Bachelor Thesis

Graph sampling for subgraph counting on directed graphs

Noah-Yannick Schmid

Course of Study: Informatik
Examiner: Prof. Dr. Kurt Rothermel
Supervisor: Michael Schramm, M.Sc.
Commenced: April 27, 2021
Completed: October 27, 2021
Abstract

Calculating the count of subgraphs in a given network is a vital graph mining primitive with multidisciplinary applicability. Many analytical methodologies, such as discovering network motifs, rely precisely on the subgraph census problem. Further, an accurate census is often unnecessary to gain the required information, and approximating the census can significantly benefit the runtime. Elenberg et al. proposed a method to approximate the census on undirected graphs for 3- and 4-profiles through sub-sampling the edges of the input network. The goal of this thesis is to generalize this method to directed networks with variable subgraph sizes. Thus providing an algorithm that sub-samples the input network, transfers it to an existing census algorithm and estimates the census from the gained result, which is then called SampleCensus. The algorithm is then tested with a set of symbolic networks, comparing the runtime and accuracy with the present algorithms, Rand-FASE and FASE. Additionally, we give an implementation using the distributed algorithm MR-GTrie and also analyze further runtime enhancements. With that, we show how this approach can contribute to a more efficient census estimation.
# Contents

1 Introduction 13

2 Preliminaries and problem statement 15  
   2.1 Graph Terminology ........................................... 15  
   2.2 Problem statement ............................................ 16

3 Related Work 17  
   3.1 Related Problems .............................................. 17  
   3.2 Related work .................................................. 18

4 Census Estimation 21  
   4.1 Structure of the Algorithm .................................. 21  
   4.2 Estimate the census ........................................... 21

5 Exact Approaches 27  
   5.1 FaSE - Fast Subgraph Enumeration ............................. 27  
   5.2 MR-GTries ....................................................... 31

6 Experimental Evaluation 35  
   6.1 Additional runtime ............................................ 36  
   6.2 Comparison of SAMPLEFaSE .................................. 37  
   6.3 Comparison of SAMPLEMR-GTries ............................. 43

7 Conclusion 47  
   7.1 Future Work ................................................... 47

Bibliography 49
List of Figures

1.1 Induced occurrences of two subgraphs, together with its frequencies, in a larger network. By Eddin and Ribeiro [ER17] .................................................. 13

3.1 An example motif of size 3. Random networks preserve the degree of each vertex in the original network. The Feed Forward Loop appears at most once in the random networks. However it appears three times in the original network. By Ribeiro and Silva [RS14] .................................................. 18

4.1 Edge sampling transitions with probabilities for some selected graphs of size 3. By Elenberg et al. [ESBD15] .................................................. 22

5.1 An example of a g-trie containing six directed graphs of size 3. The black vertices are the ones added to the graph. .................................................. 28

5.2 Example randomized search tree created by RAND-FaSE. Filled rectangles represent skipped enumeration steps. Each recursion step $i$ is done with the probability $p_i$. Figure by Ribeiro et al. [RPS+19] .................................................. 30

5.3 Shows how the symmetry breaking of FaSE does not work with a g-trie created by MR-GTRIES. No occurrence of the searched graph can be enumerated. ............... 32

5.4 Automorphisms of a sample graph and the resulting symmetry breaking conditions. Shows how the conditions change the result of the enumeration. ................... 33

6.1 Runtime of census computation for subgraphs of size 3 on different networks relative to the runtime for the exact computation. .................................. 38

6.2 Runtime of census computation for different subgraph sizes on the foldoc network relative to the runtime of the exact census. ........................................ 39

6.3 Average relative error of the estimated census for subgraphs of size 3 ............... 40

6.4 Median of relative error of the estimated census for subgraphs of size 3 ........... 40

6.5 Median of the relative error concerning the edge-count of each network. Note that the scaling on the graphics is different. ........................................... 41

6.6 Average error of subgraphs on the tested networks in relation to their frequency. 42

6.7 Average relative error of the estimated census on the foldoc network for different subgraph sizes. .................................................. 42

6.8 Proportion of subgraphs whose census gets estimated exactly. The census is calculated for subgraphs of size three. ........................................... 43

6.9 Spearman and Pearson correlation coefficients for correlation between the exact count and the estimation. Results are shown for different sample rates for a 3-subgraph census. .................................................. 44

6.10 Median of the relative error of the estimated frequencies about the speedup gained towards the exact calculation. ................................. 44
6.11 Runtime of MR-GTries with different amounts of workers used, relative to the runtime with one worker. The horizontal lines represent the relative runtime using FaSE. 45

6.12 Runtime of different setups using various sample rates, relative to the runtime of the same setup performing the exact census. 46

6.13 Runtime in relation to the median of the relative errors using both algorithms. Runtime is displayed relative to the runtime of FaSE calculating the exact census. 46
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Topological features of the used networks</td>
<td>36</td>
</tr>
<tr>
<td>6.2</td>
<td>Time it takes to sample the edges from the input network in seconds.</td>
<td>37</td>
</tr>
<tr>
<td>6.3</td>
<td>Time it takes to calculate the transition probabilities for different subgraph sizes.</td>
<td>37</td>
</tr>
<tr>
<td>6.4</td>
<td>Memory usage of SAMPLEFASE with different sample rates and decrease towards the exact FASE algorithm.</td>
<td>39</td>
</tr>
<tr>
<td>6.5</td>
<td>Correlation Coefficients for correlation between network size and the median of relative error.</td>
<td>41</td>
</tr>
</tbody>
</table>
## List of Algorithms

4.1 Calculating the transition probabilities $p_{x\rightarrow y}$ ........................................... 23  
4.2 Calculating the census for a $k$-profile with edge-sampling ................................. 24  
5.1 The FaSE Census algorithm .................................................................................. 29  
5.2 The MR-GTRIES distributed census algorithm ...................................................... 34
1 Introduction

Complex networks can represent a large variety of natural and artificial systems [AB01][New03]. The science around that topic, known as network science which is about finding exciting features on graphs, has emerged as a crucial multidisciplinary field in recent years [FRTB07]. A cause for that is that technological advances produced a large amount of data represented as graphs [Bar16]. Different techniques can be chosen to analyze such networks [FRTB07], varying from inspecting features on vertex-level up to global metrics such as graph diameter. This thesis will focus on an intermediate approach to discover small patterns in the graph and classify their appearance. Subgraphs then represent these patterns, and they form the basis for complex networks revealing their design principles [MSI+02].

One crucial related problem is computing the frequencies of subgraphs in the analyzed network. This problem is also known as performing a subgraph census. Later we will also formalize this problem and give a more detailed description. Figure 1.1 illustrates the concept by showing the frequency of some subgraphs of size 3 in another network.

The subgraph census has many applications. For example, a network motif is defined as a statistically significant subgraph [MSI+02], which suggests that its frequency in the input graph is much higher than in comparable random ones. Nevertheless, to compute the motif, we demand the ability to count subgraphs. Another example is the observation of graphs based on the graphlet degree distribution [Prž07], which also implies counting and storing the frequencies of some set of small subgraphs.

However, computing the subgraph census is a computationally hard task. It is closely related to the classic subgraph isomorphism which is known to be NP-Complete [Coo71]. Furthermore, just knowing if a given subgraph appears in a network is NP-Complete. Counting those occurrences is an even more complex task. Following that, the computation time of the subgraph census grows exponentially while increasing the network size or the size of the subgraphs being searched.

![Figure 1.1: Induced occurrences of two subgraphs, together with its frequencies, in a larger network. By Eddin and Ribeiro [ER17]](image-url)
1 Introduction

Thus, the applicability is bounded to relatively small subgraph sizes and to not too large networks. Decreasing the runtime would effectively push those limitations and allow for counting the subgraph frequencies of larger subgraphs and more significant networks. Even increasing the size of subgraphs being searched only by one already would give a new insight into a network because it can exhibit previously unknown patterns [PR15].

Previous approaches significantly diminished runtime in two ways.

- **Parallelism**: Due to increasingly large clusters for computing being available. It makes sense to distribute computation over several workers and therefore decrease runtime significantly.

- **Approximate counting**: Usually, the exact result of a census is not obliged, e.g. computing network motifs can still be done without knowing the exact frequencies of the contained subgraphs [KIMA04b]. Therefore, the approach is to trade runtime against accuracy and thus, increase the limit of maximum subgraph sizes.

In this thesis, the focus will mainly be on the reduction of runtime using approximate counting. We will also present a way of combining both approaches and thus decrease runtime even further. To estimate the subgraph census, the edges of the input network are sub-sampled. This means that a part of the edges gets removed from the network before performing the census. By doing so, different subgraphs transition into new ones, and the actual result must be estimated through solving a linear equation system. Elenberg et al. gave the base for that by presenting a way to estimate 3- and 4-profiles on undirected graphs by sampling the edges of the input network [ESBD15][ESBD16]. This thesis gives a description of an algorithm called SAMPLECENSUS, together with an implementation in C++. SAMPLECENSUS generalizes the approach of Elenberg et al. for arbitrary subgraph sizes and to be applicable for directed graphs. This approach relies on another census algorithm, that is used to perform the census on the sampled graph. We will present two modern algorithms which were used to implement SAMPLECENSUS and assess the performance and accuracy of the implementations.

The first one, FaSE by Paredes and Ribeiro [PR13], is a census algorithm based on recursively enumerating all subgraph occurrences and classifying them using a data structure called g-trie. The second one MR-GTries by Eddin and Ribeiro [ER17], is a parallel approach using Map-Reduce as a foundation. Because this algorithm is not publicly available, We will also provide a custom implementation for this algorithm, utilizing Java and the Apache Spark framework [ZXW+16]. Both algorithms rely on the g-trie data structure, which is technically a prefix tree for graphs [RS14], so this data structure will additionally be explained further.

