Karlsruhe Institute of Technology

Finding Graph Clusterings with varying Coarseness

Study Thesis of

Philipp Glaser

At the Department of Informatics
Institute of Theoretical Informatics

Reviewer:	Prof. Dorothea Wagner
Advisors:	Dipl.-Inform. Andrea Kappes
	Dipl.-Inform./Dipl.-Math. Tanja Hartmann

Duration:: January 10, 2013 – June 26, 2013

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

Karlsruhe, June 26 2013

. . . .

(**Philipp Glaser**)

Acknowledgements

I would like to thank my advisors Tanja Hartmann and Andrea Kappes for their continuous support and suggestions, for being available for my inquiries even on short notice, for their patience and for the regular meetings. I would furthermore like Prof. Dorothea Wagner for giving me the chance to write this thesis for computer science at her institute and agreeing to review my thesis.

Zusammenfassung

In dieser Arbeit untersuchen wir die Möglichkeit in Graphen Clusterungen mit Clustern unterschiedlicher Größe schnell zu berechnen. Modularity [NG04] ist ein bekanntes und häufig verwendetes Clustermaß, das allerdings zu gewissen Clustergrößen tendiert [FB06]. Da auch Cluster mit anderen Größen interessant sind, verwenden wir die Modularity-Variante, die von Reichardt und Bornholdt [RB04] vorgeschlagen wurde.

In der Modularity-Variante von Reichardt und Bornholdt gibt es einen Parameter γ, der die Größe der gefundenen Cluster beeinflusst. In unserer Arbeit benutzen wir eine absteigende Folge von Werten für γ, um Clusterungen mit unterschiedlich großen Clustern zu finden. Eine absteigende Folge wurde gewählt, da mit dem von uns verwendeten Louvain-Algorithmus [BGLL08] Knoten zusammengefasst werden und kleinere γ zu größeren Clustern führen. Lambiotte [Lam10] bezeichnet Cluster als gut, wenn sie mit vielen verschiedenen Werten von γ gefunden werden.

Bei dem Louvain-Algorithmus [BGLL08] wird zunächst mittels lokaler Suche greedy für jeden Knoten das beste Cluster gesucht, bis für keinen einzelnen Knoten mehr ein besseres Cluster gefunden wird. Danach werden die entstandenen Cluster zu Knoten kontrahiert und auf dem entstandenen Graphen wird erneut mittels lokaler Suche nach einer guten Clusterung gesucht. Durch die Kontraktion ändert sich die lokale Umgebung im Lösungsraum, so dass mittels lokaler Suche nun andere Clusterungen gefunden werden können. Dies wird so lange fortgesetzt, bis keine Knoten mehr zu Clustern zusammengefasst werden.

Zur Verbesserung der Laufzeit haben wir verschiedene Heuristiken getestet, die jeweils verschiedene Zwischenergebnisse des vorherigen Laufs, der mit einem größeren γ stattfand, verwenden. So wird mit einem neuen γ nicht mit dem ursprünglichen Graphen und der Singleton Clusterung begonnen, sondern eine bereits gefundene Clusterung verwendet. Zum Teil werden auch Clusterungen aus bereits stark kontrahierten Graphen wieder auf die “entpackten” Graphen aus vorhergegangenen Kontraktionsstufen übertragen, da dort sehr viel feingranularere Änderungen in der Clusterung möglich sind.

Bei der Untersuchung dieser Heuristiken wurde festgestellt, dass auf großen Graphen die lokale Suche auf dem unkontrahierten Graphen bereits den größten Teil der Laufzeit kostet. Die Wiederverwendung der dort gefundenen Clusterung mit dem folgenden γ bringt bei großen Graphen und 1000 verschiedenen Werten für γ bereits eine Verbesserung der Laufzeit um einen Faktor bis zu 5. Dabei ist auch die durchschnittliche Qualität (gemessen anhand der Modularity-Variante von Reichardt und Bornholdt) im Mittel über alle Werte für γ meist besser.

Startet man den Louvain-Algorithmus mit dem neuen γ auf dem letzten kontrahierten Graphen lässt sich die Laufzeit bis zu einem Faktor 200 verbessern, aber dadurch wird die Qualität etwas schlechter. Durch eine Wiederverwendung eines Zwischenergebnisses, bei dem der Graph noch nicht so weit kontrahiert ist, ist die Verschlechterung gegenüber der

schnellsten Heuristik nur halb so hoch und die Laufzeit ist trotzdem noch bis zu 100 mal schneller als ohne die Verwendung eines vorherigen Zwischenergebnisses.

Die Anzahl der Cluster weicht zum Teil erheblich ab von der Anzahl, die mit einem reinen Louvain-Algorithmus mit verschiedenen Werten für γ gefunden wird. Allerdings sind die Clusterungen, die mit den Heuristiken gefunden werden, trotzdem nicht schlechter (gemessen in der Gamma-Modularity) als die mittels Louvain-Algorithmus gefundenen.

Contents

1. Introduction

1.1. Motivation

Many interesting problems can be represented and studied using graphs. Most of these graphs contain highly connected subgraphs which are called clusters and can be an interesting topic to study by itself, e.g. communities in social networks. If a social network contains a highly-connected community, information will pass quickly through this community and it will likely take longer to spread in the whole social network. Of course, the same is true for computer viruses in the Internet or real viruses in the real world.

Modularity [NG04] is a popular measure to assess a clustering, but it has been shown, that it suffers from a resolution limit (Fortunato, [FB06], i.e. it is biased towards a certain cluster size. For example if there are hierarchical clusters modularity probably detects only a supercluster, although the small clusters might be "better" or it detects the subclusters, when the superclusters appear more natural.

Reichardt and Bornholdt [RB04] proposed a variation of modularity, which introduces a parameter γ to changes this bias to find other cluster sizes. Still it tends to find big or small clusters and maybe not a "good" combination of different sized clusters.

Lambiotte [Lam10] proposed to classify clusterings as good when they are stable over a large range of γs. He showed that these stable clusterings are not necessarily found with the original modularity formula.

So to get a good clustering we should find the same clustering over a range of γs and since we are interested in finding these plateaus fast we propose a way to quickly find clusterings with different gammas in a single run, by passing a set of gammas as parameter.

1.2. Terminology

A *graph* G consists of a set of *vertices* V and a set of *edges* E, which are unordered pairs of vertices out of the vertex set V. Sometimes a graph is also called network and the vertices are often called nodes. Two vertices are *connected* or *adjacent* if an edge contains both vertices. The two vertices of an edge are also called its *endvertices*. The vertices that are connected to a certain vertex are called its *neighbors*. A *self-loop* is an edge that connects a vertex with itself, i.e. an edge that contains the same vertex twice. An *incident* edge of a vertex is an edge that contains the vertex. A *subgraph* consists of a subset S of V and a subset of E, that only contains edges which have both endvertices in S.

The *degree* of a vertex is the number of incident edges (self-loop counted twice) and the *weight* of an edge is a value associated with an edge. The *strength* of a vertex is the sum of the edge weights of the incident edges (self-loops counted twice). A *contraction of an edge* is the process of replacing an edge and its associated vertices with a new vertex. All incident edges of the former vertices are attached to the new vertex and the old edge is added to the new vertex as self-loop. We can *contract a connected subgraph* by contracting all edges contained in the subgraph.

A *clique* is a subgraph where each vertex is connected to each other vertex in the subgraph.

A *cluster* is a subgraph that has many edges inside and few edges that are connected to vertices outside this subgraph. A *clustering* is a partition of a graph into clusters. A *singleton clustering* is a clustering where each cluster contains a single vertex.

1.3. Related Work

Besides the modularity variation of Reichardt and Bornholdt [RB04] there are also alternative ways to handle the resolution limit of modularity.

Arenas et al. [AFG08] proposed to modify the graph by adding a self-loop to each vertex and thereby decreasing the expected cluster sizes, that are found by modularity.

Lambiotte [Lam10] proposed a way of finding clusters by random walks in a graph. He defines stability as the probability for a random walker to be in the same cluster at the beginning and after time t minus the probability of two random walkers to be in the same cluster at the end. To find clusters of different sizes he modified the time the random walkers move in the graph. He also showed that this definition is equivalent up to linear transformations to the metric proposed by Reichardt and Bornholdt and the metric proposed by Arenas et al.

2. Modularity as clustering metric

Modularity is a popular measure to assess the quality of a clustering of a graph. It was introduced by Newman and Girvan in [NG04].

With an ordered set of vertices and with c_i as cluster of the vertex i and δ as Kronecker delta, which is defined as 1 if its two variables are the same and 0 otherwise, we can calculate the modularity by looking at all possible pairs of vertices:

$$Q = \sum_{i \in V} \sum_{j \geq i, j \in V} \left(\frac{e_{ij}}{m} - \frac{k_i * k_j}{2m^2} \right) \delta(c_i, c_j) \tag{2.1}$$

The first term implies large clusters (connected components). Without the second term it would put all connected components into a single cluster. The second term makes for balanced clusters.

The problem with modularity is that it is biased towards a certain cluster size as was found out by Fortunato and Barthélemy in [FB06]. They proved that some “natural” clusterings can not be found with the original modularity formula (2.1). They evaluated a graph that consisted of small cliques which are connected with a single edge to two neighbored cliques in a ring like fashion. When the cliques are small enough modularity increases if two neighbored are put into the same cluster, although two cliques which are merely connected by a single edge do not form not a very “natural” cluster.

