MAGE: EXPRESSIVE PATTERN MATCHING IN RICHLY-ATTRIBUTED GRAPHS

ROBERT PIENTA
pientars@gatech.edu
College of Computing
Georgia Institute of Technology

ACAR TAMERSOY
tamersoy@gatech.edu
College of Computing
Georgia Institute of Technology

HANGHANG TONG
tong@cs.ccny.cuny.edu
Department of Computer Science
The City University of New York

DUEN HORNG (POLO) CHAU
polo@gatech.edu
College of Computing
Georgia Institute of Technology

Technical Report Number: GT-CSE-2013-08

October 2013

ABSTRACT

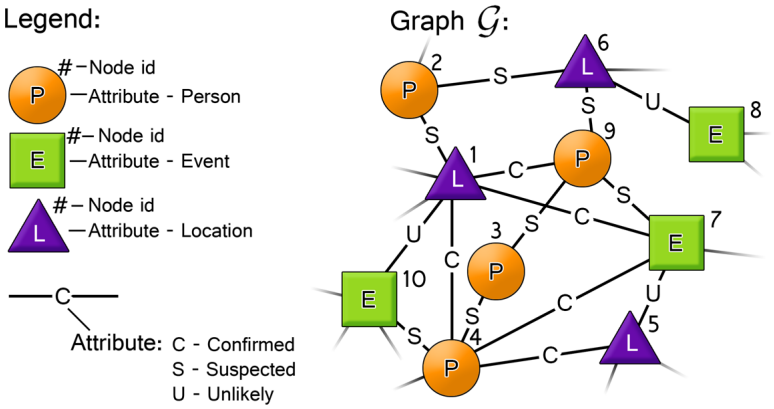
Given a large graph with millions of nodes and edges, say a social graph where both the nodes and edges can have multiple different kinds of attributes (e.g., job titles, tie strengths), how do we quickly find matches for subgraphs of interest (e.g., a ring of businessmen with strong ties)? We propose MAGE, *Multiple Attribute Graph Engine*, a subgraph matching framework that pushes the envelope of graph matching capabilities and performance, through several major innovations: (i) with *line graph* transformation, MAGE works for graphs with both node and edge attributes and return both exact as well as near matches — other techniques often support only node attributes and return only exact matches; (ii) MAGE supports a plethora of queries, including multiple attributes for each node or edge, wildcards as attribute values (i.e., match *any* permissible value), and continuous attributes via multiple discretization strategies; (iii) MAGE leverages a novel technique based on *memory mapping* to compute random walk with restart probabilities, which provides a speedup of more than 2 orders of magnitude on large graphs. We evaluated MAGE's effectiveness and scalability with real and synthetic graphs with up to 2.3 million edges. Experimental results on the DBLP authorship graph and the Rotten Tomatoes movie graph illustrate the effectiveness and exploratory functionality of our contributions to graph querying. By devising query-centric innovations, our work improves the ease with which a user can explore their graph data.

1 INTRODUCTION

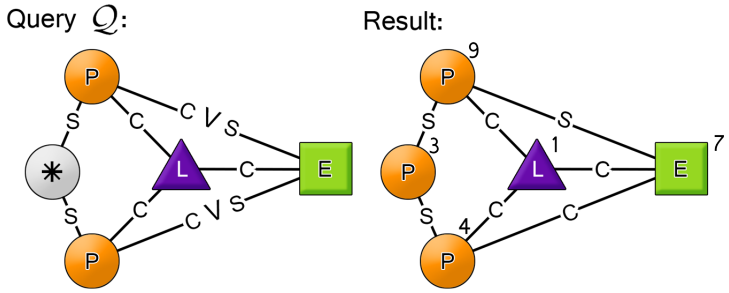
Graphs are a convenient means to represent many naturally occurring patterns. As entities are often related to one another in a pairwise fashion, many systems can be faithfully represented with graphs. Since the development and broadening of large scale information systems, collecting the requisite information for the purpose of building graphs has become a common task. For example, the words *social network* have become a household phrase in the past decade; while the social networks themselves are growing at unparalleled rates.

Many graphs are richly embedded with information, but because of the size and complexity of these graphs, spotting a particular pattern outright quickly becomes infeasible. People often want to find patterns in graphs to better understand their dynamics and to detect who or what is related to the possible anomalies within. For instance, given an intelligence graph containing various entities, which are connected with edges denoting gathered intelligence, an analyst might seek to better understand the structure of criminal activity. Given some initial structure of a terrorist cell or other organized criminal group, one could leverage a *pattern matching* system to discover potentially dangerous individuals. This is a promising notion; however, there are many challenges associated with the technical aspects of this problem.

First, we need a convenient approach that allows users to specify a rich set of possible query patterns, with appropriate values assigned to the nodes and edges. Consider the fictitious intelligence graph in Figure 1a. Here, the node attribute is the entity type, with the possible values of *Event*, *Location*, and *Person*, and the edge attribute is the amount of gathered intelligence for a pair of entities, with the possible values of *Confirmed*, *Suspected*, and *Unlikely*. For this graph, the user of the system should be able to specify a query similar to Figure 1b, which looks for two individuals — potentially unrelated to each other with current intelligence — who were both confirmed at the location of some event and are also believed to have attended the event. In the query, the two individuals are denoted by the *P*-nodes (nodes with letter *P* inside), and they are linked to the corresponding location, which is represented by the *L*-node, with confirmed intelligence denoted by the *C*-edges (similarly, the location is connected to the event with confirmed intelligence). In general, the challenge is that the user might not know what values to assign to some nodes and edges in the query; so instead of guessing the user should be allowed to leave any node or edge as a *wildcard*, meaning that it can take any value



(a) An intelligence graph linking locations, events and people. Each edge denotes a measure of confidence in the connecting relationship.



(b) A sample query that can be formed in MAGE and a potential result. The query looks for two potentially related individuals, who were both confirmed at the location of some event and are also believed to have attended the event. The node labeled with a star indicates a wildcard, which can take any attribute value.

Figure 1: Using MAGE to seek patterns in an intelligence graph. All the figures are best viewed in color.

from the set of possible values, as long as the structure of the query is satisfied. For instance, in our running example the two individuals are connected via a wildcard (*-node) in order to find out possible entities that might relate the individuals to each other. In some other cases, the user might know a couple of admissible values for a particular node or edge. Rather than forming several distinct queries for each of these values and then combining the results, the user should be able to specify in a single query those values she knows to simplify the process. Returning to the example, the condition that the individuals must be either confirmed *or* suspected of being involved in the event is denoted by specifying both values with a disjunction (i.e., $C \vee S$) on the edges linking the *P*-nodes to the *E*-node.

