Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching

Brian Gallagher

Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Box 808, L-560
Livermore, CA 94551
bgallagher@llnl.gov

Abstract

The task of matching patterns in graph-structured data has applications in such diverse areas as computer vision, biology, electronics, computer aided design, social networks, and intelligence analysis. Consequently, work on graph-based pattern matching spans a wide range of research communities. Due to variations in graph characteristics and application requirements, graph matching is not a single problem, but a set of related problems. This paper presents a survey of existing work on graph matching, describing variations among problems, general and specific solution approaches, evaluation techniques, and directions for further research. An emphasis is given to techniques that apply to general graphs with semantic characteristics.

1. Introduction

Work on pattern matching in graphs spans a diverse range of research communities within and beyond computer science. Relevant research and application areas include databases, computer vision, mathematical graph theory, artificial intelligence, information retrieval, biology, electronics, computer aided design, and knowledge discovery and data mining.

Graph-based pattern matching is not a single problem, but a set of related problems. These range from the NP-complete *subgraph isomorphism* problem, in which matches are based strictly on graph structure, to finding inexact matches to complex patterns in semantic graphs with millions of typed and attributed vertices and edges. The focus of this survey is on techniques applicable to general graphs that may have semantic characteristics. It does not cover specialized matching approaches for structural subclasses of graphs, such as trees or planar graphs.

In the remainder of this section I present a formal description of the basic graph pattern matching problem and discuss a number of common problem variations. In section 2, I outline a general strategy that has been applied by many graph matching approaches. In section 3, I discuss a number of specific approaches in detail. Section 4 covers the evaluation of graph pattern matching algorithms. Section 5 summarizes the findings of this survey and discusses directions for future research.

1.1 The Graph Pattern Matching Problem

The basic graph pattern matching problem is to find matches in a graph for a specified pattern. More formally, we are given:

1. A **data graph** $G = (V, E)$, composed of a set of vertices V and a set of edges E. Each $e \in E$ is a pair (v_i, v_j) where $v_i, v_j \in V$. The vertices and/or edges of G may be typed and/or attributed.
2. A **pattern graph (or pattern query)** $P = (V_P, E_P)$, which specifies the structural and semantic requirements that a subgraph of G must satisfy in order to match the pattern P.

The task is to find the set M of subgraphs of G that "match" the pattern P. A graph $G' = (V', E')$ is a subgraph of G if and only if $V' \subseteq V$ and $E' \subseteq E$. Problem formulations often require that P represent a single connected graph and, therefore, that $m \in M$ is connected as well. A graph is connected if there exists some path between every pair of its vertices.

The precise definition of a match varies among problems, but is generally based on a combination of (1) isomorphism (i.e., structural matching) or near isomorphism between P and $m \in M$ and (2) equality or similarity between the types and attribute values of the vertices and edges in P and those in $m \in M$. Formally, a match may be thought of as a pair of bijections, one from V_P to a subset of V and the other from E_P to a subset of E.

1.2 Problem Variations

There are a number of variations on the basic graph pattern matching problem. Variation generally occurs along the following dimensions.

Graph properties. All graphs share the basic structural elements, vertices and edges, but other structural and semantic graph properties vary among applications.

Most structural differences are due to restrictions on edges. For example, graphs may allow or disallow directed or undirected edges, weights on edges, multiple edges between a pair of vertices (multigraphs), and edges that connect a single vertex to itself (self-loops).

In their simplest form, graphs are mathematical objects. However, like many ideas from mathematics, graphs may be used to model real-world phenomena. In particular, vertices are often used to represent objects or entities (e.g., people, places, events, molecules) and edges are used to represent relationships between entities. We can extend the idea of a graph from a set of homogeneous vertices connected by homogeneous edges to a collection of individual entities, possibly of different types, each with their own unique characteristics, and each involved in relationships with one another. To support this expanded notion, graphs may be *typed*. That is, each vertex and/or edge in the graph may be assigned a type (or label) from predefined sets. For example, if our domain of interest is movies, vertex types may include *movie*, *actor*, and *producer* and edge types may include *(actor) appeared-in (movie)*, *(movie) sequel-to (movie)*, and *(producer) produced (movie)*. In addition, graphs may be *attributed*. That is, vertices and edges may contain attributes. For example, a *movie* vertex may have attributes such as *title*, *year*, and *genre* and an *actor* vertex may have attributes such as *name* and *age*.

A *semantic graph* is a graph-structured data representation in which vertices represent concepts (e.g., *movie*, *actor*) and edges represent relationships between concepts (e.g., *appeared-in*). Both vertices and edges in a semantic graph are typed and attributed. Furthermore, a semantic graph has an associated *ontology*, which specifies the possible concepts, the relationships allowed between each pair of concepts, and the attributes associated with each concept and relationship.

Structural vs. semantic matching. Many graph matching approaches find matches based strictly on structural similarity (McKay 1990; Messmer and Bunke 1995; Ullmann 1976; Washio and Motoda 2003). However, since graphs can serve as conceptual representations, it can be useful to match graphs based not strictly on their similarity as graphs, but on the similarity of their interpretations in some domain of interest. Since the meaning of semantic graphs is contained largely within the type and attribute information stored on individual vertices and edges, structural matching is often insufficient for finding conceptually similar graphs. Semantic matching approaches attempt to match graphs based on their meaning by taking into account vertex and edge types and attributes as well as graph structure (Aleman-Mesa et al. 2005; Coffman, Greenblatt, and Marcus 2004; Wolverton et al. 2003).

