Contents

Aims and objectives

Several model structures describe shape and properties of graphs. One of such models is referred to as *community structure*. The definition itself of community structure however is not well delineated nor unified among the scientific community, and as a result a wide variety of algorithms, quality functions and comparison metrics have been proposed in the research community, all focusing on somewhat different aspects.
The principal idea though that best describes the concept of community is that of a group of vertices that are densely connected to other vertices within the community, and at the same time sparsely connected to vertices outside the community.

The discovery of communities has become a broadly popular topic, and it found applications in the fields of sociology, biology, chemistry. Rarely these studies took into account directed graphs, probably because most of the times it is hard (if not impossible) to extend algorithms for undirected graphs to their respective directed versions.

The aim of this project is to develop methods to hide the community structure in a network. I.e. given a directed network with community structure, what are the manipulations that should be applied in order to destroy a given community?
In order to do so, during my thesis work I investigated the presence of already existing methods that aim to achieve similar results, finding that not much has been done, bar for a study that was set in the undirected setting and based its manipulation strategies on classical centrality measures.

In my work I devise alternative approaches to hide the community structure that make use of either local or global information models, but allow only local alterations when operating the manipulation strategies. This is because the hypothesis underlying my work is that the entity wishing to hide a target community may be the community itself, thus it is sensible to assume that its access powers are confined within the community's boundaries. The most of the effort was put into the research for a manipulation strategy that could be as much independent as possible from the specific community detection tool an adversary may pick to detect the target community.

Chapter 1

Community detection problem

The objective of this work is to find a way to hide communities from detection, so in order to achieve this goal it is important to understand how the community detection algorithms work. In this chapter I will provide basic graph theory concepts and introduce the community detection problem, comparing it against the similar graph partitioning problem. Without diving in unnecessary algorithmic details, I will discuss how these two problems, although apparently related, find different applications and require different treatment, highlighting how partitioning is used for decision making while community detection for structural analysis.

I will then quickly discuss the evolution of community detection techniques, starting from the traditional hierarchical methods, to the more efficient modularity based techniques.

1.1 Preliminaries of graph theory

The elementary building blocks of a graph G are nodes (vertices) and edges (links). In the common conception nodes represent entities and edges represent some relations that run through them. Usually the set of nodes is denoted as V and the set of edges as E.
Graphs can be *directed* or *undirected*. In undirected graphs an edge is simply a line between two nodes. In a directed graph edges go *from* a node *to* another, and the direction is represented with an arrow. Depending on whether the graph is undirected or directed the set E shows two slightly different shapes: in the first case it is a set of unordered pairs of nodes from V, in the second case it is a set of ordered pairs of elements of V.

The *size* of a graph G is the number m of its edges, thus $m = |E|$, and the *order* is the number n of its nodes, thus $n = |V|$. The number k_i of other nodes to which a node i is linked, or equivalently the number of edges departing from it, is its *degree*.
A *path* between two nodes is the set of edges that need to be traversed in order to get from one node to another; the number of such edges is the *length* of the path; the *distance* between two nodes is the length of the shortest path between the two nodes. The *diameter* of a graph is the largest distance between any pair of nodes.
Two nodes for which there is a linking edge are said to be *neighbours*. A *complete* graph, or a *clique*, is a graph in which all nodes are linked by an edge.

Clustering is the classification of nodes into groups. The *clustering coefficient* of a vertex

for an undirected graph i is $C_i = \frac{2 \cdot |\{e_{jk}\}|}{k_i(k_i-1)}$, where the set $\{e_{jk}\}$ represents all edges between all possible pairs of neighbours j and k of i. The clustering coefficient hence is the ratio between the number of edges that run between its neighbours and the number of all such possible edges. It measures how close a vertex and its neighbours are to form a clique. The directed version hasn't got the factor 2 at the numerator.

A graph G is said to be *sparse* if the number of edges m is comparable with the number of nodes n, or $O(|V|) = O(|E|)$. Graphs are *regular* if each node has the same degree, *heterogeneous* otherwise.

A graph is said to present *community structure* [GN02] when within it a number of clusters are detectable, and their nodes are densely connected to nodes within the cluster they belong, while loosely connected to nodes belonging to other clusters. The notion itself of community structure is not unambiguously defined among the scientific community. It's been presented in many flavours, depending mostly on the applications or desired features. However this high level idea is broadly shared in the graph theory research community.

1.2 Graph partitioning and community detection

A particular interest arose in network studies in the last years, in part due to the fact that large data sets describing the structure of big networks now exist (eg. World Wide Web (WWW)). Before the availability of these huge data sets, problems modelled with the graph abstraction were of manageable dimensions, allowing fairly fast analysis just by plotting down the graph with pencil and paper. With networks of size $\in O(10^2)$ and above this is no more true (the WWW's recent estimate is about 8 billion pages[1]), and automatic tools for elaboration and analysis are demanded.

Graph partitioning is the problem defined as the division of a graph G into p subgraphs of given size, such that the cut-size (i.e. the total number of edges between the subgraphs) is minimised. An example of graph partitioning is given in figure 1.1. The number and size p of the partitions is strictly necessary in the formulation of the problem, otherwise trivial solutions would be obtained. For example putting all nodes in one only partition is optimal as it gives a cut-size of 0, or separating from the graphs just the node with smallest degree gives minimum cut-size as well. It is clear that these solutions are not of much interest.

However the graph partitioning problem finds many applications to problems in which these parameters are well known, like integrated circuit design (as in fact electronic circuits partitioning onto boards was the problem that inspired one of the earliest and most popular works in the field [SK88]),

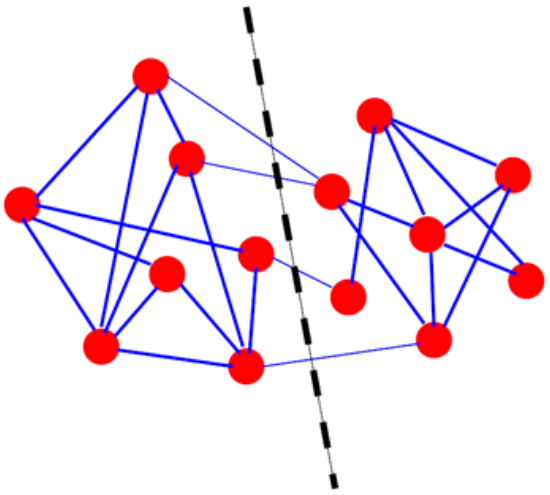


Figure 1.1: An example solution for a graph partitioning problem. Graph G has order 14, p is 2 and the minimum cut is 4. (From [For10].)

[1] source: http://www.worldwidewebsize.com/

and parallel computing, where given a certain number of tasks and processors one wants to distribute the tasks among the processors in a way such that the interprocessor communication is minimised. Graph partitioning in many flavours is NP-hard, although many approximation methods present a lower complexity ($O(n^3)$ in sparse graphs).

Conversely the problem of community detection consists into finding those groups of nodes in a graph which are densely connected to nodes within the group and at the same time sparsely connected to nodes outside the group. The first difference with respect to graph partitioning is that in community detection one doesn't know the number nor the dimension of the communities in the graph. For this reason algorithms for graph partitioning are not applicable to community detection problems. On the contrary, one desires the informations about number and size of the communities to be an output of the algorithm. Hence graph partitioning can be seen as a decision making algorithm, while community detection can be better seen as a data analysis algorithm, employed to extract interesting informations on the structure of large data sets, that would otherwise result into an impenetrable fog of nodes and edges.

Another important difference is that when applying partitioning one wants to find the best possible partitions while being not interested into knowing whether a good one actually exists, by contrast community detection in principle should allow the possibility that a graph doesn't show community structure, thus desiring the algorithm to return no division at all.
Community detection is a method that aids both the identification and the interpretation of functional subgraphs within larger graphs based on the nature of the relation attributed to the edges, and it finds application in several fields. For instance in sociology communities in a social network may be pinned to different social groups [GN02], in citation networks and WWW the communities may represent a common topic [Red98], email networks may reveal both the formal and informal organisational structure within a university, or a company department [GDDG$^+$03], in biological food webs communities may represent subsystems in ecosystems [WM00], in neural and biochemical networks communities discriminate functional groups [HH03]. Some of these findings often lead to the deduction of further non obvious links between members of a community.
Another interesting application is network visualisation [NG04], where large networks are reduced so to show only the main components and the strength of their correlation.

1.3 Hierarchical clustering

Hierarchical clustering is a traditional clustering technique that finds many applications in the fields of Social Network Analysis, biology, engineering and marketing. Hierarchical clustering's proceeding is divided in two steps. A first step in which the similarity W_{ij} between every pair of nodes i and j is measured, irrespective of whether they are directly connected or not. This similarity function represents how closely related the nodes are, and several such functions were proposed. A popular similarity function for instance is the number of all possible node(edge)-independent paths that connect two nodes, where node(edge)-independent means that no two such paths have a node(edge) in common.

The second step implies grouping (dividing) nodes by adding (removing) edges [2] starting with those having the highest (lowest) value W_{ij}.

According to whether edges are added or removed, hierarchical clustering techniques are then divided into agglomerative or divisive techniques. The first is a bottom-up approach that starts from completely disconnected nodes and builds up groups by adding edges iteratively, the second is top-down as starting from the whole graph, edges are removed iteratively to form smaller groups. These proceedings produce some nested sets that increase (diminish) their size as edges are added (removed). Among these, connected subsets represent communities. These nested divisions are suitably representable via special trees as figure 1.2 shows, known as dendrograms.

A horizontal slice at the *k-th* level of the dendrogram represents the community structure found just after the *k-th* edge was added (or removed).

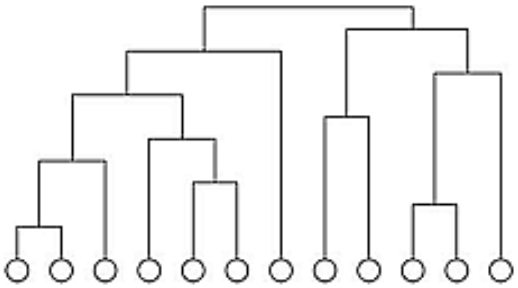


Figure 1.2: A dendrogram. It is a tree in which nodes are represented by its leaves. The tree shows in what order nodes are joint together to form a cluster. (From [GN02].)

Hierarchical methods present a nice feature as opposite to partitional clustering: one doesn't need to know the size or number of clusters. However sometimes they have failed to present a significant accuracy when tested on networks in which the community structure is well known. The problem with them is that they must give a hierarchical structure to the found division, although the considered graph may not be intrinsically hierarchic. Moreover they do not provide a criterion to determine which of the found partitions is the best one, and the partitions' shape strongly depends on the similarity function adopted. Another big problem with these methods is that peripheral nodes with just one neighbour tend to be isolated while they should belong to the community "on the other end" of the edge. This unpleasant effect is due to the fact that they have one only edge, hence the associated weight W_{ij} tends to be very small.
Hierarchical agglomerative algorithms were far more popular than the divisive ones [For10], this unless Girvan and Newman [GN02] introduced their well renown divisive algorithm.

1.4 Girvan-Newman divisive algorithm and modularity function

In this section I will talk about the Girvan-Newman algorithm (GN-algorithm), a hierarchical divisive algorithm for community detection introduced by Girvan and Newman [GN02], and improved later by the same authors in [NG04]. I will talk about how it differs from traditional hierarchical methods and what are its main features, comprising the innovative selection criterion based on what then will become the very popular *modularity function*.

[2] assuming an edge between i and j exists

Algorithm

In [GN02] Girvan and Newman propose a new algorithm, which represents a turning point [3] in the community detection research field. They use hierarchical clustering algorithms as a starting point, but criticise all the shortcomings that they have, and then show how their idea overcomes those problems.
They point out that, unlike traditional hierarchical clustering methods that focus on the most central edges, their proposed method focuses on those edges that are least central (or more in between), and rather than constructing the communities by adding vertices, they disrupt the network in communities by removing edges.

Their observation is that if a graph presents community structure, then all the shortest paths between nodes in different communities must pass through those edges that represent the "loose" connections between the communities. In other words there are a few inter-community edges, and all the shortest paths between different communities must pass through these few inter-community edges. Once identified, these loose connections can be removed in order to isolate communities.
They draw their centrality measure from Freeman's definition of node betweenness [Fre77]. Node betweenness represents to what extent nodes are influential in a network. More formally it is the number of shortest paths that pass through a node. They extend this definition to edges, with the addition that multiple shortest paths between a pair of nodes are given a weight equal to the inverse of the paths multiplicity. Therefore the edges with a high betweenness score will represent the "loose" connections between communities.
The algorithm they propose is as follows:

1. compute edge-betweenness for all edges
2. remove edge with highest edge-betweenness
3. recompute edge-betweenness for all remaining edges
4. go back to step 2

It is evident how they built on the precedent hierarchical clustering methods to get to this method. However the main difference is that upon each edge removal the centrality measures of the graph's components is recomputed. They justify this extra step with this argument: once an edge is removed, the remaining measures do not reflect the resulting graph anymore. For example if two communities are connected by two or more edges, maybe only one of those shows high betweenness; now if this edge is removed then the betweenness of the other edges may increase as they may replace the previous edge in some of the shortest paths that used to go through that edge. Later, for the sake of completeness, in [NG04] they provided experimental evidence of this argument by showing the loss in accuracy the algorithm suffers when this recalculation step is omitted.
However this extra step comes with a cost. Naively one may think that since for a graph with n nodes and m edges, shortest path lengths can be found in time $O(m)$ and there are $O(n^2)$ possible node pairs then the recalculation step takes $O(mn^2)$ time overall. As a matter of fact it can be instead computed in time $O(nm)$ with an algorithm based on breadth-first

[3] according to Google Scholar this paper is cited over three thousand times

search that Newman himself developed [New01]. Now since the recalculation step needs to be remade for every removed edge, the total complexity of the method is $O(m^2n)$, or $O(n^3)$ for sparse graphs, which makes the algorithm quite slow for large networks. In most cases however one needs to recompute betweenness only for those edges belonging to a component subgraph from which the last edge was removed. The good thing is that this method breaks the graph in subgraphs quite early, having that the necessary work in practice is less than the theoretical worst case.

In summary the recalculation step, although computing intensive, pays back with an accuracy so high that results in an acceptable trade-off. In fact when applied to computer generated networks there is an outstanding gap in accuracy between their proposed method and the traditional methods.

Modularity function as selection criterion

Subsequently to the publication of their first work [GN02] their method became broadly accepted and divisive algorithms were seen as a more promising alternative to the formerly more popular agglomerative counterpart. Newman and Girvan built upon it and presented some extensions to their work [NG04], including the introduction of the very popular Modularity function.

The first version of the GN-algorithm found all possible community divisions, in a nested fashion, starting from the whole graph ending to the graph with all nodes disconnected (singletons). This meant that one had to derive which division was of most interest by direct observation of the dendrogram. If one expected a certain number of communities, or a certain size the communities had to show, this job could be more easily done. But as said before, these informations are desired to be an output of the algorithm.
Their algorithm, although high performing on graphs for which the community structure was well known, gave no criteria to assess which and whether a found division was indeed of interest. Moreover the algorithm *always* provided a set of divisions, also for those graphs that had no natural community structure. Therefore arose the need for a criterion to evaluate how good a division found was. For this reason they introduced the now broadly adopted *modularity function*.
Let C_i be the community to which node i belongs, and let $\mathbf{A}$ be the adjacency matrix for a graph G. A rough approximation of the modularity function could be

$$Q = \frac{\sum_{ij} A_{ij} \cdot \delta(C_i, C_j)}{2m} \tag{1.1}$$

where $\delta(C_i, C_j)$ is Kronecker's delta, and it is 1 when i and j belong to the same community, 0 otherwise. The quantity in equation 1.1 is the sum of all the edges between nodes that belong to the same community, over the total number of edges (the factor 2 in the denominator takes in account the symmetry of the adjacency matrix for undirected graphs). Therefore this quantity represents, for a given division in communities, the fraction of intra-community edges. In a graph with strong community structure one would expect this quantity to be as high as possible (note that $Q \in [0, 1]$). Although very intuitive, the problem with this approximation is that it is highly influenced by the number of communities found in the division. As a an extreme example let us consider a division that

comprises just one community: the whole graph. In this case the Q function will assume value 1, which is the highest possible, but the division is of no interest at all.
A corrective term needs to be added to overcome this problem. The intuition to follow is that in a graph with community structure one expects the quantity in equation 1.1 to be statistically significantly higher than the same quantity computed for a random network. Then 1.1 can be rewritten as:

$$Q = \frac{\sum_{ij}(A_{ij} - \frac{k_i k_j}{2m}) \cdot \delta(C_i, C_j)}{2m} \tag{1.2}$$

where k_i is the degree of node i, so $\frac{k_i k_j}{2m}$ is the probability that an edge exists in the standard configuration model (better known as *null configuration model*), that is a random network in which edges are randomly rewired but keeping the original nodes degrees. Therefore the term $\sum_{ij} \frac{k_i k_j}{2m}$ represents the fraction of intra-community edges *expected* in such random network.

Positive values of equation 1.2 indicate that the considered community division for the graph G is more significant than the same division applied to a graph with random edges. The higher the value of Q, the more we find edges within a community than expected by chance. Newman and girvan found that values to be considered "good" fall in $[0.3, 0.7]$, and higher values are generally rare.
With respect to the problem pointed above with the first approximation, it is important to notice that now if equation 1.2 is computed for the division with just one macro-community one would get $(m - m)/m = 0$, hence indicating that such a division is of no interest.

Now that they defined a function to assess to goodness of a division, they moved on and introduced a criterion to pick the best division. The criterion itself is quite intuitive: it is just enough to compute this quality measure for each of the divisions produced iteratively by the algorithm, and pick the one with the highest peak.
Despite its simplicity the method shows that the peaks are found where they would be expected to be found, when tested on graphs for which the community structure was well known.
For its power and simplicity this method was applied to several studies, like metabolic networks, genetic networks, collaboration networks, email networks and others.

In this section I talked about how Girvan and Newman started a new wave of studies and applications in the field of community detection. I have highlighted how their approach is different from *traditional* hierarchical clustering and how it overcomes the problems the latter had. Lastly I talked about the modularity function and the importance it has into assessing the goodness of a community division.

1.5 Families of modularity based techniques

The modularity function was introduced as a way to pick the best community division in a hierarchy of divisions, but for its good features it has been employed as a central ingredient in many community detection algorithms. The intuition is that a high modularity score is likely to indicate a good partition, so several proposed algorithms aim to maximise

the modularity score over the possible partitions. However modularity maximisation is a NP-hard problem [BDG+06] since there are way too many ways to partition a graph, even if of small dimensions. I will give a quick overview of the main families of algorithms based on the modularity function ([For10] is a thorough overview about the subject).

Greedy algorithms represent a not very accurate, but still computationally fast, technique category in which clusters are agglomerated starting from singletons, adding edges one by one, picking those that increase the modularity score first. These techniques lead to n different divisions, each having from n to 1 clusters. The one division with the highest modularity is taken to be the best division. These algorithms suffer of the problem that small communities add very little to the modularity score as opposite as large communities, having so a bias into exploring large communities first and tendentially ignoring small ones.

Based on physics principles, *Simulated Annealing* techniques are very accurate but also computationally very demanding. They are probabilistic processes that aim to optimize a given function by bringing the system from a state to another as a result of a *move*. The move happens with probability p, where $p = 1$ if the new state is such that the function improves, otherwise p is proportional to the decrease. In the community detection setting the function to optimise is the modularity function, and the moves can be either local (moving a node from a cluster to another) or global (merging/splitting clusters).

Extremal Optimisation represents a compromise between greedy and simulated annealing techniques. Starting with two random partitions, nodes are moved from a partition to another one at a time, picking those that present the lowest fitness to a local form of modularity function.

Spectral Optimisation techniques aim to maximise the modularity function relying on the eigenvalues and eigenvectors of matrices derived from the adjacency matrix. Such techniques can be faster and also slightly more accurate than Extremal Optimisation ones

In this chapter I gave a broad overview of the aspects of community detection I covered during my research. I shown how in community detection it is important to have techniques that require no a priori knowledge with respect to the community divisions. I talked about the modularity function and the vast literature that followed it.
These concept will be extended to the directed setting in the next chapter.

