$ , and for each GPAR  $R(x, y) : Q(x, y) \Rightarrow q(x, y)$  in  $\Sigma$ ,  $S_i$  checks whether  $v_x$  is in  $P_R(x, G_d(v_x))$ ,  $P_Q(x, G_d(v_x))$  and  $P_q(x, G_d(v_x))$ , and whether  $v_x$  has an outlink labeled  $q$ .

(3) *Assembling.* Compute  $\text{conf}(R, G)$  for each  $R$  in  $\Sigma$  by assembling the partial results of (2) above. This is also done *in parallel*: first partition  $L$  into  $n$  fragments; then each processor operates on a fragment and computes partial sup-

port. These partial results are then collected to compute  $\text{conf}(R, G)$ . Finally, output those  $v_x$  when there exists a GPAR  $R$  such that  $v_x \in P_R(x, G)$  and  $\text{conf}(R, G) \geq \eta$ .

*Analysis.* To show that  $\text{Match}_c$  is parallel scalable, observe the following. (1) Step 1 is in  $O(|L||G_d^m|/n)$  time, since BFS is in  $O(|G_d^m|)$  time, where  $G_d^m$  is the largest  $d$ -neighbor for all  $v_x \in L$ . (2) Step 2 takes  $O(t(|G_d^m|, |\Sigma|)|L|/n)$  time, where  $t(|G_d^m|, |\Sigma|)$  is the worst-case sequential time for processing a candidate  $v_x$ . (3) Step 3 takes  $O(|L||\Sigma|/n)$  time. (4) By  $|L| \leq |V|$ , steps 1 and 2 take much less time than  $t(|G|, |\Sigma|)$ , since  $t(\cdot)$  is an exponential function by Proposition 5, unless  $P = \text{NP}$ . (5) In practice,  $t(|G_d^m|, |\Sigma|)|L| \ll t(|G|, |\Sigma|)$  since  $t(\cdot)$  is exponential and  $G_d^m$  is much smaller than  $G$ . Indeed, (a) in the real world, graph patterns in GPARs are typically small, and hence so is the radius  $d$ ; as argued in Section 4.2,  $G_d(v_x)$  is thus often small. Putting these together, we have that the parallel cost  $T(|G|, |\Sigma|, n) < O(t(|G|, |\Sigma|)/n)$ , and better still, the larger  $n$  is, the smaller  $T(|G|, |\Sigma|, n)$  is.

**Remark.** Algorithm DMine (Section 4.2) takes  $t(|A|/n, k)$  time and is parallel scalable if the problem size  $|A|$  is measured as  $|G|+|Q|+|\Sigma|$  [29]. Indeed, if one wants all candidate GPARs  $R$  with  $\text{supp}(R, G) \geq \sigma$ , then  $|\Sigma|$  is the size of the output, and  $|\Sigma|$  is not large (due to small  $d$  and large  $\sigma$ ).

## 5.2 Optimization Strategies

Algorithm  $\text{Match}_c$  just aims to show the parallel scalability of EIP. Its cost is dominated by step 2 for matching via subgraph isomorphism. To reduce the cost, we develop algorithm  $\text{Match}$  that improves  $\text{Match}_c$  by incorporating the following optimization techniques. To simplify the discussion, we start with a single GPAR  $R(x, y) : Q(x, y) \Rightarrow q(x, y)$ .

**Early termination.** For each candidate  $v_x \in L$  that resides in fragment  $F_i$ , we check whether *there exists* a match  $G_x$  of  $P_R$  in which  $v_x$  matches  $x$ . As soon as one  $G_x$  is verified a match of  $P_R$ , we include  $v_x$  in  $P_R(x, F_i)$ , *without* enumerating *all* matches of  $P_R$  at  $v_x$ . This is done locally at  $F_i$ : by our partitioning strategy,  $G_d(v_x)$  is contained in  $F_i$ .

**Guided search.** To identify  $G_x$  at  $v_x$ ,  $\text{Match}$  starts with pair  $(x, v_x)$  as a partial match  $m$ , and iteratively grows  $m$  with new pairs  $(u, v)$  for  $u \in P_R$  and  $v \in G_d(v_x)$  until a complete match is identified, *i.e.*,  $m$  covers all the nodes in  $P_R$ . A complete  $m$  induces a subgraph  $G_x$ . It is in PTIME to verify whether  $m$  is an isomorphism from  $P_R$  to  $G_x$ .

