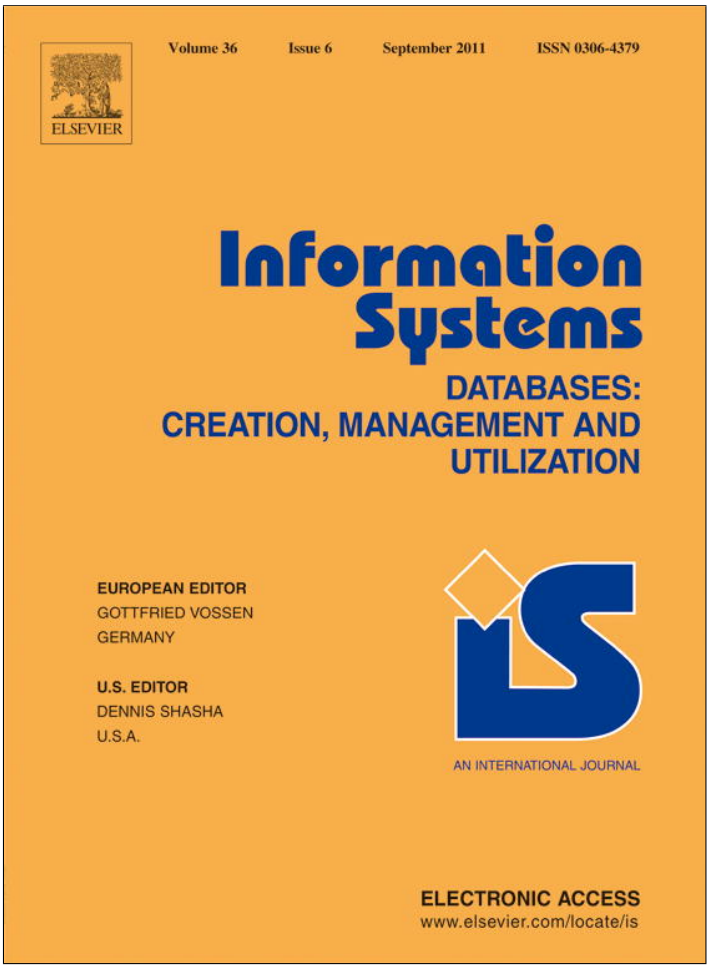

Information Systems 36 (2011) 958–972

Contents lists available at ScienceDirect

Information Systems

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

Structure and attribute index for approximate graph matching in large graphs

Linhong Zhu [a,*,1], Wee Keong Ng [b], James Cheng [b]

[a] Data Mining Department, Institute for Infocomm Research, Singapore
[b] School of Computer Engineering, Nanyang Technological University, Singapore

ARTICLE INFO

Article history:
Received 22 January 2010
Received in revised form
24 March 2011
Accepted 29 March 2011
Recommended by: F. Korn
Available online 2 April 2011

Keywords:
Approximate graph matching
Graph indexing
Social network analysis

ABSTRACT

The increasing popularity of graph data in various domains has lead to a renewed interest in developing efficient graph matching techniques, especially for processing large graphs. In this paper, we study the problem of approximate graph matching in a large attributed graph. Given a large attributed graph and a query graph, we compute a subgraph of the large graph that best matches the query graph. We propose a novel structure-aware and attribute-aware index to process approximate graph matching in a large attributed graph. We first construct an index on the similarity of the attributed graph, by partitioning the large search space into smaller subgraphs based on structure similarity and attribute similarity. Then, we construct a connectivity-based index to give a concise representation of inter-partition connections. We use the index to find a set of best matching paths. From these best matching paths, we compute the best matching answer graph using a greedy algorithm. Experimental results on real datasets demonstrate the efficiency of both index construction and query processing. We also show that our approach attains high-quality query answers.

1. Introduction

Data and their relationship in various domains such as the Semantic Web, images, videos, social networks, bioinformatics, and so on, exhibit graph-like structures. In recent years, the accelerated growth of such data and their applications have sparked renewed interests in developing approximate (inexact) graph matching techniques in large graphs. Approximate matching is an important extension to the exact graph matching, as to allow more flexibility and cater for the need of many applications where exact information is often not available. Approximate graph matching has a wide range of applications. For example, in biology, approximate graph query is used to search for protein complex[2] in a large protein–protein interaction network [1]. Another example is terrorist modus operandi detection [2], which is a form of searching for a query graph (a threat pattern) in a graph that stores a large volume of observed threat activities data.

We give an example of approximate graph matching as follows:

Example 1. Fig. 1 shows a sample data graph and query graph. The data graph shows the relationship among scholars from different research areas. Suppose that we want to find Alice in machine learning and her connection to other scholars in different research areas, we can create an abstract query graph as shown in Fig. 1(a). Clearly, exact matching returns no answer to this query (indeed, Alice may not be connected to scholars in every area).

* Corresponding author. Tel.: +6564082307; fax: +6567761378.
E-mail addresses: LZHU@i2r.a-star.edu.sg (L. Zhu), AWKNG@ntu.edu.sg (W. Keong Ng), jamescheng@ntu.edu.sg (J. Cheng).

[1] This work was done when the author was a Ph.D. candidate in School of Computer Engineering, Nanyang Technological University, Singapore.

[2] Usually a protein complex is a dense subgraph whose nodes (proteins) perform similar biological functions.

0306-4379/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2011.03.009

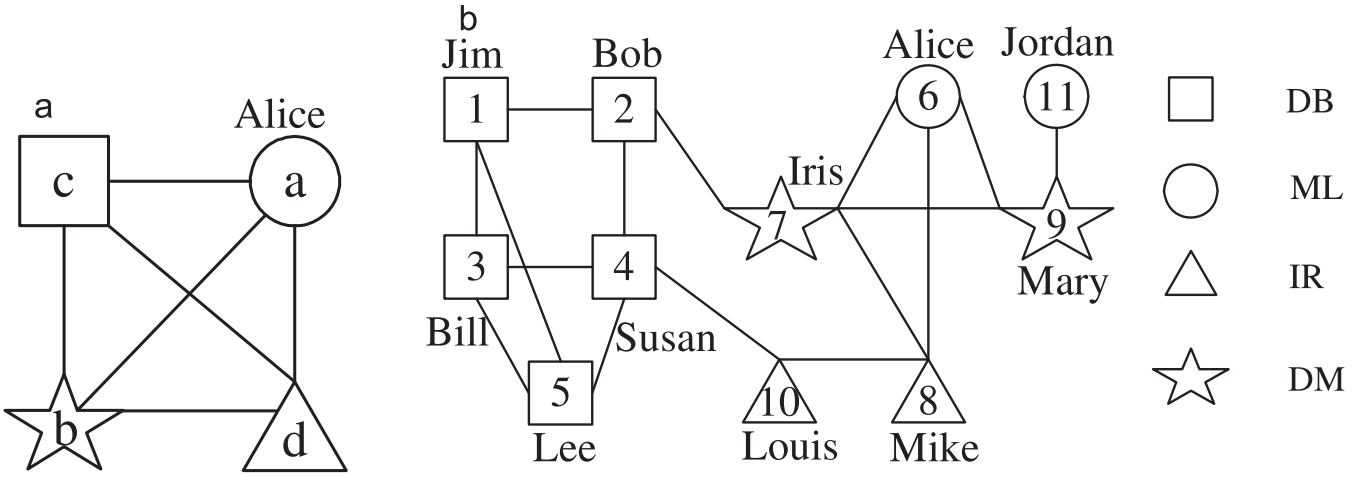

Fig. 1. An illustrative example. (a) A sample query. (b) A sample data graph.

On the contrary, with approximate matching, the induced subgraph by {2, 6, 7, 8} is a good answer.

There are several challenges in tackling the approximate graph matching problem. First, a query graph may not contain the details of every node, since users may not know all the details (often that is why the query is asked). For example, in Fig. 1(a), only node *a* carries both label information (i.e., the name "Alice") and attribution (i.e., the area "machine learning"), while other query nodes contain only attribution information. Second, an edge in the query may not necessarily have a match in the data graph. For example, the edge (*a*,*c*) in Fig. 1(a) does not match any edge in Fig. 1(b). In this case, we want to find the best path that connects the corresponding nodes of *a* and *c* in the data graph (e.g., the path 6-7-2 in Fig. 1(b)). Third, we cannot simply find the best matching edge/path for an individual edge in the query graph, but need to consider the overall query graph that is to be matched, which is a difficult optimization problem.

In this paper, we model the approximate graph matching problem as an optimization problem. We prove that the problem is NP-hard and propose an index-based greedy algorithm that are both effective and efficient. Our proposed method consists of three phases:

Indexing: In the first phase, we develop an index that is both structure-aware and attribution-aware, namely *SA-Index*. At the first level of SA-Index, we index on the structure similarity and attribute similarity of the data graph, which partitions the data graph into clusters of nodes that are close to one another both in structure and in attribute values. Then at the second level, we further encode the connectivity among the clusters in a hash table.

Best path computation: In the second phase, we define the set of best paths[3] in the data graph for matching each edge in the query graph. Then, we utilize SA-Index to efficiently compute this set of best paths.

Partial answer connection discovery: In the third phase, we propose two greedy algorithms to compute the best matching answer graph, by selecting the best paths computed from the second phase.

We evaluate the performance of our method using a set of large real datasets. Our experimental results show that SA-Index is efficient to construct, and query performance using the index is efficient and scalable to different sizes of data graphs and query graphs. Our method is over an order of magnitude faster than G-ray [3] and more specific than the memory-based implementation of TALE [4].[4] We also use various quality measures to show that the answer graphs obtained by our method are of high quality by case studies.

Paper organization: The rest of the paper is organized as follows. Related work is discussed in Section 2. We give a formal definition of approximate graph matching problem in Section 3. Sections 4, 5.1 and 5.2 describe the three phases of our solution, respectively. We report experimental results in Section 6. Finally, we conclude our work in Section 7.

2. Related work

The popularity of graph model has attracted much interest in graph research such as querying graph databases [1,5–17], approximate subgraph matching [2–4], frequent subgraph mining [18–22], and correlation subgraph discovery [23–26]. In this section, we review the works related to sub/graph matching.

Ullman [27] introduced a basic algorithm for sub/graph isomorphism (i.e., exact matching). Tsai and Fu [28] studied error-correcting isomorphisms of attributed graphs for image analysis. They extended their pattern deformation model so that numerical attributes and probability distribution can be introduced into primitives and relations in non-hierarchical relational graphs.

[3] Note that an edge can also be regarded as a path.

[4] We changed the disk-based implementation of TALE (i.e., going through PostgreSQL) provided by authors to memory-based implementation for fair comparison.

Cordella et al. [29] proposed a new algorithm which can handle sub/graph isomorphism test efficiently in large graphs. Tong et al. [3,30] proposed an algorithm to find best-effort matching in a large graph based on random walk.

