Finding common structures in data using graph Mining

Akshata Patil
Student at RIT

Advisor: Carlos Rivero
Professor at RIT

Abstract

Graphs are helpful for representation of complicated structures such as Protein structures, images, biological networks, social networks, Web. Graph mining is the process of mining information or discovering knowledge from such complex structures. The basic idea is that data can be represented in graph forms and common patterns can be observed in such sets of data that can then be analyzed and used in various applications. In this paper, I have given a brief description of our research conducted on different graph mining techniques for a specific application of finding semantically similar codes. Based on this research I could finalize on certain techniques and specific algorithms implementing those techniques. I will see how NeMa works and how I have modified certain existing implementations to get better results with NeMa.

1. Introduction

Mining is the process of knowledge discovery from data. Graph mining restricts the scope of data to only structured data or data that can be represented in graphical form. Although graph mining is not completely new, the research in this field is relatively new. Most of this research is concentrated in fields of chemical compounds, social and biological networks and has been an effective mining technique in those streams. In this paper, I am aiming to apply these methods in the computer domain. To be more specific I shall explore data mining to find an algorithm that can group similar computer codes.

Single graph mining technique can be implemented in numerous ways depending upon the approach or the application for which the technique is being used, giving rise to a large number of algorithms. Some of the popular graph mining techniques are listed in the table below.

Table 1. Popular Graph Mining Techniques [14]

<table>
<thead>
<tr>
<th>Frequent Subgraph Mining</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlated graph pattern mining</td>
</tr>
<tr>
<td>Optimal graph mining</td>
</tr>
<tr>
<td>Approximate graph pattern mining</td>
</tr>
</tbody>
</table>
To apply graph mining techniques to the problem of finding semantically similar code, each of the techniques needs to be studied thoroughly to determine its applicability. In this paper, we shall see some of the popular graph mining techniques in section 3. Section 2 explains the use case or the problem. The most suitable graph mining techniques for our application and the most relevant algorithms, their comparisons will be detailed in section 4. In sections 5 and 6 I shall discuss the modifications to the selected algorithms to fit our application and the tests with their corresponding results. In the paper, I have referred to the large which is our basis of comparison while finding similar graphs as the data graph and all the graphs that I am trying to compare with this data graph as the query graphs.

2. Use Case

```
public static void main(String[] args) {
    int i = 0;
    while(i < 10) {
        print(i);
        i++;
    }
}

public void print(int i) {
    System.out.println(i);
}
```

*Figure 1. Code for Data Graph*  
*Figure 2. Code for Query graph*

The purpose of this paper is to find a graph mining technique that will match semantically similar graphs. Consider the simple code snippet in fig. 1. I have refactored the code to get a semantically similar code in fig. 2. The codes in figures 1 and 2 are converted into graphs depicted in figures 3 and 4 respectively.
Our aim is to calculate the similarity between the two graphs and determine if they can be clustered together.

3. Graph Mining Techniques

This section entails different graph mining techniques studied and the corresponding papers referred for analyzing different algorithms. As this was a joint study with different colleagues, another colleague studied the remaining techniques from the techniques listed above.

3.1. Correlated graph pattern mining

- Correlated graph pattern mining deals with finding similarity between different graphs. There are number of ways of calculating correlation between graphs.
- The correlation pattern mining can be applied to many graph-mining techniques such as spectral clustering, frequent hyper clique pattern mining, graph classification, biological network mining, social network pattern mining, structural leap search. Basically any mining pattern that mines graph patterns based on the relativeness of the nodes can be classified as a correlated graph mining technique.
- Research papers:
  - Mining Correlated Subgraphs in Graph Databases
  - Structural Correlation Pattern Mining for Large Graphs

3.2. Optimal graph pattern mining

- Most of the graph mining techniques require obtaining all possible patterns from given graph. However, in real world applications such as social networks or even biological networks finding all enumerations is
infeasible. Therefore, a need of an optimal graph pattern mining solution has arisen owing to the scalability issues [1].

- Research paper:
  - Mining Significant Graph Patterns by Leap Search

3.3. Graph kernels
- Graph kernels are used for comparing chemical compounds by directly comparing the graphical forms of their molecules. Many graph kernels have been developed.
- Research paper:
  - Cyclic pattern kernels for predictive graph mining

3.4. Link mining
- Link mining is a graph mining technique that gives additional importance to the relation between data.
- Its applications include web page classification where each web page is a node and the web links to other pages is the link which can be used for classifying the web page based on its link relations.
- Research paper:
  - Link Mining: A New Data Mining Challenge
  - A link mining algorithm for earnings forecast and trading

3.5. Web structure mining
- It deals with the hyperlink structure of the web thus link mining discussed earlier is a type of web structure mining.
- Research paper:
  - Web Structure Mining: An Introduction