The rest of the thesis will be structured as follows. Chapter 2 introduces the solved problem and gives the graph terminology used throughout the thesis. Chapter 3 presents some work previously done on that subject and related problem statements. Chapter 4 describes the estimation through edge-sampling while Chapter 5 will cover the used the exact census algorithms. Chapter 6 addresses the results of the experiments that were done. Those will be concluded in Chapter 7. Chapter 7 will also give a quick view of what can be done to extend this work.
2 Preliminaries and problem statement

This chapter introduces the common graph terminology that will be used throughout the thesis. Furthermore, the problem this thesis tackles will be explained and formally defined.

2.1 Graph Terminology

A graph $G = (V, E)$, also called network, is defined as a 2-tuple consisting of a finite set of vertices $V$ and a finite set of edges $E$. Those are often also written as $V_G, E_G$ for arbitrary graphs $G$. The size of a graph is given by its number of vertices $|V|$. Given the size, we can also define a $k$-graph, which is a graph of size $k$. Moreover, a $k$-profile is a set containing all graphs of size $k$. By calling a $k$-profile connected, we mean all connected graphs of size $k$. Connected in this context means, that for all vertices $v \in V$ of a graph, there is a path to all other vertices in the graph. This path can ignore the direction of the edges. In a directed graph, edges are defined as 2-tuples $(a, b)$ that represent a directed edge from vertex $a$ to vertex $b$. In undirected graphs, however, edges don’t have a direction and therefore two tuples, one for each direction, are saved. If the graph has no self-loops or multiple edges connecting the same pair of vertices, it is considered a simple graph. In the context of this thesis, we are only dealing with simple graphs, and we assume that all vertices in a graph are represented by consecutive integers starting from 0. Given this assumption, we can compare vertices like integers. This assumption should not be confused with labelled graphs, where each vertex is assigned a label, and two graphs are only isomorphic if also their labels match.

The degree of a vertex is defined by the number of edges it has to other vertices. Thereby, it is not relevant whether these edges point to or point away from that vertex. When accounting the directions however, in- and out-degrees can be defined by the amount of edges pointing to or away from a vertex. Further, the neighbourhood $N(n)$ of a vertex $n$ is defined as all vertices which are connected to. Or more formally $N(n) = \{ m \mid (m, n) \in E \lor (n, m) \in E \}$.

A subgraph $G_k = (V_k, E_k)$ of a network $G = (V, E)$, also denoted as $G_k \subset G$, is a $k$-graph in which $V_k \subseteq V$ and $E_k \subseteq E$ holds true. We can further define the subgraph to be induced if $\forall u, v \in V_k : (u, v) \in E_k \iff (u, v) \in E$ also holds true. In this thesis, the main focus will be on induced subgraphs. Similar to vertices, this subgraph can also have a neighbourhood $N(G_k)$ which is the union over all neighbourhoods of the vertices in $G_k$, i.e. $N(G_k) = \bigcup_{v \in V_k} N(v)$. This definition allows us to also define the exclusive neighbourhood of a vertex $v$ in $G_k, N_{exc}(G_k, v)$ as all vertices that are in $N(v)$ but not a neighbour of any other vertex in $G_k$.

Two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ are considered isomorphic if there exists an isomorphism $\varphi : V_G \rightarrow V_H$ such that for all vertices $u, v \in V_G$ in $G$ there exists an edge $(\varphi(u), \varphi(v))$ in $H$, if and only if there exists an edge $(u, v)$ in $G$. For labelled graphs, we additionally demand that the label of any vertex $v \in V_G$ is the same as the label of $\varphi(v) \in V_H$. The set of all automorphisms of a graph $G$, i.e. isomorphisms that map $V_G \rightarrow V_G$, is denoted by $Aut(G)$. 

15
Understanding the frequency of a graph $H$ in a broader network $G$, we first have to define a match. A match of $H$ in $G$ is defined as a set of vertices in $V_G$ that induce $H$. More precisely, a subgraph $G_k \subseteq G$ that is isomorphic to $H$ is called a match. With that, the frequency of $H$ is given as the number of different matches of $H$ in $G$. For the thesis, we are defining two matches different when they consist of a vertex that is not part of the other match.

### 2.2 Problem statement

Given the previous concepts and terminology, we will present a formal definition of the problem we are solving in this thesis.

**Definition 2.2.1 (General Subgraph Census Problem)**

Given a set of subgraphs $S_G$ and a network $G$, for each subgraph $a$ in $S_G$, calculate the exact count of all matches of $a$ in $G$. Two matches are considered different if they have at least one vertex that they do not share.

It can be beneficial to note that graphs searched in the census usually have the same size and are connected. Approaches diverge on what subgraphs they count in the end. **Subgraph-centric** methods only pick one subgraph and search for the frequency of it in $G$, **network-centric** methods always search for an entire (connected) $k$-profile, and **set-centric** approaches focus on a set of interesting subgraphs and count those in $G$ [RPS+19]. Also, approaches differ on what they consider different matches in the network $G$. The standard way is to allow overlapping between matches, i.e. they can share vertices and edges [RS14]. Nevertheless, different concepts exist with introducing constraints on what is considered a different match [SS04], e.g. it may not be allowed to share edges or nodes between matches. Different definitions can be helpful in different fields of research. In this thesis, as stated in the previous section, we are focusing on the standard definition. Also, we are not trying to solve the problem defined in 2.2.1 because it states to find the exact frequencies of subgraphs. We want to solve a slight variation of the Problem:

**Definition 2.2.2 (Approximative $k$-Profile Census Problem)**

Given a positive integer $k$ and a network $G$, estimate the frequencies of all connected $k$-subgraphs in $G$.

We aim to solve problem 2.2.2 in significantly less time than existing algorithms solve problem 2.2.1. Furthermore, our approach uses another algorithm that solves 2.2.1 for its computation and therefore can make use of upcoming algorithms that may solve 2.2.1 more efficiently. Doing so is helpful because many applications do not need an accurate census [KIMA04a][RS10a], e.g. discovering network motifs does not require having the precise frequencies of the subgraphs [RPS+19][KIMA04b].
3 Related Work

This chapter covers problems that are closely related to the problem tackled in this thesis. It will also give an overview of the previously done work in this area of research. The focus here lies predominantly on the earlier developed algorithms that try to solve the subgraph census problem.

3.1 Related Problems

Around analysing graphs and further inspecting subgraphs in larger networks, a vast amount of problems arose. One main problem is the problem of subgraph isomorphism. Given an (extensive) network $G$ and a graph $H$, the problem is to answer whether $H$ has a match in $G$. This problem is known to be NP-Complete for general graphs [Coo71]. Nonetheless, there exist less complex solutions for particular graph types like planar graphs [Epp99]. It is related to the subgraph census in the way that it can be seen as a simple version of counting the subgraph.

Related to subgraph isomorphism is the problem of graph isomorphism [MP14], which is about determining whether two subgraphs are isomorphic. It is related because many approaches rely on enumerating subgraphs and classifying them, i.e. checking whether they are isomorphic to the enumerated graph. A speedy tool, which solves the graph isomorphism problem, and is used in this thesis and among a variety of algorithms, is Nauty by McKay and Piperno [MP14].

Instead of searching for multiple subgraphs and their count, the problem of subgraph search takes a database of extensive networks and filters the ones out that contain a specific subgraph. Through the problem statement being somewhat different, different approaches and optimizations emerged [YYH04][YM11].

Another previously mentioned problem is the one of motif detection. Motifs are defined as subgraphs statistically being overrepresented in a given network $G$. That means motifs have a significantly higher number of matches in $G$ than in similar networks. Similar networks are those who share the same intrinsic global and local properties with $G$ [RS14]. A popular way of defining a similar network is by keeping the in- and out-degree of the vertices. Figure 3.1 shows an example of a network motif.

A giant pallet of applications has emerged applying this technique. Among them is the analysis of biological networks (e.g. brain [SK04] or protein interactions [YSK+04]), social networks (e.g. online social networks [DT14]) or software networks (e.g. function-call graphs [WWT18]). Discovering motifs is often done by generating a large set of similar networks and comparing the frequency of subgraphs on those with the frequencies on the input network [KIMA04a]. Although methods exist that avoid the generation of many similar networks and thus avoiding calculating a census on those, algorithms must always calculate the census on the original network [RPS+19]. Furthermore, an approximative census is usually acceptable for determining network
motifs [RPS+19]. Thus, improving the runtime of estimating a subgraph census also affects the performance of calculating motifs. The definition of motifs also gives origin to similar problems like colored motifs [LFS06] or looking for underrepresented subgraphs, i.e. anti-motifs [MIK+04]

### 3.2 Related work

Now that we have seen many related problems, we want to focus on work related to the subgraph census. There has been a large amount of work on efficiently counting subgraph frequencies in the past. Some aim to spread the problem over several machines to increase performance, while others try to only approximate the census, and by doing so, decrease runtime significantly.

The first practical algorithm aiming to solve the problem was Mfinder by Milo et al. [MSI+02]. In 2005 with ESU by Wernicke [Wer05], an algorithm was found that can avoid symmetries, accordingly avoid redundant computation and increase runtime significantly. The approach was to enumerate all subgraphs and then categorize them using a graph isomorphism tool like Nauty [MP14]. ESU laid the foundation for future enumeration approach algorithms.

