Fast Hierarchical Graph Clustering

Master Thesis - Extended Abstract*

Miguel Duarte
Instituto Superior Técnico, Universidade de Lisboa
miguel.martins.duarte@tecnico.ulisboa.pt

ABSTRACT
This work studies how graph clustering methods can be used to solve other graph problems, such as hierarchical clustering and graph reordering. In this context, it explores how Faststep [1], a recent method for matrix factorization and graph clustering, can be adapted to solve these other problems.

Our study develops a new method for this task, based on Faststep, and compares the proposed algorithm with state-of-the-art clustering methods: Layered Label Propagation and Louvain method. The results obtained are evaluated by several tests, which include comparison with ground-truth communities, networks generated by random models and direct comparison of algorithms. This study takes also advantage of Webgraph, a framework for succinct representation of graphs, and explores how clustering can be a powerful tool to improve the compression rates. We conclude that Faststep, although designed to solve a different problem, can still obtain acceptable results for this one. We also show that Louvain method, a hierarchical clustering method, can in fact obtain really promising results in the graph reordering task.

KEYWORDS
Network Science, Graph Clustering, Hierarchical Graph Clustering, Graph Compression

1 INTRODUCTION
1.1 Problem
Graph clustering or community finding is an important task to further understand real networks. Community finding can be seen naturally as finding groups of vertices in a network with denser connections between themselves and sparser connections with other elements of the network. This problem is computationally hard in general, and several greedy methods, with origins in multiple fields, have been proposed to solve it. These algorithms are usually based on general assumptions on community structure and are evaluated using ground-truth clusterings, i.e., graphs where communities are already known.

For a more formal goal, we can consider quality measures, with the most commonly used being the modularity [2]. Some algorithms are designed specifically to maximize some metric. For a more complete discussion on community finding we refer the reader to a review by Fortunato [3].

The complexity of the problem depends on the specific definition of clustering and it is tightly related with the quality measure we are optimizing. In general, this optimization is an NP-Hard problem [4]. Also, it is not usually possible to obtain an approximate solution which is within defined quality bounds.

Hierarchical graph clustering is a similar problem, but in this case we are looking for a hierarchy of clusters, i.e., how larger communities can be divided into smaller ones, and how each of these smaller ones can also be divided, and so on until we reach an indivisible community. Some of the algorithms used for the non-hierarchical problem already use an agglomerative or divisive approach, which implicitly generates an hierarchy. Non-hierarchical algorithms can somehow be used for this task, if there is a parameter which allows us to define the granularity of the communities to be detected.

A large number of methods for graph clustering and hierarchical graph clustering have already been introduced. If the reader is interested, the review by Fortunato [3] surveys most of the classical approaches to the problem. Currently, there is some focus on solving the problem for really large networks. Real networks can have several millions of nodes, which demands a linear or near-linear time algorithm. Layered Label Propagation [5] and Louvain method [6] are two greedy methods that can solve the clustering problem in near-linear time and provide solutions with reasonable quality.

A first goal of this work is to study how FastStep [1], a recent method for boolean matrix factorization, can be used for graph clustering. It was designed to factorize a large, sparse matrix, in a multiplication of two smaller matrices with a fixed width, but keeping the error as low as possible. This matrix factorization can also be interpreted as a clustering method, with a fixed number of communities. In fact, FastStep is a method which is able to find rich community structures and can run in near-linear time. This method takes a parameter \( k \), which can be used to define the granularity of the communities intended.

A second goal is to study how to apply it to the hierarchical community finding problem and test how it can be used for community refinement, based on the ideas used in the Layered Label Propagation method [5]. One important application of clustering is graph compression. As described in [7], it’s possible to obtain a better compression of the graph after a reordering of its vertices. Having information about communities or even better, hierarchical communities, allows us to increase the compression rate. Based on this, it is possible to infer the quality of a clustering algorithm just by analysing how much the graph can be compressed using this information. We will also follow this approach in the evaluation of our results.

A third goal is to integrate multiple community finding algorithms into a common framework for hierarchical graph clustering.

*Thesis Advisors:
• Alexandre Francisco (aplf@tecnico.ulisboa.pt) - INESC-ID and Instituto Superior Técnico, Universidade de Lisboa
• Pedro Ribeiro (pribeiro@dcc.fc.up.pt) - Faculdade de Ciências da Universidade do Porto
2 BACKGROUND

2.1 FastStep Algorithm

FastStep is an algorithm for matrix factorization proposed by Araújo et al. [1]. It factorizes a boolean matrix into two matrices with smaller dimensions and non-negative elements. It allows to understand the underlying structure of a matrix, and can be helpful for compression, prediction and denoising. The algorithm gives an approximated result with low error and runs in near-linear time on the number of non-zero elements.

An important feature of this method is the ability of finding rich community structures. The results obtained experimentally allow to obtain hyperbolic clusters, which, as shown in a recent work by Araújo et al. [8], is a good approximation to the internal organization of real communities. Common algorithms work only in the specific cases of finding rectangular like structures in the matrix, and that is the reason why the authors present it as a method that goes Beyond Blocks.

Another interesting property of FastStep is the strong interpretability of the boolean matrix decomposition. The use of non-negative factors in the matrices allows to establish the importance of elements and is in fact what enables representations of non-block structures. Also, the boolean reconstruction allows clear predictions and explanations of the non-zeros.