To work around this problem Reichardt and Bornholdt proposed a modification in [RB04] henceforth called *gamma modularity*, which introduces a weight γ to the second term in the modularity formula. They also negated the whole formula, but we ignore this negation in our work:

$$Q_\gamma = \sum_{i \in V} \sum_{j \geq i, j \in V} \left(\frac{e_{ij}}{m} - \gamma \frac{k_i * k_j}{2m^2} \right) \delta(c_i, c_j) \tag{2.2}$$

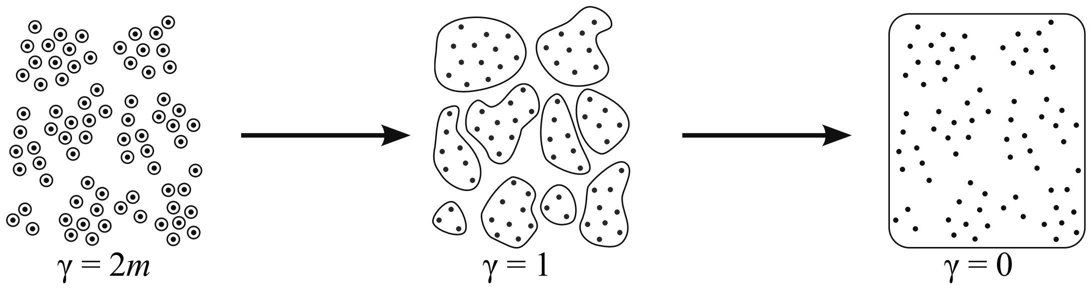


Figure 2.1.: Effect of decreasing γ on the cluster size

With this formula the target size of the clusters can be adjusted with the γ parameter. With increasing cluster sizes both terms get larger. If we decrease γ the right term becomes less important and therefore we get larger clusters as shown in Figure 2.1. Another effect of decreasing γ is that the modularity increases even if the clustering stays the same, because the right term becomes smaller.

Lambiotte [Lam10] classified a clustering as good if the clustering does not change over a large range of γ values[1]. He researched a randomly generated hierarchical graph as well as the football graph. In both cases he found clusterings, which are not found by the unmodified modularity, but are better according to this metric.

We now propose a way to quickly find clusters with different expected target sizes. To get a singleton clustering we set $\gamma = 2m$ and to get connected components we set $\gamma = 0$. Although the target size can now be given it can still not find “natural” clusters with big differences in their size, but we can assume that found clusters that are the same for a large range of γ values are “good”. If the whole clustering does not change within a large range of γ values we probably have found a good clustering.

[1]He used a different metric,that is equivalent to the gamma modularity up to a linear transformation

3. Finding good clusterings fast

In this chapter we shortly describe the Louvain method [BGLL08] to find a clustering and then split it up into small steps. Using these steps with the gamma modularity we can get clusterings with varying coarseness in a single run by reusing intermediate results from a previous γ. Afterwards we introduce new steps and then show how these steps can be combined in different ways to get several heuristics with varying properties, which are evaluated in Chapter 4.

3.1. The Louvain Method

The Louvain method that was used as base for our algorithm works as follows. All steps are explained in great detail in Section 3.2. We start with a singleton clustering, i.e. at first each cluster contains just a single vertex.

In the next step which is illustrated in Figure 3.1 we visit each vertex in a random order and find the best cluster for each vertex by a local search. The search is explained in Section 3.2. This is done for all vertices in the graph and is called *iteration*. We repeat this step until we visited the whole graph once without reassigning any vertex. After repeating this step we get a new clustering for the given graph which has a better modularity and cannot be improved by moving any single vertex. Iterating over the graph multiple times until we reach the local maximum is called an *iteration phase*.

Afterwards we create a new graph by contracting the clusters as also shown in Figure 3.2. After this step we end up with a new graph with a singleton clustering. We also keep a map from the clusters of the old graph to the vertices of the new graph so we can find the cluster of the old vertex on the new graph. A graph together with a clustering (and the map to the above vertices, if there is one) is called *level*. Each cluster on each level has a corresponding vertex on the level above.

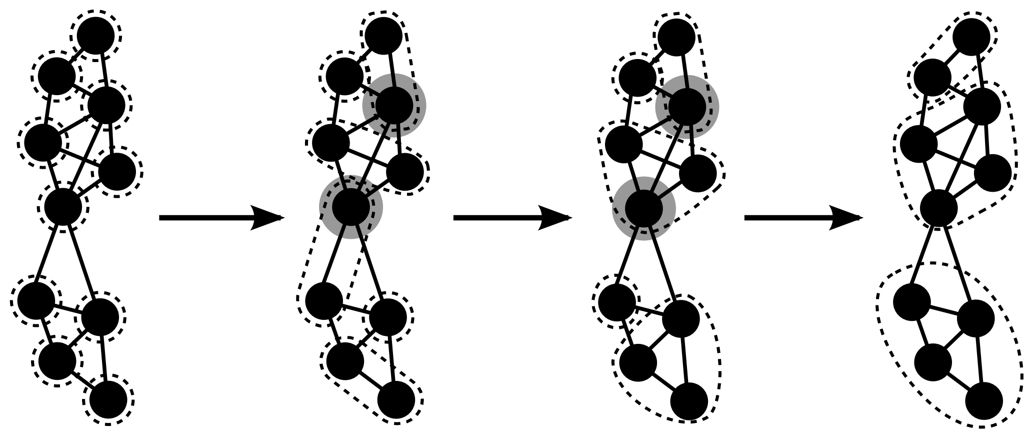


Figure 3.1.: Iteration Phase: After the first iteration the marked vertex ended up in the "wrong" cluster, After several iterations we might end up with the clustering in the right and cannot find any single vertex move to improve the clustering. The intermediate as well as the final result depends on the order in which the vertices are processed.

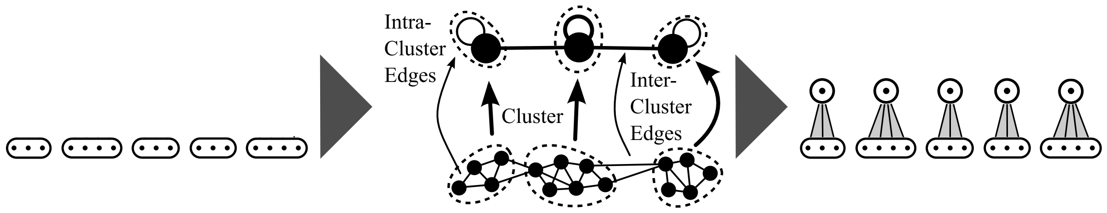


Figure 3.2.: Going up: Each cluster is contracted to a vertex. Intra-cluster edges become self-loops and all edges get the sum of the corresponding edges weights below assigned

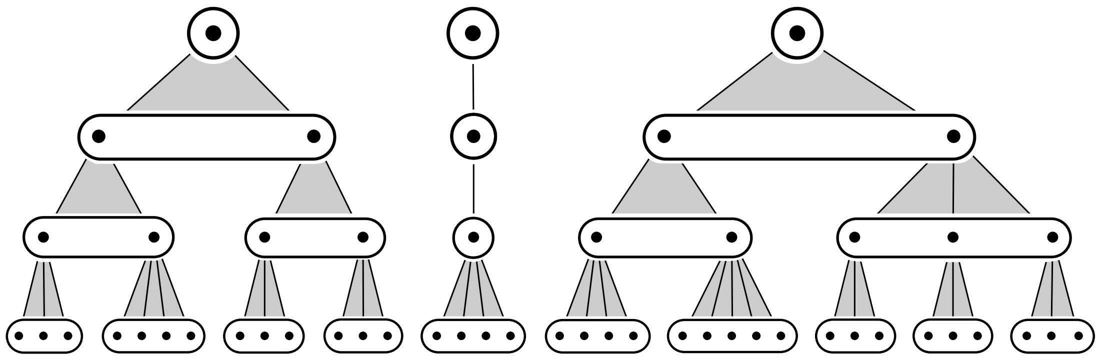


Figure 3.3.: Cluster hierarchy: After repeating the steps Iteration Phase and Going Up we get a cluster hierarchy. To get the final cluster of each vertex we follow the maps up until we reach the singleton clustering.

The step which creates the new contracted graph with a singleton clustering out of the given graph and clustering and populates the map which connects these levels is called *Going Up*.

Now we repeat the last two steps (Iteration phase and going up) with the newly created graph and singleton clustering to get another level above. We repeat this until the modularity is not increased any more and therefore no more vertices can be contracted. If iterating on the highest level does not move a vertex to a different cluster the singleton clustering is kept and the algorithm ends. At the end we get a cluster hierarchy as shown in Figure 3.3.

To extract the resulting clustering we follow the maps from the lowest level to the highest, which always has a singleton clustering, because otherwise we could go up once more.

All these steps can also be used with the gamma modularity from [RB04], which is done in the next section.

3.2. Breaking down the Louvain method

In this section we define names for the steps used in the Louvain method, so we can combine them later with other steps to quickly get clusterings for multiple gammas. We also give the strings that were used in our tool to combine these steps in different configurations.

Iteration Phase

An iteration phase always starts with a graph and a clustering. To find the best cluster for the current vertex during the iteration phase we only need to look at the clusters of the neighbors of the vertex, because if the current vertex does not have an edge into a cluster, it cannot improve the gamma modularity to move this vertex into that cluster.

We can derive an easy formula to calculate the gamma modularity ([RB04]) gain when moving a vertex. To do that we also need the following definitions:

e_{ij} Weight of the edge from i to j. 0 if there is no edge.

k_i Strength of vertex i

K_c Strength of cluster c

o, n The old resp. new cluster

O, N The old resp. new clustering

o', n' The old resp. new cluster without the current vertex

c_i^O The cluster of vertex i in the clustering O

ΔE The difference of intra-cluster edges between the old and the new clustering

ΔK The difference of strength between the new cluster and the old cluster without the current vertex

Using Equation 2.2 we subtract the modularity of the old clustering from the modularity of the new clustering:

$$\begin{aligned}
\Delta Q_\gamma =& \left(\sum_{i \in V} \sum_{j \geq i, j \in V} \left(\frac{e_{ij}}{m} - \gamma \frac{k_i k_j}{4m^2} \right) \delta(c_i^N, c_j^N) \right) - \left(\sum_{i \in V} \sum_{j \geq i, j \in V} \left(\frac{e_{ij}}{m} - \gamma \frac{k_i k_j}{4m^2} \right) \delta(c_i^O, c_j^O) \right) \\
=& \left(\sum_{i \in V} \sum_{j \geq i, j \in V} \frac{e_{ij}}{m} \delta(c_i^N, c_j^N) - \frac{e_{ij}}{m} \delta(c_i^O, c_j^O) \right) - \\
& \left(\sum_{i \in V} \sum_{j \geq i, j \in V} \gamma \frac{k_i k_j}{4m^2} \delta(c_i^N, c_j^N) - \gamma \frac{k_i k_j}{4m^2} \delta(c_i^O, c_j^O) \right) \\
& \text{Without the unaffected Clusters we get:} \\
\Delta Q_\gamma =& \frac{\Delta E}{m} - \gamma \frac{1}{4m^2} \left(\left((K_{n'} + k)^2 + K_{o'}^2 \right) - \left(K_{n'}^2 + (k + K_{o'})^2 \right) \right) \\
=& \frac{\Delta E}{m} - \gamma \frac{1}{4m^2} \left(\left(\left(K_{n'}^2 + 2K_{n'}k + k^2 \right) + K_{o'}^2 \right) - \left(K_{n'}^2 + \left(k^2 + 2K_{o'}k + K_{o'}^2 \right) \right) \right) \\
=& \frac{\Delta E}{m} - \gamma \frac{1}{4m^2} \left(\left(2K_{n'}k + k^2 \right) - \left(k^2 + 2K_{o'}k \right) \right) \\
=& \frac{\Delta E}{m} - \gamma \frac{k \Delta K}{4m^2}
\end{aligned} \tag{3.1}$$

With this formula the modularity gain or loss of moving a single vertex can be calculated in $O(1)$, if the strengths of the clusters and vertices are known. An iteration phase ends up in a local maximum, i.e. no single vertex movement to another cluster leads to an improved gamma modularity. To start an iteration phase we use “it” in our evaluation tool. Iterating once over the graph takes $O(m)$ time. The number of iterations in each phase is unknown, but we observed that on a larger graph it also takes more iterations until a local maximum is found in a phase.

Going up

To specify when to go up as shown in Figure 3.2 we use “up”. Going up takes $O(m)$ time with the m from the old, lower graph. So the old graph is called the lower level and the newly created graph is called the upper level (or top level). Going up does not change the found clustering, so the gamma modularity stays the same.

Loops

In our tool the Louvain method can the described as loop that just contains “it” and “up” after first initializing the graph with the singleton clustering. The syntax for loop consists of parenthesis and the loop ends if there is no gamma modularity gain, so the Louvain method is just “(it up)”.

3.3. New Steps

Because we are interested in clusterings with varying coarseness we use a sequence of γ values. Some of the followings steps are necessary when dealing with multiple gammas and others are useful to get better or faster results. With these steps we can reused intermediate results, which allows us to save a varying amount of running time.

Decrease Gamma

As we are interested in clusterings with different gammas we need to select the next gamma, which is done with “dec”. The first gamma is the largest, so we start with many small clusters and end up with few large clusters. The reason to start with small clusters is that in the Louvain method clusters are merged but not divided. In the Louvain method there is no way to decrease the cluster size other than going back to a lower level, where the clusters are not yet merged.

The gamma modularity also increases with each new gamma unless we change the clustering. This gain can be easily calculated using the strength K of each cluster c and the difference between the old and the new gamma $\Delta\gamma$:

$$\Delta Q_\gamma = \Delta\gamma \sum_{i \in C} \frac{K^2}{4m^2} \tag{3.2}$$

Selecting a non-existing gamma terminates the program.

Reset

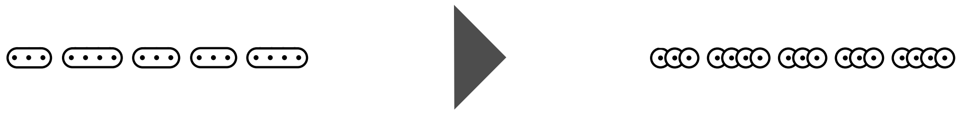


Figure 3.4.: Reset: After going to a lower level we can use a singleton clustering instead of the clustering there from a previous iteration phase.

The step of assigning each vertex its own cluster, that is implicitly used in the Louvain method at the very beginning, is called *resetting* and is shown in Figure 3.4. To specify when to reset the clustering we use "re" in the string which defines our heuristic. After a reset of the clustering the gamma modularity needs to be recalculated. The reset itself takes $O(n)$ time. The modularity could be calculated in $O(n)$ time, because each cluster contains at most one self-loop, which could be saved for each vertex, and we only need to subtract the γ weighted sum of the squares of all vertex strengths (which are save for each vertex). Our tool does a full recalculation of the modularity, which takes $O(m)$ time, because the reset step is still very fast compared to the other steps and then we do not need to save the self-loops, because they are not required for anything else.

Going down

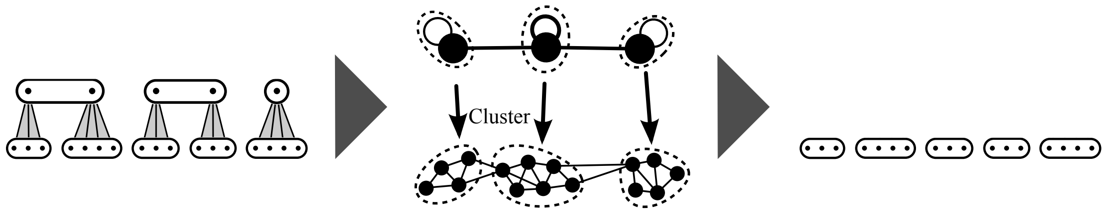


Figure 3.5.: Go down: With this step we just go back to a lower level our cluster hierarchy.

If we go down without using the clustering we found on the higher level, we call this step *going down*. Going down is shown in Figure 3.5 and generally changes the found clustering, because on lower levels the clusters are not contracted and therefore smaller. This leads to a degradation unless the clustering did not change after the last going up. Unlike going up which can only be done for one level, going down can go to any level below.

In the input language of our tool going down is split up into "fr", which is going down by a fraction and takes denominator as parameter, and "do", which goes down by one level, if there is a lower level. To go down to the lowest level we use "fr:1".

```
Data: Graph $G$ with singleton clustering
Data: Set $\Gamma = \gamma_1 > \cdots > \gamma_r$
for $\gamma_1, \cdots, \gamma_r$ do
    Calculate $Q_\gamma$;
    repeat
        Iterate over $G$ with $\gamma_1$;
        Go up;
    until Gamma modularity does not increase anymore;
    Put out clustering;
    Go down to the lowest level;
    Reset to singleton clustering;
end
```

Algorithm 1: Using the Louvain Method with multiple gammas or short "((it up) dec fr:1 re)".

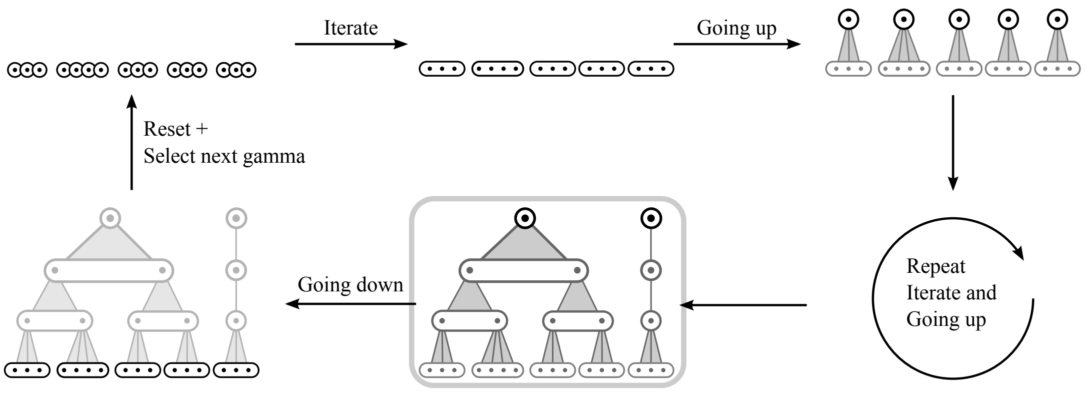


Figure 3.6.: The Cluster hierarchy during the Louvain method. The highlighted hierarchy in the middle of the bottom row, represents the best clustering found for this γ. Of course, the hierarchy changes slightly with each new γ.

As an example how we specify the heuristic, if we want to evaluate the Louvain method repeated once for each γ in our set. This can be done by specifying "((it up) dec fr:1 re)" as parameter in our tool and results in Algorithm 1. A graphic representation of this algorithm is shown in Figure 3.6

Projection

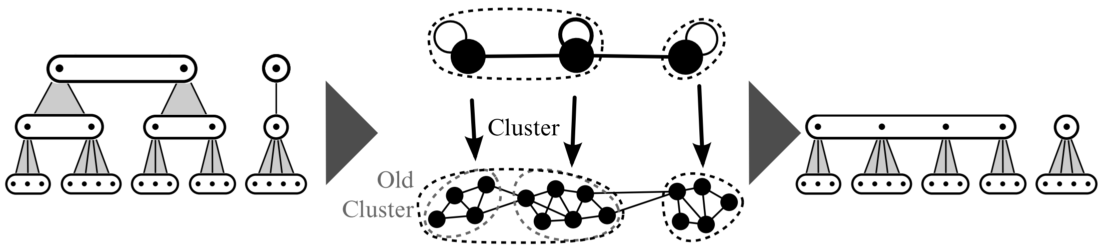


Figure 3.7.: Projection: We can use the clustering we found on a level above the current level.

We can also take advantage of the merged clusters on a higher level and use the higher clustering on a lower level. We call this *projecting* the clustering and is illustrated in Figure 3.7. The projection keeps the gamma modularity from the level above, but allows us to move the "small" vertices from the low level to new clusters.

In our tool Projection also takes the denominator of the fraction of the levels we want to project to and is called "pr". "pr:0" is treated separately and projects down a single level.

So to project the clustering from the highest level to the middle level we use "pr:2".

3.4. Details

In this section we describe implementation details and additional features of the implementation that are informative, but not necessary to understand the heuristics. At first we explain how rounding is done in steps that involve multiple levels in the cluster hierarchy and later we describe additional features, that are implemented, but were not evaluated in a systematic manner, because first tests were not very promising with regard to the resulting quality.

Rounding

The fraction given by "fr" is always rounded to the higher level. So if there are 4 levels and we want to go to the middle level with "fr:2" we end up on the second highest level.

To understand how rounding in the projection step works it is necessary to know that in our implementation the projection step is in fact done as a separate step after going down. The higher levels, which have a valid clustering, are only thrown away if there was a projection, a reset or a successful iteration. So the parameter to the projection function is the fraction between the current level and the highest level in our hierarchy. This fraction is rounded down. So if the highest level is 3 and we are on level 0, “pr:2” projects the clustering of level 1 onto level 0.

To project the middle level to the lowest level we first go down to the lowest level with “fr:1” and then use a projection from the middle level with “pr:2”. If we have 4 Levels we would get the clustering of the second lowest level on the lowest level.

Additional features of the implementation

In the iteration phase a parameter can be given to limit the number of iterations. Especially on the lowest level this can be used to shorten the running time at the expense of quality. To do this we put “it:n up” in front of our string. If we do this for the lowest level(s), this results in a speed up at the cost of quality. It is interesting to note that we observed, that it also results in higher cluster hierarchies if we use this for multiple levels.

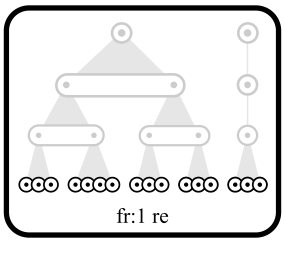
The loops can also be limited by giving a parameter in front of the closing parenthesis. So if we want to iterate and go up at most twice and then project down one level we could do so by specifying “(it up 2) pr:0”. These two features were both not used in our systematic evaluation in the next section.

4. Evaluation

We tested several heuristics using the aforementioned steps in different configurations to find combinations with a good quality/time trade-off. All heuristics are compared to using the Louvain method for community detection once for each γ. The Louvain method is also called static algorithm.

All the dynamic heuristics we tested result in quite similar modularity values, but with much shorter run times. The tests on the eu-2005-graph, which is used as reference for most of the Evaluation section, ran on an Intel® Xeon® CPU E5-2670 0 @ 2.60GHz machine with 64GB RAM. The other graphs were evaluated on an Intel® Core™2 Duo CPU E7300 @ 2.66GHz with 4 GB RAM.

In all heuristics we used a set of 1000 γ values starting with 100 and decreasing by 0.1 in each step. The heuristics are shortly described in the text and all have an icon which symbolizes the start of the outer loop[1]. The icon for repeating the Louvain method, where each outer loop starts with the singleton clustering on the lowest level, is given as example on the right.



In the next section we shortly describe the graphs used for the evaluation. In the following sections we describe the evaluated heuristics starting with the ones close to repeating the Louvain method with different gammas as in Algorithm 1. Later we describe the heuristics that use more information from the previous runs. At the end we give our conclusions about heuristics with a good quality/time trade-off.

Graph	Vertices	Edges
eu-2005	862664	16138468
cnr-2000	325557	2738969
smallworld	100000	499998
cond-mat-2005	40421	175691
jazz	198	2742
football	115	613
adjnoun	112	425
dolphins	62	159

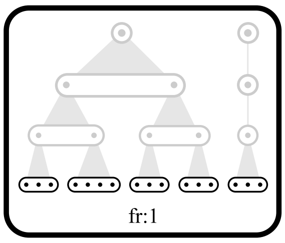
Table 4.1.: List of Graphs used in the evaluation

4.1. Graphs used for the evaluation

The number of vertices and number of edges of the graphs used for the evaluation of the heuristics are shown in Table 4.1. All graphs besides the smallworld-graph are real world graphs and can be downloaded from the Homepage of the 10th DIMACS Implementation Challenge in the downloads section[2]. The cond-mat-2005 graph was the only graph with multiple connected components.

The eu-2005 and cnr-2000 graph are web crawls. The dolphin graph is a social network of associations in a dolphin community. The jazz and cond-mat-2005 graph are based on the collaboration of jazz musicians respectively physicists (in the area condensed matter). The adjnoun graph resembles adjacencies between common nouns and adjectives in Charles Dicken's David Copperfield. The football graph is based on matches in an American college league. All the aforementioned graphs have been analyzed with regard to their possible clusterings before. The smallworld graph has been generated using the small world generator of the Boost Graph Library.

4.2. Going down to the lowest level



The heuristics evaluated in this section all have in common that they go back down to the lowest level, i.e. to the graph where no clusters have been contracted. The accumulated running time, speed-up and average modularity difference to the static algorithm is shown in Table 4.2.

The first evaluated heuristic just keeps the previously found clustering, i.e. just goes down to level 0 ("fr:1") after decreasing the

[1]To keep the icons as simple as possible the middle level in the icons is the second lowest level, although in our implementation it would be the second highest.
[2] `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`

Instructions	Time (s)	Speedup	ΔQ_γ
eu-2005			
fr:1_re	149766.9	1.0	0.0000
fr:1	35123.2	4.3	0.0005
fr:2_pr:1	14531.9	10.3	0.0029
pr:1	14751.1	10.2	0.0028
fr:2_re	1527.8	98.0	−0.0021
fr:2	1436.6	104.3	−0.0017
fr:4_pr:3	1186.7	126.2	−0.0014
pr:2	1139.4	131.4	−0.0017
do_re	823.0	182.0	−0.0032
do	743.9	201.3	−0.0034
...	664.7	225.3	−0.0034
smallworld			
fr:1_re	2525.8	1.0	0.0000
fr:1	1370.1	1.8	0.0000
fr:2_pr:1	674.9	3.7	−0.0003
pr:1	676.2	3.7	−0.0002
fr:2_re	619.8	4.1	−0.0001
fr:2	516.5	4.9	−0.0001
fr:4_pr:3	370.9	6.8	0.0001
pr:2	387.7	6.5	0.0000
do_re	299.3	8.4	−0.0004
do	272.0	9.3	−0.0003
...	229.9	11.0	−0.0004
jazz			
fr:1_re	5.6	1.0	0.0000
fr:1	3.5	1.6	0.0002
fr:2_pr:1	3.4	1.6	0.0003
pr:1	3.5	1.6	0.0003
fr:2_re	4.3	1.3	−0.0002
fr:2	3.3	1.7	−0.0002
fr:4_pr:3	3.3	1.7	−0.0002
pr:2	3.3	1.7	−0.0002
do_re	4.7	1.2	−0.0010
do	3.3	1.7	−0.0007
...	1.2	4.7	−0.0014
adj-noun			
fr:1_re	0.9	1.0	0.0000
fr:1	0.6	1.4	0.0001
fr:2_pr:1	0.6	1.4	0.0002
pr:1	0.6	1.4	0.0002
fr:2_re	0.7	1.2	−0.0003
fr:2	0.5	1.6	−0.0002
fr:4_pr:3	0.5	1.6	−0.0002
pr:2	0.6	1.5	−0.0002
do_re	0.8	1.1	−0.0006
do	0.6	1.5	−0.0004
...	0.2	4.0	−0.0008

Instructions	Time (s)	Speedup	ΔQ_γ
cnr			
fr:1_re	36591.3	1.0	0.0000
fr:1	6814.0	5.4	−0.0003
fr:2_pr:1	4010.5	9.1	0.0019
pr:1	4194.0	8.7	0.0019
fr:2_re	643.7	56.8	−0.0037
fr:2	528.0	69.3	−0.0036
fr:4_pr:3	413.1	88.6	−0.0035
pr:2	396.5	92.3	−0.0034
do_re	318.9	114.7	−0.0046
do	268.4	136.3	−0.0047
...	243.9	150.0	−0.0048
cond-mat-2005			
fr:1_re	922.3	1.0	0.0000
fr:1	443.9	2.1	0.0004
fr:2_pr:1	231.6	4.0	0.0019
pr:1	232.8	4.0	0.0018
fr:2_re	130.0	7.1	−0.0031
fr:2	102.0	9.0	−0.0030
fr:4_pr:3	73.0	12.6	−0.0028
pr:2	73.5	12.5	−0.0027
do_re	66.3	13.9	−0.0043
do	55.7	16.5	−0.0039
...	40.1	23.0	−0.0049
football			
fr:1_re	0.4	1.0	0.0000
fr:1	0.3	1.3	−0.0001
fr:2_pr:1	0.3	1.2	−0.0001
pr:1	0.3	1.2	−0.0001
fr:2_re	0.3	1.1	−0.0002
fr:2	0.3	1.3	−0.0003
fr:4_pr:3	0.3	1.2	−0.0002
pr:2	0.3	1.2	−0.0003
do_re	0.3	1.1	−0.0001
do	0.3	1.3	−0.0003
...	0.2	1.6	−0.0015
dolphins			
fr:1_re	0.3	1.0	0.0000
fr:1	0.2	1.4	−0.0000
fr:2_pr:1	0.2	1.3	0.0002
pr:1	0.2	1.3	0.0001
fr:2_re	0.3	1.2	−0.0003
fr:2	0.2	1.6	−0.0000
fr:4_pr:3	0.2	1.5	0.0000
pr:2	0.2	1.5	−0.0001
do_re	0.3	1.1	−0.0005
do	0.2	1.4	−0.0002
...	0.1	3.7	−0.0005

Table 4.2.: Accumulated running time over all gammas, speedup compared to the static algorithm and average Modularity difference of the evaluated heuristics. The "(it up) dec" is omitted as well as the outer loop. "..." means just continue with the cluster hierarchy from the previous γ.

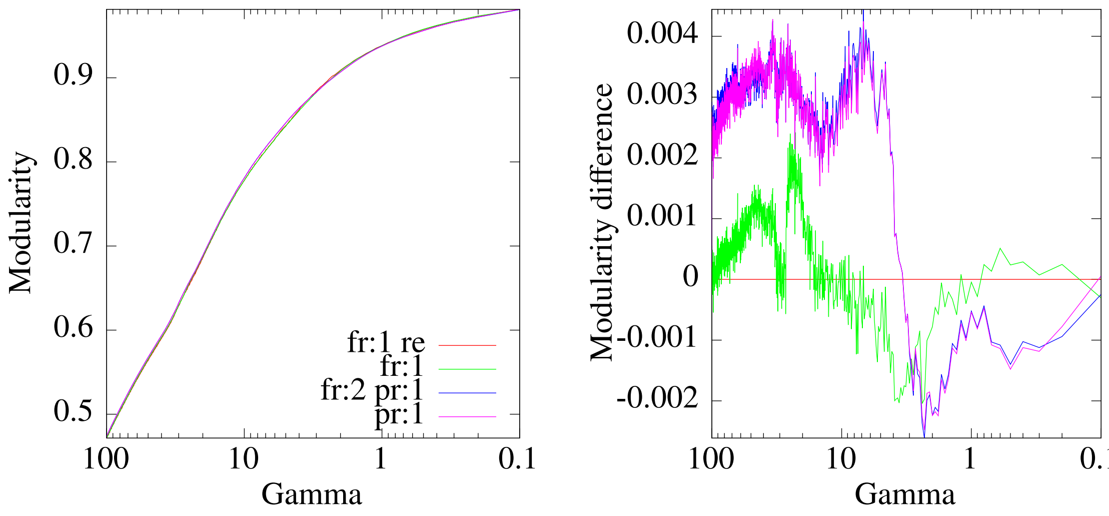
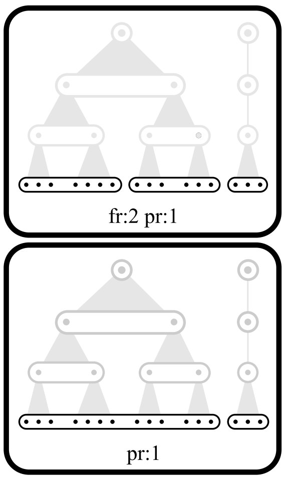


Figure 4.1.: Heuristics which go down to the lowest level for the eu-2005-Graph. On the left hand you can see the modularity values of the different heuristics and on the right hand you can see the difference to the static algorithm. The “(it up) dec” is omitted as well as the outer loop.

gamma (“dec”) like in the repeated Louvain method (Algorithm 1), but does not reset (“re”) the clustering to the singleton clustering. As shown in Figure 4.1 the gamma modularity does not differ much from the gamma modularity found with the repeated Louvain method. With the first gammas it even results in better clusterings. Although with the medium gammas it becomes worse than the static algorithm, on average it is still better. It is also faster than repeating the Louvain method. The speedup can be explained by the decreased number of iterations required on the bottom level until a local maximum is found.



In the second heuristic we project the clustering from the middle level (“fr:2 pr:1”) to the lowest level. Now we can see a better gamma modularity for the first gammas, but also a fast decline between gamma 4 and 2. Although the gamma modularity later gets closer to the one found with the repeated Louvain method, it does not get better than it. On average the quality is better than the Louvain method and it is faster by one order of magnitude. Now in addition to the decreased number of iterations on the first level there are less levels because the clusters found on the lowest level become larger.

The better quality of this heuristic compared to the repeated Louvain can maybe be explained by repeated tries to find a better clustering and keeping the good results.

The last heuristic which goes down to the lowest level projects the clustering from the highest level. As you can seen in Figure 4.1 the results are extremely similar to the projection from the middle level.

4.3. Going down to the middle level

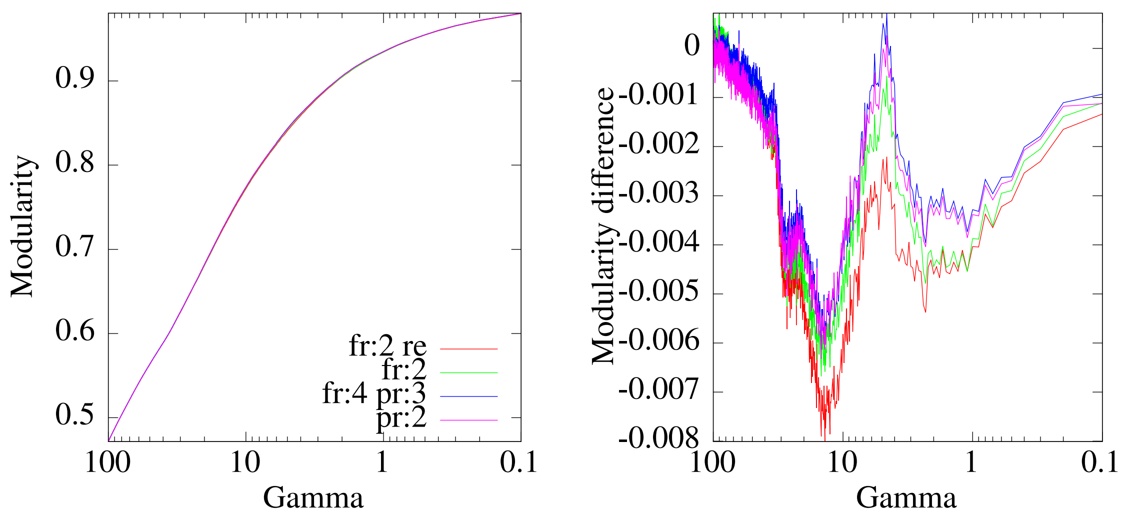
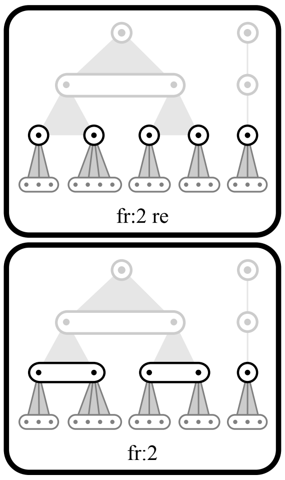


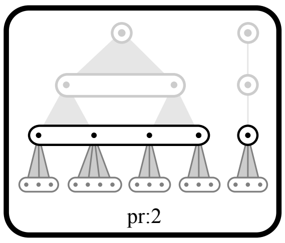
Figure 4.2.: Heuristics which go down to the middle level for the eu-2005-Graph. On the left hand you can see the modularity values of the different heuristics and on the right hand you can see the difference to the static algorithm. The "(it up) dec" is omitted as well as the outer loop.

In this section we take a look at the heuristics which go down to the middle level. As you can see in Figure 4.2 the modularity values of the heuristics in this category are quite close to each other.



The first heuristic goes down to the middle level and resets the clustering to a singleton clustering. As shown in Figure 4.2 the modularity of the found clustering is a little bit worse than the one found with the Louvain method, but at most 0.008 lower (with a γ between 12 and 10). Until a γ of 5 the gamma modularity difference shrinks down to 0.003, but than grows again up to 0.005 until it shrinks again. Among the heuristics in this section this is the slowest for the same reasons the static algorithm was the slowest in the previous section: It likely requires the most iterations on the middle level and results in larger graphs on the levels above.

Without the reset we use the previously found clustering and the modularity values are a tiny bit better than with the reset, and we get the results a bit faster. The modularity values are a bit higher in general but only by about 0.001.



With a projection from higher levels (top level respectively half way to the top) the values are even better but the differences to the previous heuristic are very small on the whole set of gammas. The arguments from the previous section about the reasons for the different running times are still valid.

Compared to the heuristics which go down to the lowest level all heuristics in this section are much faster, but provide a bit worse results.

4.4. Go down one level

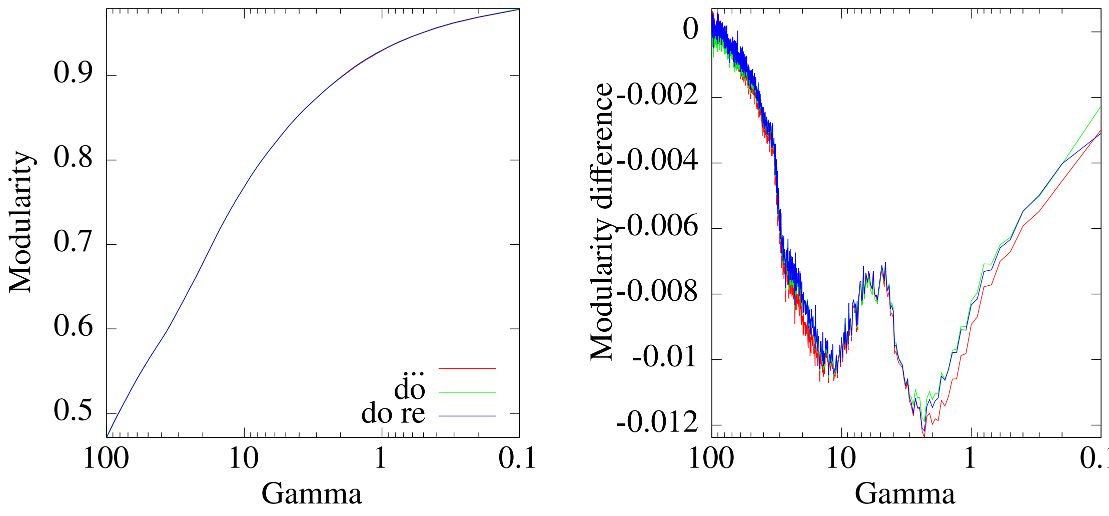
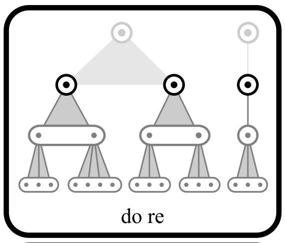
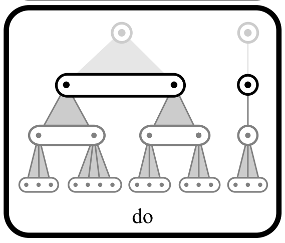


Figure 4.3.: Heuristics which go down one level or just continue going up for the eu-2005-Graph. On the left hand you can see the modularity values of the different heuristics and on the right hand you can see the difference to the static algorithm. The “(it up) dec” is omitted as well as the outer loop. “. . . ” is just going up.



