In the previous chapter, we have described a method to find spatio-temporal tree patterns. A more general representation for patterns with arbitrary relationships is the graph model.

Data mining in graph databases has received much attention. We have witnessed many algorithms proposed for mining frequent graphs. Inokuchi, Washio, and Nishimura (2002) and Karpis and Kumar (1998) introduce the Apriori-like algorithms, AGM and FSG, to mine the complete set of frequent graphs. However, both algorithms are not scalable as they require multiple scans of databases and tend to generate many candidates during the mining process. Subsequently, Yan and Han (2002) and Nijssen and Kok (2004) propose depth-first graph mining approaches called gSpan and Gaston, respectively. These approaches are essentially memory-based and their efficiencies decrease dramatically if the graph database is too large to fit into the main memory.

Recognizing this problem, Wang, Wang, and Pei (2004) focus on the problem of mining graphs on large, disk-based databases. An effective index structure
ADI is proposed to facilitate major graph mining operations. However, this solution does not work well when the graph database is still evolving. This is because the ADI structure has to be rebuilt each time the graph database is updated.

In short, while previous studies have made excellent progress in mining graph databases, many of them assume that the graphs in the databases are relatively static and simple, that is, the number of possible labels in the graphs is small. They do not scale well for mining graphs in a dynamic environment. For example, a spatio-temporal database can contain millions of different structures and the number of different labels in the graphs is easily in the range of hundreds. Changes to spatio-temporal databases cause changes to the graph structures that model the relationships in the spatio-temporal data. Re-execution of the mining algorithm each time the graphs are updated is costly, and may result in an explosion in the demand for computational and I/O resources. Consequently, there is an urgent need to find an algorithm that is scalable and can incrementally mine from only those parts of the graph databases that have been changed.

We design a partition-based approach to graph mining. Our idea is to isolate update changes to a small set of sub-graphs and re-execute the graph mining algorithm only on the isolated sub-graphs. Instead of finding frequent graph patterns on large and complex graphs, we recursively partition complex graphs into smaller, more manageable sub-graphs until these sub-graphs can fit into the main memory. With this, existing memory-based graph mining algorithms can be utilized to discover frequent patterns in the sub-graphs. The discovered patterns are then joined via a merge-join operation to recover the final set of frequent patterns that exist in the original complex graphs.

In this chapter, we design a partition-based algorithm to divide graphs into $k$ smaller sub-graphs ($k$ is determined by the size of the main memory) with the goal of reducing connectivity among sub-graphs while localizing most, if not all, the updated nodes to a minimal number of sub-graphs. Once divided, we can utilize existing efficient memory-based graph mining algorithms to discover frequent patterns in these sub-graphs. We develop a merge-join operation to losslessly recover the complete set of frequent sub-graphs in the database from the set of sub-graphs found in the partitions. We also give a theoretical proof to ensure that mining of frequent sub-graphs in the partitions will be equivalent to mining in the original graph database. In the following, we present the details of our partition-based graph mining algorithm, called PartMiner. Here, we make use of the cumulative information obtained during