For a formal definition of Faststep, we will consider the following notation. Let $M$ be a $n \times m$ boolean matrix. The goal is to find a $n \times r$ non-negative matrix $A$ and a $m \times r$ non-negative matrix $B$, so that the product $AB^T$ is a good approximation of $M$ after thresholding.

$$\min_{A,B} \|M - u_T(AB^T)\|_F^2 = \sum_{i,j} (M_{ij} - u_T(AB^T)_{ij})^2$$

(1)

where $\|\cdot\|_F$ is the Frobenius norm and $u_T(X)$ simply applies a step function to each element $X_{ij}$.

$$[u_T(X)]_{ij} = \begin{cases} 1 & \text{if } X_{ij} \geq \tau \\ 0 & \text{otherwise} \end{cases}$$

(2)

The thresholding operator renders the objective function non-differentiable. In order to solve it, the function can then be approximated by another with similar objective:

$$\min_{A,B} \sum_{ij} \log \left( 1 + e^{-M_{ij} \times \left( \sum_{k=1}^{r} A_{ik}B_{jk} - \tau \right)} \right)$$

(3)

Here, $M$ is a matrix with values in $\{-1, 1\}$, where zeros are represented by $-1$ and ones by $1$. This function is the objective function used by Faststep.

The objective function can then be optimized in several ways. The authors used the gradient descent method. Let us consider $S_{ij} = \sum_{k=1}^{r} A_{ik}B_{jk}$. The gradient of the objective function, for $A_{ij}$, is then given by:

$$\frac{\partial F}{\partial A_{ik}} = \sum_{j=1}^{m} B_{jk} \left( \frac{1}{1 + e^{S_{ij} - \tau}} - \sum_{j \in M_i} B_{jk} \right)$$

(4)

The update rules for $B$ are similar.

It might be important to refer some implementation details. Matrices $A$ and $B$ are projected after each iteration, and projected to a small value $\epsilon$ instead of 0, as $A = B = 0$ is a stationary point of the objective function and there would be no improvement. The $\tau$ variable assumed a value of 20 in the authors implementation, because it allowed to achieve good results.

A straightforward implementation of the algorithm would take $O(Tnm^2)$, where $T$ is the number of iterations needed, $n$ and $m$ are the dimensions of the boolean matrix and $r$ is the rank of the decomposition (width of both $A$ and $B$, as stated previously).

This algorithm runs in squared time in relation to the size of the matrix; grows linearly in $O(nm)$. For a sparse matrix, it is possible to introduce some approximations to the algorithm that further reduce its complexity.

As we can see in equation 4, calculating the gradient for each element of the matrix $A_{ik}$ requires $O(nm)$ time. Still, the gradient can be approximated to have a total number of operations in the order $O(E)$, with $E$ being the number of non-zero elements in the matrix. Real networks are generally sparse: the number of edges is in the order of magnitude of the number of vertices. Also, This approximation introduces some error in the decomposition, but it is considerably low.

For brevity, the approximations introduced in the method are omitted, but the entire algorithm is explained in the original publication.

With these modifications, the total complexity can be revised to $O(Tr(E + P \log(min(n,m)) + n \log n + m \log m + S))$, where $S$ is the number of samples used to check convergence. When our matrix is the adjacency matrix of a graph, the complexity is:

$$O(Tr(E + (P + n) \log n + S))$$

(5)

As we can see, the algorithm scales linearly with $E$, the number of non-zeros. However, when the number of communities we want to find gets closer to $n$, the complexity of the method becomes quadratic on $n$. Also, there is no result that states the independence of $T$ and $r$, as the time the algorithm takes to converge might depend on the size of the matrix we want to obtain.

For community detection, i.e., to find if an element belongs to a factor $k$, it is possible to do it directly based solely on the principles of the decomposition.

For this specific purpose, we can replace the matrices $A$ and $B$ for one single matrix $A$, and use the matrix $A$ instead of $B$ in all the steps of the algorithm (i.e., $B = A$). This approach is acceptable, as we are looking for communities, and communities are considered undirected. The gradient and its minimization remains similar.

As such, a row element $i$ belongs to a factor $k$ if there is a non-zero in the reconstructed matrix in row $i$ and if this factor contributed with a weight above $\frac{r}{\max(b_k)}$.

$$A_{ik} \geq \frac{r}{\max(b_k)} \text{ and } S_i \arg \max b_k \geq \tau$$

(6)

Running the algorithm with different values of the rank $r$, makes it possible to obtain communities with a different granularity.

The authors show two empirical results on real data. The error obtained by their tests presented lower squared error than other scalable methods: SVD [9], NNMF [10] and HyCom [8]. The algorithm scales linear in the number of non-zeros, as proven previously. In the context of graph data, which is the concern of the present work, we know that non-zero elements are the
edges of the network, which means the algorithm scales near-linearly with the number of edges. It depends also on the number of iterations needed for it to converge, which is assumed to be small.

2.2 Label Propagation

Label Propagation [11] (LP) is a near-linear time algorithm for finding communities in a graph. Its time complexity makes it useful for large networks, where algorithms that take more than linear time take too long to run. Check the original publication for a description of the algorithm. The communities obtained, formed by nodes of the same label, are non-hierarchical and non-overlapping clusters.

The Label Propagation algorithm, even though is very simple and has a very good performance, tends to include the majority of the nodes in the same community, due to the inherent topology of real networks. That problem gave rise to several Label Propagation algorithms that keep the original idea and change the update rule of the labels.

One of the most interesting extensions to LP is the absolute Potts Model algorithm (APM) [12]. In this algorithm, let us consider, for a given node $x$, that its neighbours have labels $\lambda_i$, with $i$ between 1 and $K$, $K$ being the number of distinct labels in the neighbourhood of $x$. There are $k_i$ neighbours with label $\lambda_i$. In the simple LP algorithm, the new label $\lambda_i$ is chosen such that $k_i$ is maximum. APM, instead, maximizes the following expression:

$$k_i = \gamma(v_i - k_i),$$

where $v_i$ is the number of nodes in the whole graph with label $\lambda_i$ currently assigned.