The concept of g-tries [RS10b] vastly reduced runtime. It is a data structure that can be used to classify partial subgraphs while enumerating them. This data structure allows to preserve the topology in the recursion and saves much categorizing. Later new labelling techniques for g-tries were proposed [RS14], and many more enumeration based algorithms used them. The FaSE algorithm by Paredes and Ribeiro [PR13] also uses g-tries as a basis. However, it proposes a different use, where the g-trie is created at runtime. Besides, it uses a fast labelling technique that also allows isomorphic duplicates in the g-trie.

QuateXelero by Khakabimamaghani et al. [KSD+13] also proposes a tree data structure to speed up the process of classifying the enumerated subgraphs. Both proposed tree structures however have a mayor problem: A lot of memory is used, especially when counting larger subgraphs. The
3.2 Related work

Patcomp algorithm by Jain et al. [Jai+17] aims to solve that problem by proposing the Patricia tree, which saves memory by avoiding redundancy. Patcomp trades that memory efficiency against a slight increase in runtime but is therefore suitable on machines with less memory.

Enumeration based approaches often came with an approximative version of them. Those implemented a random enumeration strategy. This strategy skips random parts of the search tree of the enumeration and estimates the original census based on the obtained result. Namely, Rand-ESU by Wernicke [Wer05] was the first one to implement that approach followed by Rand-GTrie [RS10a] and Rand-FaSE [PR15] improving the implementation the same way they did in the exact version. Our proposed algorithm, which follows another principle, is later compared to those approximative algorithms.

First Tsourakakis et al. [TKMF09], and later Elenberg et al. [ESBD15] [ESBD16] took a different approach on estimating the frequencies of subgraphs. The idea was to sample the edges of the input network, calculate the exact census on the sampled graph and estimate the census on the original network with the gathered information. Tsourakakis et al. first implemented this method for triangle counting, and later Elenberg et al. expanded this method to find the census for undirected 3- and 4-profiles. Those papers also inspired this thesis, giving a generalization of their approaches to directed subgraphs of arbitrary sizes.

Elenberg et al. also gave a parallel approach on how to calculate the 3-/4-profiles together with the sampling technique. Nevertheless, from the known enumeration based algorithms, also parallel algorithms emerged. Those can be distinguished as how they synchronize the work of their worker threads. In detail DM-ESU [RSL10b] synchronizes workers using distributed memory, i.e. synchronizing using messages passed through the network. DM-GTries [RSL10a] applies the same platform on the GTries algorithm. Some time later with SM-GTries [ARS14] and SM-FaSE [APR14] the enumeration based algorithms were implemented to use shared memory. Using shared memory environments restricts the algorithms to only run on a single machine. However, machines with dozens of cores exist nowadays, and those algorithms almost give a linear speedup. Thus, those algorithms gave a meaningful contribution to solving the problem. One very new algorithm applies the paradigm of Map-Reduce, which is often found in distributed algorithms [Boy08]. MR-GTries by Eddin and Ribeiro [ER17] can be run on several machines and minimizes network traffic between worker nodes. Which is often the bottleneck of algorithms with distributed memory [RPS+19]. For the sake of testing our method, we provide an implementation of MR-GTries using Apache Spark [ZXW+16] and Java.
4 Census Estimation

This chapter will describe the algorithm based on sampling the edges of the input network. The objective of this algorithm is to estimate the census of a $k$-profile on a large input graph $G$. The general algorithm is accordingly called SAMPLECENSUS. Its core idea is to remove edges from the input network, feed it into an existing census algorithm and estimate the census from the obtained result. The main advantage is that this concept operates with any census algorithm. Thus adapting it to future procedures can be done without any effort.

4.1 Structure of the Algorithm

First, we are going to discuss the general structure of the algorithm. SAMPLECENSUS consists of three main steps:

1. Sample the edges from the input network.
2. Run a census algorithm on the sampled network.
3. Estimate the subgraph frequencies on the original network from the obtained result.

This approach is a generalization of what was done by Elenberg et al. Consider the input network $G = (V, E)$. In the first step of the algorithm, we exclude every edge from $E$ with a probability of $1 - p$. So the probability of an edge staying in the network is $p$. We call this probability $p$ the sampling rate of the algorithm. This edge-sampled network is then given to a census algorithm that exactly counts the frequencies of a given size. The central part of our contribution proceeds by estimating the frequencies on the initial network given the solution on the sampled network.

4.2 Estimate the census

To understand how to estimate the census on the original network, we have to look at what happens to subgraphs when we sample the network’s edges, e.g. a directed 3-clique would be preserved if no edge is removed, which would happen with a probability of $p^6$. Otherwise, it transitions to another subgraph with fewer edges. Figure 4.1 illustrates what happens to graphs on sampling with some graphs of size 3.

Now let $p_{x \rightarrow y}$ be the probability of subgraph $x$ transitioning to subgraph $y$ on sampling. Then the expected frequency $m_y$ of a subgraph $y$ on the sampled network given the frequencies $n_x$ on the original network would be

$$m_y = \sum_x p_{x \rightarrow y} \cdot n_x$$  \hspace{1cm} (4.1)
Figure 4.1: Edge sampling transitions with probabilities for some selected graphs of size 3. By Elenberg et al. [ESBD15]

When considering a $k$-profile with the graphs $\{x_0, \ldots, x_i\}$, in expectation, the following linear system gives the estimation for the frequencies $(n_0, n_1, \ldots, n_i)$ depending on the frequencies $(m_0, \ldots, m_i)$ on the sampled network:

$$
\begin{bmatrix}
\mathbb{E}(m_0) \\
\mathbb{E}(m_1) \\
\vdots \\
\mathbb{E}(m_i)
\end{bmatrix}
= 
\begin{bmatrix}
p_{x_0 \rightarrow x_0} & p_{x_1 \rightarrow x_0} & \cdots & p_{x_i \rightarrow x_0} \\
p_{x_0 \rightarrow x_i} & p_{x_1 \rightarrow x_i} & \cdots & p_{x_i \rightarrow x_i} \\
\vdots & \vdots & \ddots & \vdots \\
p_{x_0 \rightarrow x_i} & p_{x_1 \rightarrow x_i} & \cdots & p_{x_i \rightarrow x_i}
\end{bmatrix}
\begin{bmatrix}
m_0 \\
n_1 \\
\vdots \\
n_i
\end{bmatrix}
$$

Elenberg et al. [ESBD15] calculated the values of $p_{x \rightarrow y}$ by hand and hardcoded them into the program depending on the value of $p$. One of the main contributions from us however is to provide an algorithm that calculates the $p_{x \rightarrow y}$ values dynamically.

4.2.1 Calculate transition probabilities

Consider a labelled graph $H = (E_H, V_H)$ with $k$ vertices and $i$ edges. Then this graph only transitions into graphs with equal or fewer edges. Further, the chance of $H$ transitioning into a labelled graph $U$ where $j$ edges are removed is

$$p_{H \rightarrow U} = (p - 1)^j \cdot p^{i-j}$$

However, we are not interested in labelled graphs. So we can add together all probabilities from graphs that are isomorphic to each other when the labels are removed. Doing so then leads to the concept shown in algorithm 4.1.
Algorithm 4.1 Calculating the transition probabilities \( p_{x \rightarrow y} \)

**procedure** \textsc{calcEstimationMatrix}(profile, \( p \))

// profile contains a list of all graphs of size \( k \) in their canonical representation

\[
\text{for all } G = (V, E) \in \text{profile} \text{ do}
\]

\[
\text{for all } H \in \{ H = (V, E_H) \mid E_H \in \mathcal{P}(E) \} \text{ do}
\]

\[
H_{\text{canon}} \leftarrow \text{canonical label of } H
\]

\[
p_{G \rightarrow H_{\text{canon}}} \leftarrow p_{G \rightarrow H_{\text{canon}}} + p^{|E_H|}(1 - p)^{|E_{H} \setminus E_h|}
\]

end for

end for

\[
\text{return } (p_{x \rightarrow y})_{x, y \in \text{profile}}
\]

end procedure

Improving calculation

Inspecting the values of \( p_{H \rightarrow U} \), we observed that they have the structure:

\[
p_{H \rightarrow U} = t \cdot p^{|E_U|}(1 - p)^{|E_{H} \setminus E_U|}
\]

For some natural number \( t \in \mathbb{N} \), and \( E_H, E_U \) being the set of edges in \( H, U \). For every sampling rate \( p \) the value of \( t \) stays the same. At the same time, calculating \( t \) is the actual expensive part of the calculation because it requires iterating through the power set \( \mathcal{P}(E) \) of every graph in the profile, which by definition has exponential size. The other values \( p, |E_U|, |E_{H} \setminus E_U| \) can be calculated in constant time given the graphs \( H \) and \( U \). So the computation can be optimized by calculating the \( t \)'s of the \( p_{H \rightarrow U} \) only once and preserving it for future computations. With that, in every computation \( p^{|E_U|}(1 - p)^{|E_{H} \setminus E_U|} \) can be computed and multiplied with the appropriate \( t \).

Final algorithm

To finish the algorithm, we have to solve the linear equation system, which consists of the calculated probabilities \( p_{x \rightarrow y} \). That can be done through a standard technique like Gauss elimination. However, Gauss elimination has a complexity of \( O(n^3) \) over the size of the \( k \)-profile. Thus, improving this could give a significant speedup for the algorithm, especially for larger \( k \).

