Partition-based approach for a fast approximation of the clustering coefficient

Partitionsbasierter Ansatz für eine schnelle Approximation des globalen Clusteringkoeffizienten

Bachelor-Thesis von Patrick Alexander Ritter aus Frankfurt am Main

Februar 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
P2P Networks

Partition-based approach for a fast approximation of the clustering coefficient
Partitionsbasierter Ansatz für eine schnelle Approximation des globalen Clusteringkoeffizienten

Vorgelegte Bachelor-Thesis von Patrick Alexander Ritter aus Frankfurt am Main

1. Gutachten: Prof. Dr. Thorsten Strufe
2. Gutachten: Benjamin Schiller

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 01.02.2013

(Patrick Alexander Ritter)

Contents

Abstract

The global clustering coefficient (gcc) is an important metric to analyse graphs. For large graphs, absolute approaches are not feasible, since all current absolute approaches have a high computing complexity and mostly a large memory footprint due to matrix-based calculations. In this thesis, we provide an algorithm to get a fast approximation of the gcc even for big graphs.

In our algorithm, the graph is divided in partitions. For each partition, the absolute gcc is calculated. Each partition gcc is corrected concerning the ratio of existing edges to the maximal possible edges of the partition in relation to the ratio of the complete graph. This relation is the main contribution of this thesis, since it seems not to be previously published. The so corrected partition gccs are averaged, weighted by the node count of each partition.

We compare our algorithm to another previously published approximative algorithm on a broad set of graphs. We show that for most evaluated graphs, we can provide a significant faster sufficient approximation than the previous work. Since the time-consuming task in our approach, the calculation of the gcc for each partition, is independent for each partition, our approach can be computed easily in distributed systems, what makes it scalable for large graphs.

1 Introduction

In computer science, many structures are modelled by graphs. To characterise graphs, different graph metrics are used. Metrics provide information about the structure of a graph. When analysing graph represented data, we often want information about the graph structure.

In this thesis, we want to present an efficient methodology to calculate a fast approximation for the global clustering coefficient (gcc), also called transitivity. The gcc represents how clustered the nodes in a graph are and is defined by the number of triangles (three fully connected nodes) divided by the number of triplets (three nodes connected by at least two edges) of the graph. The gcc is for example very interesting when analysing social networks, since in such network often communities of highly clustered nodes exists.

Since the computation time to determine the gcc absolute growth super linear for bigger graphs, approximative algorithms are needed for big graphs. The most current algorithms try to approximate the gcc by sampling a graph. They only consider a subset of all nodes and edges of the graph. Besides deciding which nodes and edges should be considered, the algorithms have to adapt the resulting gcc with respect to the missing nodes and edges.

Our approach is partition-based. The graph is divided in multiple partitions. The joined partitions can span the hole graph, but our approach also can handle overlapping partitions and sets of partitions not covering the hole graph. Edges only persists between the nodes inside a partition.

The gcc is calculated first isolated for each partition. For each partition, the ratio of existing nodes relative to the number of maximal possible edges, based on the number of nodes (n nodes can have maximal $(n-1)\cdot n$ edges in a undirected graph) is calculated. Additionally, this ratio is calculated for the origin graph. Each partition gcc is multiplied with the ratio of possible edges of the origin graph divided by the ratio of possible edges for the specific partition, powered by $\frac{2}{3}$. Then the weighted average of all partition gccs is calculated, weighted by the number of nodes in each partition.

The number of nodes in each partition is a linear fraction of the number of nodes in the origin, but the number of edges is reduced considerably super linear. Because of this, the summed time to calculate the gcc of all partition is in most cases explicitly lower than the time to calculate the gcc for the origin graph.

We compare our work with a simple, but effective previously approximative approach which removes a big fraction of edges from the graph and weights the remaining edges to approximate the origin gcc. We use different random generated and real existing graphs to compare our approach with the existing work in calculation time and precision.

The main contribution of this thesis is the algorithm to aggregate the gcc of the partitions to a gcc for the origin graph, since the correlation of the number of possible edges between partitions and the origin graph does not seem to be described previously.

In Chapter 2, we will take a look at the existing approaches. The formal definitions are presented in Chapter 3. We introduce our new partition-based approach in chapter 4. The details of the implementation used for the evaluation is presented in Chapter 5. We conclude our results in Chapter 7. Possible further work is presented in Chapter 8.

2 Existing approaches

In this chapter, we will look at the latest existing approaches to calculate the global clustering coefficient (gcc). We differentiate between absolute approaches, which compute the exact gcc, and approximative approaches, which only compute an approximation.

2.1 Absolute approaches

2.1.1 Simple approach

To calculate the gcc of a graph g, the number of possible triangles T_p [1] in g and the number of actual triangles T_c [2] in g are necessary. A simple approach to determine them is to iterate trough all sets of actual and possible triangles and count them. Such an algorithm is sketched in Listing 2.1.

The iterating trough all possible triangle leads to a high complexity of $O(n^3)$[1], what is the big disadvantage of this approach.

Listing 2.1: Pseudocode representation of the simple approach

```
G = (V,E)
//Count triples and triangles
Tp = 0  // Number of possible triangles
Tc = 0  // Number of actual triangles
foreach n1 in V begin
  foreach n2 in neighbors(n1) begin
    /*
     * n1, n2 and each neighbor of n2 except (n1,n2,n1)
     * create a possible triangle
     */
    Tc = Tc + count(neighbors(n2)) + 1
    foreach n3 in neighbors(n2) begin
      if (n1 != n3) begin
        Tp := Tp + 1
      end
    end
  end
end
//Calculate gcc
gcc = Tc / Tp
```

2.1.2 Recent absolute approaches

All considered absolute approaches seem not to be suitable for big graphs. Modern absolute approaches as the approach by by Matthieu Latapy et al. [2] are matrix-multiplication based. Besides the fact, that a relatively high complexity remains, the main problem of this approaches is the necessary memory. The approach of Matthieu Latapy et al., for example, needs to hold the complete adjacency matrix in memory. Since the adjacency matrix of a graph grows quadratic with the node count, this results in a massive memory footprint.

Because of the big memory footprint and relatively long runtimes for absolute approache on big graphs, we will not consider absolute approaches further..

[1] Definition 3.2.3

[2] Definition 3.2.3

2.2 Approximative approaches

2.2.1 Approach by Charalampos E. Tsourakakis et al.

The approach of Charalampos E. Tsourakakis1 , Mihail N. Kolountzakis , and Gary L. Miller: "Approximate Triangle Counting" from 2009[1] is based on the idea of removing a fixed ratio $p \in (0,1)$ of edges from the graph. The remaining edges are reweighed with $\frac{1}{p}$ to compensate the removed edges.

Specifications

Each edge in the origin graph is reduced with a possibility $1-p, p \in (0,1)$, so that the ratio p of all edges is removed. Earlier work proposed for a graph with n the following choice for p:

$$p = \frac{1}{\sqrt{n}}$$

The author of [1] claim that this choice of p is not sufficient in all cases and propose to use a very low p and iterative increase p until the difference of the approximated gcc is adequate small enough for the own purpose.

To calculate the approximation of the gcc, the gcc of the reduced graph is calculated with a traditional absolute algorithm as shown in Listing 2.2 to count triples and triangles. To consider the weighted edges, the approximative gcc C is calculated as follows:

$$C^* = \frac{T_c \cdot \frac{1}{p^3}}{T_p} = \frac{T_c}{T_p \cdot p^3}.$$

Listing 2.2: Pseudocode representation of the approximative approaches

```
G = (V,E)
//Remove amount of (1-p) edges
foreach e in E begin
  if(rand < (p) // 0 <= rand() <= 1
    E:=E/{e}
end
//Count triples and triangles
Tp = 0  // Number of possible triangles
Tc = 0  // Number of actual triangles
foreach n1 in V begin
  foreach n2 in neighbors(n1) begin
    /*
     * n1, n2 and each neighbor of n2 except (n1,n2,n1)
     * create a possible triangle
     */
    Tc = Tc + count(neighbors(n2)) + 1
    foreach n3 in neighbors(n2) begin
      if (n1 != n3) begin
        Tp := Tp + 1
      end
    end
  end
end
//Calculate gcc
gcc = Tc / (Tp * p^3)
```

Problems

The approach has two principal problems: The choice of p and the use of multi core or distributed environments.