Single-graph vs. graph-transaction setting. Graph data may consist of a single large graph or a set of relatively small graphs (often called *transactions*). These cases are referred to as the *single-graph setting* and the *graph-transaction setting*, respectively (Kuramochi and Karypis 2005). The single-graph setting is more general and algorithms developed for this setting can be readily applied to the graph-transaction setting, although the converse is generally not true. Throughout this survey, I use the generic term *graph data set* (or just *data set*) to refer to a collection of related graph-structured data, whether it is organized as a single large graph or many small graphs.

Exact vs. inexact matching. A graph matching algorithm may return only results that match a specified pattern exactly or it may return a ranked list of the most similar matches (i.e., inexact matching). Inexact matching algorithms are often referred to as *error-correcting* since they enable matching in the presence of noise or errors in data (Shapiro and Haralick 1981; Tsai and Fu 1979). In addition, some systems allow patterns to be left only partially specified (e.g., using "wildcards" or cardinality operators) (Blau, Immerman, and Jensen 2002; Wolverton et al. 2003). In such cases, while results may match the pattern exactly, this is a form of inexact matching because the pattern itself is imprecise (e.g., "find all *movie* vertices" vs. "find all *movie* vertices linked to an *actor* vertex").

Optimal vs. approximate solutions. Distinct from whether an algorithm performs exact or inexact matching, algorithms vary in terms of solution quality guarantees. Optimal algorithms are guaranteed to find a correct solution (e.g., for exact matching, a set of exactly those subgraphs that match the pattern; for inexact matching, the closest match or a correctly ranked list of matches), but have exponential worst-case complexity (Shasha, Wang, and Giugno 2002). Approximate algorithms (Christmas, Kittler, and Petrou 1995; Umeyama 1998) often have polynomial complexity, but are not guaranteed to find a correct solution (e.g., for exact matching, some, but not all matches; for inexact matching, a close match, but not the closest). Optimal algorithms are generally search-based and approximate algorithms tend to be numerical (Kim, Yun, and Lee 2004).

Graph matching vs. graph mining. There are a variety of problems that build on graph matching. One such problem is that of *graph mining* or *structural motif finding*. Whereas the goal of graph matching is to find occurrences of a specific pattern in a graph, the goal of graph mining is to find a set of the most common or most "interesting" patterns in a graph (Cook and Holder 1994; Goethals, Hoekx, and Van den Bussche 2005; Kuramochi and Karypis 2005; Washio and Motoda 2003; Wörlein et al. 2005). This survey discusses only the graph matching aspects of graph mining systems. For a more thorough treatment of graph mining, see the surveys by Washio and Motoda and Wörlein et al. (Washio and Motoda 2003; Wörlein et al. 2005). In addition, an earlier version of this paper discusses several graph mining algorithms in more detail (Gallagher 2006).

2. A General-Purpose Matching Approach

Subgraph isomorphism is the problem of determining whether one graph P is isomorphic to a subgraph of

another graph *G* (i.e., whether the pattern *P* has a structural match in *G*). Since subgraph isomorphism is NP-complete (Washio and Motoda 2003), all known algorithms are exponential in the size of the input graphs. Therefore, it is impractical to solve subgraph isomorphism directly in large graphs. This leaves two options for fast pattern matching in large general graphs: (1) use an approximate algorithm, which may yield non-optimal solutions or (2) use an optimal algorithm, but apply it to only a subset of the data. In general, this second approach is achieved by performing some pre-processing to filter out unpromising portions of a data set before any direct matching takes place. This data filtering step is known as *candidate selection*.

The GraphGrep graph matching algorithm (Giugno and Shasha 2002) consists of three basic components: index construction, database filtering, and subgraph matching. This framework can be generalized to describe the majority of optimal graph matching algorithms. In general, these algorithms have three phases: data analysis and metadata construction, candidate selection, and matching.

Data analysis and metadata construction

Many algorithms perform some sort of pre-processing on a data set to create summary information, which informs the pattern matching process. *Graph invariants* are a common example of such summary information. An invariant is a quantity used to characterize a graph (Washio and Motoda 2003). If two graphs are identical, they will have identical invariants, although the converse is not necessarily true. Due to this property, a simple comparison of invariant values between pattern and data graphs may be sufficient to eliminate many non-matches. Graph invariants are most commonly used in the graph-transaction setting and are generally applied to exact matching. The nauty algorithm (McKay 1990) computes graph invariants for each vertex in a graph (e.g., degree). Several algorithms use a canonical graph representation to derive invariants (Washio and Motoda 2003). GraphGrep (Giugno and Shasha 2002) creates a "fingerprint" for each graph in a data set using path-based invariants.

A complimentary approach to calculating invariants is to create statistical summaries of an entire graph data set. For example, Statistical Relational Models (SRMs) (Getoor 2001) model dependencies among attributes in relational data by utilizing conditional independence properties among attributes. SRMs can provide approximate answers to counting queries (e.g., how many *movies* have *actors* with the *last name* "Smith"?). These approximate counts are helpful for determining an efficient ordering of filtering criteria when multiple criteria exist for eliminating non-matches (e.g., multiple graph invariants).

In addition to the types of summary information already mentioned, it is often useful to construct indices into a graph data set. Indices for graph data can take many forms. For example, we may index occurrences of common structural patterns in a large graph so that we can locate them quickly. The *embedding lists* used by graph mining algorithms are an example of this (Wörlein et al. 2005). In semantic graphs, it is common to retrieve all vertices or edges of a particular type or with a particular attribute value (e.g., 'find all *movies*' or 'find all *actors* with the last name "Smith"'). These operations can be sped up using indices on types and attributes.