**Lemma 4.2.1**

For arbitrary subgraphs \( x, y \) of size \( k \), the following holds true:

\[
p_{x \rightarrow y} \neq 0 \implies x \equiv y \text{ or } |E_x| > |E_y|
\]

**Proof:** Let \( x \neq y \) with \( |E_x| \leq |E_y| \) two arbitrary subgraphs of size \( k \) in their canonical form. So it is further clear that \( x \neq y \) holds true. If \( |E_x| < |E_y| \) it is clear that \( p_{x \rightarrow y} \) must be 0. If \( |E_x| = |E_y| \), given \( x \neq y \), it follows that \( E_x \neq E_y \) and further there exists an edge \( e_y \in E_y \) that is not in \( E_x \). Because on sampling, a subgraph can only loose edges, \( x \) cannot transition into \( y \) and further \( p_{x \rightarrow y} = 0 \)

Applying Lemma 4.2.1 the linear equation system can be solved in quadratic time over the size of the \( k \)-profile by substituting the results of equations from graphs with more edges into those of graphs with fewer edges. This method is then also used in the final implementation. So we receive the algorithm 4.2 to estimate the census with edge-sampling.
Algorithm 4.2 Calculating the census for a $k$-profile with edge-sampling

procedure SAMPLECENSUS(CENSUSALG, $k$, $p$, Network)

// CENSUSALG is a procedure, that calculates the exact frequencies of given subgraphs in a network
 profile ← GETALLGRAPHSOFSIZE($k$)
estMatrix ← CALCULATIONMATRIX(profile, $p$)
sampledNetwork ← SAMPLENETWORK(Network)
sampledResult ← CENSUSALG(sampledNetwork, profile)
result ← SOLVELINEAREQUATION(estMatrix, sampledResult)
return result
end procedure

The algorithm 4.2 still relies on another algorithm that calculates the exact census. Since most approaches can only calculate the census for connected subgraphs, we have to get more into detail on that issue. Our linear equation system contains all subgraphs, including unconnected ones. Therefore solving the linear equation system without the census for unconnected subgraphs is not possible without additional work.

Theorem 4.2.2
The result of a connected subgraph’s estimated frequency does not depend on the frequency of any unconnected subgraph on the sampled network.

Proof: We proof that the estimated frequency $n_i$ of a graph $x_i$ only depends on the estimated frequencies of graphs $x_j$ where $p_{x_j \rightarrow x_i} \neq 0$. Note that if $x_i$ is connected, there exists no unconnected subgraph $x_j$ such that this equation holds true. Because of Lemma 4.2.1 the edge-count of $x_j$ is always higher than the one of $x_i$. So we are proving it through induction over the edge-count.

Induction base: Let $x_i$ be a fully connected graph, then only $p_{x_i \rightarrow x_i} \neq 0$ so calculating $n_i$ can be done using equation 4.1:

$$n_i = m_i \cdot \frac{1}{p_{x_i \rightarrow x_i}}$$

Induction step: We want to calculate the frequency of a connected graph $x_i$. Assume that we know all frequencies $n_j$ of graphs $x_j \neq x_i$ where $p_{x_j \rightarrow x_i} \neq 0$. All those have more edges than $x_i$ and are connected. Then the frequency $n_i$ can be calculated with the equation 4.1, giving us:

$$m_i = \sum_j (p_{x_j \rightarrow x_i} \cdot n_j) + p_{x_i \rightarrow x_i} \cdot n_i$$

$$\iff m_i - \sum_j (p_{x_j \rightarrow x_i} \cdot n_j) = p_{x_i \rightarrow x_i} \cdot n_i$$

$$\iff \left( m_i - \sum_j (p_{x_j \rightarrow x_i} \cdot n_j) \right) \cdot \frac{1}{p_{x_i \rightarrow x_i}} = n_i$$

Because all values of the left side are known, we proved, that the frequency of a subgraph $x_i$ can be estimated by only knowing the estimations for subgraphs $x_j$ with $p_{x_j \rightarrow x_i} \neq 0$. Moreover, the frequencies of connected subgraphs can be estimated without knowing the estimations of unconnected subgraphs. □
Given Theorem 4.2.2, unconnected subgraphs can be left out of the linear equation system. Furthermore, we can use census algorithms that only calculate the frequencies for connected subgraphs for the execution of SAMPLECENSUS.
5 Exact Approaches

Now the two algorithms used as the base for the edge-sampling approach are presented. Both algorithms are based in the field of enumeration and make use of a data structure called g-trie, which we will also explain. Though SAMPLECENSUS can be easily used with every census algorithm as shown in the previous chapter, those two algorithms are known to be among the best algorithms currently available [RPS+19].

5.1 FaSE - Fast Subgraph Enumeration

One more or less new census algorithm that we used is FaSE by Paredes and Ribeiro [PR13]. FaSE is an enumeration based exact census algorithm that gives the frequencies for directed and connected graphs of a specified k-profile. This section will cover how the algorithm works to understand better how enumeration-based census algorithms work. It will briefly introduce g-tries, on which the FaSE algorithm depends. Moreover, it introduces the RAND-FaSE algorithm, which is a slight variation of FaSE, that calculates the approximative census. We will later also compare our edge-sampling approach against this approximative algorithm.

The general idea of the algorithm consists of two main parts that are tightly integrated: enumeration and encapsulation. Every subgraph occurrence from the input network is formed during the enumeration phase. The algorithm, to accomplish that, incrementally expands a connected set of vertices. The encapsulation part stores the topological information about the subgraphs. It also updates the frequency of the respective class whenever a complete subgraph is enumerated.

5.1.1 G-Tries

The FaSE algorithm is heavily based on g-tries, so we first observe what g-tries are and how they work. A g-tries is essentially a “prefix tree for graphs”. Which is also the origin of its name, which originates from “Graph reTRIEval”. The idea behind g-tries is that they follow the search tree of the enumeration recursion. So it is a tree whose nodes represent abstract models of graphs, as shown in Figure 5.1. In the tree, a parent node represents the prefix of its child nodes. Thus, when stepping down the tree, a vertex with new edges is added to the saved graph. This procedure resembles the execution of the incremental enumeration of the subgraphs. So in every enumeration step, the g-trie can be walked down further while preserving information over the topology. The standard sequence of execution would then be.

- First, insert the empty graph in the g-trie.
5 Exact Approaches

Figure 5.1: An example of a g-trie containing six directed graphs of size 3. The black vertices are the ones added to the graph.

- For each, during the enumeration process added vertex \( v \), calculate a label that resembles how \( v \) is related to the previously added vertices, i.e. what edges are between them. This label then is used to determine the next node in the g-trie. Consequently, in every expansion step in the enumeration, a new g-trie node gets calculated. If the node does not exist in the g-trie, add the node and connect it to the node before the expansion.

This entire process is deterministic. Therefore leading to the characteristic that whenever two graphs \( x, y \) in the enumeration end up in the same g-trie node, \( x \cong y \) are isomorphic. Note also, that the labelling technique heavily influences the structure of the g-trie. In FaSE, a fast labelling technique called LSLLabel is used.

5.1.2 Structure

Next, we want to focus on how the algorithm is structured and uses the g-trie structure. As already mentioned, the algorithm consists of two main parts, enumeration and encapsulation. We will inspect their implementation and how they make use of the g-trie.

Enumeration

To enumerate all subgraphs of a given size \( k \), we choose each vertex \( v_0 \) from the input graph. With that, two sets \( V_S = \{v_0\} \) and \( V_E = N(v_0) \) are initialized. Here \( V_S \) holds the vertices of the currently considered partial subgraph. \( V_E \) holds the vertices to which the subgraph can expand in the next step. The algorithm then recursively removes each element \( u \) from \( V_E \) and changes \( V_S = V_S \cup \{u\} \) and \( V_E = V_E \cup \{v \in N_{exc}(u, V_S) \mid v > v_0\} \). Here, \( N_{exc}(u, V_S) \) means the exclusive neighborhood of \( u \). \( V_E \) cannot simply be enlarged with every neighbour of the graph because that way, subgraphs would be counted multiple times. Double counting would happen because choosing a vertex in both the neighbourhood of \( u \) and the previous \( V_S \) would give us two ways to enumerate the same subgraph.
Algorithm 5.1 The FaSE Census algorithm

procedure FaSE\((G = (E, V), k)\)
    EnumerateAll\((G, k, \emptyset, 0)\)
    for all \(n \in G.Trie.leaves()\) do
        frequency\([\text{CanonicalLabel}(n.graph)]\rangle = n.count
    end for
    return frequency
end procedure

procedure EnumerateAll\((G, K, V_S, d)\)
    if \(d = K\) then
        \(G.Trie.current.count\rangle = 1\)
    else
        while \(nS \leftarrow \text{EnumerateNext}(V_S)\) do
            \(w \leftarrow nS.NextNode()\)
            \(nL \leftarrow LSLabel(S, w)\)
            \(G.Trie.Deepen(nL)\)
            \(nS.Subgraph \leftarrow nS.Subgraph \cup \{w\}\)
            EnumerateAll\((G, K, nS, d + 1)\)
        end while
    end if
end procedure

Encapsulation

Because FaSE expands vertices recursively, it would make sense to store information about the current partial subgraph gradually. That way, deeper in the recursion tree, information gathering can be saved, and we can reuse the stored information. To achieve that, FaSE holds a g-trie node that shares its topology with the currently enumerated (partial) subgraph. For moving through the g-trie, FaSE proposes two methods, Deepen and Jump. Here Deepen requires a label of the current subgraph. From there, it moves the held g-trie node down one level in the g-trie to the node with the same topology. If no node with the received label exists, it adds the node to the g-trie through this procedure. The Jump procedure jumps up one level in the g-trie to again represent the subgraph that is one level above in the enumeration recursion tree. With those two operations, FaSE navigates the g-trie and stores topology information over the partial subgraphs. Whenever a complete subgraph is enumerated, a counter on the g-trie leaf is incremented. The census can then be calculated from the counters in the leaves by bringing the connected labels in canonical form. This is done using nauty and shown as CanonicalLabel in 5.1.

