Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

# Scalable Hypergraph Partitioning

Heiko Geppert

**Course of Study:** Softwaretechnik

**Examiner:** Prof. Dr. Kurt Rothermel

**Supervisor:** Dipl.-Inf. Christian Mayer

**Commenced:** 2016-11-7

**Completed:** 2017-5-9

**CR-Classification:** C.2.4, E.1, G.2.2

# Abstract

The interest in graph partitioning has become quite huge due to growing problem sizes. Therefore more abstract solutions are desirable. In this thesis, hypergraph partitioning is investigated since hypergraphs provide a better level of abstraction than normal graphs. Further, restreaming approaches are examined because the partitioning results of real time strategies are often not satisfiable. It will be shown that they can perform up to 15% better than real time approaches and can sometimes even hold up to polynomial approaches. By putting more thought into the restreaming, the partitioning results become even better. This is shown empirical when proposing Fractional Restreaming a novel 'Partial Forgetting' strategy. Meanwhile, the additional runtime needed is negligible compared to polynomial strategies. Finally SHP, a novel graph partitioning and evaluation framework is introduced.

# Contents


1 Introduction 15

2 Foundation & Problem Definition 19
- 2.1 Hypergraphs . . . . . 19
- 2.2 Graph partitioning . . . . . 19
  - 2.2.1 Metrics . . . . . 20

3 Background 23
- 3.1 All On One . . . . . 23
- 3.2 Random . . . . . 23
- 3.3 Greedy . . . . . 23
- 3.4 Balance big . . . . . 24
- 3.5 Prefer Big . . . . . 24

4 Own Approaches 27
- 4.1 Real Time Approaches . . . . . 27
  - 4.1.1 Greatest Topic Intersection . . . . . 27
  - 4.1.2 Smallest Non Topic Intersection . . . . . 28
  - 4.1.3 Never Max . . . . . 29
  - 4.1.4 Degree Aware . . . . . 29
- 4.2 Restreaming Approaches . . . . . 30
  - 4.2.1 Two Pass . . . . . 30
  - 4.2.2 Vertex Correction . . . . . 31
  - 4.2.3 Meta Restreaming . . . . . 31
  - 4.2.4 Fractional Restreaming . . . . . 31
  - 4.2.5 Topic Correction . . . . . 32
  - 4.2.6 Topic Correction Narrow Scope . . . . . 34
  - 4.2.7 Adaptive Balance . . . . . 34

5 Evaluation 35
- 5.1 Evaluation Survey . . . . . 36
- 5.2 Real Time Strategy Evaluation . . . . . 42
  - 5.2.1 Greatest Topic Intersection . . . . . 42

5.2.2 Smallest Non Topic Intersection . . . . . . . . . . . . . . . . . . . . 43
5.2.3 Degree Aware . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
5.3 Restreaming Strategy Evaluation . . . . . . . . . . . . . . . . . . . . . . 50
5.3.1 General Restreaming . . . . . . . . . . . . . . . . . . . . . . . . . 50
5.3.2 Partial Forgetting . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
5.3.3 Adaptive Balancing . . . . . . . . . . . . . . . . . . . . . . . . . . 54
5.4 Decision Guidance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 60

6 Simple Hypergraph Partitioner 63
6.1 Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
6.2 Data Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
6.3 Algorithms and Extensibility . . . . . . . . . . . . . . . . . . . . . . . . 66
6.4 Usage . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.5 Automation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70

7 Conclusion 73

Bibliography 75

# List of Figures

5.1 Survey I on the Amazon graph considering the Cut Size . . . . . . . . . . 37
5.2 Survey I on the Amazon graph considering the execution time . . . . . . 37
5.3 Survey I on the Amazon graph considering the Max Edge metric . . . . . 38
5.4 Survey I on the small MovieLens graph considering the Cut Size . . . . . 38
5.5 Survey I on the MovieLens 20m graph considering the Cut Size . . . . . 39
5.6 Survey II on the Book-X graph considering the Cut Size . . . . . . . . . . 40
5.7 Survey II on the Amazon graph considering the Cut Size . . . . . . . . . 40
5.8 Survey II on the MovieLens 20m graph considering the Cut Size . . . . . 41
5.9 Survey II on the Book-X graph considering the Max Edge . . . . . . . . . 41
5.10 Survey II on the MovieLens 20m graph considering the Max Edge . . . . 41
5.11 Survey II on the Book-X graph considering the execution time . . . . . . 42
5.12 Survey II on the MovieLens 20m graph considering the execution time . 42
5.13 GTI's Cut Size with different parameters on the ml20m graph . . . . . . 43
5.14 GTI's Cut Size with different parameters on the Amazon graph . . . . . . 44
5.15 GTI's Cut Size with different parameters on the mlSmall graph . . . . . . 44
5.16 GTI's Cut Size with different parameters on the mlSmall graph when setting the balance constraint border to 50 . . . . . . . . . . . . . . . . . 45
5.17 Cut Size on mlSmall, SNTI vs Greedy . . . . . . . . . . . . . . . . . . . . 46
5.18 Cut Size on amazon, SNTI vs Greedy . . . . . . . . . . . . . . . . . . . . 46
5.19 Cut Size on ml20m, SNTI vs Greedy . . . . . . . . . . . . . . . . . . . . 47
5.20 Max Edge on ml20m, SNTI vs Greedy . . . . . . . . . . . . . . . . . . . . 47
5.21 Execution Time on ml20m, SNTI vs Greedy . . . . . . . . . . . . . . . . 48
5.22 Cut Size of Degree Aware vs Greedy & GTI on Book-X . . . . . . . . . . . 48
5.23 Max Edge of Degree Aware vs Greedy & GTI on Book-X . . . . . . . . . . 49
5.24 Cut Size of Degree Aware vs Greedy & GTI on amazon . . . . . . . . . . 49
5.25 Execution Time of Degree Aware vs Greedy & GTI on amazon . . . . . . 50
5.26 Cut Size of restreaming vs real time on the Book-X graph . . . . . . . . . 51
5.27 Max Edge of restreaming vs real time on the Book-X graph . . . . . . . . 51
5.28 Execution Time of restreaming vs real time on the Book-X graph . . . . . 52
5.29 Cut Size of restreaming vs real time on the MovieLens 20m graph . . . . 52
5.30 Cut Size of Fractional Restreaming with different parameters on Book-X 53
5.31 Cut Size of Partial Forgetting strategies vs Meta Restreaming on Book-X . 54
5.32 Max Edge of Partial Forgetting strategies vs Meta Restreaming on Book-X 54

5.33 Execution Time of Partial Forgetting strategies vs Meta Restreaming on Book-X . . . 55
5.34 Cut Size of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X 56
5.35 Cut Size of GTI, Meta Restreaming, Adaptive Balance and hMetis on Amazon . . . 56
5.36 Max Edge of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X . . . 57
5.37 Max Edge of GTI, Meta Restreaming, Adaptive Balance and hMetis on Amazon . . . 57
5.38 Execution Time of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X . . . 58
5.39 Cut Size when balancing vertices . . . 59
5.40 Max Edge when balancing vertices . . . 59

6.1 Layered architecture of SHP with packages and accesses . . . 64
6.2 Program pipeline . . . 65
6.3 Data model of SHP . . . 66
6.4 Abstraction of partitioning strategies . . . 67

# List of Tables

5.1 Systems running the evaluations . . . . . . . . . . . . . . . . . . . . . . . 35
5.2 Hypergraphs used for evaluating . . . . . . . . . . . . . . . . . . . . . . . 36
5.3 Adaptive Balance on the Book Rating graph with 105282 vertices with several starting balances monitoring the Cut Size and number of relocated vertices after each pass . . . . . . . . . . . . . . . . . . . . . . . . . . . 58

6.1 SHP parameter survey . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
6.2 SHP implemented strategies . . . . . . . . . . . . . . . . . . . . . . . . . . 70

# Listings

2.1 Example build-up of a vertex list . . . . . . . . . . . . . . . . . . . . . . 20

6.1 Build-up of the results.csv . . . . . . . . . . . . . . . . . . . . . . . . . . 68
6.2 Build-up of the resultsHR.txt . . . . . . . . . . . . . . . . . . . . . . . . . 69

# List of Algorithms

1 Greedy strategy from Alistarh et al. . . . . . . . . . . . . . . . . . . . . . . . 24

2 Balancing in GTI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

3 Fractional Restreaming . . . . . . . . . . . . . . . . . . . . . . . . . . . . 32

4 Topic Correction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33

# 1 Introduction

Nowadays, large data structures are common in many aspects of modern life. Social media platforms like Facebook or Twitter have millions of users [staa; stab] and services as Netflix or Amazon Instant Video provide large amounts of media, ratings and recommendations. Wikipedia stores endless numbers of pages [Wik] but also moderator elections and communications. Instant messenger like WhatsApp or Facebook Messenger have more than a billion users and billions of communications [staa; stac]. With the rise of the Internet of Things (IOT), the produced amount of data is expected to grow even bigger [Gar16]. To put these data to the maximum usage it is necessary to process it quickly in a scalable way. Otherwise providing real-time responding systems like instant messengers or user optimized online shops would not be possible for a large number of users.

Using larger computers and servers is not feasible for modern systems. Hence, the usage of distributed systems - server clusters - has become state of the art. Therefore, the data has to be spread over several machines. This makes it quite easy to provide more space and computation power. The challenge thereby is scattering the data as disjoint as possible while still having a balanced workload. If the disjoint scattering is violated, there is more overhead needed. For example, all machines could store the complete dataset. However, the machines would then either calculate redundant (so the cluster is useless, one machine would be enough) or they would split the computation. Because of the splitting they would have an enormous communication overhead which machines calculates which part of the data. If the data is distributed well but the load balance is violated, some machines need much more computation time than others. This results in much idle time for iterative distributed algorithms. Consequently a balanced data scattering without many replications is indispensable for distributed systems.

Finally the data structure itself has to be considered. Large tables like they are used in SQL databases can not be distributed easily. By partitioning the tables the machines would have a gigantic communication overhead and by partitioning the tables the overhead would still persist because join operations would cause the machines to match the whole data between them. Taking this into account another data structure has to be used. A widespread solution are graphs. Graphs are easy to understand for the programmers and provide intuitive computation paradigms (e.g. ’think-like-a-vertex’

[Mal+10; May+16a; May+16b]). Moreover, a partitioned graph provides additional benefit considering distributed systems. Every machine can store a part of the graph and compute the vertices stored on its own memory. To realize the links to adjacent graph areas some communication to other machines is necessary. However the cuts can be minimized by cutting the graph in a clever way.

A classical graph consists of a set of vertices and several edges each connecting two vertices. To represent a group based community (like Facebook groups or group chats in instant messengers), bipartite graphs can be used. Those have two groups of vertices (in this case users and groups) and each vertex can only be linked to vertices of the other group. This however has the disadvantage of having different semantics in the vertices. A more abstract way to represent such data are hypergraphs. Hypergraphs consist also of a vertex set and edges connecting them. But every edge can connect an arbitrary number of vertices. That way groups can be represented as hyperedges. The advantage gained from hypergraphs is a better abstraction level and clear semantics for edges and vertices.

Considering general purpose graph processing systems [HZY15a; HZY15b; Mal+10] there are many ways to use the benefits of graphs by providing different algorithms which are executed on the graph. Depending on the graph size (web graphs can easily go into billion scale [Man]), the change rate (e.g. constant growth of the internet) and the algorithm to be performed it is crucial to be able to partition the graph quickly. The input data could either be so huge and because of changes needed to be recalculated frequently so it would not be feasible anymore to use offline partitioning strategies like hMetis [KK00]. Furthermore using a partitioning with a runtime larger than $\mathcal{O}(n)$ would not be satisfiable when the graph algorithm executed afterwards is linear or even sub linear. For these reasons this thesis will take a look at linear graph partitioning strategies - both real time strategies and restreaming strategies. Real time strategies can be used to perform a very fast partitioning or as part of a larger meta strategy. They read the data input only once and their partitioning choices are irreversible. The restreaming strategies on the other hand read the input data several times and reassign the vertices when they find better placements. Restreaming still has a linear runtime and can be an excellent choice if the partitioning can take more time. However, a fast scattering is still guaranteed

## Arrangement & Contributions

In this thesis, the impact of restreaming compared to real time strategies is examined. Thereby, several novel real time and restreaming strategies are proposed. Strategies

which have proven to provide good results are examined in detail like Fractional Restreaming. In particular, the following contributions are provided:

- a survey of hypergraph partitioning strategies having linear runtime,
- a proposal of new strategies with linear runtime,
- an evaluation of all the strategies and a resulting decision guidance,
- an empirical evidence that restreaming significantly improves partitioning up to 15% while still maintaining feasible runtimes and does not have to fear the competition with polynomial strategies,
- a further improvement of the partitioning by adding more logic into the restreaming resulting in Fractional Restreaming,
- Scalable Hypergraph Partitioner - a novel partitioning and evaluation framework which can be used to evaluate partitioning strategies and is expendable with further strategies easily.

This thesis is structured as follows:

**Chapter 2 – Foundation & Problem Definition:** Providing background knowledge and defines the problem attended

**Chapter 3 – Background** provides a survey over linear algorithms provided from other sources

**Chapter 4 – Own Approaches** provides a survey over the linear algorithms created while working on this thesis based on the strategies from Chapter 4

**Chapter 5 – Evaluation** Own approaches are evaluated and the results discussed in this section.

**Chapter 6 – Simple Hypergraph Partitioner** introduces the hypergraph partitioning and evaluation framework implemented during this thesis

**Chapter 7 – Conclusion** concludes the thesis, covers related work and provides prospects to future work

# 2 Foundation & Problem Definition

This section covers the theoretical background necessary to understand the topic and provides a formal problem definition. It also covers the metrics used to evaluate the quality of a partitioning strategy.

## 2.1 Hypergraphs

Hypergraphs are a more abstract form of graphs. An undirected, unweighed graph $G$ is defined as pair $(V, E)$ with $V$ being a set of vertices and $E$ being a set of edges which are defined as 2-tuple with two vertices. An undirected, unweighed hypergraph $H$ can be defined as pair $(V, E)$ with $V$ still being a set of vertices and $E$ being a set of edges. However an edge is now defined as $(e \subseteq V | e \neq \emptyset)$ [Dev+15].

Alternatively, hypergraphs can be seen as a set of items (vertices) $N = \{1, 2, .., n\}$, a set of topics (hyperedges) $M = \{1, 2, .., m\}$ and a demand matrix $D = (d_{i,t}) \in \{0, 1\}^{n \times m}$ [AIV15]. The demand matrix shows which items and topics are connected.

$$d_{i,t} = \begin{cases} 1 & \text{, if item } i \in \text{topic } t \\ 0 & \text{, otherwise} \end{cases} \tag{2.1}$$

By connecting only two vertices, hypergraphs can also be used to represent classical graphs. Those on the other hand can be seen as bipartite graphs to represent hyperedges meaning hypergraphs and classical graphs are equal in function [AIV15].

## 2.2 Graph partitioning

The graph partitioning problem is the task to split a graph $G$ into $k$ subgraphs $(P_1, P_2, .., P_k)$ with $G = P_1 \cup P_2 \cup ... \cup P_k$. This means each partition holds an own smaller graph with a set of Items $N_k = \{n_1, n_2, .., n_k\}$, a set of topics $M_k = \{m_1, m_2, .., m_k\}$ and an demand matrix $D_k = (d_{k_i,t}) \in \{0, 1\}^{n_k, m_k}$.

Thereby, the goal is to optimize several quality attributes discussed in 2.2.1. Graph partitioning is known to be NP-Complete [Pet+15]. Thus, when partitioning a graph (or hypergraph) in linear runtime only a local optima will be found with high probability. Therefore, different quality aspects have to be weighed up against each other. By having a much longer execution time, the strategy can find better solutions in terms of other quality metrics like balancing or replication. When ignoring replicas the partitioning, time and balancing can be optimized. Far large-scale real-world applications, it is crucial to find a balance between several goals.

Within this thesis it is assumed the graph size is not known beforehand. Neither does the partitioning strategy know the number of vertices nor the number of hyperedges before the partitioning. This restriction has to be kept in mind since making assumptions about the graph size becomes impossible for real-time strategies. Furthermore, the graphs will be read (and therefore streamed to the partitioning strategy) as vertex list. An example for a vertex list can be seen in Listing 2.1. The first number of each line is the vertex ID, the following numbers are the IDs of the connected hyperedges. It is important to mention that vertices and hyperedges have separate IDs. For example the ID '1' is given to the first vertex and the first hyperedge. A single exception to this input format will be hMetis [KK00] which is used as benchmark.

