Subgraph Mining

Ingrid Fischer
Friedrich-Alexander University Erlangen-Nürnberg, Germany

Thorsten Meinl
Friedrich-Alexander University Erlangen-Nürnberg, Germany

INTRODUCTION

The amount of available data is increasing very fast. With this data the desire for data mining is also growing. More and larger databases have to be searched to find interesting (and frequent) elements and connections between them. Most often, the data of interest is very complex. It is common to model complex data with the help of graphs consisting of nodes and edges that often are labeled to store additional information. Applications can be found in very different fields. For example, the two-dimensional structure of molecules often is modeled as graphs having the atoms as nodes and bonds as edges. The same holds for DNA or proteins. Web pages and links between Web pages also can be represented as graph. Other examples are social networks as citation networks and CAD circuits; graphs can be found in a lot of different application areas.

Having a graph database, it is interesting to find common graphs in it, connections between different graphs, and graphs that are subgraphs of a certain number of other graphs. This graph-based data mining has become more and more popular in the last few years. When analyzing molecules, it is interesting to find patterns—called *fragments* in this context—that appear at least in a certain percentage of molecules. Another problem is finding fragments that are frequent in one part of the database but infrequent in the other. This way, this substructure is separating the database into active and inactive molecules (Borgelt & Berthold, 2002). Similar problems occur for protein databases. Here, graph data mining can be used to find structural patterns in the primary, secondary, and tertiary structure of protein categories (Cook & Holder, 2000).

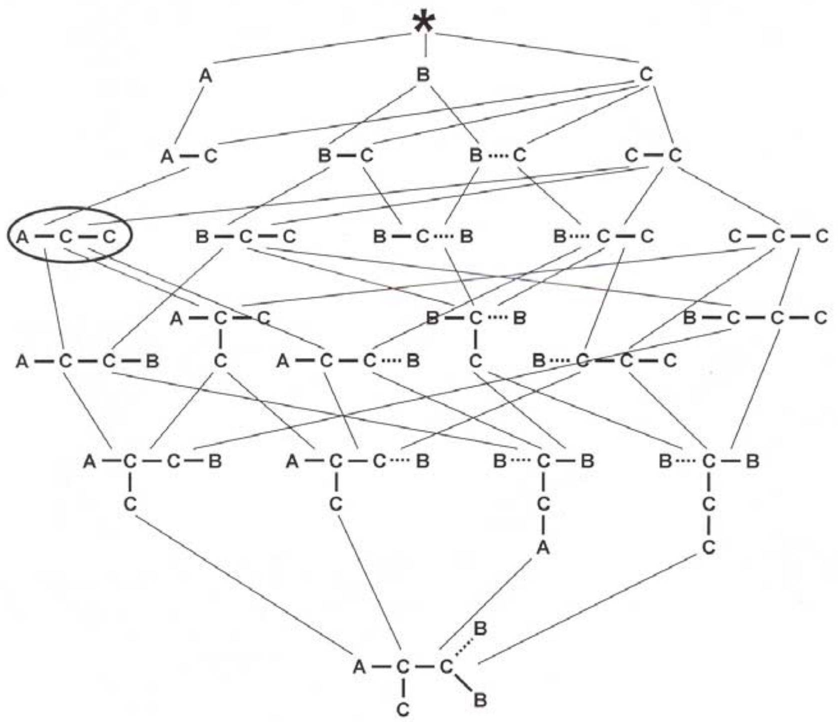
Another application area is Web searches (Cook, Manocha & Holder, 2003). Existing search engines use linear feature matches. Using graphs as underlying data structure, nodes represent pages; documents or document keywords and edges represent links between them. Posing a query as a graph means a smaller graph has to be embedded in the larger one. The graph modeling the data structure can be mined to find similar clusters.

There are a lot of application areas where graph data mining is helpful. Despite the need for graph data mining, the first published algorithm in this area appeared in the mid-1990s. Subdue (Cook & Holder, 2000) is the oldest algorithm but is still used in various applications. Being the first, the number of extensions available for Subdue is enormous. The algorithm is combined with background knowledge, inexact graph matching, and there also is a parallelized variant available. Supervised and unsupervised mining is possible. It took a few more years before more and faster approaches appeared. In Helma, Kramer, and de Raedt (2002), graph databases are mined for simple paths; for a lot of other applications, only trees are of interest (El-Hajj & Zaïane, 2003; Rückert & Kramer, 2004). Also, inductive logic programming (Finn et al., 1998) was applied in this area. At the beginning of the new millennium, finally more and more and everytime faster approaches for general mining of graph databases were developed (Borgelt & Berthold, 2002; Inokuchi, Washio & Motoda, 2003; Kuramochi & Karypis, 2001; Yan & Han, 2002). The latest development, a system named *Gaston* (Nijssen & Kok, 2004) combines mining for paths, trees, and graphs leading to a fast and efficient algorithm.

BACKGROUND

Theoretically, mining in graph databases can be modeled as the search in the lattice of all possible subgraphs. In Figure 1, a small example is shown based on one graph with six nodes labeled A,B,C as shown at the bottom of the figure. All possible subgraphs of this small graph are listed in this figure. At the top of the figure, the empty graph modeled with * is shown. In the next row, all possible subgraphs containing just one node (or zeros edges) are listed. The second row contains subgraphs with one edge. The parent-child relation between the subgraphs (indicated by lines) is the subgraph property. The empty graph can be embedded in every graph containing one node. The graph containing just one node labeled A can be embedded in a one-edge graph containing nodes A and C. Please note that in Figure 1, no graph

Figure 1. The lattice of all subgraphs in a graph



with one edge is given containing nodes labeled A and B. As there is no such subgraph in our running example, the lattice does not contain a graph like this. Only graphs that are real subgraphs are listed in the lattice. In the third row, graphs with two edges are shown, and so on. At the bottom of Figure 1, the complete graph with five edges is given. Each subgraph given in Figure 1 can be embedded in this graph. All graph mining algorithms have in common that they search this subgraph lattice. They are interested in finding a subgraph (or several subgraphs) that can be embedded as often as possible in the graph to be mined. In Figure 1, the circled graph can be embedded twice in the running example.

When mining real-life graph databases, the situation, of course, is much more complex. Not only one, but a lot of graphs are analyzed, leading to a very large lattice. Searching this lattice can be done depth or breadth first. When searching depth first in Figure 1, the first discovered subgraph will be A followed by A-C, A-C-C, and so forth. So, first, all subgraphs containing A, in the next branch all containing B are found. If the lattice is traversed breadth first, all subgraphs in one level of the lattice (i.e., structures that have the same number of edges) are searched before the next level is started. The main disadvantage of breadth first search is the larger memory consumption, because in the middle of the lattice, a large amount of subgraphs has to be stored. With depth-first search, only structures whose amount is proportional to the size of the biggest graph in the database have to be recorded during the search.

Building this lattice of frequent subgraphs involves two main steps: candidate generation, where new subgraphs are created out of smaller ones; and support computation, where the frequency or support of the new subgraphs in the database is determined. Both steps are highly complex, and, thus, various algorithms and techniques have been developed to find frequent subgraphs in finite time with reasonable resource consumptions.

MAIN THRUST

We will now have a more detailed look at the two main steps of the search mentioned previously—candidate generation and support computation. There are two popular ways of creating new subgraphs: merging smaller subgraphs that share a common core (Inokuchi et al., 2002; Kuramochi & Karypis, 2001) or extending subgraphs edge by edge (Borgelt & Berthold, 2002; Yan & Han, 2002).

The merge process can be explained by looking at the subgraph lattice shown in Figure 1. The circled subgraph