Graph Search Beyond Text: Relational Searches in Semantic Hyperlinked Data

M. Goldberg*, J. Greenman*, B. Gutting*, M. Magdon-Ismail*, J. Schwartz*, W. Wallace†
* Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180.
Email: {goldberg, green7, guttiba, magdon, schwaj3}@cs.rpi.edu
† Industrial and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY 12180.
Email: wallaw@rpi.edu

***Abstract*—We present novel indexing and searching schemes for semantic graphs based on the notion of the i.degrees of a node. The i.degrees allow searches performed on the graph to use "type" and connection information, rather than textual labels, to identify nodes. We aim to identify a network graph (fragment) within a large semantic graph (database). A fragment may represent incomplete information that a researcher has collected on a sub-network of interest. While textual labels might be available, they are highly unreliable, and cannot be used for identification of hidden networks. Since this problem comes from the classically NP-hard problem of identifying isomorphic subgraphs, our algorithms are heuristic.**

I. Introduction

In this paper we present an approach to searching large databases for small networks. This technique does not use text comparison, but instead leverages destinquishible types, and the network topology. In this problem the database and the querys (which we will call *fragments*) are each represented as semantic graphs. The following practical situation motivates the problem. A researcher discovered a small hidden network that connects people, events, places, etc., and presented the result as a *fragment*. The researcher expects that the fragment can also be found in a central database. The database will have information not in the fragment, and the fragment may also contain information not in the database. To compare the fragment with the data in the central database, the problem of fragment identification must be solved. While text-labels, such as "Joe" for a name, may be incorrect, the "category" or "type" of the node, such as "person" is much more likely to be correct. In this paper, we make the assumption that all objects in the database and in the fragment are assigned with reliable types. Since there are only a small number of types, they partition the set of objects into large groups which make it impossible to uniquely identify all objects. The search algorithms that we developed identify the fragment within the database by using the relationships/links between the objects and their types in the fragment, and by attempting to find a "similar" pattern in the database.

There are two specific subproblems that arise: **(1)** searching for a fragment which does not contain information new to the database; and **(2)** approximate matching of the fragment which may contain new information. We focused on the first of these problems. In the theoretical setting, the problem relates to the well known NP-hard optimization problem of subgraph isomorphism([1, 2]).

We apply the general indexing idea ([3, 5]) to our problem of searching for a sub-structure (a fragment match) in a large structure (the database graph), and present it as a two-stage process: **(1)** an off-line indexing of the graph database; and **(2)** an on-line search which employs the results of indexing. In the practical setting, indexing would be performed infrequently. The indexing-enabled search would have to be executed for every specific search request and is expected to be very fast.

We index an arbitrary semantic graph via the computing of i.degrees of its nodes. The i.degrees of the nodes are used to construct a hierarchical set of node-partitionings, where each next partitioning is a sub-partitioning of the previous one.

Our search algorithm starts by computing the i.degrees of the nodes of the fragment and selecting some representative node, *the anchor*. The i.degrees of the anchor are then compared to that of the database nodes, and a few of the most similar nodes are returned. The identification of a match for the anchor is an important first step in the general fragment identification problem.

II. Definitions

We assume that the nodes of the input graph are labeled by "non-confusable" categories, or types such as "person," "place," "event," and so on. The types are distinct from general labels that may be incorrect, and often are not reliable. Thus, while a person is not confused with an event, the names (labels) "John" and "Josef" can be confused. In real-life applications, the user may deal with labels that s/he considers completely reliable, so those labels can be used as types. For convenience, we

assume that types are enumerated $1, 2, 3, \ldots, t$, where t is the total number of types used. Our graphs are directed; all notations used but not defined here can be found in [4].

i.outdeg: measure of the types in a nodes i.Neighborhood. Given node x, the $i.outdeg(x, k)$ is the number of paths of length i that start at x and end at nodes of type k. The $0.outdeg(x, k)$ will be 0 for all types k except the type of x, where it is 1.

i.indeg: given node x, the $i.indeg(x, k)$ is the number of paths of length i that start at nodes of type k and end at x.

i.dist: this function computes the "dissimilarity" of two nodes of the same type by comparing their $i.(in/out)deg$. This is calculated by the L_1 distance between the vectors:

$$i.dist(v, w) = \sum_{j=1}^{i} \sum_{k=1}^{t} |j.deg(v, k) - j.deg(w, k)|$$

where the deg function is either $indeg$ or $outdeg$.