3.6. Work-flow mining
- It is like the concept of reverse engineering. In this technique, the process models are constructed from the event logs.
- Research paper:
  - Workflow mining: discovering process models from event logs
  - Workflow mining: A survey of issues and approaches

3.7. Biological network mining
- Biological networks are dense networks of modules comparable to social networks in terms of complexity. One type of biological network is modular network. Networks that can be naturally decomposed into smaller components with similar properties [1].
- Research paper:
  - GraphWeb: mining heterogeneous biological networks for gene modules with functional significance.
4. Selected Graph Mining Techniques

Thorough research was conducted on the graph mining techniques listed in Table 1 to find a technique most optimistic with respect to the use case defined in section 2. Since I wanted to find similar graphs many graph mining techniques became non-applicable, as they matched exact graphs or sub graphs. Also, most of the research in the Graph mining field has been concentrated on finding protein compounds or chemical structures, that made it very difficult to apply these techniques directly to code graphs. In case of chemical compounds the labels of each node in the query graph should match a node in the data graph i.e. I need exact topologies as well as node labels. However, in our case I want to relax on both these properties, and therefore, approximate graph mining and graph clustering were selected for further research.

I had also considered graph kernels, however due to limited research in the field I were not able to find an algorithm that could fit our criteria. Marginalized Kernels Between Labeled Graphs was one of the papers that was promising. However, the random factor in terms of path discovery was not favorable as compared to the other techniques that I selected.

4.1. Graph Clustering

Graph clustering is comparable to data clustering wherein it matches vertices to form groups of similar graphs or vertices. There are two main types of graph clustering algorithms:

4.1.1. Node Clustering: Nodes from one graph are compared to form clusters of similar vertices. These algorithms are beneficial in studying different atoms and their concentrations in complex compounds. This technique is for analyzing the graph itself for its properties such as similar nodes or edge paths.

4.1.2. Graph Clustering: This algorithm clusters graphs together based on their similarity. The basis of similarity can vary according to the requirements such as vertex similarity, edge path similarities, combinations of the two. This technique best matches our requirements. However, internally it uses approximate graph matching or frequent pattern matching algorithms. One can use any of the graph mining techniques for finding similar or matching graphs and then grouping them together.

4.2. Approximate Graph Mining Technique

As the name suggests this technique matches graphs that are approximately same or in other words similar or not exact. Research on this technique has been concentrated to deal with biological and social network issues. This technique helps take care of errors or missing data in case of large and complex data sets which have always been problems in these cases. It calculates the similarity between the graphs based on their topologies and node labels and graphs with
similarity measure above a threshold limit are considered a match in these cases. It therefore solves our problem of non-exact match, if I can determine a threshold that gives minimum errors.

Following two algorithms were studied for implementation purposes. Both these algorithms have been developed for analyzing network topologies.

4.2.1. GRAAL (GRaph ALigner) [15] [16]: It is used to align networks. It calculates similarity between two graphs by comparing and aligning every vertex of the query graph with that of the data graph. The seed is selected by finding the best matching node i.e. the one with minimum cost and then the entire graph alignment process takes place. It repeatedly finds a new seed and realigns the graph. It calculates and associates a cost with every node of query graph with that of the data graph. This cost reflects the cost of making additions or subtractions on a node of query graph to match it exactly with a corresponding data graph. These costs consider the node signatures as well as the neighborhood similarity. Neighborhood similarities concentrates on the different size graphlets it touches, thus it’s time complexity goes very high.

4.2.2. NeMa (Network Matching): It is very similar to GRAAL. However, it eliminates the additional calculation of number of different sized graphlets each node touches, thus increasing the speed. It has been explained in detail in the next section of the paper.

5. Contributions

There are two major sections to this project after research. First is to convert code into correct graph format and second an algorithm that performs the matching of these graphs. I have worked with existing implementations for both requirements and tried to make a few additions to the existing system, which are both in Java. I will first discuss the existing system followed by the additions.

5.1. Code to convert program code to graphs:
One of the most crucial steps of this project is to obtain graphs from code. My advisor had an implementation ready for this purpose. The current program converts java and Python code to graph forms. When you give it for example a java file consisting of a class containing two methods, it would generate two graphs one for each method. For example, consider the code in Fig. 1. Will be converted into two graphs as shown in figures 5 and 6 below:
5.2. NeMa

It is a network matching algorithm that is used to match network topologies. This algorithm associates cost to every node of the query graph to match a node in the data graph. This cost is a combination of label similarity and neighborhood. Contributions factor of both these costs towards the final cost of the vertex can be configured as per requirement. For our implementation, I have given both equal importance. I have referred NeMa: Fast Graph Search with Label Similarity [17] for understanding NeMa. In the implementation label matching has been achieved using Levenshtein distance. For matching the neighborhood, neighbors up to certain number of hops are considered. The number of hops can be specified. Neighbors of say vertex $u$ of query graph are compared with vertex $u'$ of the data graph. The cost of all nodes of the query graph is then summed up to find the total cost of the entire graph. NeMa does return more than one solution, first being the best. I can specify that number, for our program I have fixed this number to 5.