Another type of related works use index to efficiently find matches in a large data graph. TALE [4] utilize degree information and neighborhood connectivity to filter unmatched node pairs. Zhang et al. [31] proposed another indexing approach based on graph distance. Their methods are not efficient enough to handle graphs with up to millions or billions of nodes and edges. In the preprocessing stage of [31], given a data graph, they generate a set of intersecting subgraphs from pairs of nodes within a predefined distance threshold and a radius threshold. The number of intersecting subgraphs may increase significantly when the size of data graph grows. Tian et al. [4] used a maximum weighted bipartite graph matching algorithm during the stage of matching important nodes, which can be costly if the bipartite graph is large. Moreover, these approaches are either structure-based indexes or attribute-based indexes. A strong link between structure and attribute has not been studied previously. On the contrary, our work focuses on indexing in a large attributed graph and we index both the structure and attribution information of the graph.

3. Preliminaries

3.1. Notations

We denote a data graph (or simply a graph) as a 4-tuple, $G=(V,E,l_G,\Sigma)$, where V is the set of nodes of G; $E\subseteq V\times V$ is the set of edges of G; and $l^G: V\rightarrow\Sigma$ is a function that maps a node to a label (e.g., name of people, name of protein, etc.) in Σ, and Σ is the alphabet set. In addition, each node in a graph is associated with an attribute, which is denoted as A^G. For example, in Fig. 1(b), each node represents a researcher and is associated with attribute "research area" with values "database", "data mining", "information retrieval", "machine learning", and so on.

For easy representation, throughout the paper, we use $V(G)$ and $E(G)$ to denote the node set and edge set of a graph G. For a node $v\in V(G)$, we use $A^G(v)$ to denote the value of its attribute, and $l^G(v)$ its label. We use $p\in G$ to denote a simple path p that appears in G. In addition, we use u_e and v_e (u_p and v_p) to denote the two end nodes of an edge $e\in E(G)$ (a path $p\in G$), respectively.

A query graph (or simply a query) is defined as $Q=(V,E,l^Q,\Sigma\cup\{\lambda\})$, which is the same as a data graph except that $l^Q: V\rightarrow\Sigma\cup\{\lambda\}$, where λ is the *null* string. For example, in Fig. 1(b), each node is assigned with a label which represents a person's name (e.g., $l^G(1)$="Jim"); however, in the query shown in Fig. 1(a), the label of a node may be missing (e.g., $l^Q(d)=\lambda$).

Table 1 gives the notations used throughout the paper.

Table 1
Notations.

Notation	Description
Q	A query graph
G, G_a	A data graph/an answer graph
$V(G)$, $E(G)$	The node/edge set of G
$A^G(v)$, $l^G(v)$	The value of attribute/label of v in G
$p\in G$	A path p that appears in G
u_e, v_e	The two ending nodes of an edge e
u_p, v_p	The two ending nodes of a path p
$P^*(G)$	The partition-based index of G
HT	The hash table
G_s	The summary graph of G
$h(v)$	Homogeneity of a node v
$(\varphi(S))\phi(S)$	(Generalized) conductance of node group S
$G'\approx Q$	A graph G' approximately matches Q

3.2. Problem definition

In this subsection, we present our problem formulation of approximate graph matching as follows.

Definition 1 (*Approximate graph matching*). Let Q and G' be the query graph and the data graph (or G' be a subgraph of the data graph). An approximate matching from Q to G', denoted as $G'\approx Q$, is defined as a bijection $f:\widehat{V}(Q)\subseteq V(Q)\leftrightarrow\widehat{V}(G')\subseteq V(G')$ s.t. $\forall u\in\widehat{V}(Q)$, $f(u)\in\widehat{V}(G')$, $A^Q(u)=A^{G'}(f(u))$, and $(l^Q(u)=l^{G'}(f(u))$ or $l^Q(u)=\lambda)$, and $\forall u,v\in\widehat{V}(Q)$, where $(u,v)\in E(Q)$, there is a path from $f(u)$ to $f(v)$ in G'.

Example 2. In the examples shown in Fig. 2, the query graph is matched to two subgraph G_1 and G_2 of data graph G in Fig. 1(b). The dashed lines indicate the matched nodes in query graph and data graph. Note that not all nodes are required to be mapped, for instance, as shown in Fig. 2(b), node c in query Q has no mapping in G_2. Also note that an edge in the query graph could be mapped to a path in the data graph, e.g., in Fig. 2(a), the edge a–c in the query Q is matched to the path 6-7-2 in G_1.

We now define the problem of processing an approximate graph query in an attributed graph as follows.

Problem 1 (*Approximate graph matching problem*). Given a query Q and a graph G, the problem of approximate graph matching (*AGM*) is to find the best matching answer graph G_a as defined by the following equation:

$$G_a=\operatorname*{argmax}_{G'}\ \mathtt{sim}(Q,G'), \tag{1}$$

where $G'\subseteq G$ and $G'\approx Q$, and $\mathtt{sim}(Q,G')$ is the degree of matching from Q to G'. If $\mathtt{sim}(Q,G')=1$, then Q is graph isomorphic to G'.

The degree of matching between two graphs can be measured by various metrics. For example, Montes-y-Gómez et al. [32] defined graph similarity as the number of common edges shared by two graphs. The term "graph edit distance" covers a class of metrics that measure the degree of matching between two graphs. Its variants have been studied theoretically [33] as well as applied in domains as diverse as image retrieval [34] and querying graph database [7]. The choice of different metrics for the degree of matching is orthogonal to this work. In the following sections, we propose an approximate graph

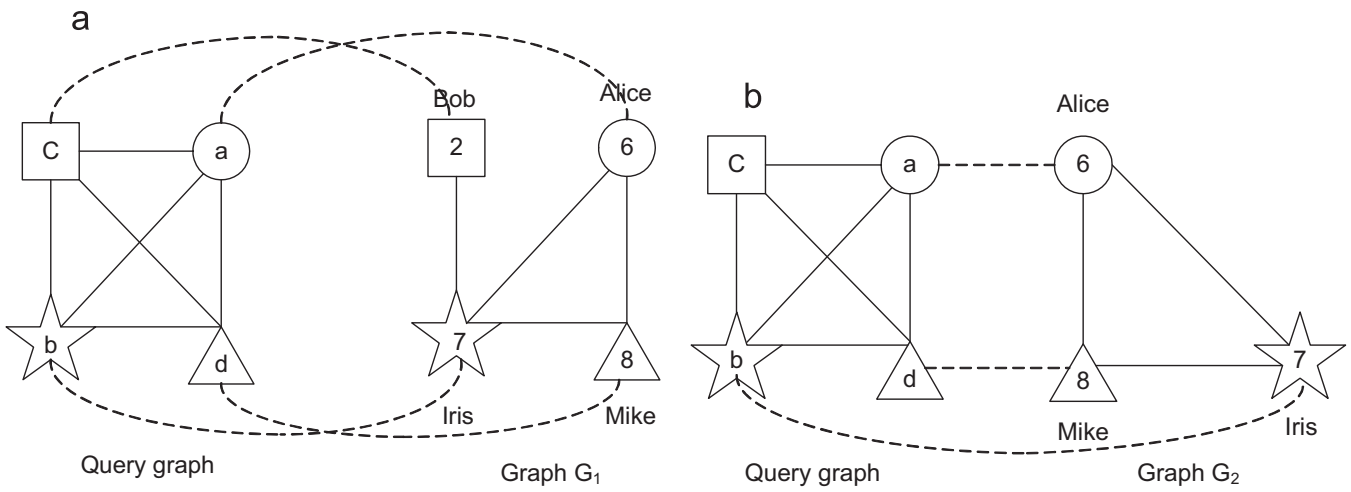

Fig. 2. Examples of approximate graph matching.

query algorithm that takes a general metric as input. Furthermore, we would discuss briefly how our work supports the two metrics "maximum common edges" and "graph edit distance" in Appendix A.

3.3. Hardness of AGM problem

We analyze the hardness of the simplified version of our AGM problem, i.e., the decision problem of AGM. A reduction from the γ-quasi clique problem establishes the intractability of that problem.

Theorem 1 (*NP-completeness*). *Given a query Q, a graph G, and a value* $0 < \delta \leq 1$, *the decision problem of AGM, i.e., to determine whether there is a subgraph* G_a *of G such that* $\mathtt{sim}(Q,G_a) \geq \delta$ *is NP-complete.*

Proof (*Sketch*). It is easy to verify that the decision problem of AGM is in NP. Next, we show the NP-hardness of the decision problem of AGM by reducing the problem of γ-quasi clique to our problem. The problem of γ-quasi clique is defined as follows: Given a graph G, an integer K, and $0 < \gamma \leq 1$, is there a subgraph $\tilde{G} = (V(\tilde{G}), E(\tilde{G}))$ of G with K or more nodes such that $2|E(\tilde{G})|/|V(\tilde{G})|(|V(\tilde{G})|-1) \geq \gamma$ (if $\gamma = 1$, then $\tilde{G}$ is a clique). The problem of γ-quasi clique with $0 < \gamma \leq 0.5$ and $\gamma = 1$ is shown to be NP-complete [35,36]. Although it is not proved whether the γ-quasi clique problem of $0.5 < \gamma < 1$ is NP-complete or not, [36] infer the γ-quasi clique problem is NP-complete. Now, given G, K and γ, we build a complete graph Q such that Q has K nodes. Clearly G has a subgraph G_a such that $\mathtt{sim}(Q,G_a) \geq \delta$ if and only if G has a γ-quasi clique. The relationship between γ and δ is determined by the type of degree of matching metric used. In Table 2, we give some examples of different types of degree of matching metric and the corresponding relationship between γ and δ. Thus, the reduction consists of constructing a complete graph with K nodes, which can clearly be done by using logarithmic auxiliary space. □

Example 3. Consider the graph G shown in Fig. 3. Suppose that $\gamma = \frac{2}{3}$ and $K=4$. Now we build a complete graph Q with four nodes. Assume that the input degree of matching metric uses number of common edges [32].

Table 2
Examples of degree of matching metrics and relationship between γ and δ.

Metric	Common edge [32]	Edit distance [7]	Our experiment
Relation	$\gamma = \delta$	$\gamma = \delta$	$\gamma = \min\left(\frac{1}{\alpha}\delta, 1\right)$

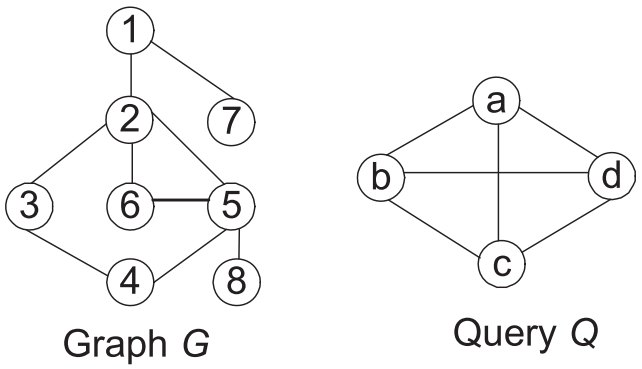

Fig. 3. An example of reducing from γ-quasi clique to decision problem of approximate graph matching.