Chapter 2

Community detection in directed graphs

In the first years of the research in community structure detection the directedness of graphs was just ignored. Conversely many of the networks of interest are indeed directed, like for example the WWW, food webs, biological and even social networks. In some cases ignoring directedness gave good results anyways. In [NG04] one can read *"we have found that in many cases it is better to ignore the directed nature of a network in calculating community structure. Often an edge acts simply as an indication of a connection between two nodes, and its direction is unimportant"*. However this is not always the case, and in fact I will start this chapter talking about the contents of a paper that a few years later the same Newman co-authored. In this work it is highlighted that indeed in some networks direction is of importance in community detection.

I will then talk about a method that was proposed to treat directed graphs as weighted graphs in order to retain the information represented by the direction of edges, so to use CD algorithms based on maximisation of the weighted version of the modularity function. Lastly I will present two CD algorithms that take direction of edges into account, but do not aim to maximise the modularity function. They are based on different concepts, and will come in handy for evaluation purposes later.

In summary, the algorithms presented in this chapter will either inspire some of the work related to the manipulation strategies or will be used to measure the fitness of such strategies.

2.1 Modularity function for directed graphs

Leicht and Newman [LN08] criticized the fact that previous to their work the directedness in graphs was just ignored when it came to community detection. For this reason they introduced a way to extend community detection algorithms so to exploit the information included in the direction of the edges.
They remarked how high modularity can be interpreted as an indication that a statistically surprising configuration is present in a graph. To integrate the edge direction to this intuition they set up the following scenario: suppose there are a node i with many outgoing

edges and a few ingoing ones, and a node j which conversely has many ingoing edges and only few outgoing. Statistically speaking one would more likely expect to have an edge from i to j than *vice versa*. Their suggestion is that in case the *vice versa* happens, this should give a bigger contribution to the modularity function, just because it captures the "statistically surprising" nature of the network at study.
They proposed an extension to the modularity function (equation 1.2) in this terms:

$$Q_d = \frac{1}{m} \cdot \sum_{ij} (A_{ij} - \frac{k_i^{in} k_j^{out}}{m}) \cdot \delta(C_i, C_j) \tag{2.1}$$

where k_i^{in} represents the in-degree for node i and k_j^{out} represents the out-degree for node j. Therefore with this formulation of the modularity score the most of the contribution is given by those edges that run from nodes with a small k_j^{out} to nodes with small k_i^{in}.
With this new formulation one can adaptively use any already existing modularity maximisation technique to find communities in directed networks.

They applied this enhanced function to a directed network for which the community division was unknown, representing a glossary of words related to graph theory in which an edge runs from a node to another if the first was used in the definition of the second. Their claim here is that their enhanced algorithm identifies communities which are better coalesced in terms of concepts represented by the words with respect to those communities found by the undirected counterpart.
With this examples Leicht and Newman have effectively shown the importance of the information included in edge directedness, more over they remarked that the adoption of the enhanced definition of modularity function does not imply any increment to the computational complexity of the community detection algorithm. This is quite a big result, implying that one can obtain an accuracy improvement at no cost.

2.2 Directed graphs as weighted graphs

In 2009 Kim et al [KSJ09] criticise the work by Leicht and Newman claiming that *"our [Kim et al] investigation of this method shows that the method they [Leicht and Newman] used does not exploit direction information as they proposed"*.
To include the information deriving from direction properly they then propose to transform the network at study into an undirected weighted network, where the weights should reflect the statistically surprising nature of the direction of the original edges. Once this transformation is operated, then any method based on modularity optimisation for weighted graphs can be used to detect the communities in the network.

They agree with Leicht and Newman's intuition about the superior statistical significance deriving from those configurations in which an edge runs from a node with small outgoing degree to another with small ingoing degree. To capture this significance better they define

$$p_{ij} = \frac{k_i^{in} k_j^{out}/2m}{(k_i^{in} k_j^{out}/2m) + (k_j^{in} k_i^{out}/2m)} = \frac{k_i^{in} k_j^{out}}{k_i^{in} k_j^{out} + k_j^{in} k_i^{out}} \tag{2.2}$$

as the probability to find an edge that goes from j to i when the links are placed at random while keeping the degree sequence of the nodes.

If such a probability is small, then the edge directed from j to i represents a statistical surprise that should contribute to the identification of i and j as part of the same community. Thus they define the *relatedness* score between two nodes as:

$$w_{ij} = A_{ij}(1 - p_{ij}) + A_{ji}(1 - p_{ji}) \tag{2.3}$$

where A is the adjacency matrix. The w_{ij} relatedness score then will be the weight of the edge between nodes i and j so to have a weighted network that embodies information from the original direction of its edges. Any community detection algorithm that maximises the modularity function 2.4 for weighted networks can be used to analyse this network.

$$Q_w = \frac{1}{2W} \cdot \sum_{ij} (W_{ij} - \frac{s_i s_j}{2W}) \cdot \delta(C_i, C_j) \tag{2.4}$$

This extension to modularity was defined in [New04], W is the sum of all the weights, while s_i is the *strength* of node i (i.e. the sum of the weights of all edges touching i).
When using a community detection algorithm that maximises the modularity function Q_w, the edges with largest W_{ij} (i.e. smallest p_{ij}) are the ones that contribute the most to its maximisation if considered as part of the same community.

2.3 Infomap

Infomap [RAB09] is a community detection algorithm based on principles from information theoretics, Markov processes and spectral methods. It works with weighted and directed graphs, which are seen as flow networks. The links' direction and weight represent direction and capacity of flows. The flow in the network is then modelled as a random walk. Rosvall et al set the following problem: find the most efficient code that compresses information about the trajectory of a random walker around the network. Their claim is that finding an optimal encoding of this walk is equivalent to deriving the community structural properties of the network at study.

Compressing data and finding significant patterns in such data are two tightly correlated problems. In fact to concisely describe the trajectory of a random walker it is necessary to exploit the regularities of the patterns such a random walker performs during its walk.
In this setting a codebook that associates each node of the network to a code is necessary. The length of the code is such that it is shorter if the node tends to be visited more often, so when encoding the walker's trajectory less bits will be necessary. This, however, is not enough to have the shortest possible code.

The intuition to follow is that once entered a community, a random walker tends to spend some time inside the community (because of the relatively sparse nature of inter-community edges), before eventually moving to another community and so on. So it would be convenient to have two levels of codebooks: one index level that encodes the community in which the random walker is, and one level made of as many codebooks as the number of communities. This way once a symbol that univocally describes a community is found, all the following nodes are encoded according to the codebook corresponding to its community, obtaining eventually a code formed by much less bits than in the single level variant. With respect to figure 2.1, (a) represents the flow network and a random walk, (b) is a single-level

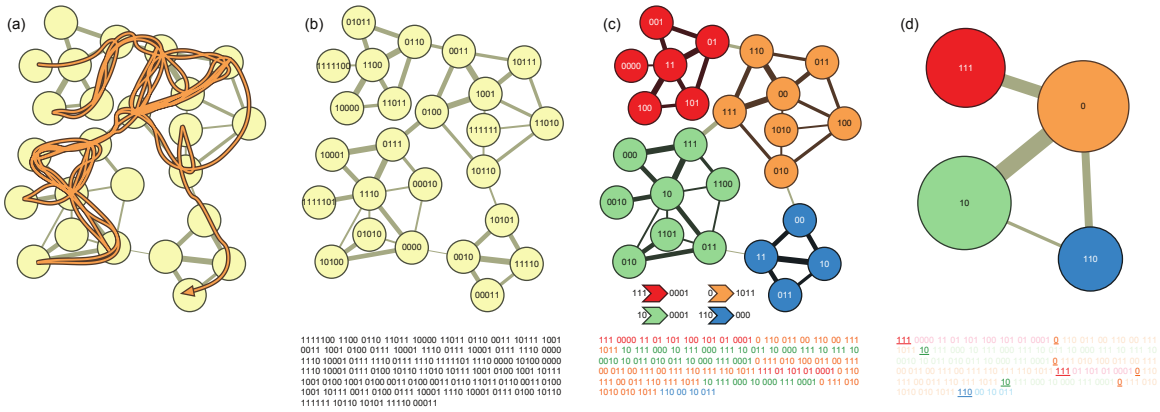


Figure 2.1: (a) A random walk in a flow network. (b) Single-level encoding of the random walk. (c) Two-level encoding of the random walk. (d) Detail of the first level code-book. (From [RAB09])

encoding of the nodes in the random walk, (c) is a two-levels encoding of the same walk, (d) highlights the stages in which the random walk goes from a community to another and the relative appearance of the communities' codes in the encoding. In the example it is possible to notice how the code in (c) is shorter than the one in (b).
This is why the problem of minimizing the encoding of the random walker trajectory is equivalent to the problem of finding the best partition of the network with respect to the flow.

To answer the question to what is the best partition, Infomap makes use of the map equation [RB08], which specifies the theoretical upper limit about how concise the optimal code would be for any given partition (without actually having to describe the codebooks). Minimising this equation over the possible partitions of the network is equivalent to discovering the community structure.
They derive a functional form of the map equation that assigns a score to a given partition over the network. Therefore any existing algorithm that optimizes an objective function over the possible network partitions can be adapted to minimize the map equation.

2.4 Oslom

Community detection algorithms provide as an output a division that is supposedly representative of the community structure underlying the graph at study. However the found structure could just be the result of random circumstances.
In this sense not much has been done in the research to understand whether these divisions are statistically significant or not. Some researchers initially proposed to use modularity function based techniques to assess this significance, but they were found to be not adequate, while only recently were proposed new techniques based on the perturbation of the graph [MD06, KLN08].

Lancichinetti et al [LRR10] devised a method to assess the significance of just a *single* community that doesn't rely on the modularity function. Their method does not compare the outcomes of an algorithm against *averages* over the null model, but against the "best"

possible outcome in the null model. It thus makes use of extreme statistics[1] concepts.
In this setting, the statistical significance of a community is given by the extreme probability of finding an equivalent (or better) community in a set of random graphs drawn from the null model. The null model they adopt is such that all the links inside the community stay the same, while the others are randomly rewired but keeping the original nodes' degrees.

Let k_i^{int} be the number of links that connect node i to the community, and let k_w^{int} be the internal degree of the worst (i.e. has the lowest internal degree) node w in the community, then assuming that the community detection algorithm aims to maximise k_i^{int} within a community, they define the *C-score* as $c = Pr(k_{w'}^{'int} > k_w^{int})$ which represents the extreme probability that the worst node w' in a random graph has internal degree greater or equal than the one in the community at study. Notice that for now the ranking criteria to pick the worst node was simplified. The actual ranking criteria is further discussed in section 5.4.
If such a probability is low (less than 5%) then the community can be considered significant.

However considering only the worst node as a comparison criteria can be too strict sometimes, bringing to the problem that if a community detection algorithm misclassifies just one node of the community, then the *C-score* can grow too much indicating the community as not significant. To avoid this problem the formulation of the *C-score* can be adapted to take into account a list of worst nodes (the "border" of the community), rather than only one. This reformulation is called *B-score*, and it was experimentally proven to be more accurate than the *C-score*.

Once defined what is to be regarded as a significant community, the same authors came up with OSLOM [LRRF11], a new community detection algorithm that make uses of the above concepts. OSLOM can handle very complex networks, including directed and weighted ones. It locally optimises the B-Score fitness function.
At a high level OSLOM defines the following tasks:

- **Single Cluster Analysis**
  Let C be a generic cluster of nodes. Nodes outside the cluster may actually be significantly linked to it, while nodes inside may be not significant. During the single cluster analysis, all the nodes outside the community are ranked according to their statistically exceptional correlation to the community. The best nodes are then considered for inclusion in the cluster. Starting with the best node, it is included into the cluster and the local fitness score is computed for the resulting cluster. The procedure goes on as long as the score is above a certain tolerance value.
  Once all the best nodes are included, the cluster is searched for the worst nodes for eviction. A similarly symmetrical procedure is thus adopted.
  In summary the single cluster analysis optimises the community's significance as defined by the B-Score by inclusion and eviction of nodes.

- **Network Analysis**
  Single Cluster Analysis *cleans* a given cluster, but remains the question of how to come up with the clusters to process. During each iteration of the Network Analysis

[1] a part of the statistics theory that deals with extreme cases

phase, a single vertex is randomly extracted and the cluster $C = \{i\}$ is considered. Then a random number of *most significant* neighbours of i is added to the cluster C. At this point the single cluster analysis can be performed on such a crafted cluster. This process is repeated for several random vertices i, and it is stopped when very similar clusters keep on being obtained.
This phase delivers a set of clusters, each of which optimises the fitness score.

- **Check For Union**
  All the possible pairs of clusters are fused and their fitness score is computed. If there is a gain into doing so, then the pair becomes a single cluster.

This procedure eventually delivers a partition of the network in which each community is a statistically significant cluster that minimises the B-Score.

In this chapter were covered both the importance of the information given by direction of edges, and the techniques that aim to exploit it. Not much has been done in this sense in the history of community detection, however these recent results highlight how in some cases it is important to keep direction in account. The concepts that form the basis of the first two algorithm based on modularity, plus the algorithm based on community significance, will be used to develop hiding strategies. Infomap and Oslom will be used (together with a CD algorithm based on Extremal Optimisation ,see section 1.5) for testing purposes.

Chapter 3

Rationale and related work

Here I will discuss what are the possible reasons why hiding from community detection may be desired. I will talk about a recent study about attack tolerance in graphs with community structure, then about a previous work that treated the hiding from community detection, pointing out how my work is different and what are the innovations that I bring.

3.1 Why hide a community from detection

The identification of the communities in a graph is of great interest, both because they represent functional building blocks within networks and because they provide insight about the internal dynamics and the processes that determine the shape of the graph. For example in a web graph, a group of web pages may be related to the same topic [FLGC02], in metabolic networks it is easier to visualise in a modular way the dynamics between groups that perform different functions [GA05, HH03], in a social network a group might correspond to some social unit [GN02].

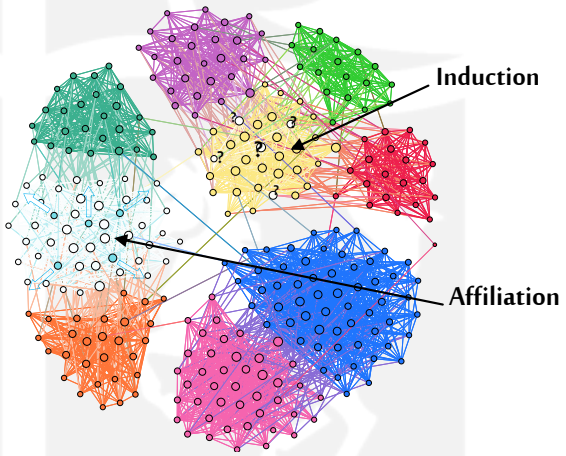


Figure 3.1: Visualisation of affiliation and induction scenarios. (Generated with Gephi [BHJ09])

However there are situations in which community detection may be undesired. These may include scenarios in which privacy or security are of primary importance to the entities belonging to a community. There are at least two ways in which community detection can be used in a way that can represent a threat to privacy and security. I will refer to these as *induction* and *affiliation*. Assume a third party has possession of the graph representation for any given reality of interest, she can then obtain the division in communities of the network using a community detection tool. If by any other means she knows that in a given community the majority of entities has a set of common features then by *induction* she can assume that also the remaining entities in the community, for which she wasn't able to

gather directly those informations, has indeed the same set of common features. For example imagine a social network in which the majority of members of a certain community are openly homosexual, thus by induction it may be inferred that also the remaining entities of the community are homosexual as well, even though for privacy reasons they may have chosen not to disclose this information. This can extend to religious belief, political preference and other sensible informations.

As another side of the same coin, if the third party knows that some entities belong to a certain functional unit, then she can infer that the *affiliation* of the other entities to the same community implies belonging to the same functional unit. For example imagine that some indivuals are known to belong to a rebel cell, in this case it may be possible to place these individuals in a communication network and identify which is the community that includes them. Then the third party may infer by affiliation that the other individuals in that community actually belong to the same rebel cell, find the identity of these individuals and persecute them.

There are many ways in which a third party may get possession of network data. Some of these include traffic analysis or direct access to communication databases (i.e. the third party carrying on the attack may be a huge provider of email services). Traffic analysis is an intrusive practice that puts privacy at high risk. Anonymous communication channels are in continuous arm-wrestling with traffic analysis to protect the legitimate privacy of the end points of a communication. Companies carry on traffic analysis for the sake of market researches, governments do it to apply censorship or persecute individuals. Eventually their actions result in the violation of the privacy of a huge number of individuals. In at least one documented case governments and companies also colluded in order to perpetrate the traffic analysis: in 2006 AT&T's Daytona system (a storehouse of telephone call records) was suspected to be the mean that NSA used to mine records without a warrant [Fou].
Therefore there is a need for some manipulation strategies that could help a community to hide itself against community detection.

There are some considerations to make and take into account when devising the strategies to adopt. Firstly if a party wishes to hide a community it means that somehow it is part of that community and may have the power to apply selective graph topology manipulations within the community, while at the same time it may have no power to alter anything that doesn't touch the community directly, thus the strategies should only include addition or removal of features that are local to the community. Secondly, even if granted total powers over the community, it may not be possible to alter it without any limit. In most cases it is hard, if not impossible, to create or destroy vertices, while it is certainly more feasible to hide or create interconnections among them. For example if the nodes are people it may result hard to *delete* or *create* a person. Creating or hiding edges that run among individuals are instead feasible actions, as for example to hide or create a link in an email network one may either never send an email or send many to a given individual. There are also more sophisticated communication protocols that can let entities achieve the *concealing* of an edge without actually interrupting the communication, like for example TOR[1] [FJSA07], which

[1]The Onion Router is an anonymity network of volunteer servers that conceals users' connections from anyone conducting traffic analysis. A conversation is created by chaining a number of nodes together, between a source and client, and no single node in the chain knows about both end points of the real communication. Each message along the chain is encrypted so that an attacker won't know whether the

is a communication protocol that provides anonymity for the end point users of a communication (i.e. an adversary observing messages in the TOR network can't trace source and destination of any message, thus can't observe an edge between them). As another example, consider a study about online Blogging Communities [CX07] in which hate groups were detected. The nodes are blogs and the edges are web links between the blogs, then for the "hating" community it may be undesired the deletion of a blog, while it may be much easier to *conceal* a web link (e.g. rather than writing *"http://www.otherblog.domain"* it may be enough to rewrite it as *"otherblog dot domain"* to hide it from web-spiders and crawlers), whereas to *create* edges, one may just randomly put a link to some other blog (e.g. write somewhere in a post *"do not visit http://www.otherblog.domain"*, so that a crawler will find the link and create the edge).

In this thesis work I will devise what are the actions that the elements of a community should take in concert in order to achieve the best possible hiding from community detection. These actions are however subject to feasibility and strongly depend on the actual scenario. For this reason I devised several techniques that give different degrees of accuracy, that depend on feasibility, on the amount of information available to the community and their computing resources.

3.2 Attacking the community structure

To the best of my knowledge, and apparently also the one of Cheng et al [CLDF10] not much attention has been paid in the literature with respect to the effects of attacks to the community structure present in a network.
In this section I will talk about what maybe is the first of these studies. It is important to notice that the following attack methods differ with respect to the objective of my thesis as what Cheng et al explore here is how different the community division appears after edges in the whole network are potentially tampered with, whereas my work wants the hiding of a given community while being able to operate only local manipulations.

Cheng et al present two targeted methods to alter the community structure. Their main focus is on the number of triangles[2] z_{ij}^3 built on the edge from node i to node j, and the derived edge-clustering score C_{ij}^3 defined as

$$C_{ij}^3 = \frac{z_{ij}^3}{min[(k_i - 1), (k_j - 1)]} \tag{3.1}$$

which represents the ratio between triangles built on one edge and how many could be built on the same edge.
They draw from the intuition that intra-community edges tend to be in triangles, thus having high clustering coefficient. Therefore their proposed attacking strategies would focus on two related targets: in one those edges with high clustering C_{ij}^3 (EC-method) are targeted, and in the other edges with high z_{ij}^3 (T-method) are targeted.
In both methods a fraction of edges, chosen in descending order according to C_{ij}^3 and z_{ij}^3 scores, is removed and randomly rewired. However these fractions refer to different sets. In

message comes from the originator or any other intermediate point.