**Listing 2.1:** Example build-up of a vertex list

```
1:1,2,3,4;
2:1,4,5;
3:2,5,6,7;
4:1,7,8,9,10,11,12,13,14;
5:3,8,10;
6:9;
...
```

### 2.2.1 Metrics

In the following, several hypergraph partitioning metrics used in this work are presented. Here, the metrics mostly focus on low network communication overheads in the later graph processing. However a balanced partitioning is also necessary to avoid uneven calculation times after the partitioning. This also includes using all given partitions. If partitions stay empty, a server of the computing cluster will not participate the calculation. This would mean a server of the cluster using the graph afterwards would not participate and therefore be useless.

### Cut Size

Replicating edges on several partitions causes the need of network communication. Therefore, minimizing the number of edges which have to be split at least once and minimizing the number of edge splits in general is the main goal to reduce the overall network traffic. There are several possibilities to measure the number of edge cuts [KK00]. For example the sum of external degrees (SOED), which counts the number of partitions an edge is spanned by if the edge is placed on more than one partition. Another metric is the $(K-1)$ metric also calculating the number of partitions spanned by each edge but subtracting the number of edges since every edge has to be placed on at least one partition. Hence, this metric is more straight forward to calculate since no distinction of cases has to be made while the information value is the same to SOED. Since $(K-1)$ is the only metric used to determine the number of edge cuts it will be called 'Cut Size' in this thesis. The Cut Size is defined as follows:

$$CutSize = (\sum_k M_k) - |M| \tag{2.2}$$

### Max Edge (Maximum number of hyperedges on a single partition)

The partition with the most edges connected to it is the bottleneck in iterative graph computation algorithms [Che+15; ST04]. Therefore, optimizing this bottleneck is another goal. However this bottleneck could be dissolved by using strong hardware. Hardware constraints though are not part of this work. The maximum number of hyperedges on a single partition is defined as Max Edge.

$$MaxEdge = \max(|\bigcup_{i \in N_k} h_k(i)|) \tag{2.3}$$

$h_k(i)$ defines the set of topics the item i in partition k is connected with.

$$h_k(i) = \{x | d_{k-i,x} = 1\} \tag{2.4}$$

### Balance

The balance tells how well the graph is distributed over the partitions. Alistarh et al. tried to minimize the maximal number of hyperedges [AIV15]. In their approach the balancing was done based on hyperedges. Therefore, the approaches introduced in this thesis will also use hyperedges as load metric to be balanced. So each partition should have about the same number of hyperedges. Most strategies have the ability to set a

balance via parameter. If the balance can be given in a percentage then 5% seem to be a good balancing goal because it restricts the balance not be be to lax and at the same time should provide a high degree of freedom. This means the load difference between the least loaded and the most loaded partition should not be greater than 5%. Strategies which have no percentage parameter but a fixed value use a load difference of 100 edges what is suggested [AIV15].

In contrast to this balance matric, hMetis balances the number of vertices and does not use the least loaded partition for the calculation but the average partition load [KK00]. Thus, some evaluations will be made with the own approaches balancing the vertices. It will be mentioned when these changes are applied to the strategies.

#### Execution time

The more time is given to perform a graph partitioning, the better results are possible. Hence, the execution time of the tested strategies has to be kept in mind. However, the precise execution time is quite insignificant because the complexity of all tested strategies is except for hMetis at most $\mathcal{O}(n)$ and therefore fast enough.

#### Metric summary

The previous metrics can be rated as follows: A partitioning has to be balanced to be valid. This does not strictly mean a partitioning is invalid if the load difference between two partitions is higher than 100 edges. Some graphs are way to hard to partition totally balanced (e.g. if a large hub - vertex with many topics - appears at the end of the graph stream). If the partitioning is valid the Cut Size is the primary metric determining the quality. The Max Edge metric serves as a secondary quality attribute to have a fair and realistic view on the partitioning. The execution time is used to classify the strategies in different runtime classes. It is also used as tie breaker if the previous metrics show no significant difference.

# 3 Background

This chapter presents some strategies proposed in [AIV15] which will be compared with the own approaches developed during this thesis. Therefore, the strategies will be explained in short.

## 3.1 All On One

The All On One strategy is a trivial algorithm assigning all incoming vertices on one partition. Therefore, the Cut Size metric is optimal and although the number of hyper-edges on the partition is at maximum, no network traffic would be needed. However the balancing is not valid. The results of this strategy will be used as an upper bound as less trivial strategies have to perform better in order to justify the additional effort.

## 3.2 Random

The random strategy spreads the vertices randomly over the partitions. Thus, the Random strategy is probably almost perfectly balanced with high probability. However the Cut Size is far worse than the Cut Size of other strategies since no logic is put into the assignment. Like All On One, this strategy it is used as upper bound benchmark which has to perform worse in every case.

## 3.3 Greedy

Alistarh et al. presented the Greedy approach as their best real time strategy [AIV15]. Greedy is a fast strategy which is also used by several other strategies presented in this thesis. The algorithm tries to map incoming vertices on the partition having the highest topic intersection while maintaining a balancing constraint. Every partition which does not meet the constraint is not considered for the assignment. The constraint checks for

every partition if the number of assigned vertices is higher than the number of edges on the least loaded partition plus a fixed number of edges. This number is determined with a parameter and was introduced as slack.

First the empty partitions ($S_1, ..., S_k$) are created. Then the vertices of the graph are streamed into the algorithm. For each incoming vertex, the following steps are taken. First the vertex ($v$) and all its topics ($R$) are read. Next the balancing constraint is checked by considering only the partitions having less topics than the partition with least topics plus the slack (100 has been proven to be good). For the remaining partitions, the intersection sizes of the partition's and vertex's topics are calculated. The vertex is finally stored on the partition having the greatest intersection size. If several partitions have the greatest intersection size, the partition load is used as tie breaker storing the vertex on the least loaded partition.

**Data:** a hypergraph as vertex list and a capacitiy slack $c$
**Result:** partitioning into $k$ parts
Set initial partitions $P_1, P_2, ..., P_k$ to be empty sets ;
**while** *vertices are left* **do**
  Receive the next vertex $v$ and its topics $R$ ;
  $I \leftarrow i : |P_i| \leq \min |P_j| + c$ ;
  Compute $r_i = |P_i \cap R| \forall i \in I$;
  $j \leftarrow argmax_{i \in I} r_i$;
  $P_j \leftarrow P_j \cup R$
**end**
return $P_1, ..., P_k$

**Algorithm 1:** Greedy strategy from Alistarh et al.

## 3.4 Balance big

The balance big strategy treads the incoming vertices in two different ways depending on the vertices. Small vertices are assigned according to the Greedy strategy. Big vertices are placed on the least loaded partition. A vertex is considered as big if it has more than 100 topics.

## 3.5 Prefer Big

Prefer Big is an extension to the balance big strategy. It uses a vertex buffer which can hold up to 100 small vertices. Incoming big vertices are placed on the least loaded

partition (like Balance Big). Small vertices are stored in the vertex buffer. As soon as the buffer is full, all small vertices in it are assigned according to the Greedy strategy.

# 4 Own Approaches

Most of the strategies proposed in Chapter 3 seem insufficient considering the unknown input data and their partitioning performance was not very good [AIV15]. Consequently, a set of novel partitioning strategies developed during this thesis addressing these draw backs will be proposed in this chapter.

## 4.1 Real Time Approaches

The following strategies address the insufficient partitioning quality of the strategies proposed in Chapter 3 while maintaining real time partitioning times. Further, they provide parameter independent partitioning solutions which perform with constant quality for every input size.

### 4.1.1 Greatest Topic Intersection

Greatest Topic Intersection (GTI) is the attempt to modify the Greedy strategy to be less parameter dependent. Greedy has a slack parameter telling the algorithm how much imbalance is allowed. This parameter defines the imbalance as the difference in the number of connected hyperedges. This metric is independent from the graph size. Hence, the strategy could possibly perform worse if the parameter is chosen poorly (e.g. to small for large graphs or to large for small graphs). This can easily happen when the size of the input graph is unknown. Therefore, GTI uses a percentage balance to be dependent from the graph size.

GTI works similar to Greedy. An intersection size of the partition and the incoming edge is calculated for every partition. Afterwards the balancing constraint is checked. Thereby a percentage is used to determine the allowed edge imbalance. The vertex is finally placed on the valid partition which is least loaded. At the beginning of the stream the percentage can not be used. E.g. if $5\%$ imbalance would be allowed, it would take 20 edges per partition to allow one edge difference. For this reason the balancing is not considered at the beginning. The decision process whether a partition is considered

balanced or not is shown in algorithm 2. So the first 100 topics per partition can be assigned without considering the balance. As soon as the 100 topics have been exceeded the next vertex to be assigned has to fulfil the balancing constraint.

**Data:** the number of edges connected with the partition $n$,
the allowed imbalance $\lambda$ (for 5% imbalance enter 1.05)
**Result:** boolean if the partition is considered as balanced
**if** $n < 100$ **then**
    `// no balancing at the beginning`
    return true;
**else if** $n > \lambda * \min |P|$ **then**
    `// balancing constraint is met`
    return true;
**else**
    `// balancing constraint is not met`
    return false;
**end**

**Algorithm 2:** Balancing in GTI

### 4.1.2 Smallest Non Topic Intersection

Inverting the previous idea, the Smallest Non Topic Intersection strategy (SNTI) calculates the difference of partition and vertex (number of hyperedges they do not share), trying to minimize it.

SNTI was developed during this thesis as an evolution of GTI because its first results were not promising and other viewpoints were considered. Instead of maximizing the same metric as Greedy (topic intersection) the strategy reverses the metric and tries to minimize it (difference). This way each vertex is not placed on a partition it has not much in common and consequently assigned to a partition sharing many of its hyperedges. The vertex assignment and the optimization goal is shown in Equation 4.1.

$$v \rightarrow \arg\min_{p \in P} diff_size(v, p) \tag{4.1}$$

$$diff_size(v, p) = |p| - (p \cap v.topics) \tag{4.2}$$

SNTI solves the balancing problem native. The more vertices are assigned, the more topics are on the partition. Hence, the probability for future vertices to have a lot of

topics it is not connected to it is much higher. Therefore, no balancing parameter is required. The first vertices are automatically hashed because the partitions without assignments minimize the assignment goal. This results in a possible bad assignment at the beginning but the strategy is totally independent from the graph size.

### 4.1.3 Never Max

Considering the Max Edge metric, the approach of assigning anywhere but the maximal loaded partition was developed. The strategy checks for every partition if the assignment of the next vertex would negatively effect the Max Edge metric and considers all options which would not affect the metric for an assignment with Greedy. If every assignment would affect the Max Edge metric, all partitions are considered as valid options for the Greedy strategy.

The goal of the strategy is to keep the bottleneck (partition with the highest number of connected edges) in equally distributed systems as low as possible. This however can result in suboptimal assignments considering the Cut Size if the Max Edge metric would be violated. This may happens even if the assignment preferring the Cut Size would not violate the general balancing constraint. Hence, this strategy is not optimizing the Cut Size with first priority and should be used with caution and awareness for the optimization goal. However this trade-off could be a good choice when using meta strategies to optimize the Max Edge metric at the last part of the stream, while aiming for a good Cut Size beforehand.

### 4.1.4 Degree Aware

Looking at real-world graphs, many of them have a power law distribution of the vertex degree. This means there are few nodes with a very high degree and many nodes with a low degree. Albert et al. proposed that deleting these high degree vertices (hubs) partitions a power law graph in independent subsets [AJB00]. Later Petroni et al. designed the HDRF strategy [Pet+15] to partition power law graphs with remarkable results.

Inspired by these insights the Degree Aware strategy tries to gain a benefit from considering the vertex degree. Therefore, the strategy is designed as meta strategy which uses other strategies as parameters. It uses one strategy to handle low degree vertices and one for high degree vertices. The definition of high and low degree vertices is done via a parameter and based on the vertices seen at runtime. This means it is possible to define a percentage of $x\%$ which shall be considered as low degrees. While the graph

is streamed, the strategy creates a percentile list holding the received degrees and the number of their appearance. Based on this information the incoming vertices are either assigned by the high or the low degree strategy. When receiving the vertices sorted by their degree the strategy would 'crash' because all vertices would be handled as high/low degree vertices. However, this problem is not relevant because such orders would not be feasible and can therefore be neglected.

## 4.2 Restreaming Approaches

The potential of real time strategies is very limited because information is still gathered while assigning. Many vertex assignments are made uninformed especially the first ones suffer from the cold start problem[1]. Hence, a higher degree of freedom is necessary to achieve better partitionings. Consequently, the input data has to be accessed several times so the previously gained knowledge can be put to better use. When at the same time trying not to break the runtime boundaries of linear algorithms restreaming is the only option. Restreaming means the input data is read several times and the knowledge of previous streams is available. A simple example would be to assign every vertex when it is read and reassigning the vertex to a partition it fits better when reading it again in a later pass.

### 4.2.1 Two Pass

Taking the idea of Albert et al. [AJB00] and Petroni et al. [Pet+15] further it should be promising to have more knowledge about the high degree vertices and to be able to spread them over the partitions first. Therefore, a fair balancing can be obtained since the high degree vertices which could crash the partitioning at the end of the stream are already assigned. Furthermore, it offers the possibility to assign the remaining independent clusters properly.

The required information can only be gained by restreaming the graph. Therefore, the strategy streams the graph three times. Two of the streams are used for actual vertex placement giving the strategy its name. The first steam is for counting the number of vertices. This run could be avoided if the graph size (number of vertices) was known beforehand. But the assumptions of this thesis collide with this requirement. At the second stream a given percentage of the high degree vertices ($x\%$ of the vertices with the highest degree) are searched and afterwards distributed via SNTI 4.1.2. This ensures

[1]cold start problem: making bad decisions at the begin of the stream because of a lack of information

a fair distribution of the hubs which still tries to perform a good partitioning. The last stream assigns the missing low degree vertices based on the given strategy.

### 4.2.2 Vertex Correction

The lack of information at the beginning of the partitioning leads to suboptimal placements. All previous ideas suffer from the cold start problem (even two pass does not completely avoid the problem). The Vertex Correction strategy tries to compensate the problem by replacing vertices through a better informed placement later. The strategy streams the graph several times reassigning every vertex with every stream consulting the old placement.

The first stream uses the Greedy strategy for an initial placement. Every following pass creates a candidate set of partitions for every incoming vertex. A candidate partition is a partition with more than 90% edge intersection with the edges of the vertex. The maximal loaded partition (in terms of the number of edges) is no candidate because the strategy tries to shift load from this partition to maintain a good balancing. Finally, the vertex is assigned to the best candidate partition. Those iterations are made several times according to a parameter.

A disadvantage of this strategy is the high memory usage because the strategy has to keep two placements in memory.

### 4.2.3 Meta Restreaming

Another idea to avoid the cold start problem is a general restreaming. Therefore, the Meta Restreaming strategy was implemented. The strategy takes an arbitrary number of strategies and their parameters which are executed consecutively. From the second strategy on every vertex is deleted from the partitioning before it is reassigned. This is due to avoid a complete matching at the old partition which would not change the assignment in the end. Therefore, better results can emerge since the cold start problem can be minimized and the global knowledge can be used to improve the partitioning even further.

### 4.2.4 Fractional Restreaming

Restreaming tries to benefit from the knowledge of the previous passes. However, bad assignment decisions affect later passes. Fractional Restreaming forgets some gained

assignment information after each pass. Furthermore, Fractional Restreaming is a meta strategy which uses several other strategies. Those are defined via a parameter similar to the Meta Restreaming strategy.