Once a query is formed, we need to find exact matches as well as near matches, because the query might have wildcards or not exist exactly as specified within the graph. As the query in Figure 1b contains a wildcard linking two individuals, the system should be able to return a match filling in the wildcard, e.g., the location entity for node 1 in the result. Even with the wildcard, the exact specified structure may not exist in in the graph; under this scenario the system would return a "best-effort" match of the query containing additional nodes and edges. By generating both exact and near matches, we can provide the user the top-k most closely matched subgraphs even if their initial query was not exactly present.

In this paper we present MAGE; a pattern matching system for graphs with node and edge attributes, which overcomes the challenges stated above. MAGE produces top-k closest subgraph matches and broadens the number of possible input graphs as well as possible queries. MAGE is a system which can be used in an enormous number of domains; from intelligence applications to understanding the patterns of movie success. Our approach is also scalable to many large graphs, offering new insight through graph querying.

1.1 INNOVATIONS AND CONTRIBUTIONS

We make several major contributions to graph querying in this work.

Support for node and edge attributes. The first is that we have created a pattern matching system for graphs that supports queries with categorical node and edge attributes. Using both node and edge attributes expands the effectiveness of our system on real world data and expands the types of questions that can be answered through graph querying. Although many graphs contain numerical or continuous attributes, several methods are offered to discretize them for use in MAGE.

Flexible queries with rich attributes. The second contribution is focused on improving the ease of querying. Sometimes query-information is limited and the exact attribute of a node or edge may be unknown.

The wildcards allow the user to avoid having to specify a value for an unfamiliar attribute. Many graphs have several dimensions of data associated with both the nodes and edges. Multiple attributes on nodes and edges of the query and graph help users query richer datasets. We propose a method to allow the specification of queries across multiple categorical attributes simultaneously.

Fast & scalable algorithm. Our third contribution is a novel technique based on memory mapping to improve the performance and scalability of the random walk with restart (RWR) algorithm. MAGE leverages approximate RWR steady-state probabilities as proximity scores between nodes within the input graph and the query graph to determine how well a subgraph matches the query. The RWR probabilities are needed at multiple times in MAGE, therefore we propose a fast and highly scalable single-machine approach for RWR calculations. The experimental results show that our solution provides significant improvements over traditional approaches to compute RWR probabilities on a single machine.

2 PROBLEM DEFINITION

In its general form, we are given two graphs $\mathcal{G}$[1] and $\mathcal{Q}$, and we wish to know if $\mathcal{G}$ contains a subgraph that is equivalent to $\mathcal{Q}$. This problem is often referred to as the subgraph isomorphism problem. Unfortunately the subgraph isomorphism problem is NP-Complete [6], making all general solutions computationally infeasible for even modest sized graphs.

The problem we are solving is subtly different than the pure subgraph isomorphism problem. The problem we attempt to solve is formally defined as follows:

GIVEN: (i) A graph $\mathcal{G}$ whose nodes and edges have categorical attributes, (ii) a query graph $\mathcal{Q}$ showing the desirable configuration of nodes connected with edges, each assigned one or more attribute values (or a wildcard), and (iii) the number of desired matching subgraphs k.

FIND: k matching subgraphs $\mathcal{Q}_i$ ($i = 1, ..., k$) that match query graph $\mathcal{Q}$ as closely as possible, according to a goodness metric.

2.1 PRELIMINARY: QUERYING ON NODE ATTRIBUTES

Several approaches have been proposed to subgraph isomorphism problem. In this paper we utilize the techniques of inexact subgraph matching approaches to form the foundation of MAGE.

Algorithm 1 G-Ray Algorithm [20]

Require: Node-attributed graph $\mathcal{G}$, query graph $\mathcal{Q}$, and desired number of results k
Ensure: k node-attribute matched subgraphs from $\mathcal{G}$
  for i=1→k **do**
    $\mathcal{Q}_i$ = approximate-subgraph($\mathcal{G}, \mathcal{Q}, k$)
  end for
  return $\mathcal{Q}_i$ where $i = 1 \rightarrow k$

The G-Ray algorithm [20], a best-effort subset selection algorithm, relies heavily on RWR values as the selection criterion when constructing query results[2]. This approach uses single nodes as the restarts when calculating the RWR values. We leverage this approach but considerably modify it to allow multiple attributes as restarts in the RWR

1 Table 1 gives all the symbols used in the paper.
2 We refer the reader to [21] for the details of the RWR algorithm.

Symbol	Description
$\mathcal{G}$	the $n \times n$ adjacency matrix for $\mathcal{G}$
$\mathcal{A}$	the $n \times t$ node-attribute matrix for graph $\mathcal{G}$
$\mathcal{Q}$	query subgraph to be extracted from $\mathcal{G}$
$\mathcal{G}'$	the $(m+n) \times (m+n)$ linegraph-modified bipartite graph
$\mathcal{A}'$	node-edge-attribute matrix for graph $\mathcal{G}'$
$\mathcal{Q}'$	query subgraph after edge augmentation
M	a bijective mapping between edges of $\mathcal{G}$ and edge-nodes in $\mathcal{G}'$
$\langle s, t \rangle$	an edge leading from node s to node t
n	the number of nodes in $\mathcal{G}$
m	the number of edges in $\mathcal{G}$
t	the number of distinct categorical attributes

Table 1: Terminology and notation

approximations (see Section 4.2 for details). Approximate RWR is still a computationally expensive step that must be performed often when matching subgraphs. In Section 4.4 we show our approach to reducing query latency by decreasing RWR calculation times.

While G-Ray is an integral facet of MAGE, the limitations of the original algorithm are far too constrictive. The G-Ray algorithm is inadequate in supporting expressive querying as it does not support (i) attributed edges, (ii) unknown query attributes, (iii) multiple attributes, and incurs (iv) sizable query latencies on large graphs. These points are exactly what we aim to address in this paper and in the development of MAGE. Using our linegraph augmentation we support edge attributes, and with wildcards and multiple attributes we support queries with limited information. In addition, we aim to speed up the subgraph search computation, which is doubly critical in the case of queries containing wildcards, multiple attributes, and edge attributes.

3

MAGE OVERVIEW

Unlike many previous systems, we focus on both the edge and node attributes in $\mathcal{G}$. We have chosen to use an edge-augmentation method based on the intuition from the linegraph transformation [24]. Under the canonical linegraph transformation, each vertex in the line graph $\mathcal{L}$ of $\mathcal{G}$ is an edge from $\mathcal{G}$. Two vertices in $\mathcal{L}$ are connected if and only if their corresponding edges (from $\mathcal{G}$) share a common endpoint in $\mathcal{G}$. Figure 2a demonstrates an example transformation with the key linegraph transformation occurring in the rightmost two figures.