To grow  $m$ ,  $\text{Match}$  performs *guided search* based on  $k$ -hop neighborhood sketch. For each node  $v$  in  $G$ , a  $k$ -hop sketch  $K(v)$  is a list  $\{(1, D_1), \dots, (k, D_k)\}$ , where  $D_i$  denotes the distribution of the node labels and their frequency at  $i$  hop of  $v$ . Given a pair  $(u, v)$  newly added to  $m$  and a pattern edge  $(u, u')$  in  $Q$ ,  $\text{Match}$  picks “the best neighbor”  $v'$  of  $v$  such that the pair  $(u', v')$  has a high possibility to make a match. This is decided by assigning a score  $f(u', v')$  as  $\sum_{i \in [1, k]} (D_i - D'_i)$ , where  $D'_i \in K(u')$ ,  $D_i \in K(v')$ , and  $D_i - D'_i$  is the total frequency difference for each label in  $D_i$ . Indeed, (1)  $v'$  does not match  $u'$  if for some  $i$ ,  $D_i - D'_i < 0$ ; and (2) the larger the difference is, the more likely  $v'$  matches  $u'$ . If  $(u', v')$  does not lead to a complete  $m$ ,  $\text{Match}$  backtracks and picks  $v''$  with the next best score  $r(u', v'')$ .

**Example 10:** Consider GPAR  $R_1$  of Fig. 1. For its designated node  $x$ , the 2-hop neighborhood sketch  $L_2(x)$  in  $P_{R_1}$  contains pair  $(1, D_1 = \{(\text{city}, 1), (\text{cust}, 1), (\text{French Restaurant}, 4)\})$  and  $(2, D_2 = \{(\text{city}, 1), (\text{cust}, 1), (\text{French Restaurant}, 4)\})$ .

Given  $R_1$  and  $G_1$  of Fig. 2,  $\text{Match}$  identifies  $P_{R_1}(x, G_1)$  as follows. (1) It finds  $P_{q_1}(x, G) = \{\text{cust}_1 - \text{cust}_4, \text{cust}_6\}$ , while  $\text{cust}_5$  accounts for  $\text{supp}(\bar{q}_1, G_1)$ . (2) It computes  $P_{R_1}(x, G_1)$  by verifying candidates  $v_x$  from  $P_q(x, G_1)$ , and calculates  $f(x, v_x)$  in  $G_1$ , *e.g.*,  $L_2(\text{cust}_2) = \{(1, D_1 = \{(\text{city}, 1), (\text{cust}, 2), (\text{French Restaurant}, 8)\}), (2, D_2 = \{(\text{city}, 1), (\text{cust}, 2), (\text{French Restaurant}, 8)\})\}$ . Hence  $f(x, \text{cust}_2) = 5 + 5 = 10$ .  $\text{Match}$  then ranks candidates  $\text{cust}_2, \text{cust}_1, \text{cust}_3, \text{cust}_4$ , where  $\text{cust}_6$  is filtered due to mismatched sketches. (2) At  $\text{cust}_2$ ,  $\text{Match}$  starts from  $(x, \text{cust}_2)$ , and extends to  $(x', \text{cust}_3)$  since  $f(x', \text{cust}_3)$  is the highest. It continues to add pairs  $(\text{city}, \text{NewYork}), (\text{French Restaurant}, \text{LeBernardin})$  and three pairs for French Restaurant<sup>3</sup>. This completes the match, and  $\text{cust}_2$  is verified a match. (3) Similarly,  $\text{Match}$  verifies  $\text{cust}_1$  and  $\text{cust}_3$ , and finds  $P_{R_1}(x, G_1) = \{\text{cust}_1, \text{cust}_2, \text{cust}_3\}$ .

