

# Toward Efficient Search for a Fragment Network in a Large Semantic Database

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; wallaw@rpi.edu

**Abstract**—In this paper, we present a novel indexing scheme for semantic graphs, which is based on the notion of the  $i$ -degree of a node. We apply this scheme to identify in a large semantic graph, the database, a fragment network, which may represent incomplete information that an analyst collected on an adversary network of interest.

## I. INTRODUCTION.

We present an approach to searching in a large semantic graph, the *database*, for a subgraph which approximately matches a small network, the *fragment*. The following practical situation motivates the problem. An intelligence analyst has discovered a small hidden network that connects people, events, places, etc., and presented the result of the investigation as the fragment graph. Suppose the analyst wishes to compare the data s/he has collected so far (the fragment) with the data in a central database which contains the union of accumulated intelligence. While the information collected by the analyst might be scarce, incomplete, and even unreliable, the information in the database accumulated by other analysts over a long period of time should be more complete and reliable. Inversely, the newly accumulated data may contain items not found, or not available, earlier. The task is to find, in the central database, statistically significant approximate matches to the fragment.

The difficulty of the problem arises from its close relation to the classical NP-hard computational problem of identifying, in a given large graph, a subgraph which is isomorphic to a given fragment graph ([1, 2]). On the other hand, the nature of the input— semantic graphs with labeled nodes and edges— offers a possibility to achieve efficiency for an average input. However, the theoretical aspect of the problem is only one part of the challenge. The main difficulty of the problem stems from the lack of a strict definition of an “approximate match”. In the real life situation, the analyst working on the problem relies on his/her intuition and experience to identify a significant “likelihood” of his/her guess. Thus, to develop a system that helps the analyst, it is necessary to create a tool which can easily incorporate this knowledge and intuition. Our approach can easily accommodate the analyst’s input, or *hints*, during deployment.

A similar problem is that of querying a database of graphs. Given a graph query, the task is to quickly retrieve graphs from a large database which contain the query. The traditional approach ([4, 6]) is to perform the search via graph-based indices. We apply the general indexing idea to our problem of searching for a substructure (a fragment match) in a large structure (the database graph), and present it as a two-stage process: **(a)** an off-line indexing of the graph database; and **(b)** an on-line search which employs the results of indexing. In the practical setting, indexing would be performed infrequently, while 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. It turns out, that for the classes of graphs that we tested (large random graphs and the Wikipedia-graph on 1.1 million nodes) even the 3-degree partitioning splits the node set in partitions of average size close to 2. This implies that when the fragment contains a near-complete information about some node, it can be located in the database almost uniquely. This property is useful even if the information is not complete.

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 is then compared to that of the database nodes, and a few “closest” nodes are returned. The identification of a match for the anchor is an important first step in the general fragment identification problem. This anchor search can be complemented by finding *network communities* that the anchor is a member of (see [3]).

We test our search algorithms on two classes of inputs: randomly generated graph and the Wikipedia-graph (see Section IV.) It turns out that the search on randomly generated graphs yields very good results. Thus random graphs are easy, as opposed to the real life graphs. It is considerably harder to achieve good accuracy for the Wikipedia-graph due to its non-uniformity. It is likely

that any successful search would have to be based on assumptions provided by an analyst related to the “shape” of the subgraph to match a given fragment.

## II. DEFINITIONS

We assume that the nodes of the input graph are labeled by non-confusable “types” such as “person,” “place,” “event,” and so on. The types are distinct from general labels that may be incorrect. Thus, while a person is not confused with an event, the names (labels) “John” and “Josef” can be confused. In real application, 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  $1, 2, 3, \dots, t$ , where  $t$  is the total number of types used. Our graphs are directed; all notations used but not defined here can be found in [5].

*i-deg*: given node  $x$ , the  $i\text{-deg}(x)$  is a vector of length  $t$ , where the  $k^{\text{th}}$  coordinate is the number of paths of length  $i$  that start at  $x$  and end at nodes of type  $k$ . By extension,  $0\text{-deg}(x)$  is the vector representing the type of  $x$ .

