École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Computer Aided Decision Support
International Research Master 2
2013 - 2014

Pattern recognition project draft

Multi-scale graph comparison

Supervisors
Romain RAVEREAUX
romain.raveaux@univ-tours.fr

Student
Thomas NOGUER
thomas.noguer@etu.univ-tours.fr

Université François-Rabelais, Tours

M2RI CADS 2013 - 2014

Version of January 7, 2014

Formalization of the problem

When we wish to do the comparison of two graphs, we can come to a limitation when handling high sized graphs. The community detection algorithms allow to simplify a graph by finding communities of highly related nodes. It is then legitimate to want to use such algorithm to reduce the size of graphs in order to perform calculations that are greedy for computation time.

For this project we choose to use a community detection algorithm for graph comparison. The Louvain's method is very fast and easy to implement. The graph edit distance method is known to be very slow when dealing with high scaled graphs. In this project we want to use the community detection to simplify the graphs to be used in the graph edit distance so we can still compare them with a good computation time.

There is still an issue to solve, how do we use one method with the other. The Louvain's method can be used in several iterations to make the graph more and more simple. We must find a relation between the edit distance between two graphs at different scales and the edit distance between the unchanged graphs. At which scale do we decide that the edit distance is close enough to our original graph? Can we use the different scales together in order to be closer to the distance between the origal graphs? These questions are at the center of our problem.

Graph Edit Distance

We use the following algorithm in order to find the edit distance between two graphs $G1$ and $G2$:

Algorithm 1 Graph Edit Distance

repeat
  for each node $n \in G1$ **do**
    Add every possible transformation n into the search tree.
    Select a path using A^* and heuristics.
  end for
  for each remaining node n of $G2$ **do**
    Add every needed insertion of existing node into the search tree.
    Select a path using A^* and heuristics.
  end for
until The path is complete
return The cost of the found path.

Community detection: Louvain's method

This method is based on two formulas: the modularity Q and the composite modularity gain ΔQ. The modularity of a graph is found following this formula:

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j),$$

Where m is the sum of the weights of all the links in the graph ($m = \frac{1}{2}\sum_{ij} A_{ij}$), A_{ij} the weight of the edge between i and j, k_i is the sum of the weights of the edges attached to vertex i, c_i is the community to which vertex i is assigned and finally δ is a function where $\delta(u, v)$ is 1 if $u = v$ and 0 otherwise.

The gain in modularity from moving a node i into a community C is defined by the following formula:

$$\Delta Q = \left[\frac{\sum_{in} + k_{i,in}}{2m} - \left(\frac{\sum_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

Where $\sum_{in}$ is the sum of the weights of the links inside the community C, $\sum_{tot}$ is the sum of the weights of the links incident to nodes in C and $k_{i,in}$ is the sum of the weights of the links from i to nodes in C.

We use the following algorithm in order to find the communities of our graphs:

Algorithm 2 Louvain's method

```
Assign each node to a unique community.
Compute the initial modularity.
repeat
    for i ∈ V do
        for j ∈ V do
            Remove i from its community and place it into j's.
            Compute the composite modularity gain ΔQ.
        end for
        if There exists a positive gain then
            Choose j with the maximum gain and truly move i to j's community.
        else
            i stays in its community.
        end if
    end for
until No further improvement in modularity
```

Multi-scale graph comparision

We want to compare two graphs G1 and G2. We first use the Louvain's method on G1 until the graph isn't simplified (it is a unique node), we have stored the communities at each scale of the nodes. We then apply the GED algorithm between the most simplified version of G1 and G2, only when we will evaluate the cost of deletion/substitution of a community (i.e. a node that isn't a node in the original graph G1) we will use the GED of the subgraph of this community and G2.

In other words, the algorithm will compare every community (which are sub-graphs) with G2. We hope that this approach will reduce the computation time of the total distance.

The algorithm is the same as the Graph Edit Distance, this only difference is the cost function which is $GED(GetSubGraph(c), G2)$ where c is a community and the function $GetSubGraph(c)$ returns the sub-graph of the community c.

Multi-scale graph comparison

Computer Aided Decision Support
International Research Master 2
2013 - 2014

Pattern recognition project draft

Abstract: abstract

Keywords: keywords

Supervisors
Romain RAVEREAUX
romain.raveaux@univ-tours.fr

Student
Thomas NOGUER
thomas.noguer@etu.univ-tours.fr

Université François-Rabelais, Tours