Knowledge-Based Systems 109 (2016) 256–265

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Efficient pattern matching for graphs with multi-Labeled nodes

Ali Shemshadi[a], Quan Z. Sheng[a], Yongrui Qin[b,*]

[a] School of Computer Science, The University of Adelaide, Adelaide, Australia
[b] School of Computing and Engineering, University of Huddersfield, Huddersfield, United Kingdom

ARTICLE INFO

Article history:
Received 13 December 2015
Revised 3 July 2016
Accepted 5 July 2016
Available online 9 July 2016

Keywords:
Graph matching
Multi-labeled graph
Graph simulation
Metropolis hastings

ABSTRACT

Graph matching is important for a wide variety of applications in different domains such as social network analysis and knowledge discovery. Despite extensive research over the last few decades, graph matching is still challenging particularly when it comes with new conditions and constraints. In this paper, we focus on a new class of graph matching, in which each node can accept multiple labels instead of one. In particular, we address the problem of finding the top-*k* nodes of a data graph which best match a labeled query node from a given pattern graph. We firstly prove this to be an NP-Complete problem. Then, to address this issue and improve the scalability of our approach, we introduce a more flexible graph simulation, namely *surjective simulation*. This new graph simulation reduces the unnecessary complexity that is due to the unnecessary constraints imposed by the existing definitions while achieving high-quality matching results. In addition, our approach is associated with an *early stop* strategy to further boost the performance. To approximate the maximum size of a simulation, our approach utilizes Metropolis Hastings algorithm and ranks the top-k matches after computing the set of surjective simulations. The experimental results over social network graphs demonstrate the efficiency of the proposed approach and superiority over existing approaches.

1. Introduction

Graph pattern matching is fundamental to various applications in many domains including, but not limited to, social computing [1], computer vision [2] and computational chemistry [3]. It has been extensively studied in different contexts within the past few years. Assorted types of conditions and requirements impose different constraints on pattern matching techniques. In general, pattern matching aims to solve the problem of finding subgraphs of a given data graph *G*, which match a given pattern graph *Q*. One of the particular conditions, i.e., processing graphs with labeled nodes [4], is increasingly receiving attention.

The existing approaches for labeled graph pattern matching use a particular definition of labeled nodes where each node is associated with a single label [1]. Thus, for the pattern *Q* and the data graph *G*, the nodes of *G* can be categorized based on the set of labels of nodes in *Q* without any conflict. This approach is useful for many applications in social computing but does not cover some of the new domains, such as the Internet of Things [5], and some sophisticated problems in social networks, i.e., when the labels are uncertain or when the data is incomplete. In any of these cases, assigning a single label to each node could be unrealistic and we need to define graphs with *multi-labeled* nodes. However, using multi-labeled graphs compared to the single-labeled graphs can potentially increase the complexity of the problem. In this case, revising the current pattern matching approaches for multi-labeled graphs is beneficial.

Example 1. To provide a clear image of the problem, in this paper we explain an application in the context of the Internet of Things. For example, a search service extracts the pattern graph from one network of things and the data graph from another. Fig. 1 illustrates two graphs that are obtained from these two networks. Each edge represents a relationship between two nodes in the same network.

Each node (i.e., a thing) is described with a set of metadata about its sensors and actuators. We can take each tag as a label due to some reasons including (1) there is not universal description for things connected to the network, and (2) each node can be registered partially on different networks. The labels are selected from a language $\Sigma.s = \{$sensor/thermal, sensor/weather, sensor/signal, sensor/current, sensor/motion$\}$ and $\Sigma.a = \{$actuator/screen, actuator/switch, actuator/fan, actuator/speaker, actuator/alarm$\}$. Table 1 shows the labels assigned to each node.

* Corresponding author.
E-mail addresses: ali.shemshadi@adelaide.edu.au (A. Shemshadi), michael.sheng@adelaide.edu.au (Q.Z. Sheng), y.qin2@hud.ac.uk (Y. Qin).

http://dx.doi.org/10.1016/j.knosys.2016.07.009
0950-7051/

Table 1
The set of label assignments.

Node	Labels
u_1	$l.s_0$, $l.s_1$, $l.a_0$, $l.a_3$
u_2	$l.s_2$, $l.s_3$, $l.a_1$, $l.a_3$
u_3	$l.s_1$, $l.s_3$, $l.a_2$, $l.a_3$
u_4	$l.s_0$, $l.s_1$, $l.a_0$, $l.a_1$
v_1	$l.s_2$, $l.s_3$, $l.a_1$, $l.a_3$
v_2	$l.s_0$, $l.s_1$, $l.a_0$, $l.a_2$
v_3	$l.s_1$, $l.s_2$, $l.s_3$, $l.a_1$, $l.a_2$
v_4	$l.s_2$, $l.a_0$, $l.a_1$, $l.a_2$
v_5	$l.s_3$, $l.a_0$, $l.a_1$, $l.a_3$
v_6	$l.s_0$, $l.s_1$, $l.a_0$, $l.a_1$
v_7	$l.s_1$, $l.s_2$, $l.s_3$, $l.a_3$
v_8	$l.s_0$, $l.s_2$, $l.a_0$, $l.a_1$
v_9	$l.s_1$, $l.s_2$, $l.s_3$, $l.a_0$
v_{10}	$l.s_0$, $l.s_2$, $l.a_0$, $l.a_1$, $l.a_2$
v_{11}	$l.s_1$, $l.s_2$, $l.a_1$, $l.a_2$
v_{12}	$l.s_0$, $l.s_1$

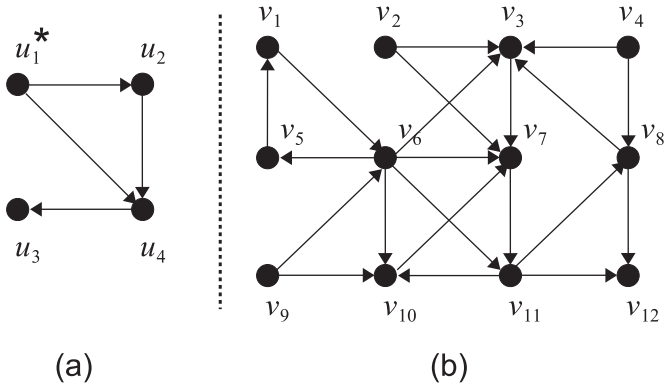

Fig. 1. Querying two networks of things (a) the pattern graph, (b) the data graph.

Table 2
Nodes with label similarity above threshold.

Query graph node	Similar data graph node
u_1	v_2, v_6, v_{12}
u_2	v_1, v_3, v_5, v_7
u_3	v_3, v_7
u_4	v_2, v_6, v_8, v_{10}, v_{12}

To examine the similarity between two nodes, we use a similarity ratio as follows:

$$s(v_i, v_j) = \frac{|L(v_i) \cap L(v_j)|}{|L(v_i) \cup L(v_j)|} \quad (1)$$

where $|L(v_i) \cap L(v_j)|$ denotes the number of common labels between two nodes and $|L(v_i)|$ denotes the number of the labels of v_i. We can compare the similarity score with a threshold t.

Based on the present model, if we set the similarity threshold as 0.5, for each node $u_i \in Q$, Table 2 lists the nodes $v_i \in G$ that can be a match. All of the nodes from the data graph appear more than once in the set of similarity lists. Due to this conflict, no unique label can be associated with any of the nodes. For instance, although v_2 is specified a match of the query node (u_1), it can match node u_4 as well. The set of similarity lists is more complex than the case when each node is assigned only one table. This is because in that case, each node of the data graph would appear only once in the final list. Therefore, the pattern matching process can be more complex and new situations must be considered.

