BIOINFORMATICS ORIGINAL PAPER

Vol. 23 no. 2 2007, pages 232–239
doi:10.1093/bioinformatics/btl571

Data and text mining

SAGA: a subgraph matching tool for biological graphs

Yuanyuan Tian[1], Richard C. McEachin[2], Carlos Santos[3], David J. States[3] and Jignesh M. Patel[1,*]

[1]Department of Electrical Engineering and Computer Science, [2]National Center for Integrative Biomedical Informatics and [3]Department of Human Genetics and Bioinformatics Program, University of Michigan, Ann Arbor, MI 48109, USA

Received on August 22, 2006; revised on November 7, 2006; accepted on November 8, 2006

Advance Access publication November 16, 2006
Associate Editor: Martin Bishop

ABSTRACT

Motivation: With the rapid increase in the availability of biological graph datasets, there is a growing need for effective and efficient graph querying methods. Due to the noisy and incomplete characteristics of these datasets, exact graph matching methods have limited use and approximate graph matching methods are required. Unfortunately, existing graph matching methods are too restrictive as they only allow exact or near exact graph matching. This paper presents a novel approximate graph matching technique called SAGA. This technique employs a flexible model for computing graph similarity, which allows for node gaps, node mismatches and graph structural differences. SAGA employs an indexing technique that allows it to efficiently evaluate queries even against large graph datasets.

Results: SAGA has been used to query biological pathways and literature datasets, which has revealed interesting similarities between distinct pathways that cannot be found by existing methods. These matches associate seemingly unrelated biological processes, connect studies in different sub-areas of biomedical research and thus pose hypotheses for new discoveries. SAGA is also orders of magnitude faster than existing methods.

Availability: SAGA can be accessed freely via the web at http://www.eecs.umich.edu/saga. Binaries are also freely available at this website.

Contact: jignesh@eecs.umich.edu

Supplementary material: Supplementary material is available at http://www.eecs.umich.edu/periscope/publ/saga-suppl.pdf.

1 INTRODUCTION

Graphs provide a powerful primitive for modeling biological data, such as pathways and protein interaction networks. Naturally, the biomedical community has created many graph databases. For example, PathGuide (http://www.pathguide.org) lists more than 200 pathway databases. Many of these databases are large and rapidly growing in size. To fully exploit the wealth of information in these graph databases, effective and efficient graph querying tools are critical.

Previous graph querying tools have largely focused on relatively simple graph operations, such as retrieving matches based on the node attributes or finding linear paths. However, in practice, more sophisticated approximate graph matching methods are often needed. Note that the emphasis here is on *approximate* matching, as biological graphs are often noisy and incomplete, which makes approximate matching much more useful than exact matching. This paper presents a tool called the **S**ubstructure Index-based **A**pproximate **G**raph **A**lignment (SAGA), which addresses this need.

More formally, the problem that we address is approximate subgraph matching: Given a query graph and a database of graphs, we want to find subgraphs in the database that are similar to the query, allowing for node mismatches, node gaps (node insertions or deletions), as well as graph structural differences. Node mismatches model the behavior that two nodes representing different cellular entities can exhibit similar functionality. For example, two different proteins may be in the same protein orthologous group, which indicates similar functionality. Node gaps represent the situation where a certain node in one graph cannot be mapped to any node in the other graph. Graph structural differences allow for differences in node connectivity relationships. For example, two nodes may be directly connected in one graph, whereas the corresponding matching nodes in the other graphs may be indirectly related through one or more additional nodes.

As a motivating example for the approximate subgraph operation, consider the following scenario: a scientist working on a certain disease has constructed a small portion of a pathway based on analysis of various experimental data. This pathway fragment, which is modeled as a graph, contains nodes that represent cellular entities (proteins, genes, mRNA, etc.) and edges that represent interactions. The scientist is interested in finding the biological processes that may be affected by the disease. This task can be expressed as a query that searches a database of known pathways using the query graph. Furthermore, the search can identify similar subcomponents shared between the query and graphs in the database, which may reveal clues about what information might be missing or spurious in the query graph, and provide a way of generating additional hypotheses.

While there is a long history of research on graph matching, most of this work has focused on exact subgraph matching, i.e. the subgraph isomorphism problem, which is known to be NP-complete. GraphGrep (Shasha *et al*., 2002) and GIndex (Yan *et al*., 2004) are index-based filtering methods for exact subgraph

[*]To whom correspondence should be addressed.

matching. Grafil (Yan *et al*., 2005), PIS (Yan *et al*., 2006) and Closure-Tree (He and Singh, 2006) introduce some approximation for subgraph isomorphism. However, these approximate models are very limited. None of these tools allow node gaps in their models. PathAligner (Chen and Hofestaedt, 2004) is a tool for aligning pathways. However, it assumes that all pathways are linear paths. The tools most closely related to our work are PathBlast (Kelley *et al*., 2004) and the successive NetworkBlast (Sharan *et al*., 2005), which are designed for aligning protein interaction networks. Their graph similarity model allows node mismatches and node gaps, but graph structural differences are largely confined to short paths. As shown in Section 3.5, our graph similarity model tolerates more general structural differences, and can find biologically relevant matches, when both PathBlast and NetworkBlast fail. Another related method (Koyuturk *et al*., 2005) has been proposed for aligning protein interaction networks. However, the match technique used in this method largely focuses on capturing the penalty associated with gene duplication. Finally, PathBlast, NetworkBlast and the method proposed in Koyuturk *et al*. (2005) can only perform one graph comparison at a time. To match a query against a database of graphs, the matching algorithms must be run for each graph in the database. As a result, these methods are not computationally efficient when querying large graph databases.