**Data:** a hypergraph as vertex list, a percentage $p$ how many vertices shall be forgotten, a list of strategies with their parameters
**Result:** partitioning into $k$ parts
Set initial partitions $P_1, P_2, ..., P_k$ to be empty sets ;
**foreach** *strategy* $s$ **do**
  assign all vertices according to $s$;
  **if** *not the last strategy* **then**
    **foreach** *Partition* **do**
      delete $p\%$ of the vertices with $\min ExternalFitness$;
    **end**
  **end**
**end**
return $P_1, ..., P_k$

**Algorithm 3:** Fractional Restreaming

The rating which vertices are considered bad is done with a metric called '$ExternalFitnessScore$' (cf. Equation 4.3). The score determines how well the vertex could fit to any other partition. If the score is high the strategy will forget the vertex with high probability so it can be reassigned in the next pass. In each pass the $p\%$ with the highest score are forgotten. When reassigning the vertex is placed on the partition it fits best. As the $ExternalFitnessScore$ was quite high the probability the vertex is reassigned to a better location is also quite high.

$$ExternalFitness(v, P_i) = \frac{\max_{p \in P \setminus \{P_i\}} p.topics \cap v.topics}{degree(v)} \tag{4.3}$$

### 4.2.5 Topic Correction

Since the main metric in this work is the Cut Size, deleting vertices rather than hyper edges does not necessarily lead to better results. This is due to the fact that many hyper edges have a lot of vertices and the Cut Size metric can only improve if all vertices of an edge are removed and placed elsewhere.

The Topic Correction strategy (TC) uses Greedy for vertex assignment and takes a number of $passes$ as parameter determining how often the Topic Correction phase shall be executed. In the Topic Correction phase $q\%$ of the edges having the worst edge score

(Equation 4.4) and $w\%$ random edges are removed by forgetting all connected vertices. In the next pass all vertices are reassigned according to the Greedy strategy. When reassigning a vertex which was not deleted before, it has to be removed from the model right before the assignment. Otherwise the old location would have a perfect matching and the strategy would not be able to improve the placement of not deleted vertices. However the probability to reassign such a vertex to the old partition is quite high since the matching was not totally bad before.

$$edge_score(e,p) = \frac{edge_binding(e,p)}{\sum_{v_i \in P} edge_binding(e,p_i)} * \frac{size(e,p)}{max_edge_size} \tag{4.4}$$

$$edge_binding(e,p) = |e.vertices \cap p.vertices| \tag{4.5}$$

$$size(e) = |e.vertices| \tag{4.6}$$

$$max_edge_size = \max_{e \in E} size(e) \tag{4.7}$$

**Data:** a hypergraph as vertex list, a percentage $q$ for bad edges, a percentage $w$ for random removal, a number of $passes$ (internal it uses $passes + 1$ for initial assignment)
**Result:** partitioning into $k$ parts
Set initial partitions $P_1, P_2, ..., P_k$ to be empty sets ;
**foreach** *pass* **do**
  assign all vertices with Greedy;
  **if** *not the last pass* **then**
    **foreach** *Partition* **do**
      delete $q\%$ of the edges with $\min edge_score$;
      delete $w\%$ random edges;
    **end**
  **end**
**end**
return $P_1, ..., P_k$

**Algorithm 4:** Topic Correction

### 4.2.6 Topic Correction Narrow Scope

Topic Correction Narrow Scope strategy (TCNS) is an enhancement of the Topic Correction strategy. While TC had removed the same amount of topics in every single restream, TCNS removes less topics with each pass.

At the beginning of the stream the partitioning performs not best due to uninformed decisions. Hence, the result of the pass is not trustworthy. Consequently, most of the assignments will be removed and only the very best remain. In each following pass the partitioning becomes more trustworthy since the partitioning was made based on more information. As only the very best information was kept the information for the next pass should become better with each pass.

The number of topics to be removed is dependent on the pass ($passCounter$) and the number of passes in total ($numberOfPasses$). Equation 4.8 shows the interaction of these parameters to the percentage of edges which are removed. When implementing index shifts may be necessary if the pass counting starts with 0 (removing all edges after the first pass will help nothing). After the removal of the untrustworthy edges all vertices are restreamed. This way even the vertices which where not removed may be reassigned if another placement looks more promising.

$$percentage = 100 - \frac{passCounter}{numberOfPasses} * 100 \tag{4.8}$$

### 4.2.7 Adaptive Balance

Inspired by the idea of changing a parameter over the passes the Adaptive Balance strategy was invented. It adjusts the balance for each pass allowing big imbalance at the first passes and becoming more strict at the later passes.

The strategy takes three parameters. At first the allowed imbalance at the last pass named $\lambda_{last}$. Then, the allowed imbalance at the first pass ($\lambda_{first}$). Finally, the number of passes to be made ($k$). Since the strategy aims to a specific percentage of imbalance the GTI strategy is used for the vertex assignment. Therefore, the allowed imbalances are given in the same way as the balance in GTI: $(\lambda \in \mathbb{R} | \lambda \geq 1)$. The balancing ($\lambda$) for the passes is calculated with Equation 4.9. The decremented $k$ is due to the fact that the pass counting starts with 0 and therefore the maximal value $i$ can reach is $k - 1$. However, it is necessary that $\lambda$ is in $[\lambda_{first}, \lambda_{last}]$ and reaches both marginal values.

$$\lambda(i) = \lambda_{last} + \frac{k - 1 - i}{k - 1} * (\lambda_{first} - \lambda_{last}) \tag{4.9}$$

# 5 Evaluation

This chapter covers the evaluation of the previously introduced strategies. For evaluating two systems were used. A server system and a notebook. The specifications can be seen in Table 5.1. Hence, the execution time evaluations should only be compared with other evaluations within the same graphic. For the other benchmarks the choice of the system is not relevant since the evaluations are deterministic with the exception of the randomized algorithms.

For evaluating the strategies several hypergraphs have been used. All data sets are available online but were modified to match the needs of this thesis (e.g. reading the input as a vertex list). The MovieLens graph occurs in two versions: the small mlSmall graph and the large ml20m graph [Gro]. Both graphs were created by GroupLens Research from collected movie ratings. Users represent items and the movies they rated (independent from the rating itself) are their topics. The Amazon graph comes from the Stanford Large Network Dataset Collection which is part of Stanford Network Analysis Project [Les]. Since the original data set had a lot of information it was modified for this thesis. It now represents the articles (items) and the customer who rated them (topics). The rating itself was not considered. The Book-Crossing (Book-X) contains book ratings collected by Cai-Nicolas Ziegler from the Book-Crossing community [Zie]. The user (items) who rated a book (topic) were represented in the resulting graph. Again the rating itself was not considered. Table 5.2 provides a short survey.

**Table 5.1:** Systems running the evaluations

| System | cores | clock rate | RAM | OS |
|---|---|---|---|---|
| **Server** | 32 | 2,3 GHz | 280 GB | CentOs 6.8 |
| **Notebook** | 2 | 2,3 GHz | 8 GB | Windows 10 |

**Table 5.2:** Hypergraphs used for evaluating

| Graph name | #Vertices | #Hyperedges | Source |
|---|---|---|---|
| mlSmall | 670 | 9 066 | [Gro] |
| ml20m | 138 492 | 26 744 | [Gro] |
| Amazon | 402 723 | 1 555 170 | [Les] |
| Book-X | 105 282 | 340 552 | [Zie] |

## 5.1 Evaluation Survey

A comparison of all introduced strategies will be the subject of the following section. The Figures 5.1 to 5.3 compare several real time strategies on the Amazon graph. The strategies parameters are displayed in the graphics in square brackets and runtimes are given in milliseconds.

Except for the Greedy strategy the strategies presented [AIV15] perform bad especially on the Cut Size metric. However, the optimization goal in the paper was the Max Edge metric [AIV15]. Yet even on this metric, their performance is not compelling. The same results can be seen when looking at the Figures 5.4 and 5.5 presenting the Cut Size for the MovieLens graphs. Hence, most of them are not considered as benchmarks furthermore.

The Greedy and Never Max strategy show mostly even results considering the Cut Size and Max Edge. Greedy provides minimal better results. Therefore, the additional computation overhead of the Never Max strategy does not pay off in the direct comparison.

When comparing Greedy with GTI the results are as expected. Due to a more lax balancing the GTI strategy has a better Cut Size (cf. Figure 5.1). On the MovieLens graphs GTI is only slightly better than Greedy. Considering the Max Edge GTI is just minimal better and they can be called even. The improvement of the Cut Size comes besides the lax balancing with additional costs in the execution time

Additionally to the strategies evaluated in the Figures 5.1 to 5.3, the remaining strategies are evaluated in Figures 5.6 to 5.12. In the following the strategies performance will be discussed in detail.

The Two Pass strategy sticks up in Figure 5.6 since the Cut Size is much worse than the other strategies evaluated on Book-X. The bad performance also applies on the Amazon graph (cf. Figure 5.7). When partitioning the MovieLens 20m graph shown in Figure 5.8, Two Pass performs about as well as some other strategies. This is due to the fact that Two Pass only restreams the graph three times and assigns the vertices only once. Hence, the benefits of restreaming are hardly exploited. On the other hand this