Another important work based on this method is the Layered Label Propagation (LLP) [5], an algorithm for matrix diagonalization or graph reordering.

Large networks require special storage concerns, as they usually cannot fit in the memory of a computer with standard resources. To reduce the space used to store vertex indices, distances between indices are stored instead. To obtain a better encoding, vertices are reordered to minimize these distance values. This way, in a adjacency list, we expect to store smaller encoding, vertices are reordered to minimize these distance values between indices are stored instead. To obtain a better compression, vertices are reordered to minimize these distance values between indices are stored instead. To obtain a better compression, vertices are reordered to minimize these distance values.

In other words, elements with different labels are ordered with respect to these labels, elements with the same label keep the ordering of the previous iteration.

Several choices for the different $\gamma_k$ values can be made. The authors obtained better results by using a random $\gamma_k$, chosen uniformly from the set $\{0 \cup \{2^{-i}, i = 0, ..., K\}$. They also tried to use the same value $\gamma$ for all the iterations, but the results were always worse. In our point of view, this might indicate that combining communities of different granularity is a way of getting a better compression, which implies that a good hierarchical clustering method can be used to obtain an even better compression rate.

2.3 Modularity Maximization

Modularity is a metric introduced by Girvan and Newman [2] to measure the quality of a clustering. Initially, defined only for unweighted networks but, in a later work, also defined for weighted ones. Louvain method, that we will present next, works with multigraphs, which can be represented as weighted graphs. Therefore, we will use the definition using weighted edges to cover that case too.

This metric is based on the idea that a random graph does not have a community structure. For that, we consider a null-model, a new graph where the degree distribution is the same as the original graph, but the edges are rewired in a random way. The existence of a community can be discovered by comparing, for a specific subgraph, the expected density of edges and the real density of edges. Naturally, if the density of edges within a subgraph is much higher than expected, it means we might have found a community.

2.3.1 Definitions. Let us consider a function $f : V \times V \rightarrow N$, that assigns a non-negative edge weight to each pair of vertices of the graph. For unweighted graphs, can be defined as one when there is an edge between the two nodes and as zero otherwise. For multigraphs, we consider the number of edges between the two nodes. We consider also $deg(v) = \sum_{u \in V} f(u, v)$, which represents the degree of the node. $f$ and $deg$ can be generalized for a set of vertices $V$, where $f(V) = \sum_{u \in V, v \in V} f(u, v)$ and $deg(V) = f(V, V)$. A graph clustering $C = C_1, C_2, ..., C_k$ partitions the vertices into $k$ disjoint non-empty subsets $C_i \subseteq V$.

The expected number of edges within a community $C$ is given by:

$$\frac{deg(C)^2}{\sum_{C} f(V, V)}$$

which is the expected fraction of edges of the graph that are within $C$, i.e., the fraction of edges of the graph that are within $C$ in the null-model.

The real number of edges within a community $C$ is given by:

$$\frac{f(C, C)}{f(V, V)}$$

which is the fraction of edges of the original graph that are within $C$.

The modularity of a clustering $C$ is the sum, for all considered communities, of the difference between the real number of edges
and the expected number of edges. It is given by:

\[
Q_C = \sum_{C \in \mathcal{C}} \left( \frac{f(C, C)}{f(V, V)} - \frac{\text{deg}(C)^2}{2 \text{deg}(V)^2} \right)
\]  

(9)

2.3.2 Algorithms. Maximizing the modularity of a graph is an NP-Hard problem [4]. Several algorithms were proposed to solve the problem in a greedy way. As an effort to organize existing solutions into a coherent design space, Noack and Rotta [13] define two types of algorithms for modularity maximization: greedy coarsening algorithms and refinement algorithms.

The modularity optimization may fail to identify modules smaller than a scale which depends on the total number of links of the network, as shown by Fortunato et al [14]. As such, it is considered that the modularity optimization methods have a resolution limit: communities smaller that some specific size, which varies with the graph considered, may not be found by the methods and these communities are therefore included in larger ones.

2.3.3 Louvain Method. The Louvain Method [6] is a simple method for hierarchical graph clustering, based on the optimization of the modularity.

The algorithm starts with a weighted network of N nodes, and each node i belongs to a different community \( C_i \).

1. For each node i in the network
   1.1. For each neighbour j of i
       1.1.1. move node i to community \( C_j \), i.e., assign \( C_i \) to \( C_j \), if the modularity gain of this move is positive
   2. Merge the nodes inside each community into one single node
   2.1. The edges between nodes of the community become a self-loop, with weight equal to the sum of the weight of all those edges
   2.2. All other edges are kept and merged if their endpoints are the same.
   3. If step 2 made any change in the community structure, proceed to step 1. Otherwise, the algorithm stops.

The modularity gain from moving an isolated node i into community C is given by:

\[
\Delta Q = \left[ \frac{f(C, C) + f(i, C)}{2m} - \left( \frac{\text{deg}(C) + \text{deg}(i)}{2m} \right)^2 \right] - \left[ \frac{f(C, C)}{2m} - \frac{\text{deg}(C)^2}{2m} - \frac{\text{deg}(i)^2}{2m} \right]
\]  

(10)

where \( m \) is the sum of the weights of all the links in the network.

A similar expression can be derived for the modularity change when node i is removed from community C. The algorithm can then calculate the overall gain by removing i from its initial cluster and moving it to one neighbouring cluster.

Usually, the first pass (running step 1 followed by step 2 in the beginning of the algorithm) is the heaviest computational task and takes most of the computing time. The method can then run in near-linear time, if we can consider that the degree of a node is constant and does not depend on the size of the graph. The number of hierarchy levels resultant from the algorithm is small, which implies few steps of the algorithm to reach its conclusion. Also, several different tests were made by these authors [6] that show results with both high precision and good execution times.