[2] a triangle is a set of three nodes, all connected to each other

the EC-method any edge can be subject to rewiring, in the second only those edges which belong to a triangle. Unlike other perturbation methods [KLN08] this rewiring doesn't keep the original nodes degree, however it keeps unchanged the total number of edges $|E|$.

To assess these methods they compare them to the effects of a totally random rewiring of the edges.
Dissimilarity between two sets A and B can be seen as

$$d_{AB} = \frac{|(A \cap \bar{B}) \cup (\bar{A} \cap B)|}{|A \cup B|} \tag{3.2}$$

To measure the effects of the perturbations they define a dissimilarity score as follows

$$D = \frac{\sum_{i=1}^{k} d_{X_i Y_i}}{k} \tag{3.3}$$

where k is the number of communities found, and X_i and Y_i represent the $i-th$ community of two distinct divisions, and $D \in [0, 1]$.
The attacks are then tested on some computer generated benchmarks, as Girvan-Newman networks (GN-networks) and LFR-networks (see section 4.1). They chose Extreme Optimisation [DA05] as an algorithm for community detection. Their finding is that for both attacks the dissimilarity function departs from the one due to random perturbation in most cases.

In this section was discussed a work quite close, but still different under many aspects, to the one I am approaching. The attack methods are supposed to alter the community structure present in a network by targeting and rewiring those edges that are likely to be important to the communities. The finding was that targeting those edges that participate into triangles led to some results. Their choice to consider only one CD algorithm as benchmark may be inappropriate, this is why I opted to use three CD algorithms for my tests.

3.3 Covert community hiding

In 2008 Nagaraja presented a technical report about a model of surveillance based on the detection of community structure in social networks [Nag08]. He studied the amount of surveillance on the network an adversary needs to place in order to gather enough intelligence about a target community, and what are the counter-surveillance strategies such a community should adopt to protect itself from detection. This work is thus very close to the one I am proposing here, however some distinctions and observations will be pointed out further in this section to set the two works apart.

The setting in which Nagaraja carried on his study is that of undirected weighted graphs. He picked a communication network of emails within a university, and identified two macro communities using a spectral community detection algorithm based on maximisation of the modularity function. He picked the smallest of the two communities to be the target covert community while he referred to the other as the *mainstream* community.
The assumption is that once a node is put under surveillance then intelligence can be

gathered about the node itself, its incident edges and its neighbouring nodes. The nodes to be put under surveillance are chosen to be those with either largest betweenness score or weighted degree score. In the first case the adversary must be aware of the topology of the whole network (in order to compute the betweenness scores), in the second she only needs to be aware of the amount of information that passes through each node.

Nagaraja showed how putting under surveillance different percentages of nodes in this network leads to the knowledge of what percent of the whole network and knowledge of what percent of the target covert community. The result of the application of the two surveillance techniques is that it is required of the adversary to put a much larger number of nodes under surveillance to get comparable levels of knowledge of the covert community in the case of largest degree nodes than in the one of largest betweenness.
The study was then extended to take into account the fact that a covert community may want to take some actions to counter the effects of the surveillance over the network. The counter-surveillance hiding strategies proposed are based on the creation of an edge between a node in the covert community and one in the mainstream community. The nodes are picked from each group based on one of three criterion: random node, node with highest betweenness, node with highest degree. Some of the possible combinations of these critierion were tested in the setting at study. His results showed that creating edges between high centrality nodes in the community and random nodes outside gave the best results among all the strategies.

Nagaraja's work differs with the one in this thesis in many ways. Firstly his study considers undirected weighted networks whereas this thesis work considers directed unweighted ones. The hiding strategy he proposed are only based on inter-community edge addition, while the manipulations I allow are both addition of inter-community edges and deletion of intra-community ones. More over his strategies are based either on random selection or centrality measures, whereas I explored also some more sophisticated techniques based on recent studies of statistical significance. For his experiments Nagaraja used only one Community Detection (CD) algorithm, applied his techniques to a real network (for which the community structure was not known a priori but found using the same CD algorithm), considered only two macro communities, and provided no interpretation for the obtained data. In my work I will apply my techniques to several networks with different features and the results will be tested against three CD algorithms based each on a different theory. More importantly I will of course provide an as accurate as possible interpretation of the results.

In this chapter I covered a few aspects that are closely related to my thesis. I started by talking about the reasons that may lead to desire the hiding of a community from detection in a graph, and what is the manipulation model I allow in this work.
My aim is to alter a community's structure so to make it results as if it was not a community any more. Thus I talked about the effects of targeted attacks to graphs' community structure, then moving the focus to single community structure with Nagaraja's technical report. These two studies are in line with what was my decision about allowing only deletion/creation of edges, however my remark is that the innovation in my work is in the fact that I am allowing only local access to entities in a community, I work in the directed

graphs setting, and I explore ways to exploit recent results about community's statistical significance.

Chapter 4

Development and evaluation tools

In this chapter I will talk about what are the software tools I either developed or used drawing from resources made available by the research community. I will also talk about the benchmark graphs I will use to measure the fitness of the manipulation techniques. The way I will take this measures will be defined itself in this chapter. Lastly I will highlight what are the problems that are likely to put a limit to the goodness of the manipulation strategies.

4.1 Benchmarks

For the scope of testing the goodness of a found division, many benchmark graphs have been proposed in the literature. They divide in two categories: real world graphs, for which the community structure is known a priori, and computer generated networks in which the community structure is present by construction. While the latter are more suitable to testing as they represent a reproducible and controlled environment, the former are equally of interest as they relate to real applications.
In this section I will talk about Girvan-Newman's benchmark, LFR's benchmark networks and some real world networks about blogging communities.

Girvan-Newman's benchmark

GN-networks were firstly introduced in [GN02]. These are graphs obtained using a particular configuration of the planted l-partition model [CK01]. The model generates a graph partitioned in l sub-graphs of g nodes each ($n = g \cdot l$). Two nodes within the same group are linked by an edge with probability p_{in} , therefore each sub-graph on its own constitutes a random graph. Furthermore nodes in different groups are linked with probability p_{out}. The expected degree $\langle k \rangle$ of each vertex is

$$\langle k \rangle = p_{in}(g-1) + p_{out}g(l-1) \tag{4.1}$$

It results obvious that if $p_{in} > p_{out}$ then there are more intra-group edges than inter-group ones thus community structure is present by construction.
The configuration adopted by Girvan and Newman is $g = 4$, $l = 32$ and $\langle k \rangle = 16$ resulting in a graph of 128 nodes, divided in 4 communities, and from equation 4.1 derives

$p_{in} + 3p_{out} \approx 1/2$.
When this benchmark is adopted, it is common use to refer to the terms $z_{in} = p_{in}(g-1) = 31p_{in}$ and $z_{out} = p_{out}g(l-1) = 96p_{out}$, representing respectively expected in-degree and expected out-degree.
One would expect to be able to detect communities for all those GN-networks such that $p_{in} > p_{out}$. From equation 4.1 $p_{in} = p_{out} \Rightarrow p_{in} = 1/8$ having that $z_{out} \approx 12$. So equivalently one would desire a good community detection accuracy for $z_{out} < 12$.

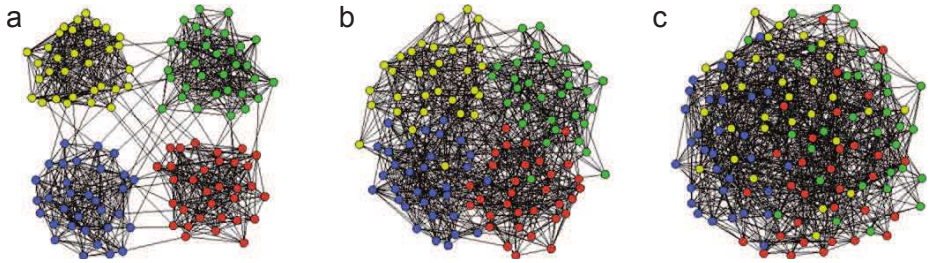


Figure 4.1: Three different configurations of GN-networks(From [For10]). They show the differences between graphs for decreasing values of z_{in}. In the leftmost the communities are easily detectable, whilst in the rightmost the division is no more distinguishable, making the graph more similar to a random graph.

Lancichinetti-Fortunato-Radicchi's benchmark

Even though a good algorithm should definitely be able to identify the community structure in GN-networks, these benchmark networks alone are not the ideal for testing. Lancichinetti et al [LFR08] highlight how the GN-networks do not resemble real networks in that nodes have approximately all the same degree and the communities have all the same size. More over networks of 128 nodes are too small for some algorithms that can actually handle millions of nodes. To better resemble the properties of real networks, the heterogeneity of node degrees and community sizes must be taken into account. Thus they introduced a software tool that can create benchmark graphs where both node degree sequence and community sizes are drawn from power law distributions. These properties are often observed in real world network, thus making these benchmark graphs a much more suitable reference.

One can produce Lancichinetti-Fortunato-Radicchi networks (LFR-networks) to have N nodes, and the presence of community structure is controlled via the mixing parameter $\mu \in [0, 1]$: each node has a fraction μ of edges linking to nodes in other communities, while the remaining fraction $1-\mu$ of edges link to nodes in the same community. Thus the smaller is μ then the more well defined is the community structure. Besides being more realistic, these graphs represent a much harder challenge to community detection algorithms, and are capable to highlight their limits.
In [LF09] they extend their tool to create more complex graphs with community structure. These new graphs can be directed, weighted and present overlapping communities. To generate directed graphs they treat separately in-degree and out-degree sequences, and theoretically it is possible to have one mixing parameter for each, however for simplicity

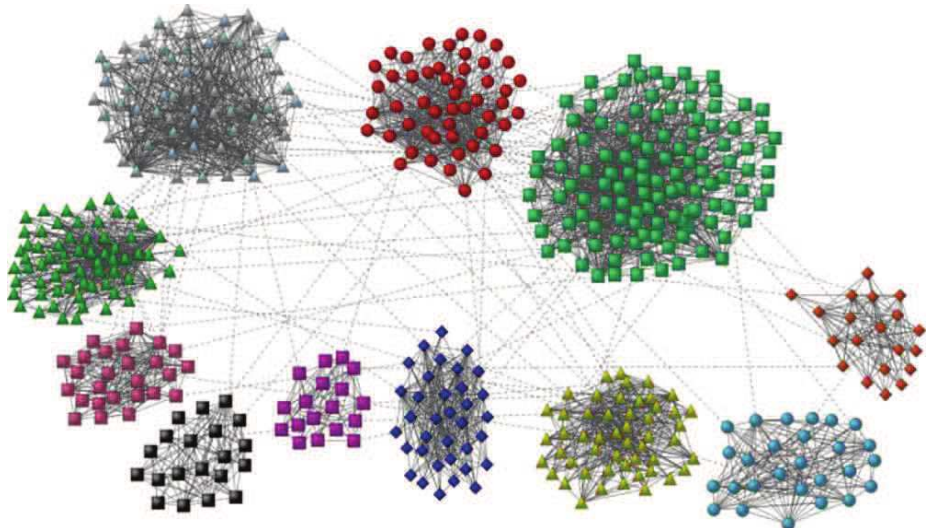


Figure 4.2: An example of LFR benchmark (From [LFR08]). The communities sizes and nodes degrees are not all the same, but drawn from power law distribution.

they are set to the same value.

Postlink and Unionblog networks from University College Dublin

These real world networks represent graphs about 614 blogs. and were used in a cultural study of the Irish blogosphere [KWLAC11]. In these graphs the blogs are represented as nodes, and the edges are mapped according to two different criterion: in Postlink there is an edge going from a blog to another if the former has hyperlinks that point to the latter occurring within the HTML content of its blog posts, in Unionblog there is an edge from a blog to another if the first recommends the second in its blogroll.

4.2 Jaccard similarity and entropy indices

An appropriate comparison metric needs to be adopted to measure the accuracy of the hiding techniques. It must measure how similar is the original community to the community identified after the manipulation strategies are applied to the network. For this purpose I chose to adopt the Jaccard similarity index [TSK05], which is a statistic used to measure similarity between sets. It is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{4.2}$$

where A and B are two sets, the numerator is the number of elements the two sets have in common and the denominator is the total number of elements belonging to either set. The Jaccard index $J(A,B)$ has values in the interval $[0,1]$, where 0 indicates that the two sets have nothing in common and 1 indicates that they are identical.

After the manipulation is applied, to measure the accuracy of the method the similarity is computed as

$$similarity = \arg\max_{C} J(T, C) \tag{4.3}$$

where T is the original target community, C is any possible community found by the CD algorithm after the application of the alterations (i.e. the similarity is the largest Jaccard index score between the original target community and any community detected in the altered network).
The jaccard index is appropriate to measure how similar two sets are, but it fails to indicate to what degree the nodes of the original community get scattered around when they get misclassified as part of other communities (ideally it is better to have the misclassified nodes to be *divided* between as many other communities as possible, rather than end up all to be included to the same community) . To measure this effect I will use the concept of entropy [Mit97] from information theory. Entropy will come in handy when comparing two techniques that show very close similiarities, as it may reveal which of the two scatters nodes around the most.

The entropy function characterises the impurity of the classifications of a collection of items. The basic form of entropy can be described with the following example. Given a collection S in which its items can be classified with a binary attribute (e.g. true, false), then the entropy is defined as

$$entropy(S) = -p_T \log_2 p_T - p_F \log_2 p_F \tag{4.4}$$

where p_T is the fraction of items classified with *True*, and p_F is the fraction of items classified with *False*.

As figure 4.3 shows, the entropy function has value 0 when all the items have the same classification[1] (i.e. when the fraction p_T of items classified as True is either 0 or 1), while it has maximum value 1 when half the items are classified as True and the other half as False (maximum spread).
This function can be extended to take into account an arbitrary number c of possible classifications of the items in the collection:

$$entropy(S) = -\sum_{i=1}^{c} p_i \log_2 p_i \tag{4.5}$$

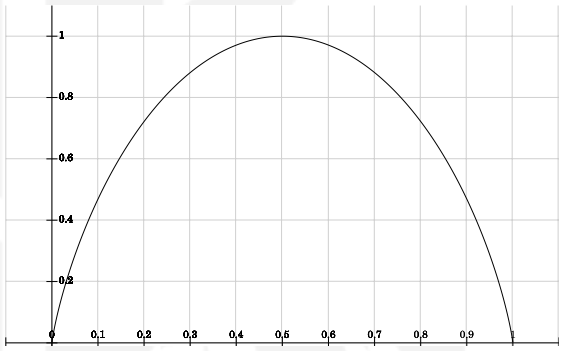


Figure 4.3: Plot of the entropy function. On the Y axis the entropy score, on the X axis the p_T fraction.

where p_i is the fraction of items classified as i. This function is no more confined to the interval $[0, 1]$ (the minimum is still 0).

In my case the entropy function is computed considering the nodes from the original partition as the collection S, while each classification i is the community to which the node belongs after the hiding technique was applied. The higher the value of the entropy the

[1] there is to specify however that even though $\log_2 0$ is undefined, in this function it is instead considered to be 0

more scattered is the division of the nodes between the communities. Only the plots in chapter 6 will report the entropy score (as in most cases low similarity corresponds to high entropy), because it adds value to the interpretation of some results in which two methods show similarity curves that are very close.

4.3 Experiments setup

To provide the most general results and a fair evaluation of the manipulation strategies I developed, I chose to test my hiding techniques against three community detection algorithms. These are Infomap, Oslom and Extremal Optimisation (see sections 2.3, 2.4 and 1.5). I chose these three algorithms due to the high degree of accuracy and their popularity in the research field, but also because they are based on uncorrelated theories (information theory, community's statistical significance and modularity score maximisation respectively). This choice should bring to an analysis of the experimental results that highlights to what degree the accuracies of the hiding techniques do not depend on the particular CD tool an adversary picks.

The manipulation techniques were tested on artificial networks drawn from the LFR benchmark (see section 4.1), and different parameters were used to create them. The size of the networks used for testing was limited by the fact that a single run of a CD algorithm or manipulation technique requires a considerable amount of time, growing too much with the number of nodes and edges in the graph.
Each community in the obtained benchmark graphs is manipulated with the developed techniques. The similarity between the original communities and the communities detected after the alterations took place is measured every 15%[2] of intra-community edges deleted in the interval between 0% and 90%, while for the testing of edge addition techniques the measures were taken every 25% of inter-community edges added (with respect to the number of intra-community edges) in the interval between 0% and 200%.
The results shown in chapter 5 come from the application of the hiding techniques to graphs generated with the following set of parameters: graph order $N = 500$, average in-degree $k = 25$, maximum in-degree $maxk = 50$, mixing parameter $\mu \in \{0.2, 0.4, 0.6\}$. The similarity and entropy scores are presented for each CD algorithm adopted to identify the target community. It is important to point out that the scores in the plots represent an *average* over all the communities in each network (each experiment was run 5 times on each single target community). Every technique is compared against a random addition/deletion of edges (I will refer to this strategy as RAND from now on).

To provide a test of the fitness of the manipulation techniques, they were also tested against the real world networks presented in section 4.1. In chapter 5 are shown the results of the application of the manipulations to the Postlink graph. Those of Unionblog graph are shown in appendix A.
While the tests against LFR benchmarks are carried in a controlled environment in which the real partition in communities is known a priori, for the real world networks no ground

[2]I picked a stride of 15% of edges rather than an expected 10% because as already mentioned both the CD algorithms and the hiding techniques take a lot of time for single execution

truth is available[3]. To carry on the tests I adopted as ground truth the division found by each CD algorithm, and I applied the manipulations to the delivered communities (i.e. when testing against Infomap, the results were compared against the division found by Infomap; similarly for Oslom and Extremal Optimisation). Another thing to point out is that some studies showed how very often the data collected may result to be incomplete, thus affecting the outcomes of the experiments [YG11]. For these reason the experiments with LFR should represent much better the fitness of the developed work, while the same degree of confidence can't be granted with real networks.

4.4 Software tools

The hiding techniques devised in chapter 5 were implemented entirely in C. This also includes auxiliary routines like the computation of the similarity scores, the r_i probability scores (see section 5.4) and the conversion between different graph file formats. The testing and harvesting of data was massively automated using bash scripting. The data collected was then organised and stored in a MYSQL database.

The igraph [CN06] C library played a central role in the development of my software. It is a software library for the study and manipulation of graphs. It provides a rich set of functions that for example allow to create graphs, query their properties or alter their features. It also implements many community detection algorithms. However these CD algorithms, with the exception of Infomap, only work with undirected graphs.
The implementation of Infomap [RAB09] present in igraph is the one that was adopted during the testing. Oslom [LRRF11] was also used for the testing and it is freely available at http://www.oslom.org/. Radatools [GFBHA11] is a set of tools for the analysis of complex networks, and includes software for communities detection. The CD algorithm based on extreme optimisation [DA05] is part of the radatools suite. Radatools is freely downloadable from Sergio Gomez's personal web page http://deim.urv.cat/~sgomez/index.php.

A visualisation of the application of the hiding techniques can be found in appendix B. The visualisation was realised with the use of the network analysis tool Pajek [BM02]. Pajek reads *.net* files, which hold formatting informations about nodes and edges in a graph (e.g. their position and color). It also accepts *.clu* partition files representing the division in communities of the graphs. Pajek provides the user with functions that automatically place nodes belonging to the same community in circles so to aid the visualisation of the network. To visualise the evolution of the misclassifications at each step during the application of the hiding techniques I developed some code that keeps the nodes' original position (which reflect the original division in communities) but assigns colors to the nodes according to the newly detected division. This way one can quickly appreciate in a visual manner how the hiding technique influences the CD algorithms in their discovery of the communities. Another instrument I adopted for the visualisation of the graphs is Gephi [BHJ09] (an example graph visualised with Gephi can be found in section 3.1).

The artificial LFR benchmark graphs [LF09] are created with the software tool the authors provide on their personal web pages. It accepts some controlling parameters as input

[3] while in the undirected setting there is a plethora of labelled networks (like the classic Zachary network [Zac77]) I wasn't able to find any freely available labelled data in the directed setting

and produces networks that reflect the selected properties, including their exact division in communities. When testing the accuracy of the hiding techniques this division is taken into account to measure the similarity scores againts the divisions found by the CD algorithms after the manipulations take place.
The C-score and B-score (see section 2.4) of a community are computed using the tool that computes the statistical significance of communities in networks provided by Radicchi on his personal web page.