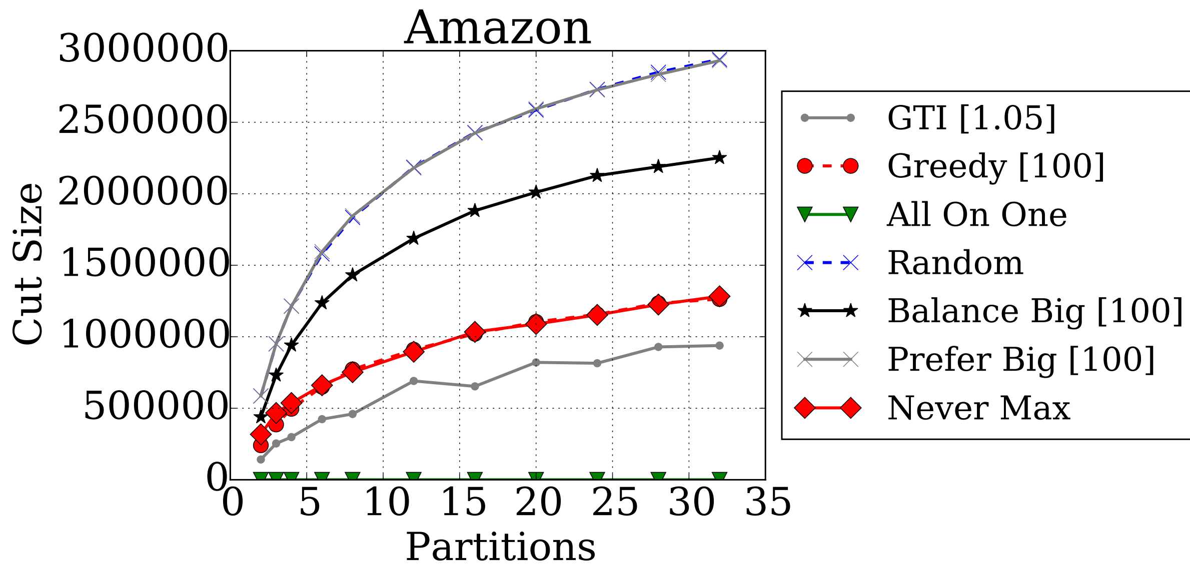


**Figure 5.1:** Survey I on the Amazon graph considering the Cut Size

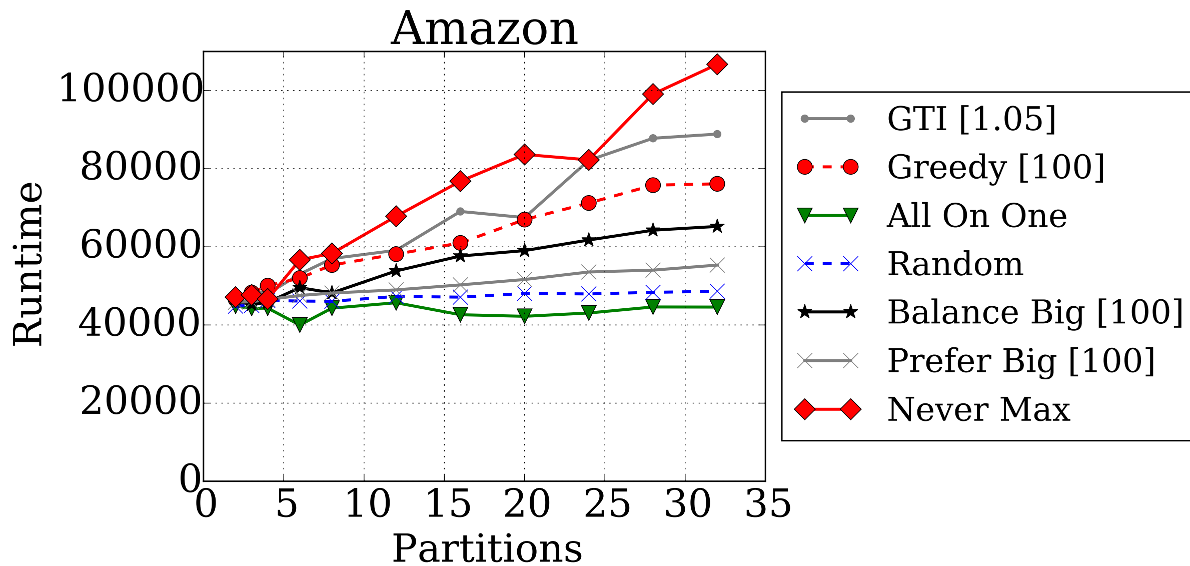


**Figure 5.2:** Survey I on the Amazon graph considering the execution time

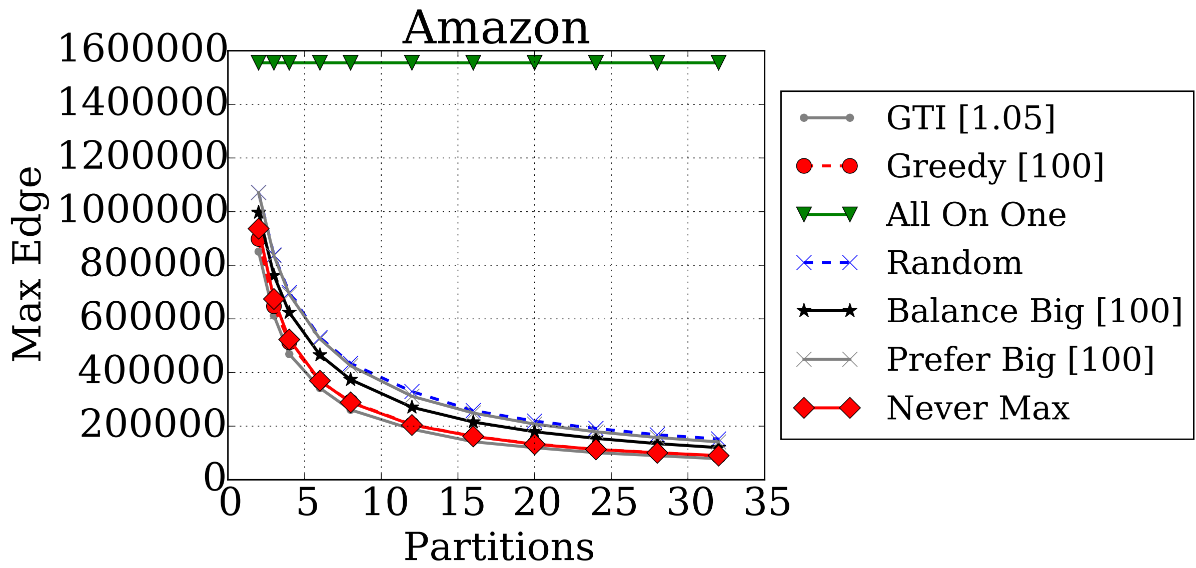


**Figure 5.3:** Survey I on the Amazon graph considering the Max Edge metric

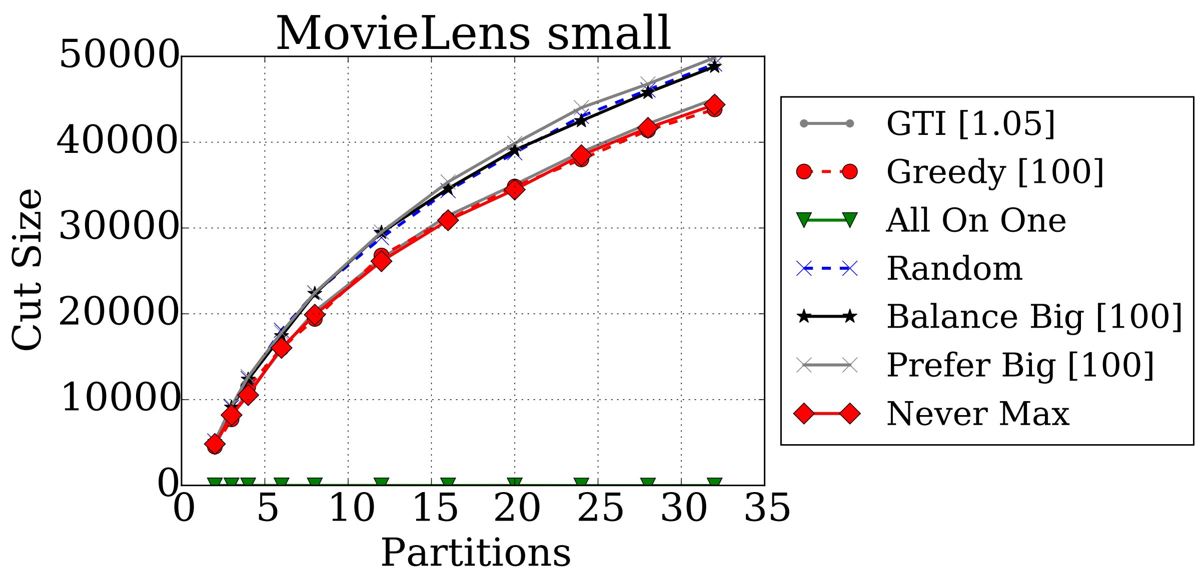


**Figure 5.4:** Survey I on the small MovieLens graph considering the Cut Size

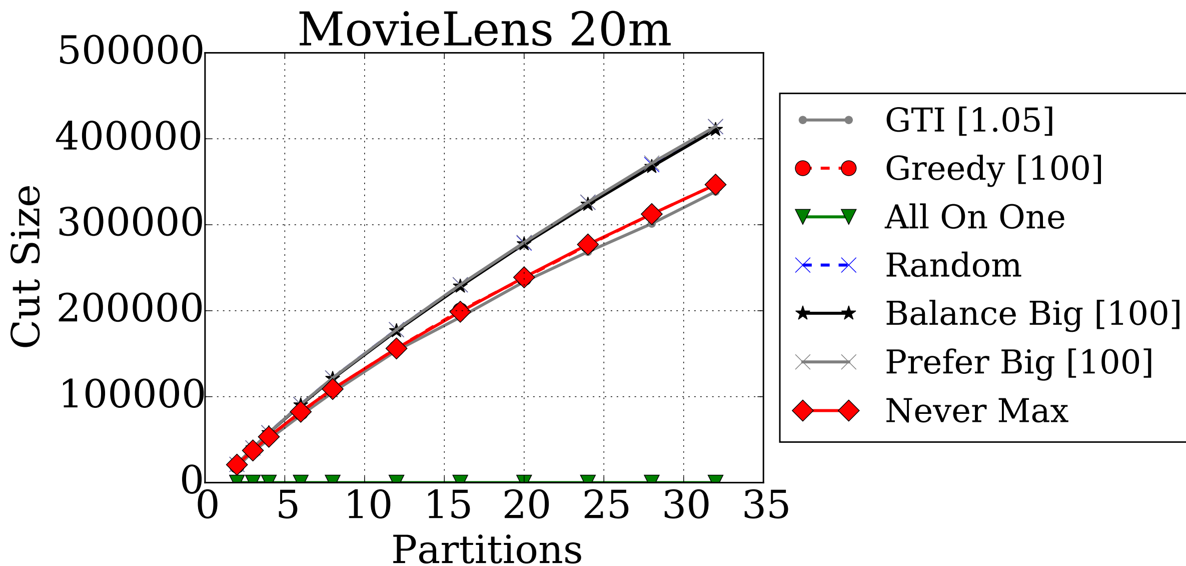


**Figure 5.5:** Survey I on the MovieLens 20m graph considering the Cut Size

limitation of streams benefits the execution time. Like Two Pass the Topic Correction strategy performs quite bad on the Amazon graph (cf. Figure 5.7). On the other graphs Topic Correction also performs not well but not as worse. Since Two Pass and Topic Correction performs way worse than GTI and take more execution time than GTI the strategies are not feasible.

Fractional Restreaming and Vertex Correction perform most of the time as well as GTI while using restreaming. On the MovieLens graph (cf. Figure 5.8) Topic Correction seems to outperform every other strategy. This is due to an invalid balancing which can be seen by the resulting Max Edge in Figure 5.10. Thus, Fractional Restreaming and Vertex Correction seem also not feasible since the computational overhead did not pay off in this evaluation.

In contrast to most other strategies in Figures 5.6 to 5.12 Degree Aware is no restreaming strategy. Nevertheless it performs surprisingly well and even outperforms some of the worse performing restreaming strategies. The Degree Aware strategy will be further discussed in Section 5.2.3.

Meanwhile Meta Restreaming and Adaptive Balance outperform all other evaluated restreaming strategies on Book-X, Amazon and MovieLens 20m in terms of the Cut Size. Considering the Max Edge metric they also clearly outperform the other strategies on the MovieLens 20m graph (cf. Figure 5.10) and perform slightly better on the Book-X graph (cf. Figure 5.9). The results of the Max Edge metric on the Amazon graph had been the same as on the Book-X Graph for all strategies and consequently the graphics are not listed. Meanwhile the execution time of Meta Restreaming and Adaptive Balance is

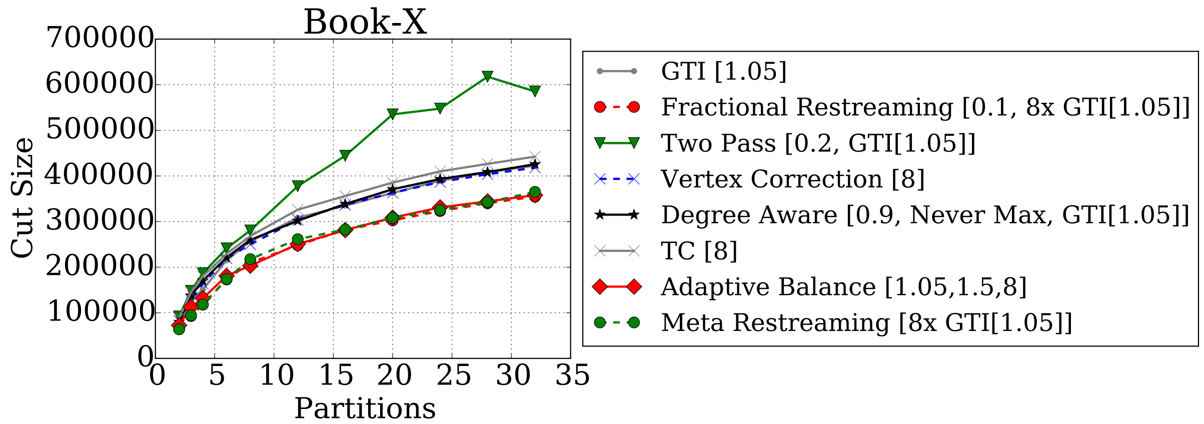


**Figure 5.6:** Survey II on the Book-X graph considering the Cut Size

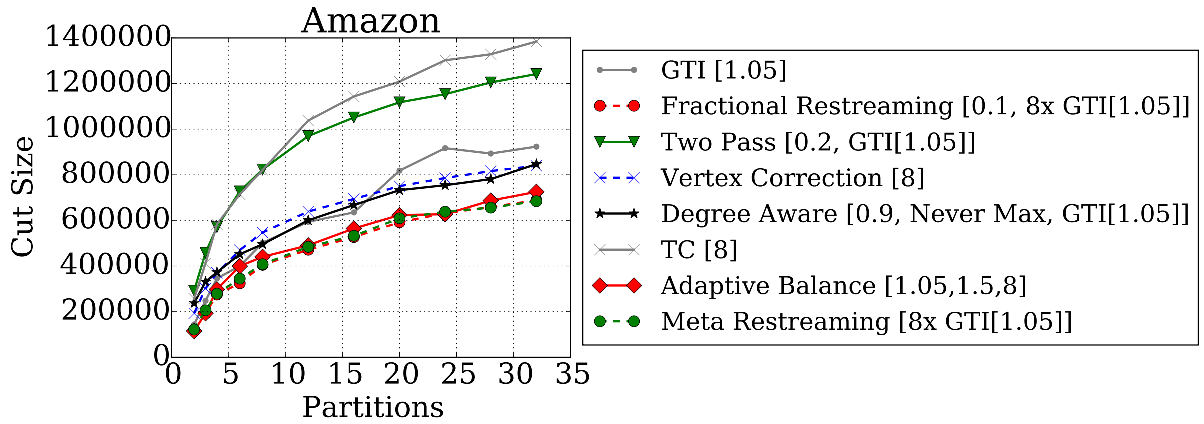


**Figure 5.7:** Survey II on the Amazon graph considering the Cut Size

within the span of the other restreaming strategies. So when using these two strategies the computational overhead of restreaming seems to pay off. The two strategies will be further investigated in Sections 5.3.1 and 5.3.3.

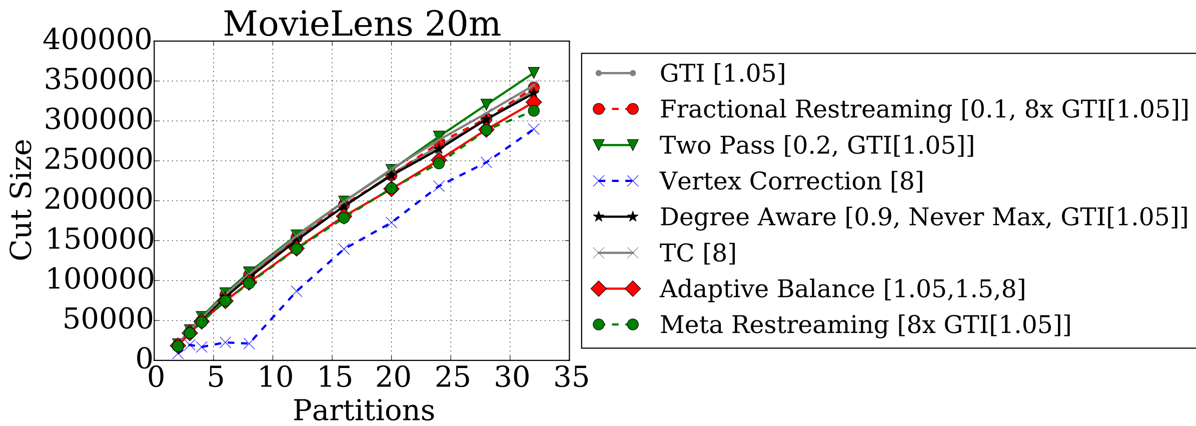


**Figure 5.8:** Survey II on the MovieLens 20m graph considering the Cut Size

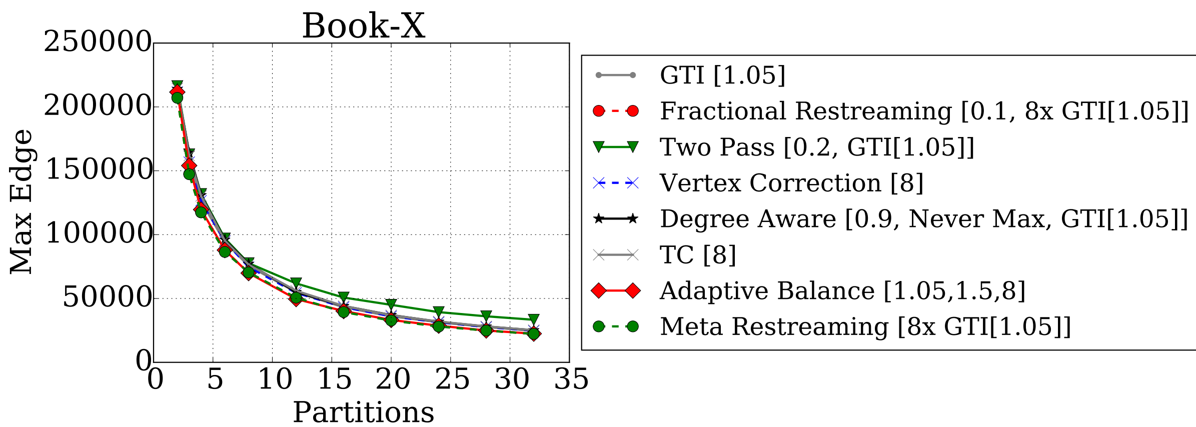


**Figure 5.9:** Survey II on the Book-X graph considering the Max Edge

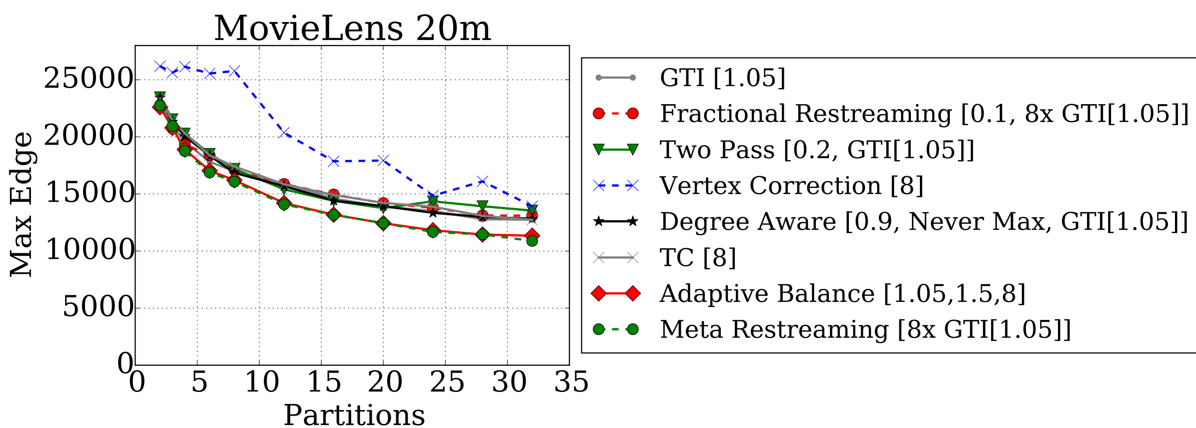


**Figure 5.10:** Survey II on the MovieLens 20m graph considering the Max Edge

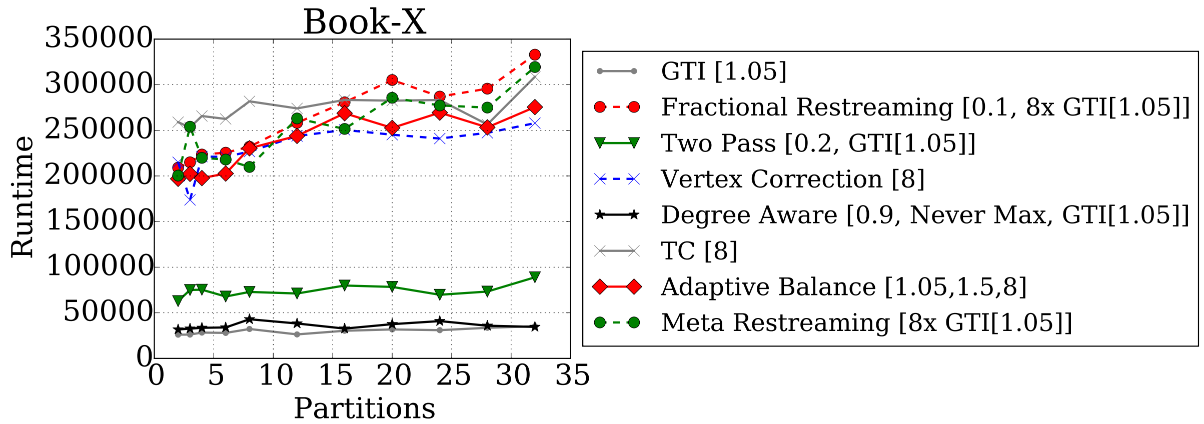


**Figure 5.11:** Survey II on the Book-X graph considering the execution time

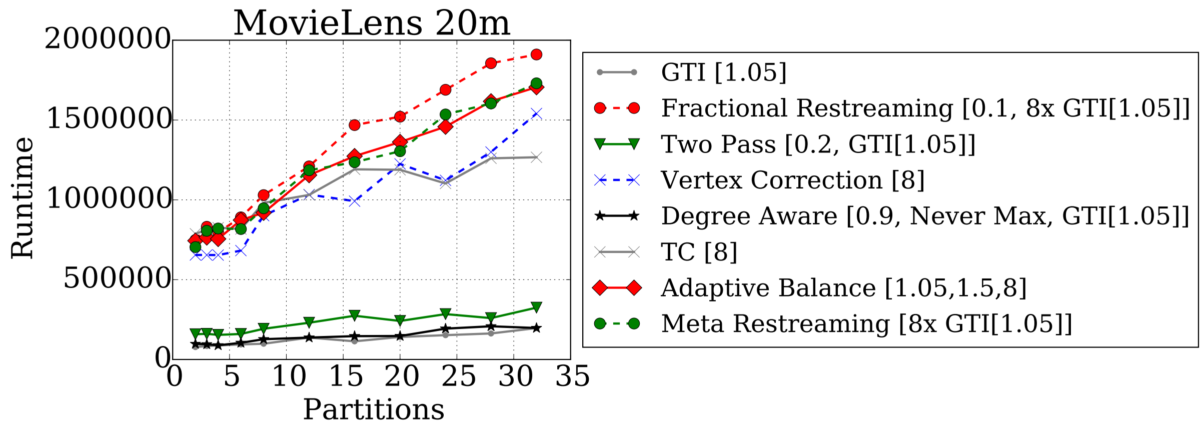


**Figure 5.12:** Survey II on the MovieLens 20m graph considering the execution time

## 5.2 Real Time Strategy Evaluation

The real time strategies which had promising results in Section 5.1 are evaluated in more detail in the following section. These are GTI, SNTI and the Degree Aware strategy in particular. In case of GTI the graph size independence is shown and some parameter tuning performed. The SNTI section will cover the parameter independence. The potential of the Degree Aware strategy will be touched in the last section.