When combining those two main elements, we get the algorithm described in 5.1 The procedure EnumerateNext in 5.1 defines the selection of new vertices as it was described in the Enumeration step.
5 Exact Approaches

Figure 5.2: Example randomized search tree created by RAND-FaSE. Filled rectangles represent skipped enumeration steps. Each recursion step \( i \) is done with the probability \( p_i \). Figure by Ribeiro et al. [RPS+19]

5.1.3 RAND-FaSE

Next, we quickly want to dive into the RAND-FaSE algorithm, which is derived from FaSE, which estimates the census and therefore decreases runtime in trade of accuracy. It is one of the most recent approximative census algorithms capable of running on directed graphs. Later we will also compare this approach in detail with our edge-sampling approach.

Idea

The idea is to slightly adjust the FaSE algorithm to decrease runtime in trade for a slight accuracy loss. Realizing that is done by sampling the search tree. So specific parts of the search tree are removed with a given probability. In detail, we assign a probability \( p_i \) to every level \( i \) of the search tree, and on that level, only move down a branch at this level with the probability \( p_i \). Figure 5.2 shows how a sampled search tree could end up.

Implementation

In the implementation, sampling the search tree is done by checking for the probability each time a new partial subgraph is enumerated. It will only be expanded further with the given probability \( p_i \) where \( i \) is the current handled partial subgraph size.

Implementing it like that ensures that any vertex is chosen with the probability \( p_0 \). Additionally, any partial subgraph of size two will be enumerated with probability \( p_0 \cdot p_1 \). Following this principle leads to choosing a subgraph of size \( k \) with the probability

\[
p_s = \prod_{d=0}^{k} p_d
\]

Further, the frequency \( n_x \) of a subgraph \( x \) can then be estimated using the frequency \( m_x \) calculated with the sampled search-tree using

\[
n_x = \mathbb{E}(m_x) \cdot p_s
\]

This technique of sampling the search tree is known as random enumeration and is also used in many other previous works, including RAND-GTrie [RS10a] and RAND-ESU [Wer05][RPS+19].
5.2 MR-GTries

As mentioned, the edge-sampling approach relies on another census algorithm that calculates the exact frequencies. Another way to speed up this census is to use parallel environments to perform it. Parallelizing gives, depending on the approach, algorithm and environment, massive speedups. The edge-sampling approach can make use algorithms with such a considerable speedup, and therefore decrease runtime even further. Different approaches to parallelize the algorithm and can be divided into four platforms on which they perform.

- **Distributed Memory** refers to a multiprocessor cluster in which every processor has its private memory. Information is shared by messages passed through the network [RPS+19]. This method was the first platform to be used for parallel census algorithms by Wang et al. [WTZ+05].

- **Shared Memory** relies on all executors having one shared memory address space. Therefore such algorithms can usually only run on a single machine. On those, however, algorithms achieve almost linear speedup by the number of cores used. That is because there is no overhead of sending messages through a network [RPS+19].

- **Map-Reduce** is a paradigm in which all workers perform very similar tasks, and results are reduced together in the end [Boy08]. Because then traditionally data only has to be synchronized once, this leads to less network traffic.

- **GPUs** have a large number of parallel threads and therefore seem very appealing for parallelization of the census algorithm. However, linear speedups on the GPU are very rare, which also comes from coarse work-unit granularity [RPS+19]. The combination between CPU cores and GPUs can be used to improve on that.

The MR-GTries algorithm also implements a Map-Reduce approach and further is the currently best in doing so [RPS+19]. To test the speedup on a parallel approach and how sampling the input graph affects its speedup, we implemented a version of the MR-GTries algorithm. This section gives a closer look at the concept of Map Reduce, how the MR-GTries algorithm works and how it is implemented.

5.2.1 Map Reduce algorithm

The Map-Reduce paradigm can be found in many distributed algorithms [RPS+19]. As the name suggests, two main procedures construct such algorithms:

- **MAP** which runs on each worker-node and processes a part of the input data.

- **REDUCE** which is responsible for combining the output of the MAP-procedures to one output.

So when applied to the subgraph census, many sequential algorithms rely on enumerating the subgraphs by incrementally expanding the neighbourhood of a given start vertex. The idea is to distribute the start vertices to the different worker-nodes, mapping them to a partial census, and in the end, all partial results are reduced to the final output of the algorithm.
5 Exact Approaches

MR-GTries is a special implementation of the Map-Reduce paradigm that also features work sharing. The name is derived from the two main features this algorithm contains, implementing parallelization through Map-Reduce and using the g-trie data structure. Work sharing in MR-GTries is implemented through timeouts on all workers after a set time window. The work then is redistributed to all worker nodes, and they are started again.

5.2.2 Symmetry breaking

The FaSE algorithm and MR-GTries both use the g-trie as an efficient data structure to preserve information over the subgraph topology. Nevertheless, in FaSE, the enumeration of the subgraphs dominates the search tree, and it only uses the g-trie to save the gathered information. Whereas in MR-GTries, the structure of the g-trie dominates the search tree, i.e. the recursion iterates through the g-trie nodes, and subgraphs that are isomorphic to a g-trie node are enumerated and expanded. This domination requires the g-trie to be constructed, contrary to FaSE, before executing the main algorithm. Further, the g-trie of MR-GTries should also not contain isomorphic duplicates of a single graph. Therefore, the subgraphs of the k-profile are labelled with a canonical labelling technique, namely Nauty [MP14], and then inserted in the g-trie. In the enumeration step of MR-GTries, only those vertices are expanded, which, together with the already expanded ones, form a subgraph isomorphic to the one currently projected in the g-trie.

Using the g-trie as MR-GTries does raises a problem: subgraphs would be counted multiple times. That can happen because expanding vertices in different orders can still result in the same graph. In FaSE, this problem was resolved by just expanding vertices which index is higher than the first expanded vertex and using the exclusive neighbourhood. The same rules cannot be utilized in MR-GTries because the order of the nodes’ indices in the subgraph may not reflect the order in the g-trie. Figure 5.3 shows an example g-trie with which, given the example network, the explored subgraph cannot be enumerated by applying the rules of FaSE. To prevent counting the same subgraph multiple times, in MR-GTries, some symmetry-breaking conditions have to be calculated to restrict vertices’ expansion order. Algorithmically, this is done by observing the automorphism-set $Aut(H)$ of a subgraph $H = (V_H, E_H)$. $Aut(H)$ consists of all Automorphisms $\varphi : V_H \to V_H$ on $H$. The size of the $Aut(H)$ set also defines in how many ways $H$ can be enumerated. The idea then is to create conditions of the form $a < b$ for two vertices $a$ and $b$ in $H$. 

![Figure 5.3: Shows how the symmetry breaking of FaSE does not work with a g-trie created by MR-GTries. No occurrence of the searched graph can be enumerated.](image)
5.2 MR-GTries

Figure 5.4: Automorphisms of a sample graph and the resulting symmetry breaking conditions. Shows how the conditions change the result of the enumeration.

Those define that the index of \( a \) in the input network must be lower than \( b \). Those rules are added until all automorphisms except the identity inherently break at least one rule. With that, there is only one way to enumerate a subgraph, and multiple counting is out of the question. Figure 5.4 also shows how such conditions could look on a sample graph. As it can be seen, without the conditions, way more subgraphs would be enumerated than there are present in the network.

5.2.3 Parallizing

This last section gives some further information on how the implementation of parallelism in MR-GTries works. As described first, the vertices are initially distributed evenly over all workers. All workers also start with a timeout, after which they return their current state. This timeout initially is set to a small value but adapts during the execution to reduce synchronization overhead. Therefore whenever all workers return with still work to do, the timeout-time is increased by 20%. As soon as a worker returns with no work to do anymore, the timeout time is adjusted by

\[
\text{timeout}_{\text{new}} = \text{timeout}_{\text{old}} - \frac{\text{timeout}_{\text{old}} \cdot \text{numIdleWorkers}}{\text{totalNumWorkers}}
\]

This adjustment minimizes the idle times of workers. Additionally, a particular rule is added for the case when there are less work-units left than there are workers on the cluster. The rule states, that the algorithm distributes the remaining work-units among the workers, and the timeout is set to infinite so that long-running workers, in the end, are not interrupted through repeated timeouts, which would increase runtime. A timeout on a worker always requires the state to be saved to return it afterwards. Therefore the recursion stack must be saved. Every recursion does this on a timeout adding its loop variables to the state. From this state, the enumeration procedure can continue. The MR-GTries algorithm then is implemented using the resilient distributed datasets (RDDs) from Apache Spark [ZXW+16]. It initially creates one RDD with all workers inside. After that, it maps workers to their \( BC \) and (partial) \( AB \) using the \text{Map} procedure of the RDD. The \( AB \) will be reduced to one, and a new RDD with the redistributed remaining work will be created.
Algorithm 5.2 The MR-GTries distributed census algorithm

\begin{algorithm}
\begin{algorithmic}
\Procedure{MR-GTries}{$G = (E, V, k)$}
\State $T \leftarrow$ GTrie filled with canonically labeled graphs.
\State \textit{verticesSets} \leftarrow \text{List of work-sets for each worker}
\State \textit{result} \leftarrow (partial) result gathered from the workers
\ForAll{$v \in V$}
\State \textit{verticesSets}[$v$ mod \(w$] = \textit{verticesSets}[$v$ mod \(w$] \cup \{v\}
\EndFor
\While{\textit{verticesSets} \neq \emptyset}
\State \textit{results, states} \leftarrow \text{Map}\left($\textit{verticesSets}, (V) \Rightarrow \text{Worker}(T, G, V)$\right)
\State \textit{result} \leftarrow \text{Reduce}\left($\textit{result, results}$\right)
\State \textit{verticesSets} \leftarrow \text{REDISTRIBUTE}(\textit{states})
\State \text{ADJUST}\text{TIME}\text{LIMIT}()
\EndWhile
\State \Return \textit{result}
\EndProcedure

