Graphical Data Mining

Carol J. Romanowski
Rochester Institute of Technology, USA

INTRODUCTION

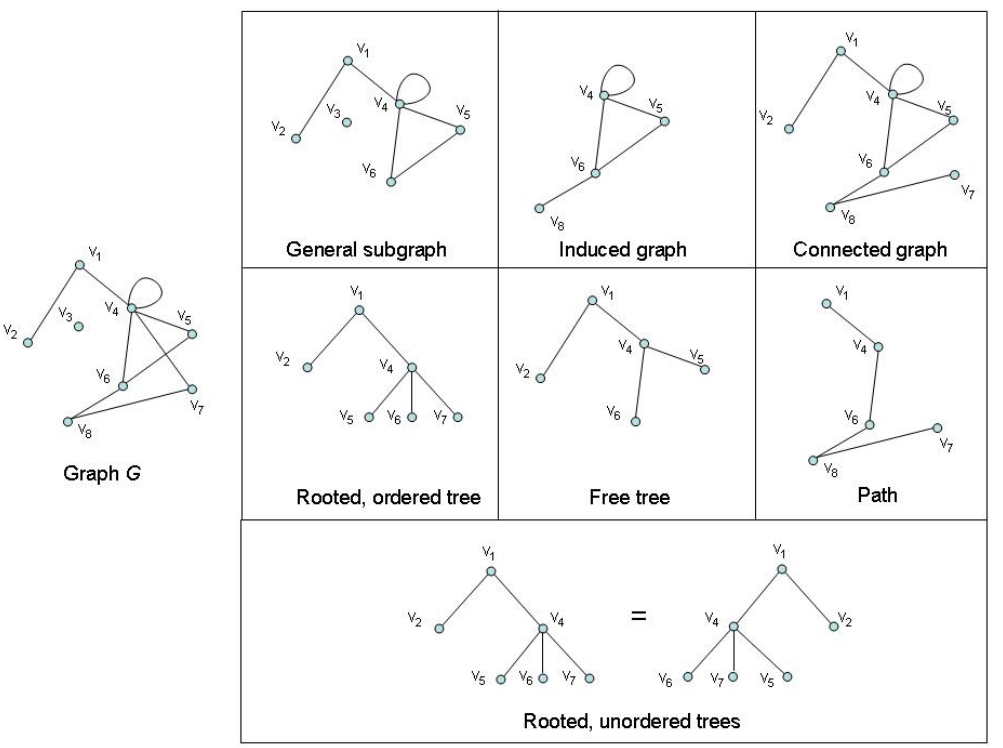
Data mining has grown to include many more data types than the "traditional" flat files with numeric or categorical attributes. Images, text, video, and the internet are now areas of burgeoning data mining research. Graphical data is also an area of interest, since data in many domains—such as engineering design, network intrusion detection, fraud detection, criminology, document analysis, pharmacology, and biochemistry—can be represented in this form.

Graph mining algorithms and methods are fewer and less mature than those designed for numerical or categorical data. In addition, the distinction between graph matching and graph mining is not always clear. In graph mining, we often want to find all possible frequent subgraphs of all possible sizes that occur a specified minimum number of times. That goal involves iteratively matching incrementally larger subgraphs, while classical graph matching is a single search for a static subgraph. Also, graph mining is an unsupervised learning task. Instead of searching for a single match to a specific graph, we are looking for known or unknown graphs embedded in the data.

BACKGROUND

A graph G is a structure that contains a set of vertices V and their incident edges E and comprises many

Figure 1. Graph structures



substructures (see Figure 1). A *general subgraph* is a subset of any vertices and edges in the parent graph. An *induced subgraph* contains a subset of the vertices and all edges between those vertices that exist in the parent graph. A *connected subgraph* is a subset of vertices that are connected with edges; no isolated vertices are allowed. *Trees* are acyclic, directed, branched subgraphs that can be ordered (the order of branch nodes is fixed) or unordered (the order of branch nodes is of no concern). Rooted trees have one vertex with no edges coming into it; all other vertices are reachable from the root, while free trees have no root. A *path* is a subgraph that has no branches.

The most common graph mining task is finding frequent subgraph structures within either a large, single graph or a database of smaller graphs (Washio & Motoda, 2003). Most methods for finding frequent subgraphs are based on the Apriori algorithm (Agrawal & Srikant, 1994) and involve four steps:

1. Starting with a defined smallest unit, generate increasingly larger subgraphs.
2. Check that the generated subgraphs appear in the database or graph.
3. Count the number of times each subgraph appears.
4. Discard subgraphs that are less frequent than the user-specified minimum, or *support*, thus avoiding investigation of their supergraphs. Also discard isomorphisms (subgraphs with identical vertices and edges).

Subgraph matching, or isomorphism testing, is thought to have no polynomial time algorithm for general graphs (Skiena, 1998). Graph mining algorithms attempt to reduce the computational load of this problem in many ways, all of which rely on the downward closure property (see step 4). This property states that a subgraph of a larger graph is more frequent than the larger graph; therefore, if a particular subgraph is infrequent, it is removed from the search space because further expansion would not yield a viable candidate. Another way to reduce search space is to restrict candidates to subgraphs with no common edges (Washio & Motoda, 2003).

MAIN FOCUS

The most basic difference among graph mining algorithms is whether the input data is a single graph or a database of smaller graphs. Algorithms written for a single graph input can be used on a database of smaller graphs, but the reverse is not true (Goethels et al., 2005). Most general graph and tree mining algorithms are frequent subgraph methods focused on a database of graphs, although the single graph approach is more versatile.

Within the main categories of single vs. database of graphs, these frequent subgraph methods are often very similar, and are mostly distinguished by the strategies for generating candidate subgraphs and reducing the support computation cost, by the method of graph matching employed, and by the basic substructure unit. Complete algorithms generate every possible candidate subgraph, while heuristic algorithms truncate the candidate generation stage.

Many of the complete methods such as AGM and AcGM (Inokuchi et al., 2000, 2003), FSG (Kuramochi & Karypis, 2004), GASTON (Nijssen & Kok, 2004) and algorithms proposed by Vanetik et al. (2004) and Inokuchi (2004) use the join method of Apriori to generate new candidate subgraphs. In this method, two graphs of size k with identical subgraphs of size k-1 are joined together to form a graph of size k+1. For example, consider two connected graphs, each with k = 3 nodes. Graph A contains nodes {B, E, and K} while Graph B contains nodes {C, E, and K}. The new candidate graph would contain nodes {B, C, E, and K} and their incident edges.

Each of these algorithms uses a different substructure unit (either a vertex, edge, or combination of vertex and edge called a *leg*) to generate larger subgraphs. All take a database of graphs as input. GASTON is particularly notable because it can find frequent paths and free trees in addition to general subgraphs.

Some complete algorithms use pattern extension instead of join operations to generate candidate subgraphs. In this method, the new candidate graph is formed by extending a vertex or edge. Examples of these non-Apriori methods include gSpan (Yan et al., 2002) and its variants, CloseGraph (Yan et al., 2003) LCGMiner (Xu & Lei, 2004), ADI-Mine (Wang et al., 2004), and GraphMiner (Wang et al., 2005). Basically, these algorithms build spanning trees for each