### 5.2.1 Greatest Topic Intersection

In the following the graph size independence and balancing of the GTI strategy will be shown. Figures 5.13 to 5.15 show the impact of GTI's balancing parameter on the

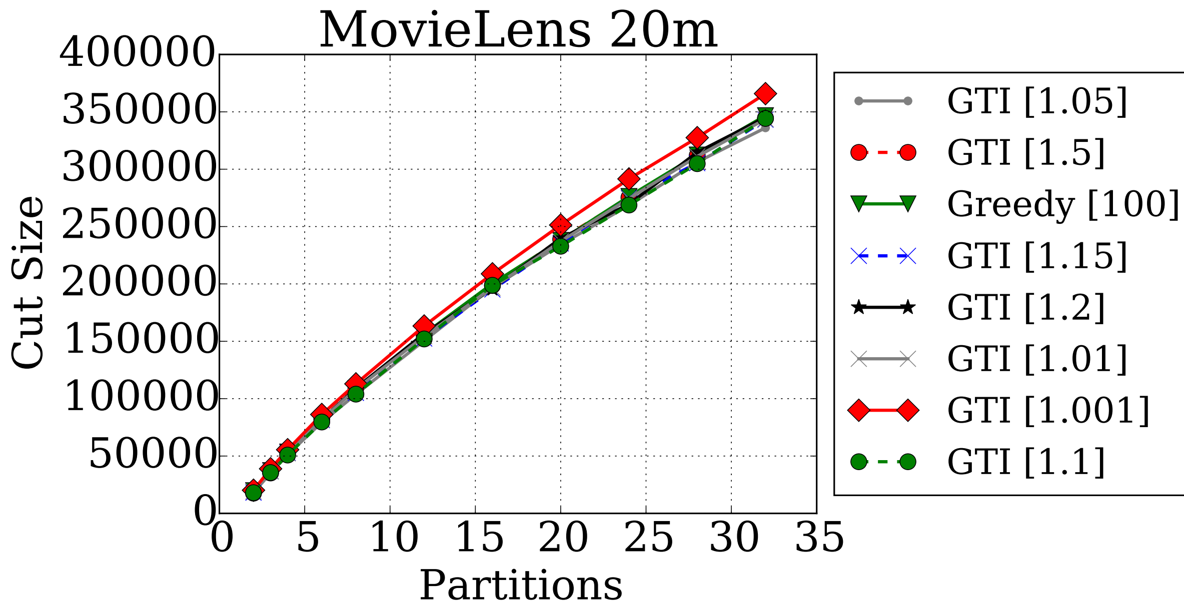


**Figure 5.13:** GTI's Cut Size with different parameters on the ml20m graph

Cut Size. On the MovieLens 20m graph the balancing factor has nearly no impact. This can be explained with the graph itself. The MovieLen 20m Graph (Figure 5.13) has a lot of items with a huge amount of topics. As soon as such a hub is assigned, the partition is imbalanced and not considered for a while. Thereby it does not make a huge difference if the balancing constraint says 5% or 50% allowed imbalance. On the Amazon graph (Figure 5.14) the Cut Size perfectly scales with the allowed imbalance as it is expected by the trade off. Figure 5.15 shows that GTI has some problems creating balanced partitionings for very small graphs. For more than 8 partitions the Cut Size does not change. This is due to the fact that not more than 7 partitions are loaded. So the balancing can not be met native for such small graphs. However, the Cut Size acts normal and the balancing is met when looking at Figure 5.16 where the border of GTI's balancing constraint was set to 50 (from 100 before). Therefore, the 5% imbalance seems to be a valid parameter choice for having been compared to other approaches in Section 5.1.

### 5.2.2 Smallest Non Topic Intersection

The parameter independence of SNTI will be evaluated in this section. Furthermore, the results of SNTI will be compared to Greedy. The Figures 5.17 to 5.19 show the Cut Size on different graphs. In general SNTI's Cut Size is slightly worse than Greedy's. This can be explained by the implicit hashing SNTI is doing at at the beginning of the partitioning.

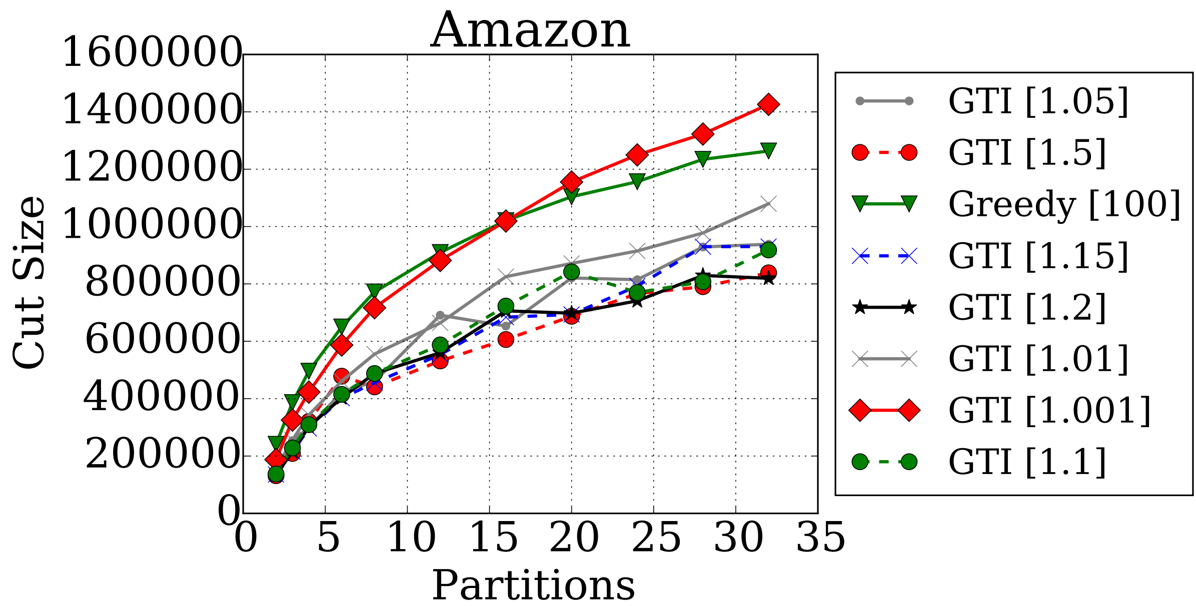


**Figure 5.14:** GTI's Cut Size with different parameters on the Amazon graph

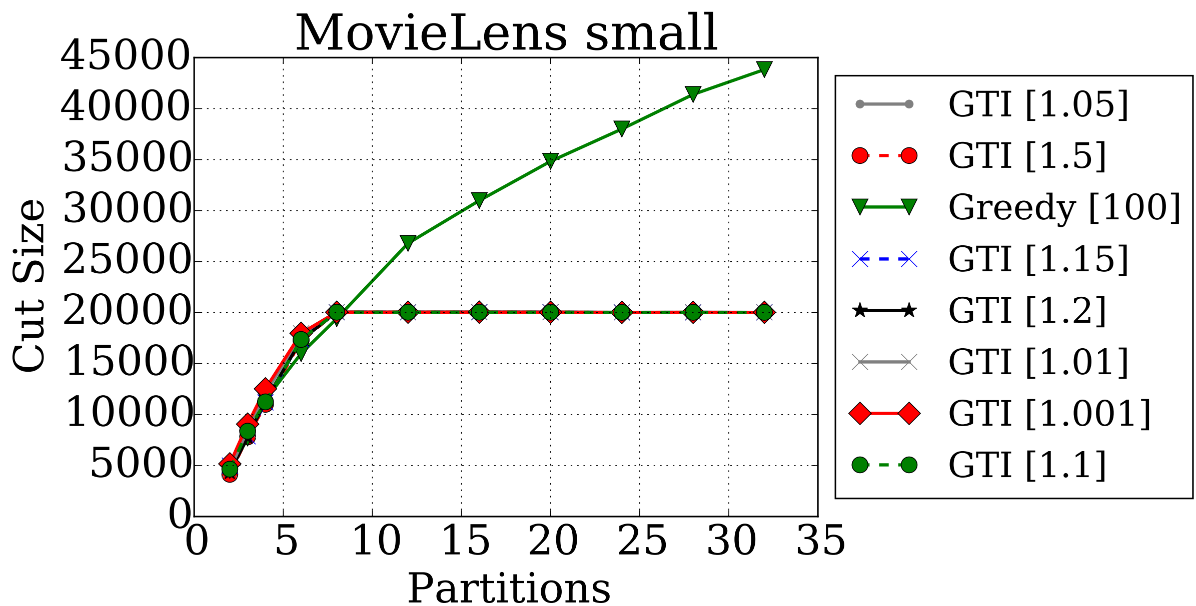


**Figure 5.15:** GTI's Cut Size with different parameters on the mlSmall graph

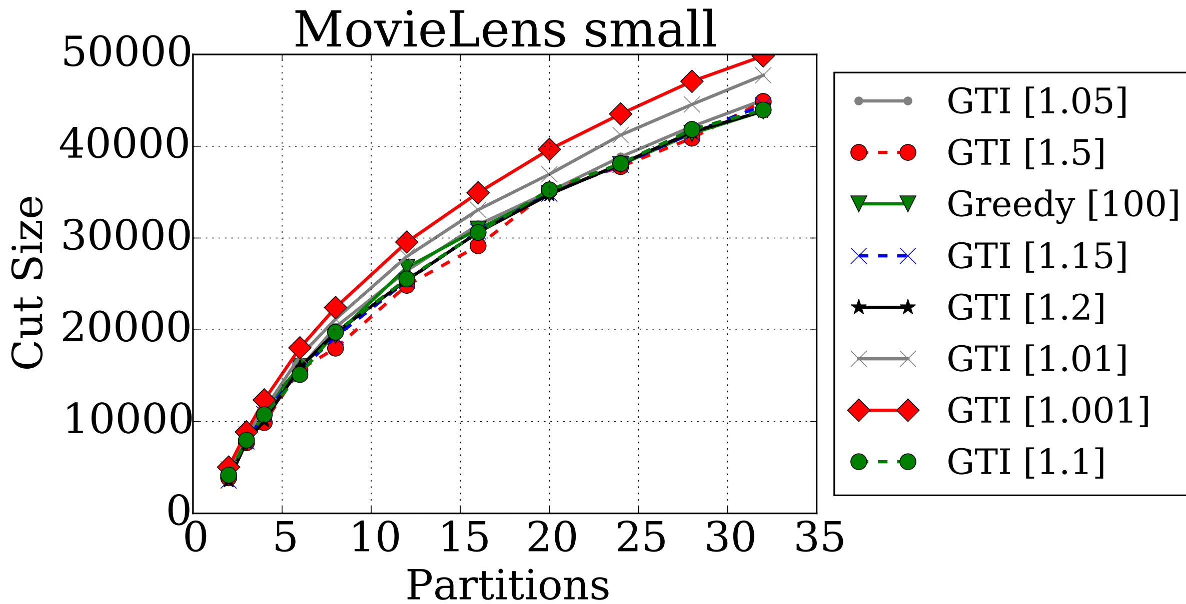


**Figure 5.16:** GTI's Cut Size with different parameters on the mlSmall graph when setting the balance constraint border to 50

Considering the Max Edge metric shown in Figure 5.20 on the ml20m graph (the other graphs showed similar resultls) both strategies can be regarded as equal. However SNTI needs more computation time (cp. Figure 5.21). Again the other graphs showed similar results and are not listed for that reason.

### 5.2.3 Degree Aware

Even though the degree aware strategy has not become the focus of the thesis since restreaming seemed more promising, it is evaluated against other real time strategies because its results had been quite good. With Degree Aware being a real time Meta Strategy it can benefit from the additional complexity while still maintaining a very short runtime. For revision: Degree Aware uses two strategies trying to give special attention to hubs. It was evaluated with GTI and Greedy while using the Never Max strategy based on Greedy for low degree vertices and GTI for high degree vertices with the percentile being set to 0.9.

Figures 5.22 and 5.23 show the evaluation on the Book-X graph. In terms of the Cut Size they perform almost equally with GTI being slightly better and Greedy being slightly worse. The Max Edge metric is also nearly equal which also had been the case in other evaluations.