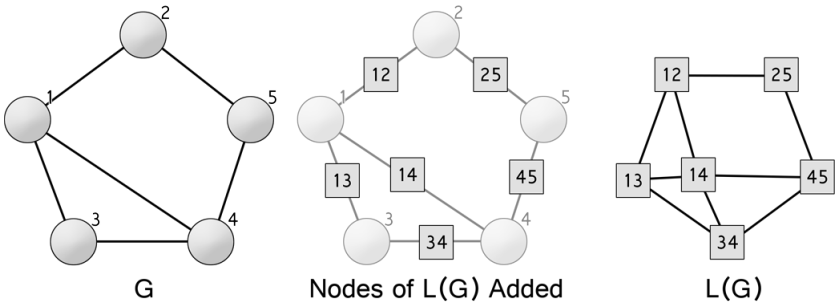
By making use of the line-graph transformation on the starting graph $\mathcal{G}$, we can produce an attributed line graph $\mathcal{L}$ where each of the edges in $\mathcal{G}$ is represented by a node in $\mathcal{L}$. Rather than working with both $\mathcal{L}$ and $\mathcal{G}$, we create $\mathcal{G}'$ which combines aspects of both $\mathcal{G}$ and $\mathcal{L}$. We achieve this by transforming each edge in $\mathcal{G}$ into an edge-node in $\mathcal{G}'$. This new edge-node is connected to the same vertices in $\mathcal{G}'$ that it connected to as an edge of $\mathcal{G}$. This process is illustrated with a toy graph in Figure 2b, where the edge-nodes are the square nodes splitting each edge of $\mathcal{G}$. Under this formulation, no two nodes in $\mathcal{G}$ will be directly connected in $\mathcal{G}'$. Similarly, no two newly introduced edge-nodes will be directly connected in $\mathcal{G}'$. The structure of the newly created graph is bipartite between the set of original nodes and the set of new edge-nodes. We make use of this fact in the development of MAGE.

3.1 SUPPORTING MULTIPLE ATTRIBUTES

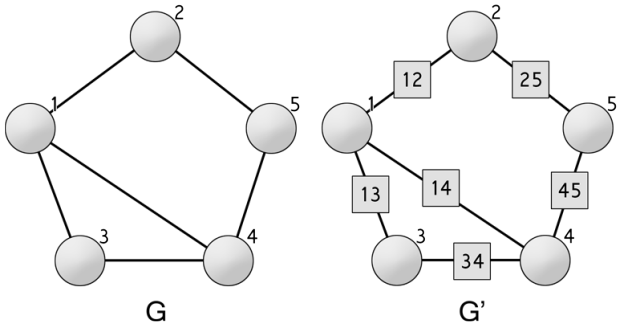
Often in real graph datasets, nodes and edges can have multiple fields of data. Similarly, a user may want to query for one or more attributes at the same time on a node or edge. These are important considerations and they are fully supported in MAGE by allowing lists of variables on each query node or edge. Each list of attributes are combined to produce query results with a logical-or for each item in the list during query time. This allows for the quick generation of queries across multiple node or edge attributes. Multi-attribute queries greatly extend the user's query possibilities and, therefore, exploratory power.

3.2 SUPPORTING WILDCARDS

In order to ease the construction of query-graphs as well as extend usability of MAGE, we developed wildcard attributes. The wildcard allows MAGE to match any attributed node or edge to a wildcard



(a) The canonical linegraph transformation of $\mathcal{G}$. The middle figure is the intermediate step of the transformation, where a node for each original edge is created. $\mathcal{L}(\mathcal{G})$ is the linegraph of $\mathcal{G}$ in which all edges from $\mathcal{G}$ that shared a node in $\mathcal{G}$ are now connected as the nodes of $\mathcal{L}(\mathcal{G})$.



(b) Edge augmentation of graph $\mathcal{G}$. Rather than performing the linegraph transformation and having two separated attributed graphs, we have embedded the nodes of $\mathcal{L}(\mathcal{G})$ directly into $\mathcal{G}$. The result is $\mathcal{G}'$.

Figure 2: The linegraph transformation and the edge augmentation approach used to support edge and node attributes in MAGE.

node or edge, respectively, while maintaining the overall query connectivity. This is carried out by relaxing the node and edge attribute constraints used during the acquisition of nodes during query-result construction. Our experimental investigations suggest that our approach to wildcards is efficient and does not incur significant query overhead.

4 METHODOLOGY

The general approach taken for MAGE is explained in Procedure 2 and the subsequent procedures are described in the following sections.

4.1 SUPPORTING EDGE ATTRIBUTES

In order to support edge attributes, a method is needed that allows the incorporation and later selection of data on each edge. For this we have chosen to embed an edge-node in each edge of $\mathcal{G}$. The pseudo-code for this edge-embedding is found in Procedure 3.

Algorithm 2 MAGE Algorithm

Require: Fully-attributed graph $\mathcal{G}$, attribute graph $\mathcal{A}$, query graph $\mathcal{Q}$, and desired number of results k

Ensure: k node-attribute matched subgraphs from $\mathcal{G}$

1: $\mathcal{G}' =$ **Linegraph-Embedder**$(\mathcal{G})$
2: $\mathcal{Q}' =$ **Linegraph-Embedder**$(\mathcal{Q})$
3: $\mathcal{A}_w =$ **Wildcard-Attribute-Injector**$(\mathcal{A})$
4: **for** i=1$\rightarrow$k **do**
5: $\quad \mathcal{Q}'_i =$ G-ray$(\mathcal{G}', \mathcal{A}_w, \mathcal{Q}', k)$
6: $\quad \mathcal{Q}_i =$ **Linegraph-Extractor**$(\mathcal{Q}'_i)$
7: **end for**
8: **return** $\mathcal{Q}_i$ where $i = 1 \rightarrow k$

Algorithm 3 Linegraph-Embedder

Require: Edge-attributed $n \times n$ graph $\mathcal{G}$
Ensure: Edge-embedded $(m+n) \times (m+n)$ graph $\mathcal{G}'$ and a mapping M from edges to newly created edge-nodes

1: Let $\mathcal{S}$ be an all-zero $(n \times m)$ matrix
2: **for all** $u = 1 \to m$ edges, $e_{i,j} \in \mathcal{G}$ **do**
3: $\quad S(u, i) = 1$
4: $\quad S(u, j) = 1$
5: $\quad M(u) = e_{i,j}$
6: **end for**
7: $\mathcal{G}' = \begin{bmatrix} \mathbf{0} & \mathcal{S} \\ \mathcal{S}^T & \mathbf{0} \end{bmatrix}$
8: **return** $\mathcal{G}'$

4.1.1 *Linegraph Augmentation*

The following shows the algorithm for constructing the line graph transformation of the original graph. This algorithm is $O(m)$ where m is the number of edges from $\mathcal{G}$. This needs only to be done once and is therefore precomputed a single time before querying.