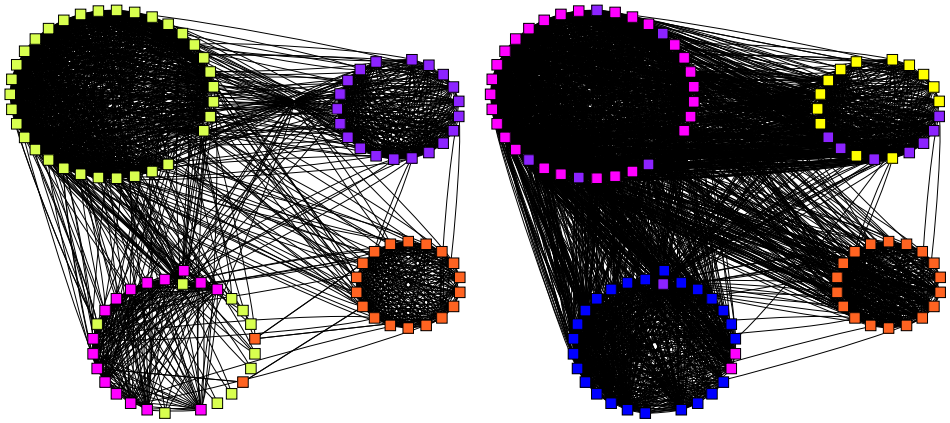
Lastly, since some graphs may account for dimensions of millions of nodes and billions of edges, some functions (like the binomial coefficient required for the calculation of the r_i scores, section 5.4) may grow way too much to fit in standard C data types. For this reason I adopted the GMP (http://www.gmplib.org/) library which allows complex mathematical computations as well as provides data structures to hold multi precision types.

4.5 Limits of the hiding techniques

Hiding a community from detection can be seen as a process that manipulates the network in a way that makes the original community look like if it wasn't a community any more. When adding or removing edges this must be taken into account. Therefore it is clear that the only edges to delete should be those within the community (intra-community edges), because removing those between the target community and other communities (inter-community edges) would have the opposite result of isolating the community even more, thus making it more easily identifiable. When creating edges, to better blur the borders between different communities, only inter-community ones should be added, otherwise adding more intra-community edges would have the opposite result of making the nodes in the community more densely connected.

The locality principle with which I chose to develop my hiding techniques leads of course to some accuracy limits. When deleting intra-community edges, what leads the CD algorithms to misclassify the nodes of the community is the set of inter-community edges. As a reference see figure 4.4a in which the bottom-left community is the target community to hide. Even though a large number of intra-community edges was deleted, the inter-community edges are the same as they were before: too few to lead CD algorithms to associate the nodes to other communities. When adding inter-community edges instead the problem is that the inter-community edges between *other* communities remain the same. Taking as an example the communities in figure 4.4b, the target community is the top-left one. Even though some of the nodes are misclassified, part of them are not. By visual inspection one can notice that there is a dense structure of links between pairs of communities in which one is the target community, while the inter-community edges structure between all other pairs of communities is as sparse as it originally was. A good CD algorithm should still be able to a certain degree to identify this structure of edges and correctly label nodes.
The first of the two problems can be solved by combining both deletion and addition strategies, however the second problem can not be solved with only local access to the network.

Finally there is to point out that in general after either manipulation every other community retains its intra-community edges structure (and partially its inter-community edges structure) thus having the CD algorithms to correctly identify them to a certain degree. Therefore the remaining nodes belonging to the target community suffer of a certain bias



(a) only intra-community edge deletion limits

(b) only inter-community edge addition limits

Figure 4.4: A visualisation of the limits of the edge deletion and addition manipulation strategies. Nodes in a circle represent the original communities, colors represent the communities identified after the application of the hiding techniques. (Made with pajek [BM02])

into being identified as belonging to the same community.

In this chapter I described the environment in which the experiments will be set. I showed how the accuracy of the techniques can be measured using a similarity function based on the Jaccard index, while I pointed out that the entropy function can be used to measure how the misclassified nodes spread around to many other communities.
Lastly I shown how the locality of the alterations puts some limits to the accuracy of the methods (one of which can be overcome by combining both deletion and addition techniques).

Chapter 5

Hiding techniques

The problem with community structure is that the definition itself of community is not quantitative, but just qualitative, so there is no objective criteria to say with absolute confidence whether a group of nodes is indeed a community or not. In fact, some of the most renown CD algorithms in the research field are non-deterministic, meaning that if used twice on the same network one may get two (ideally slightly) different partitions of the network in communities (this is also true for the CD algorithms I am using for the experiments). Therefore my aim was to capture what makes the essence of a community so to provide some manipulation techniques that would be as much independent as possible from the CD tools that an adversary may employ to identify the target community. However this process of extraction of the essence of the community started from the observation and understanding of the theories that form the basis of popular community detection algorithms.

In this chapter are discussed the manipulation strategies for the hiding of a target community from detection. A further caveat to give before proceeding concerns a restriction I imposed to the deletion techniques. When an intra-community edge is selected for deletion, such an edge is deleted only if doing so doesn't result into disconnecting the graph. This is because I want to allow the community to keep a certain degree of communication ability, rather than risking that there is no more possible communication flow between any two nodes.

5.1 Modularity based

A plethora of community detection algorithms are based on local optimisation of the modularity function (equation 1.2) introduced by Newman and Girvan [NG04]. As discussed in section 2.1 this function was adapted by Leicht and Newman [LN08] (equation 2.1) to embody information coming from the direction of the edges. I remind that their intuition is that the new formulation of the modularity function takes into account the fact that those edges that go from nodes with small outgoing degree to nodes with small ingoing degree represent a statistically surprising event. Thus ideally if one wishes to undermine the structure of a target community, he should remove or add edges that are statistically significant in this sense.

Deletion of edges

The weakening of the target community structure via deletion of edges significant to the modularity function can be achieved with the following sequence of steps:

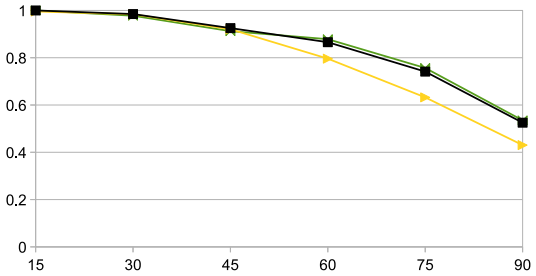
1. precompute the $k_i^{in} \cdot k_j^{out}$ score for each edge in the community that goes from node j to node i
2. delete edge with smallest score
3. update degrees of the end-point nodes of deleted edge
4. update product score for those edges that share an end point with the deleted edge
5. go back to step 2 until desired percent of internal edges were deleted

This procedure was tested against its symmetric counterpart (i.e. that targets those edges with the largest product score).
Community detection algorithms based on modularity Q_d maximisation at some point will evaluate the Q_d score assuming two nodes i and j belong to the same community. Now if there is an edge running from j to i this configuration adds $1 - \frac{k_i^{in} \cdot k_j^{out}}{m}$ to the Q_d score. Therefore removing those edges that present the smallest $k_i^{in} \cdot k_j^{out}$ corresponds to making null[1] those terms that most contribute to the Q_d score when the nodes i and j are considered as part of the same community. In other words it becomes a lot less convenient, with respect to the maximisation of the Q_d score, to consider i and j as part of the same community. Conversely, removing those edges with a large product $k_i^{in} \cdot k_j^{out}$, corresponds to zero those terms that contribute the least to Q_d when considering i and j as part of the same community having that the decision on the partitioning made by the CD algorithm is very little affected by this alteration.

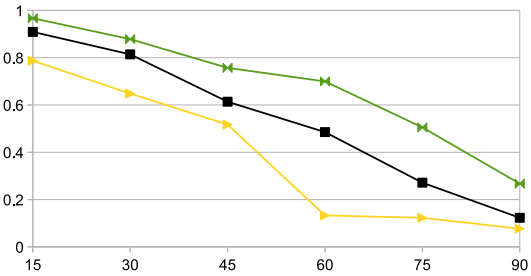
From now on I will refer to the method that targets edges with smallest $k_i^{in} \cdot k_j^{out}$ product as DESK (Delete Edges with Smallest K product), and the one that targets edges with largest product as DELK. The outcomes of their comparison are shown in figure 5.1.
When it comes to the real network DESK is significantly better than RAND against Infomap, while only slightly better against the other two, while DELK is always worse than RAND. With LFR artificial networks as expected DESK is (slightly) better than the random strategy RAND against Extremal Optimisation and Infomap, while DELK basically behaves as RAND against any CD algorithm. This is because the modularity function captures to a certain degree the statistically unusual features presented by the communities in a graph, reason why this function is very often taken as a reference point in the community detection research field. However, it doesn't provide any significant result against Oslom, which behaves exactly as it would have done against a random perturbation. More over it departs from RAND against Infomap and Extremal Optimisation only starting from 50% of deleted edges.

For this procedure to work the information necessary consists of the degree sequence of the nodes belonging to the community and the list of intra-community edges. No external topological information is necessary.

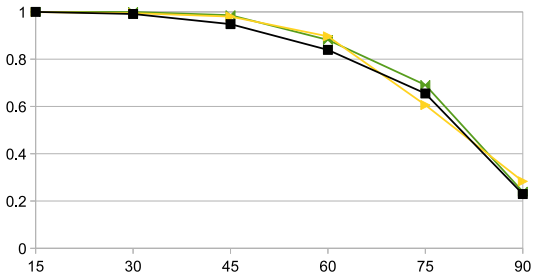
[1] if the edge is missing then the Dirac's delta in equation 1.2 becomes 0, thus making null the term



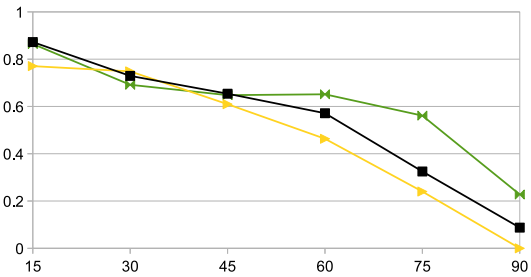
(a) Infomap - LFR



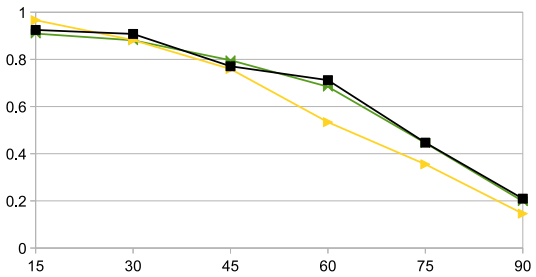
(b) Infomap - Postlink



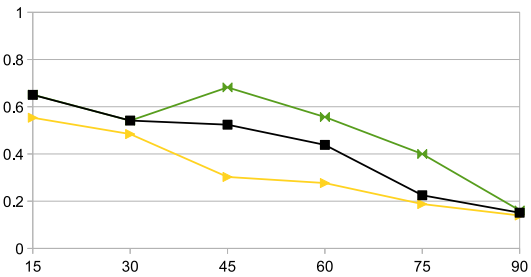
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.1: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. DESK is yellow, DELK is green, RAND is black.

Assuming the degree sequence of the internal nodes is given, then the precomputation phase in which the product scores for each edge are calculated is achieved in $O(E_{in})$ time, where E_{in} is the number of edges internal to the community (intra-community edges).
After the precomputation, in each iteration the minimum has to be found and the scores must be updated. Finding the minimum score is achieved in linear time with the number of intra-community edges $O(E_{in})$ using linear search (ordering this scores doesn't help as a single edge deletion may alter a variable number of such scores). The deletion of the picked edge between nodes i and j, and update of the end point nodes' degree are constant time $O(1)$ operations. At this point the product scores in which either of the end point nodes was involved needs to be updated. In the worst case scenario every single edge left shares an end point with the deleted edge, thus also the update has the upper limit $O(E_{in})$.
In summary both the precomputation phase and each iteration run in $O(E_{in})$ time. However there are two things to notice: in most real cases during each iteration not every product will need to be updated (as it is very unlikely that the end-point nodes are at the end of many other edges, as the reason they were picked in the first place is that they have small degree), and at each iteration the number E_{in} decreases by one. Therefore in reality the run time of each iteration is very unlikely to hit the $O(E_{in})$ worst case limit.

In this method, as well as in all methods to come, the computed scores are updated after each alteration. This is in line with Girvan and Newman's intuition (see section 1.4) that once a change is operated to the network, the old statistics do not reflect the nature of the network any more and thus should be recomputed to achieve better accuracy .

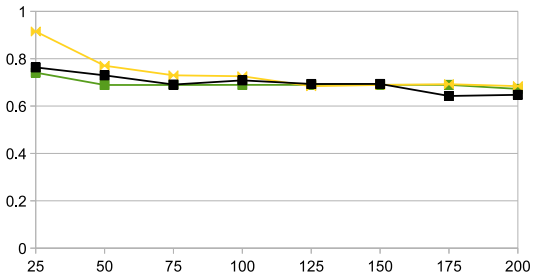
Addition of edges

To alter the shape of the target community structure via addition of edges significant to the modularity function, the following procedure is devised:

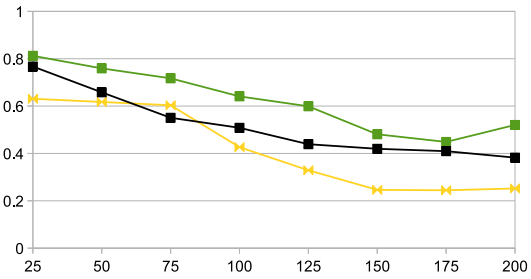
1. precompute the $k_i^{in} \cdot k_j^{out}$ score for each possible inter-community edge that doesn't exist already (nodes i and j are such that one belongs to the community and the other doesn't)
2. create the edge with smallest score
3. update degrees of the end-point nodes
4. update score of non-existing edges that share an end-point with the created edge
5. go back to step 2 until desired percent of inter-community edges were created

The symmetric procedure that creates inter-community edges with the largest product score was also tested. From now on I will refer to the method that adds edges with smallest $k_i^{in} \cdot k_j^{out}$ product as AESK (Add Edges with Smallest K product), and the one that adds edges with largest product as AELK.

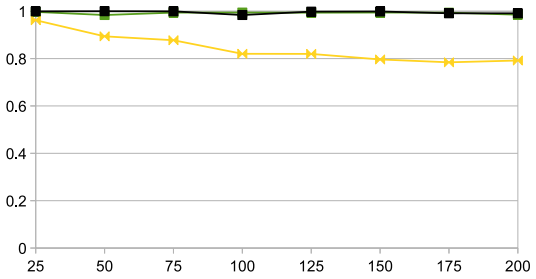
The creation of inter-community edges that minimise the degree product has the effect that when a community detection algorithm based on modularity optimisation is used, then considering the two nodes as belonging to the same community would give the largest possible contribution to the modularity score. Conversely if the inter-community edge with largest product is created, considering the two end-points as belonging to the same



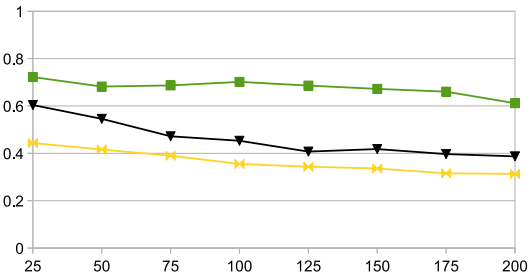
(a) Infomap - LFR



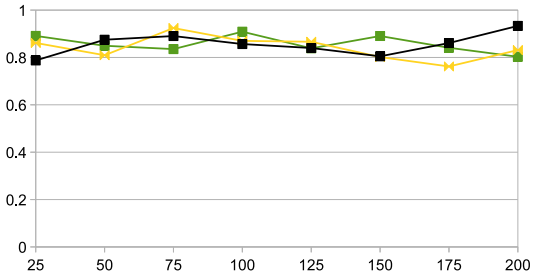
(b) Infomap - Postlink



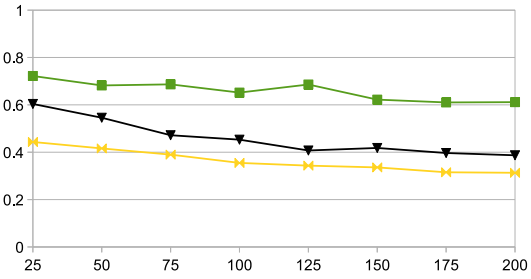
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.2: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of added edges on the X axis. Similarity is on the Y axis. AESK is yellow, AELK is green, RAND is black.

community would give the smallest possible contribution to the modularity score. Therefore it is much more likely that when applying AESK either node gets to be mistakenly considered as belonging to the community on the other end, while no change is likely to happen when using AELK.

The outcomes of their comparison are shown in figure 5.2. In the real network the manipulations show the expected results in all cases, having AESK similarity curve below RAND while AELK's is above RAND. Notice how in some cases the curves appear like constant functions (or linear with a very small slope), meaning that to achieve a small improvement a lot of alterations need to be applied. With artificial networks the trend is inverted with respect to the techniques based on edge deletion: both AESK and AELK behave like RAND against Infomap and Extremal Optimisation. Against Oslom AESK behaves like desired, while RAND and AELK keep the similarity very close to 1 no matter how many edges are added.

Unlike the deletion method, there is to notice that in this case to carry on the alteration, besides the list of already existing inter-community edges, it is also necessary to have information about the degree sequence of *every* node in the network. However no other topological information is necessary (e.g. the exact position of the edges that do not touch the community).
The analysis for the time complexity is very similar to the one for the previous technique. During the precomputation phase, the degree products for each possible inter-community edge that doesn't already exist need to be computed. The number of all possible inter-community edge is given by $2 \cdot N_{in} \cdot N_{out}$. N_{in} is the number of nodes inside the community, $N_{out} = N - N_{in}$ is the number of nodes outside the community, thus their product represent all possible pairs of nodes such that one is inside the community and the other is outside (the factor 2 represent the two possible directions of the edge between two nodes). In the worst case the community has one only inter-community edge, thus the precomputation phase gets an $O((N_{out}) \cdot (N_{in}))$ time upper bound.
Each successive iteration requires the same $O((N_{out}) \cdot (N_{in}))$ time to find the minimum product, while constant time is needed to create the edge and update the degree scores of the end-point nodes. The update of the product scores in this case requires a time different from the one of the precomputation. Only the non-existing edges with end points the selected nodes need to be updated: say an edge was created between internal node i and external node j, then the maximum number of inter-community edges that may have the internal node i as end point is N_{out} (all the nodes outside the community) while the maximum number of inter-community edges that may have the external node j as end point is N_{in} (all the nodes inside the community), thus the update phase has an $O(N_{in}) + O(N_{out}) = O(N)$ running time.
However since $N_{out} + N_{in} \leq N_{out} \cdot N_{in}$ for any $N_{out}, N_{in} \geq 2$, then $O(N) \in O((N_{out}) \cdot (N_{in}))$, so in summary both precomputation and single iteration have an $O((N_{out}) \cdot (N_{in}))$ time complexity.

5.2 Betweenness based

The algorithm that made the early history of community detection was based on the concept of edge betweenness [GN02] (see section 1.4). It identifies the edges in the graph

with largest betweenness as the loose connections between the communities (i.e. the inter-community edges). On the same line I observed what would happen to a target community if its edges were deleted based on the betweenness score.