In this paper, we present a novel approximate subgraph matching technique called SAGA. At the heart of SAGA is a flexible model for computing graph similarity, which permits node gaps, node mismatches and graph structural differences. To speed up the execution of queries with this powerful matching model, we employ an indexing method for efficient query evaluation. Through experimental evaluation, we demonstrate that SAGA is more flexible and powerful than existing models. SAGA allows additional information derived from the relationships between entities in pathways to be incorporated into comparative analysis. Our experimental results show that SAGA finds expected associations, like Insulin signaling in Type 2 Diabetes Mellitus (T2DM). SAGA also finds less well studied associations, like the Toll-like receptor, T-cell receptor and Apoptosis pathways in Helicobacter pylori (H. pylori) infection, as well as Calcium, Wnt and Hedgehog signaling in Bipolar Disorder. In addition, SAGA provides a powerful tool for biomedical text comparison.

2 SYSTEM AND METHODS

2.1 Graph model

In our model, a graph, G, is a 3-tuple $G=(V,E,\phi)$. V is the set of nodes and $E\subseteq V\times V$ is the set of (directed or undirected) edges. Nodes in the graphs have labels specified by the mapping $\phi: V\rightarrow L$, where L is the set of node labels. This model captures the features that are commonly present in most biological graph datasets, in which nodes represent molecules/complexes, labels denote molecule/complex names and edges indicate relationships between nodes. We assume that each node in the graph has a unique ID. This ID is used to establish a total order among the nodes.

In the example graph in Figure 1(a), v_i is used to represent the unique node ID and L_k is the node label. Note that two different nodes in a graph can have the same label.

Our distance model and matching algorithm (discussed below) support both directed and undirected graphs. We present our method using undirected graphs; adaptations of the distance measure and the matching algorithm for directed graphs are straightforward and omitted here.

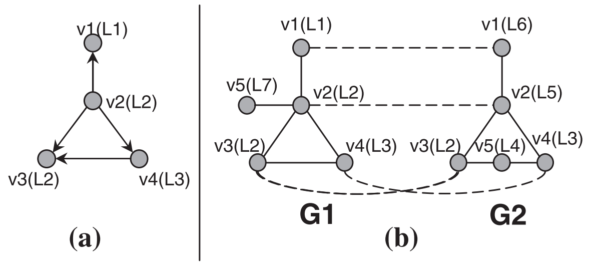

Fig. 1. (a) An example graph. (b) An example subgraph match.

2.2 Distance measure for subgraph matching

Our model measures similarity by a distance value, so graphs that are more similar have a smaller distance. Formally, the subgraph matching is defined as follows: Let $G_1=(V_1,E_1,\phi_1)$ and $G_2=(V_2,E_2,\phi_2)$ be two graphs. An approximate matching from G_1(the query) to G_2(the target) is a bijection mapping function $\lambda{:}\hat{V}_1\leftrightarrow\hat{V}_2$, where $\hat{V}_1\subseteqq V_1$ and $\hat{V}_2\subseteqq V_2$.

An example match is shown in Figure 1. The dashed lines indicate the matched nodes in the two graphs. Note that nodes can be mapped even if they have different labels. Also, note that not all nodes are required to be mapped, e.g. v_5 in G_1 has no mapping in G_2, and is a gap node.

The subgraph distance (SGD), with respect to λ, is defined as:

$$\begin{aligned}\mathrm{SGD}_\lambda(G_1,G_2) = {} & w_e\times\mathrm{StructDist}_\lambda\\ & + w_n\times\mathrm{NodeMismatches}_\lambda\\ & + w_g\times\mathrm{NodeGaps}_\lambda\end{aligned}\qquad(1)$$

where

$$\mathrm{StructDist}_\lambda = \sum_{u,v\in\hat{V}_1,\,u<v} \mid d_{G_1}(u,v) - d_{G_2}(\lambda u,\lambda v)\mid \qquad(2)$$

$$\mathrm{NodeMismatches}_\lambda = \sum_{u\in\hat{V}_1}\mathrm{mismatch}(\phi_1(u),\phi_2(\lambda u)) \qquad(3)$$

$$\mathrm{NodeGaps}_\lambda = \sum_{u\in V_1-\hat{V}_1}\mathrm{gap}_{G_1}(u) \qquad(4)$$

The distance model contains three components. The StructDist component measures the structural differences of the match, the NodeMismatches component is the penalty associated with matching two nodes with different labels, and the NodeGaps component is used to measure the penalty for the gap nodes. (Gap nodes are nodes in the query that cannot be mapped to any nodes in the target graph.) Each of these components is described in more detail in subsections 2.2.1 through 2.2.3.

In Equation (1), w_e, w_n and w_g are the weights for each component in this matching model, and can be used to change the emphasis on the different parts of the similarity model. While Equation (1) computes the subgraph distance for a specific matching λ, the actual subgraph distance from a query to its target is the minimum distance over all possible matchings, namely:

$$\mathrm{SGD}(G_1,G_2) = \min_{\lambda}\mathrm{SGD}_\lambda(G_1,G_2) \qquad(5)$$

2.2.1 The StructDist component The StructDist component measures the structural differences for the matching node pairs in the two graphs. In Equation 2, the $d_{G_i}(u,v)$ function measures the 'distance' between node u and node v in graph G_i, and is defined as the length of the shortest path between u and v. The StructDist component compares the distance between each pair of matched nodes in one graph to the distance between the corresponding nodes in the other graph, and accumulates the differences.

2.2.2 The NodeMismatches component The NodeMismatches component in Equation (3) is the sum of the penalties (quantified by the *mismatch* function) associated with matching nodes with different labels.

