Date: 16th December, 2016

OXFORD

Project Report

Visualizing equivalence classes and mapping ambiguity graph

Arun Rajan Arunesh Pandey FNU Gaurav Hemant Pandey

CS Department, Stony Brook University
Stony Brook, New York 11790, USA

Abstract

The project supplements the RapClust pipeline in a way to visualize fragmented equivalence classes using a web GUI interface. It includes incorporating optimizations to process hundred thousands of nodes and display it using multi-level clustering.

1 Introduction

In this paper, we rely on the provided data set of fragment equivalence classes. We define an equivalence relation over fragments, based on the set of transcripts to which they map. The set of fragments related under this definition constitutes a fragmented equivalence class. Let $M(f_i)$ be the set of transcripts to which fragment f_i maps, and let $M(f_j)$ be the set of transcripts to which fragment f_j maps. We say that $f_i f_j$ if and only if $M(f_i) = M(f_j)$. Consequently, a fragment equivalence class is a set of fragments such that, for every pair f_i and f_j in the class, f_i âˆ¼ f_j. An equivalence class can be uniquely labeled based on the set of transcripts to which the fragments contained in this class map. It is important to remember that, though the label consists of transcript names, the equivalence relation itself is defined over sequenced fragments and not transcripts. Finally, in addition to a label, we denote the count of each equivalence class C_i by count(C_i); this is simply the number. The details of the equivalence class data set will be provided later in the report. Now, given these fragment equivalence classes, we have used the sigma library of Java-script to display them on a web browser. Since there is a huge number of classes, the graph will comprise of a large number of nodes and thus we have used Marcov Clustering algorithm to display the nodes in a clustered manner. The MCL algorithm is short for the Markov Cluster Algorithm, a fast and scalable unsupervised cluster algorithm for graphs (also known as networks) based on simulation of (stochastic) flow in graphs. The granularity of the MCL algorithm has been set to level 4 with -i flag.

2 Dataset

The provided dataset is of the form: 1- No of transcripts
2- No of Eq classes
3- Txp1 Name
4- Txp2 Name
..
..
..
<No. of transcripts in given label> <Transcript Ids> <Fragment Count>

The file begins with a line that contains the number of transcripts (say N) then a line that contains the number of equivalence classes (say M). It is then followed by N lines that list the transcript names — the order here is important, because the labels of the equivalence classes are given in terms of the IDs of the transcripts. The rank of a transcript in this list is the ID with which it will be labeled when it appears in the label of an equivalence class. Finally, the file contains M lines, each of which describes an equivalence class of fragments. The first entry in this line is the number of transcripts in the label of this equivalence class (the number of different transcripts to which fragments in this class map — call this k). The line then contains the k transcript IDs. Finally, the line contains the count of fragments in this equivalence class (how many fragments mapped to these transcripts).

3 Implementation

3.1 Node JS

Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications. For graphical display Sigma library has been used. Sigma is a JavaScript library dedicated to graphical drawing and the purposes where nodes are interconnected on a higher level. It makes easy to publish interconnected networks on Web pages, and allows developers to integrate network exploration in Web applications.

3.2 MCL Algorithm

We have used the MCL clustering tool for our clustering purposes. The algorithm has been developed by Dr. Stijn van Dongen.

The following is an example of label input-

```
    —8<——8<——8<——8<——8<—
cat hat 0.2
hat bat 0.16
bat cat 1.0
bat bit 0.125
bit fit 0.25
fit hit 0.5
hit bit 0.16
—>8——>8——>8——>8——>8—
```

It can be clustered like this:

```
mcl cathat –abc -o out.cathat
```

The file out.cathat should now like like this

```
—8<——8<——8<——8<——8<—
cat hat bat
bit fit hit
—>8——>8——>8——>8——>8—
```

A few things to note. First, MCL will symmetrize any arrow it finds. If it sees bat cat 1.0 it will act as if it also saw cat bat 1.0. You can explicitly specify cat bat 1.0, mcl will in the first parse stage simply end up with duplicate entries. Second, MCL deduplicates repeated edges by taking the one with the maximum value. So,

```
—8<——8<——8<——8<——8<—
cat hat 0.2
hat cat 0.16
hat cat 0.8
—>8——>8——>8——>8——>8—
```

Will result in two arrows cat-hat and hat-cat both with value 0.8.

Now, we were given the equivalence classes and the information about how many transcripts each of these comprises of. First of all, the raw data file was processed in order to extract information about edges between two equivalence classes.

Each edge between two equivalence classes was given weight by the following metric:

$$Weight = \frac{E_1 \cap E_2}{E_1 \cup E_2}$$

Once we have the equivalence classes and the weights between them, like shown in the sample above, we cluster them on the basis of weights. This is first level clustering. To further process the huge data set having thousands of clusters, we apply multi level clustering. We extract information about the number of equivalence classes present in each cluster and use it to group them in weight-clusters.

3.3 json-server

We are storing the processed information in lowdb. The web application communicates with it via GET/POST HTTP requests.

- L1 Cluster- cluster of same size L2 clusters
- L2 Cluster- cluster of L3 graphs created by MCL
- L3 Node Graph- Node graph created from raw data

Our web application uses AngularJS, Bower package manager, GruntJS, SASS and SigmaJS.

4 Screenshots

Following are the snippets from our code:

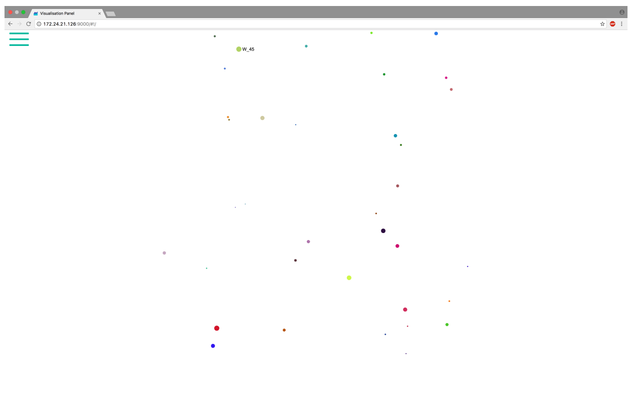

Fig. 1. top level cluster

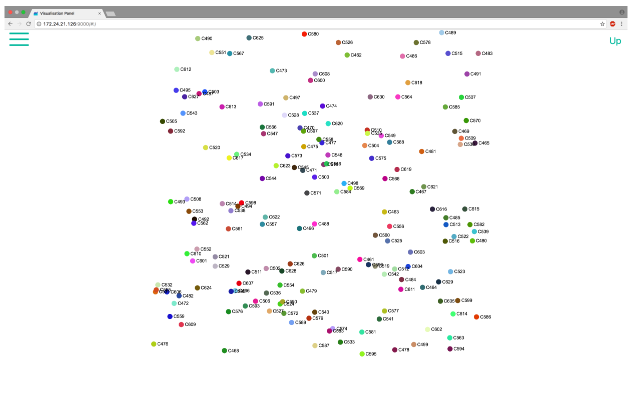

Fig. 2. mid level cluster

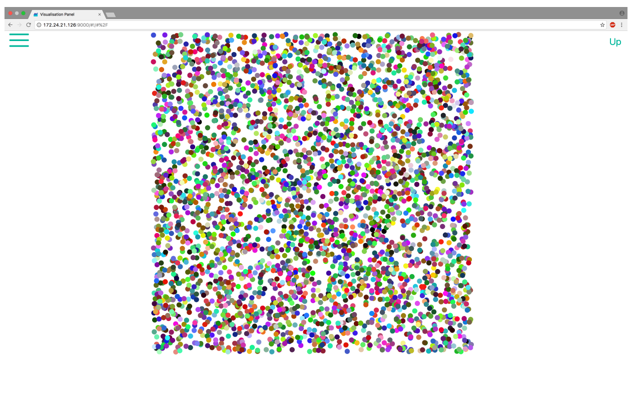

Fig. 3. zoomed out mid level cluster

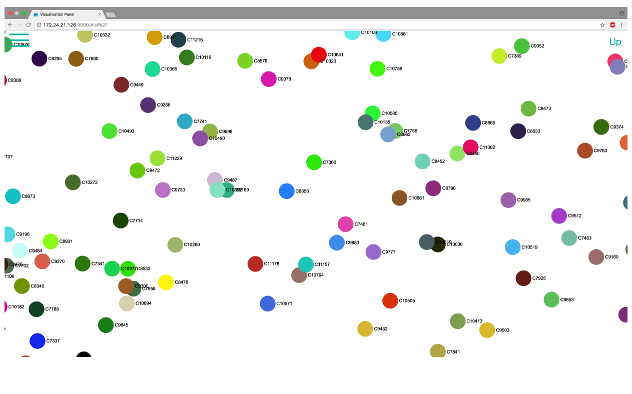

Fig. 4. zoomed in mid level cluster

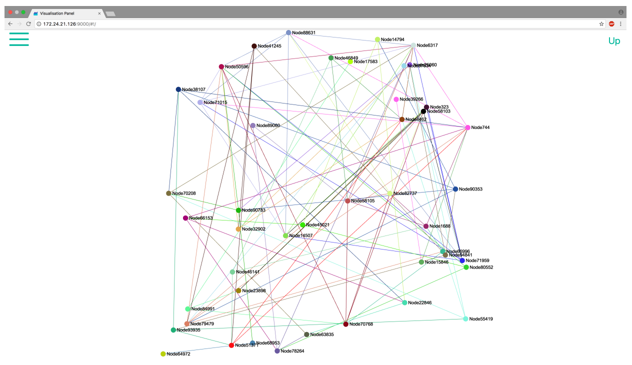

Fig. 5. bottom level node graph

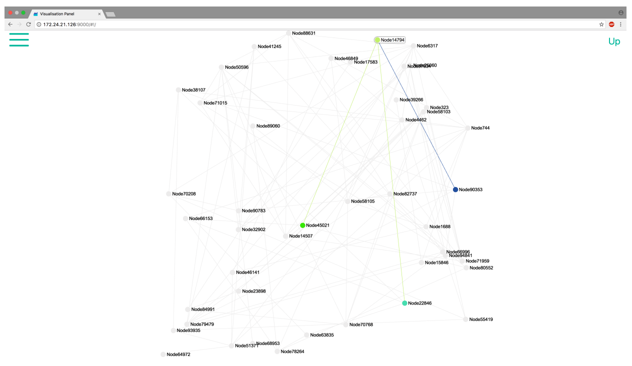

Fig. 6. pruned graph of neighbours of selected node

5 References

- http://micans.org/mcl
- https://github.com/COMBINE-lab/RapClust
- https://github.com/typicode/json-server
- https://github.com/yeoman/generator-angular
- https://nodejs.org/api