Given  $P_{R_1}(x, G_1)$ ,  $\text{Match}$  only needs to verify  $\text{cust}_5$  for  $Q_1$  in  $R_1$ ; it finds  $Q_1(x, G_1) = P_{R_1}(x, G_1) \cup \{\text{cust}_5\}$ . It also finds  $\text{supp}(q, G_1) = 5$  ( $\text{cust}_1 - \text{cust}_4, \text{cust}_6$ ),  $\text{supp}(\bar{q}, G_1) = 1$  ( $\text{cust}_5$ ), and computes  $\text{conf}(R_1) = \frac{3 \cdot 1}{1 \cdot 5} = 0.6$ .  $\square$

**Algorithm Match.** Given a set  $\Sigma$  of GPARs,  $\text{Match}$  revises step (2) of  $\text{Match}_c$  by checking whether  $v_x$  matches  $x$  via guided search and early termination; it reduces redundant computation for multiple GPARs by extracting common sub-patterns of GPARs in  $\Sigma$  [32]. It remains parallel scalable following the same complexity analysis for  $\text{Match}_c$ .

## 6. EXPERIMENTAL STUDY

Using real-life and synthetic graphs, we conducted three sets of experiments to evaluate (1) the scalability of algorithm DMine, (2) the effectiveness of DMine for discovering interesting GPARs, and (3) the scalability of algorithm  $\text{Match}$  for identifying potential customers in large graphs.

**Experimental setting.** We used two real-life graphs: (a) *Pokec* [3], a social network with 1.63 million nodes of 269 different types, and 30.6 million edges of 11 types, such as *follow*, *like*; and (b) *Google+* [20], a social graph with 4 million entities of 5 types and 53.5 million links of 5 types.

We also designed a generator for synthetic graphs  $G = (V, E, L)$ , controlled by the numbers of nodes  $|V|$  (up to 50 million) and edges  $|E|$  (up to 100 million), with  $L$  drawn from an alphabet  $\mathcal{L}$  of 100 labels.

*Pattern generator.* To evaluate  $\text{Match}$ , we generated GPARs  $R$  controlled by the numbers  $|V_p|$  and  $|E_p|$  of nodes and edges in  $P_R$ , respectively. (1) We found 48 meaningful GPARs on each of *Pokec* and *Google+*, with labels drawn from their data (domain, social groups). (2) For synthetic graphs, we also generated 24 GPARs with labels drawn from  $\mathcal{L}$ . We denote the size of a GPAR  $R$  as  $|R| = (|V_p|, |E_p|)$ .

*Algorithms.* We implemented the following, all in Java. (1) Algorithm DMine, compared with (a) DMine<sub>no</sub>, its counterpart without optimization (incremental, reductions and bisimilarity checking), and (b) GRAMI [13], an open source frequent subgraph mining tool [1]. Since GRAMI uses a *single machine* [1], we only compared the interestingness of patterns found by GRAMI with GPARs discovered by DMine. (2) Algorithm  $\text{Match}$ , compared with (a)  $\text{Match}_c$  (Section 5.1), (b) disVF2, a parallel implementation of VF2 for EIP, and (c)  $\text{Match}_s$ ,  $\text{Match}$  by using the method of [38] instead of VF2.

*Fragmentation and distribution.* We revised the algorithm of [36] to evenly partition graph  $G$  into  $n$  fragments (see Section 4.2). We find that the gap between maximum and