\Procedure{Worker}{$T, G, V$}
\ForAll{$v \in V$}
\ForAll{$n \in \text{CHILDREN}(T, \text{root})$}
\State $\text{state, result} \leftarrow \text{COUNT}(G, n, \{v\})$
\If{\text{\text{TimeLimitExceeded}}} \Return \text{state, result} \EndIf
\EndFor
\EndFor
\State \Return \text{state, result}
\EndProcedure

\Procedure{COUNT}{$G, gN, V_S, \text{state, result}$}
\State $V_E \leftarrow \text{NEXT}\text{VERTICES}(gN, V_S)$
\ForAll{$v \in V_E$}
\If{\text{\text{TimeLimitExceeded}}} \State \text{state} \leftarrow \text{SAVE}\text{STATE}() \EndIf
\If{\text{$gN.isLeaf$}} \State $\text{result} \leftarrow \text{ADD}(gN, \text{graph}, \text{result})$ \Else
\ForAll{$n \in \text{CHILDREN}(gN)$}
\State $\text{COUNT}(G, n, V_S \cup \{v\}, \text{state, result})$
\EndFor
\EndIf
\EndFor
\State \Return \text{state, result}
\EndProcedure
\end{algorithmic}
\end{algorithm}
This chapter presents some experiments done with the proposed technique to assess the performance
and accuracy of it. The results will also be compared to an existing approach and classified. The
experiments are mainly divided into two sections.

1. Experiments regarding SAMPLECENSUS implemented using the FaSE algorithm by Paredes
   and Ribeiro. We will look at what speedup the sampling approach with FaSE will give towards
   the exact calculation. The implementation using FaSE will be referred to as SAMPLEFaSE.
   We will likewise assess its accuracy on different sampling rates, and what speedup is possible
   while obtaining a useful result.

2. The second section will cover the experiments regarding the implementation using MR-GTRIES
   by Eddin and Ribeiro. We will look at what further speedup edge sampling can give when
   using it with a distributed algorithm.

The results will be compared to RAND-FaSE [PR15], which is, to the best of our knowledge, the
currently best approximative census algorithm that can handle directed networks [RPS+19]. The
SAMPLEFaSE algorithm was implemented using C++ and the RAND-FaSE implementation provided
by Paredes and Ribeiro. The SAMPLEMR-GTRIES algorithm was implemented using Java and the
Apache Spark environment [ZXW+16]. However, both implementations also make use of the
third-party tool NAUTY by Mckay and Piperno [MP14] in its C++ implementation. Experiments are
then performed on Linux machines with two AMD EPYC 7401 cores and 256 GB of memory. In
this environment, we run FaSE on a single-core. SAMPLEMR-GTRIES, being a distributed algorithm,
is run on one to three of such machines, each using up to 48 cores at once. Experiments have
shown that using all 96 threads of the machine leads to an increase in runtime. That is why, despite
hyperthreading, we only use one thread per core.

We tested the methods on a large set of real representative networks from several domains. The
focus was to get diversity in topological features and scientific fields. Following, we describe all
networks, giving the source and type of the network.

1. **neural**: a directed neural network of the small nematode roundworm [RS14].

2. **amazon0601**: a co-purchasing network of amazon where the vertices represent products and
   they feature an edge if they are frequently co-purchased [LAH07][LK14].

3. **cit-patents**: a citation network of U.S. patents from 1963 to 1999. This type of network only
   features a small set of different subgraphs [LKF05][LK14].

4. **web-berkstan**: a web graph from 2002 containig pages from berkley.edu and stanford.edu
   domains. Edges are hyperlinks between those pages. [LLDM08][LK14]

5. **web-google**: another web graph from 2002 that was released by google as a part of the
   Google Programming Contest [LLDM08][LK14].
6 Experimental Evaluation

<table>
<thead>
<tr>
<th>Network</th>
<th>Type</th>
<th>Vertices</th>
<th>Edges</th>
<th>Avg. Degree</th>
<th>Max. Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>neural</td>
<td>Biological</td>
<td>297</td>
<td>2,359</td>
<td>15.89</td>
<td>139</td>
</tr>
<tr>
<td>foldoc</td>
<td>Sematic</td>
<td>13,356</td>
<td>120,238</td>
<td>4.20</td>
<td>728</td>
</tr>
<tr>
<td>twitter</td>
<td>Social</td>
<td>81,306</td>
<td>1,768,149</td>
<td>43.49</td>
<td>3,758</td>
</tr>
<tr>
<td>amazon0601</td>
<td>Co-Purchasing</td>
<td>403,394</td>
<td>3,387,388</td>
<td>16.79</td>
<td>2,761</td>
</tr>
<tr>
<td>web-google</td>
<td>Web</td>
<td>875,713</td>
<td>5,105,039</td>
<td>11.66</td>
<td>6,353</td>
</tr>
<tr>
<td>web-berkstan</td>
<td>Web</td>
<td>685,230</td>
<td>7,600,595</td>
<td>22.18</td>
<td>84,290</td>
</tr>
<tr>
<td>gplus</td>
<td>Social</td>
<td>107,614</td>
<td>13,673,453</td>
<td>254.12</td>
<td>22,022</td>
</tr>
<tr>
<td>cit-patents</td>
<td>Citation</td>
<td>3,774,768</td>
<td>16,518,948</td>
<td>8.75</td>
<td>793</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>Social</td>
<td>4,847,571</td>
<td>68,993,773</td>
<td>28.47</td>
<td>22,889</td>
</tr>
</tbody>
</table>

Table 6.1: Topological features of the used networks

6. **soc-livejournal**: a social network of the LiveJournal online community. Edges represent friendship status between users [LLDM08][LK14].

7. **twitter**: a network containing social circles from Twitter [ML12][LK14].

8. **gplus**: a network containing social circles from Google+ Where people shared their circle via ‘share circle’ [ML12][LK14].

9. **foldoc**: an online dictionary of computing terms [How+10]. This means, a directed network where an edge \((x, y)\) means that term \(y\) is used to describe the meaning of term \(x\) [RS14].

All graphs are directed, and weights, self-loops and multiple edges were ignored when existent. Table 6.1 summarizes the topological features of the provided networks.

6.1 Additional runtime

In this first section, we start by evaluating the extra time that SAMPLECENSUS needs for estimating the census. This extra time includes sampling the input network, calculating the transition probabilities described in section 4.2.1, and solving the linear equation systems. Note that two optimizations can be applied that reduce this runtime overhead significantly, especially for larger subgraph sizes.

- The input network can already be sampled when read from the input file. Doing so would result in sampling taking no additional runtime but rather may slightly decrease network reading time. Another effect would be that less memory is required when loading the network because not the entire network has to be loaded into memory first.

- The transition probabilities, as stated in section 4.2.1, could be saved into a file dependent on the overall sampling rate \(p\). Because the calculation takes exponential time, doing so can also decrease runtime significantly, especially for large subgraph sizes.

However, these optimizations are left for future work and are not part of the given implementations.

Table 6.2 shows the runtime of sampling the edges of the cit-patents and soc-livejournal with different sampling rates. One can observe that the required time increases with decreasing sampling rate, which leads to the sampling time being a significant part of the runtime at low sampling rates.
6.2 Comparison of SampleFaSE

We will now focus on the comparison of SampleFaSE against the exact algorithm FaSE and its random enumeration variation Rand-FaSE.

6.2.1 Speed Comparison

Both Rand-FaSE and SampleFaSE are run on the provided networks to compare the approximation speed to the exact value speed. Figure 6.1 plots the execution time for various sampling rates. Note that because the algorithms follow two fundamentally different approaches, different sampling rates give completely different speedups.

Table 6.2: Time it takes to sample the edges from the input network in seconds.

<table>
<thead>
<tr>
<th>Sampling rate</th>
<th>cit-patents</th>
<th>soc-livejournal</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>3.96s</td>
<td>17.55s</td>
</tr>
<tr>
<td>70%</td>
<td>8.89s</td>
<td>42.53s</td>
</tr>
<tr>
<td>40%</td>
<td>15.22s</td>
<td>69.58s</td>
</tr>
<tr>
<td>10%</td>
<td>19.89s</td>
<td>79.63s</td>
</tr>
</tbody>
</table>

Table 6.3: Time it takes to calculate the transition probabilities for different subgraph sizes.

<table>
<thead>
<tr>
<th>k</th>
<th># graphs</th>
<th># transition probabilities</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0.00007</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>169</td>
<td>0.00168</td>
</tr>
<tr>
<td>4</td>
<td>199</td>
<td>39,601</td>
<td>0.58894</td>
</tr>
<tr>
<td>5</td>
<td>9364</td>
<td>87,684,496</td>
<td>8689.00000</td>
</tr>
</tbody>
</table>

e.g. calculating the census on soc-livejournal with a sampling rate of 10%, which means that we only explore 10% of the edges, takes about 170 seconds, from which 80 seconds would be sampling the edges.

Table 6.3 shows the runtime of calculating the transition probabilities. It shows that the amount of different subgraphs increases exponentially with size and the runtime of the creation increases even faster. This is because for each subgraph, the algorithm loops through the power set of $G_E$ which is again exponential.

Last, the linear equation system with the transition probabilities must be solved to obtain the estimation. We showed earlier how to solve it in a very fast manner. However, this step always adds runtime to the computation and cannot be eliminated completely.

In the following experiments, the optimizable steps are not considered in the runtime to compare it better to the existing algorithms.