Candidate selection

Once we have constructed appropriate metadata, we can use it to direct our search for matching subgraphs. If graph invariants differ between a subgraph *S* and a pattern *P*, there is no need to perform direct matching between *P* and *S*. If an SRM tells us that a match to our query is very unlikely, we may not bother searching for an exact match. Statistical models such as SRMS can also help us decide which non-matches to filter out first by providing *selectivity estimates* (i.e., determining which criteria in a pattern are the most selective or occur least frequently in the data set). The TRAKS (Aleman-Mesa et al. 2005) and LAW (Wolverton et al. 2003) systems use simple frequency statistics to perform this kind of selective pruning on the space of potential matches.

It is worth noting that effective candidate selection in semantic graphs is possible without generating graph invariants since semantic graphs already contain rich data on which to filter potential matches (i.e., the types and attributes of vertices and edges). For example, if a pattern contains a vertex *M* of type *movie*, we don't need to consider every vertex in a data graph as a potential match to *M*, only those vertices of type *movie*. Even so, type and attribute based indices can speed up this filtering process and statistical summaries such as SRMs may prove useful for creating an efficient ordering of candidate filters.

Upon completing candidate selection, we have a list of candidates on which to perform matching. Candidate selection aims to produce as small a list of candidates as possible without eliminating any true matches from the list.

Matching

In the matching phase, candidates identified during candidate selection are checked against the specified pattern. Researchers have proposed numerous matching algorithms, which may be generally categorized as search-based (optimal algorithms) or numerical (approximate algorithms) (Kim, Yun, and Lee 2004). The focus of this survey is on search-based solutions. The next section provides details on several specific matching approaches.

3. Specific Matching Approaches

This section discusses specific approaches for exact and inexact matching based on graph structure and semantics. The set of approaches covered here is illustrative, but not exhaustive. For additional matching approaches see the survey by Shasha et al. (Shasha, Wang, and Giugno 2002).

3.1 Structural Matching Approaches

One of the earliest and most highly-cited approaches to exact pattern matching is the subgraph isomorphism algorithm proposed by Ullmann (Ullmann 1976). This algorithm operates on single untyped graphs with directed or undirected edges. Suppose we want to find matches to the pattern graph *P* in the data graph *G* (Figure 1).

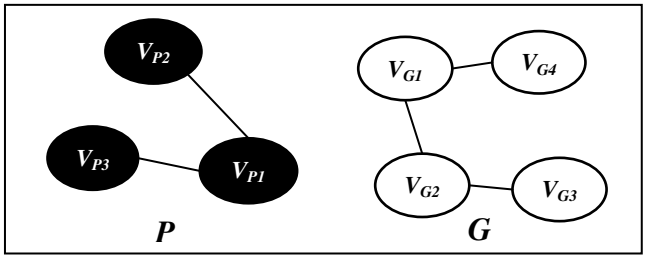

Figure 1: An example pattern graph P and data graph G

Ullmann's basic approach is to enumerate all possible mappings of vertices in *P* to those in *G* using a depth-first tree-search algorithm. Each node at level *i* of the search-tree maps vertex V_{Pi} in *P* to some vertex in *G* (Figure 2).

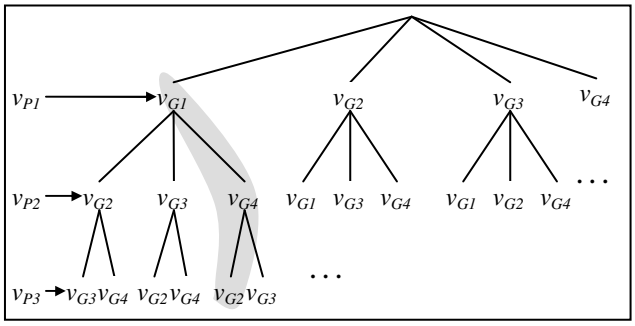

Figure 2: A partial search-tree for Ullmann's algorithm, mapping vertices from pattern graph P to data graph G. The highlighted path represents a match for P in G.

Each path from root to leaf in the search-tree represents a complete mapping of the vertices in *P* to those in *G*. Any such mapping that preserves adjacency in *P* and *G* (i.e., vertices that are neighbors in *P* map to vertices that are neighbors in *G*) represents an isomorphism from *P* to a subgraph of *G*. If no such mapping preserves adjacency, then no isomorphism exists. Since the search-space considered by this approach increases exponentially with the size of the input graphs, Ullmann suggests a refinement procedure to prune unpromising sub-trees, eliminating the need to search them. This procedure eliminates vertex mappings from consideration based on three criteria:

Vertex degree. If the degree of vertex V_{Pi} (i.e., the number of edges adjacent to V_{Pi}) is greater than the degree of V_{Gj} then V_{Pi} cannot map to V_{Gj}. For example, in Figure 1, V_{P1} cannot map to V_{G4} since degree(V_{P1})=2 and degree(V_{G4})=1.

One-to-one mapping of vertices. Once we decide to map V_{Pi} to V_{Gj}, along a particular path through the tree, we cannot map to V_{Pi} any other vertex in *G* and we cannot map any other vertex in *P* to V_{Gj}.