Dominating i.dist: this requires that the first node's i.deg dominates the second, that is at every k^{th} position the first is not less than the second; else, the dominating i.dist is considered to be some very large constant.

Fuzziness: this is a measure of the number of nodes within a graph for which a particular node v is similar to. Two nodes may be considered r-fuzzy to each other if $similarity(v, w) < r$ for some $r \geq 0$.

Indexing: indexing is the process of assigning values to nodes as a preprocessing step. Each node's i.(in/out)degs for $i = [0, k]$ are calculated; indexing yields a hierarchical partitioning of the node-set.

$N(x)$: this denotes the set of neighbors of x. The neighbors are the nodes that have incoming edges from x. That is: $v \in N(x)$ iff $(x, v) \in E$

Diversity: $dvst(x)$ is the count of the unique types in $N(x)$; a node x is more diverse than a node y if $dvst(x) > dvst(y)$ or if $dvst(x) = dvst(y)$ and $|N(x)| > |N(y)|$.

III. Search Algorithm

This section details the algorithms used for locating matches to the anchor within the database. We assume the database has been pre-processed; each node is assigned i.degrees, for $i = [0, k]$, where k is a preselected integer.

Before the query step, we have an indexed database graph G, and a query fragment graph F. We index F using the same range $i = [0, k]$ as was used for indexing G. Also, we are either given, or obtain algorithmically, an anchor node x from F.

Once an anchor x is specified, we search for a match to x within the database. All nodes of the database are ranked using the preselected similarity method, i.distance or dominating i.distance. Then the top n candidates are returned, where n is predetermined.

The basic approach we originally developed is with a single node using similarity of $i.deg$. A more advanced method makes use of the anchor and some number, m, of its neighbors . These nodes are the m most diverse neighbors of x.(See Algorithm 1)

We have improved on this by using a more sophisticated method through filtering. We filter the candidates based on if they pass some function. For our tests, we used dominating i.distance for filtering. If the candidate passes the filter, then the algorithm checks if it is possible for all the neighbors of x to have a match among the neighbors of v such that each pair also meets the filter requirement. This is done using bipartite matching from the Boost Graph Library. If the candidate can pass the filter, it is scored; at the end the best candidates are returned.

Algorithm 1 Fragment Filter Search

Require: 1. fragment graph F
2. anchor node $x \in F$
3. database graph G
4. H filter function
5. S score function
6. n for the number of candidates to return

for v s.t. $v \in G$ AND $type(v) = type(x)$ **do**
  if $H(v, x) = pass$ **then**
    if $\exists$ a bipartite matching $M : N_1(x) \rightarrow N_1(v)$ s.t. $\forall (x_j, v_k) \in M, H(v_k, x_j) = pass$ **then**
      $score \leftarrow S(v, x)$
      $candidates \leftarrow candidates \cup$ pair$(score, v)$
return the best n $candidates$

IV. Graph Generation

Wikipedia Graph

The main graphs we used for testing were constructed from Wikipedia. Each page is a node and infobox pagelinks are edges. Some, but not all, pages also have semantic information, including types. Only pages with semantic type information are used in the graph.

We used DBpedia.org[?], a website which stores a variety of Wikipedia databases. We used "Ontology Infobox Types" to gather the node and type information and "Raw Infobox Properties" for the edges. Nodes which had neither incoming nor outgoing neighbors were removed from the graph.

Types obtained from the semantic information form 26 distinct main types, which partition the pages non-uniformly. Each type has some number of subtypes

which further partition the pages with that type. We subpartition nodes with type "place" to obtain a 42 type "enhanced" graph.