Then the subgraph G' induced by the node set {2, 3, 4, 5} approximately matches the query Q with $\mathtt{sim}(Q,G') = \frac{2}{3}$ (number of common edges is four and the total number of query edges is six). Hence, G' is a "yes" instance to approximate graph matching with $\delta = \frac{2}{3}$. This indicates the subgraph G' induced by the node set {2, 3, 4, 5} is also a "yes" instance to γ-quasi clique with $\gamma = \frac{2}{3}$.

Since the decision problem of AGM is NP-complete, its optimization problem—our AGM problem is also NP-hard. Thus, for finding an optimal approximate graph matching to a given query, we develop some heuristic approaches in the following sections.

4. SA-Index

In this section, first we propose to use a novel *SA-Index* (structure- and attribute-aware index) to summarize the

connectivity, structure similarity and attribute similarity information of a data graph in Section 4.1. SA-Index consists of two levels: partition-based index and connectivity-based index. Next, in Section 4.2, we analyze the advantages of SA-Index. Finally, we discuss the efficient computation of SA-Index, see Section 4.3.

4.1. The notion of SA-Index

Partition-based index: Intuitively, at the first level of SA-Index, we partition the data graph into subgraphs based on its structure similarity (i.e., nodes that are highly connected with one another form a group) and attribute similarity (i.e., nodes that are closely related to one another in attribute values are condensed together), denoted as *partition-based index*.

In order to utilize the structure similarity, we introduce a notion of "conductance" defined by Leskovec et al. [37]. Its generalized version would form the core of our first level of SA-Index.

Definition 2 (*Conductance, Leskovec et al. [37]*). Given a graph G represented as an n-by-n matrix M, the conductance of a set of nodes S is defined as

$$\phi(S) = \frac{\sum_{u\in S, v\notin S} M_{uv}}{\sum_{u\in S} d(u)}, \tag{2}$$

where $d(u)$ is the degree of the node u in G.

It is commonly noticed that conductance captures the "gestalt" notion of a community. A group of nodes S is a good community if it is separated from the remaining of the graph by a low-conductance cut. In particular, the use of conductance as an objective function returns a good partitioning that facilities connectivity discovery in online query since optimizing conductance function also maximizes intra-partition connections and minimizes inter-partition connections.

However, utilizing a single metric such as conductance has a drawback: only structure information is adopted to measure the goodness of a group of nodes S; other information such as attribution information is ignored. In parallel to structure information, attribution information has recently been rediscovered in graph clustering/partitioning: people may seek to find a good community based on attribute similarity. Attributes also make a more targeted search in a large data graph.

To address the aforementioned issues, now we introduce a concept of *homogeneity*, to capture the intuition of what it means for a set of nodes to be attribute similar. Essentially, given a graph, homogeneity of a node v measures how many nodes in the neighborhood of v share the same attribute values. As a simple example, a group of nodes that all have the value "database" for their "research area" attribute would be considered very homogeneous; while a group of nodes each with a different value for the "research area" attribute would be considered very heterogeneous.

Mathematically, given a set of nodes $S \subseteq V(G)$, the homogeneity of a node $v \in S$, $h(v, S)$, is defined as

$$h(v,S) = -\sum_{i\in A_v^S} p(i|v)\log p(i|v), \tag{3}$$

where $A_v^S = \bigcup_{u\in S} M_{uv} A^G(u)$ is the set of attribute values in the neighborhood of v, $p(i|v) = \sum_{u\in S} M_{uv}\mathcal{I}(A^G(u) = i)/d(v)$ denotes the fraction of nodes in the neighborhood of v that have attribute value i, and $\mathcal{I}(\cdot)$ is an indicate function which returns one if the expression is true, and zero otherwise. Clearly, a node v with higher value of $h(v, S)$ is quite heterogeneous.

In the following, we explore a strong link between conductance and homogeneity to collapse nodes which are similar in terms of both structure and attribute into a group. Consider a running example in Fig. 4 which shows several possible partitions. Fig. 4(a) shows one partition according to Eq. (2): {1, 2, 3, 4, 5} and {6, 7, 8, 9, 10, 11}. People in one group are highly connected with one another, which leads to a small conductance. However, people inside the same partition may have diverse research interests: in the group {6, 7, 8, 9, 10, 11}, research interests of people distribute equally over "data mining", "information retrieval" and "machine learning". According to Eq. (3), another attribute-based partitioning is shown in Fig. 4(b). People work in the same research area are grouped together such as {1, 2, 3, 4, 5}, {7, 9}, {8, 10} and {6, 11}. Unfortunately, using such a way to partition make nodes in the same group quite isolated and nodes in different groups highly connected. Therefore, users may expect a partitioning that weighs both research interests and interactivities, as shown in Fig. 4(c).

As a start, a naive approach is to perform a hierarchical partitioning. That is, one may perform partitioning based on one dimension (i.e., a single conductance or homogeneity metric), and then refine the partitioning based on another dimension of information. This approach raises such a problem: Shall we perform structure-based partitioning first and then attribute-based partitioning? Does the order of

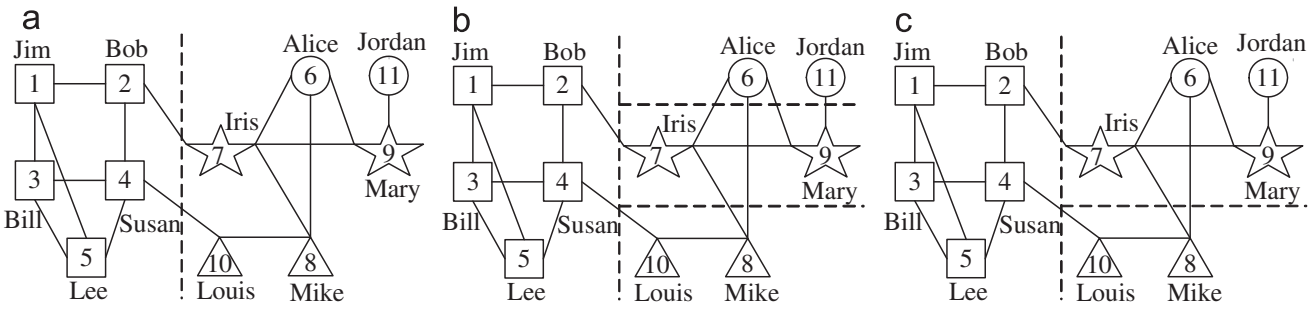

Fig. 4. A running example. (a) Structure-based partition. (b) Attribute-based partition. (c) SA partition.

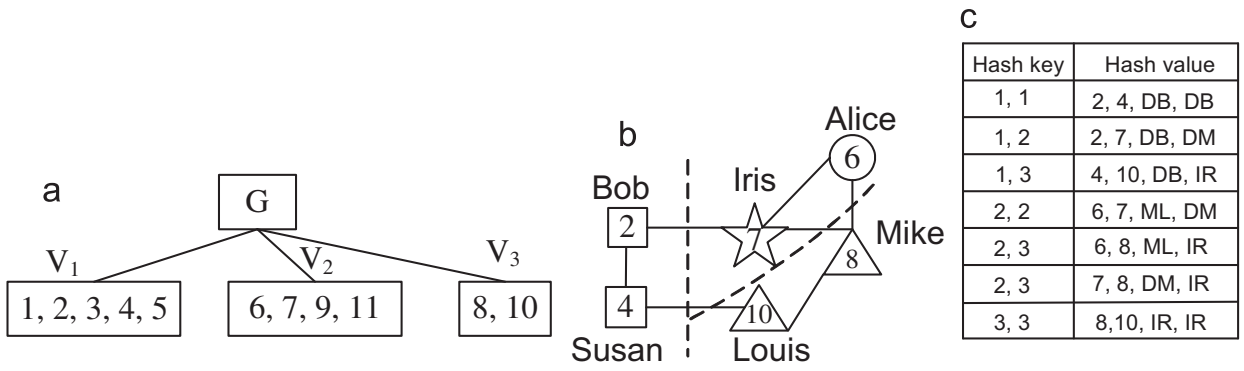

Fig. 5. An example of the SA-Index. (a) Partitioning. (b) Summary graph. (c) Hash table.

hierarchical partitioning affect the final results? If the answer is yes, how to determine the order becomes another challenging problem. Besides, an assumption behind this approach is that structure information is independent with attribute information, which is not the truth in most cases.

Hence, instead of using a hierarchical partitioning, we make use of both structure and attribute together to perform partitioning. A nature idea of our approach is that a set of nodes S is a good community if splitting S into a group results in lower conductance and higher homogeneity (lower values of $h(v,S)$ for $v \in S$). Therefore, we embed the homogeneity property into conductance, and propose a generalized conductance $\varphi(S)$ for a set of nodes S as follows.

$$\varphi(S) = \frac{\sum_{u\in S, v\notin S} A_{uv} h(u,S)}{\sum_{u\in S} d(u) h(u,V(G))}, \quad (4)$$

where the numerator indicates the conductance and homogeneity after condensing S into a group; and the denominator indicates the conductance and homogeneity before separating S from a graph G. A set of nodes S is a good community if S has a small value of $\varphi(S)$.

With Eq. (4), the first level of SA-Index of a data graph G, partition-based index $P^*(G)$, is defined as: $P^*(G) = \text{argmin}_{P(G)}\{\sum_{V_k\in P(G)}\varphi(V_k)\}$, where $P(G)=\{V_1, V_2, \ldots, V_k\}$ denotes any partitioning over G such that $(\bigcup_{i=1}^{k} V_i) = V(G)$ and $V_i \cap V_j = \emptyset$, where $i \neq j$ and $1 \le i,j \le k$.

Connectivity-based index. So far we have formally introduced the first level of SA-Index. Now, given the data graph G and $P^*(G)$, we further present the details of the second level of SA-Index, which provides a path-view over connectivity among sub-partitions in $P^*(G)$, denoted as *connectivity-based index*. More specifically, we introduce a summary graph on the basis of a fundamental terminology "boundary nodes", to keep the inter-partition information as a complementary to $P^*(G)$.

Definition 3 (*Boundary nodes and summary graph*). Given a group of nodes V_i, let $\chi(V_i)$ denote the set of nodes in V_i that are connected to the nodes outside V_i. Then, $\chi(V_i)$ is called boundary nodes set of V_i. Given a partitioning $P(G)$ over the data graph G, its boundary node set, is defined as: $\chi(P(G)) = \bigcup_{V_i\in P(G)}\{\chi(V_i)\}$. Then the summary graph G_s is a subgraph of G induced by the node set $\chi(P(G))$.

The summary graph G_s, provides a lossless information of direct connections via edges among boundary nodes. In addition, G_s is unconnected if there exists a pair of boundary nodes that are in the same partition and are not connected via edges. However, using G_s only may still suffer low speed-up since we need to scan G_s multiple times to find those connections among boundary nodes in the on-line query phase. To deal with it, we pre-compute some partial path information to prune the search space in on-line query. Specifically, we utilize the fundamental data structure *hash table* to provide a path view over inter-partition connections. Thus, both the summary graph G_s and the hash table HT together comprise the connectivity-based index. In the hash table HT, the hash key of each tuple is a pair of partition ID s of boundary nodes and the hash value consists of the set of nodes (attribute values are also included) in the shortest path connecting two partitions where the pair of boundary nodes are located.

In all, the SA-Index proposed in this work is a three-tuple relation $\langle P^*(G), G_s, HT \rangle$, where $P^*(G)$ is a partition-based index, G_s is a summary graph, and HT is a hash table that stores partial path information among sub-partitions. We use the following example to further illustrate the concept of SA-Index.

Example 4. As shown in Fig. 5, based on the structure information and homogenous property, we partition the data graph in Fig. 1(b) into three parts: $P(G)=\{\{1, 2, 3, 4, 5\}, \{6, 7, 9, 11\}, \{8, 10\}\}$. Next, we compute the boundary node set $\chi(G)$. In this example, $\chi(G) = \{2,4,6,7,8,10\}$. The summary graph, induced by the boundary node set, is shown in Fig. 5(b). Finally, we encode the connectivity information among partitions in a hash table, which is shown in Fig. 5(c).

4.2. Why SA-Index?

Why partition-based index. At the first level, we partition a data graph based on its structure and attribute. As we know, the partitioning of a given data graph is not unique. Hence, a problem need to address is: why we need a partitioning $P^*(G)$ rather than other partitionings?

Lots of partitioning algorithms have been proposed such as METIS [38], edge betweenness [39], spectral clustering [40,41], conductance [37], attribution similarity [42], and modularity [43]. In the following, we illustrate

the reason of using $P^*(G)$ by identifying restrictions of other approaches.

First, all of the above partitioning works group nodes either based on structure only or based on attribute only. Consider corresponding examples from other areas, one often makes use of both structure and attribute: In the area of image analysis and compute vision, one may adopt the joint learning of object parts and attributes to perform object detection; in the area of data clustering, SAclustering [44], also proposes a structure-based and attribute-based approach to improve the quality of clustering. However, in graph partitioning, a strong link between structure and attribute has not been previously explored. Furthermore, during query processing, both attribute equality and structure similarity are of great importance. Therefore, a good partitioning also needs to take both structure and attribute into consideration.

Next, though in SAclustering [44], nodes in one cluster are close to one another either in structure or in attribute, the inter-cluster information, may become too complex to be encoded concisely. This may incur additional processing time since in query processing we also need to access inter-partition nodes/edges to discover connectivity among partial answers in different partitions.

As an alternative, with the $P^*(G)$, we could quickly identify the set of partitions (subgraphs) that contain candidate nodes for answer graphs on the basis of attribution equality and structure similarity. In addition, partition-based index prunes the search space for partial answers from the entire data graph to a set of subgraphs. Finally, as $P^*(G)$ takes the generalized conductance as the objective function, it can also produce a low cut among different partitions and reduce the encoding cost of the second-level index.

Why need G_s and HT. It is straightforward that the connectivity among nodes in different partitions is important for query processing. One may simply choose the node or edge as unit of connectivity-index, i.e., index the set of inter-partition nodes/edges. This method has the benefit that the index size grows linearly with the number of nodes in the data graph, but suffers from low speed-up for query processing since one may still be required to find connections among inter-partition edges/nodes in on-line query processing.

In response to the above issues, we use the *graph-theoretic distance* to capture the connectivity among inter-partitions and encode a set of paths into a hash table (denoted as *HT*). One can identify connections among partitions quickly with the use of *HT*. In addition, as the number of boundary nodes is not large in reality, the size of connectivity-based index ($|HT|$ and $|G_s|$), is still affordable even if storage cost is considered as an important issue. Last but not the least, *HT* can be used to prune search space during the procedure of discovering connections among candidate answer nodes.

4.3. Efficient index construction

In this subsection, we study efficient approaches to compute the SA-Index. We first introduce a heuristic algorithm to fast compute a partitioning of a data graph with respect to the definition of $P^*(G)$. Next, we present the procedure of summary graph and hash table computation.

Our heuristic partitioning algorithm follows the traditional "extended local search" paradigm: a node could be swapped between two partitions again and again to correct previous bad assignments. In another word, in each iteration, we search for a better assignment that reduces the value of Eq. (4). The details are outlined in Algorithm 1. Initially, we just randomly bisect graph G into two partitions (line 1). Next, `partition(G)` iteratively searches a locally best assignment for each node. In other words, in each step, for each node v, we enumerate all possible new partitionings by assigning v from its original partition i to another existing partition j or a new empty partition (lines 3–7). Note that in line 5 we insert an empty node set $V_\emptyset$ into the current partition $P(G)$, so that at each iteration we also allow v to form a partition by itself.

Algorithm 1. Partition-based index construction `partition(G)`

Input: A graph G
Output: a partitioning $P(G)$
1: $P(G) = \{V_1, V_2\}$;
2: *flag* = `false`;
3: **for each** $V_i \in P(G)$
4: **for each** node $v \in V_i$
5: $P(G) = P(G) \cup \{V_\emptyset\}$, where $V_\emptyset$ is an empty set;
6: **for each** $V_j \in P(G)$, where $j \neq i$
7: Get $P^j(G) = \{V_1^j, \ldots, V_k^j\}$ by moving v to V_j;
8: $\tau = \arg\max_j \{\sum_{V_k \in P(G)} \varphi(V_k) - \sum_{V_k^j \in P^j(G)} \varphi(V_k^j)\}$;
9: **if** $\sum_{V_k \in P(G)} \varphi(V_k) > \sum_{V_k^\tau \in P^\tau(G)} \varphi(V_k^\tau)$
10: $P(G) = P^\tau(G)$;
11: *flag* = `true`;
12: **if** *flag* = = `true`
13: **repeat** lines 2-11;
14: **else**
15: **return** $P(G)$;

Then in line 8, among those new partitionings, we find the best target partition τ, such that the new partitioning produced by assigning v to partition τ obtains the minimal size of generalized conductance among all other partitionings produced by other assignments. If the value of generalized conductance of the new partitioning $P^\tau(G)$ (as defined by Eq. (4)) is smaller than that of the current partitioning $P(G)$, we replace $P(G)$ by $P^\tau(G)$ and repeat the process (lines 9–11). If $P(G)$ is ever replaced in the previous iteration, *flag* is set to `true` in line 11 and we start a new iteration of lines 2–11. Otherwise, $P(G)$ has the local minimal value of generalized conductance and is returned (lines 14–15).

Complexity analysis. Assume i is the number of iterations, then the overall time complexity of our partitioning algorithm is $O(ik(n+m))$, where n and m are the number of nodes and edges in G, and k is the number of partitions in $P(G)$. In the worst case (every node forms a partition by itself), $i=n$ and $k=n$. In practice, however, i and k are both very small. The average values of i and k are 39. Thus, our

partitioning algorithm is efficient in most cases, as also verified by our experiment.

After investigating how to partition the data graph G based on its structure and attribute similarity, now we discuss the details of computing the connectivity-based index, which is a concise representation of inter-partition connections.

Algorithm 2. Connectivity-based index construction

Input: A graph G and its partitioning $P^*(G)$
Output: a connectivity-based index $\langle G_s, HT \rangle$
1: compute G_s based on $P^*(G)$ and G;
2: order the node set $V(G_s)$ as $v_1 \prec v_2 \ldots \prec v_n$;
3: $H(G_s) = \emptyset$;
4: **for each** $(u,v) \in V(G_s) \times V(G_s), u \prec v$
5: compute the set of shortest path P_{uv} connecting u and v in G;
6: **for each** $p \in P_{uv}$
7: compute the set of features of path p including path length, weighted path length, and attribute values of nodes;
8: compute $\texttt{value}(p)$ based on the path features;
9: $\texttt{key}(p)=\texttt{make\_pair}(i, j)$, where $u \in V_i$ and $v \in V_j$
10: $T_h=\texttt{make\_pair}\ (\texttt{key}(p), \texttt{value}(p))$;
11: hash T_h into HT;
12: **return** HT and G_s;

Given the partition-based index $P^*(G)$ and the data graph G, Algorithm 2 first computes the summary graph G_s (line 1). As we are studying on undirected graphs, the set of shortest paths from u to v is equal to the set of shortest paths from v to u. Hence, instead of computing all pairs of shortest paths, we compute shortest paths of pairs of node (u, v) when $u \prec v$ (lines 4–5). Next, we hash the set of paths into HT where each hash value is computed according to a set of path features (lines 7–11).

Complexity analysis. The time complexity of Algorithm 2 is dominated by line 5: the computation of shortest paths set. As we have known, a number of shortest paths that have the same source can be computed in $O(n \log n+m)$ [45], where n and m are the number of nodes and edges in a given graph. Hence, in the worst case, the complexity of Algorithm 2 is $O(|V(G_s)||V(G)|\log|V(G)|+|V(G_s)||E(G)|)$.

5. On-line query processing

In this section, we propose an algorithm to process approximate graph matching queries using the SA-Index. In Section 5.1, we describe a method to find the set of best paths in the data graph for matching the set of query edges. Then in Section 5.2, we present two greedy algorithms to construct the best query answer from the set of best paths.

5.1. Computation of the best paths

In Definition 1, we allow the matching of an edge in the query graph with a path in the data graph, in the case when the query edge does not match any edge in the data graph. Although a path does not directly connect two nodes as does an edge, a good path can provide good insight of how the two nodes are connected. This is particularly useful when there is no edge in the data graph that matches a query edge. In fact, the user who specifies a query edge may not know the existence of such an edge (otherwise he probably will not ask the query), and the user may want to know the connection between the two nodes rather than a *strict* direct connection as by an edge.

We further explain the concept by the following example.

Example 5. Consider the edge (a, c) in Fig. 1(a) which indicates the connection from a person Alice in machine learning area to another person in database area. However, in the corresponding data graph in Fig. 1(b), there is no direct connection (i.e., an edge) from Alice to any DB person. Instead, Alice is connected to a DB person via a set of paths such as 6-7-2, 6-8-7-2, 6-7-2-4, and so on. These paths reveal to us how Alice is connected to a DB person.

As shown in Example 5, a given query edge may have a set of paths that can match it. Therefore, different edge-to-path matching may affect the quality of the answer graph returned. In the following, we discuss how we select the set of best paths for a given query edge.

Given a query edge e and two nodes x and y in a data graph G, where $A^G(x)=A^Q(u_e)$ and $A^G(y)=A^Q(v_e)$, the "best path" $p^* \in G$ that matches e and connects x and y is defined as

$$p^*(x,y,e) = \underset{p \in G}{\operatorname{argmax}}\ \texttt{sim}(e,p)$$

$$\text{subject to } u_p = x \quad \text{and} \quad v_p = y, \tag{5}$$

where $\texttt{sim}$[5] is the same input metric to measure the degree of matching from a query graph to a data graph.

The set of best paths matching query edges, is further defined as

$$P = \bigcup_{e \in E(Q)} \bigcup_{\substack{x,y \in V(G) \\ A^G(x) = A^Q(u_e) \\ A^G(y) = A^Q(v_e)}} p^*(x,y,e). \tag{6}$$

Algorithm 3. Finding the best paths $\texttt{edgetopath}(P^*(G), G_s, HT, Q, \texttt{sim})$

Input: Query Q, index $\langle P^*(G), G_s, HT \rangle$, and metric $\texttt{sim}$
Output: A path set, P
1: $P = \emptyset$;
2: **for each** edge $(u, v) \in E(Q)$
3: **for each** $x,y \in V(G)$, where $A^G(x)=A^Q(u)$ and $A^G(y)=A^Q(v)$
4: Let G_i (G_j) be the partition that x (y) belongs to;
/* compute p^* where p^* is the solution to Eq. (5) */
5: Let S denote the search space;
6: **if** G_s is connected
7: $S = G_s$;
8: **else**
9: $S = G_s \cup HT$;
10: $S = S \cup G_i \cup G_j$;
11: $p^* = \text{argmax}_{p \in S}\{\texttt{sim}(e,p)\}$;
12: $P = P \cup p^*$;
13: **return** P;

[5] Here we consider a path or an edge as a graph.

Now we discuss the details of computing P as given in Eq. (6). In Algorithm 3, given a query edge (u, v), the first step is to compute the set of nodes in data graph whose values of attributes are compatible to u and v (lines 2 and 3). Next, we compute the partitions where x and y reside, i.e., G_i and G_j (line 4). No matter whether x and y are in the same partition or not, it is required to compute the best path p^* that matches e by examining both inter- and intra-partition concepts. In order to compute those inter-partition connections, if G_s is connected, we only need to scan G_s for the best connections since G_s has encoded connections via edges among boundary nodes; otherwise, we can augment the search space to $G_s \cup HT$ for best inter-partition connections (lines 6–9). Finally, we scan all the possible combinations of both intra- (i.e., G_i and G_j) and inter- (G_s and HT) connections for the best path (lines 10–11).

5.2. Partial answer connection discovery

Algorithm 4. Greedily select path $\texttt{greedy}(Q, P, \texttt{sim})$

Input: query Q, metric $\texttt{sim}$, a candidate partial answer set P
Output: An answer graph G_a

1: $\texttt{type} = 1, G_a^1 = \emptyset$:
2: **while** not all the query edges are processed
3: $p_0 = \texttt{argmax}_{\substack{p \in P \\ p \notin G_a^1}} \{\texttt{sim}(Q, G_a^1 \cup \{p\}) - \texttt{sim}(Q, G_a^1)\}$, $G_a^1 \cup \{p\}$ $\texttt{connected}$;
4: $G_a^1 = G_a^1 \cup \{p_0\}$, mark an edge mapped processed;
5: $\forall p \in P$, if p and p_0 match to the same query edge
6: $P = P \backslash \{p\}$;
7: $\texttt{type} = 2$, $G_a^2 = \emptyset$:
8: **while** number of processed query edges $< |E_q| - 1$
9: $p_0 = \text{argmax}_{\substack{p \in P \\ p \notin G_a^2}} \{\texttt{sim}(Q, G_a^2 \cup \{p\}) - \texttt{sim}(Q, G_a^2)\}$;
10: $G_a^2 = G_a^2 \cup \{p_0\}$, mark an edge mapped processed;
11: $\forall p \in P$, if p and p_0 match to the same query edge
12: $P = P \backslash \{p\}$;
13: $p_0 = \text{argmax}_{\substack{p \in P \\ p \notin G_a^2}} \{\texttt{sim}(Q, G_a^2 \cup \{p\}) - \texttt{sim}(Q, G_a^2)\}$, $G_a^2 \cup \{p\}$ $\texttt{connected}$;
14: **if** p_0 not empty
15: $G_a^2 = G_a^2 \cup \{p_0\}$;
16: **else** let G_a^2 be the maximum connected part of G_a^2;
17: **return** $\text{argmax}\{\texttt{sim}(Q, G_a^1), \texttt{sim}(Q, G_a^2)\}$;

In this section, we compute the answer graph that matches the input query graph by greedily selecting the best paths that are computed by Algorithm 3. We design two types of greedy algorithms as follows.

The first greedy algorithm starts from an empty answer graph G_a^1, and iteratively in step k, inserts into G_a^1 the path p_k that matches an unmatched query edge (u, v), and that has common nodes with G_a^1, and that maximizes the following equation.

$$p_k = \text{argmax}_{\substack{p \in P \\ p \notin G_a^1}} \{\texttt{sim}(Q, G_a^1 \cup \{p\}) - \texttt{sim}(Q, G_a^1)\}$$

$$\text{subject to } (G_a^1 \cup \{p\}) \text{ is connected.} \quad (7)$$

The algorithm stops when all the edges in the query graph are matched.

The second greedy algorithm is similar to the first one, except that in each non-final step k, it adds path p_k without taking the connectivity constraint into consideration (connectivity constraint means that p_k should share some common node with G_a^2). In other words, it selects a path p_k that matches an unmatched query edge (u, v) and maximizes

$$p_k = \text{argmax}_{\substack{p \in P \\ p \notin G_a^2}} \{\texttt{sim}(Q, G_a^2 \cup \{p\}) - \texttt{sim}(Q, G_a^2)\}. \quad (8)$$

The second greedy algorithm only takes care of the connectivity constraint in the final step. For the last query edge, we would select the path p by solving Eq. (7). If the path p exists, we just insert it into the answer graph G_a^2; otherwise, we compute the maximal connected component of G_a^2.

Even though both of the two greedy algorithms may not perform well in some situation, there is at least one of them which is not too far away from optimum. Hence, by selecting the one with higher similarity score as answer graph, our algorithm has a decent answer quality, as verified by the experimental studies (See Section 6).

To sum up, our graph matching algorithm is summarized as follows: First, we use Algorithm 3 to find the set of best paths; Second, we compute the answer graph by Algorithm 4.

Complexity analysis. We now analyze the time complexity of the approximate graph matching algorithm. Assume the time complexity to compute Eq. (5) is $O(\Upsilon(|V(G)|, |E(G)|))$.[6] In Algorithm 3, for each node x, the worst case is to find the set of best paths from node x to other nodes in different partitions, which needs $O(\Upsilon(|V(G_i \cup G_j \cup G_s)|, |E(G_i \cup G_j \cup G_s)|) + |HT|)$. Hence, the overall complexity of Algorithm 3 is $O(\Upsilon(|V(G)|, |E(G)|)|E(Q)|d)$, since for each query edge, we need run lines 4–11 for d times in the worst case, where d is the maximum number of nodes in data graph that are attribute compatible to a query node. The time complexity of Algorithm 4 is $O(|E(Q)||P|)$ where P is the best path set computed by Algorithm 3. Unfortunately, it is non-trivial to give a tight bound for the size of the best path set P, though it is loosely bounded by the number of simple paths between $|E(Q)|$ pairs of nodes, i.e., $O(|E(Q)|2^{|\Sigma|})$. An empirical study shows that in average $|P|$ is linear or even much smaller than $|\Sigma|$. For all the real graphs that we have tested, the ratio between the size of P and $|\Sigma|$ is in the range [0.0003, 0.86].

6. Experimental study

In this section, we conduct a set of experiments on real large graphs to verify the effectiveness and efficiency of our proposed method. We compare our greedy algorithm with G-ray [3] and memory-based implementation of TALE [4]. For fair comparison, we change the original disk-based implementation of TALE to memory-based implementation. In addition, to analyze the benefit of

[6] The computation time complexity is related to the input metric of degree of matching. For instance, in our experiment, the algorithm to compute Eq. (5) is similar to Dijkstra's algorithm [45], which requires $O(|V(G)|\log|V(G)| + |E(G)|)$ time complexity. Hence, in this scenario, $\Upsilon(|V(G)|, |E(G)|) = |V(G)|\log|V(G)| + |E(G)|$.

Table 3
Statistics of datasets.

	PR	AZ	DBLP	RN	PT	LJ
$\|V(G)\|$	19 428	262 111	797 787	1 965 206	3 774 768	4 847 571
$\|E(G)\|$	36 631	899 792	1 301 298	2 765 348	16 518 943	42 851 237
Size	584 KB	14.6 MB	28.6 MB	67.6 MB	299 MB	672 MB
Synthetic	False	True	False	True	False	True
$\|A\|$	200	5000	3752	5000	987	5000
$\|\Sigma\|$	19 428	262 111	696 360	1 965 206	3 774 768	4 847 571

our index technique on overall query processing performance, we implement two algorithms SA-greedy and B-greedy. SA-greedy is a combination of our SA-Index proposed in Section 4 with our greedy algorithm for query processing. B-greedy combines a modularity-based partitioning [46][7] with our second-level indexing as an alternative to SA-Index (namely, B-Index) and feeds the output B-Index to our greedy algorithm. All algorithms proposed in this paper are implemented in standard C++. The original implementation of G-ray uses MATLAB. We convert the MATLAB code into a C++ shared library and run under the same machine configuration as ours and TALE. All experiments are run on a 64-bit machine with a Intel Xeon(R) 2.67 GHz CPU and 24 GB RAM. We use the default settings of G-ray in our experiments; that is, the value of fly-out probability $c=0.1$ and the number of iterations is 10. Regarding to those parameter settings in TALE, we set approximation ratio ρ to be 25% and percentage of important nodes to be 10%.

Datasets. In our experiments, we use the following datasets: protein (PR), amazon (AZ), DBLP, roadnet (RN), patents (PT) and livejournal (LJ). Protein is a human protein interaction network from the Human Protein Database (www.hprd.org), in which nodes are proteins and edges are protein–protein interactions. DBLP is a co-authorship graph extracted from the DBLP Computer Science Bibliography (www.informatik.uni-trier.de/ley/db/). Amazon, roadnet, patents and livejournal are from Stanford large network dataset collection (snap.stanford.edu/data/). We convert those graphs into undirected ones and generate synthetic attribute information for graphs without attribute information. We give the details of each dataset (number of nodes and edges, physical storage size, whether attribute information is synthetic, number of distinct values of attributes and number of distinct values of labels) in Table 3.

Evaluation metrics. We evaluate the proposed methods with the following metrics:

- Scalability with increasing sizes of query graphs and data graphs.
- Node ratio $|V_m|/|V(Q)|$ and edge ratio $|E_m|/|E(Q)|$, where $|V_m|$ is the number of query nodes matched by answer graphs and $|E_m|$ is the number of query edges matched by answer graphs. This metric is originated from the work [30].
- Approximate quality ratio $\mathtt{sim}(Q,G_a)/\widehat{\mathtt{sim}}_{\mathtt{opt}}$, where G_a denotes the answer graph, $\mathtt{sim}_{\mathtt{opt}}$ is the degree of matching between the query graph and the optimal answer graph, and $\widehat{\mathtt{sim}}_{\mathtt{opt}}$ is an estimated upper bound for $\mathtt{sim}_{\mathtt{opt}}$. We compute $\widehat{\mathtt{sim}}_{\mathtt{opt}}$ by Theorem 2 which we put in Appendix B.

The first metric evaluates the efficiency of the different approaches, while the remaining metrics evaluate the effectiveness of the different approaches.

Metric of degree of matching. In our experiments, we use the following metric to evaluate the degree of matching from a query graph to a data graph. Suppose we are given a query graph Q, a data graph G, and a mapping function $f: \widehat{V_q} \leftrightarrow \widehat{V}$. We define the degree of matching between an edge e in query graph and its matching path p in data graph as

$$\mathtt{sim}(e,p|f)=\alpha\mathtt{sim}_1(e,p|f)+\beta\mathtt{sim}_2(e,p|f), \tag{9}$$

where $\mathtt{sim}_1(e,p|f)$, which is the structure similarity between e and p, is defined as $1-(|w(e)-\mathtt{len}(p)|/\max(w(e),\mathtt{len}(p)))$; $\mathtt{len}(p)$ denotes the weighted length of a path p; $w(e)$ denotes the weight of an edge e; $\mathtt{sim}_2(e,p|f)$, which is the semantic similarity between an edge e and a path p, is defined as

$$\frac{\mathtt{sim}(u_e,f(u_e))+\mathtt{sim}(v_e,f(v_e))}{2}\times\frac{\sum_{v_0\in p}\mathtt{sim}(f^{-1}(v_0),v_0)}{|p|},$$

where $|p|$ denotes the number of nodes on p; $\mathtt{sim}(u,v)=1$ if $A^Q(u)=A^G(v)$ and ($l^Q(u)=l^G(v)$ or $l^Q(u)=\lambda$), and 0 otherwise. If $u(v)$ is a mismatched node, then $f(u)$ ($f^{-1}(v)$) is empty; α and β are weighting factors that allow users to adjust the importance of structure similarity and semantic similarity (we consider them equally important here and set $\alpha=\beta=0.5$). Note that although here the similarity measure is defined for weighted graphs, our method works for both weighted and un-weighted graphs.

Now we can define the degree of matching between a query graph Q and a data graph G. Given a query graph Q and a data graph G, the graph matching measure $\mathtt{sim}(Q,G)$ is defined as

$$\mathtt{sim}(Q,G)=\left(\max_f\sum_{e\in E(Q)}\max_p\mathtt{sim}(e,p|f)\right)|E(Q)|, \tag{10}$$

where $|E(Q)|$ is the number of edges in the query graph. We use $|E(Q)|$ to normalize $\mathtt{sim}(Q,G)$.

[7] http://sites.google.com/site/findcommunities/

6.1. Performance evaluation on real data

Indexing performance evaluation: We first evaluate the performance of our SA-Index.

Figs. 6(a) and (b) plot the running time and peak memory consumption during index construction. The results show that our SA-Index takes less time to construct and uses fewer memory than B-Index. Note that we cannot obtain results for B-Index on LJ data since the implementation of modularity-based partitioning [46] runs out of memory when it is tested on LJ data.

Fig. 7(a) shows a comparison on the number of boundary nodes produced by SA-Index and B-Index. In all cases, we can see that SA-Index results in smaller size of boundary nodes than B-Index. One may notice there is a sharp decrease in SA-Index for graph "RN". This is because "RN" has a very small average degree, which indicates that the graph itself has a relatively low conductance and high homogeneity. Therefore, SA-Index returns a partitioning of "RN" with a small partition number and also a small number of boundary nodes. The index storage size of SA-Index and B-Index is shown in Fig. 7(b). One may expect similar results for index storage size, since both the first-level index (i.e., partition-based index) and the second-level index (connectivity-based index) are highly related to the number of boundary nodes. This phenomena is verified in Fig. 7(b). The storage size of the SA-Index is consistently smaller than that of the B-Index. In addition, the curve variation of Fig. 7(b) is also analogous to that of Fig. 7(a).

Efficiency analysis: In this set of experiments, we use different sizes of query graphs and data graphs to assess the query performance of our proposed methods. The results reported are averaged over ten runs.

First, we report the query performance over six data graphs with number of nodes varying from 20 k to 6 M. For each graph, we randomly generate a set of queries with fixed node size 30. Fig. 8(a) shows that both SA-greedy and B-greedy are up to over an order of magnitude faster than G-ray and in the worst case, our greedy algorithms are still around four times faster than TALE. To keep this experiment manageable, we kill any G-ray query if it runs over 10 h. Hence, we do not report the query performance of G-ray on PT and LJ data since

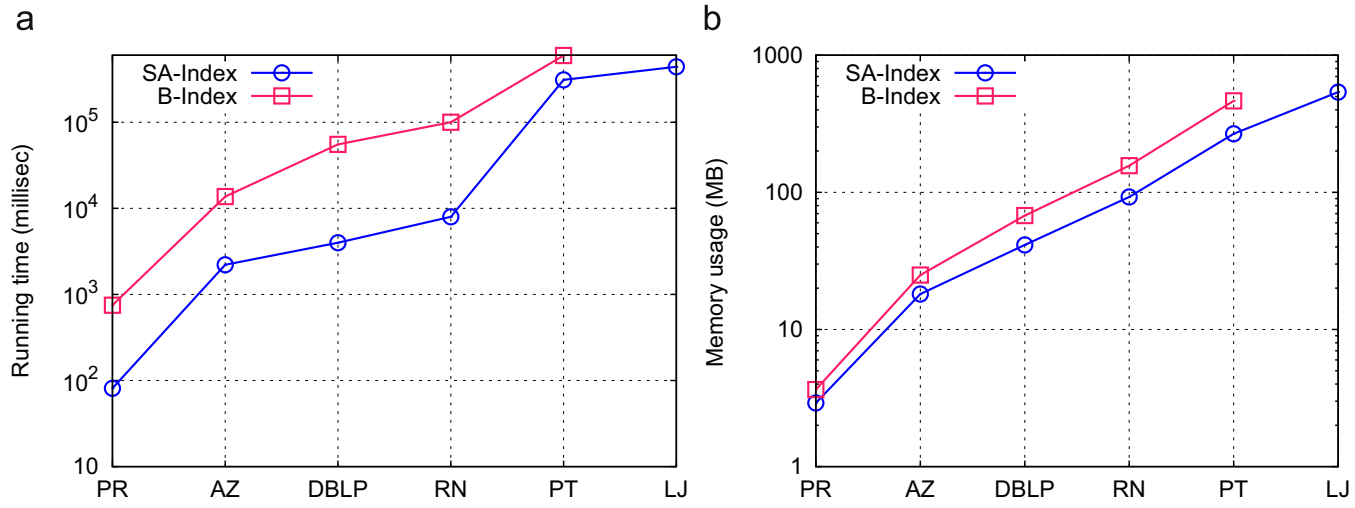

Fig. 6. Index construction performance comparison over real data. (a) Indexing time. (b) Memory consumption.

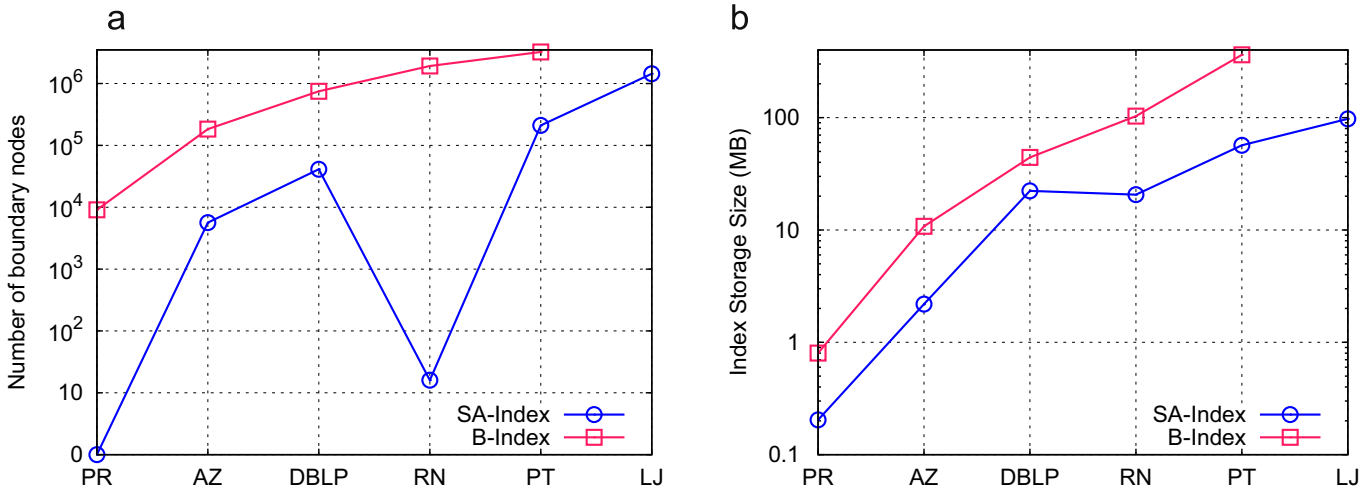

Fig. 7. Index size comparison over real data. (a) Number of boundary nodes. (b) Storage size of the index.

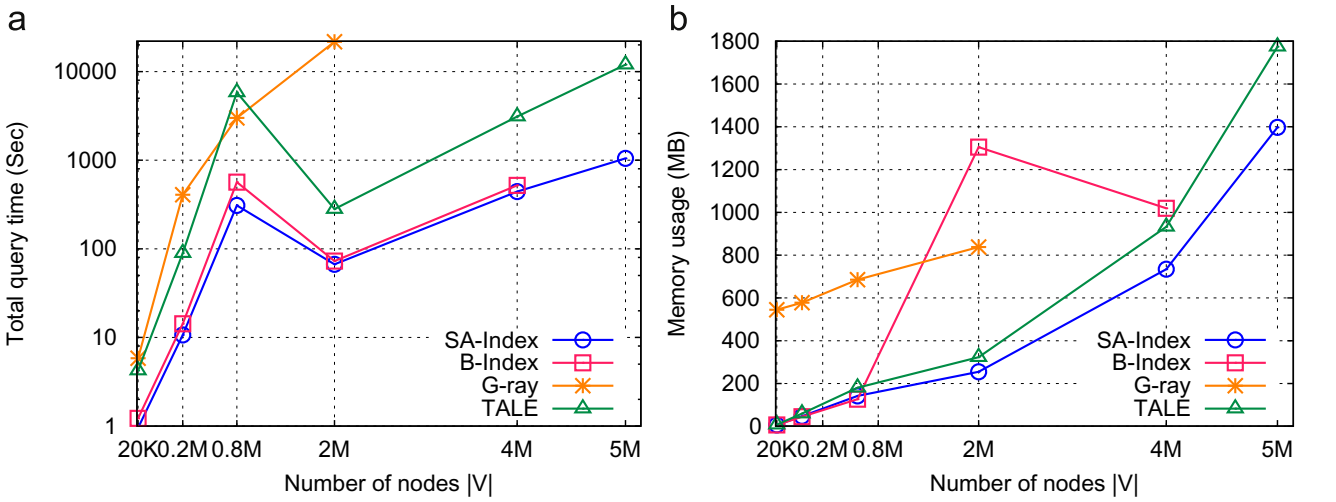

Fig. 8. Effect of data graph size. (a) Querying time. (b) Query memory consumption.

running a small query using G-ray on PT takes around 20 h. In addition, compared with B-Index, the improvement of using SA-Index could achieve 70% speed up. Furthermore, B-Index could not run on large graphs such as LJ.

The query memory consumption of SA-greedy, B-greedy, G-ray and TALE is reported in Fig. 8(b). Except B-greedy, the memory usages of the other approaches are quite scalable to the data graph size. But for B-greedy, we notice a sharp increase on RN data since it produces a large number of boundary nodes for RN data (maintaining connections among boundary nodes takes up a large volume of memory). Naturally, when the data graph size increases, the query memory consumption of B-greedy may soon grow out of memory as memory size is always limited. In addition, among the three scalable approaches, we notice that our SA-greedy achieves the smallest memory consumption.

Next, we vary the number of query nodes from 5 to 30, and report the total query time of four approaches on protein graph in Fig. 9. We do not report the query memory consumption since it is dominated by the size of data graphs other than the size of query graphs.

The above experiments demonstrate the efficiency of our greedy algorithm against G-ray and TALE. To further compare the efficiency introduced by using SA-Index against B-Index, we introduce a metric, "ratio of visited nodes", i.e., number of nodes visited using SA/B-Index against number of nodes visited without index using greedy algorithm, to examine the filtering power of these two indexing approaches. The results are shown in Fig. 10. Clearly, the average number of nodes accessed by using SA-greedy, is much smaller than that by using B-greedy. This also serve as an indicator for the query performance using the two indexes, as we verify in Figs. 8(a) and 9.

Effectiveness analysis: We present the approximate quality ratio, node ratio and edge ratio of answer graphs returned by our greedy algorithm, G-ray and TALE in Fig. 11. According to node ratio and edge ratio, as shown in Fig. 11(b) and (c), the answer graphs computed by our method capture in general more important nodes and edges than those returned by G-ray and TALE. According to the approximate quality ratio

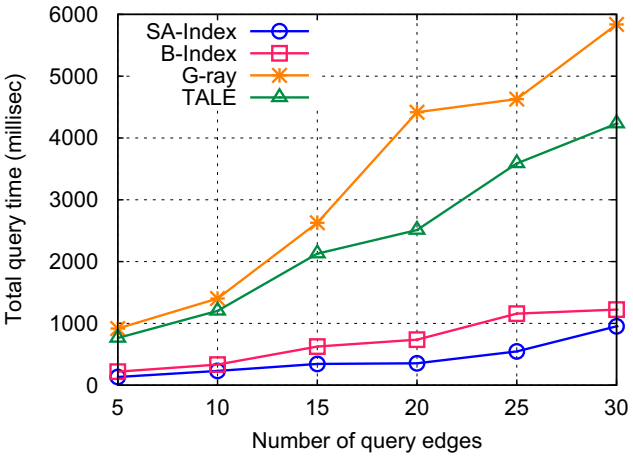

Fig. 9. Effect of query graph size.

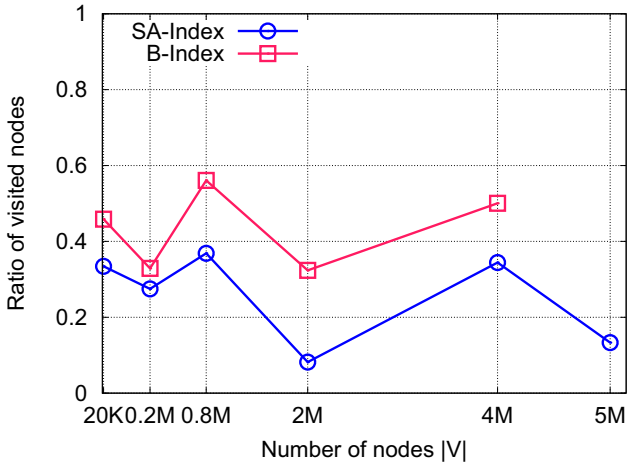

Fig. 10. Effect of using index.

comparison shown in Fig. 11(a), our answer graphs have a higher approximate quality ratio than G-ray's and TALE's answer graphs. Specifically, our significant test shows that (Greedy$\gg$G − ray$\gg$TALE), where $\gg$ means significantly better with p-value < 0.05.

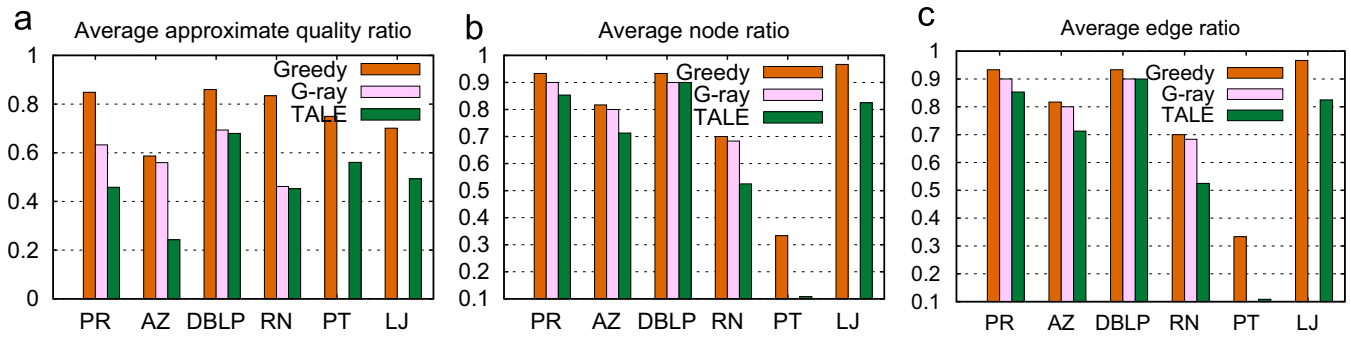

Fig. 11. Effectiveness of matching algorithm. (a) Approximate quality ratio. (b) Node ratio. (c) Edge ratio.

Fig. 12. Answer graphs of line query. (a) A line query. (b) Answer by G-ray. (c) Answer by SA-greedy.

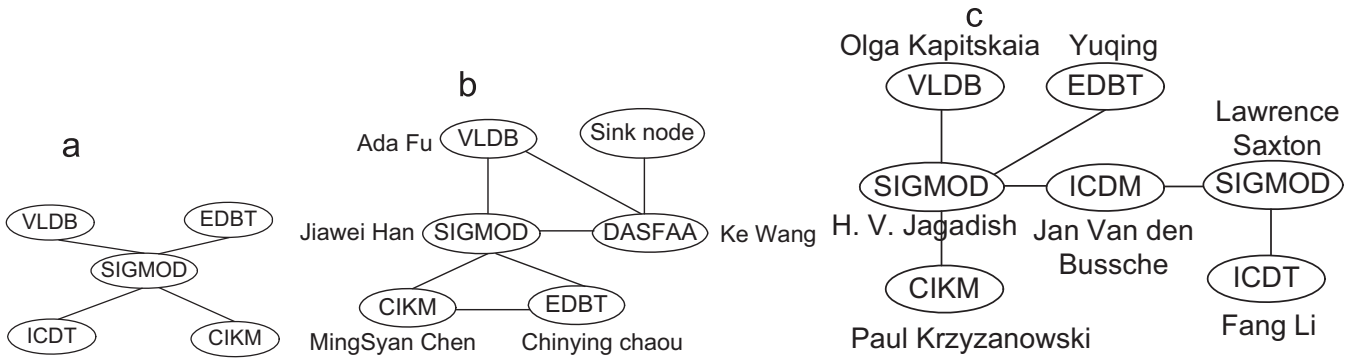

Fig. 13. Answer graphs of star query. (a) A star query. (b) Answer by G-ray. (c) Answer by SA-greedy.

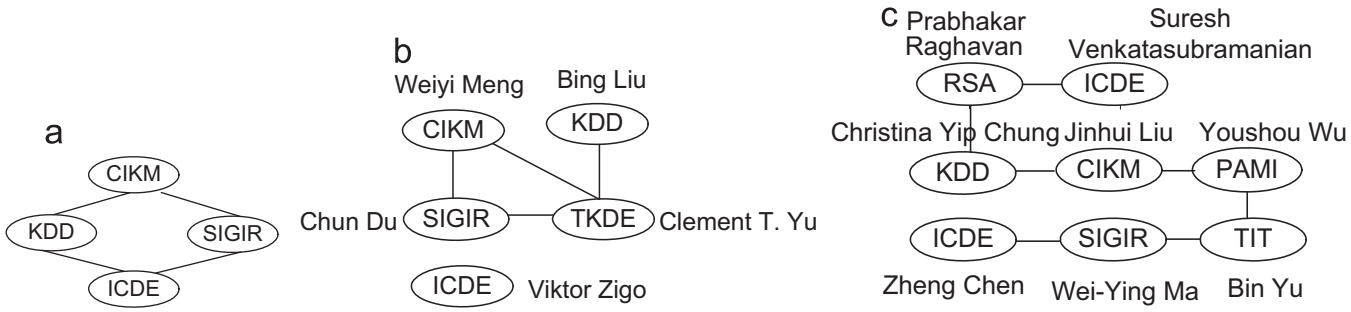

Fig. 14. Answer graphs of loop query. (a) A loop query. (b) Answer by G-ray. (c) Answer by SA-greedy.

Query examples: In Figs. 8–11 we conduct a systematic comparison on the efficiency and the quality of our method with G-ray and TALE. Here, we give a set of three typical example queries to provide the readers a more direct view on the answer graphs returned by Greedy and G-ray (We do not show the answer graphs returned by TALE as G-ray $\gg$ TALE on approximate quality ratio.)

First, let us consider an example of *line query* shown in Fig. 12. Both G-ray and our approach find an exact answer graph for the given query with respect to the attributes. In this case, our approach and G-ray have comparable quality of answer graphs. However, one may notice that in the query "Jiawei Han" is associated with "SIGMOD", but in the answer of G-ray "Michael Stonebraker" is associated with "SIGMOD" instead of "Jiawei Han". This is because G-ray matches a query only by the attributes of the query nodes. Therefore, our method supports a query type that is more expressive than that supported by G-ray.

Next, let us look at an example of *star query*. As shown in Fig. 13, there is no exact match for the given star query. Hence, both G-ray and our approach return an

approximate answer graph for the given query. However, for the G-ray, it fails to find those connections from scholars who publish in "SIGMOD" to those who publish in "ICDT". In contrast, our approach returns a short path which reflects a connection from scholars who publish in "SIGMOD" to those who publish in "ICDT".

Lastly, we show a *loop query*. The result is shown in Fig. 14. Although both answer graphs show some good connections between nodes, our method in particular finds the global connection among the given query nodes, which therefore provides more useful information than G-ray.

7. Conclusions

In this paper, we study the problem of approximate graph matching in large attributed graphs. We propose an index called SA-Index, which directs query processing more effectively to a much smaller but promising search space. Using SA-Index, we efficiently compute a set of best paths (or edges) matching the set of query edges. Then, two fast greedy algorithms are proposed to construct the best matching answer graph from the set of best paths. Our extensive experiments on real datasets verify that our index is efficient to construct, and query processing using the index is over an order of magnitude faster than the existing work. We also show, by various quality measures, that our method obtains high-quality query answers.

Acknowledgements

We are grateful to Dr Hanghang Tong for providing us the source code of G-ray. We sincerely thank Prof. Jignesh M. Patel, Dr Yuanyuan Tian and Dr Ning Zhang for providing their implementation of TALE.

Appendix A. Support for other degree of matching metrics

Maximum common edges: Our approach could be easily extended to support the metric "maximum common edges":

1. Given a query edge $e \in E(Q)$, if there exists a set of edges E' matching e in the data graph G, then the set of best paths for e is E', and empty otherwise.
2. In SA-Index construction, we only need to maintain those inter-partition edges into the hash table *HT*.
3. In on-line query processing, Algorithm 3 could be further simplified as follows: Starting from line 5, if x and y are in the same partition, we only need to scan x's or y's neighborhood in G_i in $H^*(G)$ for best paths; if G_i and G_J are different, we skip if neither x nor y is a boundary node; otherwise, we search the hash table *HT* for best paths.

Graph edit distance: Normally, a path is a sequence of consecutive edges in a graph. Hence, a pair of isolated nodes (two zero degree nodes) cannot be considered as a path. However, in order to support the metric "graph edit distance", a pair of isolated nodes in data graph is allowed to be the best "path" for a query edge. The reason is that the number of insertions that transform a pair of isolated nodes to a query edge can be equal to or even smaller than the number of deletions that transform a path to a query edge. Therefore, we make the following amendments to support "graph edit distance":

1. In the phase of "computation of the best paths", line 11 in Algorithm 3 shall be replaced by the following statements: $p' = \text{argmax}_{p \in S}\{\texttt{sim}(e,p)\}, p^* = \texttt{sim}(e,p') \geq \texttt{sim}(e,\{x,y\})?p' : \{x,y\}$, where $\{x, y\}$ denotes a pair of isolated nodes (this notation is different from (x, y), which denotes an edge).
2. In the phase of "partial answer connection discovery", in Algorithm 4, in each step k, now we can add the path p_k without considering whether p_k shares some common nodes with partial answer G_a or not. That is, two greedy algorithms can be unified as: starts from an empty answer graph G_a, and iteratively in step k, inserts into G_a the path p_k that matches an unmatched query edge (u, v) and maximizes $p_k = \text{argmax}_{\substack{p \in P \\ p \notin G_a}} \{\texttt{sim}(Q, G_a \cup \{p\}) - \texttt{sim}(Q, G_a)\}$.

Appendix B. Upper bound of optimum

Theorem 2. *Assume that* $\texttt{sim}_{\texttt{opt}}$ *is the degree of matching between the query graph Q and the optimal answer graph. Then, we have*

$$\texttt{sim}_{\texttt{opt}} \leq \left(\sum_{\substack{e \in E(Q) \\ i=0 \text{ to } |E(Q)|}} \texttt{sim}(e, p_i^*) \right) \Big/ |E(Q)| \tag{B.1}$$

where p_i^* *satisfies the following recursive equation*:

$$p_i^* = \begin{cases} \emptyset & i = 0 \\ \arg \max\limits_{\substack{p \in (P_e \setminus \bigcup_{k=1}^{i-1} p_k^*) \\ e \text{ is unmatched}}} \texttt{sim}(e,p) & i > 0 \end{cases}$$

Proof (*Sketch*). In step i, $\texttt{sim}_{\texttt{opt}}$ chooses path p_i' that results in the best connected global answer graph. However, the value of $\texttt{sim}(e,p_i')$, for each individual path p_i', may not be the largest among that of all the other paths. While on the right side of Eq. (B.1), we simply choose a path p_i^* that matches an unmapped edge e and that has the maximum value of $\texttt{sim}(e, p_i)$ among all the other paths. Hence, in each step, we have

$$\texttt{sim}(e,p_i') \leq \max_{\substack{p \in (P_e \setminus \bigcup_{k=1}^{i-1} p_k^*) \\ e \text{ is unmatched}}} \texttt{sim}(e,p)$$

and therefore Eq. (B.1) holds. □

References

[1] Y. Tian, R.C. McEachin, C. Santos, D.J. States, J.M. Patel, Saga: a subgraph matching tool for biological graphs, Bioinformatics Journal 23 (2) (2007) 232–239.
[2] S. Marcus, T.R. Coffman, Terrorist modus operandi discovery system 1.0: Functionality, examples, and value, in: 21st Century Technologies Internal Publication, Austin, TX, 2002.

[3] H. Tong, C. Faloutsos, B. Gallagher, T. Eliassi-Rad, Fast best-effort pattern matching in large attributed graphs, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA 2007, pp. 737–746.
[4] Y. Tian, J.M. Patel, Tale: a tool for approximate large graph matching, in: Proceedings of the 24th International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA 2008, pp. 963–972.
[5] D. Shasha, J.T.L. Wang, R. Giugno, Algorithmics and applications of tree and graph searching, in: Proceedings of the 21st ACM SIGMOD–SIGACT–SIGART Symposium on Principles of Database Systems, ACM, New York, NY, USA 2002, pp. 39–52.
[6] X. Yan, P.S. Yu, J. Han, Graph indexing: a frequent structure-based approach, in: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA 2004, pp. 335–346.
[7] H. He, A.K. Singh, Closure-tree: an index structure for graph queries, in: Proceedings of the 22nd International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA 2006, p. 38.
[8] J. Cheng, Y. Ke, W. Ng, A. Lu, Fg-index: towards verification-free query processing on graph databases, in: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA 2007, pp. 857–872.
[9] S. Zhang, M. Hu, J. Yang, Treepi: a novel graph indexing method, in: Proceedings of the 23rd International Conference on Data Engineering, IEEE Computer Society, Washington, DC, USA 2007, pp. 966–975.
[10] P. Zhao, J.X. Yu, P.S. Yu, Graph indexing: Tree + delta $\leq$ graph, in: Proceedings of the 33rd International Conference on Very Large Data Bases, Very Large Data Bases Endowment, 2007, pp. 938–949.
[11] H. Shang, Y. Zhang, X. Lin, J.X. Yu, Taming verification hardness: an efficient algorithm for testing subgraph isomorphism, The Proceedings of the Very Large Data Bases Endowment 1 (1) (2008) 364–375.
[12] H. He, A.K. Singh, Graphs-at-a-time: query language and access methods for graph databases, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA 2008, pp. 405–418.
[13] C. Chen, C.X. Lin, M. Fredrikson, M. Christodorescu, X. Yan, J. Han, Mining graph patterns efficiently via randomized summaries, The Proceedings of the Very Large Data Bases Endowment 2 (1) (2009) 742–753.
[14] J. Cheng, Y. Ke, W. Ng, Efficient query processing on graph databases, ACM Transactions on Database System 34 (1) (2009) 2:1–2:48.
[15] J. Cheng, Y. Ke, W. Ng, Efficient processing of group-oriented connection queries in a large graph, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, 2009, pp. 1481–1484.
[16] J. Cheng, Y. Ke, W. Ng, J.X. Yu, Context-aware object connection discovery in large graphs, in: Proceedings of the 2009 IEEE International Conference on Data Engineering, 2009, pp. 856–867.
[17] Y. Ke, J. Cheng, J.X. Yu, Querying large graph databases, Database Systems for Advanced Applications (2) (2010) 487–488.
[18] J. Huan, W. Wang, J. Prins, J. Yang, Spin: mining maximal frequent subgraphs from graph databases, in: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA 2004, pp. 581–586.
[19] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proceedings of the 2001 IEEE International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA 2001, pp. 313–320.
[20] L.T. Thomas, S.R. Valluri, K. Karlapalem, Margin: maximal frequent subgraph mining, in: Proceedings of the 6th International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA 2006, pp. 1097–1101.
[21] N. Vanetik, E. Gudes, S.E. Shimony, Computing frequent graph patterns from semistructured data, in: Proceedings of the 2002 IEEE International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA 2002, p. 458.
[22] X. Yan, J. Han, Closegraph: mining closed frequent graph patterns, in: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA 2003, pp. 286–295.
[23] Y. Ke, J. Cheng, W. Ng, Correlation search in graph databases, in: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2007, pp. 390–399.
[24] Y. Ke, J. Cheng, W. Ng, Efficient correlation search from graph databases, IEEE Transaction on Knowledge Data Engineering 20 (12) (2008) 1601–1615.
[25] Y. Ke, J. Cheng, J.X. Yu, Efficient discovery of frequent correlated subgraph pairs, in: Proceedings of the 9th IEEE International Conference on Data Mining, 2009, pp. 239–248.
[26] Y. Ke, J. Cheng, J.X. Yu, Top-k correlative graph mining, in: Proceedings of the 9th SIAM International Conference on Data Mining, 2009, pp. 1038–1049.
[27] J.R. Ullmann, An algorithm for subgraph isomorphism, The Journal of the ACM 23 (1) (1976) 31–42.
[28] W.H. Tsai, K. Fu, Error-correcting isomorphisms of attributed relational graphs for pattern recognition, IEEE Transaction on Systems, Man, and Cybernetics 9 (12) (1979) 757–768.
[29] L.P. Cordella, P. Foggia, C. Sansone, M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 26 (10) (2004) 1367–1372.
[30] H. Tong, C. Faloutsos, Center-piece subgraphs: problem definition and fast solutions, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA 2006, pp. 404–413.
[31] S. Zhang, S. Li, J. Yang, Gaddi: distance index based subgraph matching in biological networks, in: Proceedings of the 12th International Conference on Extending Database Technology, ACM, New York, NY, USA 2009, pp. 192–203.
[32] M. Montes-y Gómez, A. López-López, A.F. Gelbukh, Information retrieval with conceptual graph matching, in: Proceedings of the 11th International Conference on Database and Expert Systems Applications, Springer-Verlag, London, UK 2000, pp. 312–321.
[33] H. Bunke, On a relation between graph edit distance and maximum common subgraph, Pattern Recognition Letters 18 (9) (1997) 689–694.
[34] L. Shapiro, R. Haralick, Structural descriptions and inexact matching, IEEE Transactions on Pattern Analysis and Machine Intelligence 3 (1981) 504–519.
[35] 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, 1979.
[36] H. Matsuda, T. Ishihara, A. Hashimoto, Classifying molecular sequences using a linkage graph with their pairwise similarities, Theoretical Computer Science 210 (2) (1999) 305–325.
[37] J. Leskovec, K.J. Lang, A. Dasgupta, M.W. Mahoney, Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters, Computing Research Repository, abs/0810.1355.
[38] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.
[39] B.W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs, The Bell System Technical Journal 49 (1) (1970) 291–307.
[40] A. Gupta, Fast and effective algorithms for graph partitioning and sparse-matrix ordering, IBM Journal of Research and Development 41 (1–2) (1997) 171–183.
[41] A. Pothen, H.D. Simon, K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs, SIAM Journal on Matrix Analysis and Applications 11 (3) (1990) 430–452.
[42] Y. Tian, R.A. Hankins, J.M. Patel, Efficient aggregation for graph summarization, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, ACM, New York, NY, USA 2008, pp. 567–580.
[43] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Physical Review E 70 (2004) 066111.
[44] Y. Zhou, H. Cheng, J.X. Yu, Graph clustering based on structural/attribute similarities, The Proceedings of the Very Large Data Bases Endowment 2 (1) (2009) 718–729.
[45] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., The MIT Press, 2001.
[46] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment (2008) P10008.