Forward checking. As we work our way down the tree, for any possible vertex mapping that remains, we can eliminate the mapping if it cannot preserve adjacency between *P* and *G*. For example, suppose that we have mapped V_{P1} to V_{G1} and we are considering the possible mapping from V_{P2} to V_{G3}. Regardless of what we do further down the tree, mapping V_{P2} to V_{G3} cannot possibly preserve adjacency since V_{P1} and V_{P2} are neighbors in *P*, but V_{G1} and V_{G3} are not neighbors in *G*. So, we can eliminate the mapping from V_{P2} to V_{G3} from further consideration.

As Ullmann's algorithm expands a particular path in the search-tree, one of two things happens:

1. The algorithm eliminates all possible mappings for some vertex in *P*. In this case, the path we are on cannot yield a match. We can safely stop, without expanding additional nodes along the current path, and backtrack.
2. The algorithm reaches a leaf in the tree, having mapped each vertex in *P* to a vertex in *G*. In this case, the path represents a match for *P* in *G* (Figure 2).

As noted by Messmer and Bunke, despite the refinement procedure, Ullmann's algorithm has exponential worst-case time-complexity (Messmer and Bunke 1995). They propose an alternative method for exact subgraph isomorphism that has only quadratic worst-case time complexity. Their algorithm also operates on multiple untyped graphs with directed or undirected edges. The approach is to pre-process the graph data set to generate all possible permutations of the graph adjacency matrices offline and use them to build a decision tree. At run time the decision tree is used to classify the adjacency matrix of the pattern graph. The drawback to this approach is that the size of the decision tree grows exponentially with respect to the size of the data graph. To address this issue, the authors present pruning techniques, which are effective in reducing decision tree size. However, the pruned decision trees can no longer guarantee polynomial run times.

The nauty algorithm (McKay 1990) detects isomorphism between untyped graphs that may be directed or undirected. Nauty uses transformations to reduce graphs to a canonical form that may be checked relatively quickly for isomorphism (Washio and Motoda 2003). Specifically, the algorithm computes invariants for each vertex in a graph (e.g., degree and counts of adjacent vertices of various degrees) that are used for candidate selection. Nauty partitions a graph into non-overlapping sets of vertices based on invariant values. Sets having the same invariant values can then be compared between graphs. If all sets are isomorphic between two graphs, then the two graphs must be isomorphic. Alternatively, if two graphs contain sets with differing invariants, there is no need to test isomorphism between the sets directly.

SUBDUE (Cook and Holder 1994) operates in a single-graph setting with typed vertices and typed, directed edges. SUBDUE is a graph mining system, but performs pattern matching as a supporting step in the mining process. The general approach is similar to Ullmann's. They construct a

search-tree, where the nodes at the i^{th} level map the i^{th} vertex from P to some vertex in G. A path through the tree represents a complete mapping of vertices. Since SUBDUE performs inexact matching, each node in the search-tree has an associated cost that captures how well P matches G. If P and G are exactly isomorphic, there will be a mapping between them with cost 0. The less similar P and G are, the higher the cost will be. These costs are based on *graph edit distance* (Meyers, Wilson, and Hancock 2000). The edit distance between two graphs is the minimum cost of edit operations required to transform one graph into another. Edit operations include deletion, insertion, and substitution of vertices and edges. For inexact matching, the goal state is the final state (i.e., leaf) with the least cost of all final states. Since the search-space is again exponentially large, SUBDUE applies a branch-and-bound search to the tree. Because branch-and-bound is guaranteed to find an optimal solution, the search terminates once any complete mapping is found. The algorithm also allows an upper limit to be placed on the number of search nodes considered, which can lead to a significant savings in search time at the expense of solution quality.

3.2 Semantic Matching Approaches

So far, I have discussed techniques that match graphs based solely on structure. Here, I present several techniques that attempt to match graphs based on their conceptual interpretations.

Early approaches to inexact matching were proposed by Tsai and Fu (Tsai and Fu 1979) and Shapiro and Haralick (Shapiro and Haralick 1981). Both approaches are based on the idea of measuring similarity between graphs as the probability that one graph could result from a random alteration of the other. Both approaches match based on graph structure and on the attributes of individual graph elements. Both approaches are implemented using search-based algorithms with various pruning strategies. Tsai and Fu require an exact structural match, but allow for attribute value differences. They propose calculating empirical probabilities of attribute deformations based on observations from data. In addition, they propose the *weighted distance* and *weighted-square-error distance* measures for cases where such data is unavailable. Shapiro and Haralick support inexact structural matching by considering graphs to match only if the amount of non-matching structure falls within some threshold. More important structural elements are given more weight and the presence or absence of these elements more heavily influences the determination of a match. Likewise, graphs only match if differences between attribute values of corresponding graph elements fall within some threshold.

As previously discussed, SUBDUE determines similarity between graphs using graph edit distance. The edit operations used and their associated costs may be purely structural or they may be based on semantics as well. For example, the cost of a vertex substitution could vary based on the semantic similarity of the vertex types involved (Djoko, Cook, and Holder 1997).

GraphGrep (Giugno and Shasha 2002) operates in the graph-transaction setting on undirected graphs with typed vertices. The algorithm makes implicit use of vertex type information to perform matching. Matching in GraphGrep relies on the concept of a *label path*, which is a sequence of type labels along a path in a graph (e.g., *actor-movie-director-movie-actor*). During index construction, the algorithm computes a "fingerprint" for each graph in the data set. The fingerprint of a graph is a set of pairs $\langle h(labelPath), count \rangle$, one for each unique label path in the graph. Here h is a hash function and *count* is the number of instances of the specified label path in the graph. During candidate selection, the data set is filtered based on the fingerprint of the pattern graph P. Specifically, if a graph G has a lower *count* value than P for any *labelPath*, then G cannot contain an exact match for P and G is eliminated from consideration. During the subgraph matching phase, P is broken up into a set of overlapping label paths, which are compared against the candidate graphs. Label paths of the candidate graphs that match P's label paths may be combined into matching subgraphs.