Especially for relatively small graphs (some thousands nodes and edges), a sufficient choice of p is not easy. The proposed rule of thumb, $p = \frac{1}{\sqrt{n}}$, leads to too far reduced graphs where the approximated gcc is not usable.

Since the gcc for the complete reduced graph has to be calculated, which is the most time-consuming part in this approach, the approach is relatively hard to distribute on multiple cores or systems. The approaches to calculate the gcc parallel on multiple processing unit are matrix multiplication based, which has the big disadvantage of a high necessary amount of memory for adjacency matrix alike data structures.

3 Preliminaries

3.1 Graph definitions

Graph

Given a graph G consisting of a set of vertices V and directed edges E:

$$G = (V, E)$$

$$E \subseteq V \times V$$

If $\forall v_1, v_2 \in V : (v_1, v_2) \in E \Rightarrow (v_2, v_1) \in E$, we call this graph undirected, if not, we call this graph directed.

Weighted edges and nodes

The function $w_e \subseteq E \times \mathbb{R}$ assigns a weight to each edge, the function $w_v \subseteq V \times \mathbb{R}$ assigns a weight to each node.

Neigborhood

We define the set of all neighbour nodes directly connected to a node n_i as N_i:

$$N_i = \{v_a \in V | (v_i, v_a) \in E \wedge (v_a, v_i) \in E\}$$

E_{N_i} defines the set of all edges between neighbour nodes of a node n_i:

$$E_{N_i} = \{(v_k, v_l) \in E | v_l \in N_i \wedge v_k \in N_i\}$$

3.2 Clustering coefficient

3.2.1 Local clustering coefficient

The local clustering coefficient C_i is calculated as follows for a node $v_i \in V$:

$$C_i = \frac{|E_{N_i}|}{|N| \cdot (|N| - 1)}$$

3.2.2 Average clustering coefficient

The Network average clustering coefficient for a graph is

$$cc = \frac{\sum C_i}{|V|}$$

3.2.3 Global clustering coefficient

For the global clustering coefficient, we define a set of possible triangles, T_p, and the actual set of triangles, T_c.

$$T_p = \{(v_a, v_b, v_c) \in V \times V \times V \wedge \{(v_a, v_b), (v_b, v_a), (v_b, v_c), (v_c, v_b)\} \subseteq E \wedge |\{v_a, v_b, v_c\}| = 3\}$$

$$T_c = \{(v_a, v_b, v_c) \in V \times V \times V \wedge \{(v_a, v_b), (v_b, v_a), (v_b, v_c), (v_c, v_b), (v_a, v_c), (v_c, v_a)\} \subseteq E \wedge |\{v_a, v_b, v_c\}| = 3\}$$

The global clustering coefficient for a graph is

$$C = \frac{|T_c|}{|T_p|}$$

We are interested in approximations $cc^* \approx cc$ and $C^* \approx C$.

3.2.4 Different definitions of triangles

We use the same definition for triangles as used in [1] and [2] to make our results comparable. In case of this definition, a triangle is only a triangle if all possible edges between the three nodes exists. So each triangle has three edges in the undirected case or six edges in the directed case. For the directed case, also other definitions are used.

In [3], single directed triangles are used. The nodes $\{v_1, v_2, v_3\}$ are a triangle if the edges $\{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$ exists. The edges $\{(v_1, v_3), (v_3, v_2), (v_2, v_1)\}$ would form a second, separate triangle.

In other fields of research, motif like definitions for triangles are used. In [4], where the authors use clustering coefficients as metric for system risks in banking networks, besides the previously mentioned single directed triangles also sets of edges as $\{(v_1, v_2), (v_2, v_3), (v_3, v_1)\}$ would form a triangle.

3.3 Partitioning schemes

3.3.1 Categories

We can sort partitioning scheme roughly by two attributes: if they are overlapping or non overlapping and if they are complete or incomplete.

When using non overlapping partitions, each node is assigned to maximal one partition. This has the benefit that depending on the implementation, the data structures to store all neighbours of a node can be less memory consuming, as discussed in Chapter 5. When using overlapping partitions, nodes can be assigned to multiple partitions. This can improve the precision of the approximation.

Complete partitioning schemes result in a set of partitions which contain all nodes of the origin graph. Using incomplete partitioning schemes, some nodes may be ignored.

In this thesis, we mainly use non overlapping, complete partitioning schemes, because they have in most cases less parameters then overlapping and or incomplete partitioning schemes. Since our aggregation function presented in Chapter 4 can handle all the previously presented variants, our choice of partitioning schemes can be seen as proof of concept. The usage of the more complex overlapping and incomplete schemes in further work is discussed in Chapter 8.

3.3.2 Non overlapping Breadth-first search

We have a parameter $n \in \mathbb{N}$, defining the number of subgraphs (with $n \leq |V|$). We choose n distinct nodes $V' \subseteq V$. For each $v_i \in V'$, we define a corresponding subgraph G_i containing all nodes $v_x \in V$ where $|sp(v_i, v_x)|$ is smaller than with all other $v_i \in V'$. If there exists a $v_x \in V$ with equal shortest path hop count to multiple $v_i \in V'$, we assign it randomly to a subgraph i.

$$E_i = \{(v_1, v_2) \in E | v_1 \in V_i \wedge v_2 \in V_2\}$$

3.3.3 Unweighted Depth-first search

We have a parameter $n \in \mathbb{N}$, defining the number of subgraphs (with $n \leq |V|$), and a parameter $\alpha \in \mathbb{N}$, defining the number of steps per subgraph (with $\alpha \geq |V_i|$ for all subgraphs G_i). We choose n distinct nodes $V' \subseteq V$. For each $v_i \in V'$, we define a corresponding subgraph G_i by adding v_i to G_i and executing the following steps α times. We do this parallel for all n distinct nodes, so this results in non overlapping subgraphs.

1. Choose a random, previously unvisited neighbour node v_a of the last node added to the subgraph G_i. If v_a has no unvisited neighbour, choose the node added before v_a and do this first step again. If no such node exists, terminate.

2. Add v_a to the subgraph G_i

3. Continue with the first step

3.3.4 Weighted Depth-first search

We have a parameter $n \in \mathbb{N}$, defining the number of subgraphs (with $n \leq |V|$), and a parameter $\alpha \in \mathbb{N}$, defining the number of steps per subgraph (with $\alpha \geq |V_i|$ for all subgraphs G_i). We choose n distinct nodes $V' \subseteq V$. For each $v_i \in V'$, we define a corresponding subgraph G_i by adding v_i to G_i and executing the following steps α times. We do this parallel for all n distinct nodes, so this results in non overlapping subgraphs.

1. Choose a random, previously unvisited neighbour node v_a of the last node v_l added to the subgraph G_i. The chance to take a neighbour node v_j is inverse proportional to the edge weight of (v_l, v_j), so nodes with a low edge weight are favoured. If v_a has no unvisited neighbour, choose the node added before v_a and do this first step again. If no such node exists, terminate.

2. Add v_a to the subgraph G_i

3. Continue with the first step

$$E_i = \{(v_1, v_2) \in E | v_1 \in V_i \wedge v_2 \in V_2\}$$

4 A partition-based approximation

4.1 Observations

A first simple try to approximate the gcc with a partition based approach could be to partition the graph, calculate the gcc for each partition and calculate the average of all computed partition gccs.

The resulting gcc approximation is not accurate. When looking at each partition gcc, we see that it is higher then the actual gcc of the complete graph if the ratio of maximal possible edges in relation to the number of actual existing edges is higher and vice versa.

The number of maximal possible edges in a undirected graph G with nodes is defined as $max_e(G)$, the ratio of maximal possible edges in relation to the number of actual existing edges as $act_e(G)$.

$$max_e(G) = max_e((V,E)) = (|V|-1)\cdot|V|$$

$$act_e(G) = act_e((V,E)) = \frac{|V|}{max_e(G)}$$

The correlation is not linear proportional. If we had a Graph G and a partition G' with gccs C and C', the correction of C' with the factor $\frac{act_e(G)}{act_e(G')}$ is not sufficient. We observed that the multiplication with $\frac{act_e(G)}{act_e(G')}$ is to strong. If the gcc C' of the partition is lower than the gcc C of the complete graph, it will be higher then C after corrected with the factor $\frac{act_e(G)}{act_e(G')}$ and vice versa.

So the main task is to find a method to correct the gcc of the partitions correlating to $act_e(G)$ and $act_e(G')$. We define the method to correct the gcc C' of the partition G' of the graph G as $aggr(C', G, G')$.

4.2 Definition

We define three different aggregation schemes to correct the partition gcc. A scalar, a sum and a power based function, which weaken the corrector factor $\frac{act_e(G)}{act_e(G')}$. Each function has a parameter $\alpha \in [0,1]$.

$$aggr_{\text{pow}}(C', G, G') = C' \cdot \frac{act_e(G)}{act_e(G')}^{\alpha}$$

$$aggr_{\text{sum}}(C', G, G') = C' \cdot \left(\frac{act_e(G)}{act_e(G')} + \alpha\right) \cdot \frac{1}{1+\alpha}$$

$$aggr_{\text{scalar}}(C', G, G') = C' \cdot \frac{act_e(G)}{act_e(G')} \cdot \alpha$$

Our approximation C^* for a Graph G, divided in n partitions $G_i = (V_i, E_i)$ with $i \in [1 \dots n]$, is defined as the weighted average of all partition gccs with the exact C_i corrected by one of the previously defined aggregation functions.

$$C^* = |V| \cdot \sum_{i=1}^{n} \frac{aggr(C_i, G, G_i)}{|V_i|}$$

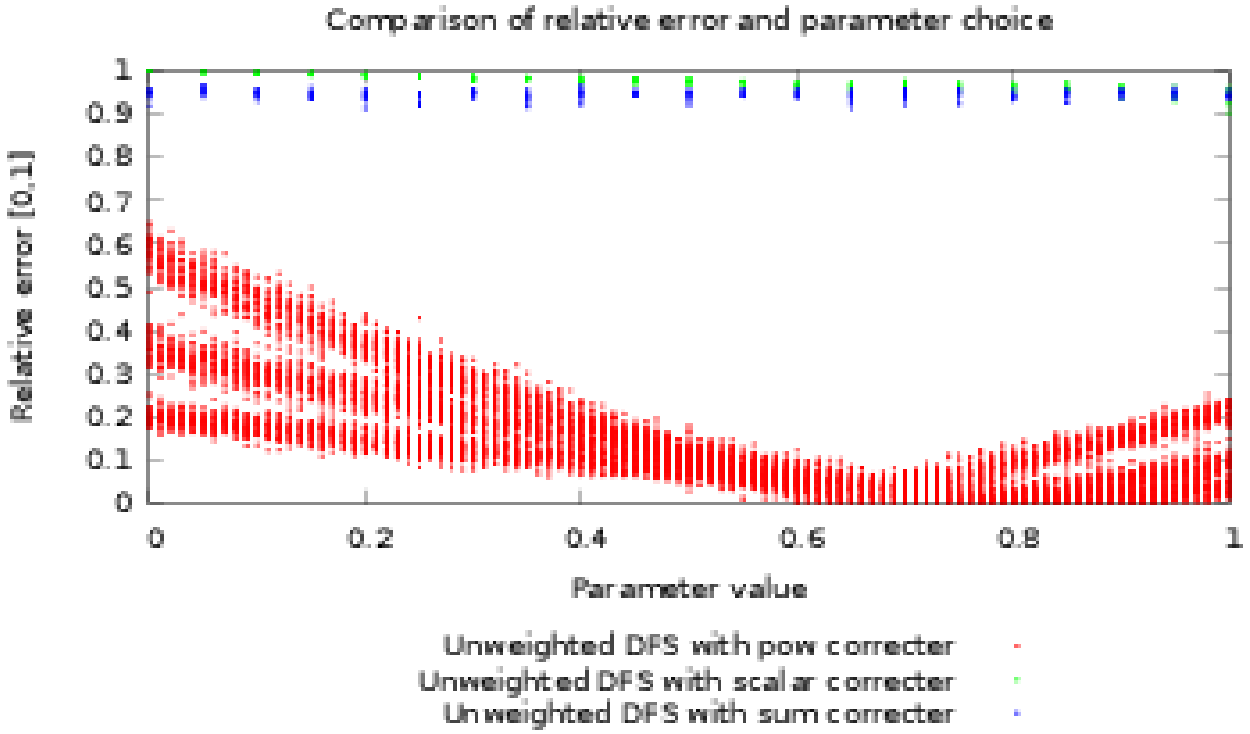

Figure 4.1: Comparison of power-based, scalar-based and sum-based aggregation schemes for the unweighted DFS partitioning scheme

4.3 Complexity

Since we want a fast approximation for the gcc in big graphs, a low computing complexity is important. The time-consuming part in our algorithm is the absolute calculation of the gcc for each partition. The partition used partition schemes in Section 3.3 have linear runtime. The aggregation functions to correct each partitioning scheme defined in Section 4.2 have constant runtime.

Since we use a simple standard approach to calculate the absolute gcc for each partition as presented in Section 2.1.1, this calculation has a complexity of $O((|V|)^3)$ for a partition with node set V. Since we partition the graph and each partition only has a fraction of the nodes of the origin graph, we reduce the complexity to $O\left(n \cdot \left(\frac{|V|}{n}\right)^3\right)$ for n partitions with a consistent node distribution to all partitions. For the complexity to calculate the absolute gcc for each of the n partitions in comparison to calculate the gcc for the complete graph with node set V holds:

$$(|V|)^3) > (\frac{(|V|)^3}{n^2})) = n \cdot (\frac{(|V|)^3}{n^3}) = n \cdot \left(\frac{|V|}{n}\right)^3$$

That means if we have n non overlapping partitions with an equal number of nodes in each partition, we can assume a much smaller complexity for our gcc approximation then for the calculation of the absolute gcc for the hole graph.

4.4 Comparison of partition aggregation schemes

We compared our previously defined three aggregation schemes (power-based, scalar-based and sum-based) to determine the best variants and a sufficient choice of the parameter $\alpha \in [0,1]$. We approximated the gccs of different bidirected graphs (see Section 9.1 for a list) for each of the three aggregation schemes with each of the three partitioning schemes of Section 3.3 with different values of α (α was set to 101 values between 0 and 1 with step size 0.01). Overall, we calculated over 450.000 approximations. The results in Figure 4.1 are only a small subset of all calculated approximations due to clearness, but the results are similar for all three partitioning schemes and most graphs.

Since we want an approximation with a low error, the pow-based aggregation scheme is clearly the best. For this scheme, the results in Figure 4.1 suggest that the best choice of α is roughly around $\alpha \approx 0.7$. Further analysis suggest that the aggregation scheme is optimal for $\alpha \to \frac{2}{3} \approx 0.67$.

4.5 Practical implementation

Based on our previous findings, we will use the pow-based aggregation scheme with $\alpha = \frac{2}{3}$. To calculate the absolute gcc for each partition, we use a simple approach as described in Section 2.1.1.

5 Implementation

In this chapter, we will describe some aspects of the concrete implementations of the previously presented algorithms used in this thesis.

All implementations are based on the Graph-Theoretic Network Analyzer [1] by Benjamin Schiller (P2P working group, TU Darmstadt) and Contributors. It is a java-based framework to analyse graph data. This framework provides especially all methods to read real world graph data and generate random graphs for evaluation purposes.

5.1 Simple approach

The simple approach described in Section 2.1.1 is implemented as shown in Listing 2.1. To manage the set of neighbours for each node, we use an one dimensional array of integer HashSets[2]. One HashSet represents all neighbours of one node.

5.2 Approximation

The approximation approach described in Section 2.2.1 is implemented similar to the previously discussed simple absolute approach. We also use here an one dimensional array of integer to represent the neighbours of the nodes as main data structure.

5.3 Partition-based approach

For our own approach, we also use the previously mentioned array of integer HashSets in a similar manner as one of the main data structures. But in this approach, we also need a data structure to store boolean data for each node, for example to store already visited nodes in the partitioning process.

Two aspects are important: Since a boolean variable in Java has in most implementations a memory footprint of multiple bytes, even as part of a boolean array. Since our memory footprint would be very high for large graphs, we use Java BitSets [3], where each bit only has a memory footprint of one bit. Since we have many accesses to the BitSets, the time for read and write accesses to the BitSet should be as small as possible. Since the java.util.BitSet implementation does not provide optimal performance, we use the OpenBitSet of the Apache Lucen Project[4].

[1] http://www.p2p.tu-darmstadt.de/research/gtna/

[2] http://docs.oracle.com/javase/6/docs/api/java/util/HashSet.html

[3] http://docs.oracle.com/javase/6/docs/api/java/util/BitSet.html

[4] http://lucene.apache.org

6 Evaluation

6.1 Setup

Choice of p for approximation approach

A problem of the approximative approach used for comparison 2.2.1 is a sufficient choice of p. The optimal choice is individual for each graph. A too high p results in a long runtime, a too low p results in a bad precision. To avoid such problems, we use a broad bandwith of parameter for $p \in [0,1]$ for a graph $G = (V,E)$:

$$0.75, 0.5, 0.2, 0.1, 0.01, 0.02, 0.05, 0.001, 0.005, 0.0001, \frac{1}{\sqrt{|V|}}$$

Choice of graphs

We also used very different graphs for evaluation. Besides different randomly generated graphs (small world graphs, random graphs, power law graphs), we use crawls of real world networks. All tested graphs could be grouped by similar results, so we limit our discussion in this thesis to the graphs listed in Section 9.1. Before starting the evaluation, all graphs were transformed to bidirected, unweighted cycle free graphs.

6.2 Runtime vs precision tradeoff for partition-based approach

6.2.1 Methodology

We run every partitioning scheme with our heuristic with different partition counts $(2,3,4,5,7,10,15,20,50)$ multiple times, which results in over 10.000 to 50.000 approximated gccs per graph. We measure the time to approximate the gcc and calculate to relative error (error relative to the absolute gcc).

6.2.2 Results

We discuss the results for two selected graphs, shown in Figure 6.1 and Figure 6.2. We see that the approximation with the BFS partitioning scheme is for both graphs very imprecise and not useful. The unweighted and the weighted DFS offer nearly equal precision, but the unweighted DFS is in many cases slightly more precise or faster.

In both graphs, the results are grouped locally by the partition count of the corresponding approximation. In Figure 6.1, the runtime seems for equal partition count even constant, but we have to consider that this effect is increased by the log scale of the runtime. In Figure 6.2, the runtime and precision varies more for the same partition count, each partition count forms a small cloud of approximations.

6.3 Comparison of heuristic and existing approaches

6.3.1 Methodology

We run every partitioning scheme with our heuristic with different partition counts $(2,3,4,5,7,10,15,20,50)$ multiple times, which results in 10.000 to 50.000 approximated gccs per graph. We also run the previous approximation approach with different values for p (see Section 6.1), which results in 13.000 to 123.000 approximation per graph. We measure the time to approximate the gcc and calculate the relative error (error relative to the absolute gcc).

6.3.2 Results

We discuss the results for a selection of graph. The remaining plots can be found in Section 9.2.

For the graphs in Figure 6.5, Figure 6.6, and Figure 6.3 our heuristic is with the weighted and unweighted DFS clearly much faster then the previous work. For the BarabsiAlbert-5000 graph in Figure 6.5, we also can provide more precision. In Figure 6.3, our heuristics provides nearly the same precision as the previous approach. For the Gilbert graph in Figure 6.6, we can provide a much faster approximation, but the previous work provides more precision for longer runtimes. The Web of Trust graph in Figure 6.4 is an outlier: Our heuristic can not provide a sufficient fast or precise approximation in comparison to the previous work. This also holds for the other Web of Trust crawls.

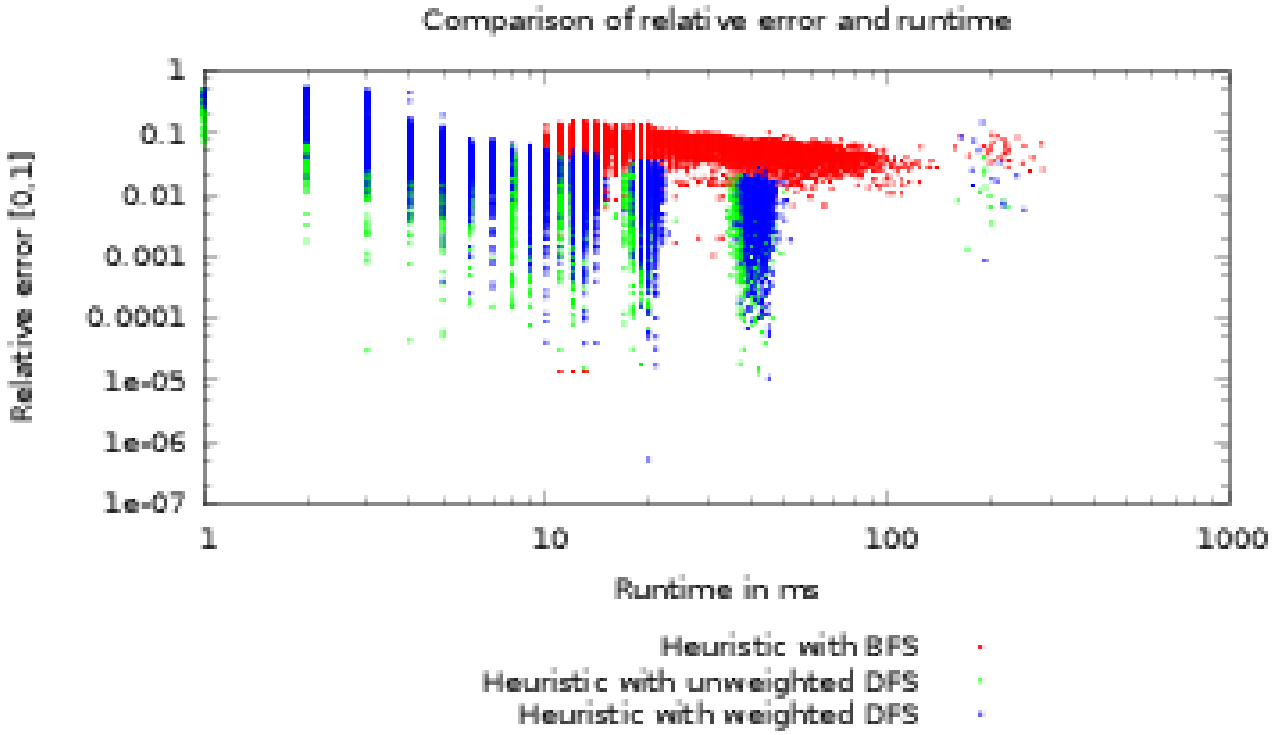

Figure 6.1: Runtime and precision for BarabasiAlbert-5000 with unweighted DFS, weighted DFS and BFS partitioning schemes

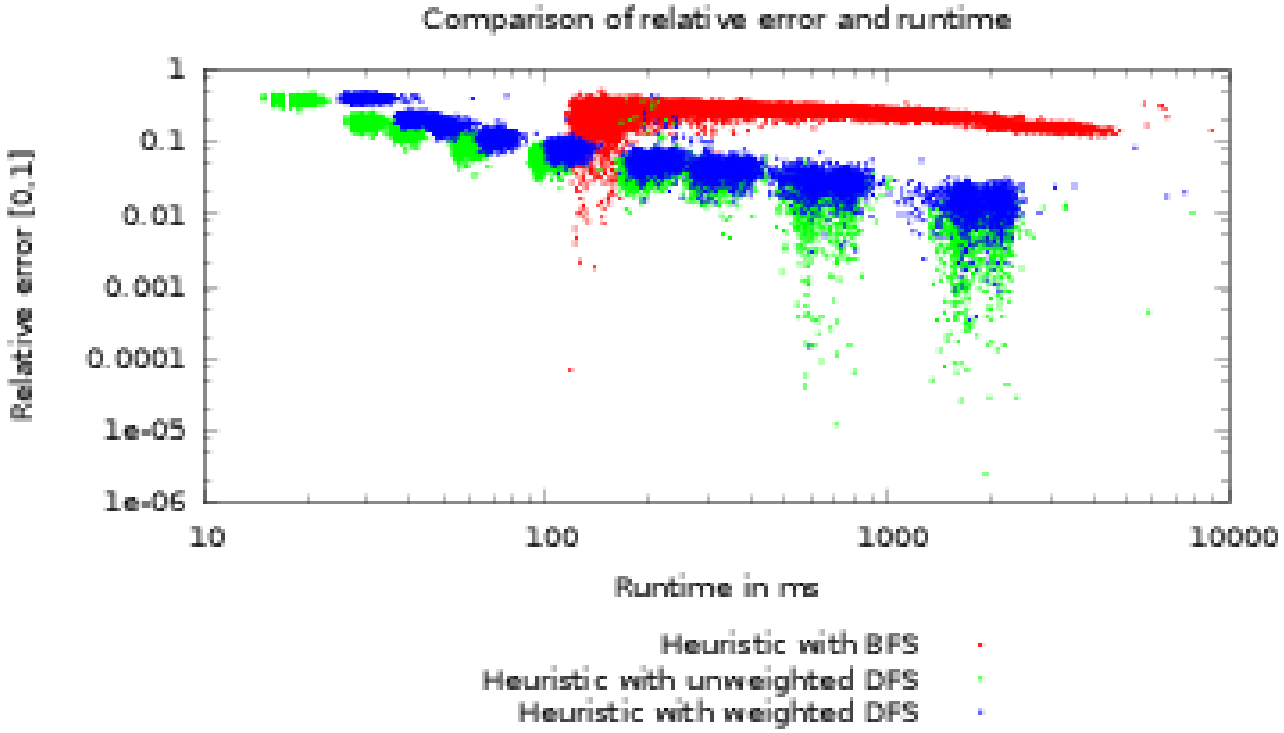

Figure 6.2: Runtime and precision for BarabasiAlbert-20000 with unweighted DFS, weighted DFS and BFS partitioning schemes

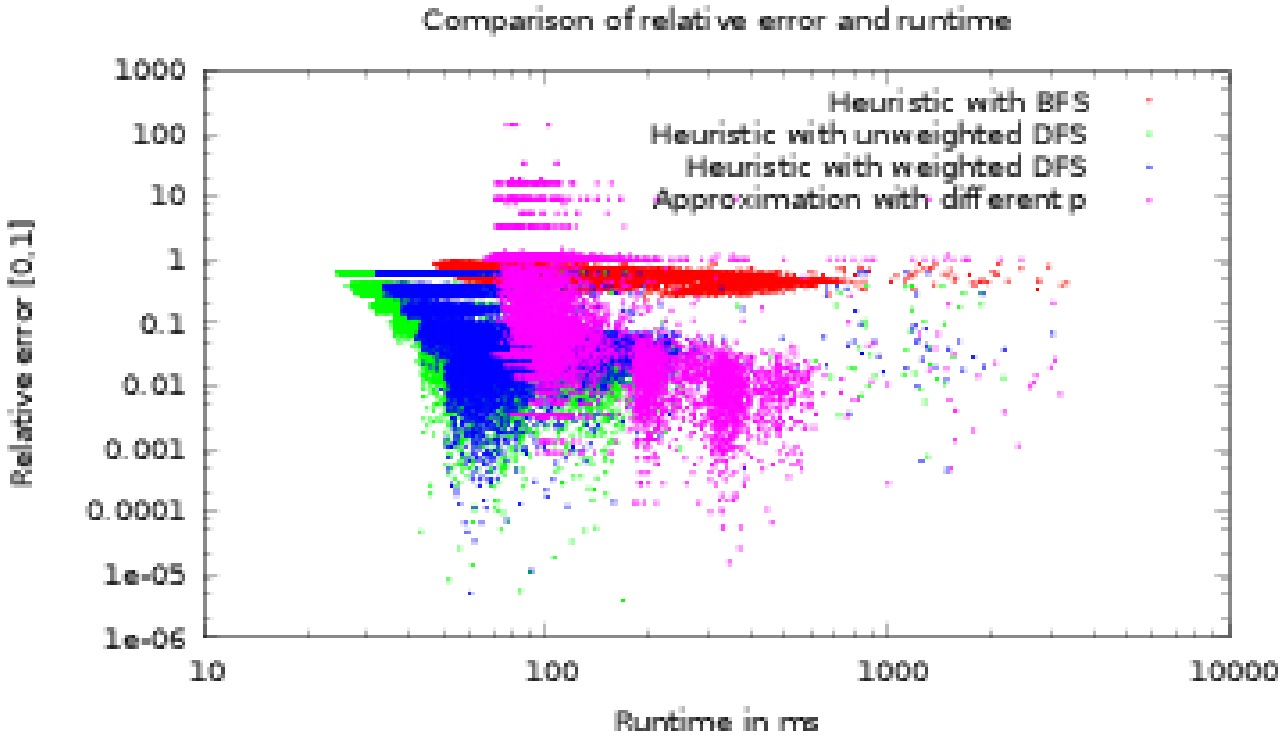

Figure 6.3: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on LSCC-6_analyze_buddy_aug_sept_2011_csv_gtna

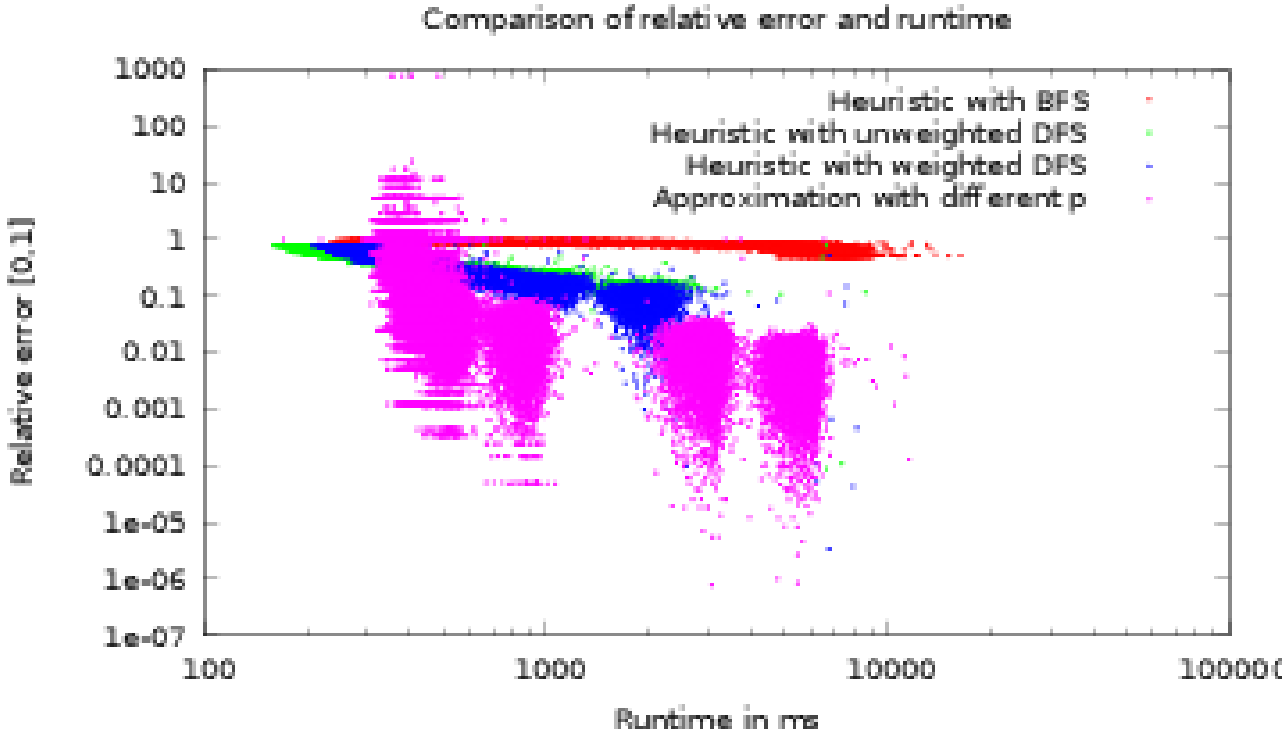

Figure 6.4: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on 2005-02-25_wot_gtna

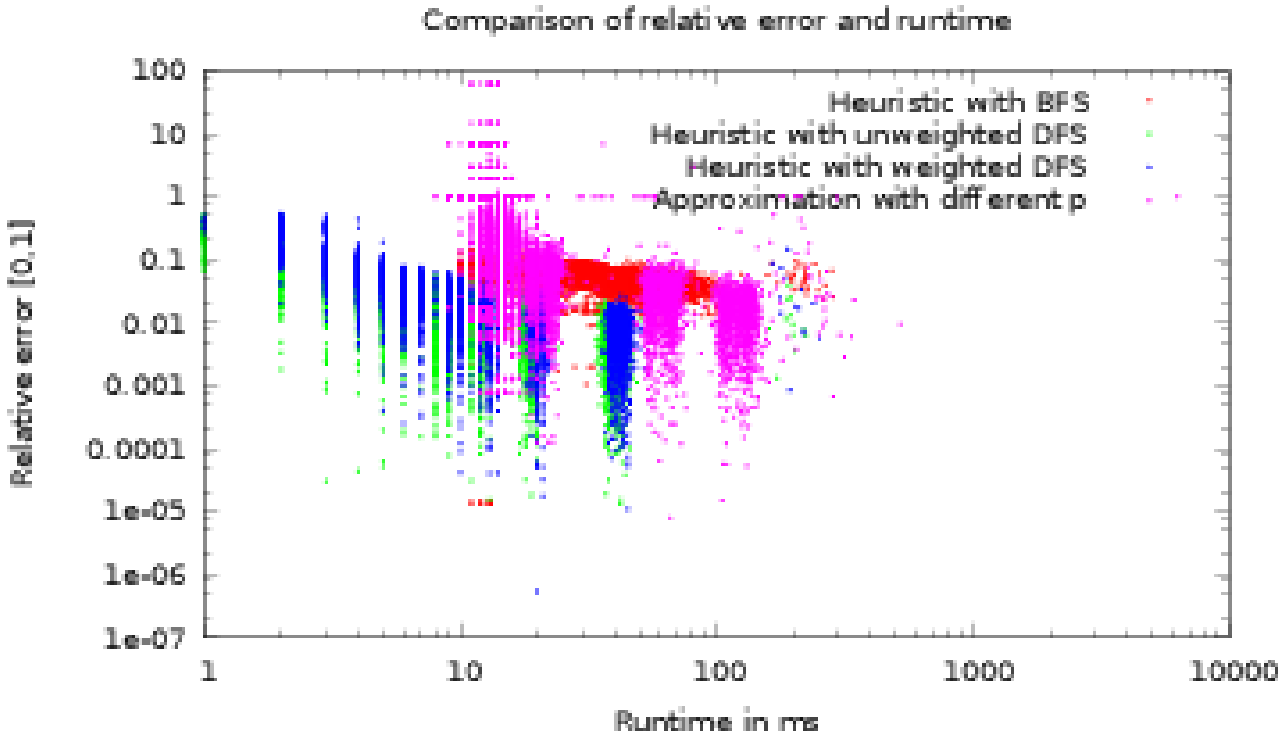

Figure 6.5: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on BarabasiAlbert-5000

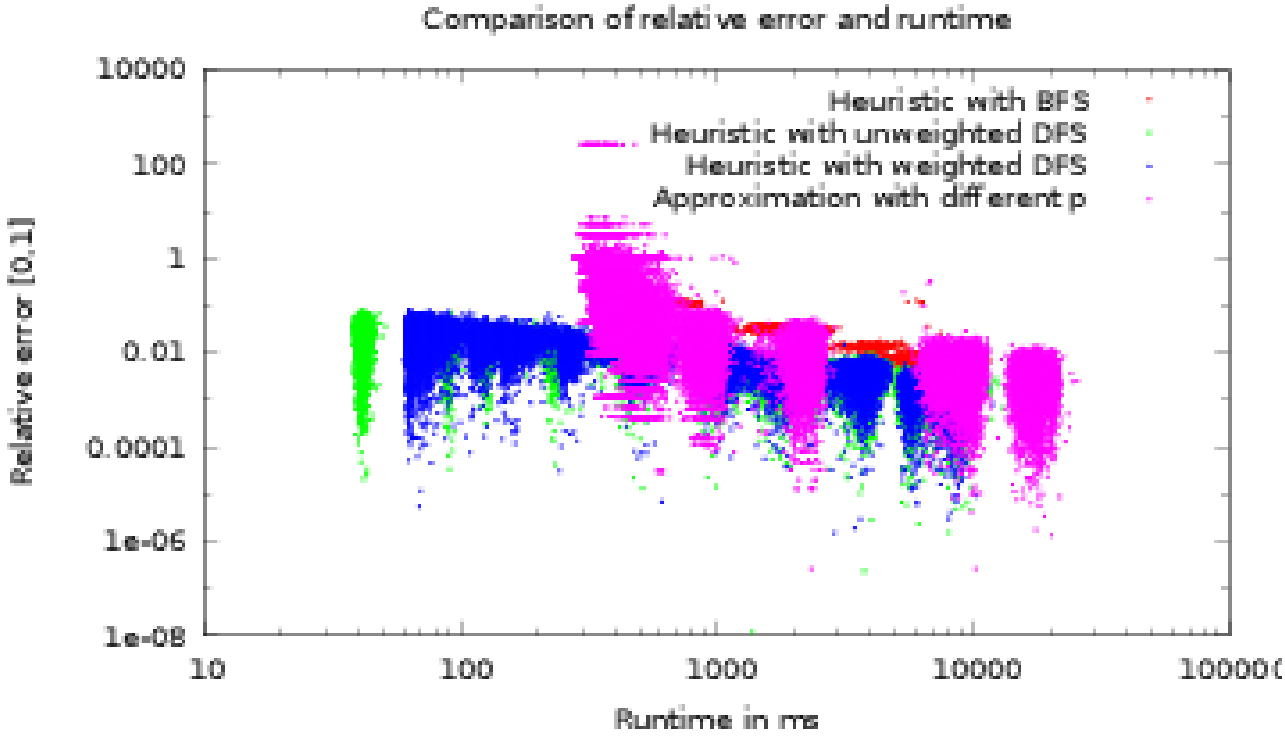

Figure 6.6: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on Gilbert

7 Summary and conclusion

In this thesis, we proposed a new partition based approach for a fast approximation of the clustering coefficient.

We took a view on previously published approaches and identified their possible problems. For our partition based approach, we choose with the unweighted DFS, the weighted DFS and the BFS three simple partition schemes. We identified the finding of an adequate aggregation scheme, which corrects the gccs calculated for each partition, as main task to solve. We experimented with a sum-based, a scalar-based and a power-based aggregation scheme. We gain as result, that with an parameter of $\alpha = \frac{2}{3}$, the power-based aggregation scheme is the best. We evaluated our heuristic with the power-based aggregation scheme against a previous approximation approach by Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller. We could provide for most graphs a much faster approximation of the gcc than their approach could, when we used the weighted or unweighted DFS as partitioning scheme.

For many of the considered graphs, we can provide a fast approximation. In this case, our heuristic should be used with the unweighted DFS, which performed slightly better than the weighted DFS. The non overlapping BFS tested by us is not useful. To provide a fast approximation, the partition count for our heuristic should be not to small. A possible methodology to use it could be to choose a high partition count, apply the heuristic and repeat this with a continuously decreasing partition count until the difference between two heuristic runs is small enough or the runtime of a heuristic run gains too long.

Due to the variable partition count, our heuristic can provide approximations even for very big graphs. This is supported by the fact, that the time-consuming part, the computation of the gcc for each partition, is independent from other partitions, so this part should be easily distributable on multiple hosts or multiple cores for big graphs.

With the used non overlapping partitioning schemes, the maximal precision of our heuristic is limited, since the minimum partition count is 2. The previous approximative approach can provide a higher precision when runtime is not so important. This disadvantage of our heuristic could be removed with overlapping partitioning schemes.

The main contribution, besides offering a heuristic for a fast approximation of the gcc, is the power-based aggregation scheme and the determination of its exponent $\frac{2}{3}$. This aggregation scheme relates the gcc of a subgraph to the gcc of the complete graph. As far as we know, this relation was not previously described.

8 Future work

In this chapter, we will take a look at possible future work based on this thesis.

8.1 Selective calculation of the partition based gcc

In our current heuristic, we approximate the gcc of the complexe graph by considering the gccs of all partitions. Since the calculation of the absolute gcc on each partition is very time-consuming, it should be tried to only calculate and consider the gcc for a subset of partitions. This could be especially interesting when using community-based partitioning schemes, if we suggest that similar communities inside a graph have a similar gcc.

8.2 Further partitioning schemes

We only used non overlapping, complete partitioning schemes. Overlapping and or incomplete partitioning schemes could perform better than our relative simple partitioning schemes. Our non overlapping BFS scheme performed not well, but overlapping, non complete BFS schemes could do better.

As mentioned in Section 8.1, community-based partitioning schemes could provide the possibility to skip the gcc calculation for some partitions.

8.3 Further use of a random selection model

To get a continuously better approximation, the following approach could be tried: As partitioning scheme, a random selection model would be used. In each iteration, a random selection of nodes would form a partition. The gcc for this partition would be calculated, corrected with the power-based aggregation scheme and combined with the previously calculated approximation. In the best case, each iteration would provide a better approximation of the absolute gcc.

This would be a monte carlo like algorithm, which could be executed until the maximal wanted runtime is reached or the approximated gcc difference between two iterations is sufficient small enough.

8.4 Investigate the constant exponent α

Our power-based aggregations scheme suggest that the constant $\alpha = \frac{2}{3}$ as exponent for the ratios of maximal possible edges of partitions and graph relates the gcc of a subgraph to the gcc of the complete graph. We did not provide a theory in thesis, why this constant is $\alpha = \frac{2}{3}$.

The factor $\frac{1}{3}$ appears in [1], this could be an initial point for the research.

9 Appendices

9.1 Used graphs

Name	# Nodes	# Edges	GCC	Description
LSCC-6-analyze-buddy-aug-sept-2011	7030	84850	0.19563	Largest strongly connected component of a crawl of the "Studentenportal Ilmemnau" http://spi.tu-ilmenau.de from September 2011
2005-02-25.wot.gtna	25487	303258	0.32628	Graph of the OpenPGP Web of trust from February 2005 (http://www.lysator.liu.se/~jc/wotsap/wots2/)
BarabasiAlbert-5000	250	20812	0.40168	Generated by GTNA class gtna.networks.model.BarabasiAlbert
BarabasiAlbert-20000	2000	195778	0.11003	Generated by GTNA class gtna.networks.model.BarabasiAlbert
Gilbert	1500	426842	0,18982	Generated by GTNA class gtna.networks.model.Gilbert
Kleinberg1D1	50000	1104202	0.08099	Generated by GTNA class gtna.networks.model.Kleinberg
DeBruijn	3375	101010	0.008336	Generated by GTNA class gtna.networks.model.DeBruijn
LWCC-BI-6-analyze-buddy-aug-sept-2011	10380	105058	0.18095	Largest weakly connected component of a crawl of the "Studentenportal Ilmemnau" http://spi.tu-ilmenau.de from September 2011
PowerLawRandomGraph-13	5000	617368	0.17113	Generated by GTNA class gtna.networks.model.PowerLawRandomGraph

