QSEA for fuzzy subgraph querying of KEGG pathways

Thair Judeh
tjudeh@wayne.edu

Tin Chi Nguyen
tin.nguyenchi@wayne.edu

Dongxiao Zhu
dzhu@wayne.edu

Department of Computer Science
Wayne State University
Detroit, MI 48202

ABSTRACT

As biological pathway databases continually increase in size and availability, efficient tools and techniques to query these databases are needed to mine useful biological information. A plethora of existing techniques already allow for exact or approximate query matching. Despite initial success, powerful techniques used for XML and RDF query matching have yet to be sufficiently exploited for use in query matching in the bioinformatics domain.

In this paper, we employ the transitive closure to focus on matching hierarchical queries, i.e., finding pathways or graphs that possess a query's overall hierarchical structure. This approach allows for a greater latitude in fuzzy matching by focusing on the overall hierarchies of queries and graphs. Since hierarchies are only inherent in directed acyclic graphs, we have also developed a robust heuristic to heuristically solve the minimum feedback arc set problem. Analysis on 53 *H. sapiens* and 23 *S. cerevisiae* cyclic KEGG pathways have shown that our heuristic performs quite favorably. We have implemented the techniques in an easy to use GUI software QSEA (Query Structure Enrichment Analysis). Binaries are freely available at `http://code.google.com/p/s-e-a/` for Windows and MAC.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*graph algorithms, network problems*

General Terms

Algorithms

Keywords

Fuzzy subgraph querying, Feedback arc set

1. INTRODUCTION

Many resources are now available that describe the pathways of different biological processes including KEGG [17, 18], Biocarta (`http://www.biocarta.com`), and Reactome [9, 19]. These databases and others contain a wealth of biological information. Since the number of overall databases and the number of pathways within a database are continuously increasing, extracting meaningful biological insights may be too tedious to do manually. There is a need for various frameworks to at least partially automate the process, and they can be divided into three major categories [22].

First, network alignment is used to compare two or more networks of the same type from different species. Some of its major goals include identifying functional or conserved protein modules and network evolution analysis. Network alignment can also be used to predict novel interactions that may exist in one species but are absent in another.

Another category, network integration, focuses on combining different networks from the same species. These networks may be transcription regulatory networks, protein-protein interaction networks, signaling pathways, and metabolic networks. Some major goals include the identification of conserved modules across several networks, the relationships between different data types, and the prediction of interactions.

Finally, network querying is used to find a subnetwork module or query across a network or database of networks. Some of its major goals include knowledge transfer across species and the identification of conserved or repeated instances of the query across a network or database of networks. In particular, network querying holds great promise to extract useful biological insights from the pathway databases on a large scale and is the focus of this paper.

Currently, there are a variety of different frameworks and tools that perform network querying varying from techniques that only produce exact matches to techniques that produce approximate matches. QPath [23], for example, takes as input a linear query and outputs a linear subpathway. It allows for results that do not contain all of the query proteins and also allows for the introduction of non-query proteins as well. QNet [10] later on extended QPath by supporting tree queries on subnetworks of bounded tree width. Both QPath and QNet rely on the color coding scheme introduced by Alon et al. [1] to identify subnetworks with a simple topology in an underlying network. Another framework, SAGA [25], also performs an approximate matching of the query network to the target network. It calculates a similarity dis-

ACM-BCB'12, October 7-10, 2012, Orlando, FL, USA

tance between the two and takes into account the structural similarity, the number of vertex mismatches, and the number of gap vertices. For more details on current network querying techniques, please refer to [13].

While the current techniques have proven useful and beneficial, there exist other useful techniques in the XML/RDF querying domain that have yet to be fully exploited to query biological pathways. In the XML/RDF querying domain, a great multitude of XML/RDF documents need to be queried both efficiently and accurately. Many techniques in this domain inherently focus on "fuzzy" hierarchical matching from which the bioinformatics domain may greatly benefit from.

Starting with XML querying, historically the focus was on querying twigs, a tree-like "network" where edges are either a direct parent-child relationship or an ancestor-descendant relationship, i.e., reachability in the latter case. One popular algorithm, TwigStack [6], has two major stages. First, it computes partial solutions for query root-to-leaf paths and compactly represents them using a chain of linked stacks. In the second stage, TwigStack merges and joins the partial results to compute twig query results. TwigStack, in conjunction with modified B-trees, can match query twig patterns in sub-linear time.

Vanilla XML documents, however, are trees and do not allow for a robust range of networks. Thus, for RDF documents, a more general and robust representation would be a DAG (Directed Acyclic Graph). For example, TwigStackD [7] is an extension of TwigStack that at its essence uses the transitive closure to process twig queries. Unlike other approaches, though, TwigStackD does not precompute the transitive closure or path indices for graphs. Instead, it represents a DAG using a combination of interval encodings on the aforesaid DAG's spanning tree. It also uses a customized predecessor index to determine the reachability of vertices based on the remaining edges not present in the spanning tree. Using this alternative representation for DAGs, the transitive closure is derived losslessly. TwigStackD can efficiently query twigs against DAGs with quadratic complexity in the average size of query variable bindings and a linear space cost for the data.

Given the powerful techniques available in the XML/RDF domain, it is worthwhile to explore their use to query biological pathways. For QSEA we specifically use the transitive closures of both query and queried graphs to focus more on the shared hierarchies between queries and pathways. The use of transitive closures allow us to focus on DAGs and go beyond linear paths and trees as in QPath and QNet, respectively. We also allow for approximate solutions by allowing any number of unmapped query vertices and absent ancestor-descendant edges from the initial query graph. Finally, we developed a robust heuristic to solve the feedback arc set problem that shows promising potential.

2. METHODS

2.1 Overview

Figures 1 and 2 present the QSEA framework and GUI, respectively. QSEA may be divided into a preprocessing stage and a query processing stage. For the preprocessing stage, QSEA 1) uses the KEGG API to extract all pathways

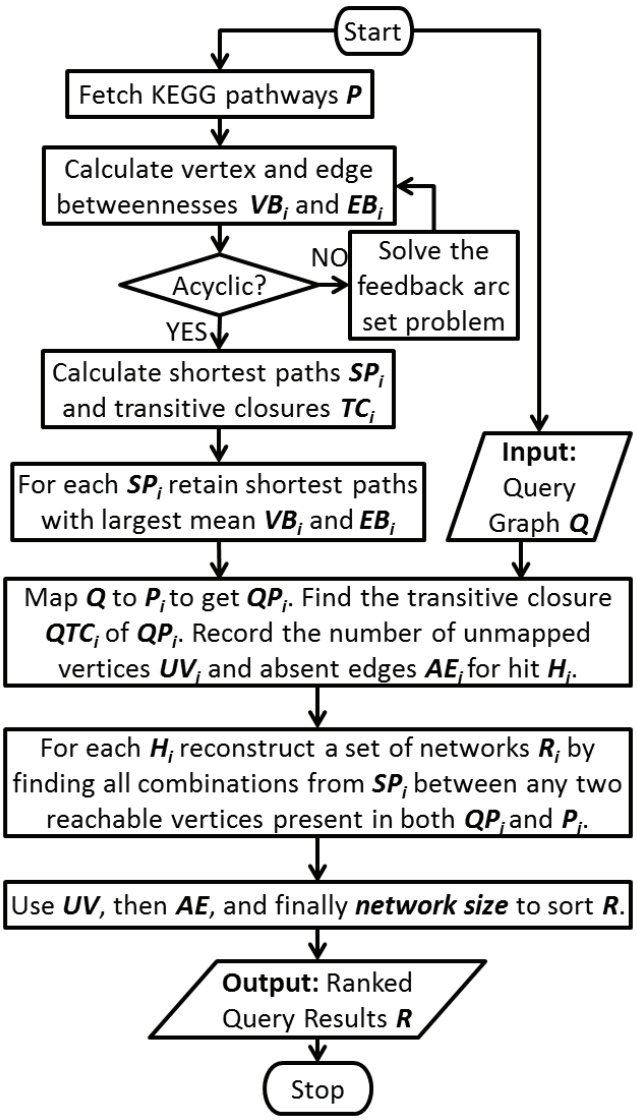

Figure 1: The QSEA framework

P for the selected organism. 2) For each pathway P_i, its edge and vertex betweennesses are calculated. 3) For any pathway with cycles, Algorithm 1 is used to heuristically remove a feedback arc set. For these pathways, their edge and vertex betweennesses are recalculated. 4) All shortest paths and the transitive closure for each pathway are then calculated. 5) For each pathway the shortest paths with largest mean edge and vertex betweennesses are retained with precedence given to edge betweenness. These steps occur only once or when an organism needs to be updated.

For processing queries, QSEA does the following: 1) a user inputs a query graph stored in a space delimited Simple Interaction Format (SIF) file that can also be used by Cytoscape [21]. 2) The user query graph is then mapped to each pathway P_i. Both the number of unmapped query vertices and the number of missing query hierarchical edges, i.e., ancestor-descendant relationships, are recorded for use in sorting later on. 3) A set of results is constructed by

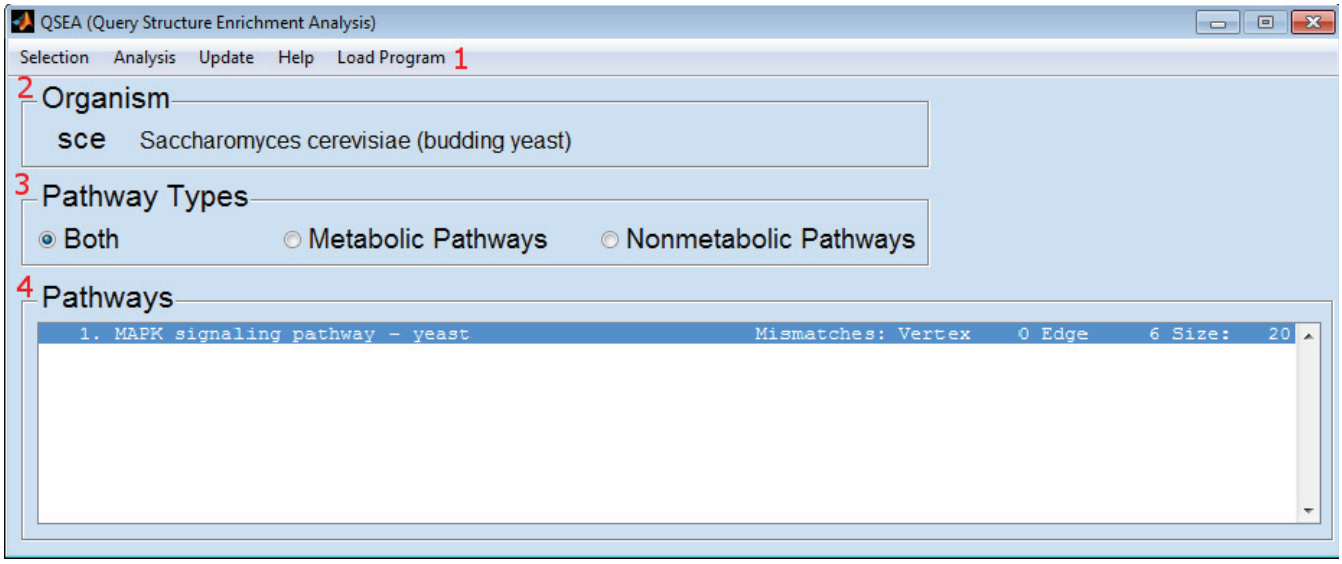

Figure 2: The QSEA GUI. 1) QSEA's menu bar where most of its functionality lies. From QSEA's menu bar, a user can select an organism, load and analyze a query, load previous results, save current results, update/fetch an organism using the KEGG API, and view the quick start guide. 2) This part of QSEA displays the current organism under analysis. 3) QSEA also allows users to select the different types of pathways to examine. 4)QSEA's results are displayed here. Upon clicking a result, the highlighted subpathway within its respective parent pathway is fetched from KEGG as illustrated in Figure 6.

using combinations of all shortest paths between any two reachable query genes that are also reachable in the target pathway. 4) Results are sorted by first maximizing the number of query vertices found, then minimizing the number of missing hierarchical edges, and finally minimizing the size of the result, i.e., the number of "gap" vertices introduced. 5) The results are displayed in the QSEA GUI as illustrated in Figure 2. Upon clicking a result, QSEA will use the default web browser to fetch the KEGG pathway and highlight the query result as illustrated in Figure 6.

2.2 Edge and Vertex Betweennesses

We represent each KEGG pathway as an unweighted, directed graph $G(V, E)$ with vertex set V and edge set E where the KEGG proteins are vertices and the relationships among the proteins form the edges. For each KEGG pathway, its edge and vertex betweennesses are calculated. Originally introduced by Anthonisse [3] and Freeman [14], vertex betweenness is a measure of centrality that quantifies the number of shortest paths that pass through a given vertex. Furthermore, if there are n shortest paths between any two vertices, then a vertex that lies on one of their shortest path receives a contribution of $\frac{1}{n}$. In essence, vertices with high betweennesses scores are quite important as a good number of a graph's shortest paths pass through them. Their loss may significantly impact if not make impossible the flow of information between various vertices.

Edge betweenness is an extension to vertex betweenness except that it applies to edges instead of vertices. Similar to vertices with a high vertex betweenness, an edge with a high edge betweenness has many shortest paths passing through it. In fact, Newman and Girvan [20] used edge betweenness to divide a pathway into different communities or

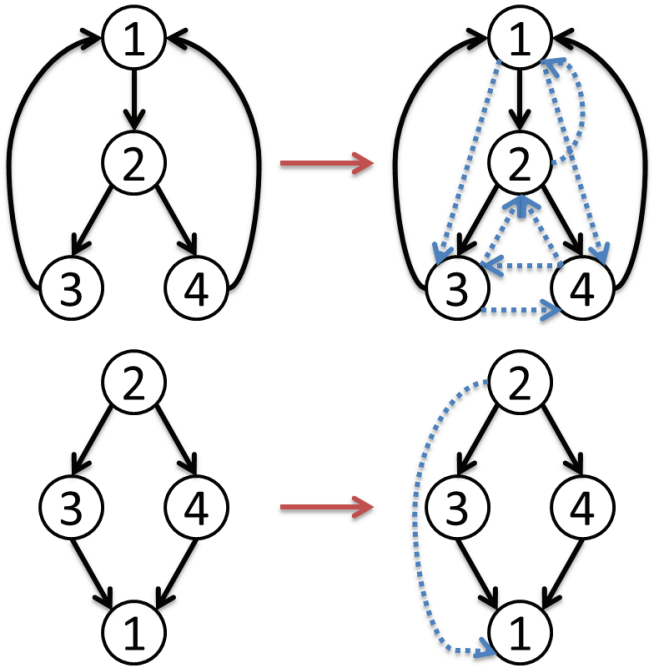

Figure 3: Top: A strongly connected component from the KEGG *H. sapiens MAPK Signaling Pathway*. Its transitive closure on the right is a fully connected graph. In this case, all DAG queries would result in a query hit. Bottom: Removing the edge $1 \rightarrow 2$ produces a diamond subgraph, which happens to be a common biological network motif [2]. Its transitive closure on the right possess a hierarchy of vertices unlike the original subgraph.

Algorithm 1: A heuristic to remove feedback arc sets

Input: An unweighted, directed graph $G(V, E)$ with vertex set V and edge set E
Output: The acyclic graph $G(V, E - E_{FAS})$ where E_{FAS} is a feedback arc set
Remove all self-loops from G
Find all non-trivial strongly connected components S using Tarjan's algorithm [24]
Find edge and vertex betweennesses EB and VB
Extract from G the adjacency lists A
`/* Prioritizes exploring edges with high betweennesses scores */`
Sort A first according to EB and then VB in descending order
for i = 1,...,$|S|$ **do**
    Run **SCC-DFS**(S_i)
end
Return G which is now acyclic. The set of edge sets $E_{S_i,j,FAS}$ removed from the strongly connected components form E_{FAS}
`/* Strongly Connected Component Depth-First Search */`
SCC-DFS
Sort the vertices V_{S_i} of S_i in descending order of vertex betweenness
Extract from A the set of sorted adjacency lists A_i that represents S_i
for j = 1,...,$|S_i|$ **do**
    Begin a depth-first search at vertex $V_{S_i,j}$ using A_i to prioritize the order of vertices to visit. Remove all of the back edges found in the depth-first search to obtain $E_{S_i,j,FAS}$.
end
Remove from E the set of edges $E_{S_i,j,FAS}$ that minimizes first the number of edges removed and then the edge betweenness of the edges removed

Algorithm 2: Minimum feedback arc set removal

Input: An unweighted, directed graph $G(V, E)$ with vertex set V and edge set E
Output: The acyclic graph $G(V, E - E_{MFAS})$ where E_{MFAS} is the minimum feedback arc set
Remove all self-loops from G
Find all non-trivial strongly connected components S using Tarjan's algorithm [24]
for i = 1,...,$|S|$ **do**
    Extract the subgraph G_i that represents S_i
    Set the number of edges to remove R to 0
    while G_i *is cyclic* **do**
        Increment R by 1
        Find all combinations C of the edges E_{S_i} choosing R edges at a time
        for j = 1,...,$|C|$ **do**
            Reverse the set of edges C_j in G_i
            if G_i *is acyclic* **then**
                Break from the inner for loop
            end
            Undo the edge set reversal
        end
    end
    Remove the last edge set C_j from E where C_j is a minimum feedback arc set for G_i
end
Return G which is now acyclic. Collectively, all Cs previously removed form E_{MFAS}

modules, and they provided an efficient $O(VE)$ algorithm as well. It should be noted that Brandes [5] has also presented an $O(VE)$ algorithm for vertex betweenness that can be extended to edge betweenness. In our case, we use edge and vertex betweennesses to both guide a depth-first search (DFS) and select shortest paths that are most significant.

2.3 Feedback Arc Set

For its fuzzy and hierarchical querying, QSEA relies on the transitive closure. Briefly, for any directed graph $G(V, E)$ with vertex set V and edge set E, its transitive closure TC is a concise representation of the reachability of the vertices in V. More specifically, TC's vertex set V_{TC} equals V while its edge set E_{TC} is a superset of E. Furthermore, an edge (i, j) in E_{TC} either denotes the presence of an edge (i, j) in E or the presence of a series of edges in E that can be traversed to reach vertex j starting from vertex i . The transitive closure may be calculated using the Floyd-Warshall algorithm in $O(|V|^3)$ since the existence of a shortest path from vertex i to vertex j means that j is reachable from i. The transitive closure may also be computed in $O(|V|^{2.376})$. For more details on efficiently calculating the transitive closure, please refer to [4].

Furthermore, it should be noted that for DAGs, the transitive closure can represent a hierarchy of vertices. It is this property that QSEA exploits to process a query graph Q against a queried graph P since an exact query hit occurs if and only if E_{TC_Q} is a subset of E_{TC_P}. In other words, one can also find a topological sort ordering that is common to the transitive closures of the query graph Q and queried graph P. Briefly, a topological sort ordering is a non-unique linear ordering of a DAG's vertices such that for any edge (i, j), i will always appear before j in the linear ordering. As a result, since QSEA relies on the transitive closure for its fuzzy and hierarchical querying, it is of particular importance that both Q and P are DAGs. Otherwise, it is not possible to exploit a hierarchy of vertices in the presence of cycles as illustrated in Figure 3. In Figure 3, the top subgraph is a strongly connected component from the KEGG *H. sapiens MAPK Signaling Pathway*. Its transitive closure is a fully connected graph. As a result, any query graph without self-loops will satisfy the subset condition and will result in a query hit, which does not allow for any discriminative biological insights. The diamond subgraph on the bottom of Figure 3, on the other hand, is a DAG and possesses an meaningful hierarchy for querying. In this case, only a subset of queries will result in a query hit.

To ensure that both the query graphs and queried graphs are DAGs, QSEA uses two different approaches. For query graphs, the solution is to simply restrict all queries to DAGs. For the queried KEGG pathways, however, such an option may remove meaningful pathways. For *H. sapiens*, *M. musculus*, and *S. cerevisiae*, about 25% of their respective pathways have cycles (53 out of 229, 53 out of 225, and 23 out

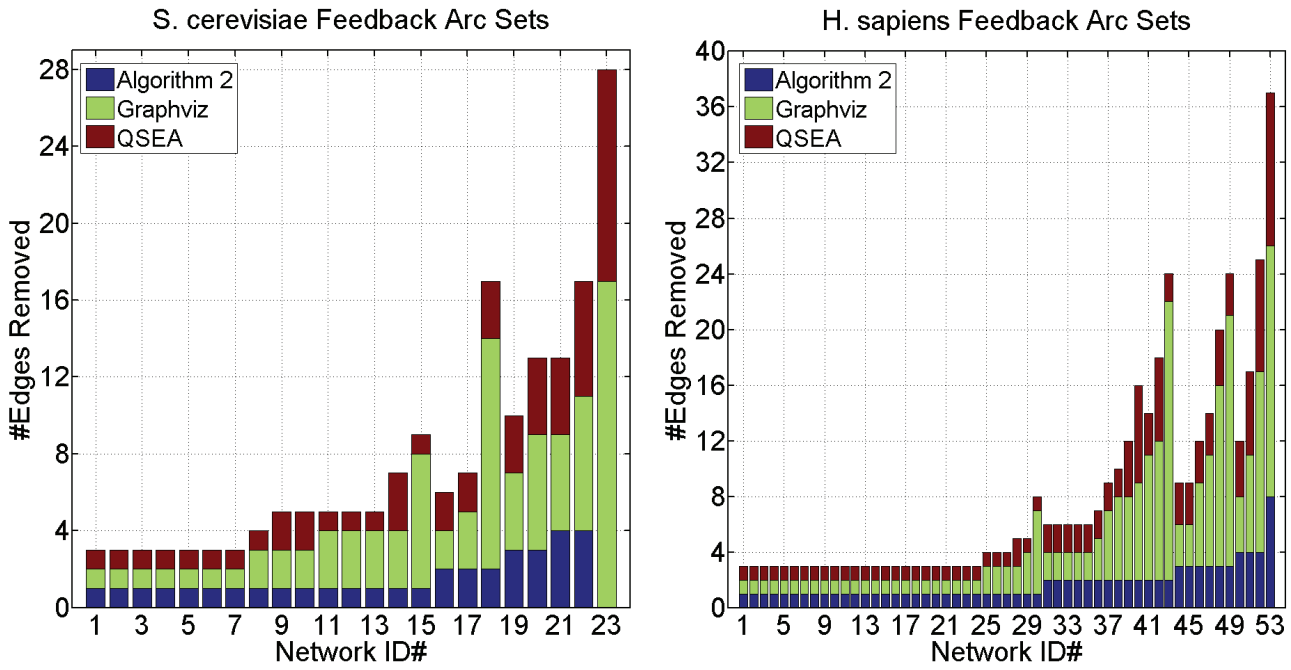

Figure 4: The number of edges removed from 23 cyclic *S. cerevisiae* and 53 cyclic *H. sapiens* KEGG pathways, respectively. In both data sets, QSEA equals or outperforms Graphviz 100% of the time. For *S. cerevisiae* the last column corresponds to the KEGG pathway *Glycine, serine and threonine metabolism*. It has a non-trivial strongly connected component of 14 vertices and 88 edges. Algorithm 2 was unable to find the minimum feedback arc set due to insufficient memory.

of 78, respectively). Simply removing them would lead to a loss in significant pathways such as *Cell Cycle* in all three species. As such, QSEA uses a robust heuristic to remove cycles from the pathways while preserving the overall directional flow of a pathway.

First, we present some background on the underlying problem. For a directed graph $G(V, E)$, $E_{FAS} \subset E$ is a feedback arc set if the removal of E_{FAS} makes G acyclic [4]. In particular, a minimum feedback arc set $E_{MFAS} \subset E$ is the minimum number of edges whose reversal makes G acyclic [4]. For arbitrary graphs, the problem is known to be NP-hard, and the best known approximation has ratio $O(\log |V| \log \log |V|)$ [12].

Heuristically solving the feedback arc set is also a well known subproblem in drawing directed graphs with minimal edge crossings. For example, Graphviz [11] uses a depth-first search (DFS) to heuristically eliminate some edges to break cycles in order to rank the vertices. Briefly, DFS divides edges into tree edges and non-tree edges consisting of forward edges, back edges, and cross edges [8]. Edges whose vertices are visited for the first time form the tree edges of a depth-first forest. Forward edges are non-tree edges that directly connect a vertex i with a descendant vertex j in a depth-first tree whereas back edges are the opposite, i.e., they directly connect a descendant j with an ancestor i. Finally, any other non-tree edge is classified as a cross edge.

It should be straightforward to observe that all back edges found by DFS form a feedback arc set. As such, Graphviz takes one non-trivial strongly component, i.e., a subgraph in which any two vertices are connected either directly or indirectly through a number of intermediate vertices, and counts the number of times an edge in the strongly connected component forms a back edge via a depth-first search. The edge with maximal count is removed, and the process is repeated until no strongly connected components remain [15].

QSEA takes a similar approach to Graphviz with some major differences as described in Algorithm 1. The most major difference is the use of vertex and edge betweennesses as presented in Section 2.2 to guide DFS when it chooses vertices to visit and edges to explore. By using betweenness to guide DFS, QSEA greedily focuses on edges and vertices with high betweenness. This allows the forest of trees generated by DFS to be extracted deterministically. More importantly, though, it is hoped that the edges in cycles that become back edges have a low betweenness score, which in turn may be of less importance for the overall pathway.

To compare the performance of QSEA and Graphviz, Algorithm 2, outlined in [16], is used as a reference point as it is able to determine a minimum feedback arc set for each strongly connected component. Algorithm 2 is able to find E_{MFAS} via a naïve approach that checks all possible combinations of edges for a given number of edges R. Given a non-trivial strongly component S with edge set E_S and $R = |E_{S,MFAS}|$, Algorithm 2 has to check $\sum_{e=1}^{R} \binom{|E_S|}{e}$ edge combinations in G in order to find the minimum feedback arc set for S. This process may be unfeasible for relatively small values of E_S as seen in Figure 4.

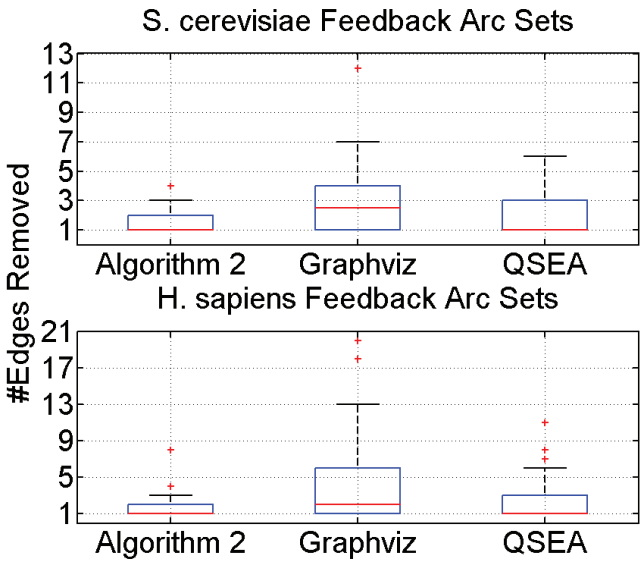

Figure 5: A box plot view of the results from Figure 4. It should be noted that the KEGG pathway *Glycine, serine and threonine metabolism* was removed from the *S. cerevisiae* box plot since Algorithm 2 failed due to insufficient memory.

In Figure 4, we compared the performance of QSEA (Algorithm 1), a naïve approach (Algorithm 2), and Graphviz on 23 cyclic *S. cerevisiae* and 53 cyclic *H. sapiens* KEGG pathways, respectively. It should be noted that we focused solely on the size of the feedback arc sets removed and do not take into account the actual edges removed by either algorithm. As illustrated in Figure 4, QSEA outperforms Graphviz 100% of the time.

We can also observe from Figure 4 that both heuristics perform quite well in general and are able to remove a minimum feedback arc set for some of the pathways selected. When they differ, though, QSEA removes less edges overall compared to Graphviz as indicated by their respective Euclidean distances from Algorithm 2. Briefly, we calculated the Euclidean distance as

$$\sqrt{\sum_{i=1}^{n}(E_{MFAS_i} - E_{FAS_i})^2}$$

where E_{MFAS} corresponds to the vector of minimum feedback arc set sizes removed by Algorithm 2 and E_{FAS} corresponds to the vector of feedback arc set sizes removed by either QSEA or Graphviz. For *S. cerevisiae* QSEA has a Euclidean distance of 3.46 from Algorithm 2 as opposed to Graphviz's Euclidean distance of 13.27. For *H. sapiens* QSEA has a Euclidean distance of 8.78 from Algorithm 2 as opposed to Graphviz's Euclidean distance of 32.95. These results are further illustrated in Figure 5.

2.4 Shortest Paths and Transitive Closure

After the feedback arc sets are removed and vertex and edge betweennesses are recalculated, all shortest paths between any two vertices i and j are calculated. Since there may be multiple shortest paths between any two vertices, only the shortest paths with largest mean edge and vertex betweennesses are kept while prioritizing edge betweenness. We hypothesize that these shortest paths play a more significant role compared to other shortest paths as they capture a larger snapshot of the pathway or graph at a global level. Regardless, there may still be multiple shortest paths between any two vertices i and j. As such, QSEA will display all combinations of shortest paths in the results when needed. The transitive closures for each pathway are also calculated and stored.

2.5 Query Matching and Output

Once preprocessing is complete, QSEA is ready to take as input user query graphs that follow a space-delimited Simple Interaction Format (SIF). Using the KEGG API, QSEA is able to support a variety of labeling schemes including NCBI GeneID, NCBI GI, GenBank, UniGene, and UniProt. In the SIF file, users can construct two types of edges similar to TWIGs presented in the introduction. The first edge is a direct parent-child edge in which vertex i connects directly to vertex j. The second edge is an ancestor-descendant edge in which there may be from 0 to any number of arbitrary vertices between i and j. Finally, it should be noted that a gene may map to multiple locations in a KEGG pathway, which QSEA is able to handle as illustrated in Figure 6.

For query matching QSEA is able to support both exact and approximate query graph solutions. For this purpose, after QSEA maps a query Q onto a pathway P to get QP_i, QSEA makes note of the number of unmapped query vertices UV_i for each pathway P_i. QSEA then also calculates the transitive closure for QP_i to get QTC_i. First, for direct edges mentioned previously, QSEA requires an exact match. Otherwise, for ancestor-descendant edges, QSEA allows for any number of mismatches. Thus, QSEA also takes into account the number of absent ancestor-descendant edges AE. Once this process is complete, QSEA constructs a set of graphs as results from all possible combinations of shortest paths between two vertices i and j that are reachable both in P_i and QP_i.

After processing all of the pathways and extracting their resulting graphs, QSEA ranks them according to three criteria. First, it maximizes the presence of query vertices. Then, it minimizes the number of absent ancestor-descendant edges. Finally, it minimizes the size of the resulting graphs. Using these criteria, QSEA prioritizes query results that possess all of the query vertices and edges with minimal size.

An example of the query matching and output is presented in Figure 6. First, a SIF file is constructed consisting of two direct parent-child edges and three ancestor-descendant edges. The SIF file is then processed into a graph where each vertex uniquely represents a gene. After mapping the query graph to its MAPK signaling pathway version, a query hit is found. The highlighted query result is fetched from KEGG and displayed in a user's default web browser.

3. CONCLUSIONS

We presented QSEA (Query Structure Enrichment Analysis). Our contributions are two-fold. First, we introduced the use of XML/ RDF techniques into biological pathway querying. Specifically, the use of the transitive closure allows

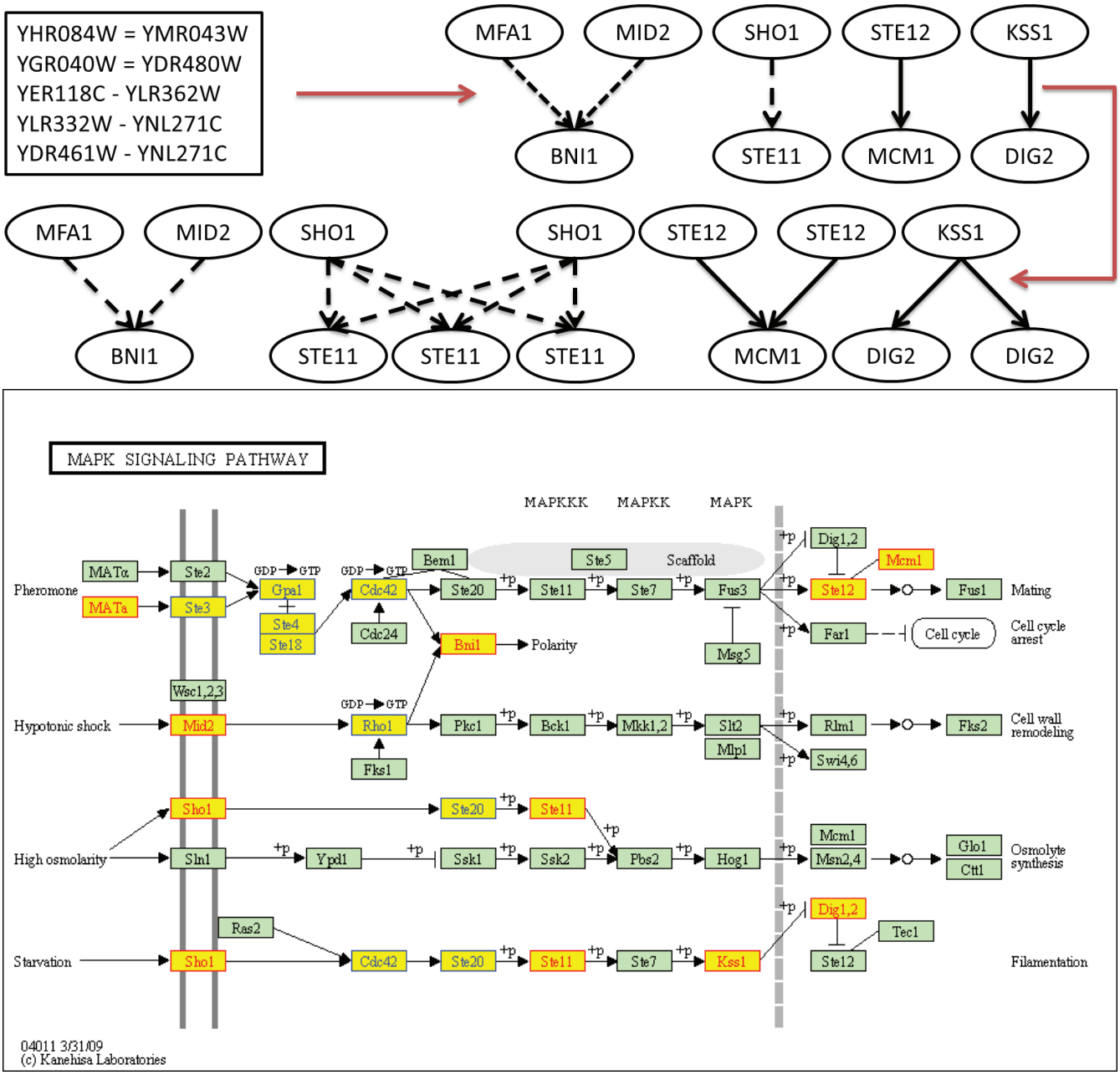

Figure 6: Top Left: An example query file. A minus sign represents an ancestor-descendant edge whereas an equal sign represents a direct parent-child edge. In this case, there are two direct parent-child edges and three ancestor-descendant edges in the query. Top right: The query represented as a graph. Dashed lines represent ancestor-descendant edges whereas solid lines represent. For ease of cross-referencing with the KEGG results below, gene names are used. Middle: QSEA transforms the query into its pathway specific form on the right for the KEGG *S. cerevisiae MAPK signaling pathway*, which has two instances of SHO1, three instances of STE11, two instances of STE12, and two instances of DIG2. Bottom: QSEA is able to display simultaneously multiple instances of a gene all at once. All vertices highlighted in yellow are part of the query result. Depending on the foreground color, red denotes a query vertex whereas blue denotes a "gap" vertex introduced to connect the query vertices. The network has a final score of 0 for vertex mismatches, 6 for edge mismatches due to redundant edges, and 20 for network size.

for focus on matching hierarchies between queries and their target pathways. Second, we introduced a robust heuristic to solve the feedback arc set problem which has a promising performance. Our contributions are implemented in an easy-to-use GUI software. Binaries for QSEA are freely available at `http://code.google.com/p/s-e-a/` for Windows and MAC.

For future work, we hope to test the potential of the QSEA's heuristic for the feedback arc set find the "information flow" of a graph. We also hope to develop a more robust score function rank query results. Finally, we hope to compare feedback arc sets found by various algorithms and compare them in terms of biological functional importance in addition to size.

4. ACKNOWLEDGMENTS

We would like to thank Nicolas Bruno for providing us with the source code for PathStack and TwigStack. We would also like to thank the reviewers for their helpful and useful feedback.

5. REFERENCES

[1] N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, July 1995.

[2] U. Alon. Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461, 2007.

[3] J. M. Anthonisse. The rush in a directed graph. Technical report, Stichting Mathemastisch Centrum, Amsterdam, The Netherlands, 1971.

[4] J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[5] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[6] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal xml pattern matching. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 310–321. ACM, 2002.

[7] L. Chen, A. Gupta, and M. E. Kurul. Stack-based algorithms for pattern matching on dags. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 493–504. VLDB Endowment, 2005.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[9] D. Croft, G. O'Kelly, G. Wu, R. Haw, M. Gillespie, L. Matthews, M. Caudy, P. Garapati, G. Gopinath, B. Jassal, S. Jupe, I. Kalatskaya, S. Mahajan, B. May, N. Ndegwa, E. Schmidt, V. Shamovsky, C. Yung, E. Birney, H. Hermjakob, P. D'Eustachio, and L. Stein. Reactome: a database of reactions, pathways and biological processes. *Nucleic Acids Research*, 39(Database issue):D691–D697, Jan. 2011.

[10] B. Dost, T. Shlomi, N. Gupta, E. Ruppin, V. Bafna, and R. Sharan. QNet: A Tool for Querying Protein Interaction Networks. *Journal of Computational Biology*, 15(7):913–925, 2008.

[11] J. Ellson, E. Gansner, L. Koutsofios, S. North, G. Woodhull, S. Description, and L. Technologies. Graphviz - open source graph drawing tools. In *Lecture Notes in Computer Science*, pages 483–484. Springer-Verlag, 2001.

[12] G. Even, J. S. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *ALGORITHMICA*, 20:151–174, 1998.

[13] V. Fionda and L. Palopoli. Biological network querying techniques: Analysis and comparison. *Journal of Computational Biology*, 18(4):595–625, 2011.

[14] L. C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, Mar. 1977.

[15] E. R. Gansner, E. Koutsofios, S. C. North, and K. phong Vo. A technique for drawing directed graphs. *IEEE Transactions on Software Engineering*, 19:214–230, 1993.

[16] I. Ispolatov and S. Maslov. Detection of the dominant direction of information flow and feedback links in densely interconnected regulatory networks. *BMC Bioinformatics*, 9(1):424, 2008.

[17] M. Kanehisa and S. Goto. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Research*, 28(1):27–30, 2000.

[18] M. Kanehisa, S. Goto, Y. Sato, M. Furumichi, and M. Tanabe. Kegg for integration and interpretation of large-scale molecular data sets. *Nucleic Acids Research*, 40(D1):D109–D114, 2012.

[19] L. Matthews, G. Gopinath, M. Gillespie, M. Caudy, D. Croft, B. de Bono, P. Garapati, J. Hemish, H. Hermjakob, B. Jassal, A. Kanapin, S. Lewis, S. Mahajan, B. May, E. Schmidt, I. Vastrik, G. Wu, E. Birney, L. Stein, and P. D'Eustachio. Reactome knowledgebase of human biological pathways and processes. *Nucleic Acids Research*, 37(Database issue):D619–622, Jan. 2009.

[20] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 69(2):026113, Feb 2004.

[21] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks. *Genome Research*, 13(11):2498–2504, Nov. 2003.

[22] R. Sharan and T. Ideker. Modeling cellular machinery through biological network comparison. *Nature Biotechnology*, 24:427–433, 2006.

[23] T. Shlomi, D. Segal, E. Ruppin, and R. Sharan. Qpath: a method for querying pathways in a protein-protein interaction network. *BMC Bioinformatics*, 7(1):199, 2006.

[24] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

[25] Y. Tian, R. C. McEachin, C. Santos, D. J. States, and J. M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2):232–239, 2007.