With the increasing complexity and volume of graphs in new contexts such as social networks and the Internet of Things, performance will be the main issue that we should tackle when we revise graph pattern matching for multi-labeled graphs. The typical definitions of graph simulation are generally too restrictive to be applied for this purpose. Nonetheless, computing all of the possible simulations will result in the inefficiency of any solution. In order to tackle these challenges, we propose a novel approach for graph pattern matching for multi-labeled graphs, and briefly, our contributions in this paper are as follows:

- We introduce the concept of *surjective simulation*, which is more flexible than graph simulation and bounded simulation. Through the use of surjective simulation, the proposed approach can notably reduce the complexity of simulation creation step to $|V_p||V|$. With this concept, we can optimize the process via removing the simulations that do not contain any match of the query nodes.
- To avoid going through the computation of each simulation to get its size, we propose an approximation procedure based on the Metropolis-Hastings Algorithm. We also devise an early stop mechanism when (1) we have at least k nodes in the results and (2) the size of the smallest simulation is equal to or greater than the rest of surjective simulations.
- We evaluate the proposed approach via extensive experimental studies. The results show the efficiency of our approach and verify the superiority of our approach over existing approaches.

The rest of this paper is organized as follows. In Section 2 we define the problem. A naive approach based on a very recent work is provided in Section 3. We provide necessary background in Section 4. Then in Section 5 we show how we can compute the set of the top-k results using the proposed concept of surjective simulation with the Metropolis–Hastings algorithm. Section 6 presents the experimental results. Finally, Section 7 reviews the related works and Section 8 provides some concluding remarks.

2. Problem formulation

Before presenting our approach for graph pattern matching, we first formally define the problem that we are going to investigate. Multi-labeled graph pattern matching is the task of matching the nodes of a given pattern graph Q with a data graph G based on structural similarity, where each node is given a set of labels.

Definition 1. Graph A graph is represented as $G = (V, E, L)$ where (1) $V = \{v_1, v_2, \ldots, v_n\}$ is a set of nodes; (2) $E \subseteq V \times V$ is a set of edges; and (3) $L = \{l_i : l_i \in V \times \Sigma\}$ is a mapping that relates each node to a set of assigned labels from language Σ.

Definition 2. Pattern Graph [4,6] A pattern graph is a directed and connected graph $Q = (V_p, E_p, L_p, u^*)$, where (1) V_p is a set of query nodes; (2) E_p is a set of query edges; (3) $L_p \subseteq V_p \times \Sigma$ is a mapping that links every node $u \in V_p$ to a set of labels in $\Sigma_p \subseteq \Sigma$; and (4) $u^* \in V_p$ that specifies the query node.

Definition 3. Graph Simulation [6] A graph G matches a pattern Q iff there exists a binary relation $S \subseteq V_p \times V$ such that (1) for each node $u \in V_p$, there exists a node $v \in V$ such that $(u, v) \in S$, referred to as a match of u; (2) for each pair $(u, v) \in S$, $L(u) = L(v)$, and for each edge (u, u') in E_p, there exists an edge (v, v') in G such that $(u', v') \in S$.

A top-k problem can be defined in the following. Given a surface $\Gamma(x, y)$, a function $f(x, y)$: $x, y \to D$, a scoring function $\delta(f)$, and a positive integer k, it is to find a subset $\phi \subseteq D$, such that $|\phi| = k$ and

$$\phi = \underset{\phi \subseteq \Gamma, |\phi| = k}{\operatorname{argmax}} \sum_{x,y \in \Gamma} \delta(f) \quad (2)$$

In the rest of this paper, we refer to the following problem as *MULTIMATCH*. Given a data graph G and a query graph Q with the query node u^*, assuming that there are at least k subgraphs in G that match the query graph (i.e., containing the nodes, edges and labels in Q), we want to find the top-k matches $M_k = \{m_1, m_2, \ldots, m_k\}$ of u^* in G such that (1) for every $m_i \in M$, there exists at least one simulation S_i that contains (u^*, m_i), and (2) $i < j$ iff $|S_i| \geq |S_j|$ and S_i, S_j represent the largest simulations containing (u^*, m_i) and (u^*, m_j), respectively. In the case that we do not have k matches of the query graph in the subsets of the data graph, we look for all matches instead. In other words, we want to compute the following equation:

$$M = \underset{M \subseteq V, |M|=k}{\operatorname{argmax}} \tilde{\sum_{v \in V}} |S = (V_s, E_s, L_s, M_s)| \tag{3}$$

where S denotes a simulation of pattern $Q = (V_p, E_p, L_p)$ in data graph $G = (V, E, L)$ and $|(V_s, E_s, L_s, M_s)|$ denotes the size of the simulation.

Example 2. For the pattern Q and the data graph G in Fig. 1, we can retrieve simulation $S = \{(u_1, v_6), (u_2, v_{11}), (u_3, v_7), (u_4, v_{10})\}$. S is one of the possible simulations. (3) is strict and does not include other subsets of S.

3. The naive approach

The naive approach explores the whole set of possible permutations for the answers. To the best of our knowledge, one of the closest efforts that only works for single-labeled nodes is the *diversified graph pattern matching* in [7].

As mentioned, high complexity is the first critical issue that we encounter when employing the graph simulation to deal with graphs containing multi-labeled nodes. This is mainly the result of the restrictions imposed by the graph simulation definition. A more flexible definition that is used in naive approaches is the *bounded simulation*.

Definition 4. Strong (Bounded) Simulation [8] A pattern graph Q matches a data graph G via strong simulation, denoted by $Q \prec_D^L G$, if there exists a node v in G and a connected subgraph G_s of G such that (1) $Q \prec_D G_s$, with the maximum match relation S; (2) G_s is exactly the match graph w.r.t. S; and (3) G_s is contained in the ball $\widehat{G}[v, d(Q)]$, where $d(Q)$ is the diameter of Q.

We take the gist of this work to develop a naive approach. We use two algorithms for both Directed Acyclic Graphs (Algorithm 1) but modify them to support the possible worlds of multi-labeled nodes.

Algorithm 1 works as follows. The underlying idea of this algorithm is to execute TopKDAG [7] for all possible label assignments of the both graphs (i.e., Q and G). We first compute the possible worlds for label assignment with $\prod_i L(v_i)$ for G in line 1 and $\prod_j L_p(u_j)$ for Q in line 2. Then, for every possible combination, we use the following process to get the biggest possible simulation S^*. We initialize the min-heap S that is used to store the simulation and the *termination* variable (line 5). The topological rank $r(u)$ is used to keep track of the distance from the starting point. We define $r(u)$ for $u \in G$ as (a) $r(u) = 0$ if the mapped node of u is a leaf in G_{SCC}; and otherwise (b) $r(u) = max\{(1 + r(u'))|(u_{SCC}, u'_{SCC}) \in E_{SCC}\}$. This algorithm dynamically maintains a vector $v.T$, which contains (1) a Boolean equation $v.bf$ of the form $X_v = f$, where f is a Boolean formula that indicates whether v is a match of u; (2) a subset $v.R$ of its relevant set $R(u, v)$; and (3) integers $v.l$ and $v.h$ to estimate the lower and upper bounds of $\delta_r(u, v)$, respectively. The algorithm iteratively computes the set of matches and updates the vectors of other candidates by "propagating" the partially evaluated results.