A number of semantic matching approaches have grown out of the conceptual graphs literature. The theory of conceptual graphs is extensive and a full treatment is beyond the scope of this paper. See Chein and Mugnier (Chein and Mugnier 1992) and Sowa (Sowa 1984) for more background on conceptual graphs. For our purposes, conceptual graphs may be thought of simply as graphs with typed vertices and edges. Here we describe OntoSeek as an example of a conceptual graph matcher. In section 3.3, we describe the work of Poole and Campbell, which makes use of a more sophisticated similarity model for matching conceptual graphs.

OntoSeek (Guarino, Masolo, and Vetere 1999) is an information retrieval system that matches documents to queries by representing each as conceptual graphs and then measuring the semantic similarity between graphs. The matching of a query Q to a document R requires an exact structural match between Q and a subgraph of R. However, individual vertices (i.e., concepts) are considered to match only if the type of a vertex in R is subsumed by the type of the corresponding vertex in Q (i.e., if the query concept is a generalization of the document concept). Matches are found using search with a "first-fail" heuristic (i.e., the least probable links are checked first, so that non-matches are discovered more quickly).

TMODS (Coffman, Greenblatt, and Marcus 2004; Greenblatt, Marcus, and Darr. 2005) uses genetic algorithms to find exact and inexact pattern matches in directed, attributed graphs. Patterns may specify both structural and attribute characteristics. TMODS searches for patterns from the bottom-up, finding sub-patterns first and then composing them into more complex higher-level patterns. Coffman et al. do not describe the TMODS pattern matching algorithm in further detail.

TRAKS (Aleman-Mesa et al. 2005) performs inexact pattern matching in typed, directed graphs. Matches are ranked by similarity to the original pattern, taking into

account ontological distance between types. Entities in a pattern are processed in ascending order of the frequency of their type to eliminate non-matches more quickly. The algorithm searches for matches in a depth-first fashion by expanding partial matches by one vertex or edge at a time.

LAW (Wolverton et al. 2003) performs inexact pattern matching on typed, directed graphs. Patterns are represented as graphs with typed vertices and edges. The pattern language also supports the construction of more sophisticated pattern queries through constraints between vertices, hierarchy (i.e., sub-patterns), disjunction, and cardinality (i.e., the number occurrences of a vertex or edge). Like SUDBUE, LAW uses graph edit distance to measure similarity between potential matches. LAW's graph edit operations include deletion and replacement of vertices and edges. LAW uses ontological distance to measure differences between types. The LAW search algorithm is based on A* and selects tree nodes for expansion based on the minimum worst-case cost. This cost is calculated as the true cost of the mappings so far plus the cost of deleting all unexplored vertices and edges in the pattern. Although the worst-case cost heuristic is not admissible (in fact, it is an upper bound on the actual cost), LAW does find the lowest-cost matches because, unlike pure A*, LAW uses the heuristic only as a selection rule and not as its termination condition (Wolverton 2006).

Like TRAKS, LAW generates start states by selecting the vertex in a pattern with the fewest legal mappings in the data. Partial matches are expanded by selecting unexplored vertex mappings and generating adjacent edge mappings. LAW uses an "anytime" version of A* that may be terminated at any point and will return the matches it has found so far. The set of matches is guaranteed to monotonically improve as the algorithm continues to run. The LAW pattern matcher uses a "search plan" to determine the order in which query elements are processed. Search plans may be specified by the user or automatically calculated based on a "statistical analysis of the data" (Wolverton et al. 2006). The authors do not describe the details of this statistical analysis or the search plan calculation.

Statistical Relational Models (SRMs) model the joint distribution over tuples in relational data and capture the frequencies with which the tuples join (Getoor 2001). Although this work does not provide an explicit pattern matching algorithm, it does demonstrate that SRMs exhibit substantially lower relative error than previous methods for estimating the size of a query's result set (i.e., methods that assume attribute independence or join uniformity). This suggests an approach for optimizing pattern queries for semantic graphs using query optimization techniques, as have been studied in the XML and database communities (Gibbons and Garofalakis 2001; Ioannidis and Poosala 1995; Polyzotis and Garofalakis 2002; Wang et al. 2004).

3.3 Similarity-Based Matching Approaches

Inexact matching approaches, such as those employed by SUBDUE, OntoSeek, TMODS, TRAKS, and LAW, as well as the approaches of Tsai and Fu and Shapiro and Haralick, match graphs based on their "similarity" to each other. Such approaches depend upon similarity measures to make this determination.

Since graphs may contain type and attribute information as well as structural information, a graph similarity measure must potentially take all of these characteristics into account. Structural similarity is often measured using graph edit distance, as described in sections 3.1 and 3.2 above. Edit distance can also capture semantic similarity if edit operations and costs reflect graph semantics. For example, LAW determines vertex replacement costs based on the ontological distance between the vertex types involved (Wolverton et al. 2003). Graph edit operations and costs can also take into account differences in attribute values of vertices and edges.