A common and biologically intuitive mismatch penalty model is to implicitly group node labels based on similarity, allowing for a node label to be associated with more than one group. Nodes can then be compared based on the group labels. This model of node comparison is quite general and practical for many biological applications. For example, the functional similarity between two enzymes can be determined based on the length of the common prefix of the corresponding Enzyme Commission (EC) numbers. For general proteins, one can use databases like KEGG (Kanehisa *et al*., 2006) and COG (Tatusov *et al*., 1997) which organize proteins into orthologous groups, and consider two proteins to be functionally similar only if they are in the same group. This mismatch model can also be generalized to other settings, such as comparing nodes belonging to different classes based on the positions of the two classes in a classification hierarchy, such as Gene Ontology (http://www.geneontology.org).

We utilize the concept of orthologous groups for our node mismatch model. The mapping from a node label to a set of orthologous groups, allowing a node to belong to more than one orthologous group, is defined as $\varrho: L \rightarrow P(\mathrm{GL})$, where L is the set of node labels, GL is the set of group labels, and $P(\mathrm{GL})$ is the power set of GL. Under this model, $mismatch(L_i, L_j) = \infty$ if $\varrho(L_i) \cap \varrho\ (L_j) = \emptyset$, and $mismatch(L_i, L_j) < \infty$, otherwise.

2.2.3 The NodeGaps component The NodeGaps component in Equation (4) measures the penalties associated with the gap nodes in the query graph, thereby favoring matches that have fewer gap nodes. In our model, different nodes in the query graph can have different penalty values, and nodes with the same label can have different penalties as well.

The model also gives users the freedom to choose between gapped matches (matches that allow gap nodes) and ungapped matches. If $\mathrm{gap}_{\mathrm{G}}(u)$ is set to ∞ for every node, then the model only supports ungapped matches, otherwise it allows gapped matches.

For simplicity, for the rest of the discussion, we will assume that all nodes have the same gap penalty value denoted as SingleGapCost.

2.2.4 Characteristics of the subgraph distance model Our subgraph matching model is very flexible and allows for incorporation of domain knowledge into the scoring criteria. The only restriction is that the gap penalty must be positive and the mismatch penalty must be non-negative. These restrictions ensure that the subgraph distance is a non-negative value. With these restrictions, if the query graph is subgraph-isomorphic to the target graph, the subgraph distance is 0, and vice versa.

2.3 The index-based matching algorithm

A naïve technique for evaluating subgraph matching queries is to compare the query with every graph in the database and report the matches, which is prohibitively expensive. We propose a novel index-based heuristic algorithm that allows for a much faster evaluation of the approximate subgraph matching operation.

First, an index is built on small substructures of graphs in the database. This index is then used to match fragments of the query with fragments in the database. Finally, the matching fragments are assembled into larger matches. The actual method is described in detail below.

2.3.1 The index structures The index on small substructures of graphs in the database is called the FragmentIndex. It is probed by the matching algorithm to produce hits for substructures in the query.

The indexing unit is a set of k nodes from the graphs in the database. We call each such set a fragment. Here k is a user specified parameter, and is usually a small number like 2, 3 or 4. However, simply enumerating all possible k-node sets is expensive in terms of both time and space. At the same time, if any pair of nodes in a fragment is too far apart by the pairwise distance measure (refer to Section 2.2.1), this fragment does not correspond to a meaningful substructure, thus is not worth indexing. Therefore, a parameter $d_{\max}$ is specified to control whether a fragment is to be indexed. For a given k-node set $v_1, v_2, \ldots, v_k$, if any two nodes v_i and v_j satisfy $d(v_i, v_j) \leq d_{\max}$, we connect the two nodes by a pseudo edge. Then, we index this fragment only if the k nodes form a connected graph by the pseudo edges. Using this heuristic, we can dramatically reduce the size of the FragmentIndex.

Note that in contrast to existing methods, which index connected subgraphs, the fragments in SAGA do not always correspond to connected subgraphs. The reason for using the more general definition of fragments is to allow node gaps in the match model. For example, in Figure 1(b), nodes υ_3 and υ_4 in G_1 can be matched to nodes υ_3 and υ_4 in G_2, respectively. Although υ_3 and υ_4 do not form a connected subgraph in G_2, they correspond to a fragment that needs to be indexed so that this match can be detected.

An entry in the FragmentIndex has the following format: {nodeSeq, groupSeq, distSeq, sumDist, gid}, where nodeSeq is the sequence of node IDs for the nodes in the fragment, groupSeq is the sequence of group labels associated with the nodes, distSeq is the sequence of pairwise distances between the nodes in the fragment, sumDist is the sum of these pairwise distances, and gid is a unique graph ID. Recall that a node label can be associated with multiple group labels. In this case, we generate all possible group label sequences for a fragment, and index each one. An example showing the FragmentIndex on a sample database is presented in Section 1 of the supplemental material.

To efficiently evaluate the subgraph distance between a query graph and a database graph, an additional index called DistanceIndex is also maintained. This index is used to look up the precomputed distance between any pair of nodes in a graph (Section 2.2.1).

2.3.2 The matching algorithm The matching algorithm proceeds as follows: First, the query is broken into small fragments and the FragmentIndex is probed. Then, the hits from the index probes are combined to produce larger candidate matches. Finally, each candidate is examined to produce the actual results. Each of these three steps is described in detail below.

Step 1: finding small hits. In this step, the query is broken into small fragments and the FragmentIndex is probed to find database fragments that are similar to the query fragments.

Given the query, fragments (k-node sets) are enumerated in the same way as we did for the database graphs. Next, for each query fragment, the groupSeq, nodeSeq, sumDist, and distSeq values are computed. Then, the FragmentIndex is probed with each of these query fragments.

The actual index probe uses the following multi-level filtering strategy: First, the groupSeq and sumDist values are used to filter out fragments that cannot match. Next, additional false positives are removed using the distSeq values.