6.2.2 Comparison of SampleFaSE with datasets cit-patents and soc-livejournal

We already know that we can calculate the census on soc-livejournal with a sampling rate of 10%, which means that we only explore 10% of the edges, takes about 170 seconds, from which 80 seconds would be sampling the edges.

Table 6.3 shows the runtime of calculating the transition probabilities. It shows that the amount of different subgraphs increases exponentially with size and the runtime of the creation increases even faster. This is because for each subgraph, the algorithm loops through the power set of $G_E$ which is again exponential.

Last, the linear equation system with the transition probabilities must be solved to obtain the estimation. We showed earlier how to solve it in a very fast manner. However, this step always adds runtime to the computation and cannot be eliminated completely.

In the following experiments, the optimizable steps are not considered in the runtime to compare it better to the existing algorithms.

6.2.3 Comparison of SampleFaSE with the datasets cit-patents and soc-livejournal

We will now focus on the comparison of SampleFaSE against the exact algorithm FaSE and its random enumeration variation Rand-FaSE.

6.2.1 Speed Comparison

Both Rand-FaSE and SampleFaSE are run on the provided networks to compare the approximation speed to the exact value speed. Figure 6.1 plots the execution time for various sampling rates. Note that because the algorithms follow two fundamentally different approaches, different sampling rates give completely different speedups.

Table 6.2: Time it takes to sample the edges from the input network in seconds.

<table>
<thead>
<tr>
<th>Sampling rate</th>
<th>cit-patents</th>
<th>soc-livejournal</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>3.96s</td>
<td>17.55s</td>
</tr>
<tr>
<td>70%</td>
<td>8.89s</td>
<td>42.53s</td>
</tr>
<tr>
<td>40%</td>
<td>15.22s</td>
<td>69.58s</td>
</tr>
<tr>
<td>10%</td>
<td>19.89s</td>
<td>79.63s</td>
</tr>
</tbody>
</table>

Table 6.3: Time it takes to calculate the transition probabilities for different subgraph sizes.

<table>
<thead>
<tr>
<th>k</th>
<th># graphs</th>
<th># transition probabilities</th>
<th>time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>0.00007</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>169</td>
<td>0.00168</td>
</tr>
<tr>
<td>4</td>
<td>199</td>
<td>39,601</td>
<td>0.58894</td>
</tr>
<tr>
<td>5</td>
<td>9364</td>
<td>87,684,496</td>
<td>8689.00000</td>
</tr>
</tbody>
</table>

E.g. calculating the census on soc-livejournal with a sampling rate of 10%, which means that we only explore 10% of the edges, takes about 170 seconds, from which 80 seconds would be sampling the edges.

Table 6.3 shows the runtime of calculating the transition probabilities. It shows that the amount of different subgraphs increases exponentially with size and the runtime of the creation increases even faster. This is because for each subgraph, the algorithm loops through the power set of $G_E$ which is again exponential.

Last, the linear equation system with the transition probabilities must be solved to obtain the estimation. We showed earlier how to solve it in a very fast manner. However, this step always adds runtime to the computation and cannot be eliminated completely.

In the following experiments, the optimizable steps are not considered in the runtime to compare it better to the existing algorithms.

6.2.2 Comparison of SampleFaSE with datasets cit-patents and soc-livejournal

We already know that we can calculate the census on soc-livejournal with a sampling rate of 10%, which means that we only explore 10% of the edges, takes about 170 seconds, from which 80 seconds would be sampling the edges.

Table 6.3 shows the runtime of calculating the transition probabilities. It shows that the amount of different subgraphs increases exponentially with size and the runtime of the creation increases even faster. This is because for each subgraph, the algorithm loops through the power set of $G_E$ which is again exponential.

Last, the linear equation system with the transition probabilities must be solved to obtain the estimation. We showed earlier how to solve it in a very fast manner. However, this step always adds runtime to the computation and cannot be eliminated completely.

In the following experiments, the optimizable steps are not considered in the runtime to compare it better to the existing algorithms.

6.2.2.1 Speed Comparison

Both Rand-FaSE and SampleFaSE are run on the provided networks to compare the approximation speed to the exact value speed. Figure 6.1 plots the execution time for various sampling rates. Note that because the algorithms follow two fundamentally different approaches, different sampling rates give completely different speedups.
6 Experimental Evaluation

![Figure 6.1: Runtime of census computation for subgraphs of size 3 on different networks relative to the runtime for the exact computation.](image)

Both algorithms were also run on the gplus network, which sets itself apart because of its density. Usually, running an accurate census on such a network is unfeasible because it takes about three days to compute using the FaSE algorithm. We ran the SampleFaSE algorithm, with a sampling rate of 40% on this network to compute the census for subgraphs of size 3. Computing this only took over 6 hours to run and gives us an insight into the total number of subgraphs and particularities in the frequencies.

**Particularities in the runtime**

Succeeding, we want to show how different characteristics of the input parameters affect the speedup of the algorithm. We tested the correlation of many characteristics of the input values with the speedups. One particularity found is that runtime decreases more when the census is computed for larger subgraph sizes. Figure 6.2 shows the speedup at different sampling rates for various subgraph sizes on the foldoc network. This effect can not be seen in the execution times of Rand-FaSE. The shorter execution time is also not unexpected, because the edge-sampling affects the enumeration step more when more nodes are expanded.

e.g. consider a 3-subgraph with two edges. With a probability of $1 - p^2$, it will not be enumerated because it is not connected anymore. Consider a 4-subgraph with three edges. This graph will not be enumerated with a probability of $1 - p^3$, which is higher than $1 - p^2$ for all probabilities $p$.

### 6.2.2 Memory efficiency

Before moving on to comparing the accuracy. We quickly want to look at how SAMPLEFASE decreases the memory needed to compute the census. Table 6.4 shows the used memory on two graphs for different sampling rates and compares them to the memory used by plain FaSE. It is to mention that RAND-FASE does not feature any better memory efficiency than FaSE itself, so comparing to FaSE is sufficient. As it can be seen the used memory is lower by a few percent but not significantly. This is because the majority of memory is used by the g-trie built by the FaSE
6.2 Comparison of SampleFaSE

Figure 6.2: Runtime of census computation for different subgraph sizes on the foldoc network relative to the runtime of the exact census.

<table>
<thead>
<tr>
<th>Network</th>
<th>Used memory with sampling rate in Gb</th>
<th></th>
<th>Memory savings</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10%</td>
<td>30%</td>
<td>50%</td>
</tr>
<tr>
<td>soc-livejournal</td>
<td>2.02</td>
<td>2.08</td>
<td>2.15</td>
</tr>
<tr>
<td>cit-patents</td>
<td>0.95</td>
<td>0.96</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Table 6.4: Memory usage of SampleFaSE with different sample rates and decrease towards the exact FaSE algorithm.

algorithm. However Patcomp, another census algorithm features a much higher memory efficiency than FaSE, and due to the structure of SampleCensus it can easily be combined with Patcomp to save more memory.

6.2.3 Accuracy Comparison

The accuracy of the algorithms was tested with several methods to find out if any algorithm performs better in a specific measure. First the relative error for each subgraph was calculated by dividing the absolute error by the exact frequency. This error was averaged over all subgraphs and shown for both SampleFaSE and Rand-FaSE in Figure 6.3. As seen from the figure, the error of SampleFaSE increases significantly with a sampling rate below 50%. This increase especially comes from subgraphs with low frequencies getting huge errors. e.g. The subgraph in the following figure with a sampling rate of 10% on the foldoc network gets an average relative error of 8,036%.

Also, because of the way the census is estimated, a negative census may be estimated for some subgraphs, which also makes them have a significant error. These huge errors do not exist in Rand-FaSE, but the error is somewhat on the same level for all subgraphs. When taking another
more stable arithmetic mean such as the median, as shown in Figure 6.4, \textsc{SampleFaSE} does not feature such significant errors. Further does the error only heavily increase below a sampling rate of about 30%.

### Particularities in the accuracy

We also want to dive in further to find out what characteristics affect the accuracy of the census estimation. Therefore the correlation between many topological features and the error was observed. What can be seen is that the census on larger networks seems to get estimated better by both algorithms. Figure 6.5 models this by showing the median of the relative error for the tested networks concerning the edge-count of that network. It shows that with higher edge counts typically on the same sample rate, better results are given. Also the Spearman and Pearson ranking values are computed to check for correlations. The result of those values are shown in Table 6.5, which shows that a correlation of those values can be expected. The Spearman and Pearson rankings are usually used to measure the correlation of two datasets [08] [BCHC09]. Both rankings scale the correlation from $-1$ to $1$ where $0$ means no correlation, $1$ means perfect positive and $-1$ perfect
6.2 Comparison of SAMPLEFASE

Figure 6.5: Median of the relative error concerning the edge-count of each network. Note that the scaling on the graphics is different.

<table>
<thead>
<tr>
<th>Method</th>
<th>SAMPLEFASE (10% sample)</th>
<th>RAND-FASE (0.1% sample)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spearman ranking</td>
<td>−0.714</td>
<td>−1.0</td>
</tr>
<tr>
<td>Pearson ranking</td>
<td>−0.743</td>
<td>−0.794</td>
</tr>
</tbody>
</table>

Table 6.5: Correlation Coefficients for correlation between network size and the median of relative error.

negative linear correlation. To calculate the Pearson correlation coefficient $r$ for two datasets $X$ and $Y$ with a size of $n$, Benesty et al. propose the following formula:

$$r = \frac{n \cdot (\sum_i X_i \cdot Y_i) - \sum_i x \cdot \sum_i y}{\sqrt{\left[n \cdot \sum_i x^2 - (\sum_i x)^2\right] \cdot \left[n \cdot \sum_i y^2 - (\sum_i y)^2\right]}}$$

The Spearman ranking $s$ can then be derived from that by first for each dataset calculating the order of the values and save the ranking of an element $x$ in $R(x)$. With those rankings the correlation coefficient is calculated by

$$s = \frac{S_{xy}}{S_x S_y}$$

where $S$ is the standard deviation. So what can be observed is that the Spearman ranking does not take the actual values into account, and therefore may yield other results than the Pearson correlation.