Deletion of edges

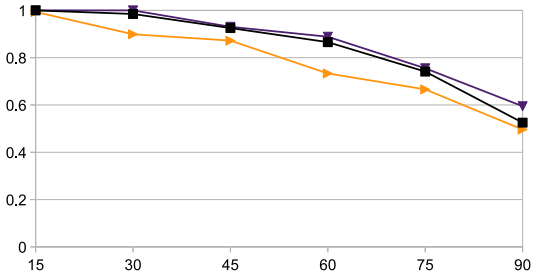
I implemented two symmetric procedures that work this way:

1. compute directed betweenness for each edge internal to the community
2. delete edge with largest (smallest) betweenness
3. go back to step 1 until desired percent of internal edges were deleted

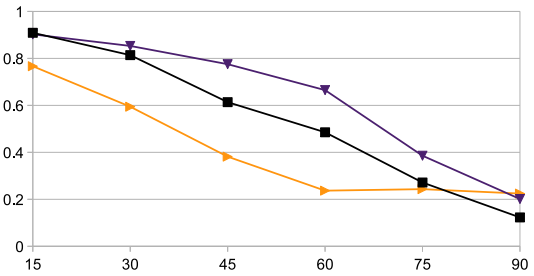
From now on I will refer to the method that targets edges with largest betweenness as DELB, and the one that targets edges with smallest betweenness as DESB. The results of this experiment are shown in figure 5.3.
Of the two methods, the one that seems to give the best performance overall is DELB. In fact in the real network DELB behaves like desired, while DESB stays above the curve of RAND. In artificial networks DESB's curve does never depart significantly from the random perturbation RAND, while DELB is only slightly below.
This effect is due to the fact that the internal edges with largest betweenness are *close* to the inter-community edges (which also have largest betweenness). Thus deleting such edges brings the nodes that make the *border* of the community to be loosely connected to the community, and thus the border gets misclassified, while the core of the community remains knit.

As observed in the artificial network the difference against the random perturbation is not much. A better way of using the concept of betweenness would be to consider node betweenness rather than edge betweenness. Since the inter-community edges are those with the largest betweenness, then the nodes touched by such edges must be characterised by a large betweenness themselves. Thus those nodes that instead have a small betweenness score must be those edges that are far from the *border* of the community, those that make its core. Therefore, to weaken the structure of a target community, it would be sensible to target those edges that run between the *core* nodes. A procedure based on this concept unrolls as follows:

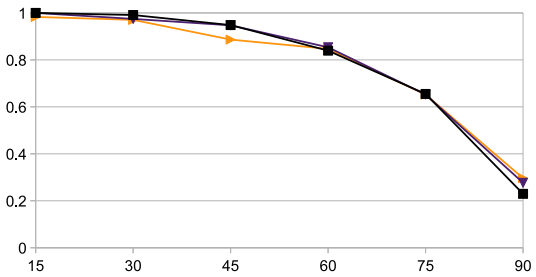
1. compute directed betweenness for each node
2. pick node j as the one with lowest betweenness (i.e. farthest from the border of the community)
3. for all edges that touch j assign to that edge a score equal to the betweenness of the node on the other side of the edge
4. delete the edge with smallest score
5. go back to step 1 until desired percent of internal edges were deleted



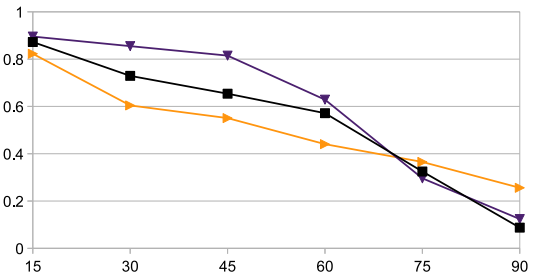
(a) Infomap - LFR



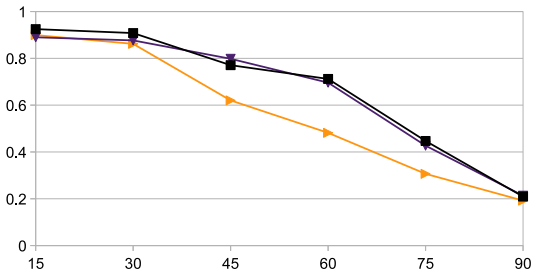
(b) Infomap - Postlink



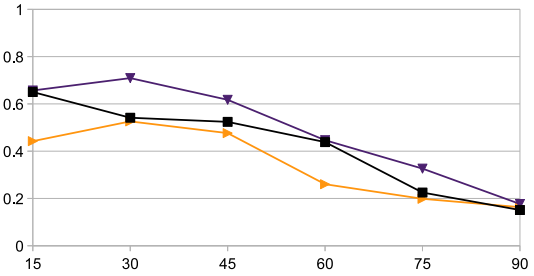
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.3: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. DELB is orange, DESB is purple, RAND is black.

This method was compared also against its symmetric version (i.e. that targets nodes with largest betweenness). Figure 5.4 shows the outcomes of their application. I will refer as DNSB to the method that targets edges between the nodes with smallest betweenness, as DNLB to the one that targets edges between nodes with largest betweenness.
In the tables one can observe that DNSB's similarity curve results to be pretty much the same no matter what is the CD algorithm used. Its shape is very linear and is tilted so that it quickly departs from the similarity curve of the random perturbation against Oslom and Infomap (against Extremal Optimisation the RAND and DNLB curves are a bit closer to DNSB). The DNLB method shows a similarity curve which in most situations follows that of RAND, or stays slightly above. These results are congruent with the prior thesis: the deletion of those edges that run between the least in between nodes weakens the community at its very core causing a linear decay of the similarity of the communities detected by the community detection tools.

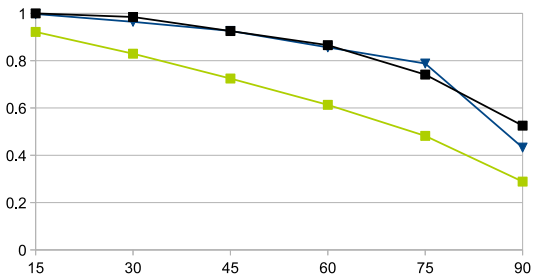
To use this technique it is necessary to be provided with the exact topology of the whole network in order to compute the betweenness scores. No precomputation can be carried in an advantageous way, as the deletion of a single edge can alter a variable number of shortest paths, and consequently a variable and unpredictable number of betweenness scores. At each iteration the betweenness scores of the nodes in the community need to be computed, and this requires an $O(N \cdot E)$ time (or $O(N^2)$ for sparse graphs) using the algorithm based on breadth-first search devised by Newman (see section 1.4). Identifying the node in the community with smallest betweenness takes $O(N_{in})$ time with linear search, and finding its neighbour with smallest betweenness takes at most another $O(N_{in})$ time. The deletion of the edge takes constant time. Overall each iteration takes $O(N \cdot E)$ time, since $O(N_{in}) \in O(N \cdot E)$, being $N_{in} \leq N$.

Addition of edges

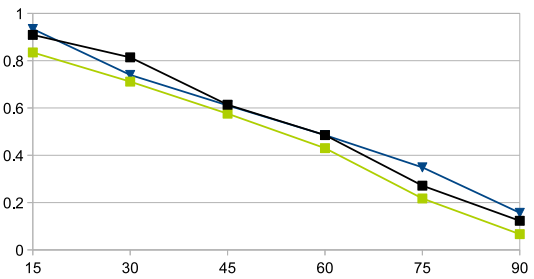
Many hiding techniques based on addition of edges that would take in account node and edge betweenness scores were tested. The one that gave the best results belongs to the family of procedures that follows.

1. compute directed betweenness for each node in the graph
2. select node in the community with largest/smallest betweenness
3. select node outside the community with largest/smallest betweenness
4. create edge between these two nodes
5. go back to step 1 until desired percent of internal edges were deleted

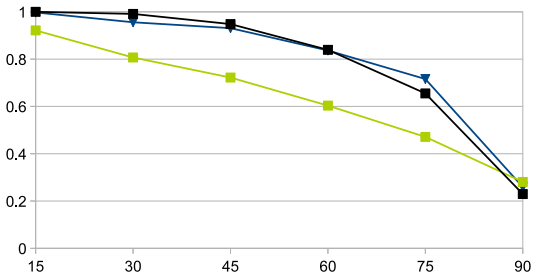
I will refer to these techniques as follows: AE-LBiLBo (Add Edge Largest Betweenness inside Largest Betweenness outside) as the one that creates an edge between the node with largest betweenness in the community and the one with largest betweenness outside the community, AE-SBiSBO as the one that creates an edge between the node with smallest betweenness in the community and the one with smallest betweenness outside the community, AE-LBiSBo and AE-SBiLBo represent the mixed strategies.



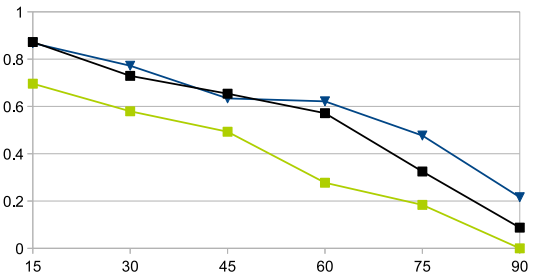
(a) Infomap - LFR



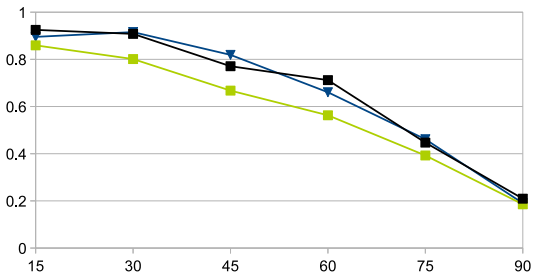
(b) Infomap - Postlink



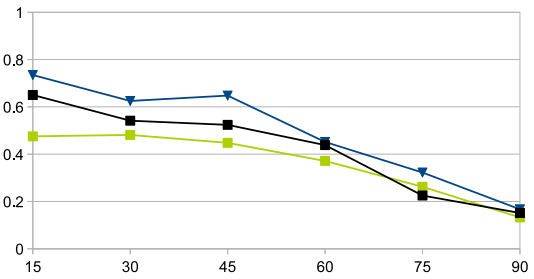
(c) Oslom - LFR



(d) Oslom - Postlink

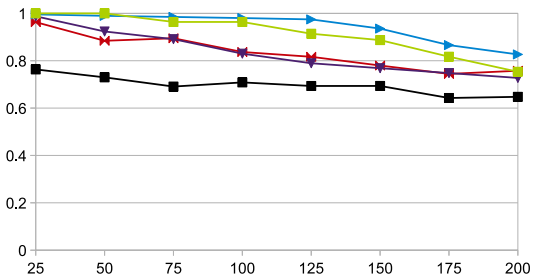


(e) Extreme Optimisation - LFR

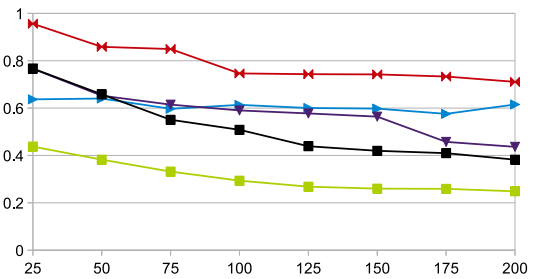


(f) Extreme Optimisation - Postlink

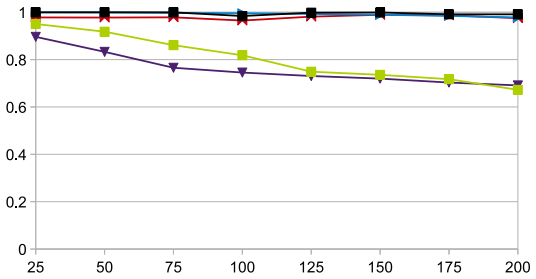
Figure 5.4: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. DNSB is green, DNLB is blue, RAND is black.



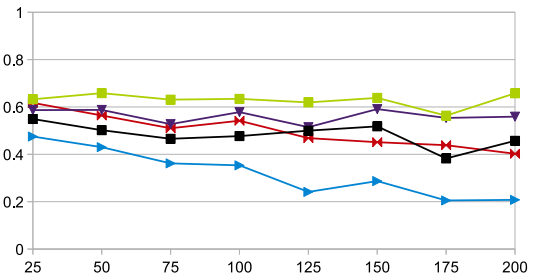
(a) Infomap - LFR



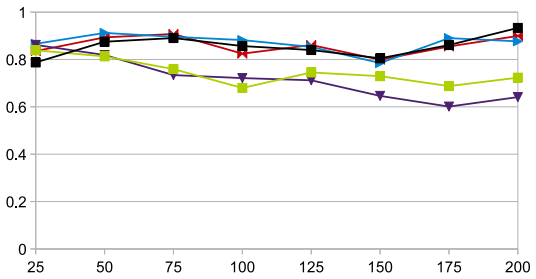
(b) Infomap - Postlink



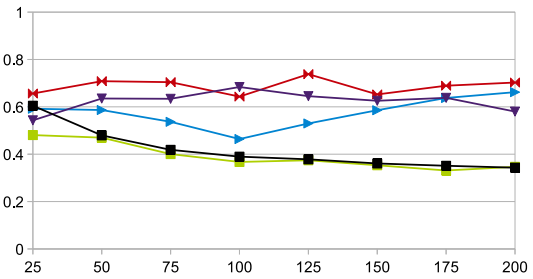
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.5: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of added edges on the X axis.Similarity is on the Y axis. AE-SBiSBo is green, AE-SBiLBo is purple, AE-LBiSBo is light blue, AE-LBiLBo is red, RAND is black.

The one among the four possible combinations to provide the best accuracy in most cases seems to be AE-SBiSBo. Only against Infomap in artificial network and against Oslom in the real network its similarity curve resides above that of the RAND technique.
The fact that AE-SBiSBo seems to provide good results can be explained this way: the procedure produces edges between the nodes that form the core of the target community and those nodes that form the core of the other communities. This action weakens the core of both communities by increasing the betweenness of their nodes (the addition of an edge incident to a node can have as an effect the increase in number of shortest paths that go through it, but not their decrease), bringing (in most cases) to considerably good results. However there is to notice that the behaviour of these techniques is not uniform against all the CD algorithms.

In order to use this technique it is again necessary to have the exact and complete topology of the network as a whole, so that the betweenness scores can be computed.
Similarly to the technique based on deletion, no precomputation can be done so to gain any advantage. Computing the directed betweenness for each node in the graph takes $O(N \cdot E)$ time (as previously discussed). Picking the node with smallest betweenness in the community takes $O(N_{in})$ time with linear search, while picking the one with largest betweenness outside takes $O(N_{out})$ time, thus the overall time to pick the two nodes is $O(N)$. Creating an edge between the picked nodes is constant time. In summary each iteration takes $O(N \cdot E) + O(N) = O(N \cdot E)$ time.
It is important to keep in mind that the behaviour of these techniques in some cases is contrasting.

5.3 Probability based

In section 2.2 was discussed a method for community detection in directed graphs, based on the translation of the directed graph to a weighted graph. The weights of the latter depend on the probability p_{ij} that an edge directed from j to i exists in the null model. Those edges with smallest p_{ij} are those that one doesn't expect to find in a random graph, thus the presence of such an edge between nodes i and j is an indicator of the likelihood of these two nodes to belong to the same community. Therefore when trying to hide the community from the detection, edges with a small probability p_{ij} should be the ones to be deleted between nodes in the same community, so that the likelihood for them to be part of the same community decreases. Those to be created between nodes belonging to two different communities, should have small p_{ij} as well, so that the likelihood for them to be erroneously considered as belonging to the same community increases.

Deletion of edges

Since the most statistically surprising edges are those that have a small p_{ij} probability score, the hiding technique based on edge deletion that aims to identify and delete these edges is as follows:

1. precompute the probability p_{ij} for each intra-community edge from node j to node i
2. delete edge with smallest score

3. update degrees of the end-point nodes of deleted edge
4. update probability score for those edges that share an end point with the deleted edge
5. go back to step 2 until desired percent of internal edges were deleted

Assuming a CD algorithm based on maximisation of the weighted modularity function Q_w is adopted to identify the target community, then removing an edge with smallest p_{ij} has the effect to zero a term that most contribute to Q_w when considering i and j as part of the same community (see end of section 2.2), thus it will be less likely for i and j to be considered as in the same community after the deletion.

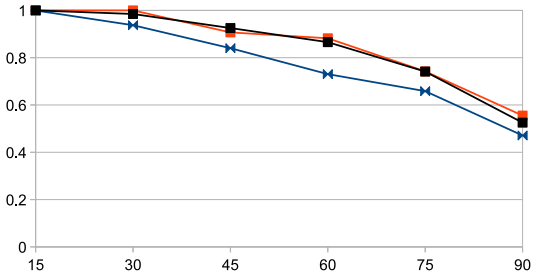
This procedure was tested against its symmetric version (i.e. that targets those edges with the largest probability). From now on I will refer to the method that targets edges with smallest p_{ij} as DESP (Delete Edge Smallest Probability), and the one that targets edges with largest probability as DELP. The outcomes of their comparison are shown in figure 5.6.
With LFR networks DESP has a similarity curve that in all cases resides below (or slightly below) the one from the RAND manipulation technique. This is because the probability p_{ij} capture well the relative significance of edges to a community. DELP on the other hand seems to follow very closely the behaviour of RAND. This is also true with the real network in all cases bar against Oslom, which shows an inverted trend.

Like the deletion method based on modularity discussed in section 5.1, this method requires as an input the degree sequence of all nodes in the community, while it is not necessary to know the exact topology of the community, nor any other information whatsoever about the outside of the community.
Given the degree sequence of the nodes in the community, the precomputation phase in which the probability scores of each edge are computed takes $O(E_{in})$ time (with E_{in} as the number of intra-community edges). Picking the edge with smallest p_{ij} score requires $O(E_{in})$ time with linear search, while its deletion is constant time. Similarly to the modularity-based technique, the update of the probability scores may be required for every edge left as in the worst case they may all share an end-point with the deleted edge, thus this takes $O(E_{in})$ time too.
In summary both precomputation phase and each iteration take $O(E_{in})$ time to complete.

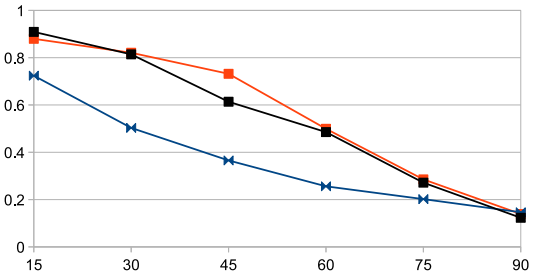
Addition of edges

To hide the community, new inter-community edges that represent a statistically surprising configuration should be added to the existing community in a way that affects how CD algorithms choose to assign memberships to each node. Taking this into account, a procedure that creates inter-community edges with small p_{ij} probabilities is devised as follows:

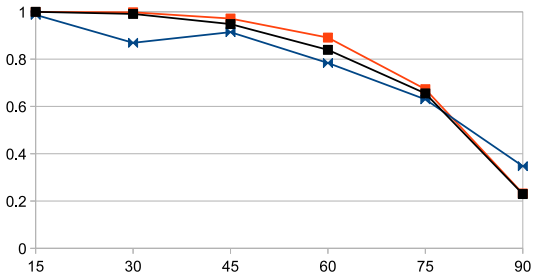
1. precompute the p_{ij} probability score for each possible inter-community edge that doesn't exist already
2. create the edge with smallest probability
3. update degrees of end-point of the end-point nodes



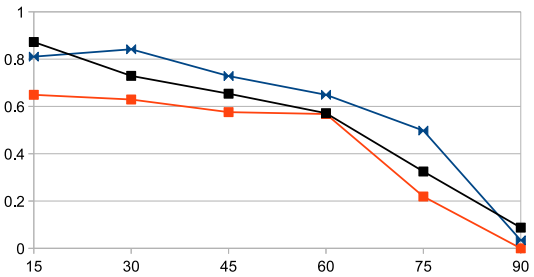
(a) Infomap - LFR



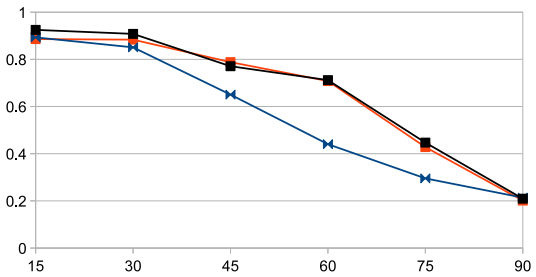
(b) Infomap - Postlink



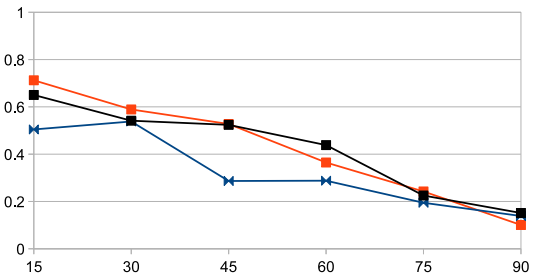
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.6: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. DESP is blue, DELP is orange, RAND is black.