3 APPROACH

As stated before, the focus of this work was to study if Faststep can be used to find hierarchical communities and/or refine communities, while comparing its results with LLP and Louvain method. This was the major goal of the project.

A first task consisted of understanding how Faststep can be used as a graph clustering tool. Obtained clusterings could then be evaluated using common metrics. Measuring the quality of a clustering algorithm is addressed in the subsection 3.1. An important part of the evaluation is also the comparison of the results with the ones obtained by reference algorithms. We compared Faststep with Louvain method and Layered Label Propagation. Section 3.2 explains how Faststep was modified with this objective and compared with the other methods.

A second task is related with the idea that knowledge about the communities of a network can be used to reorder its vertices and achieve better compression of a graph, when using a framework like Webgraph [7]. Therefore, for a better evaluation of the clusterings, we used them for graph compression, and the compression rates were compared for the same reference algorithms. Section 3.3 explains how reordering can be obtained from the used methods.

A secondary goal of the project was the creation of a framework for graph clustering. It aggregates the developed method and also LLP and Louvain method, our reference algorithms. The user of this framework has the possibility of choosing any of the available methods. This tool outputs a clustering or a reordering for the provided network. It also includes some functionalities to ease the task of comparing and evaluating clusterings and/or reorderings of the graph. The source files for the tool are hosted on Github [15].

3.1 Evaluation

Testing the quality of a clustering algorithm was an essential step of this work. To make sure the developed approaches are useful in practice, they must be validated systematically.

Complementary to the quality tests, we will finish with a complete benchmark of time used by the developed method and by the reference algorithms.

3.1.1 Initial validation of the results. One simple way of asserting the quality of an algorithm is running the method with datasets where communities are known. In a first evaluation phase, it is common to use small networks where we can easily see how the algorithm did and understand why it might have failed. Zachary’s karate club [16] is a small and classical example that is often used.

We used small graphs as a first validation for our work, but tests using large networks are of utmost importance, as we wanted to make sure that our methods behaved well, not only on small instances, but also on large real examples. SNAP [17] provides a collection of social networks with ground-truth communities identified. These networks have a number of nodes ranging from hundreds of thousands to millions of nodes. Another good source of networks with communities is the set of benchmarks provided by Fortunato et al [18]. These tools allow to generate graphs and can be used to systematically create tests for the developed methods.

Another way of evaluating results is by comparing them directly with other known and tested algorithms. The Louvain
method and Layered Label Propagation are the obvious target algorithms for clustering validation. In these tests, we can use any network, as known communities are not necessary.

3.1.2 Comparison Metrics. It’s important to note that both evaluation methods require the comparison of the clustering obtained by the algorithm in test with either ground-truth communities or a clustering obtained by another algorithm. Comparing different clusterings is also an interesting problem to solve. One simple way of doing it is using the Jaccard index. The Jaccard index evaluates the similarity of two sets. Considering two sets \( A \) and \( B \), 
\[
\text{Jaccard Index} = \frac{|A \cap B|}{|A \cup B|}.
\]
When comparing two clusterings \( A \) and \( B \), we want to find, for each cluster \( c \) in \( A \), the cluster in \( B \) which has smaller jaccard distance with \( c \). This metric allows to evaluate how well each cluster is detected between two different clusterings. When looking for an overall metric, we can average the smallest jaccard distances for each cluster of \( A \). In this work, when referring to the Jaccard Index in the context of the comparison of clusterings, it is assumed that we are considering this average.

We can also use a specific metric of similarity to compare two clusterings. One of the most commonly used is Normalized Mutual Information [19] (NMI). NMI is based on Information Theory principles. The main idea is to consider that if two clusterings are similar, then we need little information to infer one clustering from the other. We will use NMI in the comparison of different clustering for the evaluation of the developed algorithms.

The only problem with the use of NMI is that it doesn’t work for ground-truth communities where nodes can belong to more than one cluster. Lancichinetti et al. [20] presented an extension for the NMI, which was further improved by McDaid et al. [21]. This last improvement is referred as the Overlapping NMI and will be used instead of the NMI when evaluating overlapping clusterings.

3.1.3 Succinct representation as clustering metric. Another good way of evaluating clusterings is testing how well they work for graph compression or succinct representation of a graph. As stated previously, Webgraph [7] uses this knowledge to obtain a reordering of the vertices of the graph, which is useful to achieve a better compression of graph files. Our method can then be evaluated by the achieved compression rates, which will also be compared with a random permutation of the vertices and with the results obtained by the reference algorithms.

3.2 Graph Clustering

3.2.1 Faststep. As explained earlier, Faststep can take near-linear time for increasing size of the graph. However, the algorithm scales quadratically with the number of factors in the matrix we want to obtain. With this consideration in mind, it is expected that the algorithm would not be fast enough as the number of factors approaches the size of the graph. In the considered reference algorithms, it is possible to obtain a high granularity of communities. It is also an important goal of this project the usage of Faststep as a reordering method to achieve better compression of a graph. High granularity of the communities is, in fact, one of the reasons that allows a clustering algorithm to produce a good reordering of the graph.