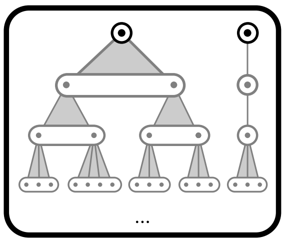


In this section we take a look at the heuristics which go down by one level. Among the heuristics that go down just one level the one that resets to the singleton clustering often resulted in the best quality (for the larger graphs) as shown in Figure 4.3. For the eu graph the resulting modularity values are all very close together and the difference to the Louvain method generally first grows up to gamma 10 and a modularity difference of 0.01. Then there is a small improvement up to a difference of about 0.008 around gamma 7 before it gets worse again with a difference of 0.012 between gamma 3 and 2. At very low gamma values the difference becomes smaller again.

Altogether the quality of the heuristics differ a lot between the graphs so it is difficult to give a recommendation in this category.

Compared to the heuristics from the previous categories the heuristics in this section are faster, but on the other hand the quality is also worse.

4.5. Do not go down



Without going down the developing of the modularity values is largely the same as with the heuristics which go down by one level, as shown in Figure 4.3.

There is a minor speedup compared to going down one level, but with some other graphs there was also a minor quality loss which is shown in Table 4.2. With larger graphs the differences compared to going down one level become smaller both in the speedup and in the quality, so it can still be an useful heuristic.

This is the fastest algorithm of all categories.

4.6. Conclusions

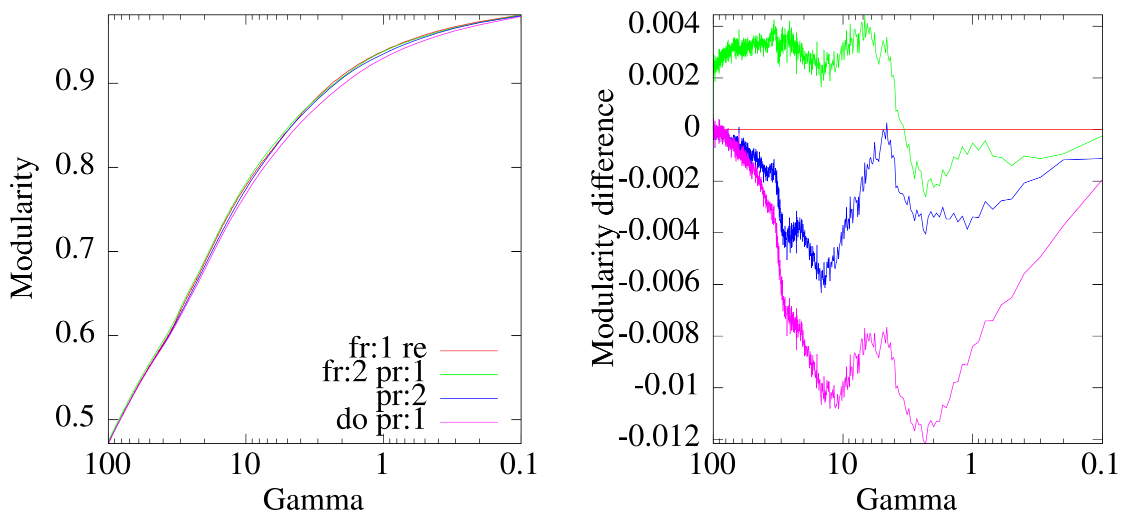


Figure 4.4.: Comparison of heuristics in the different aforementioned categories for the eu-2005-Graph. On the left hand you can see the modularity values of the different heuristics and on the right hand you can see the difference to the static algorithm.

In Figure 4.4 we can see some Pareto optimal heuristics with regard to quality and running time together with the result of running the static algorithm once for each γ.

As seen in Table 4.2 projecting the clustering from the middle level onto the lowest level not only improves the running time, but also often results in a better clustering. So if running time is less important this is probably the best solution.

The next heuristic seen in Figure 4.4 is the projection of the highest level to the middle level. This heuristic is even faster by one order of magnitude for large graphs, but the quality is also lower. With this heuristic the speedup also depends a lot on the graph size.

The required time can be further reduced by just going down one level down but this results in a lower quality and the improvement in the running time is much smaller compared to the reduction when going to the middle level instead of going down completely.

4.7. Influence of the heuristic and the parameter on the number of clusters

We have shown that the quality of the results are quite similar regarding the gamma modularity. To get an impression how much the resulting clusterings differ, we compared the number of resulting clusters.

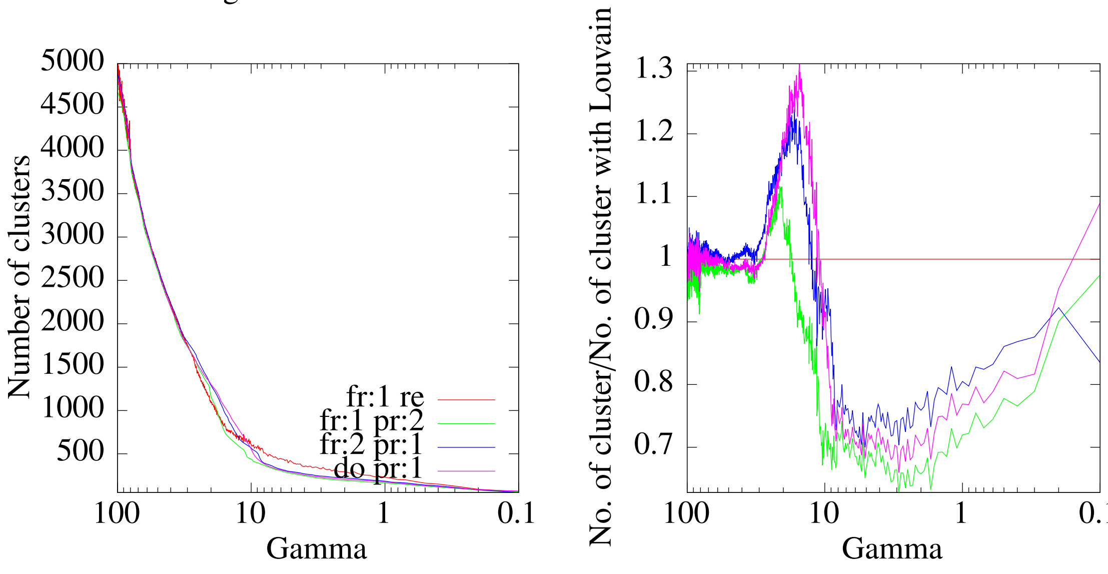


Figure 4.5.: Number of clusters in the eu-2005 graph with different γs.

At first sight of Figure 4.5 we might get the impression that the number of clusters are similar independent of the used heuristic in the eu-2005 graph. But at a closer look the distance in the y-direction is actually quite significant. The number of clusters in the found clusterings is higher between a γ of 30 and 20 compared with the static algorithm. For the heuristic that projects the clustering from the middle level to the lowest level it is up 10 % higher, for the heuristics which do not go down as far the number of clusters is up to 30 % higher. Between a γ of 20 and 8 the number of clusters drops until it is up one third lower than with the static algorithm. The plot of the number of clusters of the static algorithm also drops faster at the beginning and is flatter at the end, which causes the large differences in this range. These differences do not correspond to lower modularity values. In fact the heuristic that projects the middle level to the lowest level has a higher modularity value up to a γ of about 3, where it has about 30 % less clusters.

This was also observed with the other heuristic also with a lower modularity over all. The heuristic which projects the top level to the middle level has about the same gamma modularity as the static algorithm at a γ of 4, but has about 25 % less clusters at this point.

The number of clusters with the heuristics that use intermediate results is also less fluctuating over the whole range compared to the number of clusters we get with the static algorithm. This can be explained by the higher randomness associated with repeating the Louvain method once for each γ.

With the number of clusters alone we cannot say that the evaluated heuristics are less exact or worse than the static algorithm, because no direct correlation between the number of clusters and the quality of the clustering can be found.

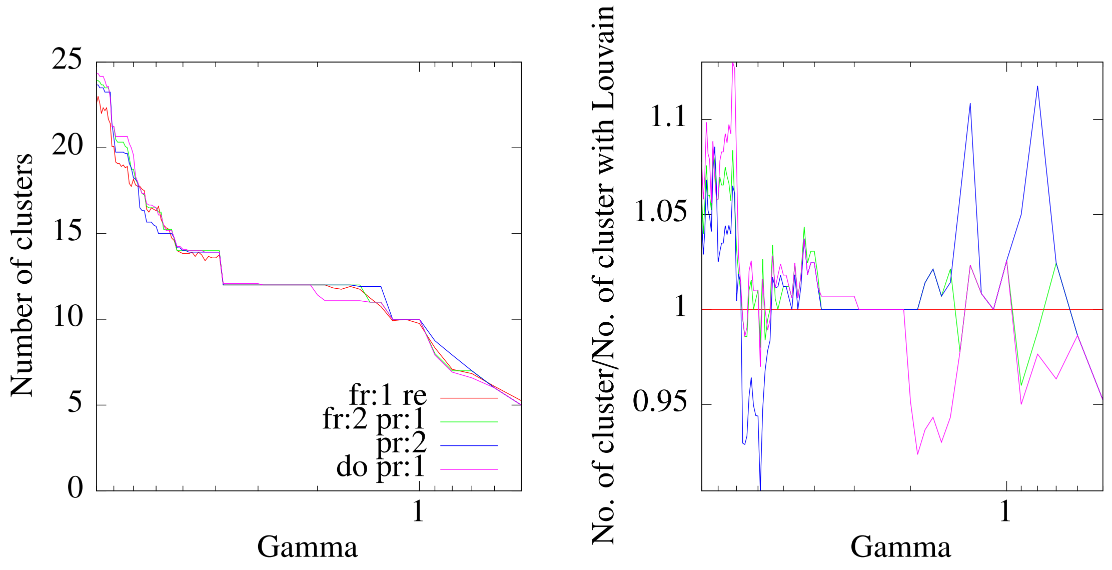


Figure 4.6.: Number of clusters in the football graph with different γs. The range was selected to match the plot in [Lam10].

If the number of clusters in a graph has a plateau over a range of gammas, which is caused by a “natural” clustering, it can be found with our heuristics. For example the results for the football graph can be seen in Figure 4.6. The range of γs has been adjusted, so that the results are comparable to the result that was found by Lambiotte in [Lam10].

The plateau is a bit shorter with the heuristic that only goes down one level, but compared to the results by Lambiotte [Lam10], the same plateau in the number of clusters with respect to the football graph can be found with a matched set of parameters. In the comparison between the heuristics the plateau is even a bit larger with the heuristic that projects the top level to the middle compared to the other heuristics or the static algorithm.

5. Conclusion

5.1. Summary

In our work we proposed a quick way to calculate clusterings with respect to gamma modularity for multiple γ values in a single run. We used the Louvain method to quickly recalculate clusterings for each γ and showed that the cluster hierarchy used in the Louvain method does not need to be recalculated for each new γ. We used different intermediate results to speed up the calculation of the next clustering.

If we use intermediate results from a higher level in our cluster hierarchy we can obtain large speedups (up to 200 times shorter running time) at the cost of a small quality loss. Surprisingly, if we use intermediate results from low levels we can get the same quality than without using intermediate results, but with a much shorter running time.

Although the number of clusters in the found clusterings differs greatly compared to the clusterings found with the static algorithm, the resulting gamma modularity was often better. That leads to the conclusion that not direct correlation between the number of clusters and the quality of a clustering can be found.

5.2. Future Work

To further improve the required time to find good clusterings it would be useful to find the plateaus without fully calculating all clusterings. Maybe this can be achieved by calculating the γ values were vertices get moved to another cluster and find a gap in this set of γ values that would correspond to a range of γ values were nothing changes in the clustering. Of course, moving vertices changes the aforementioned γ values, so this will only work if we do this while we are currently in the correct range already.

It would also be interesting to find out if other algorithms would find the same or similar clusterings with the gamma modularity as metric. When comparing the resulting clusterings it would also be interesting to find similarities between the results not just based on the number of clusters, but also using a measure to compare the clustering directly with each other.

The heuristics should also be evaluated on a graph with a well known hierarchical structure, so we can find out if it finds all interesting clusterings or if they get mixed up. It could happen that in a found clustering there are clusters from multiple levels.

In the long run it would be great to have a more direct way to find natural clusterings of a graph. For hierarchical graphs it is clear that a clustering metric has to have at least one parameter, but it should be possible, that we just pass a graph and a number k to a tool and get the k most interesting clusterings. Of course, these clusterings not only need to be good according to some metric, but also not to similar to each other.

Bibliography

[AFG08] A. Arenas, A. Fernández, and S. Gómez, “Analysis of the structure of complex networks at different resolution levels,” *New Journal of Physics*, vol. 10, no. 5, p. 053039, 2008. [Online]. Available: http://stacks.iop.org/1367-2630/10/i=5/a=053039

[BGLL08] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of community hierarchies in large networks,” 2008, preprint.

[FB06] S. Fortunato and M. Barthelemy, “Resolution limit in community detection,” in *Proceedings of the National Academy of Sciences of the United States*, 2006.

[Lam10] R. Lambiotte, “Multi-scale modularity in complex networks,” in *WiOpt*. IEEE, 2010, pp. 546–553.

[NG04] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Phys. Rev. E*, vol. 69, no. 2, 2004.

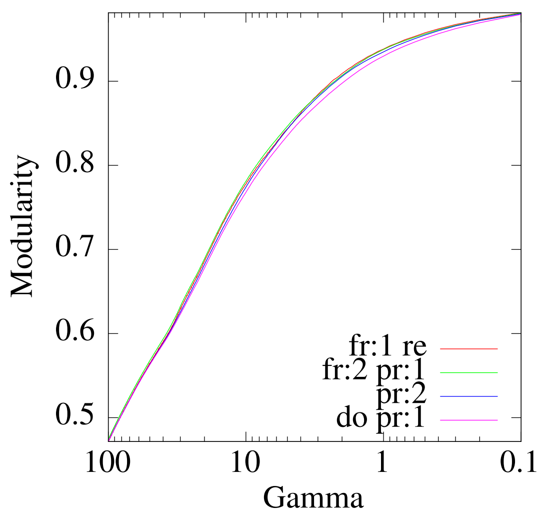
[RB04] J. Reichardt and S. Bornholdt, “Detecting fuzzy community structures in complex networks with a potts model,” *Phys. Rev. Lett.*, vol. 93, no. 21, p. 218701, Nov. 2004. [Online]. Available: http://prl.aps.org/abstract/PRL/v93/i21/e218701

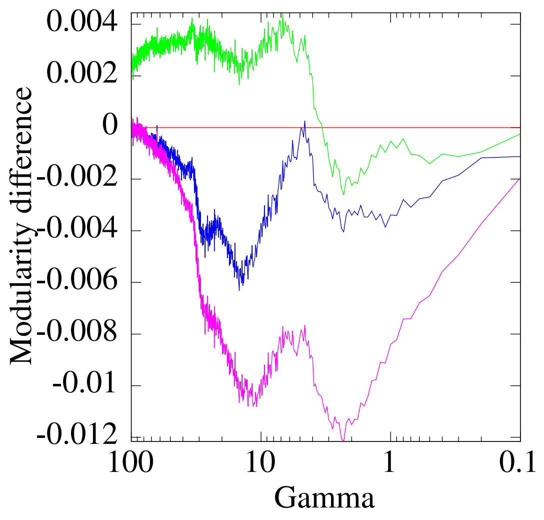
Appendix

A. Plots for all evaluated graphs

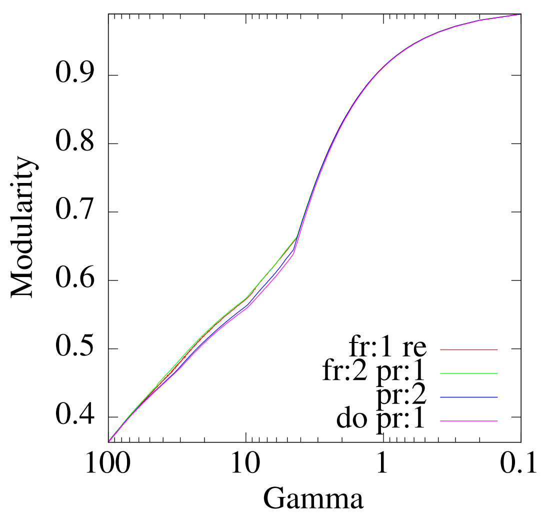
On the left the absolute modularity values are given and on the right side the difference to the static algorithm is shown.

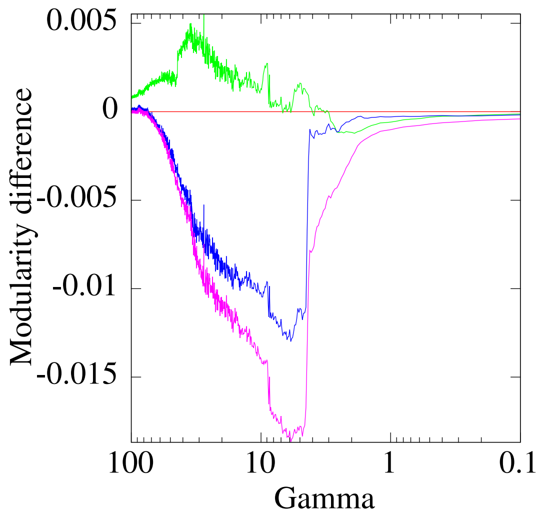
eu-2005



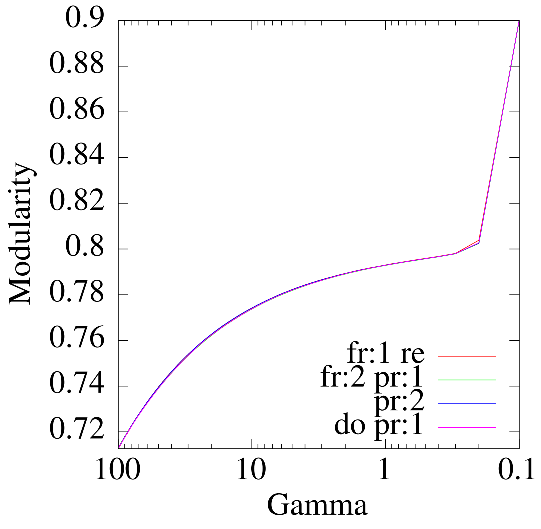


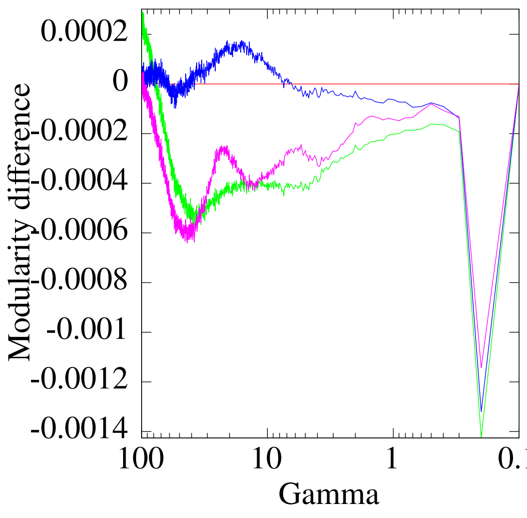
cnr-2000



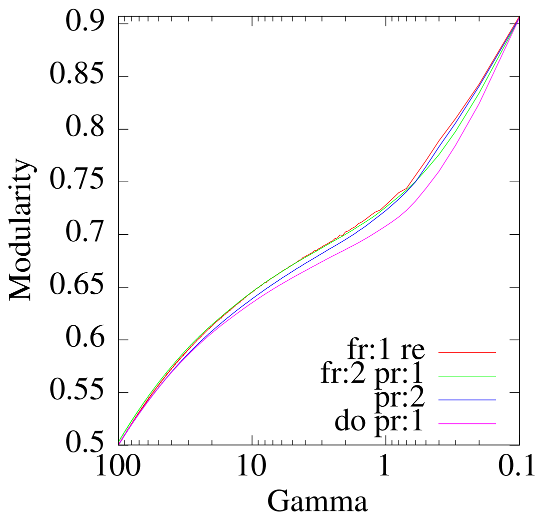


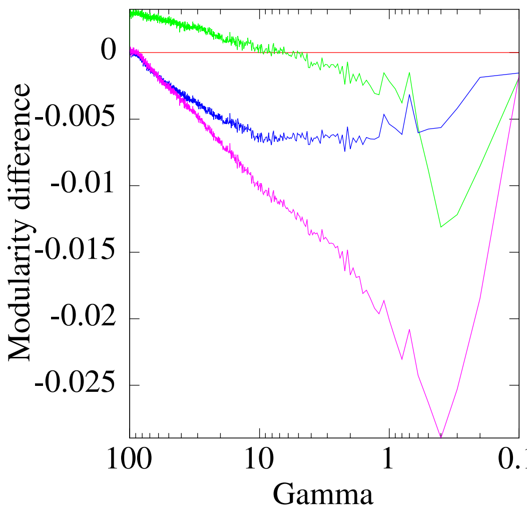
smallworld



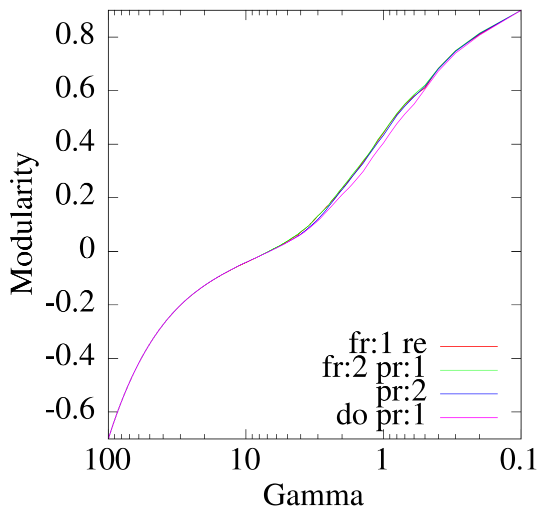


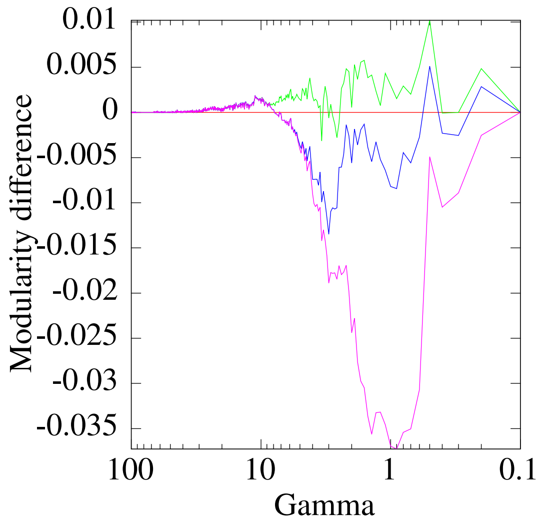
cond-mat-2005



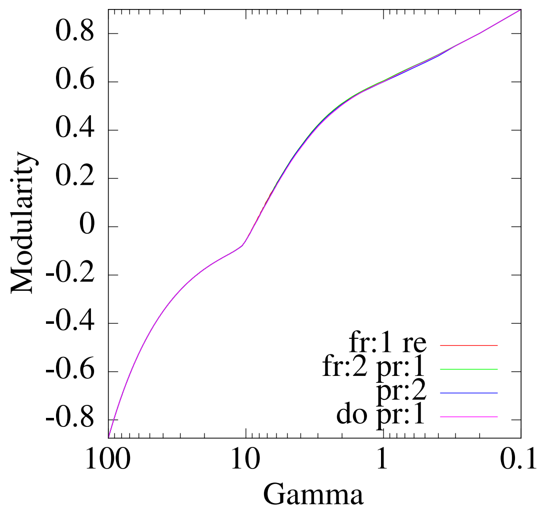


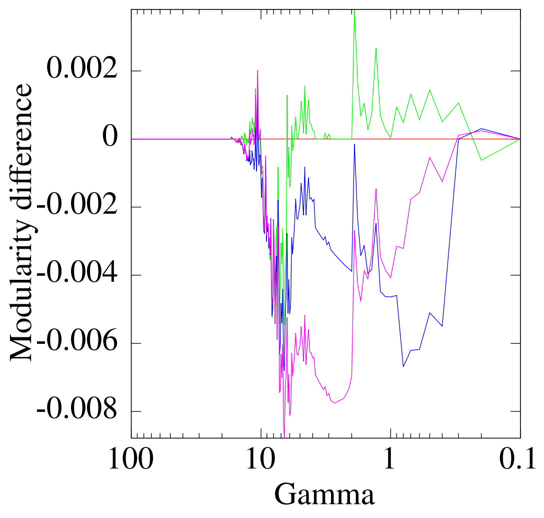
jazz



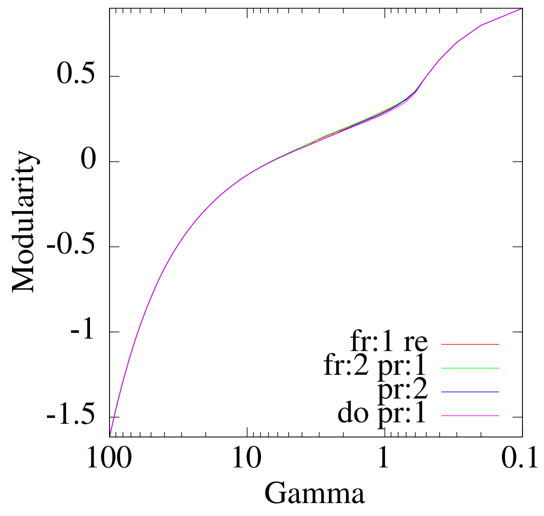


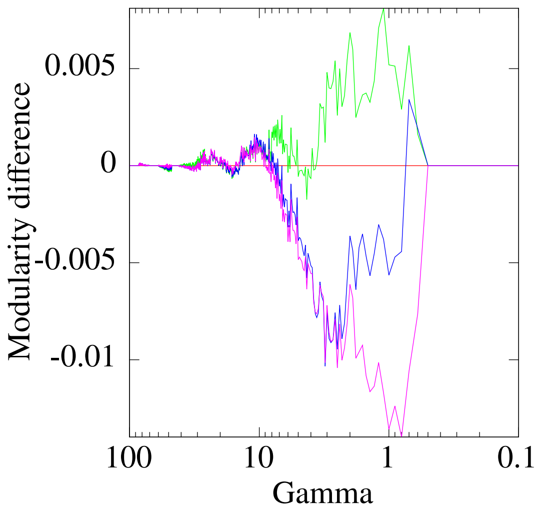
football



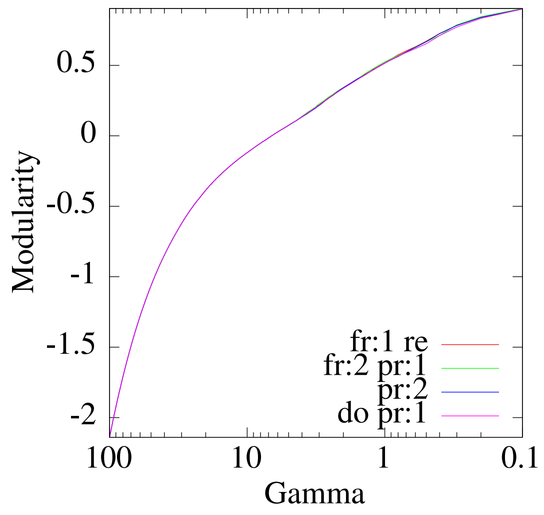


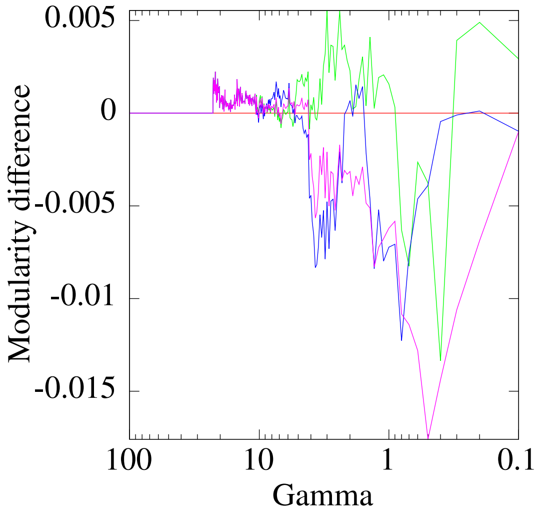
adjnoun





dolphins


Modularity
Gamma
fr:1 re
fr:2 pr:1
pr:2
do pr:1


Modularity difference
Gamma