A drawback to edit-based distance measures (e.g., graph edit distance) is that they require the specification of costs for specific edit operations. It is generally not clear how to optimally assign these costs and changes in cost assignments can dramatically affect the resulting distance measure. As an alternative to graph edit distance, Bunke and his colleagues propose graph distance metrics based on the *maximal common subgraph* (Bunke and Shearer 1998) and *minimum common supergraph* (Bunke, Jiang, and Kandel 2000). These metrics can be thought of as measuring the amount of structural overlap between graphs. Bunke and colleagues show that both metrics can be related to graph edit distance by simple equations, under certain constraints on edit costs.

Attribute values may be compared using any number of similarity measures. Many similarity measures are data-type dependent (e.g., Euclidean distance, string edit distance, cosine similarity). Other similarity measures are more general. For example, Lin presents an information-theoretic definition of similarity and shows how it applies to strings, feature vectors, ordinal values, words, and concepts in a taxonomy (Lin 1998). Few of the pattern matching algorithms surveyed here appear to match based on attribute values. Exceptions include TMODS, LAW, and the work of Tsai and Fu and Shapiro and Haralick, as described in section 3.2. The literature describing TMODS and LAW does not specify the similarity measures used.

Poole and Campbell present a similarity measure for conceptual graphs based on the idea of *shared information* (Poole and Campbell 1995). They describe an algorithm for comparing two graphs, given an *interest* function. Interest is essentially a measure of how much information a graph contains. This approach is similar to Lin's information theoretic approach, but it is applied directly to graphs. Poole and Campbell do not consider specific interest measures.

Many of the approaches described in this survey take ontological distance between types into account when matching, although the specific ontological distance measures used are often not discussed. Many proposed ontological distance measures are based on the length of the path between concepts (i.e., types) in a hierarchy (Lin

1998, Sowa 1984). Other measures take into account both the concept hierarchy and the way that concepts are actually used in the data. For example, by Lin's information-theoretic definition, similarity between concepts is measured as the ratio between the amount of information needed to state the commonality of the concepts and the amount of information needed to fully state each concept. Applying a standard definition of information content, Lin's similarity measure amounts to: (1) concepts *A* and *B* are more similar the less their nearest common ancestor occurs in the data and (2) *A* and *B* are more similar the more each of *A* and *B* occur in the data.

In addition to the specific similarity measures proposed in the literature, there has been some effort to formalize a theory behind similarity-based graph matching. Bunke offers a formal definition of error correcting graph matching and shows that an optimal edit-based matching does not depend directly on the costs assigned to individual edit operations, but only on ratios of these costs (Bunke 1999). Berry et al. describe the derivation of graph similarity measures based on a theory of inexact pattern matching that views patterns as procedures for ranking potential matches (Berry et al. 2004).

4. Evaluation

As might be expected given the number of research communities involved in work on graph matching over the past several decades, it is difficult to evaluate the performance of the various techniques in relation to one another. Different algorithms have differing goals and researchers have evaluated their algorithms on data sets that vary tremendously in terms of size and graph characteristics. In addition, the complexity of the patterns evaluated is a huge potential source of variation among the results of various studies. Evaluations generally have not attempted to analyze or quantify the complexity of the patterns used for evaluation.

Evaluation of graph pattern matching systems to date has focused on runtime performance. Even for the inexact matching techniques surveyed here, there is no systematic evaluation of solution quality (e.g., precision and recall of matches). The most common evaluation metric is runtime vs. data set size. In the graph-transaction setting, size refers to the number of graphs in a data set. In the single-graph setting, size refers to the number of vertices (or edges) in a graph. Specific evaluation metrics include runtime vs. number of matches (i.e., pattern selectivity) (Ullmann 1976), runtime vs. edge count (Berry et al. 2004), and runtime, computation steps, and tree size vs. vertex and edge count, number of graphs, and decision tree depth (Messmer and Bunke 1995).

The data sets used for evaluation by most systems mentioned in this survey are synthetically generated graphs with either random or regular linkage (e.g., mesh structure). These graphs also tend to be quite small. For example, Ullmann's graphs have between 12-14 vertices and ~20-30 edges, the graphs used by Messmer and Bunke have up to 29 vertices and up to 44 edges, and nauty uses graphs with between 10-1000 vertices. The graphs used to evaluate LAW are the largest, with between 12,000 and 240,000 edges. GraphGrep was evaluated on National Cancer Institute data sets containing up to 16,000 individual graphs representing molecules. These graphs contain an average of 20 vertices and a maximum of 270 (Giugno and Shasha 2002).

In addition to the evaluations in individual studies, researchers have conducted benchmarking activities comparing several algorithms (Foggia, Sansone, and Vento 2001; Kälviäinen and Oja 1990). These comparisons consider the runtime and (in Kälviäinen and Oja) matching performance of graph isomorphism algorithms. They do not test subgraph isomorphism performance directly. Foggia et al. use various random and regularly structured synthetic graphs of up to 1000 nodes. Kälviäinen and Oja use very small graphs (<10 nodes) derived from image data. Neither study finds a dominating algorithm. Both studies show that in general graph matching algorithms are sensitive to graph characteristics (e.g., link structure), so it is important to consider these characteristics when choosing or developing an algorithm. This also suggests that existing evaluations may not tell us much about the applicability of algorithms to large real-world data sets.

5. Summary

The problem of searching for patterns in graph-structured data has many applications in diverse areas. Accordingly, numerous techniques have been developed for matching patterns in graphs. Together, these techniques represent decades of work by researchers from a range of research communities. Despite variations in properties of graphs, data sets, and algorithms, common themes have emerged. Since subgraph isomorphism algorithms are computationally expensive, keeping isomorphism calculations to a minimum is crucial to algorithm performance. Candidate selection is an effective means by which to accomplish this and techniques for metadata construction and application (e.g., data summarization, compression, modeling, and indexing) are central to effective candidate selection.