4. update score of non-existing inter-community edges that share an end-point with the last created edge

5. go back to step 2 until desired percent of inter-community edges were created

Following what already said earlier, adding these statistically surprising edges has the effect that when considering whether the two nodes belong to the same community or not, then considering them as belonging to the same community would considerably increase the value of the weighted modularity score Q_w associated with the partition of the graph. So to push the community algorithms into misclassifying nodes, such surprising edges should be added between the community and the other communities.

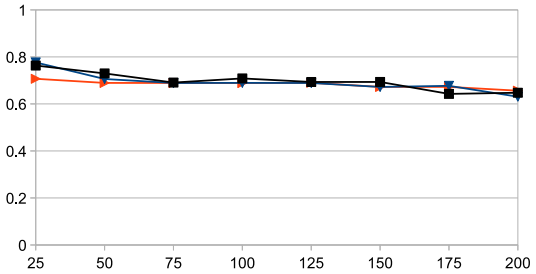
Referring to the figures in table 5.7, the behaviour of AESP and AELP resembles that of the techniques based on modularity AESK and AELK when it comes to artificial networks. Against Oslom it behaves as desired, while against Infomap and Extreme Optimisation all the techniques behave like RAND. Conversely in the real network case instead it works as desired against Infomap and Extreme Optimisation, while the trend is inverted against Oslom.

It is necessary to have as an input the degree sequence of all nodes in the graph for this method to work, however no other topological information whatsoever is required.
The precomputation phase consists of computing the probability score for each inter-community edge that doesn't already exist, and as pointed earlier the number of such edges is at most $2 \cdot N_{in} \cdot N_{out}$, thus the precomputation phase takes an $O(N_{out} \cdot N_{in})$ time to complete.
In each iteration the minimum probability is found with linear search in $O(N_{out} \cdot N_{in})$ time, the creation of the picked edge takes constant time, and the degree scores of the end-point nodes are updated in constant time too. The update phase needs to recompute the probability score of the edges that share an end point vertex with the last created one, and there are at most N such edges thus the update phase has an $O(N)$ running time (see section 5.1 for a similar analysis).
In summary both precomputation and single iteration have an $O(N_{out} \cdot N_{in})$ time complexity.

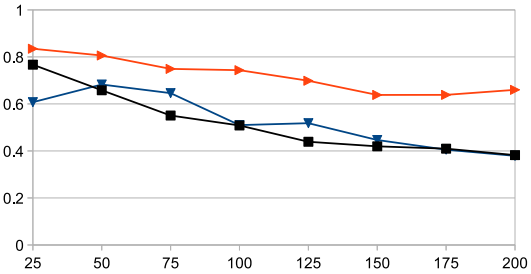
5.4 Community significance based

The concept of statistical significance of a given community [LRR10] was briefly introduced in section 2.4, however it needs to be discussed further. The assumption in section 2.4 was that a community is such that for each of its nodes the number of edges that depart from them and end within the community (*internal* edges) is larger than the number of such edges that a node outside the community would have. On this basis then, in *regular* networks, one defines a set of worst nodes as those nodes with the least number of internal edges. The significance of the community is assessed estimating the probability to have worst nodes with same or higher number of internal edges in an optimised community[2] from a random graph.

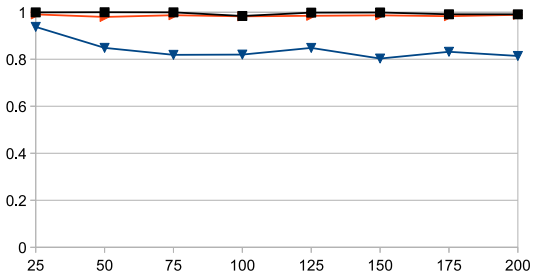
[2]optimised refers to the fact that this probability is estimated using extremal statistics, so the comparison is not made against average results but against the *luckiest* configuration that may come out of the null model



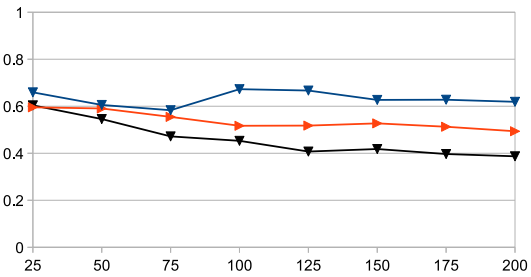
(a) Infomap - LFR



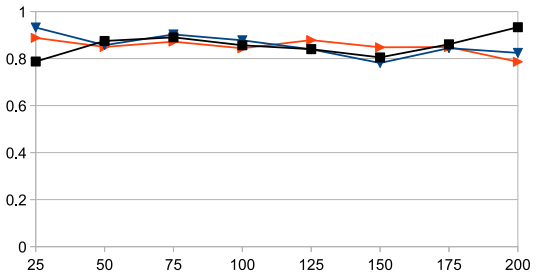
(b) Infomap - Postlink



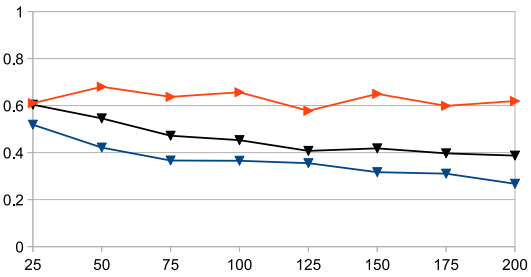
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.7: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of added edges on the X axis. Similarity is on the Y axis. AESP is blue, AELP is orange, RAND is black.

To deal with *heterogeneous* networks, the ranking to establish which are the worst nodes can't be done simply by taking into account the internal degrees k_i^{int}, otherwise a node with small degree, but all of its edges internal to the community C may be considered to be worse than one with large degree and just a fraction of its edges being internal. To overcome this problem Lancichinetti et al. [LRR10] then rank each node by the *probability* of having a node with internal degree greater or equal than k_i^{int} in the null model, having fixed C and k_i.
Let m be the total ends in the graph (twice the number of edges), m_C the number of ends in the community, and $m^* = m - m_C$ the number of ends outside the community, and let k_i be the degree of the *i-th* node. All these quantities can be divided into their internal and external contribution with respect to the community. Then the probability of node i to have k_i^{int} internal connections to C is given by

$$P(k_i^{int}|C) = \frac{\binom{m_C^{ext}}{k_i^{int}} \cdot \binom{m^* - m_C^{ext}}{k_i - k_i^{int}}}{\binom{m^*}{k_i}} \tag{5.1}$$

which represents the ratio between the total number of ways i can have k_i^{int} connections to C times the total number of ways it can have $ki - k_i^{int}$ connections outside C, and the total number of ways i can have connections placed anywhere in the network.
It follows then that the probability for node i to have k_i^{int} or more internal connections to the community C is given by

$$r_i = \sum_{j=k_i^{int}}^{k_i} P(j|C) \tag{5.2}$$

The worst nodes in the community then are those that do not represent a surprise if found in the community, thus they are the ones with largest probability r_i.
This formulation of the r_i score is defined for undirected networks, but it can be easily extended to take into account the directionality of the arcs [LRRF11]. It is enough to consider r_i as the product of the probabilities $r_{i,in}$ and $r_{i,out}$, which represent the probability for i to have incoming edges departing from C and the probability for i to have outgoing edges landing in C respectively.

Once these ranks are computed, the worst nodes can be selected to make the *border* of the community, according to which then the B-score is calculated to assess the community's significance. It is important to stress that the B-score represents the probability to find in an optimised community from the null model a list of worst nodes with sum of their scores smaller than the same sum presented by the community C. The least significant is the border of the community C, the largest is the sum of the scores of the worst nodes, thus the probability to find a smaller sum in an optimesd random graph grows (i.e. the B-score grows), and for large B-scores (greater than 5%) the community is to be regarded as not significant.

Deletion of edges

Bearing this in mind I developed an edge-deletion technique that targets those edges between the nodes with the largest r scores (i.e. the ones making the border). The procedure

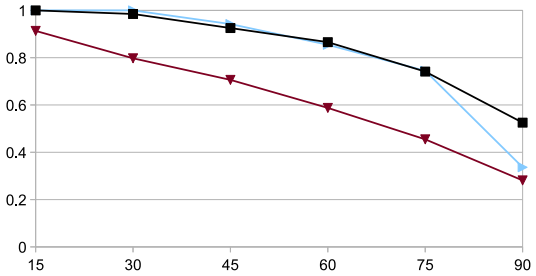
works as follows:

1. precompute directed version of r scores for each node in the community
2. pick node j as the one with largest probability r_j (i.e. least significant to community C)
3. for each edge incident to j, assign it a score equal to the probability r_i of the node i on the other side of the edge
4. delete the edge with largest score
5. update degrees and recompute r_j and r_i for the two nodes i and j between which the edge was deleted
6. go back to step 2 until desired percent of internal edges were deleted

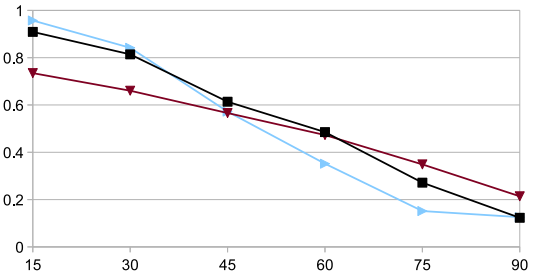
At each iteration an edge is deleted between the worst node in the community and the worst among its neighbours. After each deletion, before picking the next edge to delete, the r scores are recomputed only for those edges from which the edge was deleted. This is necessary because their significance is affected by the deletion of that edge (the quantities k_i and k_i^{int} both decrease by 1, see equation 5.1). For each other node the r score remains the same so it can be kept. It is important to notice that the recomputed r scores are larger than the older, meaning that the nodes become even less significant to the community.

This method was compared against its symmetric version (i.e. that picks nodes with smallest r). Let us call DNLR (Delete between Nodes with Largest probability R) the method that targets edges between nodes with largest r, and DNSR the one that conversely targets edges that run between nodes with smallest probability r. The results of their application are shown in figure 5.8.
No matter the CD algorithm used, both for artificial and real networks, DNLR's similarity curve is pretty much the same. Its shape is very linear and quite steep. It is fairly distant from the similarity curve of the random perturbation in artificial networks, while DNLR follows closely the RAND curve. In real networks the phenomena is inverted after 50% of edges is deleted, and DNSR method shows a similarity curve below that of DNLR (however this isn't bad as in most cases may be more realistic to desire to remove a few edges rather than more than 50%).

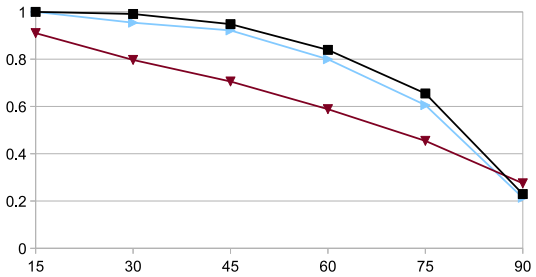
The DNLR technique aims to target the significance of the community. It deletes those edges that run between the two nodes that have the least statistically surprising number of connections to the community (i.e. its border). Removing an edge between them makes them both less significant to the community as they get even more loosely connected to C. At each iteration the worst nodes see their r score grow larger, thus increasing the average r score for the community's border. As an effect the sum of the border's ranking scores grows, so the B-score associated to the community grows more and more implying the decay of its statistical significance, so that the CD algorithms fail to assign the nodes to the original community.
The DNSR technique deletes those edges that connect instead the two best nodes in the community (i.e. those with smallest probability r). As already observed, deleting an edge between two nodes increments the r scores for both. At each iteration the r scores of the



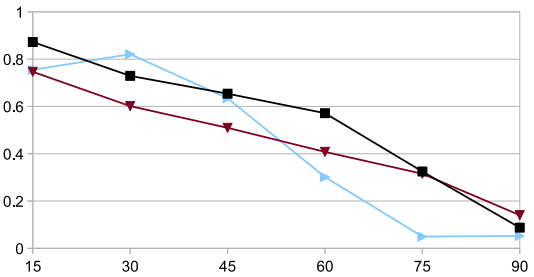
(a) Infomap - LFR



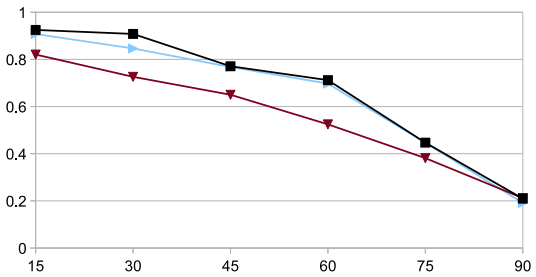
(b) Infomap - Postlink



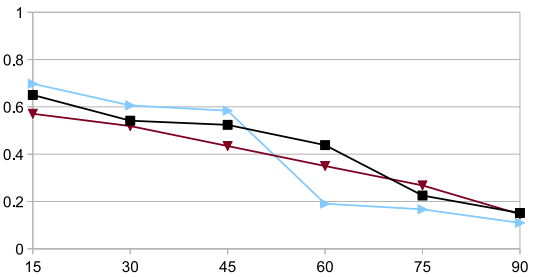
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.8: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. DNLR is bordeaux, DNSR is light blue, RAND is black.

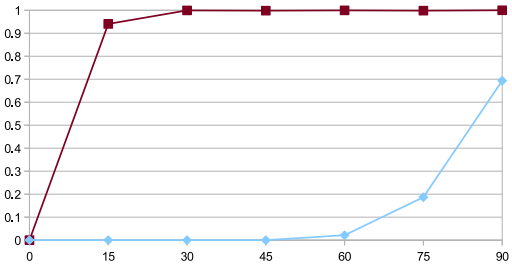


Figure 5.9: Average B-score values for DNLR (bordeaux) and DNSR (light blue).

nodes that do not make the border of the community (i.e. the core) get larger, but the r scores of the nodes in the border remains untouched. In other words the removal of such edges doesn't affect the B-score (i.e. the significance of the community) and the target community remains indeed still correctly detected, even after a large percent of edges is deleted. As a proof of the above theory, see figure 5.9. The B-score becomes almost 1 with just 15% of edges deleted using DNLR, while with DNSR it stays close to 0 even after 60% of edges are deleted.

This alteration method requires as input the degree sequence of all the nodes inside the community, while the exact topology of the community is not necessary. Furthermore it needs the sum of the degrees of all nodes in the graph (to obtain the m, m^* and m_C parameters). Absolutely no other information about the outside of the community is needed. The computation of the r probability scores involves the calculation of some binomial coefficients. To compute a single binomial coefficient $\binom{n}{k}$ it is necessary $O(nk)$ time using dynamic programming [Pun08]. To compute $P(j|C)$ three binomials are needed, of which the most expensive is the one at the denominator (m^* is greater than both m_C^{ext} and $m^* - m_C^{ext}$, while k_i is greater than both k_i^{int} and $k_i - k_i^{int}$), thus its time complexity accounts for $O(m^* \cdot k_i)$. The r score is the sum of $k_i - k_i^{int}$ different $P(j|C)$ scores (the maximum number of such terms is $k_i - 1$ if $k_i^{int} = 1$), thus the worst case for the computation of an r score accounts for $O(m^* \cdot k_i^2)$. However it can be observed that the precomputation phase consists of the calculation of the r probability score for each node in the community, and since each r_i score is the sum of a number of $P(j|C)$ terms equal to the number of inter-community edges departing from node i, then the computation of the r score for every node in the community will require the calculation of a number of $P(j|C)$ terms equal to twice[3] the number of inter-community edges E_{out} departing from the whole community. Therefore the precomputation phase will require $O(E_{out} \cdot m^* \cdot k_{i,max})$ time, where $k_{i,max}$ is the largest degree of the nodes in the community.

Let us see now what is the time complexity for each iteration. Finding the node inside the community with the largest probability requires $O(N_{in})$ time with a linear search. Assigning a score to an inter-community edge touching the picked node is constant time, but there are at most N_{in} such edges, thus this takes $O(N_{in})$ time. Finding the one edge with largest score requires $O(N_{in})$ time with a linear search. Recomputing the r scores for the end-point nodes of the deleted edge takes $O(m^* \cdot k_i^2)$ time, thus each iteration takes

[3] each edge touches two nodes

$O(N_{in}) + O(m^* \cdot k_i^2)$ time.

After this first analysis, the following observations can be made: a large number of *expensive* binomial coefficients is computed; only two r scores change at each iteration, and more specifically one of the two is the largest one, and both *increase* at the end of each iteration. Taking these points into account, things can be changed a bit to achieve better performance.

Computing many binomial coefficients $\binom{n}{k}$ with dynamic programming is time-consuming, and it would be better instead to build a Pascal's triangle [NN10] in $O(n^2)$ time and space, so that then any coefficient for $k \in \{0..n\}$ can be just looked-up in the table in constant time. If this solution is adopted, the computation of a single $P(j|C)$ probability score requires constant time, while the computation of an r score takes $O(k_i)$ time. Therefore the precomputation phase would take a total $O((m^*)^2) + O(E_{out})$ that accounts for the creation of the table and the computation of E_{out} probability scores $P(j|C)$. During each iteration, the recomputation of the r scores would take $O(k_{i,max})$ time, thus having a total $O(k_{i,max}) + O(N_{in})$ time complexity. The advantage into precomputing the Pascal triangle is thus evident.

For a slight practical speed up, a max-heap might be employed to store in an ordered way the nodes' r scores, however this doesn't affect the overall worst case time complexity. The max-heap can be built in the precomputation phase with a $O(N_{in})$ time complexity, and since $O(N_{in}) \in O(N_{in} \cdot k_{i,max})$, the overall complexity of the precomputation phase doesn't change. During each iteration the largest r score can be found in constant time thanks to the heap. The update of the r scores will see at most one of them potentially break the max-heap property by becoming larger (as the other one is already the largest in the heap, thus becoming even larger can't break the max-heap property), therefore max-heapify [CLRS01] can be used to recovery the max-heap property in $O(\log_2 N_{in})$. However finding the smallest edge score would still need $O(N_{in})$ time, as the list of such edges and their scores potentially changes at each iteration.

Addition of edges

Different hiding techniques based on the concept of statistical significance were tested, and the one that shown to be the most efficient is described here:

1. precompute as follows a score for each inter-community edge that doesn't exist: consider the communities to which the two end-point nodes belong as if they were a single community and compute the relative r scores of the end-points with respect to this artefact community; assign to the edge the sum of the r probabilities

2. create the edge with smallest sum

3. update degrees and recompute r_j and r_i of end point nodes

4. update sum scores of non-existing edges that share an end-point with the last created edge

5. go back to step 2 until desired percent of inter-community edges were created

This method revolves on the artefact communities resulting from the union of the target community with each other community in the network. Considering the two communities as a single community is a trick that enables to identify what are the nodes that would be most likely to be put in the same community, but are not because not sufficiently knit. This identification is carried on via the observation of the sum of the crafted r scores of each possible couple of nodes in which one belongs to the target community and the other to the other considered community. Considering the smallest of such sum is equivalent to picking the two nodes that with smallest probability would be found in an optimised community drawn from the null model. This means that this pair of nodes is among the most statistically significant to the crafted community, thus the nodes *could* be in the same community if just they were more knit.

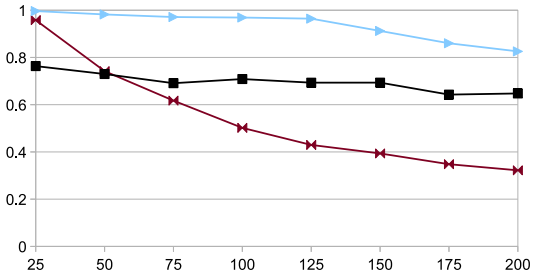
Unlike the other devised addition methods, this method assigns the same scores to symmetric edges. The sum scores for each edge depend on the two end-point vertices, thus when evaluating the sum scores for an edge from i to j and from j to i, the sum of the r scores of the nodes will of course be the same. When creating an edge, in case one of the directed edges exists already, then the one that goes in the opposite direction will be created. In case none of the edges between i and j exists which edge direction should be picked? To answer this question I make again use of the p_{ij} probability scores discussed in section 2.2. Since smallest probability is index of most surprising configuration, the direction to pick when creating an edge between i and j will be the one with smallest probability.

The creation of an edge between the two nodes has two effects: on one hand it makes them more linked, thus having an even more significant probability of belonging to the same community than they had before (while in fact they belong to different communities), on the other hand the nodes become less important to their original communities as adding the inter-community edge increases their r scores with respect to the real communities. This means that this technique weakens the statistical significance of the nodes in the core of the target community and at the same time pushes these nodes to another community (or pulls nodes from other communities).

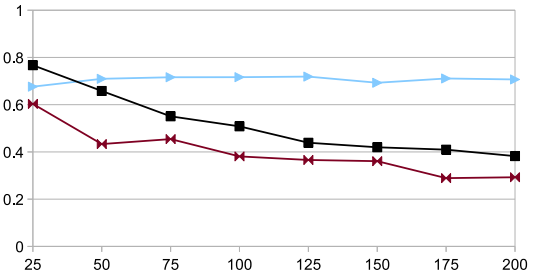
From figure 5.10 it can be seen that the ANSR's similarity curve always resides below the RAND curve (bar the first 50% in fig 5.10a), while in most cases ANLR resides above. The important thing to notice is that unlike the other developed techniques based on edge addition ANSR seems to behave good independent of what CD algorithm was used both in real and artificial networks.

This procedure needs as an input the degree sequence of all nodes in the network, and from that all the other required parameters can be computed. No other internal or external topological information is required (like edge positions).
During the precomputation phase it is necessary to build a Pascal table with m^*_{max} levels, where m^*_{max} is the largest number m^* of ends outside the crafted community, over all the possible crafted communities obtained as union of the target community with any other. The r score must be computed for each of the N nodes in the graph. Each r_i takes a number of $P(j|C)$ terms which is at most the number of edges touching node i, thus when computing r for all the nodes then a total number of $P(j|C)$ equal to at most twice[4] the number of edges E in the graph is computed. The number of all possible inter-community edges is

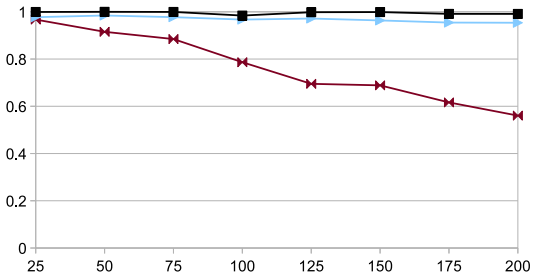
[4]each edge touches two nodes



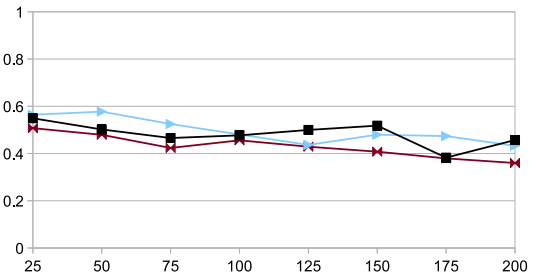
(a) Infomap - LFR



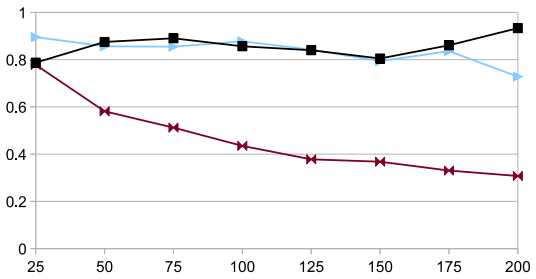
(b) Infomap - Postlink



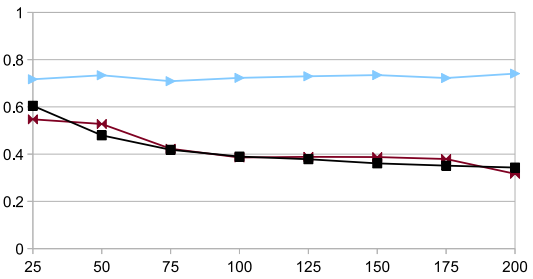
(c) Oslom - LFR



(d) Oslom - Postlink



(e) Extreme Optimisation - LFR



(f) Extreme Optimisation - Postlink

Figure 5.10: On the left column results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the right from Postlink network. Percent of removed edges on the X axis. Similarity is on the Y axis. ANSR is bordeaux, ANLR is light blue, RAND is black.

given by $2 \cdot N_{in} \cdot N_{out}$, thus this many r scores must be computed and assigned to the edges. In summary the precomputation phase takes at most $O((m^*_{max})^2) + O(E) + O(N_{in} \cdot N_{out})$ time.
Each iteration takes $O(N_{out} \cdot N_{in})$ to search the minimum among at most that many candidate edges, the creation of the picked edge is constant time, the re-computation of the r scores of the end-point nodes takes $O(k_{i,max})$ time, while the update of the scores of those edges that share at least one end-point vertex takes $O(N)$ time, and $O(N) \in O(N_{out} \cdot N_{in})$ (see section 5.1 for a similar analysis). Overall each iteration then takes $O(k_{i,max}) + O(N_{out} \cdot N_{in})$ time, but $O(k_{i,max}) \in O(N_{in} \cdot N_{out})$ since the degree of any node can't be larger than the number of nodes N in the graph, and $N_{in} \cdot N_{out} \geq N$, thus eventually the time complexity of a single iteration amounts to $O(N_{in} \cdot N_{out})$.

It is important to keep in mind that, unlike others, the behaviour of the techniques discussed in this section is quite uniform in all the presented tests.

In this chapter I discussed what were the manipulation strategies I developed to hide a community from detection. I showed how these strategies set their roots in both popular concepts and recent results from the research in the community detection field. I presented their accuracies and gave an interpretation for the obtained results. An accurate analysis of the knowledge models and the computational resources needed to employ any of these strategies were provided, so to guide a user into making a better informed decision.
Overall the finding was that even though in general the presented techniques present desired features, those based on statistical significance seem to be the most accurate and the only ones really independent from the CD algorithms the adversary may choose.

Chapter 6

Critical evaluation and summary

In the previous chapter many manipulation strategies were devised and discussed thoroughly. In this chapter I will provide a critical comparison that should help understand what are the manipulations that represent the best choice depending on the situation at hands.

Lastly I will sum up all the findings and talk about what could be the future extensions to this work.

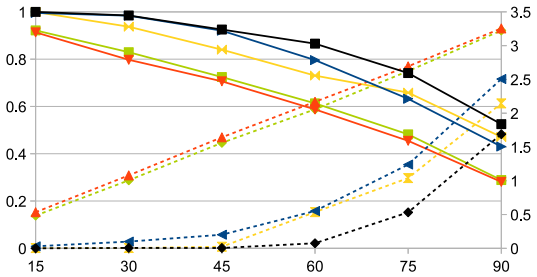
6.1 Comparison of the developed techniques

The proposed hiding techniques can be divided in two categories: those based on centrality measures and those based on statistical significance. The methods based on modularity[1] and betweenness belong to the centrality measures category while the statistical significance category includes the techniques based on edge probability p_{ij} and community's statistical significance. The techniques in the first category make use of classic and widely accepted concepts in the history of community detection while those in the latter explore and validate results coming from more recent studies.

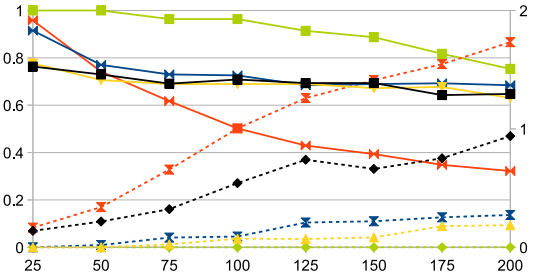
The developed techniques provide different degrees of efficiency, require different inputs and different computational complexities. These are all summarised in this section to help understand what are the most suitable techniques to adopt in different situations.

Initially let us focus on the bare accuracy of each method. As figure 6.1 shows, both when deleting than when adding edges, the techniques that give the best results are those based on betweenness and community's significance (bar one case, against Infomap, in which the addition technique based on betweenness gives bad results). While every family of techniques in general shows the desired behaviour, it is important to notice that the only one that gives *always* good accuracy is the one based on community's significance. More over in the case of addition, some techniques show a similarity curve that resembles a constant function (or linear with very small slope), meaning that adding further edges gives no (or little) gain, while the ones based on significance are quite tilted showing how it is always convenient to add more edges following this strategy. Furthermore notice how

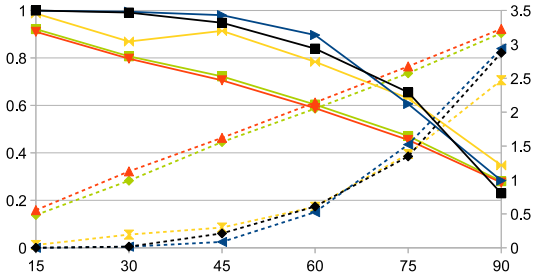
[1] node degree is a centrality measure, and the modularity score tries to capture the unlikeliness of a configuration by the observation of the degrees of the end-point nodes of all edges



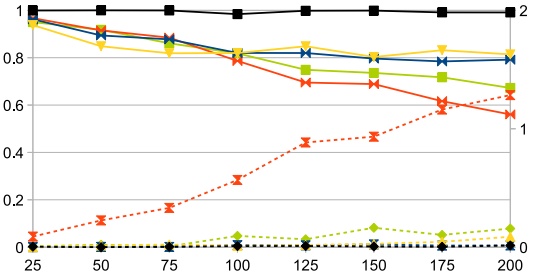
(a) Infomap - LFR - delete



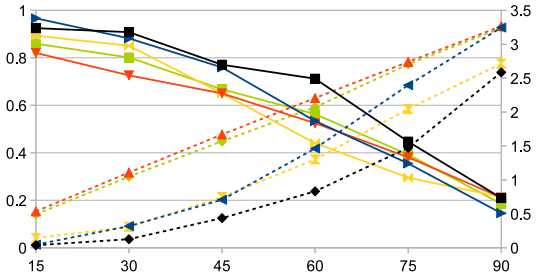
(b) Infomap - LFR - add



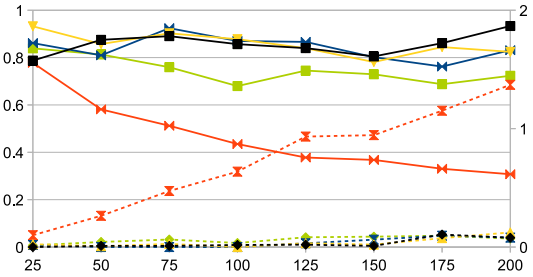
(c) Oslom - LFR - delete



(d) Oslom - LFR - add



(e) Extreme Optimisation - LFR - delete



(f) Extreme Optimisation - LFR - add

Figure 6.1: Results from LFR artificial networks with parameters $N = 500$, $\langle k \rangle = 25$, $maxk = 50$. On the left column deletion based techniques, on the right colum addition based techniques. Filled lines plot average similarity (primary Y axis), dashed lines plot average enthropy (secondary Y axis). Techniques based on degree are blue, those based on betweenness are green, probability is yellow and community's significance is orange. Percent of removed/added edges on the X axis. Filled lines plot similarity (primary Y axis), dashed lines plot entropy (secondary Y axis).

the entropy score of the technique based on significance significantly departs from those of the others.

However there are other factors that need to be taken into account. Table 6.1 reports the computational needs of precomputation phase and deletion/addition of a single edge in the target community. Please note that E_{in} is the number of intra-community edges for the target community, while E_{out} is the number of inter-community edges (i.e. E_{out} is not the number of edges outside the community). First thing to notice is that in most cases the

	Precomputation	Iteration
Modularity (delete)	$O(E_{in})$	$O(E_{in})$
Modularity (add)	$O(N_{in} \cdot N_{out})$	$O(N_{in} \cdot N_{out})$
Betweenness (delete)	-	$O(N \cdot E)$
Betweenness (add)	-	$O(N \cdot E)$
Probability (delete)	$O(E_{in})$	$O(E_{in})$
Probability (add)	$O(N_{in} \cdot N_{out})$	$O(N_{in} \cdot N_{out})$
Significance (delete)	$O((m^*)^2) + O(E_{out})$	$O(k_{i,max}) + O(N_{in})$
Significance (add)	$O((m^*_{max})^2) + O(E) + O(N_{in} \cdot N_{out})$	$O(N_{in} \cdot N_{out})$

Table 6.1: Summary of computational complexity of precomputation and iteration phases for each technique

techniques get computationally more needy when adding edges than when deleting them. The alteration techniques based on betweenness are an exception in that both addition and deletion have the same computational complexities. The most expensive techniques in terms of precomputation costs are those based on community significance. The most demanding techniques for a single edge addition/deletion are those based on betweenness. This means that in the best scenario, if the graph is sparse, then the deletion/addition of a single edge would cost $O(N^2)$ time using these techniques (more if the graph is not sparse). Thus the techniques based on betweenness are quite slow. Those based on community significance have quite a high precomputation cost, but then the cost for each iteration is reasonable. However the techniques based on modularity and probability are the most convenient if the computational resources are limited.

Let us now move the focus to the knowledge models required to adopt each technique. Table 6.2 is a summary of the informations that are necessary for each technique. The

	Degree Sequence	Topology
Modularity (delete)	local	none
Modularity (add)	global	none
Betweenness (delete)	none	global
Betweenness (add)	none	global
Probability (delete)	local	none
Probability (add)	global	none
Significance (delete)	local + partially global	none
Significance (add)	global	none

Table 6.2: Summary of input needed for each technique

column "topology" indicates what kind of knowledge about the position of edges is needed for the use of a technique. Unlike all other techniques that do not require to know anything about the position of the edges, those based on betweenness require knowledge of the exact topology of the whole graph, also for the deletion based version (which in all other cases require less informations compared to their addition based counterparts). Modularity and probability based deletion techniques need to know only the degree sequence of the nodes in the community, while the one based on community significance requires also to know the sum of the degrees of all nodes in the graph (this can be estimated as the product of the order N of the graph and the average degree $\langle k \rangle$, and in some cases these two informations may be available). Their addition-based counterparts all require the degree sequence of all nodes in the graph. In summary, despite the local nature of the alterations to the graph, the techniques based on betweenness require an unrealistic amount of knowledge about the graph. Of the remaining, those based on deletion require a reasonable amount of information while those based on addition become too demanding to be still considered as realistic in most real case scenarios.

In general, even though the accuracy of methods based on betweenness and community's significance are quite close, the former require an unrealistic amount of information and much more computing resources, thus the latter are certainly preferable. The decision as to whether adopt deletion or addition based techniques depends on the feasibility of such actions (i.e. can an edge be deleted? can it be somehow concealed? can it be created?), but in most cases the ones based on deletion should be preferable as they need only local informations and provide good performance. It would be of course even better if both addition and deletion were feasible, so to execute both manipulations. Lastly, in case both the computational resources and the available knowledge are very limited then the techniques based on modularity and probability are to be preferred.

6.2 Summary and future work

In this work I summarised what were my findings during my research for some techniques that would allow a target community to be hidden against detection. I covered the evolution of the community detection algorithms, showing how the great majority of them rely on the modularity function maximisation, and how only recently some studies have moved towards concepts of statistical significance. I showed how it was common practice to just ignore direction of the edges, while recent studies stressed how the information included in edge direction is instead of major importance to the methods' accuracy.

All the developed techniques were drawn from popular concepts in the community detection field, I tried to identify which are the features that can either make or disrupt a community, and my focus was oriented to those techniques that keep the direction of edges in account. These techniques were tested against different community detection algorithms, so to show to what degree their accuracy depends on the adversary's particular CD algorithm of choice.

The biggest challenge was to find a manipulation strategy that would work in any circumstance, independently of the CD algorithm adopted. The main achievement in this sense came with the exploitation of concepts like statistical significance of a community as a whole, and statistical importance of a node to a community, that I conveyed into strategies

for the addition and deletion of edges that could either make or disrupt a community. The experimental results highlight how these techniques seem to provide with good results in any case. A remark that needs to be made regards the fact that what seems to be the second best approach is the one based on the classic concept of betweenness, and it requires unrealistic knowledge models and excessive computational resources. Furthermore other techniques that require less computational resources and tighter knowledge models were provided, even though their accuracy on average is inferior to that of statistical significance based techniques.

Eventually my novel contributions can be summarised with the following points

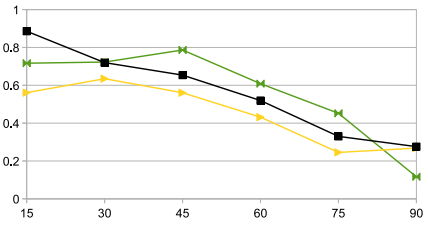
- This study is one of the first studies that aim to hide a community from detection in graphs (to the best of my knowledge there was only one other technical report [Nag08] that moved in this direction)
- I took edge direction into account, and used only local methods (to the best of my knowledge my work is the first regards each of these aspects)
- I compared several strategies against each other, evaluating critically pros and cons of each technique
- I devised two novel techniques (one based on edge deletion, the other on edge addition) that make use of concepts of community's significance [LRR10] and give good results
- The results provided are general, not biased by the choice of a particular CD algorithm (unlike the previous work I discussed in chapter 3)

As a future extension to this work, a study of the combination of deletion and addition techniques may reveal what percent of intra-community edges one should delete and what percent of inter-community edges should add to achieve the best hiding.
Another thing worth to explore resides in the deletion strategy based on community's significance: since my manipulation technique aims to weaken the nodes in the border of the community by increasing their r_i scores, a study may reveal whether once the r_i score for node i exceeds a certain threshold value (that may indicate that the node is already enough "out of the community"), then the node can be ignored and move to increase the score of the remaining nodes.
Lastly these techniques may be either easily adapted to the undirected version (bar the technique based on probability, which is based on the statistical unlikeliness of the edges direction), or find a way to extended them so to take into account edges' weight.

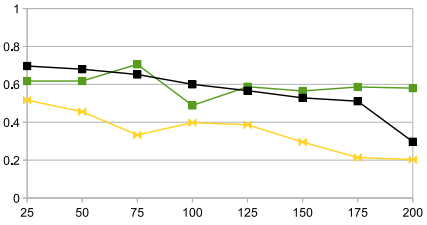
Appendix A

More experiments tables

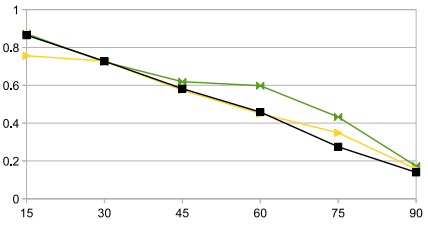
Here are included a few more tables with results coming from the application of the manipulation strategies to the Unionblog network (see section 4.1). Furthermore here is also presented the average similarity over all networks (both artificial and real) and against all CD algorithms. On the left columns the results come from edge deletion techniques, while on the right come from edge addition techniques. The first three rows represent the application of the techniques to the Unionblog graph, while the last row is the average over all networks and CD algorithms.



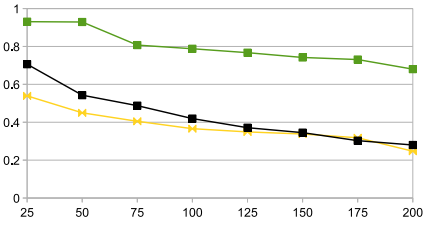
(a) Oslom - unionblog - delete



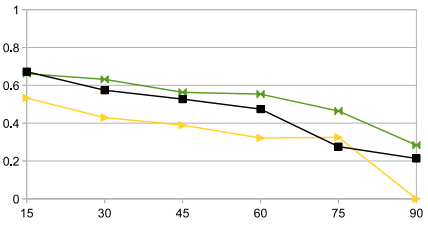
(b) Oslom - unionblog - add



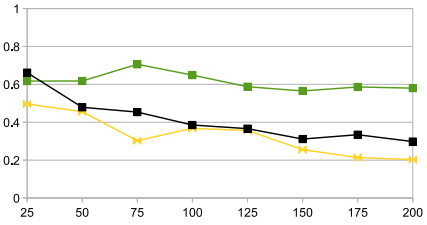
(c) Infomap - unionblog - delete



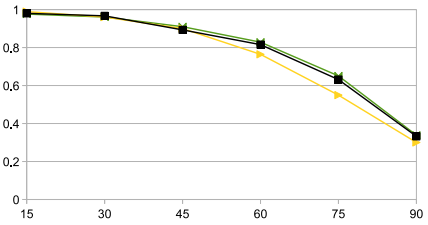
(d) Infomap - unionblog - add



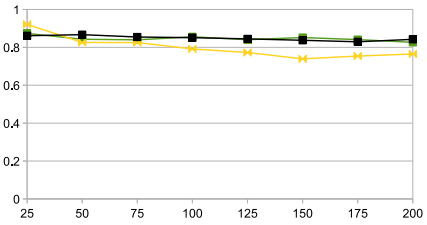
(e) Extreme Optimisation - unionblog - delete



(f) Extreme Optimisation - unionblog - add

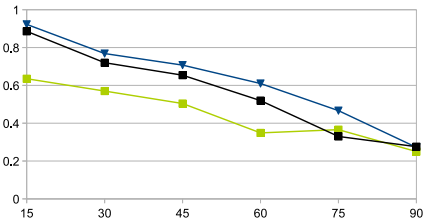


(g) Average over all networks and CD algorithms - delete

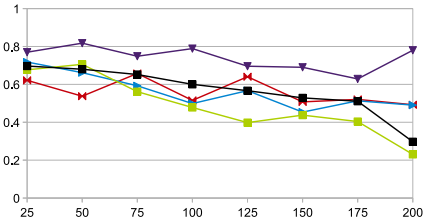


(h) Average over all networks and CD algorithms - add

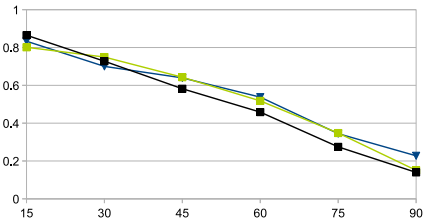
Figure A.1: On the left column averaged results from deletion based techniques. On the right results from addition based techniques. DESK and AESK are yellow, DELK and AELK are green, RAND is black. Percent of removed/added edges on the X axis. Similarity is on the Y axis.



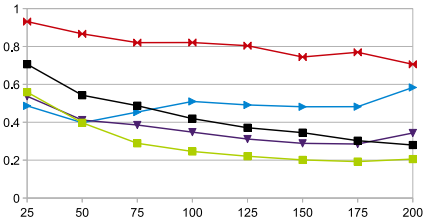
(a) Oslom - unionblog - delete



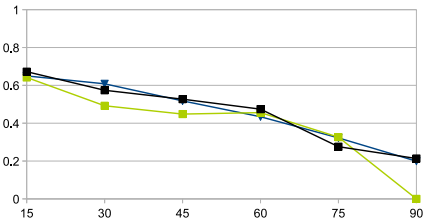
(b) Oslom - unionblog - add



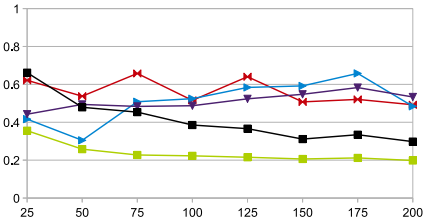
(c) Infomap - unionblog - delete



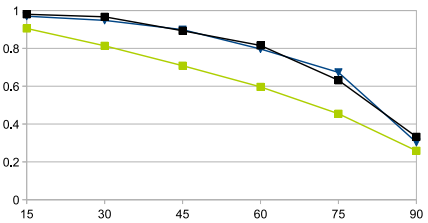
(d) Infomap - unionblog - add



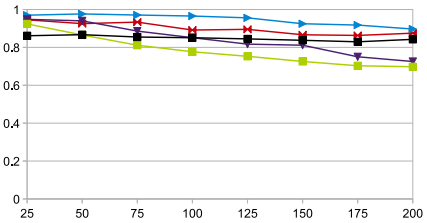
(e) Extreme Optimisation - unionblog - delete



(f) Extreme Optimisation - unionblog - add

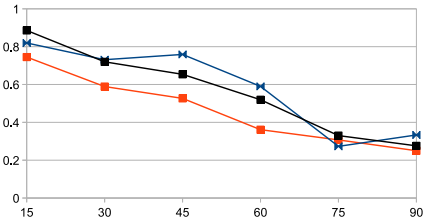


(g) Average over all networks and CD algorithms - delete

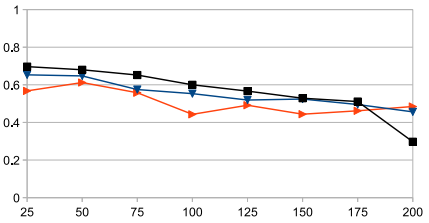


(h) Average over all networks and CD algorithms - add

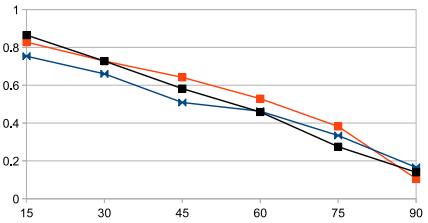
Figure A.2: On the left column averaged results from deletion based techniques, DNSB is green, DNLB is blue. On the right results from addition based techniques, AE-SBiSBo is green, AE-SBiLBo is purple, AE-LBiSBo is light blue, AE-LBiLBo is red, RAND is black. Percent of removed/added edges on the X axis. Similarity is on the Y axis.



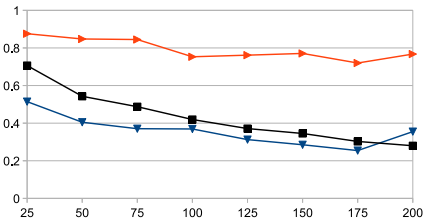
(a) Oslom - unionblog - delete



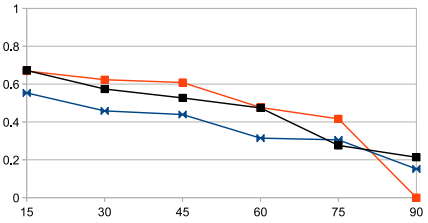
(b) Oslom - unionblog - add



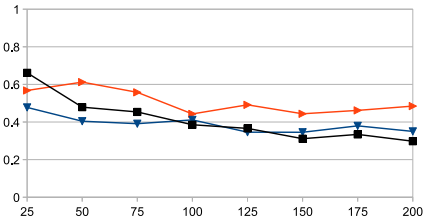
(c) Infomap - unionblog - delete



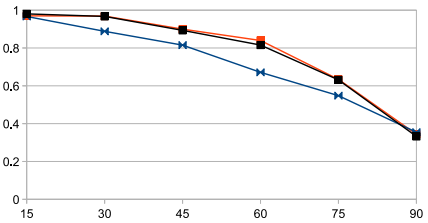
(d) Infomap - unionblog - add



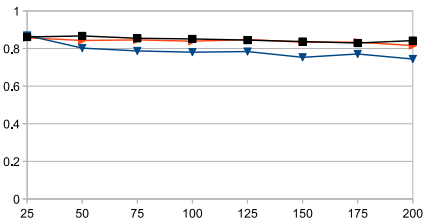
(e) Extreme Optimisation - unionblog - delete



(f) Extreme Optimisation - unionblog - add

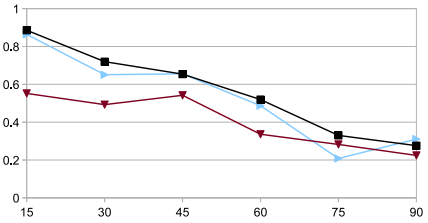


(g) Average over all networks and CD algorithms - delete

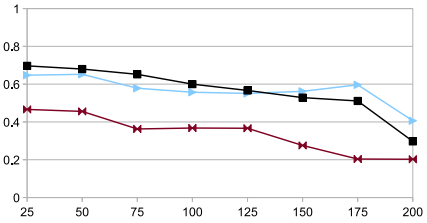


(h) Average over all networks and CD algorithms - add

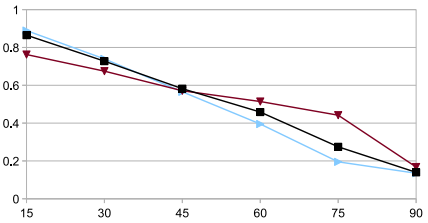
Figure A.3: On the left column averaged results from deletion based techniques. On the right results from addition based techniques. DESP and AESP are blue, DELP and AELP are orange, RAND is black. Percent of removed/added edges on the X axis. Similarity is on the Y axis.



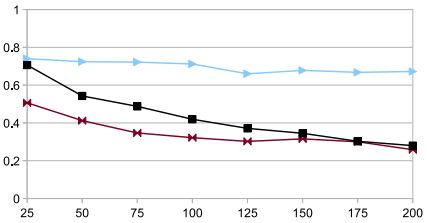
(a) Oslom - unionblog - delete



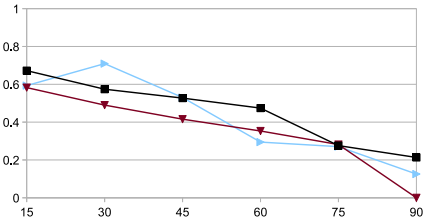
(b) Oslom - unionblog - add



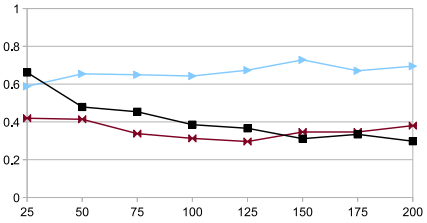
(c) Infomap - unionblog - delete



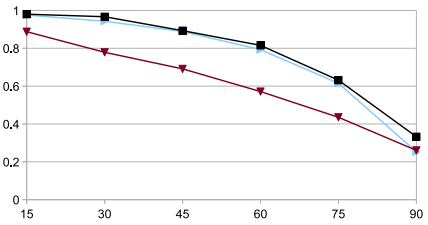
(d) Infomap - unionblog - add



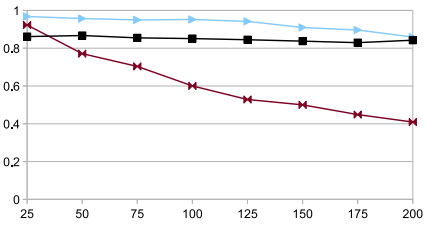
(e) Extreme Optimisation - unionblog - delete



(f) Extreme Optimisation - unionblog - add



(g) Average over all networks and CD algorithms - delete



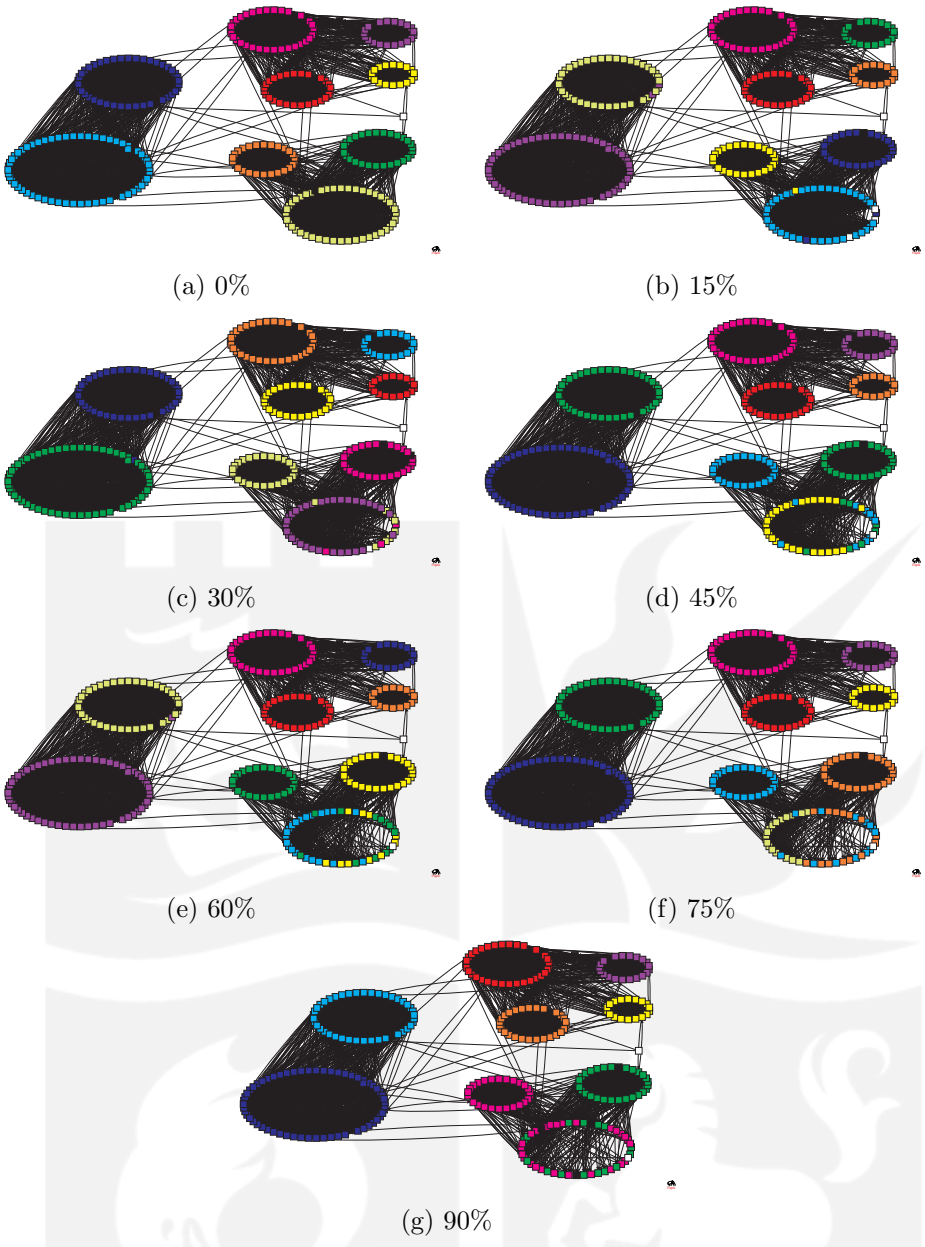
(h) Average over all networks and CD algorithms - add

Figure A.4: On the left column averaged results from deletion based techniques. On the right results from addition based techniques. DNLR and ANSR are bordeaux, DNSR and ANLR are light blue, RAND is black. Percent of removed/added edges on the X axis. Similarity is on the Y axis.

Appendix B

Example of visualisation

This visualisation technique (see section 4.4) provided a visual support that helped me quickly understand in the early stages of the development whether I was moving in the right direction. The position of the nodes represents the original division of the network in communities (i.e. each circle is a community). The colors represent the division in community found after the application of the manipulation strategy.
In figure B.1 is reported the visualisation of the application of DNSR against Oslom.



(a) 0% (b) 15%

(c) 30% (d) 45%

(e) 60% (f) 75%

(g) 90%

Figure B.1: The same network at different stages during the application of the DNSR technique. The target community is the one in the bottom-left corner.

Bibliography

[BDG+06] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity-np-completeness and beyond. http://www.bibsonomy.org/bibtex/2c791ac15d77263d7c5b17e37be4c556e/folke, 2006.

[BHJ09] Mathieu Bastian, Sebastien Heymann, and Mathieu Jacomy. Gephi: An open source software for exploring and manipulating networks, 2009.

[BM02] V. Batagelj and A. Mrvar. Pajek - analysis and visualization of large networks. *Graph Drawing*, 2265:477–478, 2002.

[CK01] A. Condon and R. M. Karp. Algorithms for graph partitioning on the planted partition model. *Random Structures and Algorithms*, 18(2):116–140, 2001.

[CLDF10] Jie Cheng, Xiaojia Li, Zengru Di, and Ying Fan. The attack tolerance of community structure in complex networks, 2010.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

[CN06] G. Csárdi and T. Nepusz. The igraph software package for complex network research. *InterJournal Complex Systems*, page 1695, 2006.

[CX07] M. Chau and J. Xu. Mining communities and their relationships in blogs: A study of online hate groups. *International Journal of Human-Computer Studies*, 65(1):57–70, 2007.

[DA05] J. Duch and A. Arenas. Community detection in complex networks using extremal optimization. *Physical Review E*, 72(2), 2005.

[FJSA07] J. Feigenbaum, A. Johnson, P. Syverson, and Acm. Probabilistic analysis of onion routing in a black-box model. *Wpes'07: Proceedings of the 2007 Acm Workshop on Privacy in Electronic Society*, pages 1–10, 2007.

[FLGC02] Gary William Flake, Steve Lawrence, C. Lee Giles, and Frans M. Coetzee. Self-organization and identification of web communities. *Computer*, 35(3):66–71, 2002.

[For10] Santo Fortunato. Community detection in graphs. *Physics Reports-Review Section of Physics Letters*, 486(3-5):75–174, 2010.

[Fou] Electronic Frontier Foundation. Nsa spying. https://www.eff.org/nsa/faq.

[Fre77] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[GA05] R. Guimera and L. A. N. Amaral. Functional cartography of complex metabolic networks. *Nature*, 433(7028):895–900, 2005.

[GDDG+03] R. Guimera, L. Danon, A. Diaz-Guilera, F. Giralt, and A. Arenas. Self-similar community structure in a network of human interactions. *Physical Review E*, 68(6), 2003.

[GFBHA11] Sergio Gomez, Alberto Fernandez, Javier Borge-Holthoefer, and Alex Arenas. Radatools, 2011.

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America*, 99(12):7821–7826, 2002.

[HH03] Petter Holme and Mikael Huss. Discovery and analysis of biochemical subnetwork hierarchies. *eprint arXiv:q-bio/0309011*, 2003.

[KLN08] Brian Karrer, Elizaveta Levina, and M. E. J. Newman. Robustness of community structure in networks. *Physical Review E*, 77(4):046119, 2008.

[KSJ09] Youngdo Kim, Seung-Woo Son, and Hawoong Jeong. Community identification in directed networks, 2009.

[KWLAC11] Derek Greene Karen Wade, Conrad Lee, Daniel Archambault, and Padraig Cunningham. Identifying representative textual sources in blog networks. Technical report, Humanities Institute of Ireland - University College Dublin, 2011.

[LF09] Andrea Lancichinetti and Santo Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 80(1 Pt 2):016118, 2009.

[LFR08] A. Lancichinetti, S. Fortunato, and F. Radicchi. Benchmark graphs for testing community detection algorithms. *Physical Review E*, 78(4), 2008.

[LN08] E. A. Leicht and M. E. J. Newman. Community structure in directed networks. *Physical Review Letters*, 100(11), 2008.

[LRR10] A. Lancichinetti, F. Radicchi, and J. J. Ramasco. Statistical significance of communities in networks. *Physical Review E*, 81(4):9, 2010.

[LRRF11] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *Plos One*, 6(4), 2011.

[MD06] Claire P. Massen and Jonathan P. K. Doye. Thermodynamics of community structure, 2006. eprint arXiv:cond-mat/0610077.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., 1997.

[Nag08] Shishir Nagaraja. The economics of covert community detection and hiding, 2008.

[New01] M. E. J. Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical Review E*, 64(1):7, 2001.

[New04] M. E. J. Newman. Analysis of weighted networks. *Physical Review E*, 70(5), 2004.

[NG04] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2), 2004.

[NN10] R. Neapolitan and K. Naimipour. *Foundations of Algorithms*. Jones and Bartlett Learning, 2010.

[Pun08] A.A. Puntambekar. *Design and Analysis Of Algorithms*. Technical Publications, 2008.

[RAB09] M. Rosvall, D. Axelsson, and C. T. Bergstrom. The map equation. *European Physical Journal-Special Topics*, 178(1):13–23, 2009.

[RB08] Martin Rosvall and Carl T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences of the United States of America*, 105(4), 2008.

[Red98] S. Redner. How popular is your paper? an empirical study of the citation distribution. *European Physical Journal B*, 4(2):131–134, 1998.

[SK88] P. R. Suaris and G. Kedem. An algorithm for quadrisection and its application to standard cell placement. *Ieee Transactions on Circuits and Systems*, 35(3):294–303, 1988.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[WM00] Richard J. Williams and Neo D. Martinez. Simple rules yield complex food webs. *Nature*, 404(6774):180–183, 2000.

[YG11] B. Yan and S. Gregory. Finding missing edges and communities in incomplete networks. *Journal of Physics a-Mathematical and Theoretical*, 44(49):17, 2011.

[Zac77] Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.