To have a basic understanding on how Faststep would be affected by an increasing number of factors, we used the youtube dataset [22], a network constituted by approximately 1 million nodes and 3 millions edges and ran Faststep with different number of factors. Faststep running time depends also on the time needed for the gradient descent to converge. So, even though a larger value of \( k \) doesn’t imply directly that the algorithm will take a longer time to run, we discovered that even for relatively small values of \( k \) (\( k=32 \)), the algorithm took days. To overcome this problem, we decided to create a new recursive method, based on Faststep and using only small values of \( k \). That method starts with the entire network. In the first step, it runs Faststep with a factor of 1. That way, we are trying to reconstruct only one community of the graph. We then partition the graph into two: the obtained community and the remaining vertices of the graph. In a second step, the method is called for each of these two graphs. It stops partitioning when a specific threshold for graph size is reached. The pseudo-code of the algorithm is the following:

```
Algorithm 1: Recursive Faststep Method.

Function divide(graph)
  if graph.size < threshold then
    graph.save();
    return;
  d = faststep(k=1);
g1 = list();
g2 = list();
for i = 0 to graph.size do
  if belongsToCluster(d.rows[i][0]) then
    g1.add(i);
  else
    g2.add(i);
  end if
end for
divide(g1);
divide(g2);
end
```

The presented method allows to obtain hierarchical communities. Also, the algorithm can be modified to work with \( k \) greater than 1, but it is not trivial to define which \( k \) value to use for two reasons: when using \( k \) greater than one, we are looking for more than 1 community, which might, in fact, not exist, and the obtained communities may be overlapping, which has also to be solved for the algorithm to work.

The function \textit{belongsToCluster} was initially defined to use equation 6, which determines if a node belongs or not to a community, according to the authors of Faststep. The results of using this approach were very unrealistic: the communities found depended a lot on the number of iterations used internally for each factor, and its behavior depended on the network used. In most of the example graphs we used, Faststep would find no communities, or very small communities, even in large graphs. The algorithm would find small communities in the graph, which usually it could divide hierarchically, while the majority of the graph would be left untouched, with no clusters found.

We decided to change the way communities are chosen, using directly the values obtained on the rows of the matrix. In this situation, with the number of factors equal to 1, we have only one
row in the matrix, which we can use to tell how relevant each node is for the community. We analysed the values obtained in this row for real networks, and we could infer that they usually followed a power-law, as seen in Figure 1, which is expected. Our idea was to consider only the elements whose values in the matrix were greater that some specific cutoff value, which depended directly on the values found. We obtained acceptable results when using the average of the values as the cutoff. That way, we managed to remove the elements that were in the long-tail of the power-law, keeping only the ones with high value. This strategy allows to approximate the community found. As such, even if there is a clear difference in value between elements inside the cluster and elements outside, it is highly unlikely that the algorithm finds the exact cluster. The only reason we had to adopt this method is that the initial equation, proposed by the authors, would simply not return valid results, as explained.

![Figure 1: Values of the matrix obtained by Faststep (single row) for a real network, in decreasing order.](image)

The method was then modified to use an iterative approach where communities are kept in a queue, and larger communities are split first. When all communities currently in the queue are smaller than the threshold, we can save this information as one valid clustering of the graph. The threshold can then be further reduced, allowing the method to continue, retrieving more clusterings of higher granularities. This approach keeps all the subgraphs in a heap, with the larger subgraph on the top. The larger graph can then be removed from the top and split into two, which are added to the heap. Every time the size of the larger subgraph is smaller than the current threshold, the clustering obtained can be stored. The entire structure of communities is kept in a tree, which is an auxiliary structure of the algorithm.

One problem with this method is that it may force the clusters to have the same size, as they are broken into smaller ones according to the number of nodes they have.

The complexity of this method is the same as Faststep (check equation 5), with the difference that the number of executions of the algorithm is greater, and cannot be directly estimated. We are using the algorithm for graphs, so \( n = m \), and \( r = 1 \) as defined by the algorithm. The complexity is then:

\[
O(IT(E + (P + n) \log n + S)),
\]

where \( I \) is the number of runs of Faststep needed for the termination of the method. The number of runs cannot be bounded directly and depends on the configuration of the network.

### 3.2.2 Louvain Method

Louvain Method is, as explained previously, a hierarchical clustering method. As such, when looking for a clustering of the graph, we can get several valid clusterings, with different levels of granularity. Each valid clustering can then be partially decomposed: we may pick one cluster and replace it with the clusters from a clustering with a higher level of granularity. Therefore, the comparison of the clusters obtained by Louvain method with the clusterings obtained by another method cannot be done directly. To simplify the process, reducing the number of comparisons to be done, the clusterings considered are only the ones that correspond to the same level of depth in the hierarchy of communities i.e., the communities that are on the same depth in the dendrogram obtained by the algorithm.

### 3.2.3 Layered Label Propagation

LLP uses internally several iterations of APM, with different values of \( \lambda \). The results of these iterations can be retrieved and used directly in the comparison of clusters.

### 3.3 Graph Compression

As stated previously, one of the goals of this work is to understand how hierarchical clustering algorithms can be used to obtain good compression of graphs.

A key step is to obtain a reordering of graph from hierarchical communities. This can easily be done by storing the hierarchical communities in a tree, where leaves are the nodes of the graph and each internal node represents a cluster. We can then traverse it using a Breadth First Search. The order of each node in the obtained reordering is given by the order in which it occurs in the search. The choice of which child node to search into first is arbitrary.

This approach guarantees that closer nodes in hierarchical structure get closer positions in the obtained reordering, which implies a higher level of obtained compression.

Therefore, any method that is capable of obtaining hierarchical communities also solves the problem of reordering the graph. Both recursive Faststep and Louvain method are hierarchical, so this algorithm can be directly applied to obtain the clustering.

### 4 RESULTS

#### 4.1 Datasets

Multiple datasets were used for the evaluation of the algorithms. We were interested in experimenting with networks from different origins and with different sizes. Another important aspect for their choice was the availability of ground-truth communities, which allows us to see how the algorithms perform under real situations.