This transformation creates $\mathcal{G}'$, a $(m+n) \times (m+n)$ adjacency matrix. Expanding both dimensions of our adjacency matrix by a factor of m may seem shockingly expensive in memory usage; however, $\mathcal{G}'$ is guaranteed to be bipartite between the original nodes and the new edge-nodes. Because only original nodes can be connected to edge-nodes, we have only the $m \times n$ and $n \times m$ regions of our augmented matrix that can possibly contain values. We can derive the sparsity as follows:

$$\frac{2mn}{m^2 + 2mn + n^2}$$

if the graph is undirected only mn edges need to be stored. Because we use sparse data structures, the memory for this augmentation grows at a linear rate with the number of edges.

The matched subgraph results produced by MAGE are still embedded with edge-nodes and should therefore be converted back to the original graph format. The linegraph-extractor (see Procedure 4) serves this purpose; to return our modified results to the style and format specified with the input graph.

4.2 SUPPORTING MULTIPLE ATTRIBUTES

The categorical attribute matrix $\mathcal{A}$ is an $n \times t$ sparse matrix where there are t distinct attribute categories for each of n nodes. Each row of $\mathcal{A}$ represents a node while each column represents a single categor-

Algorithm 4 Linegraph-Extractor

Require: Edge augmented query result Q'_i
Ensure: Attribute matrix Q_i
1: **for all** node-edge $u_j \in Q'_i$ where $j = 1 \rightarrow q$ **do**
2: s = source edge leading into u_j
3: t = target edge leading out of u_j
4: remove edge $\langle s, u_j \rangle$
5: remove edge $\langle u_j, t \rangle$
6: replace edge-node u_j with $M(u_j)$
7: **end for**
8: **return** Q_i

ical variable. Each node's categorical data is encoded as a — usually sparse — vector of ones.

While t can be very large, generally the mapping of categorical attributes to nodes is very sparse. Practically, $\mathcal{A}$ utilizes a minor amount of memory. We support multiple attributes on each edge and node, and allow them to be selected via logical OR. This is done by allowing the rows of $\mathcal{A}$ to have multiple values at once.

These rows are leveraged during the attribute-centric random walk with restart carried out in line 5 of Algorithm 2. By serving as restart sources during the RWR calculations, the correctly attributed nodes are given larger proximity scores and therefore are more likely to be selected as a result.

4.3 SUPPORTING WILDCARDS

To support the wildcard attribute we have created a universal attribute applied to all nodes and edges. This attribute is one among many that each edge or node may have at any time.

Algorithm 5 Wildcard-Attribute-Injector

Require: Attribute matrix $\mathcal{A}$
Ensure: Attribute matrix $\mathcal{A}_w$
1: **for** $i = 1 \rightarrow n$ **do**
2: $a_{i,t+1} = 1$
3: **end for**
4: **return** $\mathcal{A}_w$

We have implemented the wildcard attribute by creating a new and distinct attribute node in the attribute matrix A that points to all nodes. This technique works because the MAGE algorithm will select this wildcard attribute regardless of whatever other attributes a node or edge may have. This can be achieved by appending a column of ones to the attribute matrix as an initialization step.

4.4 QUERY LATENCY

MAGE uses proximity scores between nodes in data graph $\mathcal{G}$ and query graph $\mathcal{Q}$ to determine how well a subgraph $\mathcal{Q}_i$ matches $\mathcal{Q}$. Specifically, the proximity between nodes i and j in a graph is defined as the score of j when it performs a RWR on the graph with i being the restarting node. The issue is that the number of RWR queries required to find a matching subgraph is significant, making this phase a bottleneck of the whole approach for large data graphs. We extend a recent work on single-machine graph computation frameworks [13], which uses the memory mapping capability of the operating systems to perform fast graph computations. Memory mapping is a mechanism that allows to map a file on disk into the main memory such that the file can be accessed the same way as if it was in memory. This makes it possible to perform I/O operations faster than accessing disk directly due to the low level optimizations provided by the hardware to the operating systems. We show in Section 5.2.3 that our memory mapping based RWR implementation reduces the latency for RWR queries significantly.

5 EVALUATION

Due to the tremendous size of modern datasets we are interested primarily in the evaluation of the scalability of MAGE: how does the time-per-query increase with larger graphs? We analyze various sizes of real and synthetic graphs to answer this question. We also investigate the cost of our query extensions; wildcard and multi-attributes. For synthetic graphs we use stochastically generated Erdős-Rényi random graphs and Watts-Strogatz graphs. For real graphs we use an actor-movie graph from Rotten Tomatoes and a citation graph from DBLP.

5.1 DATASETS

MAGE operates on categorical attributes, so fields must first be encoded categorically to be implemented in our system. For continuous fields, discretization offers several methods to aggregate multiple continuous values into categories. With quantiles, histogramming, and domain specific methods for value aggregation, most continuous values can be quickly converted into categories.

5.1.1 *DBLP*

We have chosen to use the DBLP authorship dataset. With over 2 million edges, we have elected to use DBLP as our real-world scalability test. We have extracted categorical edges attributes based on the number of co-citations among the set of authors.

5.1.2 *Rotten Tomatoes*

We have also tested MAGE on a modest-sized directed graph constructed out of the Rotten Tomatoes movies and actors. The graph contains more than 20,000 movies with edges connecting similar movies and actors that performed therein. Actors are linked bidirectionally to movies in which they took part and movies have edges to movies that Rotten Tomatoes has deemed similar. Quartiles were used to aggregate continuous fields into categories over fields like critics' film scores, runtimes, release dates, and others. The edges connecting similar movies were originally weighted by the number of user-contributed "up-votes", these values were also broken into categories by quartiles.

5.1.3 *Parameters*

The random walks with restart requires two parameters; the fly-out or restart-probability and the number of iterations for the iterative method. In all experiments the restart-probability is set to 0.15 and the iterations are set to 10. After 10 iterations the performance did not significantly improve under the iterative method.