In the first level of filtering, database fragments are fetched only if they have the same groupSeq as the query fragment and their sumDist values are within the safe bounds that we have developed. Formally, the probe criteria is: $\{t\,|\,t \in \mathrm{FragmentIndex},\ t.\mathrm{groupSeq} = f_q.\mathrm{groupSeq}, f_q.\mathrm{sumDist} - \frac{k(k-1)}{2} \times \frac{\mathrm{MaxPairDist}}{w_e} \leq t.\mathrm{sumDist} \leq f_q.\mathrm{sumDist} + \frac{k(k-1)}{2} \times \frac{\mathrm{MaxPairDist}}{w_e}\}$, where f_q is the query fragment, and k is the fragment size. MaxPairDist is a user-defined parameter which restricts the weighted pairwise distance difference between the query and the database fragments as $w_e \times |\,d_{G_1}(u,v) - d_{G_2}(\lambda u, \lambda v)\,| \leq$ MaxPairDist. (See Section 2 of the supplemental material for details.)

After the first level of filtering, we get a list of candidate database fragments for every query fragment. This list can be further refined by using the distSeq information (which contains the pairwise distances) to check that all pairwise distances satisfy the MaxPairDist criterion defined above.

Step 2: assembling small hits. Step 1 produces a set of small fragment hits. These smaller hits are assembled into bigger matches as follows: First, the hits are grouped by the database graph IDs. Then, a hit-compatible graph is built for each matching graph. Each node in a hit-compatible graph corresponds to a pair of matching query and database fragments. An edge is drawn between two nodes in the hit-compatible graph if and only if two query fragments share 0 or more nodes, and the corresponding database fragments in the hit-compatible graph also share the same corresponding nodes. An edge between two nodes tells us that the corresponding two hits can be merged to form a larger match, since they have no conflicts in the union. Therefore, a clique in the hit-compatible graph represents a set of hits that can be merged without any conflicts.

After forming the hit-compatible graph, the hits assembling problem reduces to the maximal clique detection problem, which can be solved using existing efficient implementations, such as (Born and Kerbosch, 1973), or approximate methods such as (Hochbaum, 1997). The set of hits in each maximal clique is a candidate match. A detailed example of this second step for a sample query is illustrated in Section 1 of the supplemental material.

Step 3: examining candidates. This step examines each candidate match and produces a set of real matches. Here, we allow users to specify a threshold P_g to control the percentage of gap nodes in the subgraph match. With a given P_g value, the desired matches are those with at most P_g percentage of gap nodes in the query.

For each candidate match obtained from Step 2, we first check whether the percentage of the gap nodes exceeds the threshold P_g. If so, we ignore the candidate. Otherwise, we probe the *DistanceIndex* and calculate the real subgraph matching distance as defined in Section 2.2. Recall that the required subgraph matching is the one that minimizes the matching distance (cf. Equation 5). We also further examine the submatches of the candidate. A submatch can be obtained by removing one or more node mappings from the original match. This introduces more gap nodes to the query, and thus increases the subgraph distance by additional gap penalties. However, at the same time, the StructDist and NodeMismatches may be reduced according to its definition in Equations (2) and (3). Therefore, if the decreased amount exceeds the increased amount, the overall matching distance will be lower than the original one, which also means that a better match is found for the query. If two matches have the same matching distance and one is a submatch of the other, only the supermatch is considered.

2.4 Fragment size parameter

The fragment size parameter (k in Section 2.3.1) controls the size of fragments in the FragmentIndex. This parameter affects the size of the index, query performance and sensitivity of search results. A larger fragment size results in a larger FragmentIndex, which increases the index probe cost. However, a large fragment size may also results in fewer false positives in the hit detection phase (and lower query sensitivity), which reduces the cost of the remaining steps. A practical way of picking a fragment size is based on the selectivity of the queries. If queries are expected to have many matches in the database, then a smaller fragment size is preferred as it may not introduce many false positives, and also potentially lead to smaller sizes of hit-compatible graphs. However, when queries tend to have very few matches, a large fragment size may be favored to prune false positives in the early stages of the matching algorithm.

2.5 Statistical significance of matching results

The Monte Carlo simulation approach is employed to assess the statistical significance of the matches. A P-value is computed for each match based on the frequency of obtaining such a match, or a better match, when applying SAGA with randomized data. Random graphs are generated by random shuffling of edges of the graphs preserving the node degrees, and randomizing the orthologous groups of each node preserving the number of orthologous groups that each node belongs to. For a given query, in addition to querying the real database, we run SAGA on a large number of random graphs, and estimate the P-value of a match from the real database as the fraction of matches from the random graphs with the same or a larger size (in number of nodes) and the same or a smaller distance value.

3 IMPLEMENTATION AND RESULTS

In this section, we describe the implementation of SAGA and present results demonstrating its effectiveness and efficiency. The well-known KEGG pathway database (Kanehisa *et al*., 2006) is used for the experiments. In addition, we use a dataset, called bioNLP, which contains parsed PubMed documents represented as graphs. In these graphs, nodes represent genes and edges denote that two genes were discussed in the same sentence somewhere in the document. With bioNLP, graph similarity can be used to identify related documents.

3.1 Implementation

We have implemented SAGA using C++ on top of PostgreSQL (http://www.postgresql.org). For detecting maximal cliques, we use the version 2 algorithm described in (Born and Kerbosch, 1973). The DistanceIndex and FragmentIndex are implemented as clustered B+-tree indices. The fragment size was set to three. The execution times reported correspond to the running time of the C++ program (which includes reading the query specifications and issuing SQL queries to the DBMS to fetch index entries and related database tuples). All experiments were run on a 2.8 GHz Pentium 4, Fedora 2 machine equipped with a 250 GB SATA disk. We used PostgreSQL version 8.1.3 and set the buffer pool size to 512 MB.