6. Work Done

Most of our work in this project was concentrated on modifying the code that converts programs to graphs. The reason being that NeMa can match nodes if there are matching topologies, so it becomes very important to generate graphs that give optimal solutions with NeMa. The code has been updated to function without throwing exceptions for programs with static or class variables in java. As mentioned in section 5.1 the code generated different graphs for every method in the program. I have managed to generate a single graph for the entire program. For example consider the example in section 5.1. Now the same program (Fig 1) generates the graph shown in figure 7.
Before getting to this output I had reached an intermediate step, wherein I created a separate node for method declaration and connected that node to every node in that method and the calling vertex. Figure 8 shows the resulting graph for the code in figure 1 using this intermediate method.

Both these codes modify the output given by the Java8EPDGParser parser class. Our initial attempt was to modify the parser and add the class details in it. However, the calls to the class start method enterClassInstanceCreationExpression() and exitClassInstanceCreationExpression() in the class Java8BaseListener have been aborted. Due to the time limit I decided to work with the solution in the class CombineResult. However, this increases the time complexity of the program. I have also modified the output of the program to match the input of the program.

There are still several limitations to this program. It does not support polymorphism. If there are two classes in a program having methods of same name, I will get a wrong output. Also, the program does not map non-void method calls. This can be fixed by accessing the reading variable when the result of called method is assigned to a variable. However, I could not reach to a solution for connecting calls nested in a call.

7. Results

I tried to experiment with different types of program, to see how NeMa performs. Most of the verification was manual. NeMa does return a cost of 0.0 in case of exact match, but returns 0.0 even when there is no match. However, it does give the vertex mapping where none of the vertices of the query graph have a corresponding vertex in the data graph. When I entered programs in Figures 1 and 2 I got the following result “Solution: \{0=-1, 1=3, 2=4, 3=5, 4=6\} 0.0”.

Let’s take a little more complex example. Consider the codes in figures 9. Both the codes give fibonacci series using iteration. When I tried to compare graphs generated using previous code the costs were as follows:

Cost for main() method = 0.034
Cost for fibonacci2() method = 1.98.
Now I tried to compare the graph generated using the new graph generation method. The cost associated was 0.0186, with 95% vertices mapped. However, I found that NeMa does not map nodes correctly. For instance, in this example, it does not differentiate between the assignments fibo_num =1 and fibo = 1. From program 2 when I give the starting point of Prog2 as node 7 (if(number == 1 || number == 2) to match the node 1 of Prog_2, it matches all the following assignments in prog2 to the node 3 (int fibo = 1;) of Prog_2. It repeatedly showed this behavior in many such sample tests. Also in cases where there were complex methods and I provide a seed vertex to align with the initial vertex of the data graph, it aligns all the proceeding vertices efficiently, however, the preceding mappings are not correct. The output does not change even with hop count.
Since with increase in hop count the neighborhood matching increases, the matching should improve. However, since NeMa is matches multiple nodes to a single node, the room for improvement of solution is lost and therefore, if this feature can be eliminated, we might be able to get better solutions.

8. Future Work

There are certain limitations to the type of code that can be converted to the graphs. The code can be modified to work with all types of Java and Python features. NeMa can be modified to eliminate its ability to map many vertices of the query graph to a single vertex of the data graph. The big picture is to be able to find similar graphs from a large set of graphs. Graph clustering can be explored to achieve this. The aim is to form different groups of similar codes. For example, consider figure 10, The first cluster has graphs similar to the depth of 2 considering A as the seed point for each of them. However, when I increase the depth to 3 I get two different clusters as the nodes after 2 hops change.
9. Conclusion

We have observed that NeMa gives good results for matching well-formed and similar graphs. Thus, we have successfully found a graph mining technique for finding common data structures. However, we have seen that it does not work well with complex programs and the program to convert code in to graphs would need revising. With the research, modifications and testing of the existing code, the goal of finding semantically similar codes in a large graph database seems to be achievable.

Acknowledgment

I would like to thank my advisor Prof. Carlos Rivero for his guidance and support throughout the project and Prof. Matthew Fluet for his immense efforts in making sure that we were on track with our milestones, presentation skills and the required paperwork. A very sincere thanks to Rochester Institute of Technology for all the resources and opportunities made available to us and their Department of Computer Science (C.S) for the guidance and facilities.

References

doi:10.1109/ICDE.2005.86
doi:10.1007/s10618-008-0124-z
doi:10.1109/TKDE.2004.47
doi:10.1016/S0169-023X(03)00066-1
doi:10.1093/nar/gkn230