5.2 SCALABILITY AND SPEED

The query-scalability tests were carried with a 4-node 3-edge linear query graph on two types stochastically generated data-graphs of varying size. Erdős-Rényi random graphs and Watts-Strogatz graphs were stochastically generated for vary sizes n. Their node and edge attributes were selected uniformly at random with 3 distinct categorical attributes each. The test query was run on these graphs for $k = 5$ results.

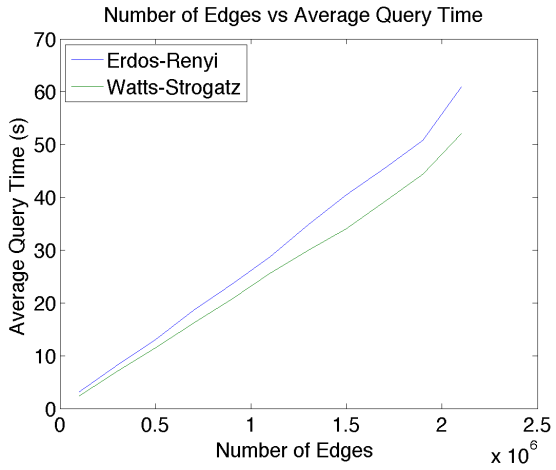


Figure 3: MAGE query performance on synthetic graphs of varying sizes. The average query time increases linearly with the size of the graph.

MAGE performs queries in linear time with the number of edges in each graph. For both the canonical ER graphs and the small-world WS graphs the increase in query time is linear in the number of edges, suggesting good scalability.

Synthetic networks can model many of the structural properties of real networks; however, synthetic scalability tests are insufficient. To address this we performed scalability experiments on the DBLP dataset. We generated randomly attributed 4-node 3 edge linear queries and averaged the result over multiple runs. Each node has more than 3000 possible categorical attributes as both conference and years of submission were encoded. Each edge takes one of 5 possible cate-

gories of co-citation strength. These categories are drawn from the quartiles over the collection of co-cititing authors in DBLP.

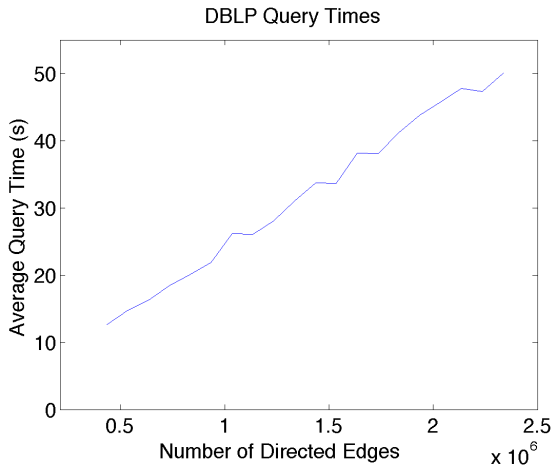


Figure 4: MAGE query performance averaged over subsets of the DBLP graph. MAGE exhibits scalable, linearly increasing query latency when tested on DBLP.

As with the synthetic graphs, the experiments over a real network demonstrate a linearly increasing scale between average query time and the number of edges.

5.2.1 *Wildcard Cost*

In order to measure the potential added overhead of wildcards, we tested across multiple sizes of graph with varying numbers of wildcards in a common query. For each number of wildcards, we averaged multiple runs and compare these against the same query graph with specified attributes.

Figure 5 shows the query time when using wildcards relative to a query with only normal attributes. The wildcard overhead increases linearly with the number of wildcards. This suggests that large queries can safely incorporate wildcards without computational constraints, reducing the cognitive load on MAGE users during their data mining.

5.2.2 *Multi-attribute Cost*

We performed a series of tests to determine the costs of multi-attribute queries by varying the size of graph and running queries with both single attributes and multi-attributes. For the multi-attributes, we embedded each node and edge with 3 distinct attributes, tripling the amount of queryable attribute data in the graph.

Figure 6 indicates a very small difference in query latency between the regular query and the multi-attribute query. Data-rich graphs can be queried in MAGE with minor increases in query latency. Multi-

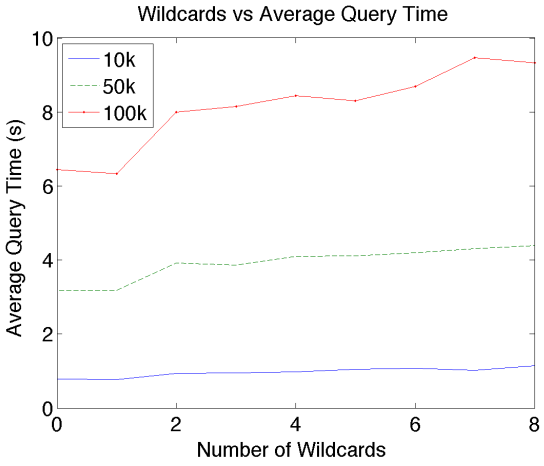


Figure 5: This experiment shows the query latencies for queries containing wildcards tested on various sizes of Watts-Strogatz networks. Adding additional wildcards to a query increases the query latency at a linear rate.

Table 2: Real graphs used in experiment

Network	Nodes	Edges
DBLP	317,080	1,049,866
LiveJournal	4,847,571	68,993,773
Twitter	41,652,230	1,468,365,182

attributes queries can be submitted without fear of incurring decreased query performance.

5.2.3 *Memory-Mapped RWR Results*

MAGE utilizes memory mapping to reduce the time needed for each RWR query. To illustrate the effect of memory mapping, here we compare our hybrid implementation that uses MATLAB plus memory mapping in Java with pure implementation in MATLAB. Previous research [13] showed the effectiveness of Java in mapping graphs to memory, however Java still lacks a sparse matrix library that can scale to graphs with more than a million of nodes. We prefer not to use MATLAB's native memory mapping capability either since it is not highly optimized and incurs overhead[1]. Therefore, our hybrid implementation takes advantage of MATLAB's efficient matrix library and Java's optimized memory mapping capability. The datasets we use in the comparison are: the DBLP graph with 1 million edges [26], a LiveJournal graph with 69 million edges [2], and a Twitter graph with 1.47 billion edges [12]. Table 2 shows the exact statistics of the graphs. Our protocol consists of executing 10 iterations of the RWR algorithm

1 http://www.mathworks.com/support/solutions/en/data/1-1OG2ML/

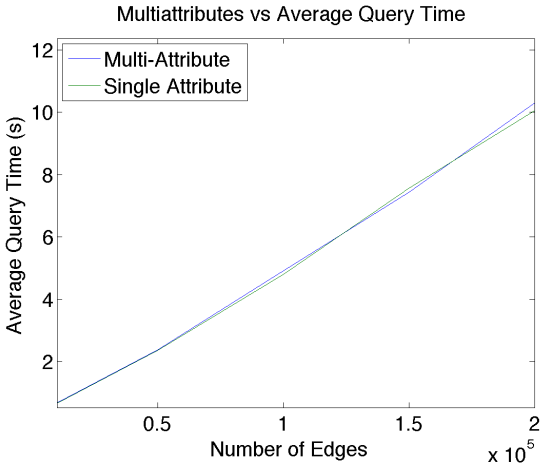


Figure 6: The introduction of 2 additional attributes per node and edge barely affects query latency. Tests were run on varying sizes of Watts-Strogatz small-world networks.

for the same node 3 times and reporting the average time needed for the query.