While there has been recent work on pattern matching based on semantics as well as structure, there appear to be opportunities to further exploit graph semantics for indexing, candidate selection, and matching. Based on the work reviewed in this survey, I wish to highlight the following observations about pattern matching in semantic graphs:

- Existing tree-search techniques (e.g., Ullmann's algorithm) can be readily extended to match and prune based on semantics as well as structure.
- Existing candidate selection and indexing strategies focus on graph structure. Type and attribute information can potentially help filter out irrelevant data more quickly. However, more sophisticated graph statistics are

required to capture the combination of attributes, type, and structure.

- Existing graph similarity measures do not incorporate all of attributes, type, and structure. An important question for inexact matching in semantic graphs is how to combine these different kinds of similarity to produce matches that are consistent with graph semantics. For example, if G_1 and G_2 are structurally similar, but have different attribute values and G_1 and G_3 have similar attribute values, but differ structurally, which of G_2 and G_3 is a better match for G_1?
- Many existing matching algorithms focus on the graph-transaction setting, where individual graphs tend to be very small. Therefore, many techniques are not directly applicable to large graphs (i.e., millions of vertices). Note that if candidate selection is effective in breaking large graphs into smaller graphs, techniques developed for the graph-transaction setting may applicable to these smaller graphs.
- Existing evaluation focuses on relatively small graphs, often with connections that are either random or regular. Real-world semantic graphs (e.g., social networks and the World Wide Web) are large and exhibit scale-free network characteristics (e.g., power-law degree distribution, high clustering coefficient, and short characteristic path length) (Dorogovtsev and Mendes 2003, Chapter 3.11). Therefore, existing evaluations tell us little about the applicability of algorithms to matching in semantic graphs.

Based on these observations, the following appear to be promising research directions for pattern matching in semantic graphs:

- Application of query optimization techniques from relational and XML databases to graph data sets. Specifically, the optimization of pattern queries based on selectivity estimates derived from probabilistic relational models. This may include learned models such as SRMs as well as statistical, mathematical, or probabilistic models that do not require learning, but are based on simple measures calculated from data graphs.
- Techniques for indexing and candidate selection that utilize graph structure as well as type and attribute information.
- Techniques for inexact matching that utilize both graph structure and semantics (i.e., distance measures that incorporate ontological distance between types as well as differences in attribute values and graph structure).
- Evaluation of matching algorithms on large real-world semantic graph data sets.
- Generation of synthetic data sets that reproduce characteristics of real-world data.
- Evaluation techniques that take into account the complexity and selectivity of the patterns used for evaluation.
- Evaluation of correctness or utility of inexact matching techniques.

6. Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48. UCRL-CONF-220852. Many thanks to Tina Eliassi-Rad for helpful comments on this survey.

7. References

Aleman-Meza, B.; Halaschek-Wiener, C.; Sahoo, S.S.; Sheth, A.; and Arpinar, I.B. 2005. Template Based Semantic Similarity for Security Applications. *Lecture Notes in Computer Science* 3495:621-622.

Berry, P.M.; Harrison, I.; Lowrance, J.D.; Rodriguez, A.C.; Ruspini, E.H.; Thomere, J.M.; Wolverton, M.J. 2004. Link Analysis Workbench. Tech Report AFRL-IF-RS-TR-2004-247, Air Force Research Laboratory.

Blau, H.; Immerman, N.; and Jensen, D. 2002. A Visual Language for Querying and Updating Graphs. Tech Report 2002-037, Dept. of Comp. Sci., Univ. of Mass. Amherst.

Bunke, H. 1999. Error Correcting Graph Matching: On the Influence of the Underlying Cost Function. *IEEE Trans. Pattern Analysis and Machine Intelligence* 21(9):917-922.

Bunke, H.; Jiang, X.; and Kandel, A. 2000. On the Minimum Common Supergraph of Two Graphs. *Computing* 65(1): 13-25.

Bunke, H. and Shearer, K. 1998. A Graph Distance Metric Based on the Maximal Common Subgraph. *Pattern Recognition Letters* 19(3-4): 255-259.

Chein, M. and Mugnier, M.-L. 1992. Conceptual Graphs: Fundamental Notions. *Revue d'Intelligence Artificielle* 6(4): 365-406.

Christmas, W.J.; Kittler, J.; and Petrou, M. 1995. Structural Matching in Computer Vision Using Probabilistic Relaxation. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 17(8):749–764.

Coffman, T.; Greenblatt, S.; and Marcus, S. 2004. Graph-Based Technologies for Intelligence Analysis. *Communications of the ACM, Special Issue on Emerging Technologies for Homeland Security* 47(3):45-47.

Cook, D.J. and Holder, L.B. 1994. Substructure Discovery Using Minimum Description Length and Background Knowledge. *J. Artificial Intelligence Research* 1:231-255.

Djoko, S.; Cook, D.J.; and Holder, L. B. 1997. An Empirical Study of Domain Knowledge and its Benefits to Substructure Discovery. *IEEE Trans. on Knowledge and Data Engineering* 9(4):575-586.

Dorogovtsev, S.N. and Mendes, J.F.F. 2003. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford, UK.: Oxford University Press.