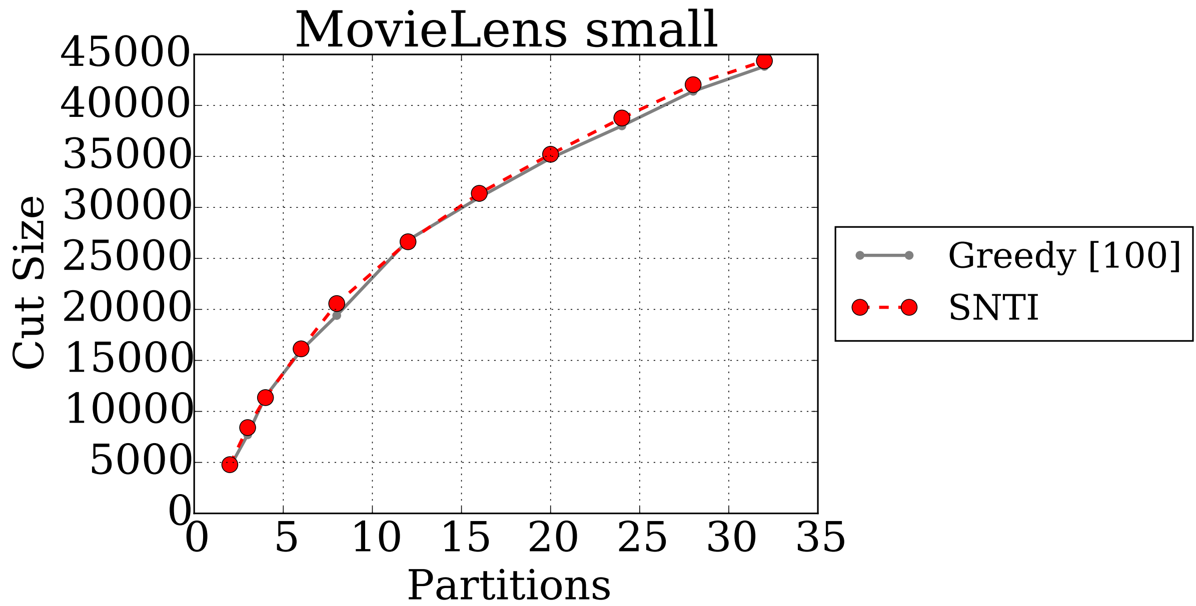


**Figure 5.17:** Cut Size on mlSmall, SNTI vs Greedy

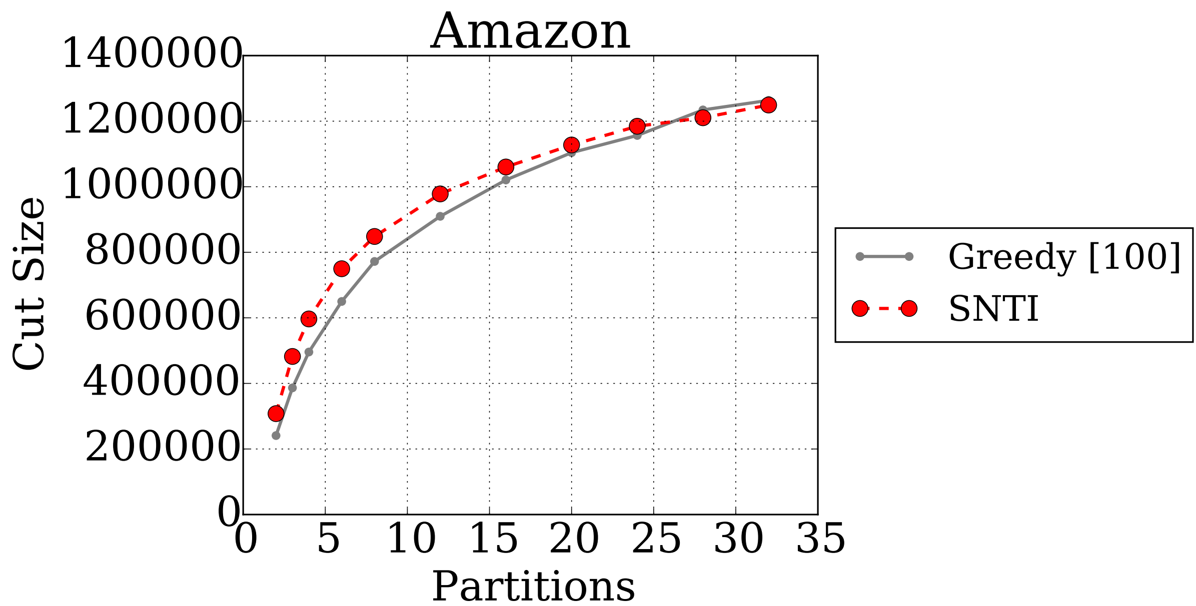


**Figure 5.18:** Cut Size on amazon, SNTI vs Greedy

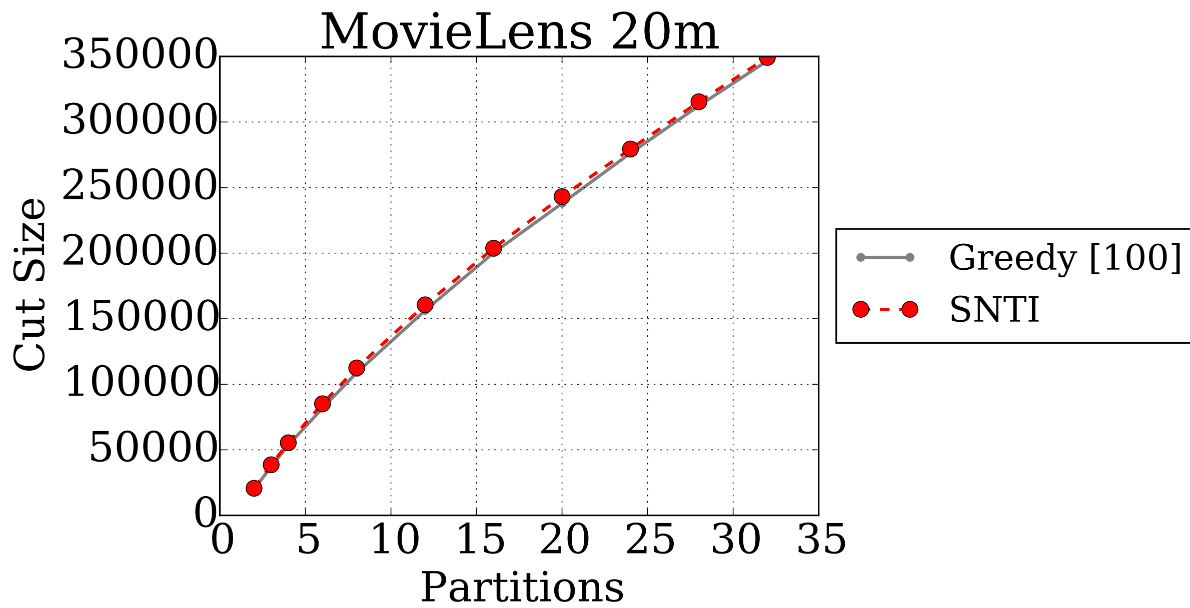


**Figure 5.19:** Cut Size on ml20m, SNTI vs Greedy

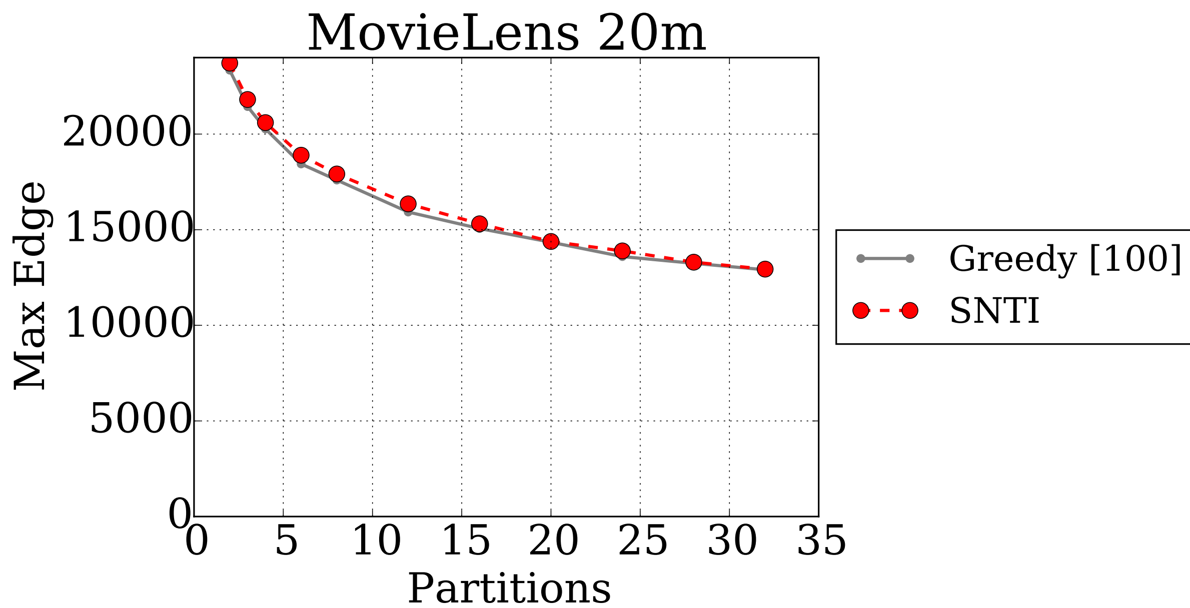


**Figure 5.20:** Max Edge on ml20m, SNTI vs Greedy

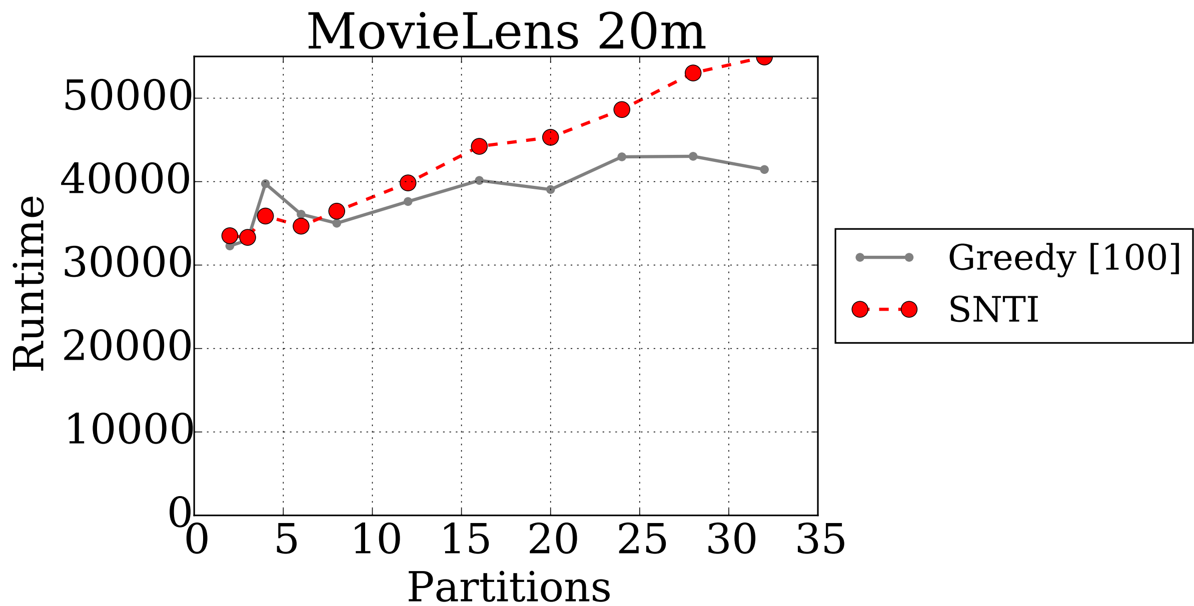


**Figure 5.21:** Execution Time on ml20m, SNTI vs Greedy

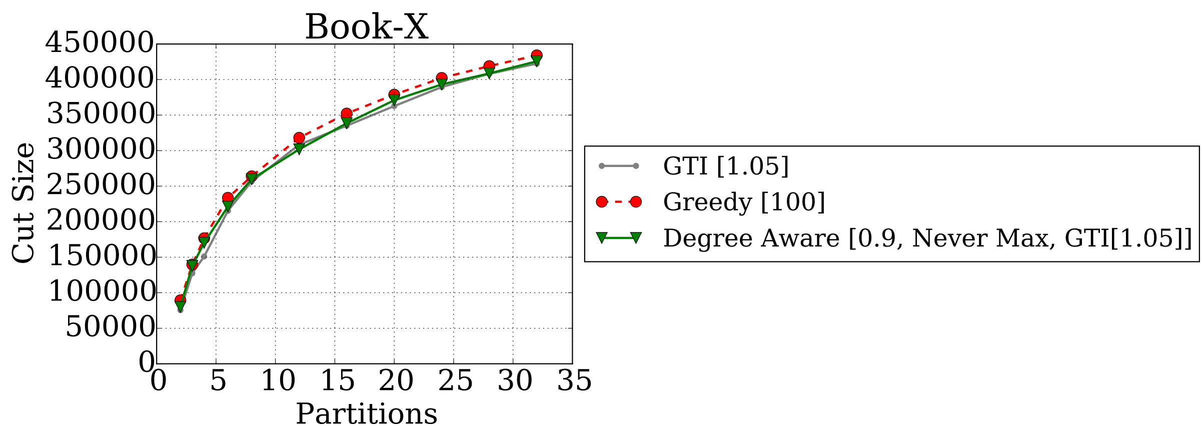


**Figure 5.22:** Cut Size of Degree Aware vs Greedy & GTI on Book-X

It is very interesting to investigate the Cut Size when looking at the Amazon graph seen in Figure 5.24. GTI outperforms Greedy by far but is kind of unstable. With 20 and 24 partitions the Cut Size increases way more than at other points and having a spike downwards at 6 partitions. However, GTI's Cut Size is all the way through better than the one performed by Greedy. When looking at the Degree Aware strategy it is revealed that it performs without such spikes and outperforms GTI with higher number of partitions. This is especially remarkable since most of the placement logic is done via Greedy and only a part of it is done with GTI. However, this cooperation results in a Cut Size which can be compared with GTI while having a more strict balancing for most

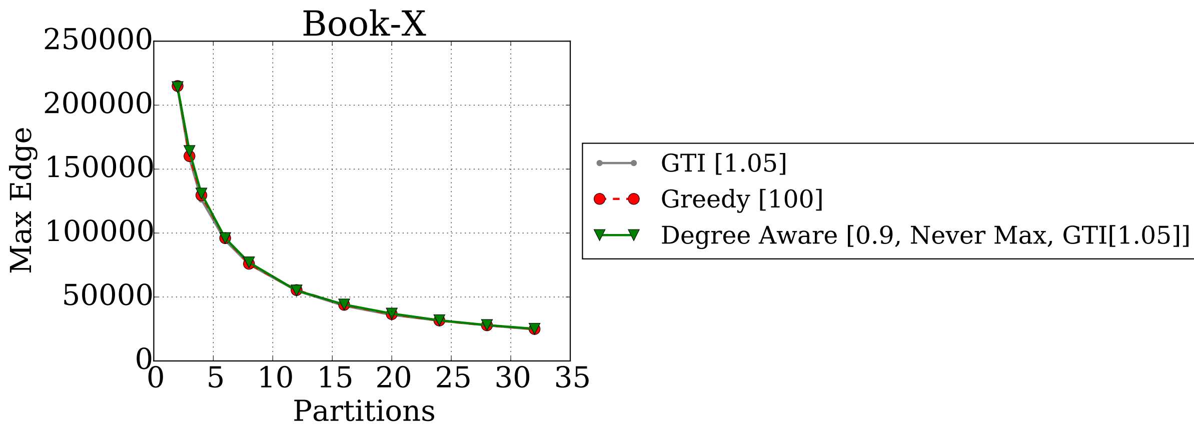


**Figure 5.23:** Max Edge of Degree Aware vs Greedy & GTI on Book-X

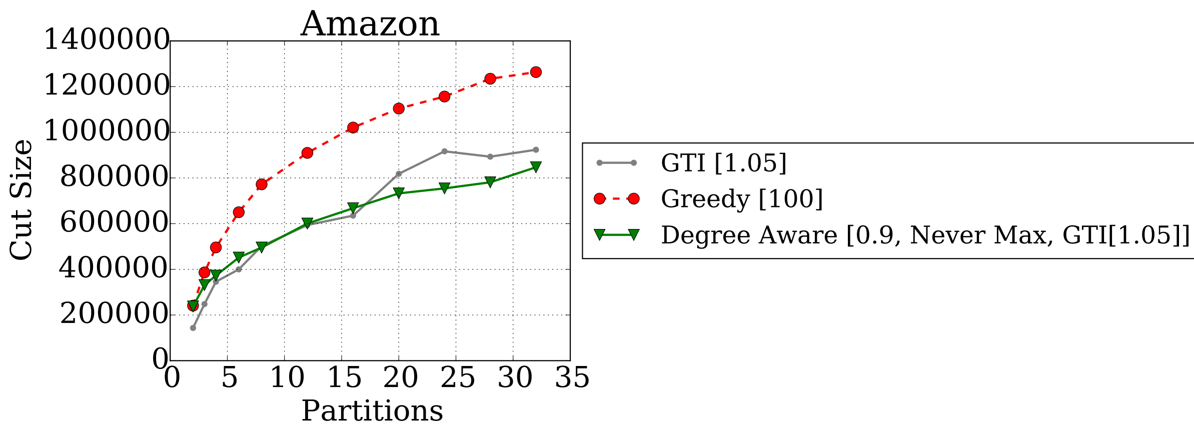


**Figure 5.24:** Cut Size of Degree Aware vs Greedy & GTI on amazon

vertices provided from Greedy. Furthermore, the spikes from GTI are avoided but also more positive spikes are missed.

Figure 5.25 shows a runtime comparison of the strategies. As expected Greedy and GTI are almost equal with Greedy being a little bit faster. Moreover, it is no surprise that the execution time of the Degree Aware strategy is a bit longer and increases a bit faster. Nevertheless, Degree Aware is a real time strategy outperforming the runtime of restreaming strategies.