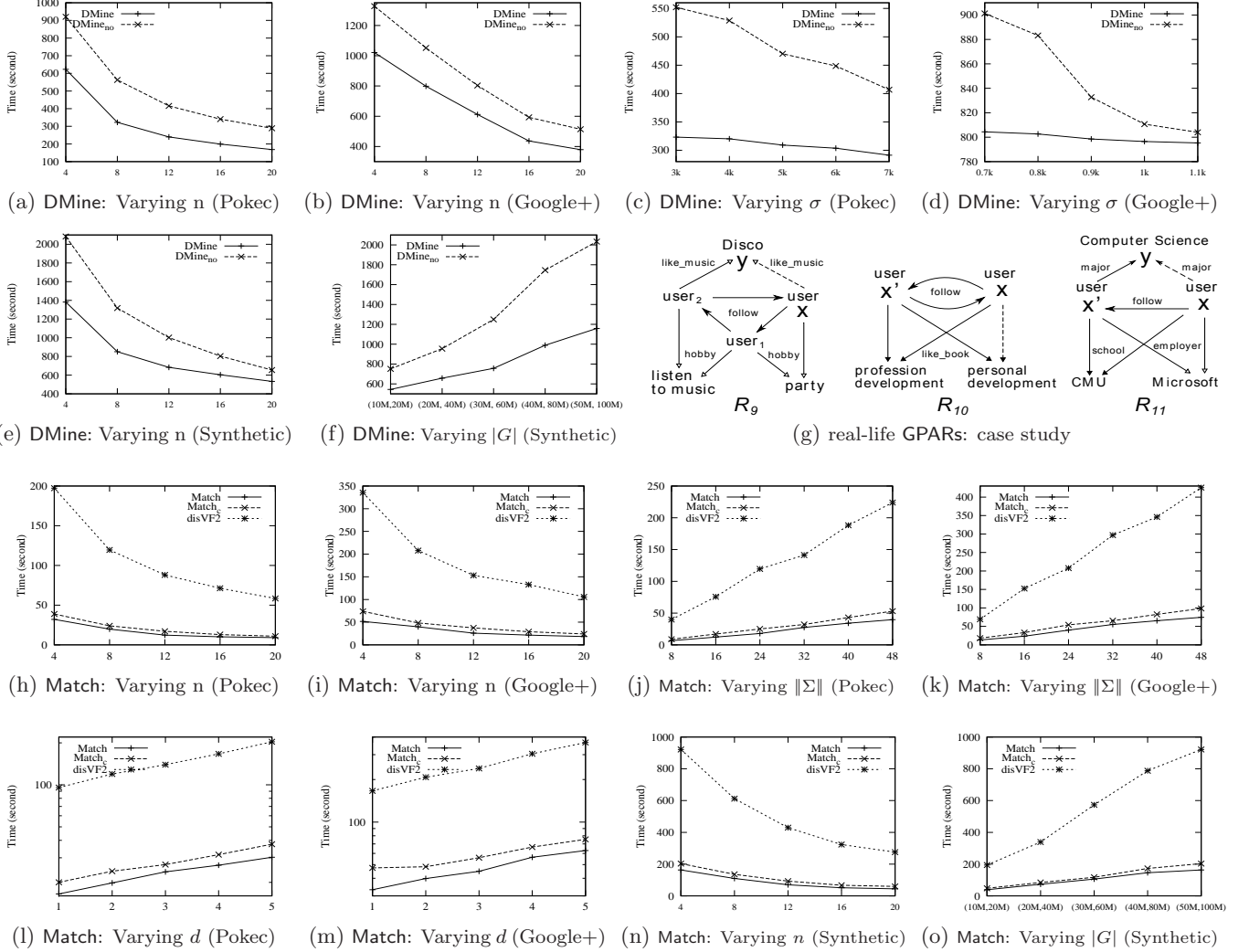


Figure 5: Performance evaluation

minimum time spent on different fragments by DMine is at most 14.4% (resp. 8.8%) of the time for processing fragments of *Pokerc* (resp. *Google+*), and at most 6.0% (resp. 5.2%) of the time for identifying matches by Match. These indicate that the impact of skew from partitioning is fairly small.

We deployed the algorithms and  $n$  fragments on  $n \in [4, 20]$  Amazon EC2 M3 instances, each has 2.6GHz 2vcpu with 7.5G memory, and 32GB SSD storage. Each experiment was run 5 times and the average is reported here.

**Experimental results.** We next report our findings. We fixed parameter  $\lambda = 0.5$  for diversification in Exp-1.

**Exp-1: Scalability of DMine.** We first evaluated the scalability of DMine vs. DMine<sub>no</sub>. We used  $k = 10$ , and found that different  $k$  had little impact. We found that GPARs mined in real-life graphs with infrequent edge labels usually denote unrelated facts. Hence we used 20 most frequent edge patterns, *i.e.*, graph patterns consisting of a single edge (with both node and edge labels), to grow GPARs in *Pokerc*. We used all 5 types of edges in *Google+*.

*Varying n.* Fixing radius  $d = 2$  and support  $\sigma = 5000$  (500 for *Google+*), we varied the number  $n$  of processors from 4 to 20. The algorithms generated up to 300 patterns to

be verified. As shown in Fig. 5(a) (resp. Fig. 5(b)), (1) DMine scales well with the increase of processors: the improvement is 3.7 (resp. 2.69) times when  $n$  increases from 4 to 20; and (2) it is on average 1.67 (1.37) times faster than DMine<sub>no</sub>; this verifies that our optimization strategies effectively reduce confidence checking time, which is a major bottleneck in DMine<sub>no</sub>. With 20 processors, DMine takes 168.3 (resp. 379) seconds on *Pokerc* (resp. *Google+*).

*Varying  $\sigma$ .* Fixing  $d = 2$  and  $n = 4$ , we varied  $\sigma$  from 3K to 7K (resp. 700 to 1100) on *Pokerc* (resp. *Google+*). Figures 5(c) and 5(d) tell us the following. (1) All algorithms takes longer with smaller  $\sigma$ , because more patterns satisfy the support constraint and are checked. (2) DMine outperforms DMine<sub>no</sub> in all cases. Moreover, it is less sensitive to the increment of  $\sigma$ . This is because DMine checks much less patterns than DMine<sub>no</sub> due to its filtering strategy.

Using large synthetic graphs of size up to (50M, 100M), we evaluated the impact of  $n$ , the size of  $G$  and radius  $d$ .

*Varying n (Synthetic).* Fixing  $|G| = (10M, 20M)$ ,  $d = 2$  and  $\sigma = 100$ , we varied  $n$  from 4 to 20. The results (Fig. 5(e)) are consistent with Figures 5(a) and 5(b). DMine takes 533.2 seconds over synthetic  $G$  with 20 processors.

Varying  $|G|$  (Synthetic). Fixing  $n = 16$ ,  $d = 2$  and  $\sigma = 100$ , we varied  $|G|$  from  $(10M, 20M)$  to  $(50M, 100M)$ . As shown in Fig. 5(f), (1) both algorithms take longer on larger graphs; and (2) DMine outperforms DMine<sub>no</sub> by 1.76 times, verifying the effectiveness of our optimization methods.

Varying  $d$ . Fixing  $n = 16$ ,  $|G| = (50M, 100M)$  and  $\sigma = 100$ , we varied  $d$  from 1 to 3. We find that DMine and DMine<sub>no</sub> take longer over larger  $d$  (not shown), as expected. However, DMine is less sensitive to  $d$ , since its optimization strategies reduces GPAR candidates and checking time.

**Exp-2: Effectiveness of DMine.** We manually examined GPARs discovered by DMine from *Pokec* and *Google+*. Three GPARs are shown in Fig. 5(g), with support above 100:

(1)  $R_9$  (*Pokec*): if  $x$  follows  $user_1$ ,  $user_1$  follows  $user_2$ ,  $user_2$  follows  $x$ ,  $user_1$  and  $user_2$  share the hobby to listen to music,  $x$  and  $user_1$  share the hobby of party, and if  $user_2$  likes Disco music, then  $x$  likes Disco. This suggests regularity between types of music people like and their friends’ hobbies.

(2)  $R_{10}$  (*Pokec*): if  $x$  and  $x'$  follow each other and both like books of profession development, and if  $x'$  likes books about personal development, then so does  $x$ . This suggests that potential customers  $x$  favor books liked by their friends.

(3)  $R_{11}$  (*Google+*): if  $x$  follows  $x'$ , both  $x$  and  $x'$  went to CMU, both  $x$  and  $x'$  are employees of Microsoft, and if  $x'$  was majored in CS, then  $x$  was also likely majored in CS. This indicates a social pattern between Microsoft employees and CMU computer science students.

We also found that most patterns mined by GRAMI are cycles of users. These patterns, although quite frequent, reveals little insight about entity associations.

GPARs with different metrics. We also evaluated different confidence metrics for GPARs (Section 3). Given a GPAR  $R$ , we define its (1) PCA confidence [17]  $PCAconf(R, G)$  as  $\frac{\text{supp}(R, G)}{\text{supp}(Q_{\bar{q}}, G)}$ , and (2) image-based  $lconf(R, G)$  by replacing  $\text{supp}(\cdot, G)$  in  $\text{conf}(R, G)$  with the image-based support [7].

We evaluated prediction precision of these metrics for social networks following [17]. We partitioned *Pokec* into two fragments  $F_1$  (as training data) and  $F_2$  (for cross validation), and selected 5 predicates as in Exp-1 from  $F_1$ . We set  $\lambda = 0$  to focus on the relevance of GPARs, and mined top 10, 30 and 60 GPARs from  $F_1$  with highest  $\text{conf}$ ,  $PCAconf$  and  $lconf$ , respectively. We evaluate the precision for each GPAR  $R$  as  $\text{prec}(R) = \frac{\text{supp}(R, F_2)}{\text{supp}(Q, F_2)}$ , indicating correctly predicted customers in  $F_2$ , constrained by GPARs mined from  $F_1$ .

	top 10	top 30	top 60
<b>PCAconf</b>	0.276	0.280	0.277
<b>lconf</b>	0.267	0.273	0.265
<b>conf</b>	0.423	0.388	0.381

As shown in the table above, (1) DMine is able to identify GPARs that “predict” predicates with average precision up to 42.3%, and (2) GPARs ranked by our  $\text{conf}$  metric provides better prediction precision than  $PCAconf$  and  $lconf$ .

**Exp-3: Scalability of Match.** Finally, we evaluated (1) the scalability of Match with the number  $n$  of processors, and the impacts of (2) the number  $\|\Sigma\|$  of GPARs in  $\Sigma$ , (3) the maximum radius  $d$  of GPARs in  $\Sigma$ , and (4) the size  $|G|$  of graphs. We started with real-life graphs and fixed  $\eta = 1.5$ .

Varying  $n$ . Fixing  $\|\Sigma\| = 24$ ,  $|R| = (5, 8)$  and  $d = 2$ , we varied  $n$  from 4 to 20. Figures 5(h) and 5(i) report the results on *Pokec* and *Google+*, respectively, which tell us the following.

(1) Match, Match<sub>c</sub> and Match<sub>s</sub> allow a high degree of parallelism. For instance, Match is 3.52 (resp. 3.54) times faster when  $n$  increases from 4 to 20 on *Pokec* (resp. *Google+*). This is consistent with Theorem 6. The algorithms are efficient. In particular, Match takes 9.1 seconds on social graph *Pokec* with 20 processors, and it scales better than Match<sub>c</sub> and disVF2. We find that Match<sub>s</sub> and Match have very similar performance, and thus we report Match only.

(2) Our optimization strategies are effective. (a) Compared to disVF2, Match<sub>c</sub> and Match are 4.79 and 6.24 times faster on average, since for each GPAR  $R : Q \Rightarrow q$ , disVF2 invokes two isomorphic checks at each candidate  $v_x$  (one for  $P_R$  and one for  $Q_{\bar{q}}$ ) vs. one by Match<sub>c</sub> and Match; this justifies the need for new algorithms for EIP instead of applying conventional pattern matching algorithms. (b) Match is 1.2 and 1.35 times faster than Match<sub>c</sub> on *Pokec* and *Google+*, respectively, demonstrating the effectiveness of early termination and guided search, without enumerating all matches.

Varying  $\|\Sigma\|$ . Fixing  $n = 8$  and  $d = 2$ , we varied  $\|\Sigma\|$  from 8 to 48. As shown in Figures 5(j) and 5(k), (1) all algorithms take longer time with larger  $\|\Sigma\|$ , as expected; (2) Match is less sensitive to  $\|\Sigma\|$  than Match<sub>c</sub> and disVF2; (3) the improvement of Match over the others is greater on larger  $\Sigma$ . These are because optimization by early termination and guided search works better for more GPARs in  $\Sigma$ .

Varying  $d$ . Fixing  $n = 8$  and  $\|\Sigma\| = 20$ , we varied  $d$  from 1 to 5. As shown in Figures 5(l) and 5(m) (in logarithmic scale), all algorithms take longer time with larger  $d$ , since more nodes in the  $d$ -neighbors of candidates need to be visited. Nonetheless, Match and Match<sub>c</sub> are less sensitive to  $d$  than disVF2 due to their optimization techniques (data locality leveraged by Match<sub>c</sub>, and early termination by Match).

Synthetic graphs. Using larger synthetic graphs, we evaluated the impact of  $n$ . Fixing  $|G| = (50M, 100M)$ ,  $d = 2$ ,  $\eta = 1.5$  and  $\|\Sigma\| = 24$ , we varied  $n$  from 4 to 20. As shown in Fig. 5(n), the result is consistent with its counterparts on real-life graphs (Figures. 5(h) and 5(i)). The improvement for Match is 3.65 times when  $n$  increases from 4 to 20.

Fixing  $n = 4$ ,  $\|\Sigma\| = 24$ ,  $\eta = 1.5$  and  $d = 2$ , we varied  $|G|$  from  $(10M, 20M)$  to  $(50M, 100M)$ . As shown in Fig. 5(o), (1) all the algorithms take longer on larger  $|G|$ , as expected; (2) Match performs the best, and is less sensitive to  $|G|$  than the others; and (3) despite Proposition 5, Match is reasonably efficient: when  $|G| = (50M, 100M)$ , Match takes 163 seconds with 4 processors, while disVF2 takes 922 seconds.

**Summary.** We find the following. (1) It is not very expensive to mine diversified top- $k$  GPARs in large social networks. For instance, DMine takes 533.2 seconds on graphs with  $|G| = (10M, 20M)$  by using 20 processors, when  $k = 10$ ,  $\sigma = 100$  and  $d = 2$ . (2) The number of candidate GPARs is not very large (up to 300), and hence DMine is “parallel scalable” (Section 5.1): it is 3.2 times faster on average when  $n$  increases from 4 to 20, on real-world social networks. (3) Moreover, discovered GPARs based on our  $\text{conf}$  metric predict more precise potential customers in social networks than its PCA and image-based counterparts. (4) Match is parallel scalable: it is 3.53 times faster on average when  $n$  increases from 4 to 20 over real-life social networks. (5) It is practical to apply GPARs to large graphs: on graphs with  $|G| = (50M, 100M)$  and a set  $\Sigma$  of 24 GPARs, Match takes less than 45 seconds with 20 processors. (6) Our optimization

strategies are effective: DMine outperforms DMine<sub>no</sub> by 1.52 times, and Match is 1.27 and 6.24 times faster than Match<sub>c</sub> and disVF2, respectively, on real-life graphs, on average.

## 7. CONCLUSION

We have proposed association rules with graph patterns (GPARs), from syntax, semantics to support and confidence metrics. We have studied DMP and EIP, for mining GPARs and for identifying potential customers with GPARs, respectively, from complexity to parallel (scalable) algorithms. Our experimental study has verified that while DMP and EIP are hard, it is feasible to discover and make practical use of GPARs. We contend that GPARs provide a promising tool for social media marketing, among other applications.

We are currently exploring real-life social graphs to experiment with. Another topic for future work is to extend GPARs by supporting graph patterns as consequent, and by allowing other matching semantics such as graph simulation.

**Acknowledgments.** Fan and Xu are supported in part by 973 Program 2014CB340302. Fan is supported in part by NSFC 61133002, 973 Program 2012CB316200, ERC-2014-AdG 652976, EPSRC EP/J015377/1 and EP/M025268/1, NSF III 1302212, and a Google Faculty Research Award. Fan and Wu are also supported in part by Shenzhen Peacock Program 1105100030834361 and Guangdong Innovative Research Team Program 2011D005. Wang is supported in part by NSFC 61402383 and 71490722, Sichuan Provincial Science and Technology Project 2014JY0207, and Fundamental Research Funds for the Central Universities, China.

## 8. REFERENCES

- [1] GraMi. <https://github.com/ehab-abdelhamid/GraMi>.
- [2] Nielsen global online consumer survey. [http://www.nielsen.com/content/dam/corporate/us/en/newswire/uploads/2009/07/pr-global-study\\_07709.pdf](http://www.nielsen.com/content/dam/corporate/us/en/newswire/uploads/2009/07/pr-global-study_07709.pdf).
- [3] Pokec social network. <http://snap.stanford.edu/data/soc-pokec.html>.
- [4] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD Record*, 22(2):207–216, 1993.
- [5] S. Amer-Yahia, L. V. Lakshmanan, S. Vassilvitskii, and C. Yu. Battling predictability and overconcentration in recommender systems. *IEEE Data Eng. Bull.*, 32(4), 2009.
- [6] M. Berlingerio, F. Bonchi, B. Bringmann, and A. Gionis. Mining graph evolution rules. In *Machine learning and knowledge discovery in databases*, pages 115–130, 2009.
- [7] B. Bringmann and S. Nijssen. What is frequent in a single graph? In *PAKDD*, 2008.
- [8] P. Burkhardt and C. Waring. An NSA big graph experiment. Technical Report NSA-RD-2013-056002v1, U.S. National Security Agency, 2013.
- [9] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *NSDI*, pages 197–210, 2012.
- [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] X. Dong et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *KDD*, 2014.
- [12] A. Dovier, C. Piazza, and A. Policriti. A fast bisimulation algorithm. In *CAV*, pages 79–90, 2001.
- [13] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. GRAMI: frequent subgraph and pattern mining in a single large graph. *PVLDB*, 7(7):517–528, 2014.
- [14] W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for capturing data inconsistencies. *TODS*, 33(1), 2008.
- [15] W. Fan, X. Wang, and Y. Wu. Distributed graph simulation: Impossibility and possibility. *PVLDB*, 2014.
- [16] P. Fournier-Viger and V. S. Tseng. Mining top-k non-redundant association rules. In *ISMIS*. 2012.
- [17] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. AMIE: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.
- [18] M. A. Gallego, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world SPARQL queries. In *USEWOD workshop*, 2011.
- [19] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, 2009.
- [20] N. Z. Gong et al. Evolution of social-attribute networks: measurements, modeling, and implications using google+. In *IMC*, 2012.
- [21] I. Grujic, S. Bogdanovic-Dinic, and L. Stoimenov. Collecting and analyzing data from e-government facebook pages. In *ICT Innovations*, 2014.
- [22] L. B. Holder, D. J. Cook, S. Djoko, et al. Substructure discovery in the subdue system. In *KDD workshop*, 1994.
- [23] J. Huang, K. Venkatraman, and D. J. Abadi. Query optimization of distributed pattern matching. In *ICDE*, 2014.
- [24] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Principles of Data Mining and Knowledge Discovery*. 2000.
- [25] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *Knowledge Eng. Review*, 28(01):75–105, 2013.
- [26] M. Kamber and R. Shinghal. Evaluating the interestingness of characteristic rules. In *KDD*, pages 263–266, 1996.
- [27] Y. Ke, J. Cheng, and J. X. Yu. Efficient discovery of frequent correlated subgraph pairs. In *ICDM*, 2009.
- [28] S.-H. Kim, K.-H. Lee, H. Choi, and Y.-J. Lee. Parallel processing of multiple graph queries using MapReduce. In *DBKDA*, 2013.
- [29] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, 2011.
- [30] C. P. Kruskal, L. Rudolph, and M. Snir. A complexity theory of efficient parallel algorithms. *TCS*, 71(1), 1990.
- [31] S. Lallich, O. Teytaud, and E. Prudhomme. Association rule interestingness: Measure and statistical validation. In *Quality measures in data mining*, pages 251–275, 2007.
- [32] W. Le, A. Kementsietsidis, S. Duan, and F. Li. Scalable multi-query optimization for SPARQL. In *ICDE*, 2012.
- [33] L. Lü and T. Zhou. Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, 390(6):1150–1170, 2011.
- [34] S. A. Myers, C. Zhu, and J. Leskovec. Information diffusion and external influence in networks. In *KDD*, 2012.
- [35] J. Pei and J. Han. Constrained frequent pattern mining: a pattern-growth view. *SIGKDD Explorations*, 4(1), 2002.
- [36] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi. Ja-be-ja: A distributed algorithm for balanced graph partitioning. In *SASO*, 2013.
- [37] R. Raman, O. van Rest, S. Hong, Z. Wu, H. Chafi, and J. Banerjee. PGX.ISO: Parallel and efficient in-memory engine for subgraph isomorphism. *GRADES*, 2014.
- [38] X. Ren and J. Wang. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *PVLDB*, 8(5):617–628, 2015.
- [39] C. Romero, S. Ventura, and P. De Bra. Knowledge discovery with genetic programming for providing feedback to courseware authors. *UMUAI*, 14(5):425–464, 2004.
- [40] D. Saha. An incremental bisimulation algorithm. In *FSTTCS*, 2007.
- [41] C. Schmitz, A. Hotho, R. Jäschke, and G. Stumme. Mining association rules in folksonomies. In *Data Science and Classification*, pages 261–270, 2006.
- [42] P. Shelokar, A. Quirin, and Ó. Córdón. Three-objective subgraph mining using multiobjective evolutionary programming. *JCSS*, 80(1):16–26, 2014.
- [43] C. Smith. Twitter users say they use the site to influence their shopping decisions. *Business Insider Intelligence*, 2013.
- [44] D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy-aware top-k patterns. In *KDD*, 2006.
- [45] W.-S. Yang, J.-B. Dia, H.-C. Cheng, and H.-T. Lin. Mining social networks for targeted advertising. In *HICSS*, 2006.
- [46] C. Zhang and S. Zhang. *Association rule mining: models and algorithms*. 2002.