Foggia, P.; Sansone, C.; and Vento, M. 2001. A Performance Comparison of Five Algorithms for Graph

Isomorphism. *Int'l Workshop on Graph-based Representations in Pattern Recognition*, 188-199.

Gallagher, B. 2006. The State of the Art in Graph-Based Pattern Matching. Tech Report UCRL-TR-220300, Lawrence Livermore National Laboratory.

Getoor, L. 2001. Learning Statistical Models from Relational Data," Ph.D. diss., Stanford University.

Gibbons, P.B. and Garofalakis, M. 2001. Approximate Query Processing: Taming the Terabytes! Tutorial in *Int'l Conf. on Very Large Data Bases*.

Giugno, R. and Shasha, D. 2002. Graphgrep: A Fast and Universal Method for Querying Graphs. In *Proc. IEEE Int'l Conf. on Pattern Recognition*, 112-115.

Goethals, B.; Hoekx, E.; and Van den Bussche, J. 2005. Mining Tree Queries in a Graph. In *Proc. ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 61-69.

Greenblatt, S.; Marcus, S.; and Darr, T. 2005. TMODS - Integrated Fusion Dashboard - Applying Fusion of Fusion Systems to Counter-Terrorism. Presented to *Int'l Conf. on Intelligence Analysis*.

Guarino, N.; Masolo, C.; and Vetere, G. 1999. OntoSeek: Content-Based Access to the Web. *IEEE Intelligent Systems* 14(3):70-80.

Ioannidis, Y.E. and Poosala, V. 1995. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *Proc. ACM SIGMOD Int'l Conf. on the Management of Data*, 233-244.

Kälviäinen, H. and Oja, E. 1990. Comparisons of Attributed Graph Matching Algorithms for Computer Vision. In *Proc. Finnish Artificial Intelligence Sypm. (STeP)*, 354-368.

Kim, D.H.; Yun, I.D.; and Lee, S.U. 2004. A Comparative Study on Attributed Relational Graph Matching Algorithms for Perceptual 3-D Shape Descriptor in MPEG-7. In *Proc. ACM Int'l Conf. on Multimedia*, 700-707.

Kuramochi, M. and Karypis, G. 2005. Finding Frequent Patterns in a Large Sparse Graph. *Data Mining and Knowledge Discovery* 11(3):243-271.

Lin, D. 1998. An Information-theoretic Definition of Similarity. In *Proc. Int'l Conf. on Machine Learning*, 296-304.

McKay, B.D. 1990. Nauty User's Guide (Version 1.5). Tech Report TR-CS-9002, Dept. of Comp. Sci., Australian National Univ., Canberra.

Messmer, B.T. and Bunke, H. 1995. Subgraph Isomorphism in Polynomial Time. Tech Report TR-IAM-95-003, Institute of Comp. Sci. and Applied Math, Univ. of Bern.

Meyers, R.; Wilson, R.C.; and Hancock, E.R. 2000. Bayesian Graph Edit Distance. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 22(6):628-635.

Polyzotis, N. and Garofalakis, M. 2002. Statistical Synopses for Graph-structured XML Databases. In *Proc. ACM SIGMOD Int'l Conf. on the Management of Data*, 358-369.

Poole, J. and Campbell, J.A. 1995. A Novel Algorithm for Matching Conceptual and Related Graphs. In *Proc. Int'l Conf. on Conceptual Structures*, 293-307.

Shapiro, L.G. and Haralick, R.M. 1981. Structural Descriptions and Inexact Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 3: 504-519.

Shasha, D.; Wang, J.T.L.; and Giugno, R. 2002. Algorithmics and Applications of Tree and Graph Searching. In *Proc. ACM Symposium on Principles of Database Systems*, 39-52.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Reading, Mass.: Addison-Wesley.

Tsai, W.-H. and Fu, K.-S. 1979. Error-Correcting Isomorphisms of Attributed Relational Graphs for Pattern Analysis. *IEEE Transactions on Systems, Man and Cybernetics* 9:757-768.

Ullmann, J.R. 1976. An Algorithm for Subgraph Isomorphism. *J. ACM* 23(1):31-42.

Umeyama, S. 1988. An Eigendecomposition Approach to Weighted Graph Matching Problems. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 10(5):695-703.

Wang, W.; Jiang, H.; Lu, H.; and Yu, J.X. 2004. Bloom Histogram: Path Selectivity Estimation for XML Data with Updates. In *Proc. Int'l Conf. on Very Large Data Bases*, 240-251.

Washio T., and Motoda, H. 2003. State of the Art of Graph-based Data Mining. *ACM SIGKDD Explorations Special Issue on Multi Relational Data Mining* 5(1):59-68.

Wolverton, M. 2006. Personal communication with author.

Wolverton, M.; Berry, P.; Harrison, I.; Lowrance, J.; Morley, D.; Rodriguez, A.; Ruspini, E.; and Thomere, J. 2003. LAW: A Workbench for Approximate Pattern Matching in Relational Data. In *Proc. Innovative Applications of Artificial Intelligence Conf.*, 143-150.

Wolverton, M.; Harrison, I.; Lowrance, J.; Rodriguez, A.; and Thomere, J. 2006. Software Supported Pattern Development in Intelligence Analysis. Publication 1147, Artificial Intelligence Center, SRI International.

Wörlein, M.; Meinl, T.; Fischer, I.; and Philippsen, M. 2005. A Quantitative Comparison of the Subgraph Miners MoFa, gSpan, FFSM, and Gaston. *Lecture Notes in Computer Science* 3721:392-403.