*i-dist*: this function computes the “similarity” of two nodes of the same type by comparing their  $i$ -degrees. This is calculated by the Manhattan distance between the vectors:

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

*Dominating i-Distance*: this, like  $i$ -distance, is also a measure of “similarity,” but requires that the first node’s  $i$ -deg dominates the second, that is at every  $k^{\text{th}}$  position the first is not less than the second; else, the dominating  $i$ -distance 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  $\text{similarity}(v, w) < r$  for some  $r \geq 0$ .

*Indexing*: indexing is the process of assigning values to nodes as a preprocessing step. Each node is assigned  $i$ -degrees for  $i = [0, k]$ ; 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  $|N(x)| > |N(y)|$ .

The problem we are solving is that of detecting the location of the fragment anchor in the database. *Input*: semantic graphs  $G$  and  $F$ ; and a node  $z \in V(F)$ ; *Output*: a node  $c \in V(G)$ , which is “similar” to  $z$ .

## III. SEARCH ALGORITHM

This section details the algorithm used for locating anchors 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  $F$ . Two possible methods of determining  $x$  include: choosing the most diverse node in  $F$ , or the node with the largest neighborhood.

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.

Searching for a single node using similarity of  $i\text{-deg}$  is the basic approach we use. A more advanced method makes use of some number of neighbors of the anchor,  $m$ . These nodes are the  $m$  most diverse neighbors of  $x$ .

---

### Algorithm 1 Fragment Search

---

- Require:**
1. fragment graph  $F$
  2. anchor node  $x \in F$
  3. database graph  $G$
  4.  $i$  for the maximum  $i$ -degree to compare
  5.  $n$  for the number of candidates to return
  6.  $m$  for the number of neighbors to use in the search
  7. a similarity function  $\text{Similarity}(\text{node}, \text{node})$

Let  $D$  be the sorted list of the  $m$  most diverse nodes in  $N(x)$

**for**  $v$  s.t.  $v \in G$  AND  $\text{type}(v) = \text{type}(x)$  **do**

$d \leftarrow \text{Similarity}(v, x)$

**for**  $j : 1 \rightarrow m$  **do**

$d_{\text{temp}} \leftarrow \min_{w \in N(v)} (\text{Similarity}(w, D[j]))$

$d \leftarrow d + d_{\text{temp}}$

$\text{candidates} \leftarrow \text{candidates} \cup \text{pair}(d, v)$

**return** the best  $n$  candidates

---

## IV. GRAPH GENERATION

### Wikipedia Graph

The main graph we used for testing was constructed from Wikipedia. Each page is a node and page-links 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. The resultant graph has the following properties.

TABLE I  
WIKIPEDIA GRAPH PROPERTIES

Nodes	Types	Max Outdegree	Edges
1188437	26	649	3614485

### Random Graph

The random graphs are the second type of graphs used for experimentation; we generate them using statistics collected from the Wikipedia graph. To do this we redistributed all the incoming edges uniformly amongst the nodes, and then reassigned the types randomly between nodes. Since the edges are uniformly distributed, this alters their  $i$ -degrees, which in particular, develops more unique  $i$ -degs. This graph has the same number of edges, but slightly fewer nodes, 1187004, because some nodes were isolated during the random edge distribution, and these nodes are removed by post-processing.

### V. PARTITIONING

The search for an anchor (or the fragment) is influenced by the number of nodes with the same  $i$ -deg. If this number is too large, the search may require an  $n$  which is larger than is reasonable for an analyst to use. Thus, it is important that the partitioning of the database by  $i$ -degrees yields small, on average, partitions. The partitioning data for the Wiki graph is presented in TABLE II.