The datasets used are presented in Table 1. Check the original master thesis document or the original sources for a more complete description of these datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Nodes</th>
<th>Edges</th>
<th>Directed</th>
<th>Ground-truth communities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Airports [23]</td>
<td>3425</td>
<td>3.7 × 10^4</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Amazon [24]</td>
<td>3.3 × 10^5</td>
<td>9.2 × 10^5</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Web-google [25]</td>
<td>8.8 × 10^5</td>
<td>5.1 × 10^8</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Youtube [22]</td>
<td>1.1 × 10^8</td>
<td>3.0 × 10^8</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Wiki [26]</td>
<td>1.8 × 10^6</td>
<td>2.9 × 10^9</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1: Datasets used: source and information.
4.2 Detection of communities in artificial networks

As a first basic test for the clustering algorithms, we generated random networks with a defined number of communities and evaluated the results obtained.

The networks created had two fixed parameters: number of nodes equal to 10,000 and average node degree, $k$, equal to 50. We also considered two variable parameters: number of communities $C$ and probability of rewiring $p_{\text{rewire}}$. We then distributed the nodes randomly between communities: each node belongs to one and only one community. Each community $C_i$ would then have $n_i$ nodes. We then chose, for each community $i$, a node chosen uniformly between the communities of the graph and between the nodes of the chosen community.

These benchmarks were quite simple and offer only tests with communities of similar size. The parameter $C$ took values from 2 to 10, and values from 10 to 50 with intervals of 5, and $p_{\text{rewire}}$ took values from 0.0 to 0.9, with intervals of 0.1. The results of the tests were evaluated against the original communities, using both NMI and Jaccard Distance as defined in subsection 3.1. Each test was run using 5 random networks with the defined parameters and the average of the metric value obtained was taken. Figure 2 presents the results for the three algorithms considered, for the NMI metric.

![Figure 2: Benchmark results, using the NMI metric, for the tested methods: Louvain (left), LLP (center), Faststep (right).](image)

We can easily see that Louvain Method and LLP obtained similar results: when the communities were not excessively degenerated by the rewiring, they detected all communities with small errors.

Faststep obtained poor results: the values of both NMI and Jaccard metrics do not exceed 0.6, for all the tests considered. As expected, lower values of rewiring probability imply better reconstruction of the clustering.

We can argue that the communities and the nodes of the test graphs present a very homogeneous network, with low clustering coefficient relatively to the number of edges in the graph.

4.3 Detection of communities in networks with ground-truth communities

The Amazon, Youtube, and Wiki datasets provide ground-truth communities, i.e., the datasets contain information that allows us to infer known communities of these networks. In these datasets, the ground-truth clusters have some degree of overlapping, different granularities and sizes, which make difficult for the tested methods to obtain good results.

We present the results for the three datasets using both Overlapping NMI and Average Jaccard Distance between the ground-truth clusters and the found communities, comparing directly the score of obtained clusterings for the different methods. To keep this paper short, we include graphs only for the Amazon dataset (figure 3). The results of the other datasets are described in the text.

We then find, for each ground-truth community, the respective obtained cluster that has the smallest Jaccard distance. I.e., we present (Figure 4) the plot of the sizes of the ground-truth communities against the Jaccard distance with the most similar community found. This way, we can access how biased the methods are to finding communities of specific sizes. As before, only results for the Amazon dataset are presented.

![Figure 3: Detection of ground-truth communities in the Amazon dataset, using different levels of granularity of the methods. The result is evaluated with two different metrics: ONMI (plots on the left column) and Jaccard Distance (plots on the right column).](image)

We can argue that the communities and the nodes of the test graphs present a very homogeneous network, with low clustering coefficient relatively to the number of edges in the graph. Faststep is designed for real networks, which usually are scale-free networks, have high clustering coefficient and contain hyperbolic communities. Also, as the communities of graphs are equally strong, i.e., have similar density and size, it is difficult for Faststep to pick only one. The error accumulated by these assumptions might affect the performance of the algorithm.

This test presents strong indications that Faststep might not perform well in the next tests, but we do expect better results when using real networks.
would continually increase on each merge.

Amazon dataset: relation between size of each ground-truth community and quality in which the methods found it. The methods used are Louvain method (left column), LLP (middle column) and Faststep (right column).

results for ONMI were too low, which allows to understand the communities found were not, in any way, similar to the ground-truth ones.

The results for the Wiki dataset are also quite low for both metrics. The Wiki dataset is a graph naturally denser, which usually can make the found clusters to be larger. The Louvain method would usually stop in 2-3 iterations, as the modularity would continually increase on each merge.

The results for Faststep are of almost 0% for both metrics and for all datasets. We can only suppose that the communities found by the method are simply different from the known communities used.

The observation of the Figure 4, allows us to conclude that there is no correlation between the size of the ground-truth community and Jaccard distance of the most similar cluster found.

As an overall evaluation of this set of tests, LLP and Louvain method detected the ground-truth communities of the Amazon dataset, with some errors. The results for the other datasets were not as good. It is also important to remember that these two methods run in near-linear time for the number of nodes of the graph, compromising the quality of the results obtained.

Faststep presented really low results in this test. We could accept some degree of difference between known communities and found clusterings. However, the other methods performed much better at finding the known communities, which implies that these can be partially approximated using the structure of the graph, and Faststep is simply failing to detect them. The next results will help to clarify this situation.

4.4 Comparison of clusterings obtained with different methods

Clusterings obtained by different algorithms can then be compared directly. For this test, ground-truth networks are not needed, allowing us to use any graph. As done previously, we compare each clustering of each algorithm. For a visualization of the results of these comparisons, check the original master thesis publication.