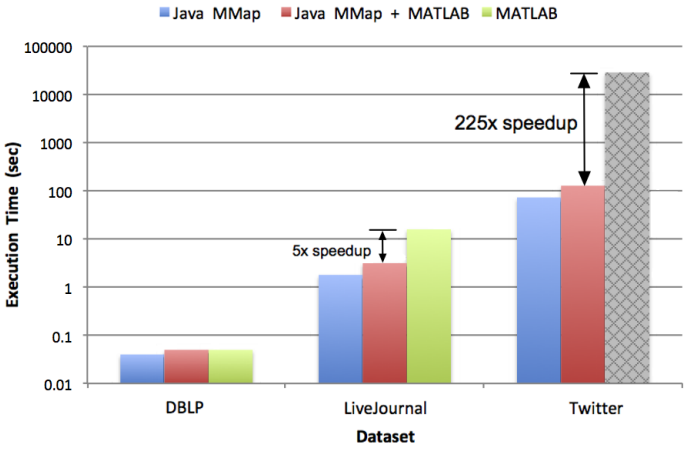


Figure 7: Comparing the elapsed times of three approaches for a RWR query with three graphs. Java does not have a matrix library and is included as a lower bound. MATLAB is not able to scale to large graphs (diamond pattern means it could not compute the result within 8 hours). Our hybrid approach significantly improves the query latency. For example, on Twitter graph (1.47 billion edges), our method is at least 2 orders of magnitudes faster.

Figure 7 shows the elapsed times for the RWR queries. We include the results for the pure Java implementation as a lower bound. From the figure, we observe three outcomes: (i) the three approaches perform similarly when the graph is small, (ii) pure MATLAB implementation cannot scale to large graphs (it could not compute the RWR probabilities within 8 hours in the case of the Twitter graph), and (iii) our hybrid approach is slightly slower than the pure Java implemen-

tation. The results show that our hybrid approach can scale to very large graphs while retaining low query latency.

5.3 EFFECTIVENESS

For the purposes of demonstrating the result quality we show the results for several queries first on a synthetic toy-graph and second on Rotten Tomatoes movie data.

5.3.1 *Query Examples*

The following graphs illustrate the ranked results return by MAGE when run on the toy graph presented in Figure 8.

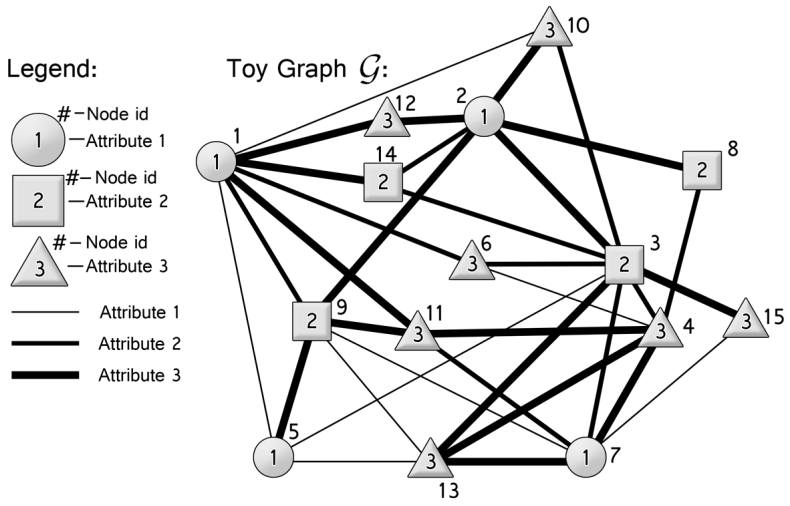


Figure 8: A toy graph with categorically attributed nodes and edges. In this graph the edges are *encoding categorical attributes* (see legend above, left), not edge weight.

5.3.2 *Rotten Tomatoes Query Results*

In order to test the effectiveness of MAGE we have queried a movie graph constructed from Rotten Tomatoes similar movies. Rotten Tomatoes allows users to up-vote algorithmic and suggested similar films. We discretize this crowd sourced similarity score into two main categories, weakly similar and strongly similar. For each movie we have extracted and encoded a plethora of categorical attributes. From MPAA rating, runtime, critic review, to genres the query possibilities are rich. Figure 11 shows a few queries and results we scoured from this dataset.

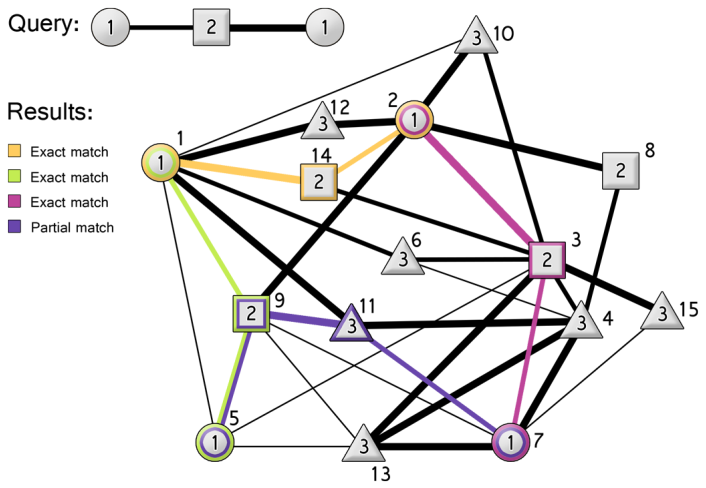


Figure 9: The first 4 results (with each result in its own color) for the linear query at the top of this figure. The fourth result (in purple) inserts and extra edge-node-edge into the pattern to complete the approximation. Nodes or edges shared by query results will contain both result colors.

