Frequent subgraph mining is an important problem in data mining with wide application in science. For instance, graphs can be used to represent structural relationships in problems related to network topology, chemical compound, protein structures, and so on. Searching for patterns from graph databases is difficult since graph-related operations generally have higher time complexity than equivalent operations on frequent itemsets. Even though there is a significant body of work on graph mining, most techniques work outside the database system. Programming frequent graph mining in SQL is more difficult than traditional approaches because the graph must be represented as a table and algorithmic steps must be written as relational queries. In our research, we study three fundamental problems under a database approach: graph storage and indexing, frequent subgraph search, and identifying subgraph isomorphism. We outline main research issues and our solution towards solving them. We also present preliminary experimental validation focusing on query optimizations and time complexity.

Recent subgraph mining algorithms can be roughly classified into two categories: depth-first search and level-wise search. Algorithms in the first category use a depth-first search for finding candidate frequent subgraphs. This algorithm builds a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. In the second category the algorithm uses a level-wise search scheme like Apriori to enumerate the recurring sub-graphs. This algorithm finds all frequent induced subgraphs using an Apriori-based approach which extends subgraphs by adding one vertex at each step.

In this project, we propose an efficient algorithm to generate frequent subgraphs inside a relational DBMS. We implement all the graph mining steps with SQL. The representation of graphs using indices could be applied to most graphs. We also implement the canonical ordering technique to identify similar subgraphs with different order of vertex and edge labels. Our algorithm achieves good performance and is scalable to large graph datasets.

Frequent Subgraph Search

In order to count the isomorphic subgraphs across graphs, we need to systematically generate subgraphs of increasing size in all of the input graphs and count them. In our algorithm, the subgraphs are allowed to expand on any matching vertex(belong to the same graph) along the edges that do not exist in the subgraphs that are being expanded. This unconstrained expansion can generate all the possible subgraphs. For example, to expand a one-edge subgraph into a two-edge subgraph we join the Subgraph 1 table with itself on matching vertices. To make sure that the expansion is done within the same graph we use the constraint that of GID of both one-edge subgraphs should be the same. In general, subgraphs of size i are generated by joining the Subgraph (i - 1) table with the base table Subgraph 1. Since the join operation is performed on the vertex label and edge label attributes, we also need to set vertex label attributes VL1, . . . , VLI, and edge label attributes EL1, . . . , ELi + 1) to be the primary key of the Subgraph i table.

A graph may have many subgraphs of the same substructure. If we count the frequency of the subgraph from the candidate table, it will be counted many times. Hence, in order to obtain the correct frequency of a subgraph in the graph dataset, we need to include one instance per subgraph within one graph. For this purpose, we project distinct vertex labels, edge labels, connectivity map, and GID and store them in the Distinct i table. Then we perform a GROUP BY operation on the vertex labels, edge labels, and connectivity map in Distinct i to obtain the correct frequency of each subgraph. Since the subgraphs with less frequency than provided support value will not contribute to future generation, we prune the Subgraph i table by removing those subgraphs.

Conclusion and Future Work

In this project we present an efficient algorithm to generate frequent subgraphs with SQL. We extend the graph representation of DB-SUBDUE by adding the GID attribute to accommodate the mining algorithm to a set of graphs. We also create some necessary indices in the Edge, Vertex, and Subgraph tables to improve the speed of the join operation which is necessary in the frequent subgraph search process. In the frequent subgraph search step we implement unconstrained expansion to generate subgraphs of increasing size. This expansion method, which expand subgraphs on any matching vertex along the edges that does not exist in the subgraphs, allows our algorithm to generate all the possible subgraphs. Although unconstrained expansion is efficient at generating all subgraphs, it allows the similar subgraphs to grow in different order, making the frequency counting incorrect. In order to solve this problem, we introduced the canonical ordering technique in the subgraph isomorphism step. With the canonical ordering technique, we reorder all the subgraphs based on their vertex labels, edge labels, and connectivity attributes. After the reordering process, the similar subgraphs growing in different way will have the same order of vertex labels, edge labels, and connectivity attributes which help to identify the subgraph isomorphism.