TABLE II  
WIKIPEDIA GRAPH PARTITIONING STATISTICS

	#Parts	Max size	Avg. size	Std. Dev
0-deg	26	368775	457009	96913
1-deg	18077	103073	65.7	1418
2-deg	327609	17672	3.6	63.8
3-deg	469097	13421	2.5	36.4

This tables shows us that partitioning by 1-deg provides on average 65.7 nodes with the same 0-deg and 1-deg. A partitioning by 3-deg shows an average of 2.5 nodes with the same 0-,1-,2-, and 3-deg. This however only solves part of the problem. Because we are attempting to locate fragments which may be missing data or contain new information, we must consider that nodes can have different  $i$ -degs. That is, even if  $y \in G$  is the ideal match to  $x \in F$ ,  $x$  may not have identical  $i$ -degrees to  $y$ .

Consider  $r$ -fuzziness, as we increase maximum difference,  $r$ , these groups grow very quickly. This indicates that searching for a node by  $i$ -deg alone will not guarantee good precision.

TABLE III  
3-DISTANCE FUZZINESS FOR 100 RANDOMLY SAMPLED NODES

r	0	1	2	3	4
Avg.	1.6	4.6	16	40	62
Max	54	211	1717	5354	6968

### VI. 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 algorithms below describe different methods of creating fragments. 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 'e' (exact 1-Neighborhood):*

---

**Require:** 1. large graph  $G$ ;

2. a maximum fragment size  $n$

$F \leftarrow (V \leftarrow \emptyset, E \leftarrow \emptyset)$

Randomly select  $x \in G$  such that  $|N(x)| < n$

$V(F) \leftarrow \{x\} \cup N(x)$

**if**  $|2 - N(x)| + |V(F)| > n$  **then**

    choose  $n - |V(F)|$  nodes at random from  $2-N(x)$

**else**

    add all  $2-N(x)$  to  $F$ .

---

The first fragment generation method, 'e', guarantees that the 1-deg( $x$ ) will remain the same. The second method, 'n', is a variation on the first in that it does not guarantee complete information about the structure of  $N(x)$ . The final generation method uses a random walk with restarts. In our testing we used  $p = 0.9$  and  $\alpha = 0.7$ .

### VII. VALIDATION AND RESULTS

To validate the accuracy of the algorithm we test it with several different input parameters: fragment generation methods; search parameters; levels of  $i$ -dist indexing, etc. In table IV, fragment size is listed across the top. Bold text represents searches using the dominating  $i$ -dist, instead of the  $i$ -dist. Each cell represents a query for 100 anchor nodes,  $x$ , with  $m$  neighbors for  $m=0..3$ . The fragment samples are generated around the same 100 anchor nodes across all tests and were constructed from the  $P$ -random walk algorithm unless otherwise specified.

For each of these tests, increasing  $m$  increases the precision. The greatest benefit for this database is seen when increasing from 0 neighbors to 1 neighbor. Furthermore, if we look at the difference between the  $i$ -dist and dominating  $i$ -dist, we see a great improvement

---

**Algorithm 3** *Fragment Generation 'n'*

---

**Require:** 1. a graph  $G$   
2. a maximum size  $n$

Randomly select  $x \in G$  such that  $|N(x)| > 3$   
Add  $x$  to  $V(F)$   
 $m \in [3, \min(|\text{neighbors}(x)|, n - 1)]$   
Choose at random  $m$  nodes from  $N(x)$  and add these nodes to  $F$   
**if**  $|V(F)| < n$  **then**  
     $V \leftarrow \{v | v \in G \text{ AND } v \notin F \text{ AND } \exists(u, v) \text{ where } u \in F\}$   
    **if**  $|V| + |V(F)| \leq n$  **then**  
        add all  $v \in V$  to  $F$   
    **else**  
        select at random  $n - |V(F)|$  unique nodes from  $V$  and add each to  $V(F)$

---

---

**Algorithm 4** *Fragment Generation P-Random Walk 'p'*

---

**Require:** 1. graph  $G$   
2. maximum size  $n$   
3. probability  $p$   
4. constant  $\alpha$

Choose randomly  $x \in G$  such that  $|N(x)| > 2$   
Add  $x$  to  $V(F)$   
 $u \leftarrow x$   
 $previousSize \leftarrow 1$   
**while**  $|V(F)| < n \wedge p \geq p * (\alpha^n)$  **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**  
             $p \leftarrow p * \alpha$   
             $previousSize = |V(F)|$