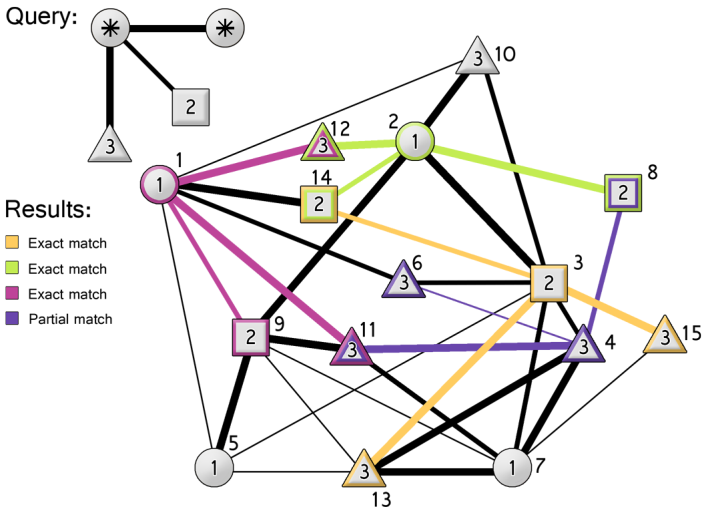


Figure 10: The first 4 results for the specified star query. Note that this query has a wildcard as the center of the star, and another at the end of one of the legs. The fourth result is a partial match.

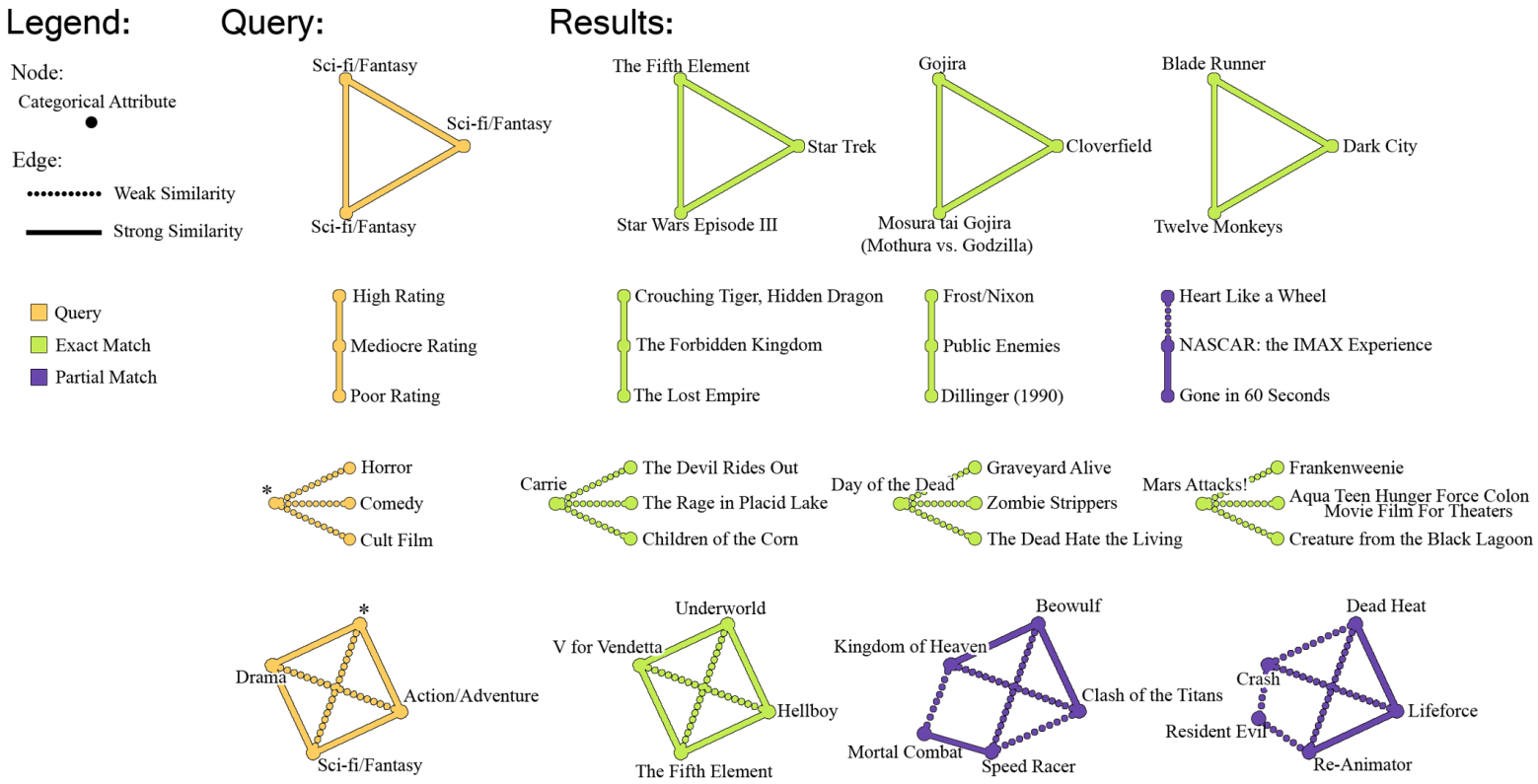


Figure 11: Rotten Tomatoes queries and example results. Each node represents a movie; each edge between two movies indicates that they are similar. The two edge classes (see legend) were derived from user-contributed similarity scores. Results that exactly matched the query are presented in green while partially matched results are presented in purple.

6 RELATED WORK

Our work finds similarities to two lines of research. Below, we review related work from each category.

Proximity search on graphs. Various techniques and approaches for graph-related problems require measures of node-to-node proximities. There has been extensive work on graph data structures and databases for this purpose [1, 17]. Methods in indexing have been proposed to improve the capabilities and responsiveness of graph query tools [27]. The two main approaches used to compute proximity in graphs are random walk with restart [11, 16, 21, 8] and PageRank [4].

Pattern matching on graphs. There has been research on pattern matching against single-large graphs [25], where Wolverton et al. developed the Link Analysis Workbench for use within the intelligence industry. The goal of this research was to create a system to aid an expert user from the intelligence community in creating, maintaining and matching patterns across large quantities of relational data. The pattern-matching module in this work heavily utilized a graph edit distance metric to create scores for graph-graph similarity. Rather than a graph edit distance, we utilized random walk with restart in the goodness of match criterion.

Matching attributed graphs has been studied in [5] again as a potential tool for the intelligence industry. With the enormous amount of content in modern graphs, the primary concern of the modern intelligence analyst is sifting for useful data amongst torrents of unrelated content. This approach allows an analyst to specify a particular pattern in an attributed relational graph and scour a much larger dataset for occurrences of such a structure. This was done primarily with observed activity graphs involving large quantities of relational intelligence data over large numbers of entities. This problem is still challenging and highly relevant today.