For all the experiments with KEGG, the values for the SAGA parameters are: $w_e = w_g = w_n = 1$, SingleGapCost $= 3$, $d_{\max} = 3$ and MaxPairDist $= 3$. The P_g value is set for every query so that each match contains at least four node mappings. For the node mismatch penalty, we use a simple model: if two nodes belong to the same KEGG orthologous group or they have the same EC number, then the mismatch penalty is 0, and ∞ otherwise. For the significance test, we generate 100 random graphs for each graph in the database, so there are totally $n \times 100$ random graphs, if n is the number of graphs in the database. We only retain matches with 0.01 significance level or better. When a query graph is also included in the database, we always exclude the self-match (the query graph matching itself) from the results. For the experiment with the bioNLP dataset, the SAGA parameter settings are: $w_e = w_g = w_n = 1$, SingleGapCost $= 0.5$, $d_{\max} = 3$, and MaxPairDist $=$ 3. For the node mismatch model, nodes with the same label have 0 penalty, otherwise the mismatch penalty is ∞.

3.2 Finding conserved components across pathways

Two experiments are used to investigate components that are shared across different pathways.

3.2.1 Querying disease-associated pathways This experiment is an exploratory analysis to find biological processes that are involved in, or are affected by, a particular disease. We use all 162 KEGG human pathways (downloaded on July 4, 2006) as the database and chose the 10 disease-associated human pathways as queries. This query set is a subset of the 162 human pathways and it includes three metabolic disorder pathways, six neuro-degenerative disorder pathways, and one infectious disease pathway. Of these pathways, only two query pathways produced significant hits (P-value ≤ 0.01): the T2DM pathway (hsa04930) and the ‘Epithelial cell signaling in H. pylori infection’ pathway (hsa05120). Results for these two pathways are presented in Table 1. The full list of the database and query pathways can be found in the supplemental material.

Table 1 shows both the P-values and the number of PubMed references for the matches, as a measure of how well the disease association has been studied in previous literature. We are particularly interested in disease-associated pathway matches that are significant but are not yet well studied.

Table 1. Significant matches for the T2DM and H. pylori disease associated KEGG pathways

Query	Match	No. of nodes matched	*P*-value	Refs
T2DM (hsa04930)	Insulin (hsa04910)	8	0.0009	21 326
	Adipocytokine (hsa04920)	5	0.0009	37
H.pylori (hsa05120)	Toll-like receptor (hsa04620)	7	0.001	12
	T-cell receptor (hsa04660)	4	0.001	2
	Apoptosis (hsa04210)	4	0.006	130

The number of PubMed references is simply produced by querying PubMed with the keywords in the pathway names.

As can be seen in Table 1, SAGA finds that the T2DM pathway (hsa04930) is significantly associated with both Insulin signaling (hsa04910) and Adipocytokine signaling (hsa04920). In the case of Insulin signaling, we find a match of eight nodes of Insulin signaling in the T2DM pathway. The number of PubMed references for 'Type II diabetes mellitus AND Insulin' is 21 326, consistent with the well-studied nature of Insulin signaling in T2DM. This result demonstrates that SAGA finds pathway matches that would be expected by researchers experienced in disease-related pathways research. In the case of Adipocytokine signaling in T2DM, we find a match of five nodes and the number of references is 37, in agreement with the less well-studied nature of Adipocytokine signaling in T2DM.

The H. pylori pathway (hsa05120) demonstrated significant matches to the Toll-like receptor, T-cell receptor and Apoptosis pathways. The association between H. pylori infection and Apoptosis is relatively well studied (130 PubMed references), while the association with Toll-like receptor signaling is less well studied (12 references) and the association with T-cell receptor signaling shows only two references. This result suggests that T-cell receptor signaling is potentially a significant but relatively unstudied avenue for research into the etiology of H. pylori infection.

3.2.2 Querying signal transduction pathways In this experiment, we use the same database of pathways as in section 3.2.1 (162 KEGG human pathways) but we choose all the 12 signal transduction pathways (KEGG IDs: hsa04010, hsa04020, hsa04070, hsa04150, hsa04310, hsa04330, hsa04340, hsa04350, hsa04370, hsa04630, hsa04910 and hsa04920) as the query set to demonstrate additional benefits to be derived from identifying pathways matches. Many of the matches are intuitive for researchers familiar with specific cellular, tissue or disease phenomena (as expected). However, pairs of pathways between which the similarities are not intuitive can be useful in both pathway annotation and disease association research. In the following discussion, we present two examples of such matches.

In the first example, Figure 2 shows components that are shared by the Hedgehog (hsa04340) and Wnt (hsa04310) signaling pathways (*P*-value 0.005). Note that nodes are matched based on functionality. For example, Slimb is matched with B-TrCp as both are SCF complex F-box proteins (KEGG Orthology, KO:K03362). While SAGA can find this orthologous match, the difference in terminology seen in the KEGG pathways database might make it difficult for many researchers to find the match. These similarities between Hedgehog and Wnt signaling are consistent with http://www.stanford.edu/~rnusse/pathways/WntHH.html, as well as Kalderon (2002) and Nusse (2003).

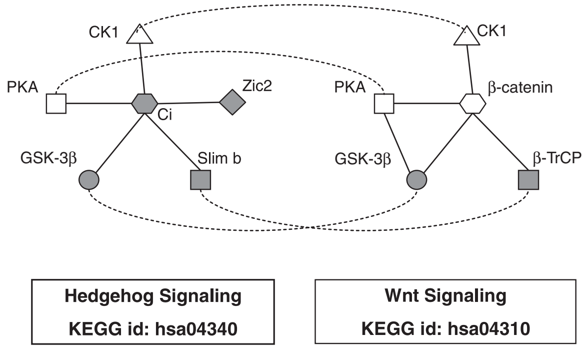

Fig. 2. Hedgehog pathway matched the Wnt pathway.