An interesting finding that is important to analyse these results is the fact that all clusterings obtained with LLP have usually higher granularity than those obtained with Louvain method. In some cases, the higher granularity clustering from Louvain method (first level of iterative modularity merges) can have more clusters than the one with lowest granularity from LLP, but they usually have a similar number of communities. With this fact in mind, we can then expect the most similarity between clusterings to be in these two, for all datasets, and is in fact what happens.

There is a high similarity between LLP and Louvain clusterings, when considering the NMI metric. For this metric, the worst results are for the Wiki dataset. As discussed earlier this dataset is much denser than the others, which makes the first iteration of Louvain method to merge more than usual. In this case, there is a big difference between the granularities of the clusterings obtained by the two methods, which naturally is reflected on the results.

The similarity between Faststep and the other two methods is rather low. The values for this measure are not greater than 0.8 for any the tests.

We can also recognize in the tests between Faststep and LLP that the NMI values are, in some degree, independent of LLP clusterings (see results for Amazon or Web-google). This means that the NMI is just increasing with the granularity of Faststep clusterings. This problem is related to the way NMI is defined: a clustering with very high granularity can be seen as a further partitioning of an already existing clustering, which corresponds to relatively high NMI value, instead of a low value, as it would be expected. NMI is designed to penalize this type of mismatch between clusterings, but the normalization is usually not enough to all the cases, and can usually penalize too much in cases where the similarity is indeed considerable.

For the tests with Louvain method, this issue doesn’t occur due to the fact that granularity of the Louvain clusterings is small. The mutual information between large clusters and really small clusters will be low. This way, we obtain a lot of values near zero.

When considering the Jaccard metric, we can see that the values are much lower, except for the Amazon dataset. It is interesting to note that both algorithms (LLP and Louvain method) obtained good results for this dataset, in the test for ground-truth communities, which indicates that this dataset might be a good testing example, having features than make algorithms perform especially well with it.

We can see that there is a large difference between the two metrics used. This difference can be accepted in the basis that NMI is a metric specifically tailored for this type of test, while Jaccard distance is a more generic one. Also, NMI evaluates how much information two clusterings have in common, while Jaccard distance is focused on the number of common elements. Even so, we can observe, for the Jaccard metric, that the clusterings with more similarity are the high granularity ones obtained by Louvain and the lower granularity ones obtained by LLP, as discussed previously.

Faststep results are extremely low, when using the Jaccard metric.

Overall, when comparing Faststep with the other two methods, we have reasons to believe that they indeed produce very different clusterings. The similarity obtained using Jaccard metric is quite low, while the similarity using the NMI is overall high, but doesn’t make much sense as there is no correlation between the granularities of the clustering and the value of NMI obtained. These results seem to indicate that clusterings obtained by Faststep are quite different from the ones obtained with other methods. We can still evaluate independently the clusterings of each algorithm, taking advantage of graph compression, which we do in the next test.
with different methods. As a final test, we can evaluate the quality of the algorithms by seeing exactly what is happening with Louvain method. In some graphs, more than in others, the first iteration of Louvain method merges a lot of nodes, achieving a lower granularity of communities. When compressing a graph, we want to do a full reorder of its nodes. To achieve that, we need high granularity clusterings: when using low granularity clusterings, some sets of nodes will simply stay in the same order, not improving compression. We believe that, for some of the datasets used, that is the case of hierarchical clustering. We compared its results with two state-of-the-art algorithms, LLP and Louvain method. We tested how these three algorithms could be used for graph reordering and compression, which remained a questionable assumption of hierarchical clustering is transformed into a reordering of the graph, the traversal method of the structure has to decide which child node to visit first. This choice is completely arbitrary, but it has an impact on the compression obtained. If two clusters are siblings in the hierarchical tree, we do not know which of them to traverse first, but if one is the parent of the other, that ambiguity disappears. Therefore, we can reduce this ambiguity if there are more levels of hierarchy which, for Louvain method, some times there isn’t. This problem could easily be overcome by using the entire hierarchical structure of the network generated internally by the method, which is not an direct output in the implementation provided by the authors. Faststep obtained reasonable results. Naturally, the compression obtained is higher when the compression obtained by the other methods is also high, although Faststep results are always much lower than the other methods. In the previous tests we have already introduced several reasons why the performance of Faststep would probably not be as good as the other algorithms. Firstly, Faststep builds one community at a time. In that way, all nodes that belong to overlapping communities tend to be included in the first community created, while nodes with less edges but some proximity with this community that is being created are simply left out. Therefore, a large error is introduced in the first step of the algorithm. This error is successively increased by the next steps of the method. Also, the threshold which defines whether a node belongs or not to the community is just a loose approximation. However, we can still detect some compression of the graphs, which implies that Faststep does indeed find meaningful communities in the context of graph reordering and/or compression, which remained a questionable assumption in the other tests performed. Still, we empirically showed that is not the best method for the job, and that there are much better alternatives for this purpose.

4.5 Compression Results

As a final test, we can evaluate the quality of the algorithms by using their clusterings to obtain a reordering of the graph, and see how much space can be saved when this graph is compressed using Webgraph. This test is a useful technique to measure the quality of a clustering. As seen previously, Faststep does appear to find really different clusterings, when comparing with other methods or known communities. This test can help us to understand if, although obtaining different results, its results are in fact acceptable.

The original permutations of the vertices of the datasets used are usually in some order which allows better compression using Webgraph. As such, the clustering/reordering algorithms are run on a randomly permuted version of the original graphs, to eliminate the impact of the initial ordering from the effective compression rate. The sizes of the compressed graphs are presented in Table 2. The space savings of the methods are presented in Table 3.

LLP is an algorithm specifically designed for graph compression using Webgraph. Nevertheless, Louvain method managed to obtain similar results, gaining by a bit in the Amazon and Web-google datasets.

