Efficient Graph Matching

Diego Reforgiato Recupero
University of Catania, Italy

INTRODUCTION

Application domains such as bioinformatics and web technology represent complex objects as graphs where nodes represent basic objects (i.e. atoms, web pages etc.) and edges model relations among them. In biochemical databases proteins are naturally represented as labeled graphs: the nodes are atoms and the edges are chemical links. In computer vision, graphs can be used to represent images at different levels of abstraction. In a low-level representation (Pailloncy, Deruyver, & Jolion, 1999), the nodes of a graph correspond to pixels and the edges to spatial relations between pixels. At higher levels of description (Hong & Huang, 2001), nodes are image regions and edges are spatial relations among them. In a Web graph (Deutsch, Fernandez, Florescu, Levy, & Suciu, 1999) nodes are web pages and edges are links between them. In all these domains, substructure queries that search for all exact or approximate occurrences of a given query graph in the graphs of the database can be useful.

Research efforts in graph searching have taken three directions: the first is to study matching algorithms for particular graph structures (planar graphs, bounded valence graphs and association graphs); the second is to use elaborate tricks to reduce the number of generated matching maps (Cordella, Foggia, Sansone, & Vento, 2004); and the third is, since graph searching problem is NP-complete, to provide polynomial approximate algorithms. In the context of querying in a database of graphs many of the existing methods are designed for specific applications. For example, several querying methods for semi-structured databases have been proposed. In addition, commercial products and academic projects (Kelley 2002) for subgraph searching in biochemical databases are available. These two examples have a different underlying data model (a web-page database is a large graph whereas a biochemical database is a collection of graphs). In these two applications regular path expressions and indexing methods are used during query time to respectively locate substructures in the database and to avoid unnecessary traversals of the database. In general graph databases, there are some searching methods where the data graph and the query graph must have the same size. Other methods allow the query to be considerably smaller than the database graphs. A common idea in the above algorithms is to index the subgraph matches in the database and organize them in suitable data structures.

BACKGROUND

A graph database can be viewed as either a single (large) labeled graph (e.g. web) or a collection of labeled graphs (e.g., chemical molecules). By keygraph searching it is intended graph or subgraph matching in a graph database. Although (sub)graph-to-graph matching algorithms can be used for keygraph searching, efficiency considerations suggest the use of indexing techniques to reduce the search space and the time complexity especially in large databases. Keygraph searching in databases consists of three basic steps:

1. Reduce the search space by filtering. For a database of graphs a filter finds the most relevant graphs; for a single-graph database it identifies the most relevant subgraphs. In this work, the attention is restricted to filtering techniques based on the structure of the labeled graphs (paths, subgraphs). Since searching for subgraph structures is quite difficult, most algorithms choose to locate paths of node labels.
2. Formulate query into simple structures. The query graph can be given directly as a set of nodes and edges or as the intersection of a set of paths. Furthermore the query can contain wildcards (representing nodes or paths) to allow more general searches. This step normally reduces the query graph to a collection of small paths.

3. Match. Matching is implemented by either traditional (sub)graph-to-graph matching techniques, or combining the set of paths resulting from processing the path expressions in the query through the database.

A well-known graph-based data mining algorithm able to mine structural information is Subdue (Ketkar, Holder, Cook, Shah, & Coble, 2005). Subdue has been applied to discovery and search for subgraphs in protein databases, image databases, Chinese character databases, CAD circuit data and software source code. Furthermore, an extension of Subdue, SWSE (Rakhshan, Holder & Cook, 2004), has been applied to hypertext data. Yan et al. (Yan, Yu & Han, 2004) proposed a novel indexing technique for graph databases based on reducing the space dimension with the most representative substructures chosen by mining techniques. However, Subdue based systems accept one labeled graph at a time as input.

GRACE, (Srinivasa, Maier & Mutalikdesai, 2005) is a system where a data manipulation language is proposed for graph database and it is integrated with structural indexes in the DB.

Daylight (James, Weininger & Delany, 2000) is a system used to retrieve substructures in databases of molecules where each molecule is represented by a graph. Daylight uses fingerprinting to find relevant graphs from a database. Each graph is associated with a fixed-size bit vector, called the fingerprint of the graph. The similarity of two graphs is computed by comparing their fingerprints. The search space is filtered by comparing the fingerprint of the query graph with the fingerprint of each graph in the database. Queries can include wildcards. For most queries, the matching is implemented using application-specific techniques. However queries including wildcards may require exhaustive graph traversals. A free and efficient academic emulation of Daylight, called Frowns (Kelley, 2002), uses the compacted hashed paths vector filtering of Daylight and the subgraph matching algorithm VF (Cordella, Foggia, Sansone, & Vento, 2004).

In character recognition, a practical optical character reader is required to deal with both fonts and complex designed fonts; authors in (Omachi, Megawa, & Aso, 2007) proposed a graph matching technique to recognize decorative characters by structural analysis. In image recognition, an improved exact matching method using a genetic algorithm is described in (Auwatanamongkol, 2007).

One way to use a simple theoretical enumeration algorithm to find the occurrences of a query graph Ga in a data graph Gb is to generate all possible maps between the nodes of the two graphs and to check whether each generated map is a match. All the maps can be represented using a state-space representation tree: a node represents a pair of matched nodes; a path from the root down to a leaf represents a map between the two graphs. In (Ullmann, 1976) the author presented an algorithm for an exact subgraph matching based on state space searching with backtracking. Based on the Ullmann's algorithm, Cordella et al. (Cordella, Foggia, Sansone, & Vento, 2004) proposed an efficient algorithm for graph and subgraph matching using more selective feasibility rules to cut the state search space. Foggia et al. (Foggia, Sansone, & Vento, 2001) reported a performance comparison of the above algorithm with other different algorithms in literature. They showed that, so far, no one algorithm is more efficient than all the others on all kinds of input graphs.

MAIN FOCUS

The idea of an efficient method for graph matching is to first represent the label paths and the label edges of each graph of the database as compacted hash functions; then, given the query graph, the paths of the database graphs not matching the query paths will be pruned out in order to speed up the searching process.

The proposed method models the nodes of data graphs as having an identification number (node-id) and a label (node-label). An id-path of length n is a list of $n+1$ node-ids with an edge between any two consecutive nodes. A label-path of length n is a list of $n+1$ node-labels. An id-path of length 1 is called id-edge. The label-paths and the id-paths of the graphs in a database are used to construct the index of the database and to store the data graphs. Figure 1 shows three input graphs with id-paths and id-edges.

Indexing Construction

Let l_p be a fixed positive integer (in practice, 4 is often used). For each graph in the database and for each node, all paths that start at this node and have length from one up to l_p are found. The index is implemented using a hash table. The keys of the hash table are the hash values of the label-paths. Collisions are solved