In the second example, the Wnt and Calcium signaling pathways share four enzymes (Fig. 3, *P*-value 0.007). However, the Calcium signaling pathway has two additional components (CALM and IP3R) that arguably belong to the Wnt pathway. By identifying the common components, we can provide information to improve the annotation of the Wnt pathway.

Based on the significant similarities between the Wnt/Hedgehog and Wnt/Calcium pathways, we hypothesize that the three pathways (Wnt, Calcium and Hedgehog signaling) could share disease associations. Calcium signaling has been investigated in relation to Bipolar Disorder (BD) for more than 40 years (Coppen, 1967). After examining the Wnt/Calcium and Wnt/Hedgehog matches, we conducted a literature search and found 335 PubMed references investigating Calcium signaling in BD, as well as 15 PubMed references for Wnt signaling in BD, consistent with our hypothesis. However, when looking for BD association with Hedgehog signaling, we found zero PubMed reference, which suggests that the Hedgehog signaling pathway has been largely overlooked in BD research, although it uses BD-associated components. This result poses new hypotheses for exploring the relationship between BD and Hedgehog signaling, and shows how SAGA can be useful in disease research.

3.3 Reactome pathways vs. KEGG pathways

SAGA can also be used to compare pathways in different databases (e.g. as a precursor to integrating data from different pathway databases). In this experiment, we compare two well-known pathway databases: Reactome (Joshi-Tope *et al.*, 2005) and KEGG.

We use the same 162 KEGG human pathways as the database. The queries are the eight newly updated pathways in Reactome version 17. The query set includes TGF-β (Reactome ID: 170834), RIG-I (168928), Toll-like receptors 3 (168164) and 4 (166016), the conjugation phase of xenobiotic metabolism (156580), aspects of the metabolism of lipoproteins (174824), cell cycle regulation by the anaphase-promoting complex (APC) (174143) and ATR activation in response to replication stress (176187).

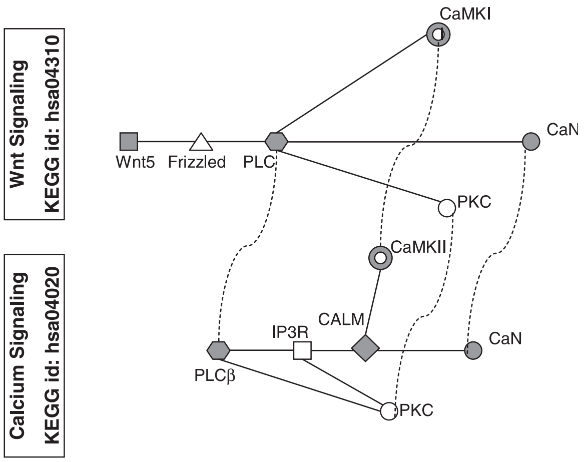

Fig. 3. Wnt pathway matched the Calcium pathway.

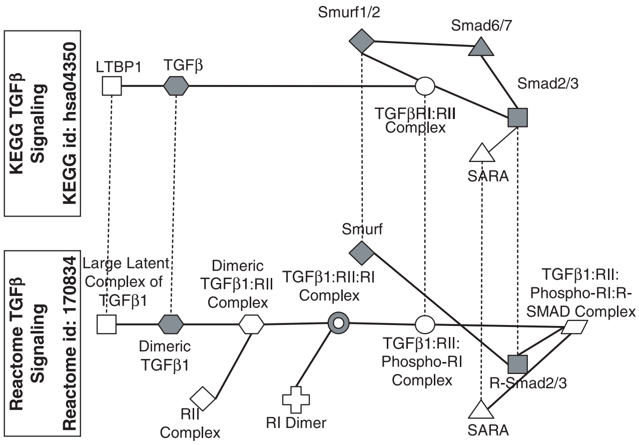

Fig. 4. The shared components between KEGG and Reactome TGF-β pathway.

Naturally, the TGF-β pathway (with 23 nodes and 25 edges) in Reactome matches the TGF-β (hsa04350) pathway (with 65 nodes and 45 edges) in KEGG. However, pathways in the two databases are not perfectly matched (graph distance >0). Each of the pathways contains some details missing in the other. Also, as shown in Figure 4, there are some differences even in the shared similar components between the two pathways. By identifying the similar subcomponents using SAGA, researchers can combine the two databases and produce more complete data.

The two databases also organize pathways in different ways. Reactome represents pathways in a hierarchy (i.e. a pathway consists of several subpathways and subpathways again can be made of subpathways). On the contrary, KEGG stores pathways in a flat fashion. As examples of the organizational difference, the Toll-like receptors 3 and 4 pathways in Reactome match the Toll-like receptor (hsa04620) pathways in KEGG, and both cell cycle regulation by the Anaphase-promoting complex (APC) and ATR activation in response to replication stress pathways in Reactome hit the cell cycle pathway in KEGG. Thus, SAGA can be used for graph data integration even if databases organize the same information in different ways.