On the subgraph level, it also seems that, in general, both algorithms estimate subgraphs with a higher frequency with higher certainty. Figure 6.6 shows the relative error of the subgraph frequencies of the tested networks with their exact frequency. It can be seen that for both algorithms, this effect is present. While for SAMPLEFASE, some estimations, especially with low frequencies, fall out of the picture, the correlation can be seen very prominently in the graphic for RAND-FASE. This relationship is explainable because, on small frequencies, a small change in the census on the sampled network hits the estimation more meaningful than it would on a subgraph with a high frequency.
One last particularity to state is that increasing the subgraph size for the census seems to decrease the accuracy of the information. This reduction can have two causes. The first one is that with larger subgraph sizes, more subgraphs with very low frequencies exist, which emphasize high relative errors. The second one is that more edges are not considered per subgraph on \textit{SampleFaSE} with the same sampling rate due to sub-sampling. Thus more information is lost on a larger subgraph than on a smaller one.

Figure 6.7 shows the average relative error of the census on the foldoc network for different subgraph sizes. What can also be seen is that the accuracy increases again on \textit{SampleFaSE} for tiny sample rates. This increase happens because, at such low rates, more frequencies on the sampled network become zero and hence, the assessed values also become lower or zero. An estimated zero however is a relative error of 100\% and thus better than the error values of the result with higher sample rates.

We also want to test the methods with two other measures of accuracy because both test for properties that are considered desirable. The first one is to test how many subgraph frequencies get estimated \textit{exact}, where \textit{exact} means that its error lies below 2\% of the exact value. Figure 6.8 shows the
6.3 Comparison of SampleMR-GTries

Figure 6.8: Proportion of subgraphs whose census gets estimated exactly. The census is calculated for subgraphs of size three.

results of that for the 3-subgraph census on some networks with different sample rates. What can be seen is that for SampleFaSE the results down to a sample rate of 50% are quite good. Lower sample rates on the other hand side deliver increasingly inexact estimations.

The second one is to test how well the algorithms preserve the order of the profile frequencies. e.g. whether a subgraph $a$ which has a higher frequency as a subgraph $b$ still has a higher frequency when estimated by the approximative algorithm. We tested this property again using the Spearman and Pearson rankings to check how the estimated dataset correlates to the exact one. Figure 6.9 shows the rankings on some networks for different sample rates. Both rankings for SampleFaSE stay very stable at a high value down to about 30%. It also to mention that the Pearson ranking also takes the difference of the frequencies itself into account and therefore shows much higher correlation values [BCHC09].

Accuracy vs Runtime

To finish the comparison between Rand-FaSE and our proposed version SampleFaSE, we want to look at the accuracy decrease combined with the gained speedup. Therefore, Figure 6.10 shows the relation between runtime enhancements and the median of the relative error for all tested networks and sample rates. As it can be seen, the Rand-FaSE algorithm provides a, on this measure, good accuracy down to speedups of three orders of magnitude. Meanwhile, SampleFaSE provides quite good solutions with a speedup of up to an order of magnitude. Further speedup gives significantly larger errors that may not be useful anymore.

6.3 Comparison of SampleMR-GTries

Since SampleCensus can be combined with any census algorithm, we want to show how performance increases when combining it with another popular method of decreasing runtime: distributed computation. Therefore, we present an implementation of the MR-GTries together with edge-sampling.
6 Experimental Evaluation

**Figure 6.9:** Spearman and Pearson correlation coefficients for correlation between the exact count and the estimation. Results are shown for different sample rates for a 3-subgraph census.

**Figure 6.10:** Median of the relative error of the estimated frequencies about the speedup gained towards the exact calculation.
6.3 Comparison of SampleMR-GTries

Figure 6.11: Runtime of MR-GTries with different amounts of workers used, relative to the runtime with one worker. The horizontal lines represent the relative runtime using FaSE.

First, we want to show how the MR-GTries algorithm decreases runtime when provided with more worker threads. Figure 6.11 shows the algorithm’s runtime with different worker counts relative to the runtime with just one core. Each horizontal line shows the runtime of the FaSE algorithm. As it can be seen, the runtime with just one worker thread is usually higher than the runtime of FaSE. Only the gplus is an exception to that. It can also be assumed that the relative speedup obtained relates with the runtime for the exact calculation because for all three networks tested it holds that higher overall runtimes correlate with a better performance of MR-GTries. The runtime benefit of FaSE against MR-GTries on a single core mainly exists because of the significantly higher efficiency of C++ versus Java and the computation overhead gained through the worker-timeouts, and the overhead through the Apache Spark framework [ZXW+16]. It is to mention that the worker also timeouts when only one worker is computing the census. Nevertheless, with higher amounts of workers, the MR-GTries algorithm takes less time to compute the census than FaSE does.

So applying edge-sampling should also map this performance increase to the approximative approach. Figure 6.12 shows the runtime of different sample rates with the runtime for the exact census using the same amount of cores. What can be seen is that FaSE reaches a higher speedup than MR-GTries when combined with edge-sampling. The speedup is even lower when more than one core is used. It also seems like, in general, with more workers on the cluster the speedup decreases even further. However, it still reaches up to one order of magnitude. This decrease in the speedup is explainable by the runtime used for setting up the Apache Spark environment and synchronizing the workers. This runtime takes constant more time, and sampling does not affect this runtime. Therefore, the speedup can be lower when using MR-GTries with more cores.

Last but not least, we quickly want to show the performance of SampleMR-GTries compared to the Rand-FaSE algorithm. Figure 6.13 shows the speed of calculating a 3-subgraph census on both, the gplus and twitter network with the obtained median of the relative error. Here it can be seen that the algorithm when used with gplus outperforms Rand-FaSE significantly and large speedups of up to 4 orders of magnitude can be reached with a decent error. The test with twitter does not show that significant performance. This may be because we showed earlier, that first, smaller networks get a worse estimation and second, smaller networks get a smaller speedup using MR-GTries. It should, however, also be accounted that SampleMR-GTries obtains a decent amount of its speedup from the distribution of the calculation.
6 Experimental Evaluation

**Figure 6.12:** Runtime of different setups using various sample rates, relative to the runtime of the same setup performing the exact census.

**Figure 6.13:** Runtime in relation to the median of the relative errors using both algorithms. Runtime is displayed relative to the runtime of FASE calculating the exact census.
7 Conclusion

In this paper, we presented a technique to estimate a subgraph census by sampling the edges of the input network. Further, we supplied two implementations of that algorithm. One of which uses an up-to-date network-centric algorithm FASE [RPS+19], and the other uses the currently best, distributed, census algorithm, that is based on the Map-Reduce paradigm, MR-GTRIES [RPS+19]. However, the implementation of the so-called SAMPLECENSUS is easily adaptable to any census algorithm and does only really depend on the Nauty library [MP14].

We also compared the proposed algorithm with a previous approximative census algorithm, namely the most recent one RAND-FASE [PR15]. Here, with different measures, the edge-sampling approach does not give an accuracy as good as the RAND-FASE algorithm at the same speedup. Especially for subgraph sizes larger than three, SAMPLECENSUS lacks a reliable estimation. Another disadvantage, which is to optimize but still needs computation time, is that SAMPLECENSUS must determine the transition probabilities first. However, we also showed that with increasing network size, the estimations get better, which is exceptionally relevant because estimation techniques are mainly used when exact approaches are not feasible anymore.

Furthermore, SAMPLECENSUS features a slightly better memory efficiency, which can be relevant especially when working with large graphs. FASE in general, is not known to have a good memory efficiency. Therefore SAMPLECENSUS can be combined with more memory-efficient algorithms such as PATCOMP [Jai+17].

More generally, one meaningful benefit of SAMPLECENSUS is that it can be combined with any census algorithm without any effort. This quality makes it possible to also combine with other runtime decreasing methods, such as distributing the computation. We implemented the algorithm with MR-GTRIES and showed that in combination, it outperforms RAND-FASE by improving the low runtime of MR-GTRIES even further. As far as we know, it is therefore the first algorithm to perform an approximative census on directed graphs in a distributed environment. Also, it makes SAMPLECENSUS future-proof for upcoming algorithms that feature lower runtimes.

7.1 Future Work

Some research on this topic is still open and can be worked on in the future. This research was outside of the scope of this thesis, or there was not enough time to cover it. Regarding the SAMPLECENSUS algorithm, the in Chapter 6 presented missing optimizations, i.e. required sampling time and multiple transition probability calculation can be optimized. One can be optimized by sampling the edges of the input network while reading it. The other could save the transition probabilities in persistent storage to reuse them in other calculations and thus have an insignificant runtime impact.
Furthermore, SAMPLECENSUS can be combined with more exact algorithms and inspect how the speed on these algorithms get affected by sampling the edges. Mainly the memory efficiency should be observed when coupling SAMPLECENSUS with the PATCOMP algorithm.

Regarding MR-GTRIES, further investigation of what network characteristics affect the speedup of the algorithm can be done. Also, different optimizations can be tried, such as first expanding vertices with a higher degree because those regularly have longer enumeration times. Because some workers tend to run significantly longer than others, it can also be observed whether work units can be made more granular by, e.g., also distributing the second expanded vertex.


All links were last followed on October 26, 2021.
Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature