Empirical analysis of K-Betweenness

Abian M. Blome

October 1, 2007

Contents

1 Introduction and motivation

This paper presents the K-Betweenness algorithm and shows the evidence gathered on its performance. The goal of the algorithm is to divide a graph into a number of clusters, each consisting of nodes and being pairwise disjoint, which closely reflect the inherent community structure of the graph. We are basically trying to identify the key groups of the graph using only the information contained within it: the nodes and the edges connecting them. This separates it from the well-known concept of *graph partitioning* where usually the number of groups and their size is at least known approximately, and where the goal is finding the best possible clustering under these constraints.

We will now introduce a small number of definitions that will be used for the remainder of this paper: First of all we assume that a graph is given as $G = (E, V)$ with $n := |V|$ and $m := |E|$.

We define a *clustering* as a partition of nodes $\mathcal{C} := \{C_1, \ldots, C_k\}$ with the elements of $\mathcal{C}$ being pairwise disjoint non-empty sets of nodes, called *clusters*. Additionally we denote the set of edges within a cluster $E(C_i) := \{\{v, w\} \in E | v, w \in C_i\}$. Let $E(C_i, C_j)$ be the set of nodes with one edge in C_i and one in C_j. We will call $E(\mathcal{C}) := \bigcup_{i=1}^{k} E(C_i)$ the set of intra-cluster edges and $E \backslash E(\mathcal{C})$ the set of inter-cluster edges. The number of intra-cluster edges shall be denoted with $m(\mathcal{C})$ and the number of inter-cluster edges with $\overline{m}(\mathcal{C})$.

To ease the manipulation of the graph we will use the following function which returns the clustering induced by the components of a graph: $\omega(G) := \{\{w \in V | \exists Path(v, w)\} | v \in V\}$

The K-Betweenness algorithm presented here is a variant of the *Newman-Girvan algorithm* [1] which attempts to perform a speed-up in exchange for a small quality loss of the resulting community structure. Like the original algorithm it is based on the concept of edge betweenness [1]. The betweenness of an edge e in a graph $G = (E, V)$ is the number of shortest paths between all pairs of nodes that pass through it. However if there are several shortest paths between two nodes the number of these shortest paths passing through each edge on the paths is divided by the number of those paths (See Equation 1).

First we define some functions to actually number the shortest paths in a graph:

$$\phi_{st}(e) := \text{ Number of shortest geodesic paths from } s \text{ to } t \text{ through edge } e$$

$$\phi_{st} := \text{ Number of shortest geodesic paths between } s \text{ and } t$$

Now we use these values to properly define the betweenness of a certain edge:

$$C_B(e) := \sum_{s \neq t \in V} \frac{\phi_{st}(e)}{\phi_{st}}$$

Equation 1: Definition of betweenness

The main idea of the *Newman-Girvan algorithm* is that a high betweenness of an edge

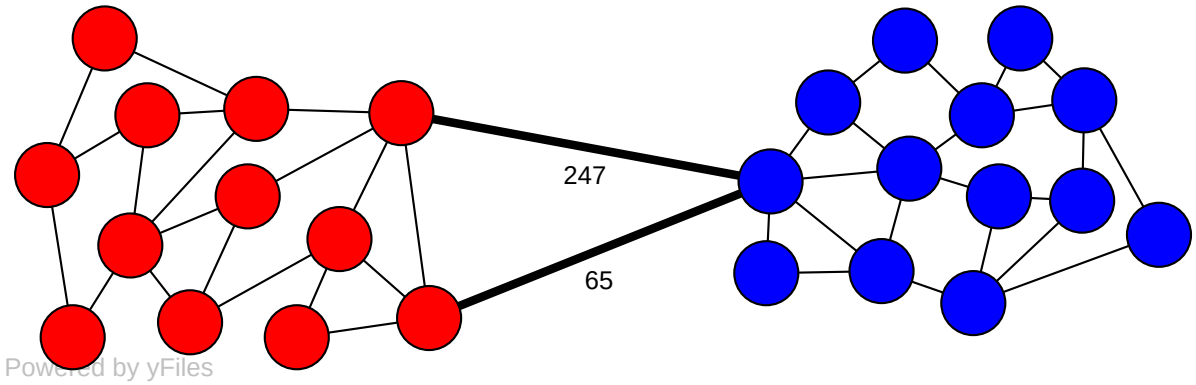


Figure 1: Obvious inter-cluster edges can have a low betweenness value due to them being masked by another inter-cluster edge as can be seen above. This makes a recomputation of the betweenness value vital.

can be seen as evidence for it being an inter-cluster edge (i.e. connecting two different communities) since the number of edges between two clusters are small. Therefore each of these edges is part of a large number of shortest paths. The information within a community structure tends to have multiple ways of flowing, in other words, there are several paths for the information to flow through, which are usually very short. If the information has to flow to another community it has to go through the few "gateway" edges which connect these two communities (and in some cases even through other communities). Thus one can sequentially remove the edge with the highest betweenness, which indicates that it is probably such a "gateway" edge, while taking care of recomputing the betweenness after each step. The recomputation is vital because, as can be seen in Figure 1, an obvious inter-cluster edge can be masked by the high betweenness of another edge.

After each step the individual components of the graph induce a clustering, i.e. a group of community structures. A method for representing the evolution of these clusterings is by means of a dendrogram, also known as a hierarchical tree. In it the clustering of the last step (i.e. each node constitutes its own community structure) is represented as a number of leaf nodes. For each time step, starting from the last and going backwards, the edge connecting two community structures is also portrayed in the dendrogram, which is thus a forest of binary trees. The roots of the trees represent the connected components of original graph, and the children of a node represent the separation of that component at a given time-step. Thus a cut in the dendrogram at a given level would return exactly the clustering at the same time-step. An example dendrogram can be seen in Figure 2.

The problem of selecting the best clustering from the dendrogram is solved, at least in the original algorithm, by simply searching through all of them using a quality index. These indices try to formalize the intuitive idea of what a good clustering is in different ways, and with different levels of success. An intuitive explanation of what a good clustering

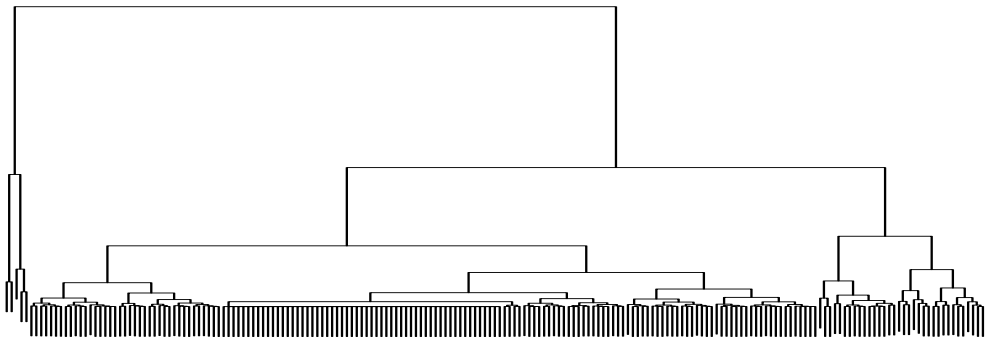


Figure 2: An example dendrogram. Each leaf represents a single node and the edges between the nodes represent the separation of components at a certain time step represented by the y-axis. A horizontal cut through the dendrogram induces a clustering of the graph.

should look like is a very subjective matter, since it depends largely on what element of the structure one hopes to find. There seem to be a few common points to most justifications of good clusterings:

- The subgraph induced by a cluster should have a large density, i.e. the nodes inside the cluster should be closely connected.
- Edges between different clusters should be minimized.
- The size of the clusters should be representative of the structure.

Of the large number of indices available to measure the quality of a community structure (See for example [3] and [4]) four will be used in this paper:

- *Modularity [3]*: This index compares the number of intra-cluster edges to the expected number given the edge-distribution probability of the analyzed graph as can be seen in Equation 2. Due to this the result can be either positive or negative, a positive modularity indicating that the agglomeration in the given clusters is higher than expected given a random graph, a negative value the opposite. Thus the modularity can be used as an indicator for clusterings and the values range between -0.5 and 1. Finding a clustering of an optimal modularity for a given graph has been proven to be NP-Complete [7].
- *Inter-cluster conductance [6]*: The inter-cluster conductance is defined by means of the conductance of cuts. This is the ratio between the size of the cut and the size of the

Let our Clustering be $\mathcal{C} = \{C_1, \ldots, C_k\}$ with each C_i being the set of nodes of that community, thus making them pairwise disjoint. Then the modularity is defined as:

$$\Phi_{Mod}(\mathcal{C}) := \sum_{C \in \mathcal{C}} \left[\frac{|E(c)|}{m} - \left(\frac{|E(c)| + \sum_{C' \in \mathcal{C}} |E(c, c')|}{2m} \right)^2 \right]$$

Equation 2: Definition of modularity

two communities induced by it. Using this one can further define the conductance of a graph, which is the minimum conductance value over all its cuts. The inter-cluster conductance is then, as seen in Equation 3, the maximum conductance value over all cuts induced by the communities. A small inter-cluster conductance value is usually an indicator for a community that has strong connections outside which may indicate that the clustering is too fine. The inter-cluster conductance reaches 0 if either all the nodes are inside a single cluster (which is problematic if we are analyzing a clique), or if there is a single cluster which has no intra-cluster edges and at least one inter-cluster edge.

Let us first define the conductance of a cut $(C, V \backslash C)$.

$$\Psi_{Inter}(C) = \begin{cases} 1, \text{if } C \in \{\emptyset, V\} \\ 0, \text{if } C \notin \{\emptyset, V\} \wedge \overline{m}(\{\mathcal{C}, V \backslash C\}) = 0 \\ \frac{\overline{m}(\{\mathcal{C}, V \backslash C\})}{\min(\sum_{v \in C} \deg v, \sum_{v \in V \backslash C} \deg v)}, \text{otherwise} \end{cases}$$

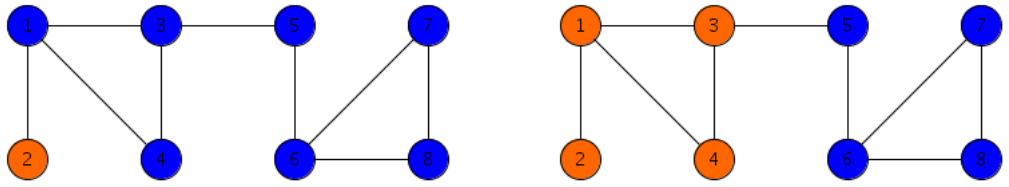
Using this we can now define the inter-cluster conductance as

$$\Phi_{Inter}(\mathcal{C}) := 1 - \max_{C \in \mathcal{C}} \Psi_{Inter}(C)$$

Equation 3: Definition of inter-cluster conductance

- *Coverage [3]*: The *coverage* index is simply the ratio of intra-cluster edges (i.e. edges within communities) to m, the number of edges in the graph (The definition can bee seen in Equation 4). This has a couple of minor problems, as the resulting value says little about the density of each community.

  As can bee seen in Figure 3 both clusterings result in the same coverage value (8/9), although we would clearly argue that the right one is, given our intuition, a better clustering. Another big problem is that if all nodes are in one big cluster the coverage is optimal, i.e. 1. This means that, unless we explicitly forbid this, our algorithm will always return this clustering (or at least with each original component in one cluster, which also shares this problem).



(a) An obviously bad clustering of a simple graph that still manages to achieve a good coverage value but fails when it comes to performance.

(b) This clustering of the same graph is noticeable better and achieves a good performance value. The coverage is still the same.

Figure 3: Example clusterings of a simple graph

The coverage can be defined as

$$\Phi_{Cov}(\mathcal{C}) := \frac{m(\mathcal{C})}{m}$$

Equation 4: Definition of coverage

- *Performance [3]*: The performance index is a variation of the coverage index which is calculated by normalizing the amount of correctly clustered pairs of nodes, with correct pairs being existing intra-cluster edges and all missing inter-cluster edges (See Equation 1). Although this might appear to actually correct some of the problems encountered with coverage there are still some issues when using this index. Still, due to it also taking into account *correct* node pairs instead of just the inter-cluster edges it works much better than coverage, as can be seen if applied to the same graph as we used in the Coverage example (Again Figure 3). This time the clustering with the singleton results in a worse performance value: 0.5. The second clustering, which intuitively looks pretty good, also gets a good rating from the performance index (23/28).

Performance can be defined as:

$$\Phi_{Perf}(\mathcal{C}) := \frac{m(\mathcal{C}) + \sum_{\{v,w\}\notin E, v\in C_i, w\in C_j, i\neq j} 1}{1/2n(n-1)}$$

Equation 5: Definition of performance

The K-Betweenness algorithm consists of two crucial ideas. Firstly the betweenness of edges is only computed every k steps. This results in a performance gain by the factor k. Secondly not all cuts of the dendrogram are analyzed or even computed. Again this results in a significant performance gain as the amount of loop iterations can be drastically reduced. These two simple modifications result in a significant speed-up of the algorithm

in exchange for a marginal quality loss as we will show in the following. One thing to keep in mind though is that the evidence is purely empirical, and thus no guarantees about the quality can be given as of this moment.

Algorithm: Newman-Girvan algorithm

Data: Graph $G = (E, V)$;
Quality index function Φ(Clustering)
Result: Clustering $\mathcal{C}$

```
1  C ← ω(G);
2  q ← Φ(C);
3  G' = (E', V') ← G;
4  while |E'| > 0 do
5  |   Calculate edge-betweenness of G';
6  |   ψ ← edge ∈ E' with max. betweenness;
7  |   E' ← E'\{ψ};
8  |   if Φ(ω(G')) > q then
9  |   |   C ← ω(G');
10 |   |   q ← Φ(C);
11 |   end
12 end
```

Algorithm: K-Betweenness

Data: Graph $G = (E, V)$;
Quality index function Φ(Clustering);
constant k;
range $r = [r_1, r_2]$
Result: Clustering $\mathcal{C}$

$\mathcal{C} \leftarrow \omega(G)$;
$q \leftarrow \Phi(\mathcal{C})$;
$G' = (E', V') \leftarrow G$;
$t \leftarrow 0$;
while $|E'| > 0 \wedge t \leq max(r)$ **do**
    if $t \ mod \ k = 0$ **then**
        Calculate edge-betweenness of G';
    end
    $\psi \leftarrow$ edge $\in E'$ with max. betweenness;
    $E' \leftarrow E' \backslash \{\psi\}$;
    if $t \in r \wedge \Phi(\omega(G')) > q$ **then**
        $\mathcal{C} \leftarrow \omega(G')$;
        $q \leftarrow \Phi(\mathcal{C})$;
    end
    $t \leftarrow t + 1$;
end

Type of graph	Number of nodes	Total number of graphs
Trees	1-283	283
Complete graphs	1-50	50
Attractor-generated graphs	1-200	200
Real world examples	34-522	6

Table 1: Types of graphs used for empirical evidence

2 Tests

As mentioned in the previous section the two modifications in the K-Betweenness algorithm are the computation of the betweenness every k'th step, k being a constant, and only analyzing the clusterings of a range of time-steps. The obvious question is what a good choice of k would be, as well as what the optimal range, lets call it $r = [r_1, r_2]$ from now on, for the analysis of the time-steps would be. In order to find good values for k and r a number of statistical tests were run against several types of graphs (See Table 1). The machine used for conducting these tests was an AMD Athlon(tm) XP 2000+ with 256 MByte of RAM running a Debian sid system. The language used for the algorithm implementation was Java (J2RE 1.5) using the yWorks yFiles library. The results of the tests will be presented in this section along with the introduction of the used concepts.

2.1 Distance of clusterings

In order to measure the similarity between the clusterings of the original *Newman-Girvan algorithm* and our variant we need a metric. One possibility is the use of quality indices already mentioned, but these wouldn't represent the resemblance of the two clusterings as even a difference of a single node can wreak havoc when it comes to the clustering quality. Thus the following metric was used: Let $G^* = (E^*, V)$ and $G' = (E', V)$ be the two graphs (whose components induce a clustering) to be compared based on $G = (E, V)$, which is the original graph. Additionally $\Phi^* : E* \rightarrow \mathbb{R}$ ($\Phi' : E \rightarrow \mathbb{R}$) assigns each edge its betweenness value (0 if the edge isn't in the clustering). Then the distance is $\Delta(G^*, G') := \sum_{e \in E} |\Phi^* - \Phi'|$. In Figure 4 two different graphs can be seen which differ in two edges. Since the distance is the difference of the betweenness value summed over all edges the right component can be ignored as it is identical in both graphs. The betweenness difference in the other edges $(2.5 + 3.5 + 2.5 + 4 + 1.5)$ amounts to 16.

Thus the distance between two clusterings of the same graph is simply the absolute difference between all their edges' betweenness.

When the distance between two consecutive clusterings in the *Girvan-Newman algorithm* is plotted, two kinds of graphs can be seen. On most graphs a sharp fall of the distance with ever-decreasing spikes can be seen (See Figure 5(a)). The beginning distance being the highest betweenness of all edges, it is no surprise that this value is near the maximum. Each time-step there are two possibilities:

One is that the removal of the edge separates the component. Lets call the number of

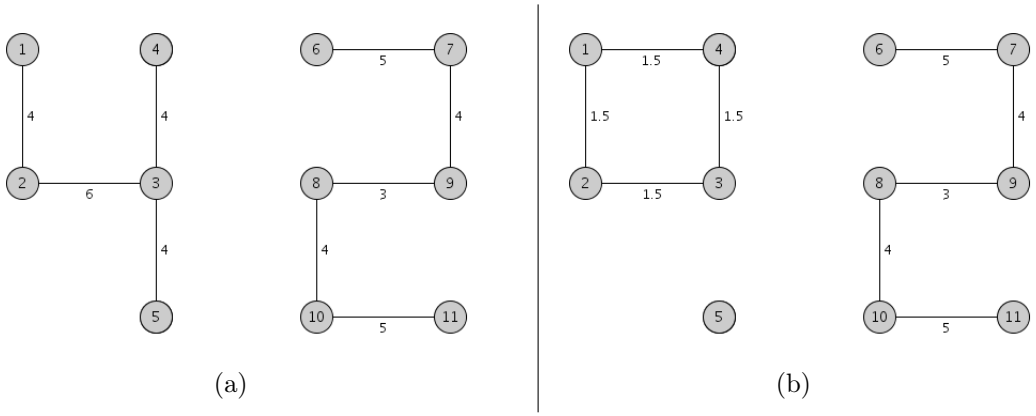


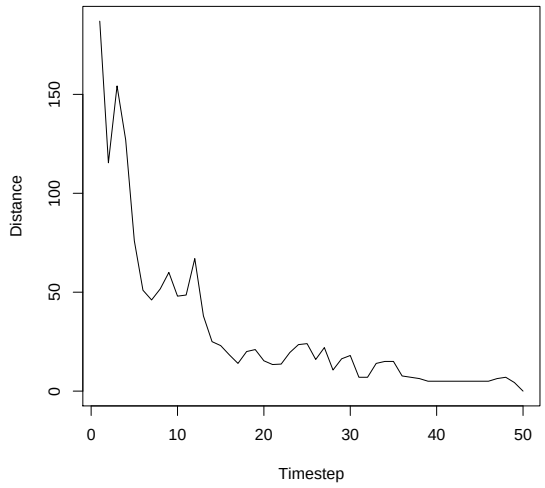
Figure 4: These two graphs differ in two edges. This leads to a different betweenness distribution within the affected graph components which leads to a total difference of 16.

nodes in the new components n_1 and n_2, and the removed edge e. Therefore the minimum distance would be $2 \cdot n_1 \cdot n_2$ and the maximum being bounded by $\frac{n_1 \cdot n_2 (n_1+n_2-2)}{2} + C_B(e)$ (See Equation 6). Now while this is a tremendous distance, and it accounts for the spikes in the decrease, the overall betweenness of all nodes is greatly reduced and thus the distance in the next steps is much smaller.

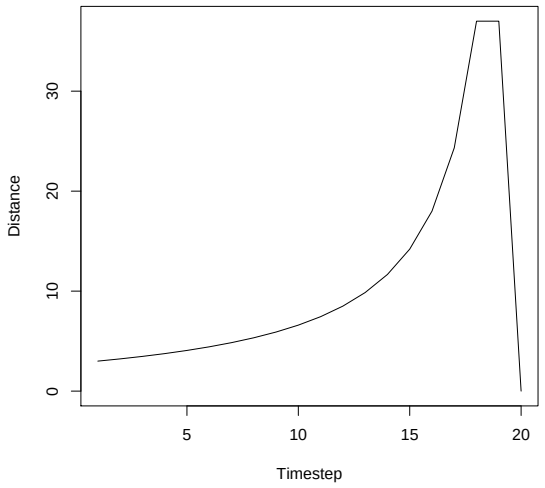
If, on the other hand, the removal of e doesn't cause a separation of components then the distance is between $3 \cdot C_B(e)$ and $C_B(e) \cdot n'$, as the betweenness value of e is added to other edges. Since these other edges have a smaller betweenness value, and typically the betweenness value of e is distributed evenly across what later will become inter-cluster edges, the maximum edge in the next time-step is much smaller, thus explaining the steady decrease, although the fast decrease stems, as already mentioned, from the separation of components with its cut of overall betweenness.

As mentioned in the previous paragraph not all graphs show the described development. The big exception are complete, or nearly complete graphs. In these graphs the betweenness of all edges is initially equal (1). Thus the removal of the first edge actually increases the weight of the others, and this continues at an ever-increasing rate leading to exponential growth which, after culminating, quickly falls to 0. (See Figure 5(b)).

The following results shed light on the development of the distance between the evolution of the clusterings in the course of the time-steps compared to a static clustering, in this case the initial graph as can be seen in Figure 6 *left*. Again there are two distinct cases, and again the exceptional case are the near-complete graphs. As can be seen in the example graph the distance increases in a clearly logarithmic fashion with little deviation which seems dependent on the density of the graph. This corresponds to the falling distance between consecutive steps, although the spikes have been smoothed out. This is due to the separation of components not having such a strong effect, since the removal of weight has been countered in previous time-steps with the increase of weight in the edges.



(a) This plot shows the distance between consecutive time-steps in a random attractor-generated graph with 50 nodes. A sharp fall can be seen with spikes symbolizing the separation of components.



(b) Here we see the distance of consecutive time-steps in a complete graph with 20 nodes. Each removed edge increases the betweenness value of all the others leading to an exponential growth until the first separation of components occurs.

Figure 5: Distance of consecutive time-steps

If the removal of an edge e causes a separation of components (with the intra-component edges named E_{N_1}/E_{N_2} and n_1/n_2 nodes) we have an upper bound of:

$$\begin{aligned}\Delta = & \sum_{f\in E}|\Phi^* - \Phi'| \\ = & \sum_{f\in E_{N_1}}|\Phi^* - \Phi'| + \sum_{f\in E_{N_2}}|\Phi^* - \Phi'| + |\underbrace{\Phi^*(e)}_{C_B(e)} - |\underbrace{\Phi'(e)}_{0}| \\ = & \sum_{f\in E_{N_1}}|\Phi^* - \Phi'| + \sum_{f\in E_{N_2}}|\Phi^* - \Phi'| + C_B(e)\end{aligned}$$

Each shortest path between elements of the two components had to go through e which was the only interconnecting edge. Obviously only a single node from the first cluster had a distance of $n-1$ to the edge e. Only 2 could have a distance $\geq n-2$, 3 over$\geq n-4$, and so on. This would be the case of a single path connecting all the nodes. Additionally we can note that $C_B(e)$ is equal to $2 * n_1 * n_2$. Using that we have:

$$\sum_{f\in E_{N_1}}|\Phi^* - \Phi'| \leq n_2 \cdot \sum_{i=1}^{n_1-1} i \leq \frac{1}{2}n_1 \cdot n_2(n_1-1)$$

And therefore (using the same for the second component):

$$\begin{aligned}\Delta \leq & \ \tfrac{1}{2}n_1 \cdot n_2(n_2-1) + \tfrac{1}{2}n_2 \cdot n_1(n_1-1) + 2 * n_1 * n_2 \\ \Delta \leq & \ \tfrac{n_1 \cdot n_2(n_1+n_2)}{2}\end{aligned}$$

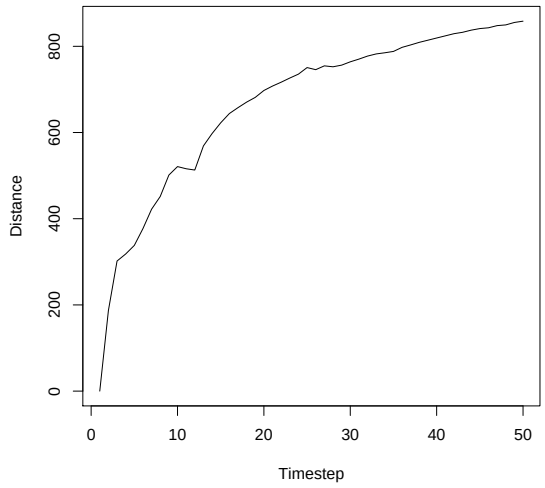
A lower bound for Δ is $C_B(e) = 2 \cdot n_1 \cdot n_2$ as each path between the nodes of the different components need to cross through e.

If the removal of an edge e doesn't cause a separation of components we have a lower bound of $3 \cdot C_B(e)$ (triangle-like construct). The exact value of $C_B(e)$ can not be determined in this case, since it depends greatly on the structure of the graph. But since it is easily computable we can use this information to make predictions on Δ early on.

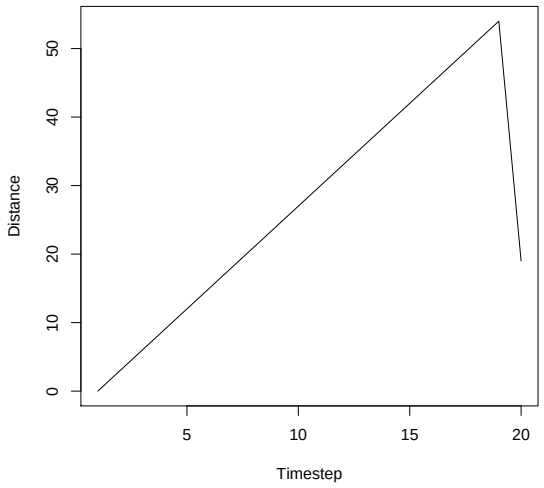
For completeness let us note that we have an upper bound of $C_B(e) \cdot n'$ with n' being the number of nodes.

Equation 6: Upper & lower bounds of distance in consecutive time-steps

In the case of complete graphs a linear increase can be seen (see again Figure 6 *right*). This stems from the fact that the removal of each edge just changes the overall betweenness by two until the separation of components begins. Later on we see a dramatic fall, as the removal of edges lets the distance approach the total initial betweenness of all edges.

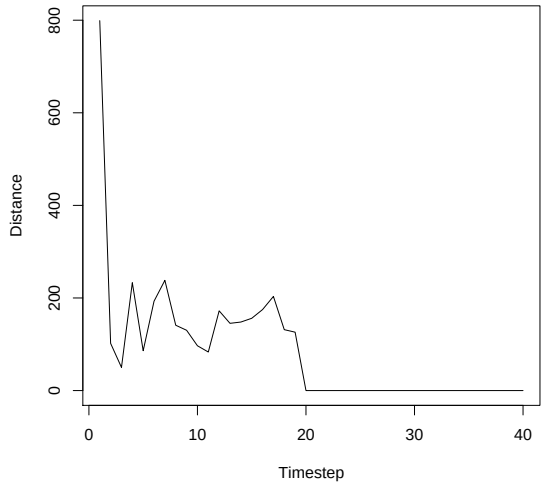


(a) This plot shows the development of the the distance compared to the original graph in a random attractor-generated graph with 50 nodes. As can be seen the development is nearly logarithmic.

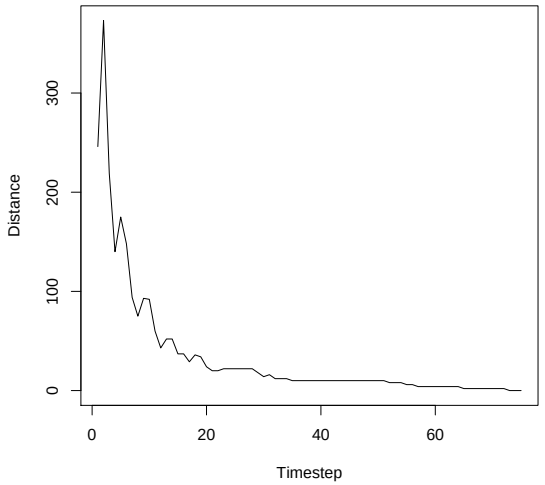


(b) Here the development of the distance in relation to the original graph in a complete graph with 20 nodes can be seen. The increase is linear (each edge removal increases overall betweenness by two) until the components begin to separate.

Figure 6: Distance of time-steps relative to original graph



(a) This plot shows the distance in an attractor-generated graph with 200 nodes for the common time-steps (i.e. same number of removed items) for $k_1 = 1$ and $k_2 = 5$.



(b) Here we see the distance in a random tree with 150 nodes in the common time-steps for $k_1 = 2$ and $k_2 = 5$.

Figure 7: Distance between two constants

2.2 Relation between distance and k

After the short look at the development of distance within the *Newman-Girvan algorithm* now the distance between the clusterings at different time-steps for two different k's in the K-Betweenness algorithm shall be examined. Let the two constants be k_1 and k_2. For the analysis only the $GCD(k_1, k_2)/k_1 + 1$'th (resp. $GCD(k_1, k_2)/k_2 + 1$'th) time-step will be compared, so that the same number of edges is removed.

Yet again, two different types of graphs appear, though luckily they are as easy to categorize as in the previous subsection. Lets start with the easiest case: Nearly complete graphs. In this type of graph the distance depends on the edge selected if several have the same weight. Here the first edge based on an arbitrary order was selected. Thus the distance throughout the time-steps was 0, with marginal deviations if the graph was near-complete.

The most common development can be seen in Figure 7. Here we can see a big initial distance, which isn't surprising if we consider the effect that a single edge has at the beginning of the algorithm as we saw in the above section. This distance then quickly falls and oscillates around the minimum.

The previous section makes this development plausible: The impact of selecting a different edge in the beginning is critical, as the change between the distance of two consecutive time-steps, and thus of the nodes appearing in that time-step, is huge. The probability

of selecting an edge already removed by the other algorithm increases with time, and the selection of a different edge has less impact, but creates the oscillation.

2.3 Good cuts in dendrograms

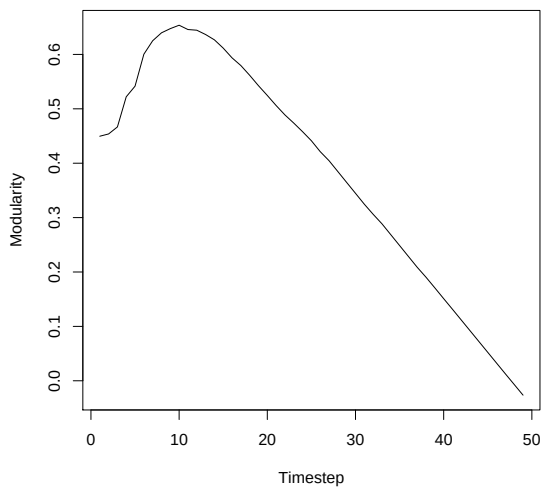
Up to this point we have been considering the effect of k on the distance and left the quality indices, as well as r (the range of time-steps for good clusterings) aside. We will focus on these two issues in this section by analyzing how the different quality indices develop for the clusterings in the course of the time-steps. Although the results presented here are based on $k = 1$ they are still valid for higher k's, and as such are always presented in percentages of the total amount of steps. The reason for this is simple: the number of edges removed at a certain percentage of the steps (obviously they need to match) is the same, and with it the quality drops/raises are similar.

2.3.1 Modularity

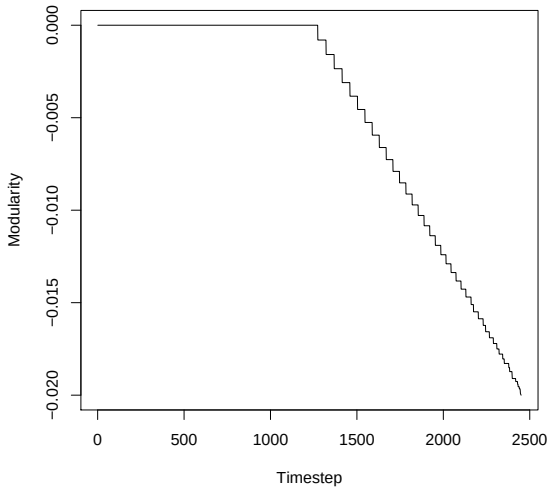
The first quality index that we will take a look at is *modularity*. A plot of the different clusterings of the *Newman-Girvan algorithm* show (See Figure 8) a linear fall of the modularity after the best clustering has been found, although at its maximum there is a short (in all tested graphs within 10% of the total number of steps) oscillation. The point at which this optimal cut of the dendrogram happens is strongly dependent on the inner structure of the graph, with complete graphs being one extreme (optimum at time-step one) and graphs whose best clustering consists of mostly singletons being at the other end of the spectrum. On most graphs tested (the trees, real world examples and the Attractor-generated ones) though the maximum is between these extremes, with the initial quality growth resembling a step-function. Usually, except for the extremes already described and which can easily be filtered out using the density, the peak is between 5% and 50% steps, with trees being mostly on the edge of the first quarter.

2.3.2 Inter-cluster conductance

The results here are, yet again, varied. In complete graphs the conductance remains at a constant 0 since the graph is always inside one big cluster until the first node separates from the main group and is placed in a singleton. Both cases lead to an inter-cluster conductance of 0. In other graphs though there is usually a short jump to a high value near the beginning if there is a single component in the graph, as the first few time-steps usually remove very obvious inter-cluster edges, and once at least two clearly cut components exist the conductance value greatly increases. If there is more than one component to begin with, the conductance is, using the definition presented in the introduction, already 1. As more edges get removed the cuts induced by the clustering become worse and worse. And since we are looking at the worst of all possible cuts, once a bad one occurs, which depends on the graph structure but is, at least based on the Attractor/Tree sample size, within the first 20% of the time-steps, the conductance value drops to nearly zero. So we

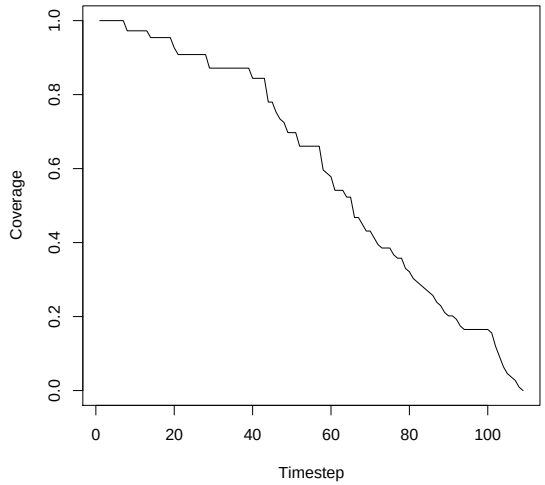


(a) This plot shows the modularity in the course of the time-steps in a tree with 50 nodes. As can be seen there is a linear fall once the peak has been reached, which is typically in the first 50% in both, trees and attractor-generated graphs.

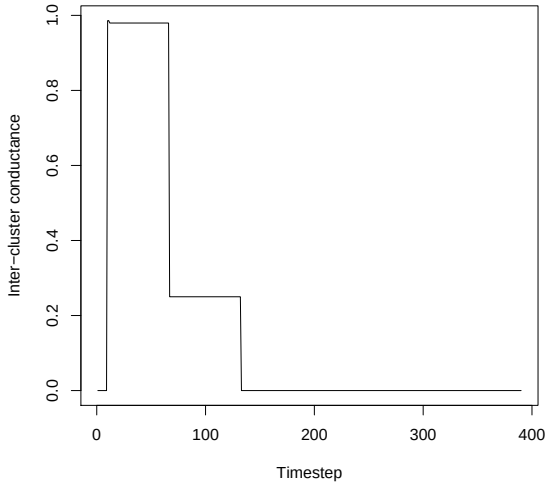


(b) This plot shows the modularity development in a complete graph with 50 nodes. Since there is no separation of components for a long time things are pretty constant until the first separation begins with a drop in quality which continues as more and more nodes are placed in singletons.

Figure 8: Development of modularity in the *Newman-Girvan* algorithm



(a) This plot shows a typical development of the coverage in non-complete graphs such as this attractor-generated example with 50 nodes. We begin with an optimal coverage value (the components as clusters) and then slowly loose quality as we gain more and more inter-cluster edges.



(b) Here we see the development of the inter-cluster conductance in an attractor-generated graph with 100 nodes. It has a single component in the beginning, which is the reason why it starts up at 0 (otherwise it would have a value of 1). With the first cut it reaches a good value that then can only become worse with each bad cut.

Figure 9: Development of coverage and inter-cluster conductance in the *Newman-Girvan* algorithm

basically only need the first two clusterings two find an optimal one based on inter-cluster conductance.

2.3.3 Coverage

In the test results one can clearly see that if the graphs are either trees or complete graphs the coverage value decreases linearly, as removing an edge (in complete graphs from the point on where the separation of components begins) simply decreases the coverage by $1/n$. In the case of Attractor-generated graphs very different kinds of development can be seen, with linear-like quality decrease to very rapid decreases. The most notable feature is that the earlier the quality drops, the smaller the value, but this should not come as a surprise given the way the algorithm works. An early separation of components, which is what would induce a quality drop, can only have a small number of edges between those components. Therefore the coverage would only fall slightly. If, on the other hand, the separation of components happens late, then usually a higher number of edges cross the community boundaries and so, once the separation takes place, a sharp drop of the coverage can be seen. But one thing is certain, the coverage is monotonously falling, since

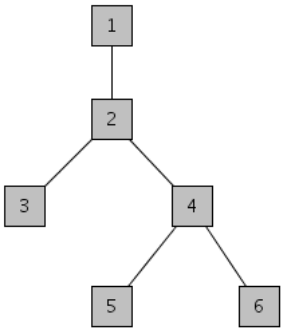


Figure 10: A low density graph such as a tree has a high amount of missing edges. For this reason if each node is placed in a singleton pretty good performance values can be achieved (the lower the density the better the performance).

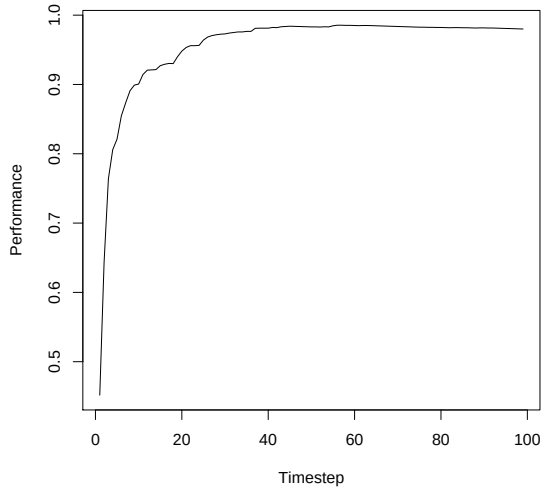
the separation of components can only decrease the amount of intra-cluster edges.

2.3.4 Performance

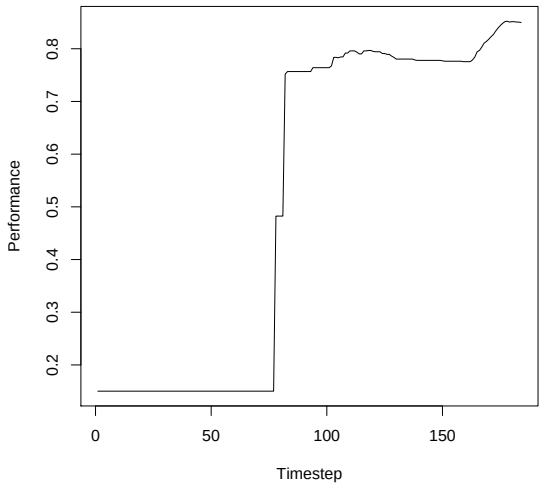
The performance index is, as was already mentioned, a variation of the coverage index which can be achieved by normalizing the amount of correct pairs of nodes. In the case of graphs with a very high density, specially complete graphs, this means that the result is (nearly) exactly like coverage, i.e. a linear decrease with maybe a bit of oscillation in the beginning. The optimal value in this case is within the first 10%. When it comes to the other types of graph there is quite a difference. Most often (in around 70% of all tested graphs, including all trees) a sharp increase in quality can be seen in the first few time-steps (See figure 11 *left*).

In the other cases a very different development takes place. The clusterings induced by the removal of edges rate very poorly in the performance scale. Once the removal of edges reaches a critical point it begins with creating very fine-grained communities consisting of only a few nodes. Since we are talking about a low to average density, the other ones develop in a coverage-like manner, this means that the amount of correct pairs of nodes greatly increases until finally culminating, once all nodes are clustered in singletons, in a performance value of $1 - m/(0.5)n * (n - 1))$. This is also the reason why trees perform so well and are in the 70% of the graphs named above. As an example look at Figure 10 which clearly depicts this peculiarity of performance. If we suppose the clustering is down to singletons then we have 25 non-existent inter-cluster edges, which greatly outnumber the 5 existing ones, yielding a performance value of 0.83.

Therefore we can analyze the last 10% of the clusterings to achieve an optimal value here.



(a) This plot shows the development of the performance in a tree with 100 nodes. The first few separations dramatically increase the index value and later on the value continues to rise due to the missing inter-cluster edges as more and more separations take place.



(b) Here we can see another development of performance in an attractor-generated graph with 50 nodes. Once the first separation takes place the quality rises rapidly and then continues to rise to reach its final value.

Figure 11: Development of performance in the *Newman-Girvan* algorithm

2.4 Relation between quality indices and k

The most interesting observation on the behavior of k, at least within the topic discussed here, is how it evolves in relation the different quality indices, as these are generally used to select a good clustering. Judging from the information on the distance related to k we can probably assume that a small value won't result in a big change of quality. This shall now be examined for each of the quality indices named above.

2.4.1 Modularity

With respect to modularity the graphs separate into two very distinct categories:

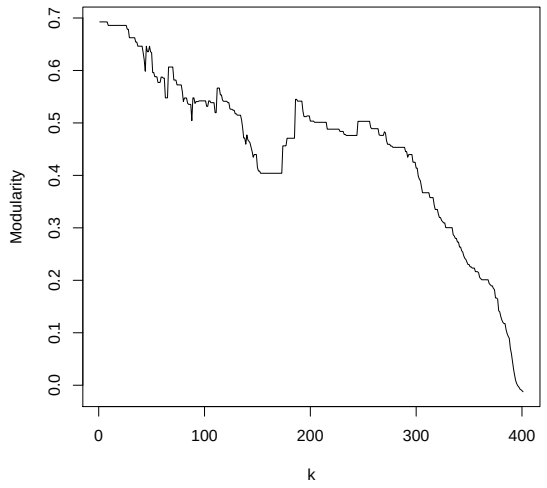
One one hand we have a large group of graphs that behave very similarly and of which 12(a) is an example. As can be seen they have a small period of steadiness with a minimum of oscillation in the beginning followed by a slow decrease. Now this period of steadiness is exactly the range of acceptable k's for our algorithm as it is in here that there is a minimum amount of quality loss. The evidence from the tested graphs indicates a maximum k of $\frac{n}{10}$ is acceptable with a lower value offering only a marginal quality increase. As usual the different types of graphs show a slight difference.

In complete graphs the variance within the above mentioned range was always 0, which is not a big surprise given that the best clustering always contains all nodes. In trees the variance in the development can be seen in 13(a). Here the maximal variance of all k's up to $\frac{n}{10}$ in a certain graph is smaller than 0.1 except in the case of 4 trees which range slightly higher. Even these trees follow the pattern of Figure 12(a).

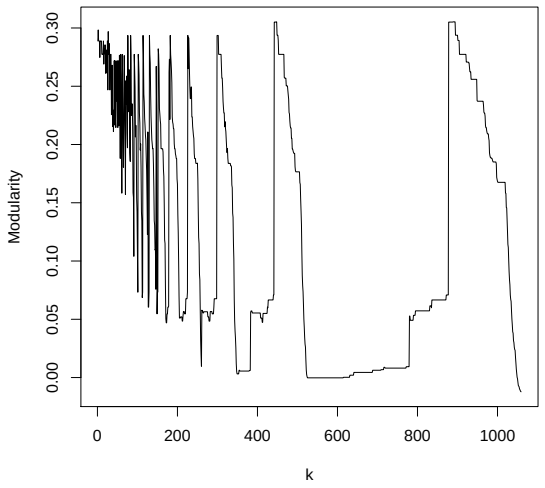
When it comes to the attractor generated graphs things start changing though. Again most graphs do have the pattern mentioned above, but now a few graphs appear (roughly 2.5%) that have a worse variance within that k-range than 0.1. A look at the statistics from these graphs reveals the following: The modularity value was always below 0.4. Also the development of these graphs shows a dramatically different behavior from the one we saw before. It is much more chaotic and show little similarity to one another. As of this moment we have not been able to find a distinguishing feature of these graphs so as to recognize before running the algorithm. Figure 12(b) shows an example development of such a chaotic graph. The real world examples show the aforementioned pattern with a small variance, but due to the small number of these it is to be expected that chaotic developments will appear here as well.

If the quality loss described above is acceptable then the value of $\frac{n}{10}$ for k allows a dramatic increase in the runtime. As described in the introduction each iteration in the *Girvan-Newman algorithm* has a runtime of $O(m \cdot n)$ which leads to a total runtime of $O(m^2 \cdot n)$ if the quality index computation is not considered. Since the removal of additional edges within the iteration step is dominated by the computation of the betweenness and we only compute every k'th step we reach a runtime $O(m \cdot n^2)$ in addition to the modularity computation.

In the testing system this resulted, when run at a graph with 100 nodes and 1400 edges, in a runtime of 4.32 seconds with $k = 10$ compared to the 24.96 seconds needed if $k = 1$.

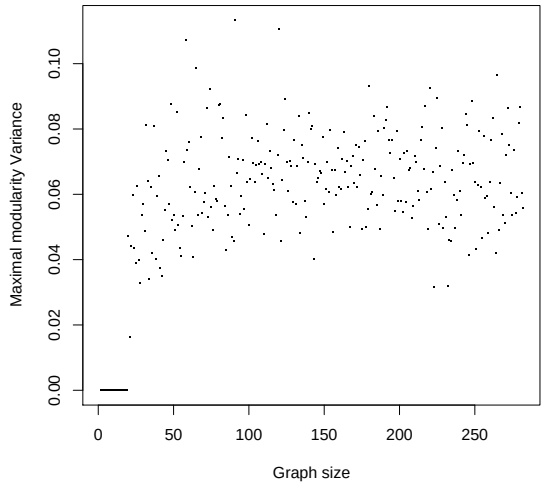


(a) Here we can see a "normal" example of the development with increasing k's. The graph was attractor-generated and had 100 nodes. As can be seen the quality loss if $k = 10$ is minimal.

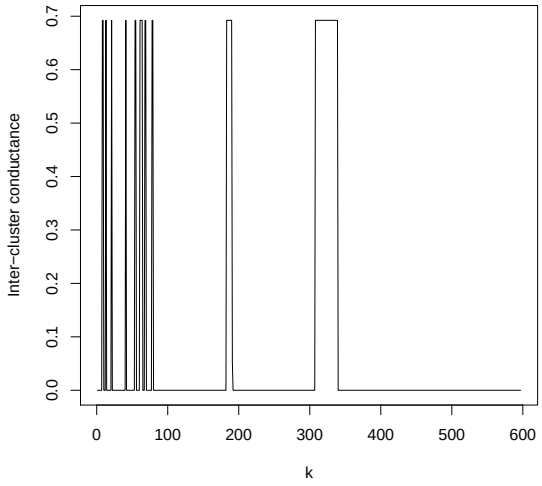


(b) This plot shows one of the extremely chaotic examples: An attractor-generated graph with 93 nodes. As can be seen it oscillates wildly in ever increasing amplitudes.

Figure 12: Development of modularity. The x-axis represents the used k, the y-axis represents the modularity of the best clustering generated using that k.



(a) Here we can see the variance of the modularity value in different trees for the accepted range for k ($n/10$). Each dot represents the variance of the worst performing k for that graph.



(b) This plot shows the development of inter-cluster conductance with increasing values of k in an attractor-generated graph with 100 nodes. As can be seen there is a rapid exchange between good and bad values at the beginning which appears to be random. The distance between those peaks differs wildly with values of up to one quarter of the total k's in some graphs.

Figure 13: Development of graphs with using the performance and coverage indices.

2.4.2 Inter-cluster conductance

As the inter-cluster conductance index concentrates on finding the worst possible cut induced by the clustering the use of a value of k other than 1 is a high risk to undertake. Often choosing a higher value of k leads to at least one cut that is, as measured by the inter-cluster conductance, catastrophic. In the case of complete graphs, for reasons already discussed in previous sections, the result is always 0. In the case of attractor-generated graphs (as well as in the real-world examples) the inter-cluster conductance using a $k > 1$ is nearly always 0 as some cuts only have inter-cluster edges (singletons). For this same reason all the trees end up with an inter-cluster conductance of 0 if such a k is used. In the attractor-generated graphs from time to time there exists a k that avoids creating such a bad cut (4% of the tested graphs, none of the real-world examples and all with a very high inter-cluster conductance in the region of 0.94). But it is just a game of chance and very unpredictable as can be seen in Figure 13(b). Therefore we suggest sticking to 1 if this index is used as a quality metric.

2.4.3 Coverage

There is a remarkable similarity in the development of coverage to the one found in the case of modularity when the Attractor-generated graphs are used (See Figure 14(a)), but we have to note that in the case of coverage we are just analyzing the clustering of the first component separation, as the quality is monotonously falling and we forbid the use of the clustering induced by the original components. Again we find mostly a small period of steadiness in the beginning with an oscillation that in this case is slightly higher. Afterwards this slowly decreases until it reaches 0. Typically the range of acceptable k's is within $(1, n/20)$ with 90% of the tested graphs having a maximal deviation in the interval $k \in [1, n/20]$ of 0.1 when compared to the case of $k = 1$.

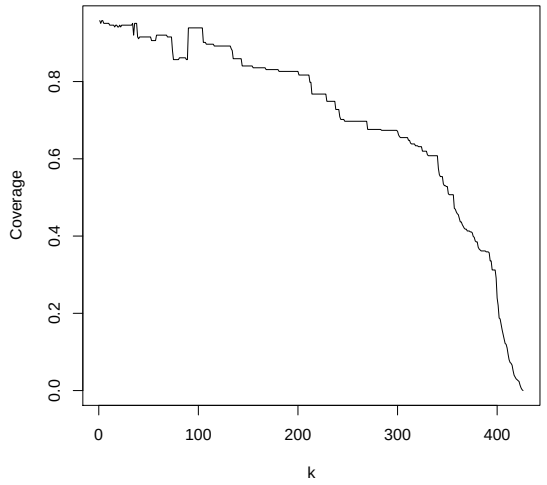
And again there are a number of cases, in which the development is chaotic. Here the coverage swings within values close to the maximum and near 0. To counter this effect it is possible to actually do runs with consecutive k's, as that reduces the probability of hitting one of those lows while still being a considerable improvement over the runtime when using $k = 1$.

In the case of trees the quality is exactly $1 - k/m$, as the first iteration removes k edges. Therefore using $k = n/20$ we have a coverage of $1 - \frac{n/20}{n-1} = 0.95 - \frac{1}{n-1}$.

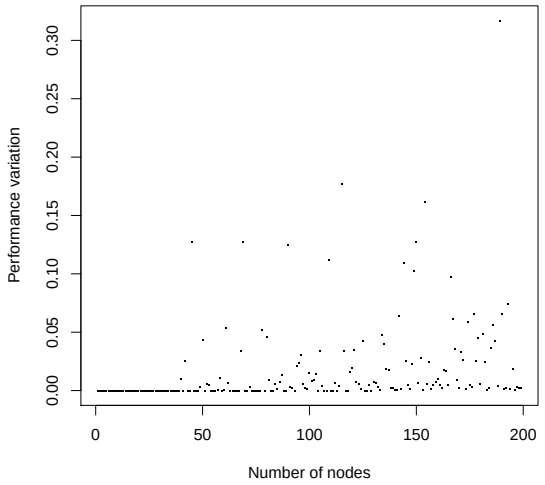
In the case of complete graphs we have, as always with coverage, a quality of 1.

2.4.4 Performance

Now performance delivers pretty unusual results. In the case of complete graphs it is the usual deal, it doesn't matter what k is chosen, the result is always a single cluster because of the definition of performance. Most of the tested graphs from the group of attractor-generated graphs, trees and real world examples were similar to the one seen in Figure 15(a). A pretty long period of steadiness with only a minor oscillation, but in the period of steadiness there are a few very sharp drops of quality for only a few time-steps. The



(a) Here we see the development of coverage in an attractor-generated graph with 100 nodes. Similar to the modularity case there is a small period of oscillation, typically within $n/20$.



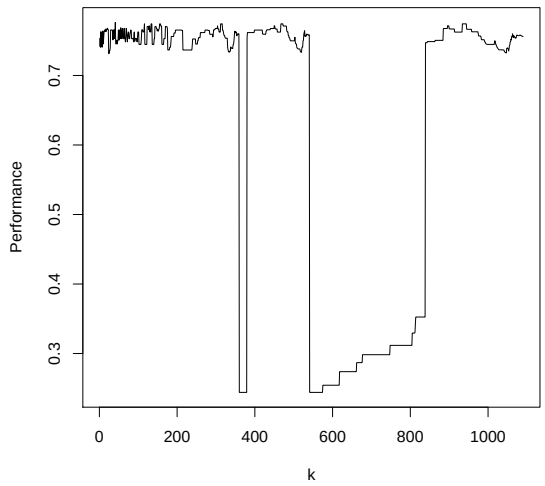
(b) This plot shows the maximal variance of performance for the tested graphs. Each dot symbolizes the difference between the worst performing k within the accepted range ($n/20$) for a specific graph.

Figure 14:

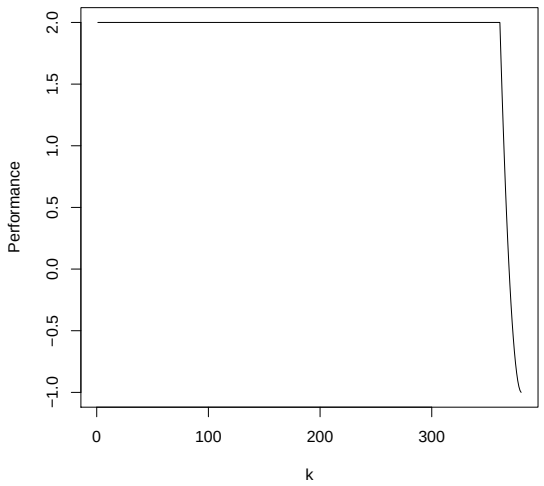
amount of these drops, which only last for a few k's ranges from 0 to 4 in the tested graphs and appear in the second quarter of the steps. The reason for this interesting phenomenon is as of this moment unknown.

Afterwards we find a couple of small drops and then another increase. Now this increase in the end is, like in the previous section, probably due to the fact that the amount of correct pairs increases radically when a small density occurs. Supporting this is the fact that in graphs with a big density the results look more like in Figure 15(b). Now, the interesting thing is why there is such a period of steadiness and why it suddenly ends. A possible explanation would be that there is a point of equilibrium in the number of correct nodes. While up to a large k strongly linked groups of nodes still remain together, and the errors are mostly on sparse areas of the graph, there comes a point where this changes but the amount of non-existing inter-cluster nodes still isn't big enough to counter this effect. Only later on, when the communities have become small enough for this effect to really pick up the performance rapidly rises again. Of course, this is again only the case in graphs that do not have a large density.

Choosing values for k within the first 20% of the time-steps ($n/20$) in order to avoid the sharp drops delivers a maximal variance of 0.1 in 93% of the graphs (See Figure 14(b)).



(a) This plot shows the development of performance with increasing values of k in an attractor-generated graph with 100 nodes. As with most graphs there is a large period of oscillation near the optimum (up until around $m/2$) with small drops to 0 in between. In the tested graphs these always appeared after the first $m/20$ ($m/17.3$ to be exact).



(b) This plot shows the development of the performance in a complete graph with 28 nodes. As can be seen the performance value stays at its top value until the very end.

Figure 15: Development of performance with increasing k in a single graph

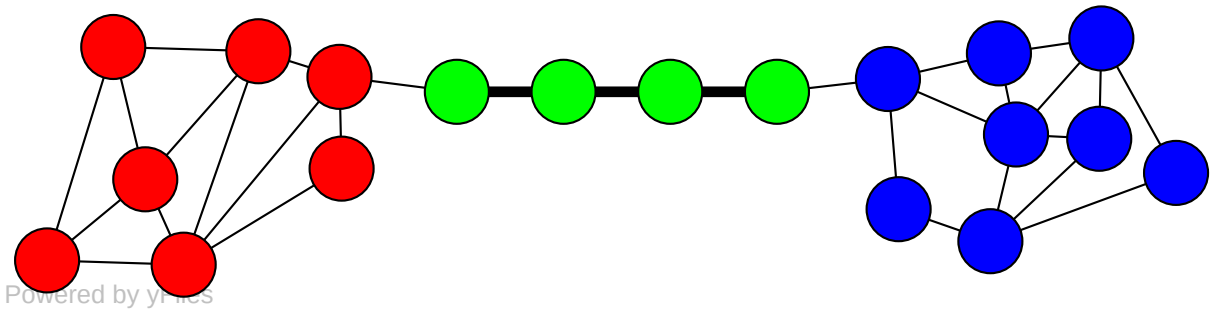


Figure 16: Using a big k will make several of the edges in the *chain of nodes* (green) vanish thus creating a number of singletons. This can easily be identified afterwards and those nodes can then be appended to a neighboring cluster.

3 Optimizing K-Betweenness

In addition to the already mentioned two changes to the *Newman-Girvan* algorithm there are a small number of additional changes that usually bring with them an increase in the quality of the found clustering. A look at the tested graphs shows that strongly connected groups of nodes, which later commonly become separate community structures, are often connected by a *chain* of nodes (See Figure 16). The edges of those chains being obvious inter-cluster candidates they are selected for removal by the algorithm. In the case of $k = 1$ this represents no problem, as one of those edges is removed and the edge-betweenness value of the others drops, but if the k is higher then several of the edges are removed. This creates a number of singletons which increase the number of community structures.

The effect of this on the quality varies with the different quality indices. In modularity, coverage and performance the effect is a negative one, although not tragically so. In the case of inter-cluster conductance though this means the quality drops to 0, something to be avoided at all costs.

This information helps to increase the quality index of clusterings created with a high k value in a pretty easy way: Each singleton is *appended* to a connected community structure. Of course, one could further reduce this deviation by simply checking which community structure the singletons should be appended to, but that could result in a significant increase of the runtime, something which K-Betweenness wants to avoid. And since in the case of $k = 1$ the edge to be removed from the chain is selected randomly, as they have the same betweenness value, the performance is in no way worse due to this randomness.

4 Results

In this paper a variant of the *Newman-Girvan algorithm* was introduced which conveys two changes: The calculation of the edge-betweenness every k'th step, and the analysis of only a certain range r of clusterings to select the best. This brings a speed-up but in turn results in worse clusterings. To make the trade-off acceptable empirical data based on several hundred graphs was used to gather information on how to select k as well as r for four different indices. The results show there is a big difference on the performance of the algorithm depending on what quality indices are used, and also show some of the potential problems:

- Modularity: This was one of the good performers in the K-Betweenness algorithm: The range was successfully reduced by over 50% without any quality loss! Additionally it performed admirably well with a $k = n/10$ in over 97% of the tested graphs. This results in a speed-up factor of over $(n/5)$.

- Inter-cluster conductance: Here the range can easily be confined to the first two, i.e. only a calculation up to first separation of components needs to be run resulting in a very fast computation. The use of a $k \neq 1$ has a dramatic effect on the quality loss as a single bad cut (for example through the creation of singletons) is enough to make the whole clustering worthless.

- Coverage: As each component separation diminishes the quality (which starts at its peak) the range plays no significance. In case the initial clustering by components is forbidden we have a development for k very similar to modularity, although performing slightly worse with good values for k going up to $n/20$.

- Performance:The range here was confined to the first and last 10% of the time-steps. This, together with an acceptable value of $k = n/20$ (over 93% of the graphs had only minimal quality losses with this k) made performance a candidate for speed-up using the K-Betweenness algorithm.

In order to help correct some of these problems, such as the drop of inter-cluster conductance to 0 due to the creation of clusters with only inter-cluster edges, an addition to K-Betweenness was presented: Group singletons to nearby clusters.

It is important to note that while while most graphs performed very similar to one another the class of complete (and nearly-complete) graphs performed quite differently with mostly constant values at a constant value (dependent on the index, e.g. 0 in the case of modularity) up to the point of the first separation.

The results presented in this paper stem purely from empirical data gathered from a number of randomly generated graphs as well as some real world examples and as such can not be taken as a guarantee for the performance. However we provide several explanations for different observed phenomena and give deductions based on the gathered data. Since most of the problems related to finding optimal clusterings are NP-hard the use of heuristics and empirical evidence to find out more about the community structure is justified.

APPENDIX

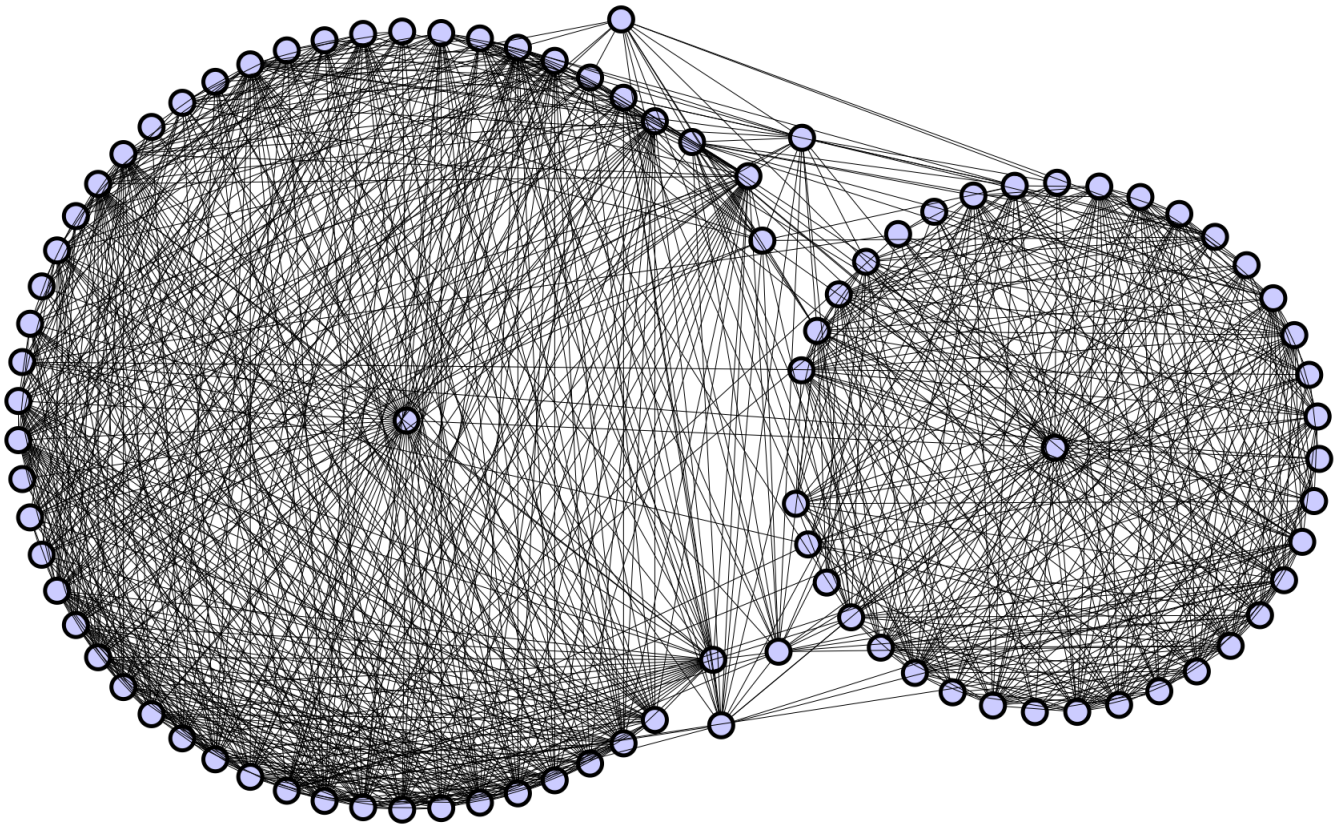


Figure 17: This is an attractor-generated graph with 93 nodes which results in a chaotic clustering if parsed using the k-Betweenness algorithm using modularity as the quality index.

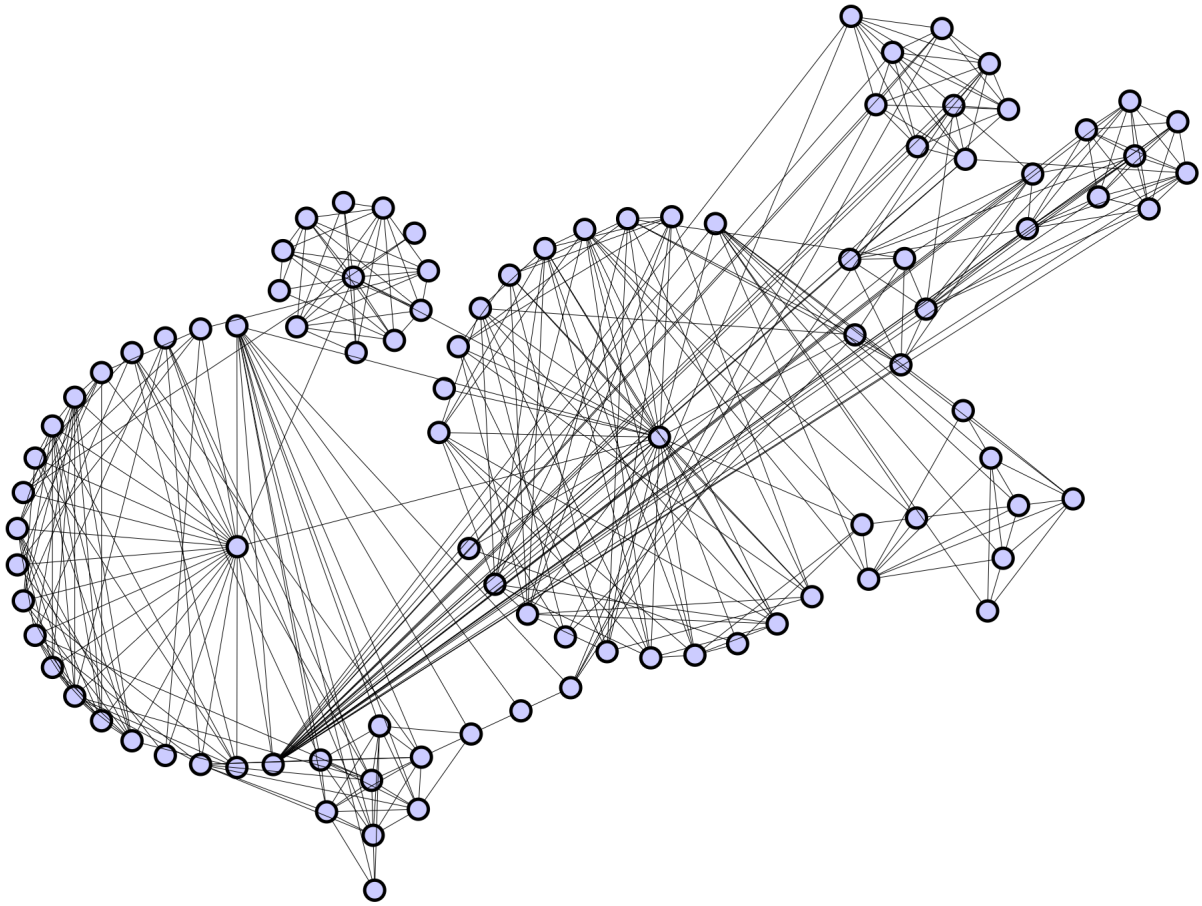


Figure 18: This is an attractor-generated graph with 98 nodes which returns a nice clustering when using the k-Betweenness algorithm using modularity as the quality index.

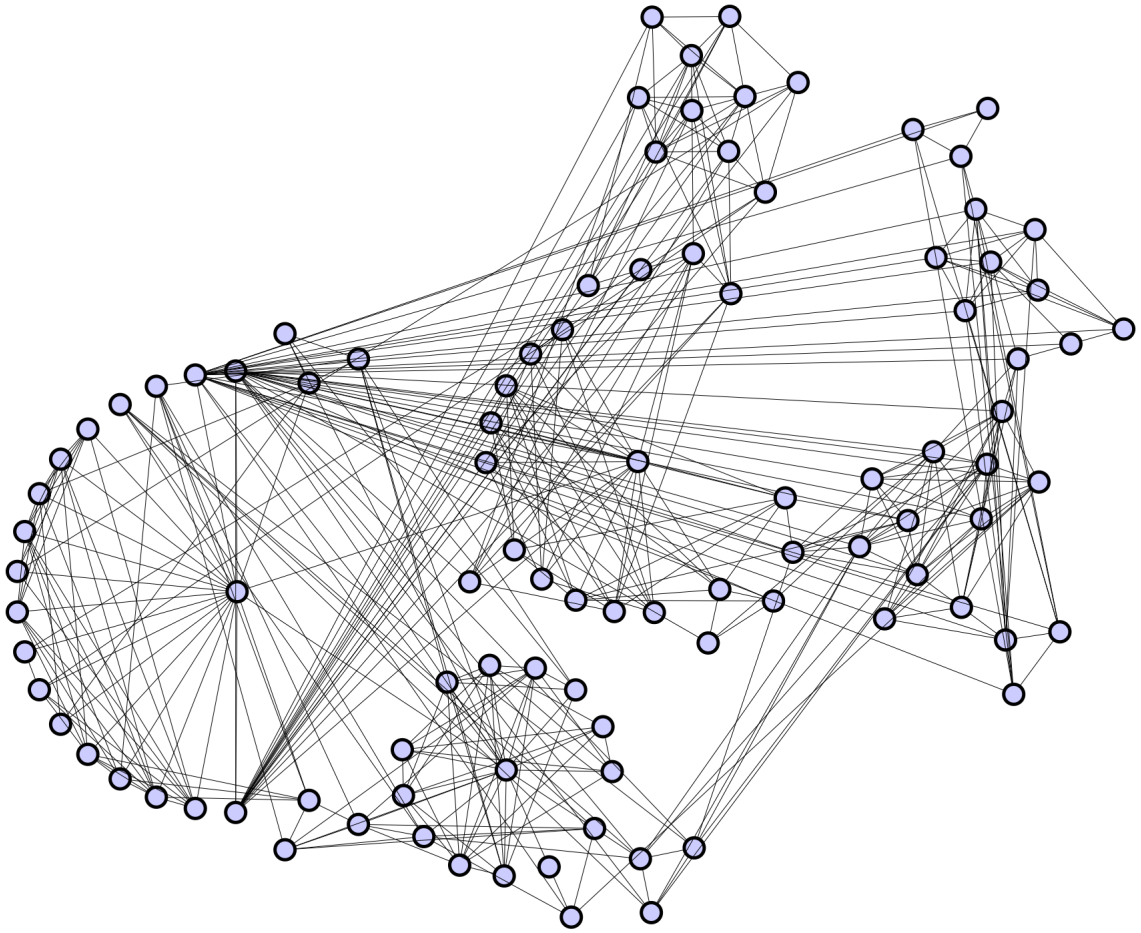


Figure 19: Another attractor-generated graph, this time with 100 nodes. Like many others this one shows a sudden drop of performance for a number of k's as can be seen in Figure 15

References

[1] M. E. J. Newman, M. Girvan: *Finding and evaluating community structures in networks*, 2003

[2] M. E. J. Newman: *Modularity and community structure in networks*, 2006

[3] D. Delling, M. Gaertler, R. Görke, Z. Nikoloski, D. Wagner: *How to Evaluate Clustering Techniques*, 2006: Technical Report 2006-24.

[4] F. Boutin, M. Haskoët: *Cluster Validity Indices for Graph Partitioning*, 2004: Conference on Information Visualization IV.

[5] M. E. J. Newman: *Modularity, Community Structure and Spectral Properties of Networks*

[6] U. Brandes, M. Gaertler, D. Wagner: *Experiments on Graph Clustering Algorithms*

[7] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, D. Wagner: *On Finding Graph Clusterings with Maximum Modularity*