Approximate matching is a common relaxation to the subgraph isomorphism problem and has been researched heavily [18, 15]. Inexact graph matching has also been investigated with special emphasis on web content scanning. The system OntoSeek utilizes inexact graph matching based on linguistic ontologies over a large collection of keywords called WordNet [9]. OntoSeek was designed to simplify and improve query results over large catalogs and yellow pages.

There are a host of purely structural graph matching systems exploring polynomial time solutions to variants of the subgraph isomorphism problem. These systems take into consideration just the structure and connectivity of a set of edges and vertices [14, 22, 23].

Others have investigated the source of intractability for various query patterns [3]. These approaches generally do not utilize any semantic content from the graphs themselves, making it challenging for these approaches to extend fully to our problem formulation.

While there are many separate approaches that work very well at their particular focus, there are few algorithms that combine the aforementioned techniques to tackle inexact matching in large, attributed graphs with highly variable content. Recent work include [7] and [10], however the former requires the user to specify a "focus" node in the query and the latter returns results that do not adhere to the query structure. The closest work is that of Tong et al. [20], which proposes graph X-ray or G-Ray, a method that finds subgraphs that exactly or approximately match a desirable query pattern. The G-Ray algorithm is based on two core concepts. The first is the random walk with restart idea [21], which is used to estimate the goodness of a match between a subgraph and a query graph. The goodness of match metric is necessary to rank the quality of approximate query matches extracted from the graph. The second key concept in G-Ray is the CenterPiece Subgraphs idea [19]. The CenterPiece approach is used to locate subgraphs that have high goodness score as candidate query results. At a high level, G-Ray works iteratively by finding a seed node and forming a seed-local base set, expanding the base set, and bridging the nodes in the base set, all according to the query graph. The main drawback of G-Ray is that it only supports graphs with node attributes. This is a significant limitation, considering the additional semantics contributed to the graphs by the edge attributes. We have leveraged many of the ideas and techniques proposed in this work to create MAGE.

7 CONCLUSION

In this paper we address the problem of pattern matching on graphs with rich attributes. A typical query we consider is, "return all the rings of businessmen with strong ties", where occupation is the node attribute and tie strength is the edge attribute. To the best of our knowledge, ours is the first pattern matching approach that is capable of performing exact and inexact matching on graphs with both node and edge attributes for which wildcards and multiple attribute values are permissible. Our technique is highly scalable; it leverages a novel technique based on memory mapping to compute random walk with restart probabilities, which provides a speedup of more than 2 orders of magnitude on large graphs. We evaluated the effectiveness and scalability of our approach with real and synthetic graphs with up to 2.3 million edges. Experimental results on the DBLP authorship graph demonstrate a scalable design on real networks. Through our experiments we demonstrate a linear cost increase when adding wildcards to a query and minimal overhead when adding multiple attributes per node and edge. The Rotten Tomatoes movie graph experiments illustrate the effectiveness and exploratory functionality of our contributions to graph querying. We believe our work improves the flexibility with which a user can explore the patterns and structure within their graph-data through exploratory queries.

BIBLIOGRAPHY

[1] I. Antonellis, H. Garcia-Molina, and C.-C. Chang. Simrank++: Query rewriting through link analysis of the click graph. *CoRR*, abs/0712.0499, 2007.

[2] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54, 2006.

[3] P. Barceló, L. Libkin, and J. L. Reutter. Querying graph patterns. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 199–210, New York, NY, USA, 2011. ACM.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[5] T. Coffman, S. Greenblatt, and S. Marcus. Graph-based technologies for intelligence analysis. *Commun. ACM*, 47(3):45–47, 2004.

[6] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[7] W. Fan, X. Wang, and Y. Wu. Diversified top-k graph pattern matching. *PVLDB*, 6(13):1304–1315, 2013.

[8] Y. Fujiwara, M. Nakatsuji, M. Onizuka, and M. Kitsuregawa. Fast and exact top-k search for random walk with restart. *CoRR*, abs/1201.6566, 2012.

[9] N. Guarino, O. Content-based, G. Vetere, and C. Masolo. Ontoseek: Content-based access to the web. *IEEE Intelligent Systems and Their Applications*, 14(3):70–80, 1999.

[10] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan. Nema: Fast graph search with label similarity. *PVLDB*, 6(3):181–192, 2013.

[11] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation, 2009.

[12] H. Kwak, C. Lee, H. Park, and S. B. Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[13] Z. Lin, D. H. Chau, and U. Kang. Leveraging memory mapping for fast and scalable graph computation on a pc. In *Proceedings*

of the IEEE International Conference on Big Data Scalable Machine Learning Workshop, 2013.

[14] B. Messmer and H. Bunke. Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In S. Li, D. Mital, E. Teoh, and H. Wang, editors, *Recent Developments in Computer Vision*, volume 1035 of *Lecture Notes in Computer Science*, pages 373–382. Springer Berlin Heidelberg, 1996.

[15] M. Mongioví, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. Sigma: A set-cover-based approach for inexact graph matching. *J. Bioinformatics and Computational Biology*, 8(2), 2010.

[16] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *Data Mining, Fifth IEEE International Conference on*, pages 8 pp.–, 2005.

[17] Y. Sun, J. Han, X. Yan, P. S. Yu, and T. Wu. Pathsim: Meta path-based top-k similarity search in heterogeneous information networks. In *In VLDB 2011*, 2011.

[18] Y. Tian and J. Patel. Tale: A tool for approximate large graph matching. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 963–972, 2008.

[19] H. Tong and C. Faloutsos. Center-piece subgraphs: problem definition and fast solutions. In *KDD*, pages 404–413, 2006.

[20] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast best-effort pattern matching in large attributed graphs. In *KDD*, pages 737–746, 2007.

[21] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622, 2006.

[22] J. R. Ullmann. An algorithm for subgraph isomorphism. *J. ACM*, 23(1):31–42, Jan. 1976.

[23] T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, July 2003.

[24] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, 54(1):150–168, 1932.

[25] M. Wolverton, P. Berry, I. W. Harrison, J. D. Lowrance, D. N. Morley, A. C. Rodriguez, E. H. Ruspini, and J. Thoméré. Law: A workbench for approximate pattern matching in relational data. In *IAAI*, pages 143–150, 2003.

[26] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 745–754, 2012.

[27] P. Zhao and J. Han. On graph query optimization in large networks. *Proc. VLDB Endow.*, 3(1-2):340–351, Sept. 2010.