3.4 SAGA for querying parsed literature graphs

This experiment examines how SAGA can be applied within an information retrieval setting. While traditional IR methods employ term-based comparisons and the cosine similarity measure (Salton and McGill, 1983) for comparing documents, we look at the document comparison problem specifically in the biomedical domain and address it using a graph matching method. Each PubMed document is represented by a graph in which a node indicates a gene studied in that document. A link is drawn between two genes if they are discussed in the same sentence (indicating there is potentially association between the two genes). The graph presentation summarizes the genes and gene associations derived from a document. By querying the graph representation of a document against those of other documents, documents that address the same topics as the query document can be identified, even if they are published in different areas of research. For example, we queried the publication (Tourigny *et al*., 2002) (five nodes and six edges) against 48 444 PubMed documents using the cut-off value $P_g = 50\%$. (This dataset has an average of 5.0 nodes and 18.8 edges per graph, and the list of documents in this set can be accessed at http://enigma.eecs.umich.edu/doc.txt.) Among the 11 matches found by SAGA, the top hit is (Luedde *et al*., 2003), which does not have a citation to (Toruigny *et al*., 2002). The shared components between the two graphs are three genes: CDK inhibitor p18(INK4c), 0610007C21Rik and Stmn1, as well as their three pairwise associations. The query publication (Toruigny *et al*., 2002) explored p18(INK4c) in the generation of functional plasma cells, while (Luedde *et al*., 2003) investigated the role of this gene in the regenerating liver. Thus, SAGA can be used to connect related studies even in different sub-areas of biomedical research.

3.5 Comparison with existing tools

GraphGrep (Shasha *et al*., 2002) and Gindex (Yan *et al*., 2004) are designed to match one graph against a collection of graphs. However, they only support exact subgraph isomorphism. Given the noisy and incomplete characteristics of biological graphs, exact matching cannot help much in our target applications. Grafil (Yan *et al*., 2005), PIS (Yan *et al*., 2006), and Closure-Tree (He and Singh, 2006) disallow gap nodes in their match models, which prohibits them from getting results that SAGA can find. For example, none of the 12 signal transduction pathways queries produce any matches (excluding self-matches) in the KEGG human pathway database using these three tools.

As discussed in Section 1, NetworkBlast is a tool for aligning large protein interaction networks. On the other hand, SAGA is designed for matching relatively small graph queries (sparse graphs with less than 100 nodes) against a large set of (large or small) graphs. Although NetworkBlast and SAGA have different characteristics, it is interesting to consider applying NetworkBlast to pathway matching. To query the set of pathways in KEGG (cf. Section 3.2.2), we have to run NetworkBlast once for each pathway in the database. In other words, for the experiment in Section 3.2.2, for each query, we need to invoke 162 calls to NetworkBlast. For the Wnt signaling pathway (hsa04310) with 73 nodes and 92 edges, the 162 runs of NetworkBlast takes more than 20 hours, while SAGA only takes about 8 minutes! Besides the more than two orders of magnitude speedup, SAGA produces results with higher quality. First, SAGA never misses any matching pathways that NetworkBlast can find. Second, SAGA can find matches that NetworkBlast cannot find. The reason is that graph structural differences in NetworkBlast are largely confined to short paths, while SAGA tolerates more general structure differences. For example, neither of the two matches shown in Figures 2 and 3 can be found by NetworkBlast.

3.6 Efficiency evaluation

This experiment evaluates the efficiency of SAGA. To measure the raw performance, we only measure the time it takes for SAGA to produce matches, and do not include the time for generating the *P*-value statistics.

We choose as queries the 10 disease associated KEGG pathways (mirroring the experiment in Section 3.2.1). To vary the database sizes, we add pathways for other species to the database. The details of the databases are described in Table 2.

The query execution times for the 10 queries with increasing database sizes are shown in Table 3. Even for the largest database, the query execution times using SAGA are less than 1 second.

Besides the database sizes, the query execution times also depend on the number of nodes and edges in the query, the actual query graph structure, and the number of hits in the database. Almost all the 10 human disease pathways have matches in the human, mouse and rat pathways, but no matches exist for them in the worm and yeast pathways. This explains why the execution times for the queries on the databases d4 and d5 are similar to the execution times against the database d3. For the databases d1 through d3, even though the database sizes roughly doubles at each step, the query execution times grow at a slower rate, since the index matching components grow at at rate that is slower than the database growth rate.

Another observation is that a larger query does not necessarily result in a larger execution time. For example, hsa05040 is a single connected graph with more matches in the databases than hsa04950, which is a graph with several connected components. The execution times with hsa05040 are more than hsa04950, although hsa04950 has more nodes and edges than hsa05040.

4 DISCUSSION

This paper discusses SAGA, a powerful method for approximate subgraph matching. SAGA employs a match model that can be used to accurately incorporate domain knowledge for capturing the domain-specific notion of graph similarity. An index-based algorithm makes approximate subgraph matching queries very efficient. Our evaluations using a number of actual biomedical applications show that SAGA can produce biologically relevant matches on actual examples, whereas existing tools fail. In addition, we have demonstrated the efficiency of the SAGA approach.

SAGA is very effective and efficient for querying relatively small graphs (ideally sparse graphs with less than 100 nodes) against very large databases, and there are many compelling applications in this setting (cf. Section 3). However, we do not recommend using the existing tool when the query graph is very dense and/or has a large number of nodes. For such large query graphs, the performance of the existing SAGA method degrades since potentially a large number of small hits can be produced by Step 1 of the matching algorithm (cf. Section 2.3.2). Assembling these hits is computationally expensive with the existing SAGA algorithm. To improve the performance of SAGA for large query graphs, one can leverage the observation that biological graphs have very strong modular structures (Tornow and Mewes, 2003). In other words, these graphs can be naturally divided into groups with dense intra-group connections and very sparse inter-group connections. As part of future work, we plan on making use of the modular property and employ a divide-and-conquer strategy to handle large queries. More specifically, we can divide the queries into several sub-queries (the union of the sub-queries should cover the original query), then use SAGA to match each sub-query. Finally, we can assemble the results for the sub-queries to produce the final matches.