---

TABLE IV  
PRECISION AFTER 60 TRIES. RANDOM GRAPH, SEARCH: 2-DIST

m	20	20	30	30	40	40
0	<b>0.93</b>	0.37	<b>0.98</b>	0.78	<b>1.00</b>	0.84
1	<b>0.97</b>	0.51	<b>0.99</b>	0.86	<b>1.00</b>	0.89
2	<b>0.97</b>	0.60	<b>1.00</b>	0.87	<b>1.00</b>	0.90
3	<b>0.89</b>	0.67	<b>1.00</b>	0.91	<b>1.00</b>	0.96

in precision. This is to be expected because more information is guaranteed. Also we notice that the larger the fragment, the higher the precision. This is to be expected, because the information of the fragment will more closely resemble the graph.

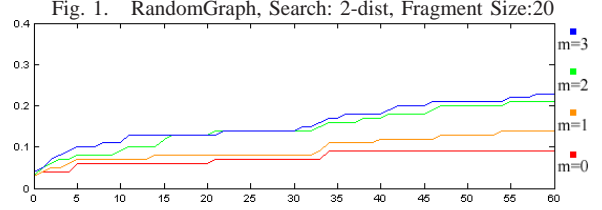
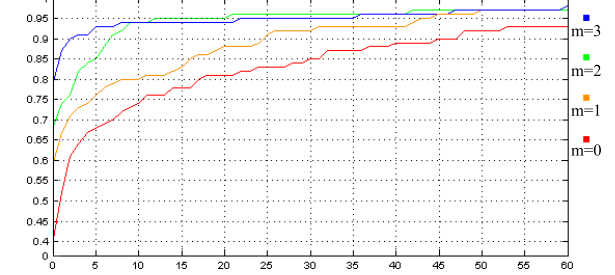


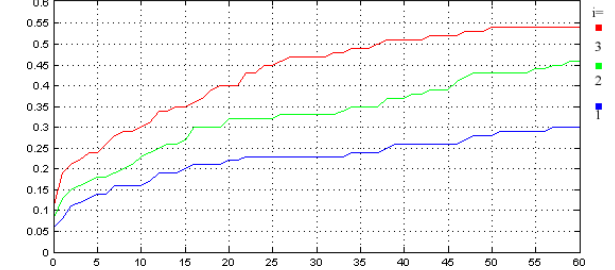
Figure 1 shows the precision of our algorithm over 60 guesses. This demonstrates that as we search for more neighbors we get better results, however we do not need all 60 guesses to obtain this performance. Many of the successful searches occur within the first few guesses. On the other hand, in some databases it may be desirable to have more than 60 results returned.

Fig. 2. RandomGraph, Search: Dominating 2-dist, Fragment Size:20



In Figure 2, even though after 60 guesses the dominating searches all reach over 90% success, many of the correct guesses were within the first 5 guesses. This reveals how much more reliable a search is with dominating  $i$ -dist. Though we can't say this trend will be true of all graphs, in our experiments the precision curves followed this general trend.

Fig. 3. RandomGraph, Search: Dominating  $i$ -dist, Fragment Size:20



When using the dominating  $i$ -dist (Figure 3), the precision increases as  $i$  increases. When using the simple  $i$ -dist, however, the precision increases until  $i = 3$  at which

point the likelihood of success decreases significantly. This is because we weight all penalties equally. However, the potential for error of a fragment node's 3-deg is higher than that of its 2- or 1-deg. When not using dominating  $i$ -dist, this over penalizes the incompleteness of the fragment.

TABLE V  
PRECISION AFTER 60 TRIES. WIKI GRAPH, SEARCH: 2-DIST

m	20	20	30	30	40	40
0	<b>0.31</b>	0.09	<b>0.39</b>	0.15	<b>0.45</b>	0.20
1	<b>0.41</b>	0.14	<b>0.50</b>	0.23	<b>0.54</b>	0.26
2	<b>0.46</b>	0.21	<b>0.49</b>	0.28	<b>0.56</b>	0.28
3	<b>0.52</b>	0.23	<b>0.62</b>	0.35	<b>0.64</b>	0.33

More nodes in the fragment yield better results, and searching for more neighbors also yields better results. In TABLE V, we see that for these parameters, the results are lower than for the random graph, due to the more difficult nature of the Wiki-graph's  $i$ -degree distribution. This is an obstacle which we hope to overcome.

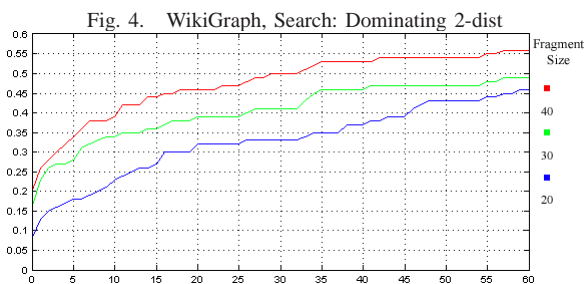
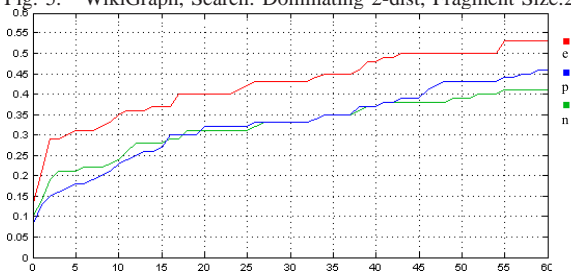


Fig. 5. WikiGraph, Search: Dominating 2-dist, Fragment Size:20



In figure 5, we show the performance of our algorithm for each of our fragment generation methods. We can see that the methods all produce similar patterns of precision over 60 guesses. Method ‘e’, however, has higher precision because it contains the full 1-deg of the anchor node. This information is slightly more distinguishing than the random-walk information.

Finally, in TABLE VI is the time requirement to run 100 tests for various fragment sizes, and  $m$  values. The

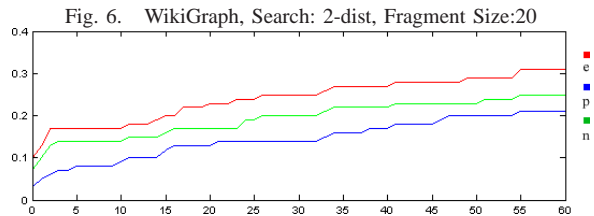


TABLE VI  
SECONDS FOR 100 QUERIES. WIKI GRAPH, SEARCH: 2-DIST

m	20	30	40
0	133	134	135
1	294	301	324
2	397	388	385
3	428	430	432

increase in run time when increasing the fragment size is negligible, because only a few nodes from the fragment are used. However, increasing  $m$  greatly increases the runtime.

## VIII. FUTURE WORK

Our experiments indicate that the probability of the search to be successful increases with the size of the fragment, which should be expected since a bigger size decreases the possibility of confusing the search. We notice that selecting “special” neighbors of the anchors has a positive affect on the search. This feature is a partial implementation of the total alignment of the fragment in the database graph. Although incorporating the total alignment should improve the accuracy, the difficulty of this improvement relates to the computational complexity of this operation. The only practical algorithm available is based on the backtracking strategy, and is time consuming, as a result. Thus, to make the operation fast, the backtracking needs to be restricted, which may negatively affect the accuracy of the resulting procedure. Another important observation is related to the role of guesses by the user. We expect that any software system for solving the fragment identification problem to be very flexible when accommodating the users’ suggestions.

**Acknowledgment.** This research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053 and y the Department of Homeland Security through the Command, Control, and Interoperability Center for Advanced Data Analysis Center of Excellence. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to re-

produce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

#### 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] M. Goldberg, S. Kelley, M. Magdon-Ismail, and W. Wallace. Overlapping communities in social networks. *Proc. 2nd Conference on Social Computation (SocialCom)*, pages 104–113, 2010.
- [4] 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.
- [5] D. B. West. Introduction to graph theory. *Prentice Hall, Upper Saddle River, NJ*, 2003.
- [6] 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.