9.2 Further performance plots

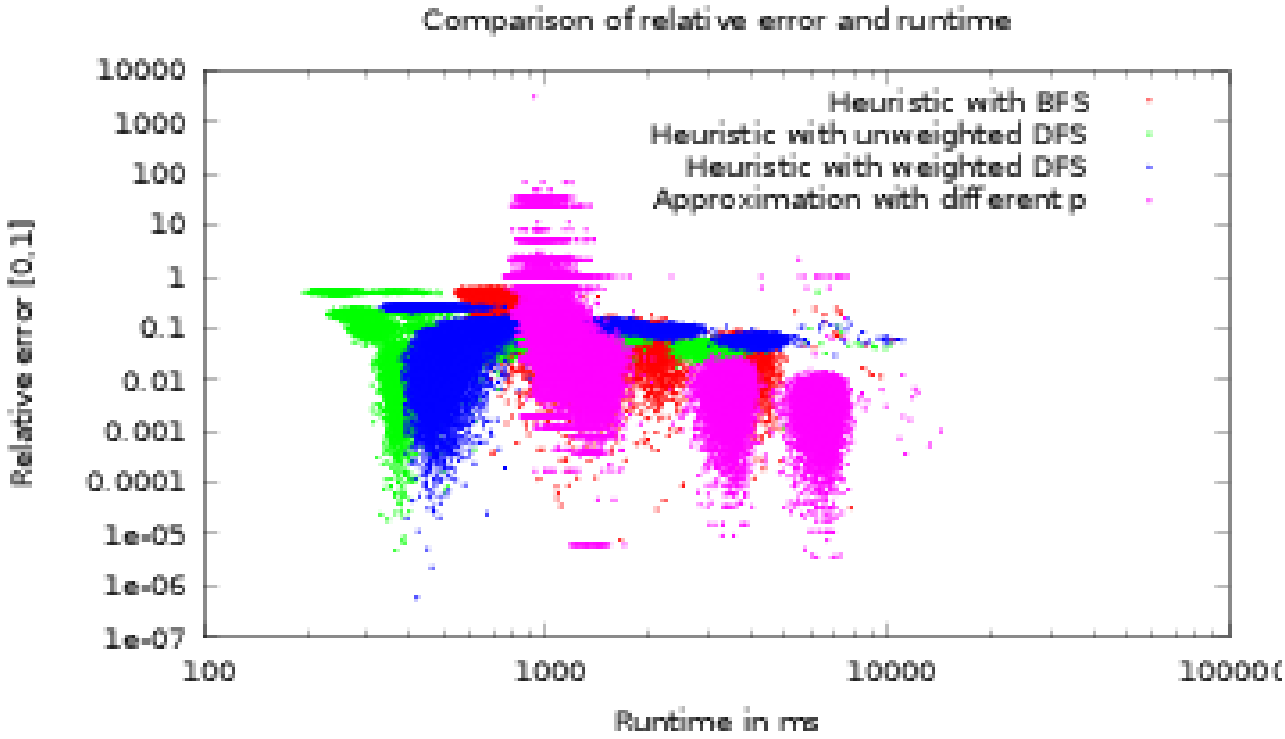

Figure 9.1: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on Kleinberg1D1

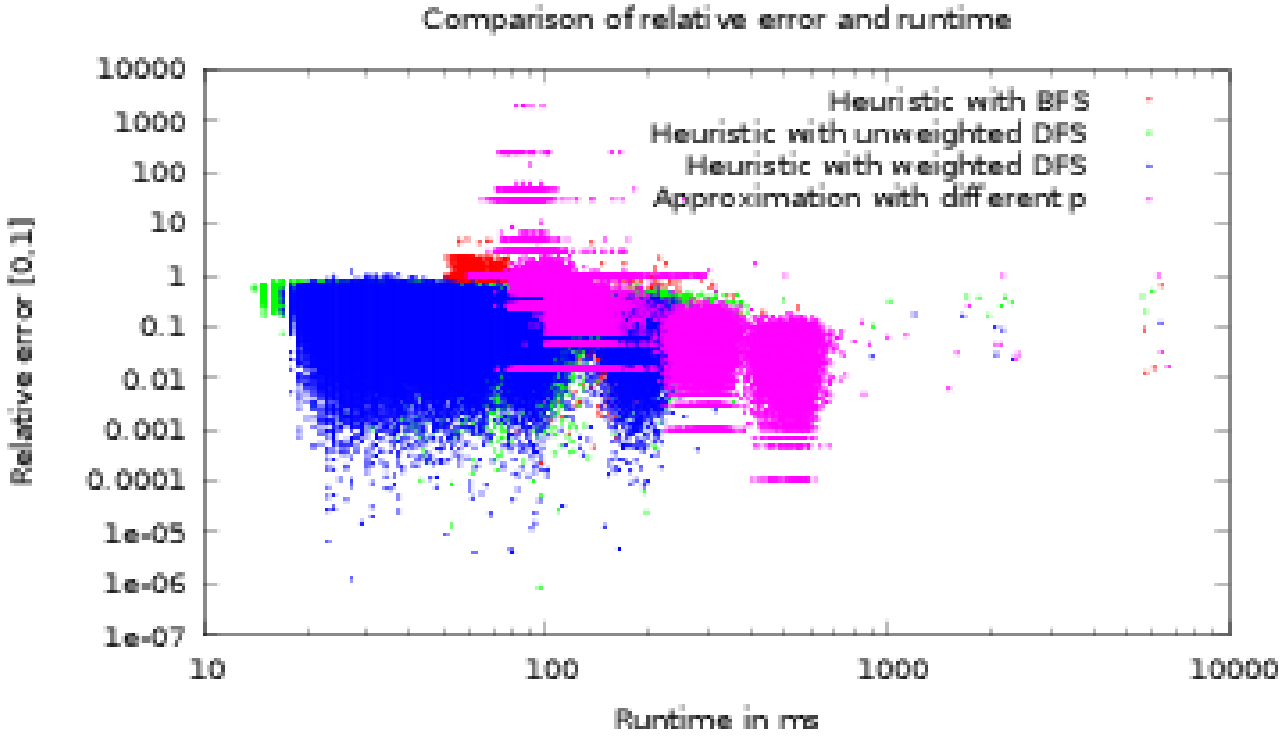

Figure 9.2: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on DeBruijn

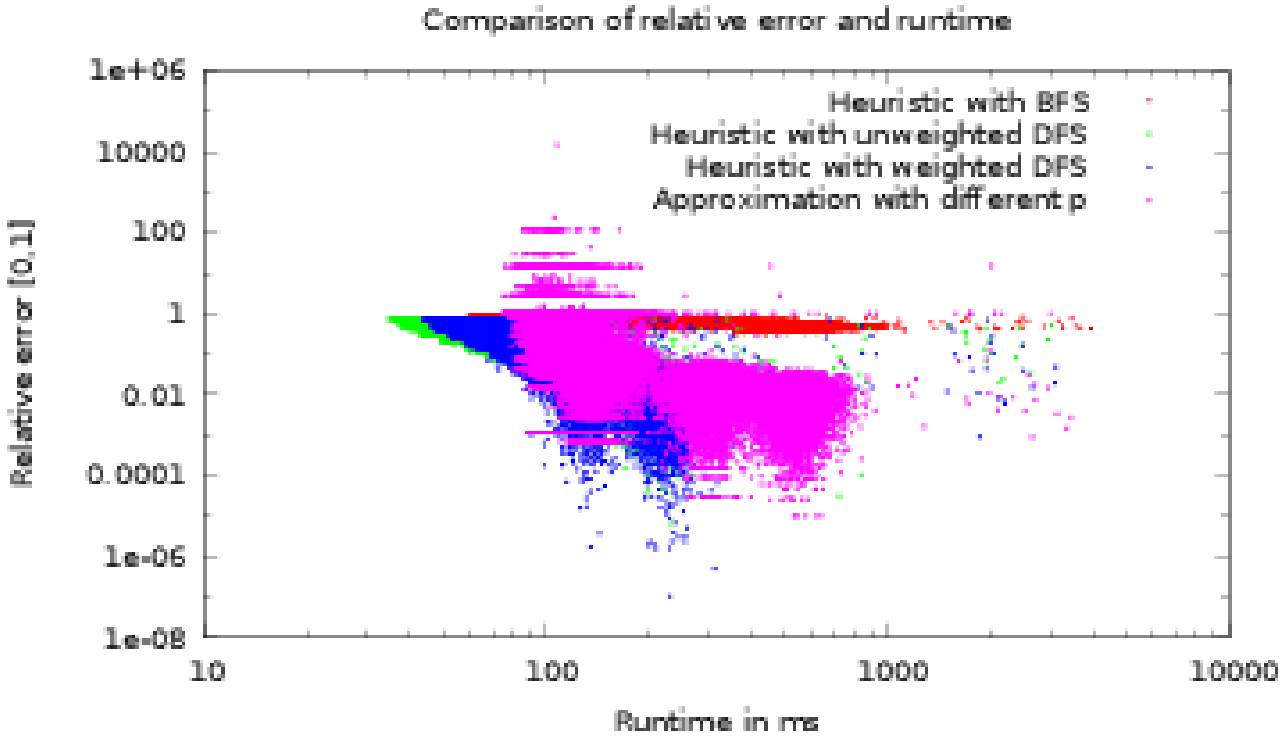

Figure 9.3: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on LWCC-BI-6_analyze_buddy_aug_sept_2011

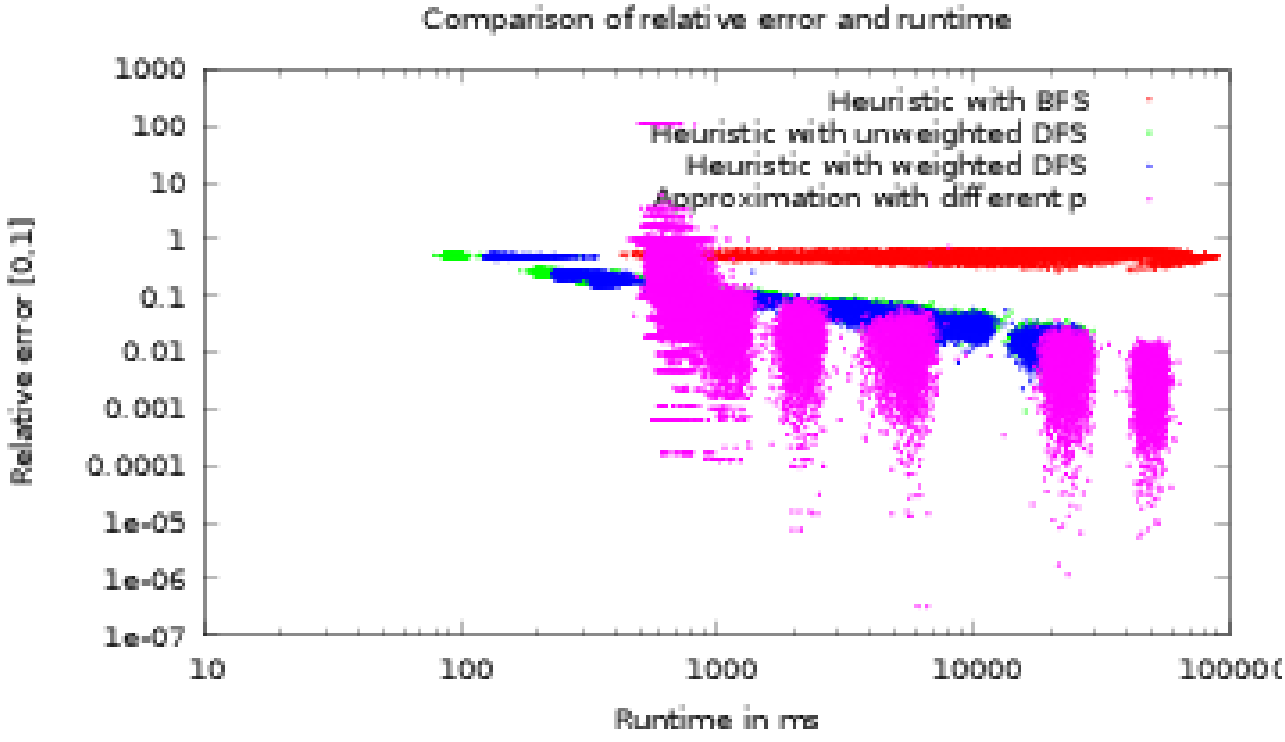

Figure 9.4: Comparison of runtime and precision for our heuristic with unweighted DFS, weighted DFS and BFS partitioning schemes and the previous approximation approach on PowerLawRandomGraph_13

Bibliography

[1] Charalampos E. Tsourakakis, Mihail N. Kolountzakis, and Gary L. Miller: “Approximate Triangle Counting”, 2009.

[2] “Practical algorithms for triangle computations in very large (sparse (power-law)) graphs” from Matthieu Latapy, 2008.

[3] Charalampos E. Tsourakakis: “Fast Counting of Triangles in Large Real Networks: Algorithms and Laws”

[4] B. M. Tabak, M. Takami, J. M. C. Rocha and D. O. Cajueiro: “Directed Clustering Coefficient as a Measure of Systemic Risk in Complex Banking Networks”