Table 2. Characteristics of various databases used for the scalability experiment

Dataset	Pathways	No. of graphs	Avg. no. of nodes	Avg. no. of edges	*FragmentIndex* Size (no. of entries)
d1	human	162	86.0	35.3	1.38×10^7
d2	d1 + mouse	320	86.3	34.8	2.94×10^7
d3	d2 + rat	470	86.6	31.7	4.07×10^7
d4	d3 + worm	567	89.0	28.5	5.34×10^7
d5	d4 + yeast	654	91.3	27.3	6.08×10^7

This table shows the number of graphs in each database, the average number of nodes and edges per graph in the databases, and the number of entries in the FragmentIndex.

Table 3. Execution time (in milliseconds) for the 10 disease-associated pathways in KEGG when querying the databases listed in Table 2

Query	No. of nodes	No. of edges	d1	d2	d3	d4	d5
hsa05050	8	10	25.6	28.6	37.1	37.3	37.4
hsa05060	11	15	45.4	53.6	62.0	62.1	62.1
hsa05020	19	10	26.8	36.9	53.7	53.7	53.7
hsa04940	22	2	0.2	0.2	0.2	0.2	0.2
hsa05010	23	17	45.2	58.0	61.9	62.1	62.1
hsa05030	24	13	42.3	42.4	52.9	52.9	53.1
hsa05040	24	28	347.2	431.3	457.4	459.1	462.4
hsa04930	33	36	243.6	411.7	540.6	541.4	546.2
hsa04950	34	33	29.6	29.6	29.6	29.7	29.7
hsa05120	57	26	116.5	160.6	182.8	183.1	183.7

In the current version of SAGA, the Monte Carlo simulation approach is employed to evaluate the statistical significance of the matching results. As the simulation is applied to a large number of random graphs, the cost of computing this statistical significance can be much higher than the query execution cost itself. However, due to the efficiency of the SAGA indexing and matching mechanism, in many cases, the significance test can still be computed quickly. For example, we generated $654 \times 100 = 65\,400$ random graphs for our largest database d5. The *P*-value evaluation for the 10 disease associated queries, on these random graphs, ranges from 1 millisecond for query hsa04940 to 19 seconds for query hsa04930. However, as the number of random graphs used for significance test increases, the overhead will become more significant. In the future, we plan on developing efficient analytical methods for assessing significance of the SAGA matching results.

ACKNOWLEDGEMENTS

The authors thank You Jung Kim for providing valuable feedback on this work. This research was primarily supported by the National Institutes of Health under grant 1-U54-DA021519-01A1, by the National Science Foundation under grant DBI-0543272, and by an unrestricted research gift from Microsoft Corp. Additional funding was provided by grant GR687 from the Michigan Economic Development Corporation, and grant R01-LM008106 from the National Library of Medicine. Funding to pay the Open Access publication charges for this article was provided by the National Institutes of Health under grant 1-U54-DA021519-01A1.

Conflict of Interest: none declared.

REFERENCES

Bron,C. and Kerbosch,J. (1973) Algorithm 457: finding all cliques of an undirected graph. *CACM*, **16**, 575–577.

Coppen,A. (1967) The biochemistry of affective disorders. *Br. J. Psychiatr.*, **113**, 1237–1264.

Chen,M. and Hofestaedt,R. (2004) PathAligner: metabolic pathway retrieval and alignment. *Appl. Bioinformatics*, **3**, 241–252.

He,H. and Singh,A.K. (2006) Closure-tree: an index structure for graph queries. In *Proceedings ICDE 2006*, pp. 38–49.

Hochbaum,D.S. (1997) *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Co., Boston, MA, USA.

Joshi-Tope,G. *et al.* (2005) Reactome: a knowledgebase of biological pathways. *Nucleic Acids Res.*, **33**, D428–D432.

Kalderon,D. (2002) Similarities between the hedgehog and wnt signaling pathways. *Trends Cell Biol.*, **12**, 523–531.

Kanehisa,M. *et al.* (2006) From genomics to chemical genomics: new developments in KEGG. *Nucleic Acids Res.*, **34**, D354–D357.

Kelley,B.P. *et al.* (2004) Pathblast: a tool for alignment of protein interaction networks. *Nucleic Acids Res.*, **32**, W83–W88.

Koyuturk,M. *et al.* (2005) Pairwise local alignment of protein interaction networks guided by models of evolution. In *Proceedings of RECOMB 2005*, 48–65.

Luedde,T. *et al.* (2003) p18(INK4c) collaborates with other CDK-inhibitory proteins in the regenerating liver. *Hepatology*, **37**, 833–841.

Nusse,R. (2003) Wnts and hedgehogs: lipid-modified proteins and similarities in signaling mechanisms at the cell surface. *Development*, **130**, 5297–5305.

Salton,G. and McGill,M. (1983) *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

Sharan,R. *et al.* (2005) Conserved patterns of protein interaction in multiple species. *Proc. Natl Acad Sci. USA*, **102**, 1974–1979.

Shasha,D. *et al.* (2002) Algorithmics and applications of tree and graph searching. In *Proceedings of PODS 2002*, pp. 39–52.

Tatusov,R.L. *et al.* (1997) A genomic perspective on protein families. *Science*, **278**, 631–637.

Tornow,S. and Mewes,H.W. (2003) Functional modules by relating protein interaction networks and gene expression. *Nucleic Acids Res.*, **31**, 6283–6289.

Tourigny,M. *et al.* (2002) CDK inhibitor p18INK4c is required for the generation of functional plasma cells. *Immunity*, **17**, 179–189.

Yan,X. *et al.* (2004) Graph indexing: a frequent structure-based approach. In *Proceedings of SIGMOD 2004*, pp. 335–346.

Yan,X. *et al.* (2005) Substructure similarity search in graph databases. In *Proceedings of SIGMOD 2005*, pp. 766–777.

Yan,X. *et al.* (2006) Searching substructures with superimposed distance. In *Proceedings of ICDE 2006*, pp. 88–99.