As discussed previously, Louvain method usually obtains communities with a lower granularity than the ones provided by LLP. In some graphs, more than in others, the first iteration of Louvain method merges a lot of nodes, achieving a lower granularity of communities. When compressing a graph, we want to do a full reorder of its nodes. To achieve that, we need high granularity clusterings: when using low granularity clusterings, some sets of nodes will simply stay in the same order, not improving compression. We believe that, for some of the datasets used, that is exactly what is happening with Louvain method.

In some networks, for example, in the Wiki one, the number of clusterings obtained by the Louvain method is also too low. For a better compression, not only we need a lot of clusterings with high granularity but it is better if each cluster in the hierarchical structure has the least number of children. In the way that a hierarchical clustering is transformed into a reordering of the graph, the traversal method of the structure has to decide which child node to visit first. This choice is completely arbitrary, but it has an impact on the compression obtained. If two clusters are siblings in the hierarchical tree, we do not know which of them to traverse first, but if one is the parent of the other, that ambiguity disappears. Therefore, we can reduce this ambiguity if there are more levels of hierarchy which, for Louvain method, sometimes there isn’t. This problem could easily be overcome by using the entire hierarchical structure of the network generated internally by the method, which is not an direct output in the implementation provided by the authors.

Faststep obtained reasonable results. Naturally, the compression obtained is higher when the compression obtained by the other methods is also high, although Faststep results are always much lower than the other methods. In the previous tests we have already introduced several reasons why the performance of Faststep would probably not be as good as the other algorithms. Firstly, Faststep builds one community at a time. In that way, all nodes that belong to overlapping communities tend to be included in the first community created, while nodes with less edges but some proximity with this community that is being created are simply left out. Therefore, a large error is introduced in the first step of the algorithm. This error is successively increased by the next steps of the method. Also, the threshold which defines whether a node belongs or not to the community is just a loose approximation. However, we can still detect some compression of the graphs, which implies that Faststep does indeed find meaningful communities in the context of graph reordering and/or compression, which remained a questionable assumption in the other tests performed. Still, we empirically showed that is not the best method for the job, and that there are much better alternatives for this purpose.

4.6 Performance Evaluation

Check original document for execution times of the tested methods with multiple datasets.

5 FINAL REMARKS

In this work, we proposed an adaptation of Faststep for the problem of hierarchical clustering. We compared its results with two state-of-the-art algorithms, LLP and Louvain method. We tested how these three algorithms could be used for graph reordering and compression of networks using the Webgraph framework. Finally, we implemented a tool which allows the retrieval of graph clusterings and reorderings and the use of these clusterings or reorderings in the compression of the graph.

Due to the fact that communities could not be retrieved directly using the method provided by the authors, we did a possible approach to this problem. Our method tries to pick one community at a time, in contrast to other algorithms that propagate labels or merge clusters incrementally. This has a significant impact on overlapping nodes. Overlapping nodes are usually hubs and tend to have many valid communities. In this situation, they cannot
be placed on more than one community, and so Faststep places them on the first that is found. The other methods can somehow balance this effect, as communities are constructed incrementally and can gather the more important nodes to themselves. This approach can lead to a lot of variations between the communities found by Faststep and the communities found by another algorithm. Also, the cut-off added to the method to decide whether a node belongs or not to the community introduces a lot of error. As said previously, the simultaneous construction of the communities allows them to control each other; Faststep, on the other side, uses a much more crude approach which can place too much nodes in the cluster, but also too few. The method can have a reasonable performance when dealing with difficult graphs, especially when communities are heterogeneous, as the boundaries between communities are not well defined and the error is acceptable. The results obtained on graph compression through reordering prove this idea. However, when dealing with graphs with clear clusterings, even ones that are perfectly bounded, the implemented cut-off is incapable of accessing this information and introduces a lot of error.

This details of the method obtained worse results than we could expected in the beginning of the work. It was specially inaccurate for artificial communities, which don’t have the features that characterize real networks. When applied to real networks, the clusterings obtained were very different from the ones provided by LLP or Louvain method. Despite these results, it managed to obtain reasonable ones in graph reordering/compression.

The performance of Faststep could benefit a lot with an improved method to decide if a node belongs or not to a cluster. However, we understood the limitations of the algorithm. In fact, Faststep was designed to solve a different problem, matrix factorization. As such, its results, although interesting for the comparison of the two problems, cannot be expected to be better than the ones obtained by algorithms specifically proposed for this matter.

We believe that our study of Faststep as a clustering, hierarchical graph clustering and reordering algorithm allowed to explore the possibilities of the method. Even though there are other approaches to this problem, we consider that there is not much that can be done after this, as the results are a long way from those of LLP and Louvain method, and any possible improvement would not, in our point of view, be enough to make it competitive in both quality and execution time.

However, we think that Faststep could be improved in a lot of ways, which could perhaps lead to much better results. These improvements were mentioned in this work and are directly related to the results in tests performed. A proper technique of defining which nodes belong or not to a community, which works for all types of networks would be the most urgent. Currently, Faststep works with a fixed number of communities. Extending the algorithm to allow it to search or refine the number of communities in the network could possibly extend the usage of algorithm to other interesting problems. Another useful improvement would be increasing the efficiency of the algorithm, as it takes much a longer time to run that other algorithms with similar time and space complexities.

In our opinion, the results with Louvain method to obtain a reordering of the graph seem promising and, as previously suggested, they could benefit if the hierarchical structure of the graph, obtained internally in the maximization of the modularity, was used in the reordering algorithm. We could also understand that the clusterings obtained by LLP have, in general, a higher granularity than those obtained by Louvain method, which presents an opportunity to study how their reorderings could be used together for an improved result.

REFERENCES