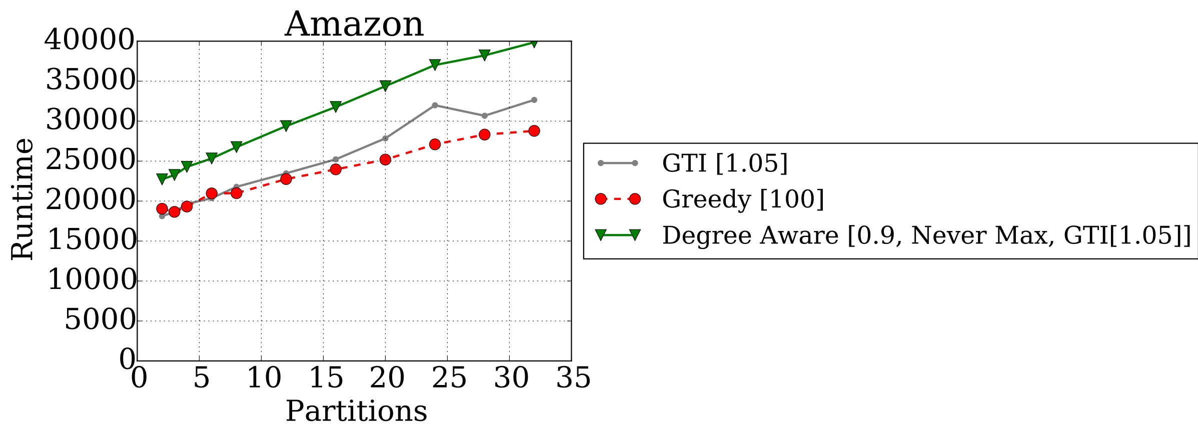


**Figure 5.25:** Execution Time of Degree Aware vs Greedy & GTI on amazon

## 5.3 Restreaming Strategy Evaluation

Real time strategies have an insufficient Cut Size due to a very limited degree of freedom. Hence, strategies with more potential have to be considered. Restreaming strategies are able to exploit more information from the data streams. Their performance will be evaluated in the following sections.

### 5.3.1 General Restreaming

In Section 5.1 some restreaming approaches stood out. That is the reason why the basic idea of restreaming will be further investigated in the following. Therefore, the real time strategies Greedy and GTI will be evaluated against the Meta Restreaming running Greedy/GTI five and eight times.

On the MovieLens 20m graph the restreaming improves the results by a few percent (cf. Figure 5.29). It is remarkable that restreaming Greedy five times just slightly outperforms the results of one GTI run. In contrast to the Book-X graph (cf. Figure 5.26) restreaming has a more positive impact on the Cut Size (up to 15%). However, it should be mentioned that using eight restreams instead of five improves the outcome only slightly. Considering the Max Edge, restreaming Greedy ends up in better results while when restreaming GTI the Max Edge seems to be nearly equal to the real time strategy. Though, the difference between the Max Edge evaluations in Figure 5.27 are minimal so they could be regarded as equal (even the Greedy runs). However, the additional runtime of restreaming is quite heavy. It is notable that the runtime of restreaming GTI increases way more with more partitions than the one of restreaming Greedy. This is due to the fact that GTI's runtime increases faster than Greedy's (cf. Figure 5.28).

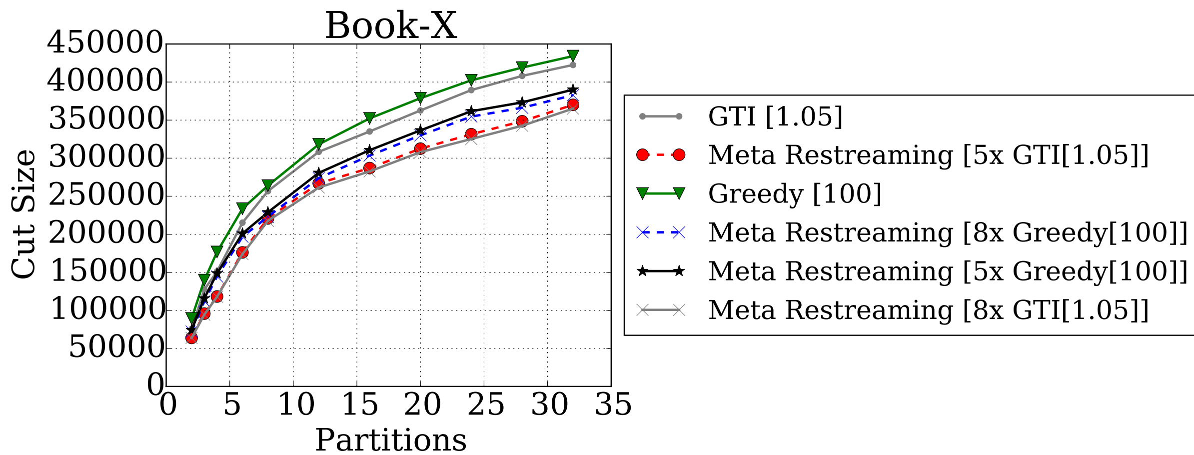


**Figure 5.26:** Cut Size of restreaming vs real time on the Book-X graph

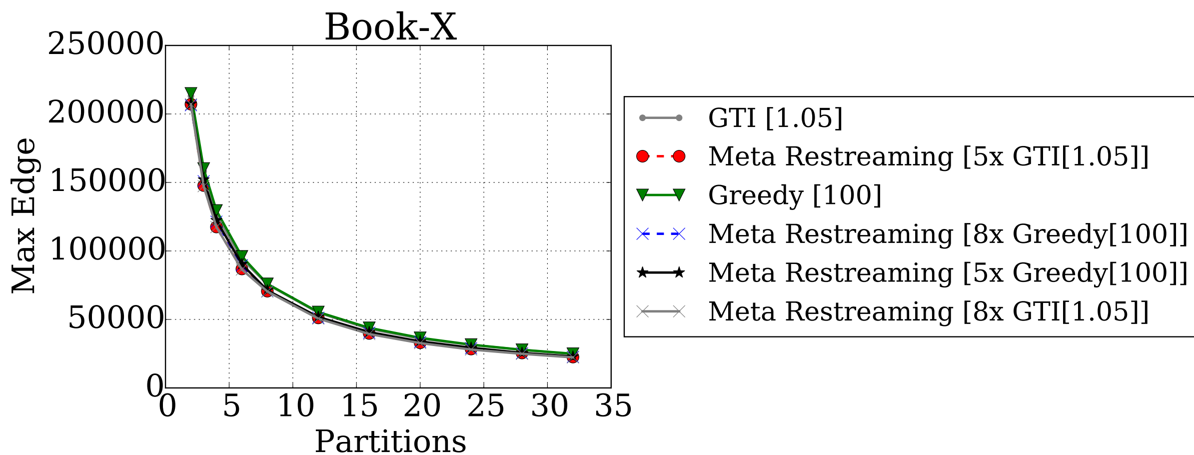


**Figure 5.27:** Max Edge of restreaming vs real time on the Book-X graph

With several passes this delta adds up of course. The increased runtime at the first few partitionings can be explained with scheduling. The evaluation was made on the notebook system which was used simultaneously.

### 5.3.2 Partial Forgetting

In contrast to the Meta Restreaming multiple restreaming strategies in Chapter 4 are based on the idea of forgetting pieces of the assignment. The idea of forgetting bad assignments to avoid bad decisions based on bad previous assignment decisions will be evaluated in the following section. All strategies following this basic idea can be summarized as 'Partial Forgetting strategies'. Thereby, the different realizations of the

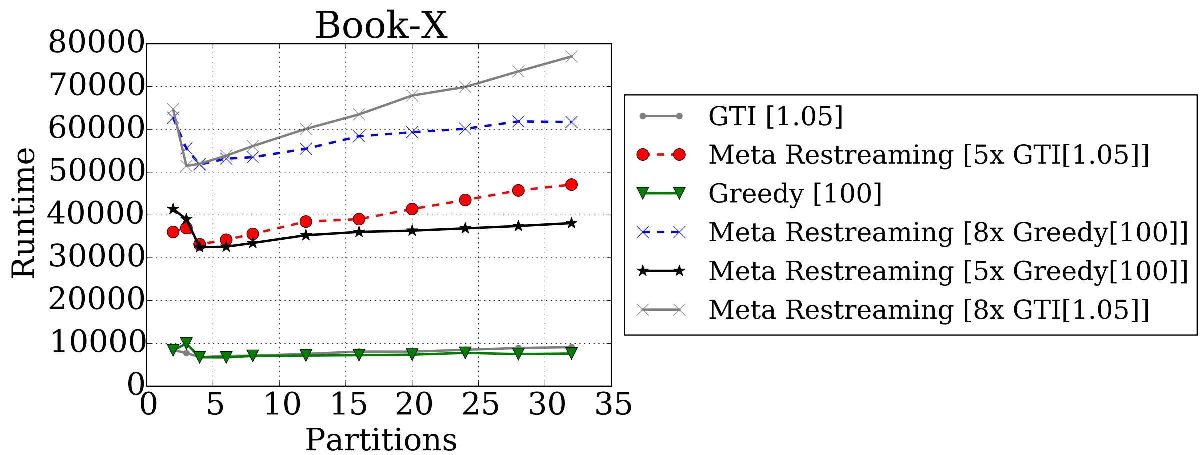


**Figure 5.28:** Execution Time of restreaming vs real time on the Book-X graph

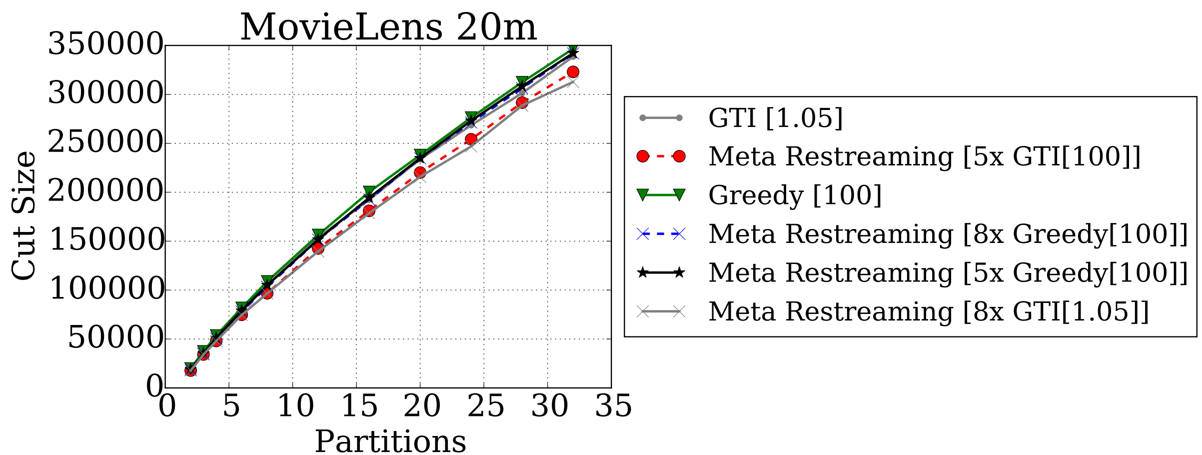


**Figure 5.29:** Cut Size of restreaming vs real time on the MovieLens 20m graph

Partial Forgetting strategies introduced in this thesis will be compared to each other. Further Topic Correction will also be evaluated even if it is no Partial Forgetting strategy because the basic idea is quite similar.

#### Fractional Restreaming

At first only the Fractional Restreaming will be evaluated to find a good parameter configuration (percentage to forget) to be comparable to the remaining strategies. For evaluating Fractional Restreaming runs on Book-X with five restreams forgetting 10%, 30%, 50%, 70% and 90% of the placement. Figure 5.30 reveals that the less of the placement is forgotten the better the Cut Size becomes. Comparing the Max Edge or the

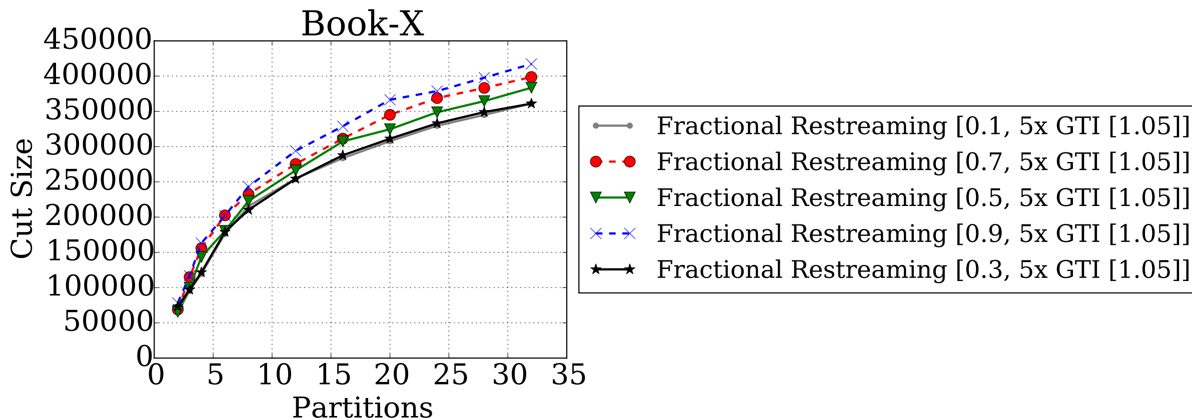


**Figure 5.30:** Cut Size of Fractional Restreaming with different parameters on Book-X

execution time was not illuminating since the configurations did not produce significant differences. Consequently these evaluations are not displayed here. For the further evaluations the Fractional Restreaming forgetting 10% will be used since it has the best results (delta to 30% is quite minimal and therefore it is hard to see the data points in Figure 5.30).

### Survey on Partial Forgetting

The Figures 5.31 and 5.32 provide a comparison of the strategies which forget assignments and Meta Restreaming as benchmark. Thereby Meta Restreaming and Fractional Restreaming use Greedy as basic strategy to provide a better comparability since TC, TCNS and Vertex Correction all internally use Greedy for vertex assignment.

TC, TCNS and Vertex Correction are outperformed by Meta Restreaming. When looking at the Max Edge (cf. Figure 5.32), they are also slightly outperformed by Meta Restreaming. This means the evaluated partitionings from TC, TCNS and Vertex Correction are truly worse in the evaluated environment than a simple restreaming approach.

Fractional Restreaming however can hold up to the performance of Meta Restreaming and produces even minimal better results. Thereby the parameter configuration is crucial as was shown in Section 5.3.2. In terms of the Max Edge metric both strategies produce equal results. The improved Cut Size is bought with additional runtime. Figure 5.33 provides a survey of the runtime of the Partial Forgetting strategies. Meta Restreaming has some spikes which can be explained with the simultaneous usage of the hardware. Apart from that it can be seen that the Meta Restreaming has a better runtime than all Partial Forgetting strategies and scales better with a higher number of partitions. Consequently a consideration has to be made if the additional runtime needed by

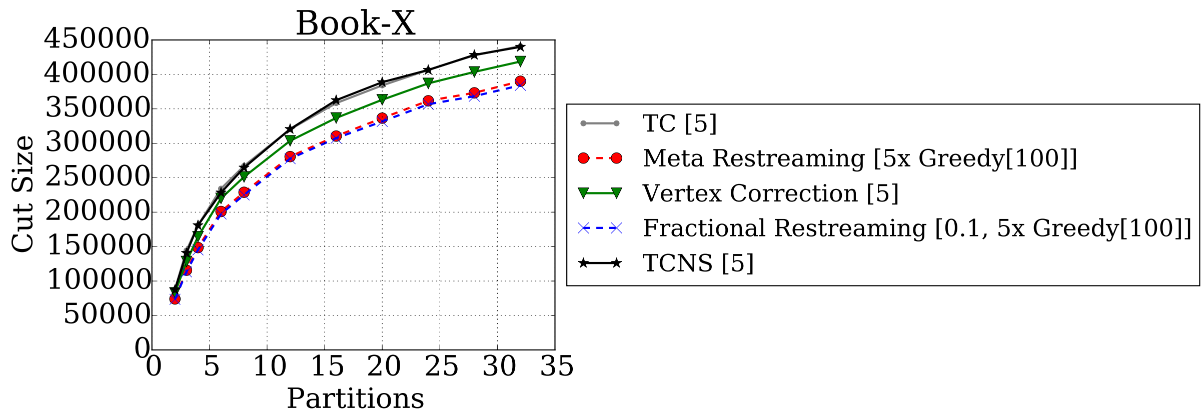


**Figure 5.31:** Cut Size of Partial Forgetting strategies vs Meta Restreaming on Book-X

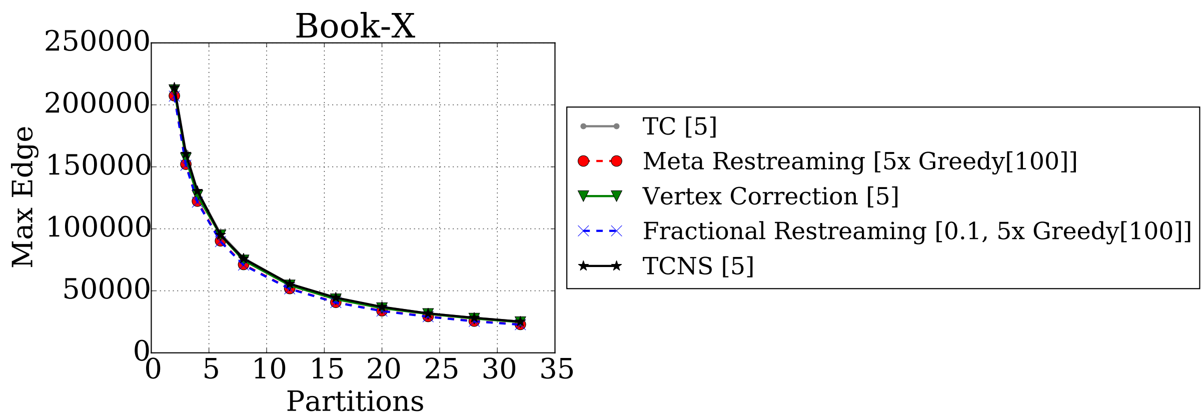


**Figure 5.32:** Max Edge of Partial Forgetting strategies vs Meta Restreaming on Book-X

Fractional Restreaming is worth the minimal benefit in the Cut Size compared to Meta Restreaming. Exploiting the idea of Partial Forgetting might be a promising field of study for future work.