Random Graph We used several methods to obtain random graphs throughout our experiments. We do not present the results here, however we found that searching random graphs to be a much more trivial task than searching real networks.

V. Fragment Generation Algorithms

As part of our testing methodology, we sought to extract fragments from the database, anonymize them, and relocate them within the database using our algorithm. The fragment generation method uses a random walk with restarts. In our testing we used $p = 0.9$ and $\alpha = 0.7$. Note that the subgraphs generated are induced subgraphs on the nodes: all possible edges that existed in the database graph G will exist in the fragment graph F.

Algorithm 2 *Fragment Generation P-Random Walk 'p'*

Require: 1. graph G
2. maximum size n
3. probability p
4. constant α

Choose randomly $x \in G$ such that $|N(x)| > 2$
Add x to $V(F)$
$u \leftarrow x$
$previousSize \leftarrow 1$
while $|V(F)| < n \wedge p >= p * (\alpha^n)$ **do**
  Choose at random a node w from $N(u \in G)$
  Attempt to add w to $V(F)$
  $u \leftarrow w$
  $restart \leftarrow false$
  With probability p, $restart \leftarrow true$
  if restart **then**
    $u \leftarrow x$
    if $|V(F)| = previousSize$ **then**
      $p \leftarrow p \times \alpha$
    $previousSize = |V(F)|$

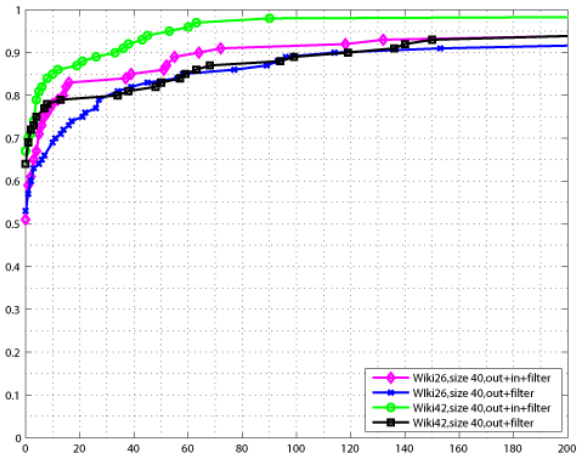
VI. Validation and Results

To determine the filtering algorithm's success, we tested it on both random and wikipedia-based graphs.

As we expand to include not only the outdegrees of nodes, but also the indegrees, we see gains averaging between 10-15% in both the Minimal and Enhanced wiki-graphs (Figure 1) for fragments of size 40.

These tests were performed over 50 samples. When we consider indegree in addition to outdegree the time taken on average does increase, however if we also use our bipartitie-matching filtering technique the average time decreases greatly. This is because many candidates can be filtered out before the bipartite matching step is necessary.

Fig. 1. Filtering with 3-out vs. 3-out/in in Minimal and Enhanced Wiki Graphs



Acknowledgment. This research was supported in part by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053, the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-12-C-7202, and the U.S. Department of Homeland Security (DHS) under Agreement 2009-ST-061-CC1002-02.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Army Research Laboratory, IARPA, AFRL, DHS, or the U.S. Government.

References

[1] M. R. Garey and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness. *W.H. Freeman and Company*, 1979.

[2] M. Goldberg. The graph isomorphism problem. *Handbook of Graph Theory*, pages 68 – 78, 2003.

[3] J. Y. Shijie Zhang, Meng Hu. Treepi: A novel graph indexing method. *Proc. of ICDE 2007; 23rd IEEE International Conference on Data Engineering*, pages 966–975, 2007.

[4] D. B. West. Introduction to graph theory. *Prentice Hall, Upper Saddle River, NJ*, 2003.

[5] J. H. Xifeng Yan, Philip S. Yu. Graph indexing: a frequent structure-based approach. *Proc. of the 2004 ACM SIGMOD International Conference on Management of Data*, pages 335–346, 2004.