Subgraph Mining

Ingrid Fischer
University of Konstanz, Germany

Thorsten Meinl
University of Konstanz, 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 are often labeled to store additional information. Having a graph database, the main goal is to find connections and similarities between its graphs. Based on these connections and similarities, the graphs can be categorized, clustered or changed according to the application area. Regularly occurring patterns in the form of subgraphs —called *fragments* in this context—that appear at least in a certain percentage of graphs, are a common method to analyze graph databases. The actual occurrence of a fragment in a database graph is called *embedding*. Finding the fragments and their embeddings is the goal of subgraph mining described in detail in this chapter.

The first published graph mining algorithm, called Subdue, appeared in the mid-1990s and is still used in different application areas and was extended in several ways. (Cook & Holder, 2000). Subdue is based on a heuristic search and does not find all possible fragments and embeddings. It took a few more years before more and faster approaches appeared. In (Helma, Kramer, & de Raedt, 2002) graph databases are mined for simple paths, for a lot of other applications only trees are of interest (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 every time faster approaches for general mining of graph databases were developed that were able to find all possible fragments. (Borgelt & Berthold, 2002; Yan & Han, 2002; Kuramochi & Karypis, 2001; Nijssen & Kok, 2004).

Several different application areas for graph mining are researched. The most common area is mining molecular databases where the molecules are displayed by their two-dimensional structure. When analyzing molecules it is interesting to find patterns that might explain why a certain set of molecules is useful as a drug against certain diseases (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).

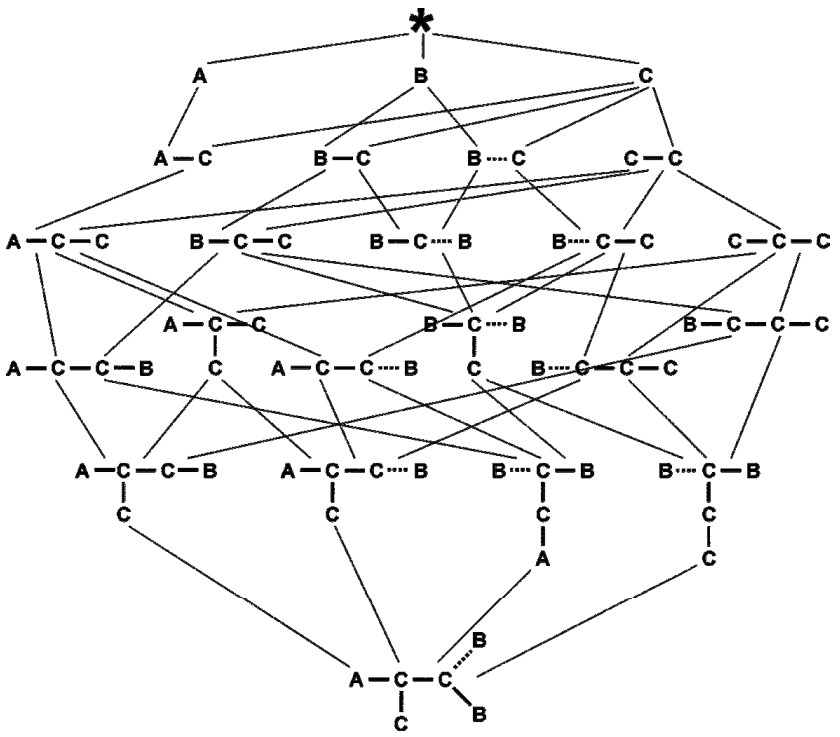
Another application area are web searches (Cook, Manocha, & Holder, 2003). Existing search engines use linear feature matches. Using graphs as underlying data structure, nodes represent pages, documents or document keywords and edges represent links between them. Posing a query as a graph means a smaller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

Quite new is the application of subgraph mining in optimizing code for embedded devices. With the help of so-called procedural abstraction, the size of pre-compiled binaries can be reduced which is often crucial because of the limited storage capacities of embedded systems. There, subgraph mining helps identifying common structures in the program's control flow graph which can then be combined ("abstracted") into a single procedure (Dreweke et.al., 2007).

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

Figure 1. The lattice of all subgraphs in a graph



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 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 appearing 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 is of course 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. Thus, first all subgraphs containing **A**, and 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 which 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.