Algorithm 1: NAIVE-DAG

Require: $Q = (V_p, E_p, L_p, u^*)$ the DAG pattern, G the data graph, t node similarity threshold
Ensure: S^* the biggest set of simulations
1: Let $\mathcal{L} \subseteq \prod_i L(v_i)$ and $\mathcal{L}_p \subseteq \prod_j L_p(u_j)$ be the set of possible permutations of the labels of G and Q
2: **for all** $l \in \mathcal{L}$ **do**
3: **for all** $l_p \in \mathcal{L}_p$ **do**
4: Let `min-heap` $S \leftarrow \varnothing$ and $termination \leftarrow false$
5: **for all** $u \in V_p$ **do**
6: get topological rank $r(u)$ and initialize $can(u)$
7: **for all** $v \in can(u)$ **do**
8: initialize v.T
9: **end for**
10: **end for**
11: **while** $termination = false$ **do**
12: select a set of unvisited candidates $S_c \subseteq can(u)$ of query nodes u in Q, where $r(u) = 0$
13: **if** $S_c \neq \varnothing$ **then**
14: Let $\langle G, S\rangle \leftarrow$ `AcyclicProp`(Q, S_c, G, S)
15: check the termination condition and update $termination$
16: **else**
17: $termination \leftarrow true$
18: **end if**
19: **end while**
20: Update S^*
21: **end for**
22: **end for**
23: **return** S^*

Algorithm 1 supports directed acyclic graphs (DAGs) only. For other types of graphs that contain cyclic paths, the complexity of the solution could rise and naive approaches could be developed based on running the SccProcess [7] for all label permutations. Otherwise, we can convert the graph with cyclic paths to a set of DAGs [9] and use Algorithm 1 for all subgraphs. However, due to the complexity and poor connection with our contribution, in this paper we do not cover this situation and leave it for future research.

Theorem 3.1. *Given a pattern graph $P(V_P, E_P, L_P)$ and a data graph $G(V, E, L)$, the complexity of the naive approach for computing the top-k matches for $u^* \in Q$ is $O(|V|^{|V_p|}|E||E_P|)$.*

Proof. We can prove the complexity of the given problem via computing the upper and the lower bounds for the complexity. To compute the upper bound, we suppose that for each node $v \in G$, $|\Sigma| = |V|$ and $|V|$ different labels have been assigned. In other words, all of the nodes have the same set of assigned labels. Respectively, let us suppose $|\Sigma_p| = |V_p|$ and for each node $u \in Q$, $|V_p|$ labels have been assigned.

This means that in the worst case, we have to go through a search space of all possible matches. In the ith iteration for matching u_i and v_j, $|V|$ possible cases are available. Thus, by the last iteration, $|V_p|^{|V|}$ matches have been traced.

On the other hand, the lower bound for the complexity of the mentioned problem occurs when each node in G is assigned with only one label. All of the nodes $u \in Q$ except one (e.g., u^*) are assigned with only one label while u^* owns n labels. Therefore, by running the TopKDAG [7] for the two possible cases, we end up with $O(n * (|Q||G| + |V|(|V| + |E|) + |V|\log(k)))$ time to compute the top-k matches where $|G| = |V| + |E|$ and $|Q| = |V_P| + |E_P|$. Thus, the time order of the NAIVE-DAG approach is $O(V^2 V_P)$. □

4. Background

In this section, we review some of the necessary background materials before we propose our approach in the next section.

4.1. Markov chains

Here we provide a concise description of the theory of Markov chains. Interested readers may refer to Jerrum et al. [10] for more details. Briefly, a Markov chain is a sequence of random variables $x_1, x_2, x_3, \ldots, x_n$ with the Markov property, namely that, given the present state, the future and past states are independent.

To be more specific, let x be a random variable, where $x(t)$ denotes the value of x at time t. Let $S = \{s_1, \ldots, s_n\}$ be the set of possible x values, denoted the state space of x. If x moves from the current state to a next state based only on its current state, then x follows a Markov process. That is, $Pr(x(t+1) = s_i | x(0) = s_m, \ldots, x(t) = s_j) = Pr(x(t+1) = s_i | x(t) = s_j)$, here s_m is the initial state. The process starts in one of these states s_m and moves successively from one state to another. Each move is called a *step*. A Markov chain is a state sequence generated by a Markov process. The *transition probability* between a pair of states s_i and s_j can be denoted by $Pr(s_i \rightarrow s_j)$. If the chain is currently in state s_i, then it moves to state s_j at the next step with a probability specified by $Pr(s_i \rightarrow s_j)$.

A Markov chain may reach a stationary distribution π over its state space S, where the probability of being at a particular state is independent from the initial state of the chain. There are two conditions of reaching a stationary distribution, including (1) irreducibility (i.e., any state is reachable from any other state), and (2) aperiodicity (i.e., the chain does not cycle between states in a deterministic number of steps). A unique stationary distribution is reachable if the following balance equation holds for every pair of states s_i and s_j :

$$Pr(s_i \rightarrow s_j)\pi(s_i) = Pr(s_j \rightarrow s_i)\pi(s_j) \tag{4}$$

4.2. Markov Chain Monte-Carlo

The concepts of Monte–Carlo method and Markov chains are combined in the Markov Chain Monte–Carlo (MCMC) method [10] to simulate a complex distribution using a Markovian sampling process, where each sample depends only on the previous sample. A Markov chain is normally to start with some transition distribution (a transition matrix in the discrete case) modelling some process of interest, to determine conditions under which there is an invariant or stationary distribution and then to identify the form of that limiting distribution. By contrast, MCMC methods involve the solution of the inverse of this problem whereby the stationary distribution is known, and it is the transition distribution that needs to be identified, though in practice there may be many distributions to choose from.

A standard MCMC algorithm is the Metropolis–Hastings (M–H) sampling algorithm [11]. Suppose that we are interested in drawing samples from a target distribution $\pi(x)$. The (M-H) algorithm generates a sequence of random draws of samples that follow $\pi(x)$. A more detailed process is shown as follows: (1) start from an initial sample x_0, (2) generate a candidate sample x_1 from an arbitrary proposal distribution $q(x_1|x_0)$, (3) accept the new sample x_1 with probability $\alpha = \min\left(\frac{\pi(x_1)\cdot q(x_0|x_1)}{\pi(x_0)\cdot q(x_1|x_0)}, 1\right)$, (4) if x_1 is accepted, then set $x_0 = x_1$, and (5) repeat from step (2).

The (M-H) algorithm draws samples biased by their probabilities. At each step, a candidate sample x_1 is generated given the current sample x_0. The ratio α compares $\pi(x_1)$ and $\pi(x_0)$ to decide on accepting x_1. The (M-H) algorithm satisfies the balance condition (Eq. 4) with arbitrary proposal distributions [11]. Hence, the algorithm converges to the target distribution π. The number of times a sample is visited is proportional to its probability, and hence the relative frequency of visiting a sample x is an estimate of $\pi(x)$. The (M–H) algorithm is typically used to compute distribution summaries (e.g., average) or estimate a function of interest on π.

5. Pattern matching

The high complexity of the naive approach affects the performance of the solution, particularly when the graphs become large. In this section, we describe the details of our approach to resolve the mentioned problem, which consists of two steps: *identifying matches* and *ranking matches*.

5.1. Identifying matches

Bounded simulation has shown to be useful for graphs containing nodes with single label assigned. But when it comes to multiple labels, the use of bounded simulation does not scale well. Thus, we need to define a more flexible definition of graph simulation. In addition, the main criteria that can be used in ranking matches is the number of connected nodes in the simulation. A graph simulation by definition demands that the nodes in the generated simulation remain connected if the nodes in the pattern graph are already connected. Regarding the fact that a bounded simulation does not ensure the connectivity of nodes of the simulation, it increases the complexity of estimating the mentioned criteria. Furthermore, this can increase the complexity of the pattern matching step. Thus, in this paper we use a more flexible definition of graph simulation as follows:

Definition 5. (Surjective Simulation). For a pattern graph $Q = (V_p, E_p, L_p, u^*)$ and a data graph $G = (V, E, L)$, a surjective simulation $S = (V_s, E_s, L_s, M_s)$ is defined as a graph with the following conditions: (1) $\forall v_s \in V_s \Rightarrow \exists (v \in G)$ such that $v = v_s$ and there exists a nonempty path ρ in G which connects every pair of nodes in V_s; (2) $E_s = \{(v_s, v'_s) | v_s, v'_s \in V_s\}$ and for all of its edges there exists a $(v_p, v'_p) \in E_p$ s.t. $s(v_p, v_s) \geq t$ and $s(v'_p, v'_s) \geq t$; (3) Labels L_s are the set of common labels from (2); (4)$M_s = \{(u, v) \subseteq V_p \times V_s \wedge s(u, v) \geq t\}$ where t is a threshold for the degree of labels similarity; and (5) S is connected and $\mathtt{d}(S) \leq \mathtt{d}(P)$ where $\mathtt{d}$ denotes diameter.

We call our proposed graph simulation *surjective* since it covers any part of bounded simulations that exists in a strongly connected component after mapping edges of the pattern and the data graphs. One important difference between the proposed surjective simulation and the graph simulation is that it does not oblige the neighbors' nodes to remain as neighbors after being simulated. Very similar to the bounded simulation, for every neighbor nodes u_m and u_n, it is sufficient to remain connected after being simulated. The idea behind introducing the concept of surjective simulation is to reduce the complexity of finding simulation stage. Based on the Lemma 1, it is possible to use the concept of surjective simulation in domains where simulation has been adopted.

Example 3. Based on the given graphs Q and G in Example 1, a surjective simulation can be obtained as shown in Fig. 2. The surjective simulation can be shown as a graph and each node is mapped to a set of nodes in Q. It is composed of several simulations and all of the nodes similar to the query node can be identified with "$*$" sign.

Lemma 1. *If $\mathcal{S}$ denotes the set of all possible surjective simulations and S_B denotes the set of all bounded simulations for data graph G and pattern Q, for every $\{s \in S_B | s = (V_s, E_s)\}$ there exists a set of surjective simulations $S_s = \{s_1, s_2, \ldots, s_n | s_i = (V'_s, E'_s) \in \mathcal{S}, \sum_i V'_s = V_s \wedge \sum_i E'_s = E_s\}$.*

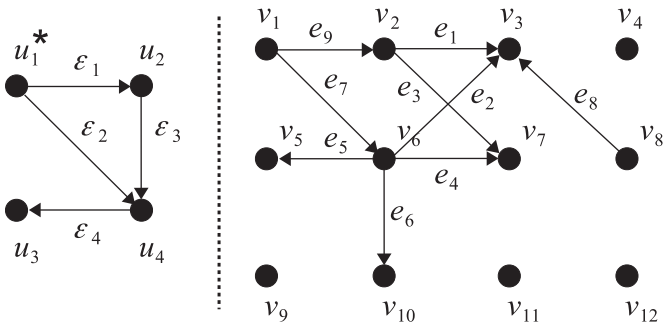

Fig. 2. The query and data graphs after removing non-matching edges.

Theorem 5.1. *The* MULTIMATCH *problem is NP-Complete.*

Proof. We prove the above theorem as follows. First, we show that The *MULTIMATCH* problem is in NP, i.e., we show that any solution to this problem can be verified in polynomial time. In order to verify an answer consisting of k given subgraphs, we show that all of the steps can be accomplished in polynomial time and space. For verifying each given subgraph, the set of nodes and edges could be sorted and matched against the pattern graph in polynomial time. In addition, to verify that the subgraphs are maximal, we can verify the extensions of each subgraph by matching the added node against the pattern. In this regards, to generate each extension, a new edge from the original graph is added to the subgraph. This part of the process can also be done in polynomial time, either.

Second, we show that this problem is a reduction of a known NP-Complete problem. Since in the least complex form, where all nodes in the data graph have unique labels, e.g., each node $v \in V$ has only one label but there does not exist a $u \in V$ where $L(u) \neq L(v)$, the *MULTIMATCH* problem is similar to *topKDP* problem[7] that is shown to be NP-Complete. If we increase the number of labels for all nodes v and assume that for all nodes $v \in V$ the labels are the same s.t. $\forall u \in V$ and $u \neq v$, $L(u) = L(v)$, the *MULTIMATCH* problem is a reduction of subgraph isomorphism [12], which is NP-complete. This completes our proof. □

Therefore, graph pattern matching is very costly. It is NP-complete for subgraph isomorphism [12], cubic-time for bounded simulation [1], and quadratic-time for simulation [6]. Taking the advantage of the surjective simulation, we can avoid processing the nodes which may not be a part of the maximum unique simulation and will be eliminated from the pool of alternatives for further processing. To achieve this goal, we propose an index for surjective simulation with a simple structure in the following: $\mathcal{I}_u = \{(u, v) | u \in Q, v \in G \wedge \sigma(L_p(u), L(v)) \geq t\}$.

Algorithm 2 provides the steps to create and maintain the set of surjective simulations. The algorithm returns a set of surjective simulations $\mathcal{S}$ based on the given data graph G, the given pattern graph Q and a given threshold for node similarity t. At the first step, matrix $\mathcal{M} = \{(\mathcal{M}_V, \mathcal{M}_E) | u \in Q, v \in G\}$ denotes a mapping from Q to G. For every edge $e_p \in Q$ and $e \in G$, we iteratively (1) compare all labels and compute the corresponding degree of similarity based on the number of common labels, and (2) add the new nodes and the new edge to $\mathcal{M}$. In the next step, we update the indexes including the set of query nodes ($s.q$), the size of connected component ($s.s$). The new surjective simulation is then added to $\mathcal{S}$.

One strength of Algorithm 2 is its *low complexity* without losing too much matching quality. In the proposed algorithm, for the two **for** loops in steps 2–7, if we use an index to get the mapped nodes in G, the complexity will be $O(|E||E_p|)$. Based on the fact that getting the connected components in step 8 is performed in a polynomial time $O(|G|)$, the whole process runs in $O(|V||V_p| + |G|)$ time.

Algorithm 2: GET-SURJECTIVE-SIMULATIONS

Require: Q the pattern graph, G the data graph, t node similarity threshold
Ensure: $\mathcal{S}$ the set of surjective simulations
1: Let $\mathcal{M}$ be the set mapped nodes in G
2: **for all** $e_p = (u_p, v_p) \in Q$ **do**
3: **for all** $e = (u, v) \in G$ **do**
4: Let sim_u be the degree similarity of u and u_p based on the set of their labels
5: Let sim_v be the degree similarity of v and v_p based on the set of their labels
6: **if** $sim_u > t$ **and** $sim_v > t$ **then**
7: Add (u, v) to $\mathcal{M}$
8: **end if**
9: **end for**
10: **end for**
11: **for all** Connected component $C \in \mathcal{M}$ **do**
12: Let $s.q$ point to the set of query nodes in C
13: Let $s.s$ point to the size of C
14: **if** $|s.q| > 0$ **and** $\mathtt{d}(S) \geq \mathtt{d}(Q)$ **then**
15: Add surjective simulation S with specifications $s.s$ and $s.q$ to the $\mathcal{S}$
16: **end if**
17: **end for**
18: Sort $\mathcal{S}$ based on $s.s$ value for each set
19: **return** $\mathcal{S}$

Table 3
Mapping of the edges of G to edges in the query graph Q.

Query graph edge	Matching data graph edge
ϵ_1	e_1, e_2, e_3, e_4, e_5, e_9
ϵ_2	e_6
ϵ_3	e_7
ϵ_4	e_1, e_2, e_3, e_4, e_8

Example 4. Fig. 2 shows the outcome of passing parameters Q, G, and the threshold from previous example to the Algorithm 2. The set of mappings of the edges is shown in Table 3. During the edge matching procedure, some of the edges in the data graph are omitted when no corresponding edge is found in the query graph for them. Thus, one of the first things that can be observed is that the data graph is divided into multiple connected components as shown. Thus, connected components with lesser nodes than the query graph can be removed from the result set. As a result, in the next step, we can limit our search only to the connected components with greater number of nodes. In this example, we have only one large connected component and other nodes including v_4, v_9, v_{11} and v_12 will no longer be considered to match any node in the query graph. In the next step, we observe that each query edges ϵ_2 and ϵ_3 are mapped to only one corresponding data edge each. On the other hand, there are multiple data graph nodes that match query nodes ϵ_1 and ϵ_4. Based on this example, no match can be retrieved using simulation and strong simulation definitions. However, based on the surjective simulation definition, different subgraphs exist in the main connected component which contain nodes and edges that match nodes and edges in the Q. These surjective matches include any connected subgraph with diameter 3 that contains $e6$ and e_7.

5.2. Top-k matches

We propose a new approach, which approximates the maximum-sized match so as to reduce the complexity introduced

by the naive approach. The main idea is to employ the set of surjective simulations to start the ranking procedure with an inter-simulations approach that first gets sufficient candidates. Then it further ranks and sorts the intra-simulations. To provide an efficient solution for finding top-k matches, we divide this step into two sub-problems. In the first step, we look for the suitable surjective simulations within the identified set of surjective simulations from Algorithm 2. In the second step, we approximate the scores and rank of each match of the query node until we obtain the top-k nodes.

Algorithm 3 details the second step to obtain the top-k results. Unlike Algorithms 1, we can process the both kinds of input (directed acyclic graphs and graphs with cycles) using the same procedure. In this algorithm, similar to TopKDAG and SccProcss [7], we use an *early termination* strategy by employing a termination variable. This variable gets updated after each iteration by checking the conditions where (1) we still have more simulations in the $\mathcal{S}$ to process, and (2) if the top k list is filled, the approximation of the size of the smallest simulation is equal to or less than the size of the next surjective simulation in the ordered set $\mathcal{S}$, which is obtained from the previous step.

Algorithm 3: GET-TOP-K

Require: $\mathcal{S}$ the set of simulations, Q the pattern graph, G the data graph, t node similarity threshold, k
Ensure: V' the list of top-k nodes

```
1: Let termination ← false
2: while termination = false do
3:   Let S ∈ 𝒮 be the next member of simulation set
4:   for all u ∈ S.u* do
5:     Let x_0 ← {u}
6:     for all u' ∈ S.U do
7:       if u' ≠ u then
8:         Initialize π(u')
9:         Add none state to π(u')
10:        Add a random member from π(u') + 1 to x_0
11:      end if
12:    end for
13:    Let terminate_s ← false
14:    while terminate_s = false do
15:      Let x_1 ← {u}
16:      for all u' ∈ S.U do
17:        if u' ≠ u then
18:          Add a random member from π(u') + 1 to x_1
19:        end if
20:      end for
21:      Compute the transition probability p
22:      Accept the new state x_1 by probability p
23:      Update terminate_s
24:    end while
25:    if |V'| < k then
26:      Add M to V'
27:    else if |M| > |V'_k| then
28:      Let V'_k ← M
29:    end if
30:  end for
31:  Update terminattion
32: end while
33: return V'
```

The rest of Algorithm 3 is designed based on the Metropolis-Hastings (M-H) Algorithm. For each match of query node in each simulation ($u \in S.u^*$), we perform matching in the following order. We generate the initial state x_0 in steps 5–10 by parsing every node u' in the set of nodes of the surjective simulation ($S.U$). The set $\pi(u')$ is initialized with the all of the labels l_i of node u' in step 8. Also there is a possibility that the node u' does not appear in a simulation. Thus we generate (step 9) and add (step 10) a *nil* state to the $\pi(u')$. We also use a termination mechanism for the next steps by using the $termination_s$ variable. The variable can be changed to true (step 19) after we get a certain number of iterations with no change in the approximate size of simulation S. In steps 14–16 we generate a new random state x_1 in a process similar to x_0 initiation. In the next steps (steps 17,18), we compute the probability of accepting the new state x_1 by the following equation:

$$p = min\left(\frac{\pi(x_1).q(x_0|x_1)}{\pi}(x_0).q(x_1|x_0\right) \tag{5}$$

The early termination strategy is implemented afterwards in steps 20–23. Finally, the top k matches are returned in step 25.

To calculate the values of $\pi(x_n)$ and $q(x_m|x_n)$, we can use the following approach: Supposing that to create $\pi(x_n)$ a procedure f in $i = 1, \ldots, N$ steps is executed. In each step, a new node is selected and added to the graph simulation. Based on the fact that the probability of state x_{i+1} in the step $i+1$ is independent from the states $1, 2, \ldots, i-1$ and only depends on the state i, we assume that the sequence is a Markov chain. Thus, for the step $i+1$, we have:

$$\pi(x_{i+1}) = \frac{1}{P(x_{i+1})} \tag{6}$$

Similarly, to calculate $q(x_m|x_n)$, we can compute the cost of converting x_n to x_m. In step i, we can convert x_n to x_m by converting the selected element to the new selection. In other words, to make it simple, we can use the following equation:

$$q(x_m|x_n) = \frac{\pi(x_m \cap x_n)}{\pi(x_m - x_n)} \tag{7}$$

where $x_m \cap x_n$ denotes the sequence of common selections between x_m and x_n and $x_m - x_n$ denotes the sequence of difference of selections in x_m.

6. Experimental evaluation

We have implemented the proposed approach and conducted extensive experiments to study its performance.

6.1. Experimental setting

We used Java 1.6 and graphstream[1] 1.2 in the implementation. The experiments were conducted on a computer with a 3.4 GHz Intel Core i7 processor and 8 GB of memory running Ubuntu 14.04. We used the following datasets:

- Facebook dataset:[2] an anonymized social circles graph from Facebook with 4039 nodes and 88,234 edges. Each node is associated with a set of features which are dynamically defined and mapped.
- Twitter dataset:[3] an anonymized social circles graph from Twitter with 81,306 nodes and 1,768,149 edges. Each node is associated with a set of features which are dynamically defined and mapped.
- Two synthetic data sets: we created a graph generator which takes the size of the nodes and the size of the edges, and then generates a graph with randomly selected labels for its nodes. We generated two graphs: one with 1000 nodes and 10,000

[1] http://graphstream-project.org
[2] http://snap.stanford.edu/data/egonets-Facebook.html
[3] http://snap.stanford.edu/data/egonets-Twitter.html

edges and the other with 1000 nodes and 100,000 edges. Each node is associated with a set of features which are randomly selected from a set of 40 labels similarly defined as the ones in Example 1.

We also implemented a pattern generator for generating the graph $Q = (V_p, E_p, L_p, u^*)$ based on a given set of parameters such as graph $G_p = (V, E, L)$, which is the source graph for generating patterns, and an integer number r, which is the Euclidean radius of the pattern graph starting from the query node.

The pattern's label assignment process is carried out in the same way as for data graphs label assignment. For this purpose, to maintain the closeness of the problem with reality, we used the set of tags and features for each node provided by each dataset. At the beginning, each node is associated with the set of its own labels. To compare the similarity of the labels of two nodes, we used the Jaro Winkler method [13]. To optimize the proficiency of the solution, we developed an index which stores the nodes that have similar labels for a given node u_i using the process shown in Algorithm 4.

Algorithm 4: INITIALIZE-INDEX

Require: Q_s the source graph for pattern, G the data graph, t node similarity threshold
Ensure: $\mathcal{I}$ nodes similarity index
1: **for all** Node $u \in Q_s$ and $v \in G$ **do**
2: Let $s_{u,v} \leftarrow$ `Jaro-Winkler`$(L(u), L(v))$ be the similarity score of u and v
3: **if** $s_{u,v} \geq t$ **then**
4: Add v to $\mathcal{I}_u$
5: **end if**
6: **end for**
7: **return** $\mathcal{I}$

To generate the pattern, a random node from Q_s is picked as the query node u^* and for the rest of the nodes, only nodes within a certain distance from u^* ($\leq r$), are picked out with related edges and added to form Q. Jaro–Winkler is a measure of similarity between two string based on the Jaro distance metric. The Jaro–Winkler score is normalized in the range of [0, 1] as the higher score denotes the higher similarity between strings. It is well suited for short strings and a good option for comparing features of different nodes.

Table 4 provides a summary for the construction of the proposed index for each pair of used datas ets. Technically, the amount of time and the size of index highly depend on the similarity threshold, or the degree of similarity for the set of nodes, in each test case. However, the results show that although the amount of time and space are still high, using the proposed index can significantly decrease the query processing time (see Fig. 4(n)). To maintain the readability of the figure, the results from Q:Twitter/G:Twitter, which could not be processed without using the index, are not shown.

6.2. Efficiency

To assure the efficiency of our approach, we performed several tests using each of the data sets mentioned above. Each efficiency test requires a pattern source graph Q_s and a data graph G. In Fig. 4, all of the efficiency test results are depicted as scatter plots along with their regression (red) lines. From all datasets, we selected several scenarios as follows:

- *Facebook as both the pattern source and the data graph*: In this scenario, we used the node similarity threshold $t = 0.97$ due to the high similarity of the metadata of features. The pattern was generated randomly by the Elucidian radius $r = 2$. The size of the set of matching nodes for a given node deviates between 0 and 3500 (Fig. 4(c)). We ran this test 100 times with 100 M-H iterations. The results are shown in Fig. 4(a) and (b). Regression lines show the linear increment in the average processing time as the number of pattern nodes, pattern edges and the average similarity degree increase.
- *Facebook as the pattern source and Twitter as the data graph*: In this scenario, we used the node similarity threshold $t = 0.7$. The pattern was generated randomly by the Elucidian radius $r = 2$. The average size of the set of matching nodes for a given node deviates between 25 to 462 (Fig. 4(f)). We ran this test 200 times. The results are shown in Figs. 4(d) and (e). Fig. 3(a) shows an example of a large pattern graph generated from the Facebook data set and Fig. 3(b) smaller patterns through breaking down larger patterns. The decreasing regression line in the pattern nodes increment is due to the special nature of the patterns extracted from Facebook dataset and the cross domain pattern matching. T
- *Twitter as both the pattern source and the data graph*: In this scenario, we used the node similarity threshold $t = 0.75$. The pattern was generated randomly by the Elucidian radius $r = 1$. The average size of the set of matching nodes for a given node deviates between 0 and 3512 (Fig. 4(i)). We ran this test 100 times with 100 M-H iterations. The results are shown in Figs. 4(h) and (g). As the Figures show, there is a linear increment in processing time as the number of edges and nodes increase. However, the average similarity degree does not show a strong impact on the processing time.
- *Two synthetic graphs, the smaller one as the pattern source and the larger as the data graph*: The pattern source graph Q_s contains 1000 nodes and 10,000 edges. The pattern was generated randomly by the Elucidian radius $r = 2$. We used threshold $t = 0.9$ to compare the similarity of labels. The size of the set of matching nodes for a given node deviates between 6 and 7 matches per node (Fig. 4(l)). We ran this test 100 times with 100 M-H iterations. The results are shown in Figs. 4(k) and (j). Same as the above case, there is a linear increment in processing time as the number of edges and nodes increase while, the average similarity degree does not show a strong impact on the processing time.

From the results, we understand that for more than 90% of the queries, the result can be prepared in a short time. Based on the input graphs, the portion of the short process time may vary, but it deviates in a limited interval for the social networks data sets. To

Table 4
Index construction summary for different data sets.

First data set	Second data set	Threshold	Time	Comparisons	Entries	Size (MB)
Synthetic	Synthetic	0.7	1,472.296 s	10,000,000 comparisons	236,491	1.5
Facebook	Facebook	0.998	2,426.721 s	16,000,000 comparisons	330,739	1.6
Facebook	Twitter	0.7	7,832.821 s	324,000,000 comparisons	4,585,485	21.7
Twitter	Twitter	0.75	19,397.461 s	6,561,000,000 comparisons	9,562,748	89.1

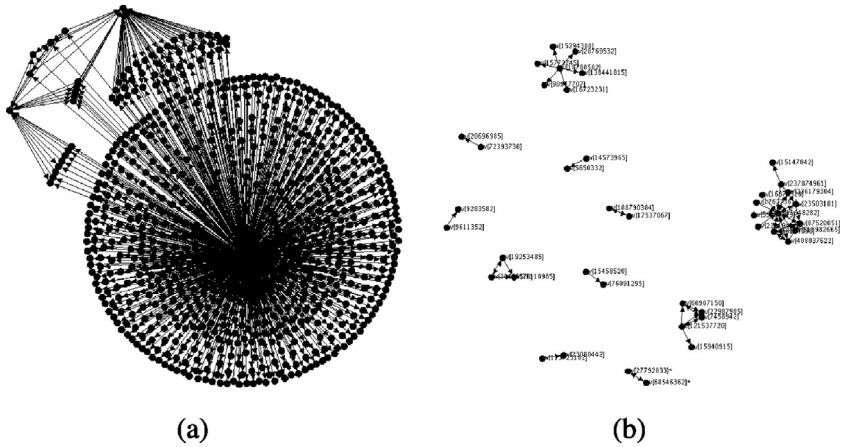

Fig. 3. (a) Large pattern from Facebook, (b) smaller patterns generated using greater thresholds.

have a better estimation of the impact of the proposed approach, we can estimate the time required by the naive approach based on Theorem 3.1.

6.3. Discussion

To examine our approach, we used a different mix of the data sets. We designed a new test, in which we used the same data set as the pattern source Q_s and data graph G with the same label setting. In other words, we selected the following data set matches for this test:

- *Facebook as both the pattern source and the data graph*: In this scenario, we used the node similarity threshold $t = 0.97$ for Jaro–Winkler label similarity due to the high similarity of the metadata of features.
- *Twitter as both the pattern source and the data graph*: In this scenario, we used the node similarity threshold $t = 0.75$ for Jaro–Winkler label similarity.

6.3.1. Time complexity

The reduced complexity of relaxed pattern matching and the proposed approximation technique does not come without any price. The precision of the estimated scores for each match depends on the ratio of the explored possible worlds in the universal set of possible matchings.

The range of approximation error can change from 1.0, which results in low precision, to very small values based on the traversed portion of possible worlds. With fixed number of iterations for Metropolis Hastings, the accuracy of the result depends on the nature of the problem. For example, for an extreme case in which all nodes have similar labels, the result from our approach will be approximated by a relatively low precision while finding a precise answer will be Np-Complete.

6.3.2. Comparison with baseline

To assure the improvement from existing works, we compare the total runtime our approach with the naive execution of the existing works. We use the synthetic data set and use a query pattern with 20 nodes. The naive approach is based on the *SccProces* and *TopKDAG* algorithms [7]. As Fig. 4(m) shows, in spite of the heavy indexing, our approach outperforms the baseline as the average degree of similarity (average number of labels for each node) increases.

6.3.3. Graph representation and Space complexity

Graph representation can affect space complexity. The space complexity in our approach can be divided into three major parts including graph representation, runtime space and index storage. However, unlike the Algorithm 1 which is not scalable due to the large space that is required to store all permutations of the labels, our approach only records trees of parsed nodes in connected components which do not highly exceed the size of the given query graph.

There are two different approaches to store graph data including *Adjacency List* and *Matrix* representation. The first approach normally reduces the file size while the second approach can potentially reduce the runtime of the process. The original version of the real-world data sets that we have used, are stored using adjacency list format. However, due to the heavy processing workload in our experiment, we convert the graphs to adjacency matrices as we import them into our application. Through the use of scalable libraries, our approach can handle graphs with millions of nodes on one machine.

As Table 4 shows, the size of the index increases when we process larger graphs or use smaller thresholds. As shown, the size of the index can take up 1.6 MB for around 330,000 entries or up to 89.1 MB for 9.5 million entries. Thus, our approach is scalable in terms of space and time.

7. Related works

Graph pattern matching has been extensively studied and employed in various domains. Typically, two main approaches are often used for this purpose: subgraph isomorphism [14–17] and graph simulation [18–20]. When it comes to new applications such as social computing, none of these approaches can meet the applications' requirements. Recently, there has been a trend towards revising pattern matching for new domains, e.g. social networks [4]. The main problem in deploying the mentioned approaches is their too much restrictive definition. This reduces the efficiency and scalability of matching process. In addition, in the case of social networks analysis, these approaches will fail to obtain many useful and meaningful solutions [4].

One of the problems which recently has been discussed along with graph pattern matching is querying top-k nodes in a data graph by matching a query node from a pattern graph [7]. In the previous works both of the pattern graph and data graph use a single tag for each node [4]. The top-k query processing is challenging and an ongoing issue particularly under special conditions such as uncertainty. For instance, in a distinct approach, given a noisy or uncertain dataset, Song et al. propose to return *top-k Oracle* instead of returning only *k* tuples [21]. Users can later sub-query the results oracle to get the best answers based on their choice. However, so far this approach has not been examined for graphs, thus this approach needs further investigations to be

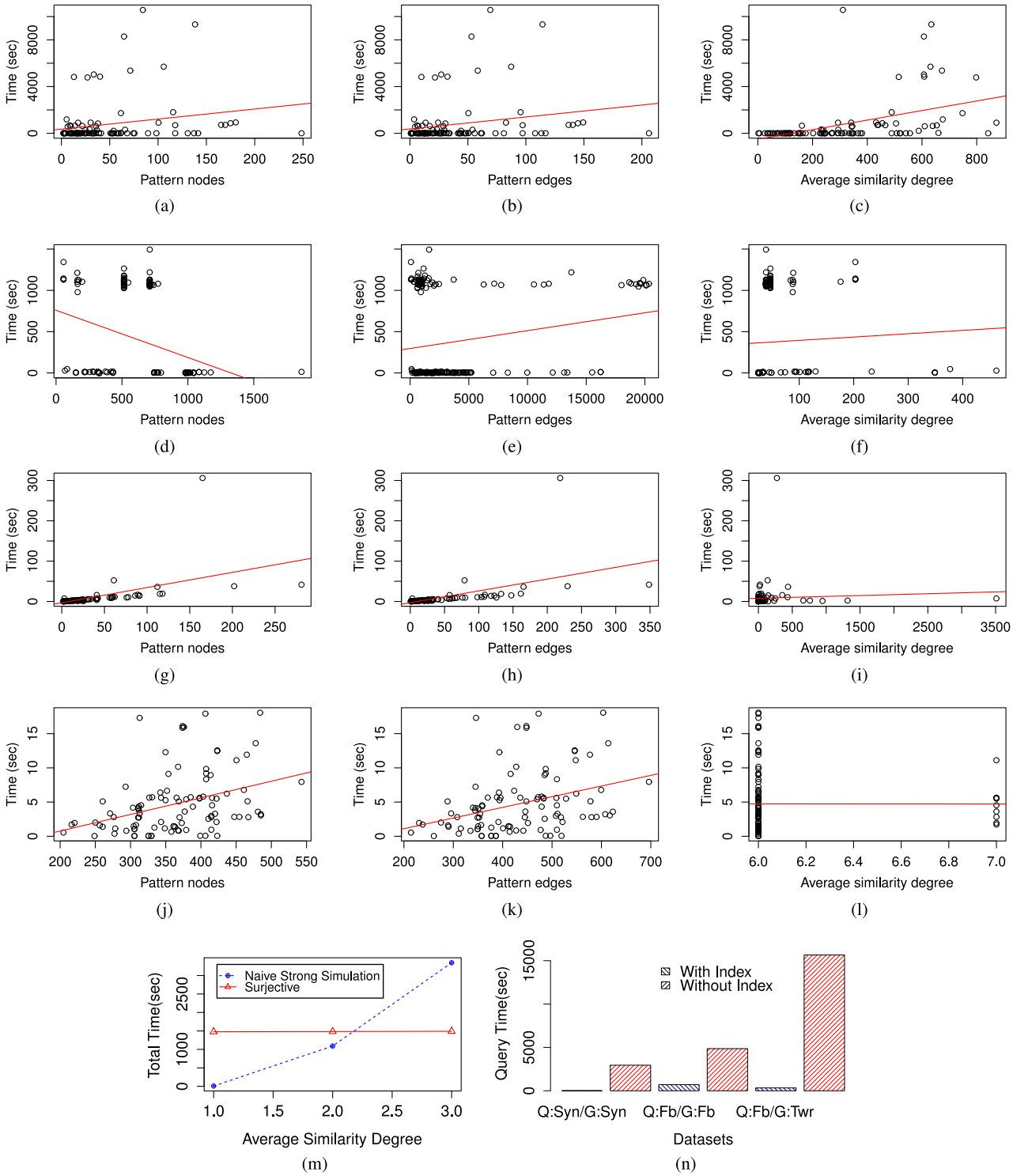

Fig. 4. The results of the experimental evaluation for (a,b,c) Q_s: Facebook and G: Facebook; (d,e,f) Q_s: Facebook and G: Twitter; (g,h,i) Q_s: Twitter and G: Twitter; (j,k,l) Q_s: Synthetic data set 1 and G: Synthetic data set 2; (m) total runtime including indexing time and comparison with naive strong simulation; and (n) query processing with and without using the similarity index.

applied on graph shaped data. In more graph oriented applications, top-k exploration has been applied to efficient graph search on RDF data, which is represented using basic graph patterns [22].

Due to the frequently changing nature of social networks, it is often too costly to recompute all of the matches of a given pattern starting from scratch. To respond to this issue, the incremental graph pattern matching [23] can be adopted in many cases [24]. This approach identifies the changes in the data graph once they are made and corrects the pre-computed match set regardless of the complexity of batch algorithms.

The trend towards reforming the concept of graph simulation and graph isomorphism has continued to form new definitions such as bounded graph simulation [1], edges relationship simulation [25], strong simulation [8,26] and strict simulation [27]. The

strict simulation has been accompanied with distributed computing to tackle the performance obstacle. Using on-the-fly ranked lists and spanning trees is another approach that can optimize the solution [28]. One of the most important factors in this evolution process is the specifications of the application context. However, none of these efforts has applied graphs with multi-labeled nodes. Nonetheless, the size of pattern in most cases is very small. Efficient processing of probabilistic graphs has also been discussed by previous works [29]. Although dealing with probabilistic graphs to some extent can have similarities with analyzing graphs with multiple node labels, their purpose and definition are quite different.

8. Conclusion

Graph matching is a fundamental method that is applied in many important areas, such as social computing, image processing and computational chemistry. With the creation of new applications (such as the Internet of Things) and the evolution of old ones, the need for more powerful matching techniques is increasing.

In this paper, we have proposed a novel approach to efficiently match large graphs with nodes that can take several features. The several aspects of the novelty of our approach include: (1) we address pattern graphs that are much larger than pattern graphs used by previous approaches, (2) in our approach each node can be associated with multiple labels while previous approaches only accept one label for each node, (3) we propose the concept of surjective simulation that is more flexible than graph simulation and strong simulations, and we have shown how it can greatly improve the efficiency of the graph matching process. We have conducted several experiments to examine the efficiency of our approach and demonstrate the superiority of our approach over existing methods.

For the future research, we plan to extend our approach to support several new specifications, such as matching probabilistic multi-labeled graphs and graph based processing for large-scale distributed pattern matching. We also plan to examine our approach on more datasets and investigate the merits of its results in new applications, such as the Internet of Things.

References

[1] W. Fan, J. Li, S. Ma, H. Wang, Y. Wu, Graph homomorphism revisited for graph matching, Proc. VLDB Endowment 3 (1–2) (2010) 1161–1172.

[2] M. Cho, J. Lee, K.M. Lee, Reweighted random walks for graph matching, in: Proceedings of the 11th European Conference on Computer Vision (ECCV), Springer, Crete, Greece, 2010, pp. 492–505.

[3] L. Livi, A. Rizzi, The graph matching problem, Pattern Anal. Appl. 16 (3) (2013) 253–283.

[4] W. Fan, Graph pattern matching revised for social network analysis, in: Proceedings of the 15th International Conference on Database Theory (ICDT), ACM, 2012, pp. 8–21.

[5] L. Atzori, A. Iera, G. Morabito, The internet of things: A survey, Comput. Netw. 54 (15) (2010) 2787–2805.

[6] M.R. Henzinger, T.A. Henzinger, P.W. Kopke, Computing simulations on finite and infinite graphs, in: Proceedings of the 36th Annual Symposium on Foundations of Computer Science, IEEE, Milwaukee, USA, 1995, pp. 453–462.

[7] W. Fan, X. Wang, Y. Wu, Diversified top-k graph pattern matching, Proc. VLDB Endowment 6 (13) (2013) 1510–1521.

[8] S. Ma, Y. Cao, W. Fan, J. Huai, T. Wo, Capturing topology in graph pattern matching, Proc. VLDB Endowment 5 (4) (2011) 310–321.

[9] J. Gao, C. Zhou, J. Zhou, J.X. Yu, Continuous pattern detection over billion-edge graph using distributed framework, in: Proceedings of the 30th International Conference on Data Engineering (ICDE), IEEE, 2014, pp. 556–567.

[10] M. Jerrum, A. Sinclair, The markov chain monte carlo method: an approach to approximate counting and integration, Approx. Algo. NP-hard Probl. (1996) 482–520.

[11] W. Hastings, Monte carlo samping methods using markov chains and their applications, Biometrika 57 (1) (1970) 97–109.

[12] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1990.

[13] W. Cohen, P. Ravikumar, S. Fienberg, A comparison of string metrics for matching names and records, in: KDD Workshop on Data Cleaning and Object Consolidation, volume 3, Washington DC, USA, 2003, pp. 73–78.

[14] C.C. Aggarwal, H. Wang, Managing and Mining Graph Data, volume 40, Springer, 2010.

[15] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, Int. J. Pattern Recognit. Artif. Intell. (IJPRAI) 18 (03) (2004) 265–298.

[16] B. Gallagher, Matching structure and semantics: A survey on graph-based pattern matching, AAAI Fall Symposia 6 (2006) 45–53.

[17] D. Shasha, J.T. Wang, R. Giugno, Algorithmics and applications of tree and graph searching, in: Proceedings of the 21st ACM Symposium on Principles of Database Systems (PODS), ACM, Madison, USA, 2002, pp. 39–52.

[18] J. Brynielsson, J. Hogberg, L. Kaati, C. Mårtenson, P. Svenson, Detecting social positions using simulation, in: Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, Odense, Denmark, 2010, pp. 48–55.

[19] J. Cho, N. Shivakumar, H. Garcia-Molina, Finding replicated web collections, in: Proceedings of the 2000 ACM International Conference on Management of Data (SIGMOD), ACM, Dallas, Texas, 2000, pp. 355–366.

[20] L. De Nardo, F. Ranzato, F. Tapparo, The subgraph similarity problem, IEEE Trans. Knowl. Data Eng. (TKDE) 21 (5) (2009) 748–749.

[21] C. Song, Z. Li, T. Ge, Top-k oracle: A new way to present top-k tuples for uncertain data, in: Proceedings of the 29th International Conference on Data Engineering (ICDE), IEEE, Brisbane, Australia, 2013, pp. 146–157.

[22] T. Tran, H. Wang, S. Rudolph, P. Cimiano, Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data, in: Proceedings of the 25th International Conference on Data Engineering (ICDE), IEEE, Shanghai, China, 2009, pp. 405–416.

[23] W. Fan, X. Wang, Y. Wu, Incremental graph pattern matching, ACM Trans. Database Syst. (TODS) 38 (3) (2013) 18.

[24] A. Ntoulas, J. Cho, C. Olston, What's new on the web?: the evolution of the web from a search engine perspective, in: Proceedings of the 13th International Conference on World Wide Web (WWW), ACM, New York, USA, 2004, pp. 1–12.

[25] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, Adding regular expressions to graph reachability and pattern queries, in: Proceedings of the 27th International Conference on Data Engineering (ICDE), IEEE, Hannover, Germany, 2011, pp. 39–50.

[26] S. Ma, Y. Cao, W. Fan, J. Huai, T. Wo, Strong simulation: Capturing topology in graph pattern matching, ACM Trans. Database Syst. (TODS) 39 (1) (2014) 4.

[27] A. Fard, M.U. Nisar, L. Ramaswamy, J.A. Miller, M. Saltz, A distributed vertex–centric approach for pattern matching in massive graphs, in: Big Data, 2013 IEEE International Conference on Big Data (IEEE BigData), IEEE, Santa Clara, USA, 2013, pp. 403–411.

[28] J. Cheng, X. Zeng, J.X. Yu, Top-k graph pattern matching over large graphs, in: Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE), IEEE, Brisbane, Australia, 2013, pp. 1033–1044.

[29] Y. Yuan, G. Wang, L. Chen, H. Wang, Efficient subgraph similarity search on large probabilistic graph databases, Proc. VLDB Endowment 5 (9) (2012) 800–811.