### 5.3.3 Adaptive Balancing

Bad assignments can prevent good assignments later because the good assignments might hurt the balancing. To avoid these the bad assignments could be forgotten as in Section 5.3.2. Another approach would be to allow the bad assignments but prevent them from having such an impact on good assignments. This could be done by a more lax balancing. The Adaptive Balancing strategy implements this idea. In the following the results are evaluated and discussed.

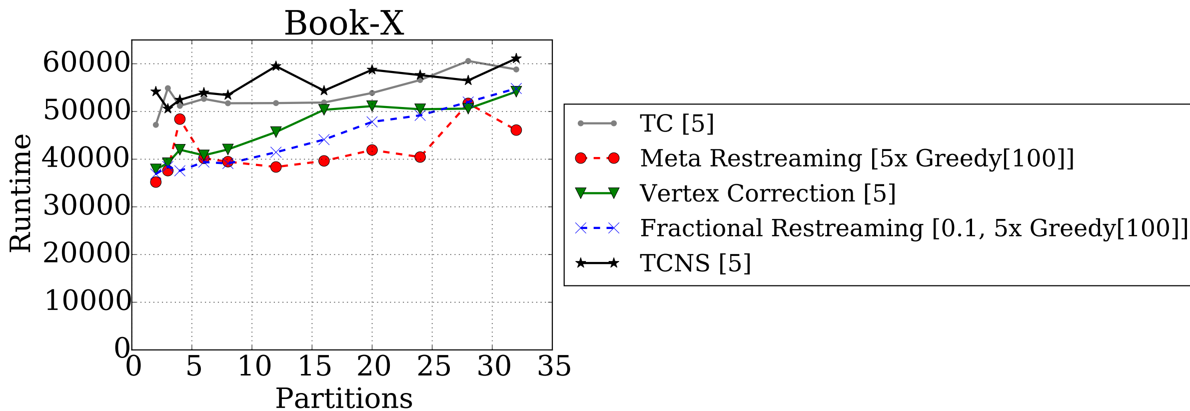


**Figure 5.33:** Execution Time of Partial Forgetting strategies vs Meta Restreaming on Book-X

Since the Adaptive Balance is besides Fractional Restreaming one of the best approaches in this thesis (cf. Section 5.1) it will also be evaluated with hMetis. hMetis is a hypergraph partitioning Framework [KK00] within a much higher complexity class and therefore a longer runtime. It is a very successful approach and consequently a good benchmark as lower bound for the best approaches. However it is important to keep in mind that hMetis has different balancing constraints than the approaches of this thesis as already mentioned in Section 2.2.1. hMetis also uses a different input data format. The strategy knows the graph size (number of vertices and hyperedges) and the graph is read as hyperedge list. This differences should be kept in mind when comparing the results of hMetis with other strategies introduced in this thesis.

The Figures 5.34 and 5.35 show the evaluation concerning the Cut Size. As expected hMetis has a better Cut Size in Figure 5.34 and every runtime class has its own domain. The restreaming approaches are up to 15% better than the real time approach and hMetis outperforms the streaming strategies by up to 12,5%. Very interesting is the comparison of Meta Restreaming and Adaptive Balance. Their performance is quite similar, but both have minimal spikes of a few percent. However, the cut size looks different on the Amazon graph (Figure 5.35). GTI'S Cut Size is still the worst but Adaptive Balance and Meta Restreaming can hold up to hMEtis and in some cases even outperform it. When looking at the Max Edge metric in Figures 5.36 and 5.37 both restreaming strategies perform equally well. At the same time hMetis is way worse - another trade off made by the strategy. When finally looking at the runtime (cf. Figure 5.38) hMetis takes much more time than required by Adaptive Balance or Meta Restreaming. In fact, the difference is that big that the runtime of GTI and the restreaming approaches seem to be equal (compare with Figure 5.28 for delta between Meta Restreaming and GTI). Hence, the runtime hMetis uses additionally has a worse exchange rate in terms of Cut Size. Especially when thinking of the hMetis performance on the Amazon graph.

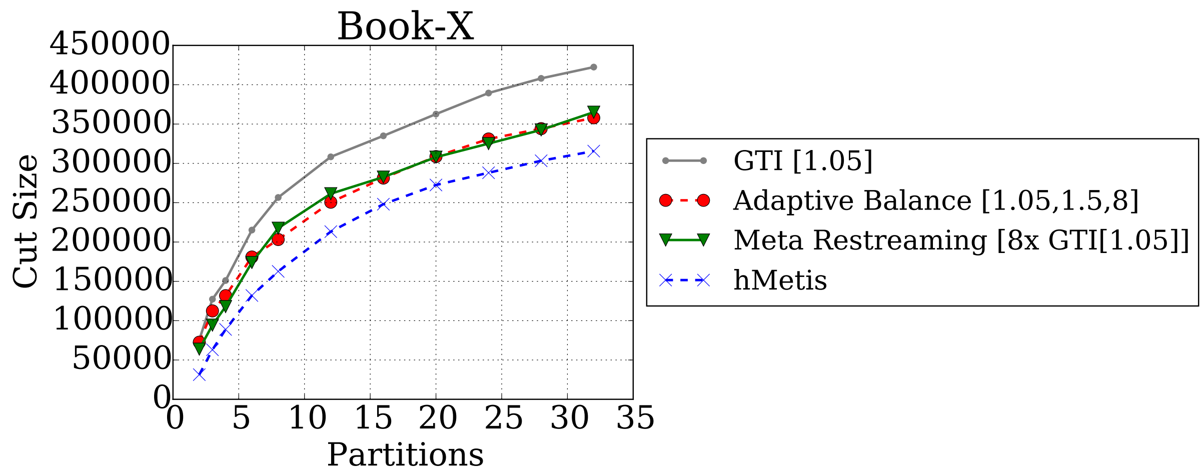


**Figure 5.34:** Cut Size of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X

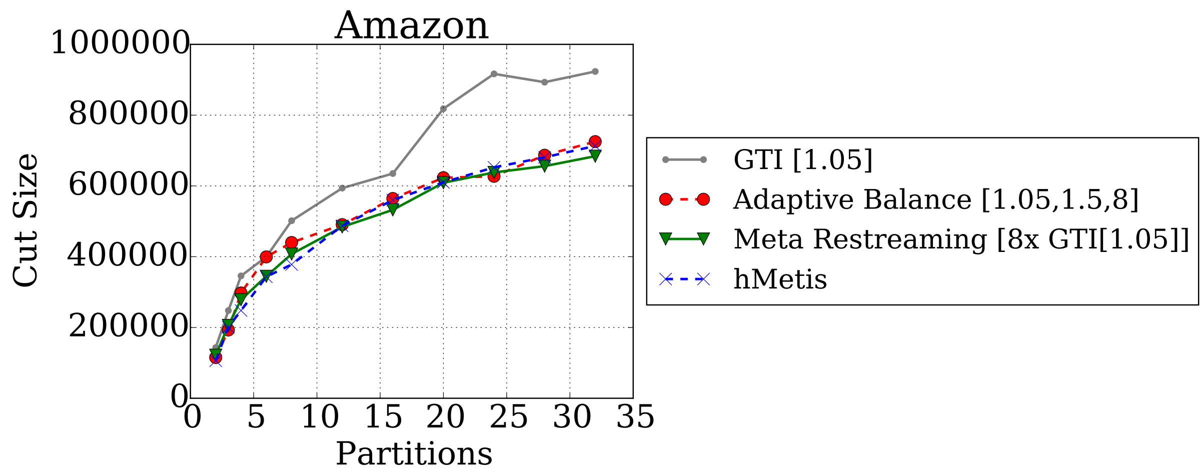


**Figure 5.35:** Cut Size of GTI, Meta Restreaming, Adaptive Balance and hMetis on Amazon

The results of Adaptive Balance in the direct competition with Meta Restreaming are not outstanding but also not worse. The strategy has the potential of better results which could be part of future work since no extensive parameter tuning was done yet. Also the Adaptive Balance offers a higher degree of freedom which could end up in better results on certain data inputs. If there are close clusters scattered over the stream the greater imbalance in the first passes will enable the strategy to find and handle these clusters earlier.

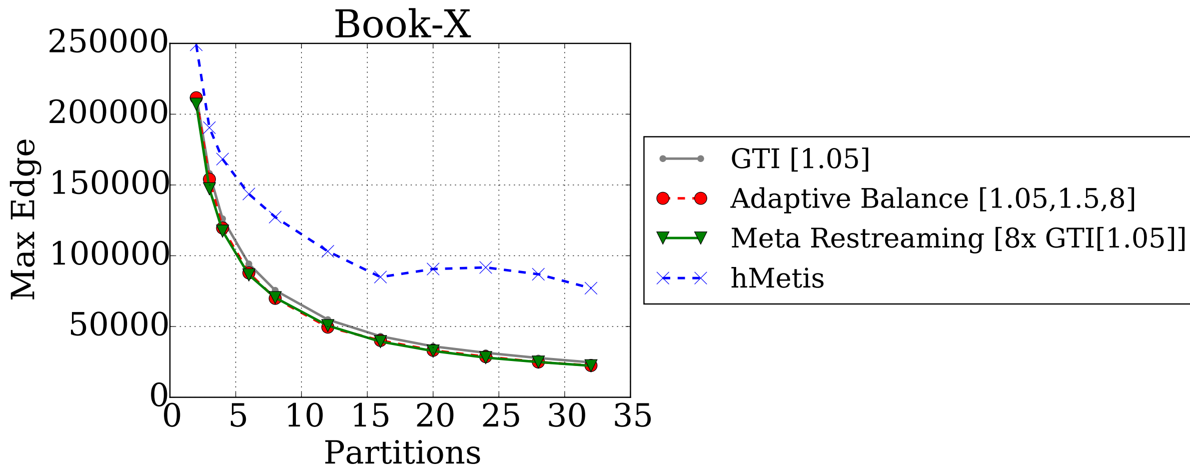


**Figure 5.36:** Max Edge of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X

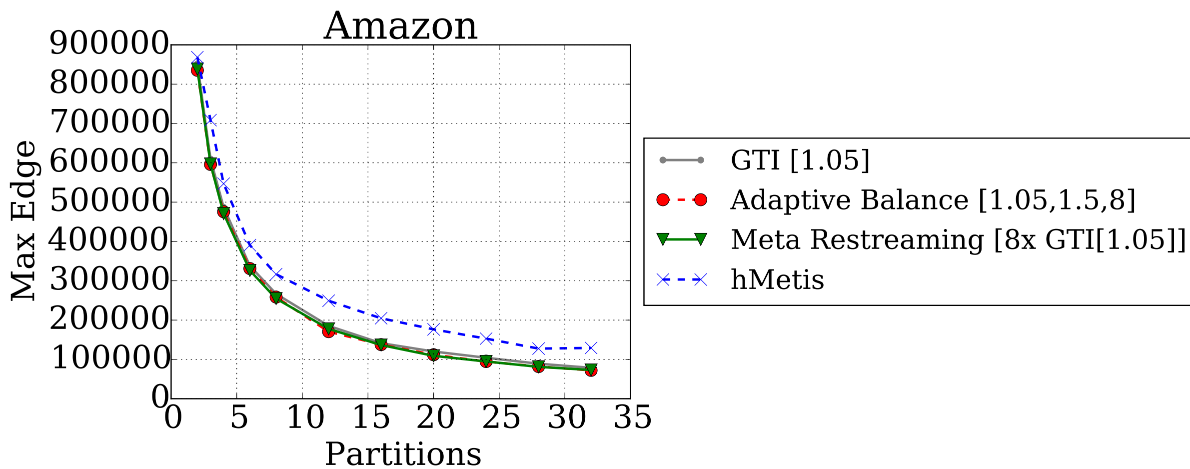


**Figure 5.37:** Max Edge of GTI, Meta Restreaming, Adaptive Balance and hMetis on Amazon

### Restreaming Deltas

When restreaming the changes between each pass may not be to extensive. Reassigning too many vertices to a new location can easily lead to bad (or not optimal) results. This is due to the fact that the old assignment is hardly used to benefit from the knowledge if most vertices are relocated.

Table 5.3 shows the impact of relocating vertices in the Book Rating graph. When looking at the run with $\lambda_{first} = 2$ it can be seen that when reassigning 58160 ( 55% of the graphs) vertices the Cut Size became worse. In comparison at the same time the run

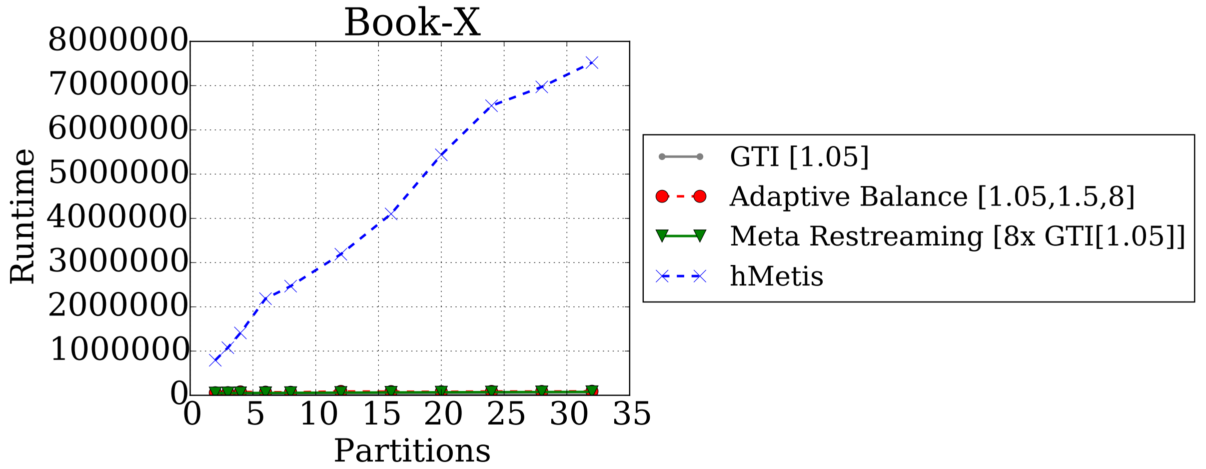


**Figure 5.38:** Execution Time of GTI, Meta Restreaming, Adaptive Balance and hMetis on Book-X

**Table 5.3:** Adaptive Balance on the Book Rating graph with 105282 vertices with several starting balances monitoring the Cut Size and number of relocated vertices after each pass

| $\lambda_{first}$ | 1.2 | | 1.5 | | 2.0 | | 3.5 | |
|---|---|---|---|---|---|---|---|---|
| Pass | #reloc | Cut Size | #reloc | Cut Size | #reloc | Cut Size | #reloc | Cut Size |
| 0 | 105282 | 256456 | 105282 | 250072 | 105282 | 246408 | 105282 | 225072 |
| 1 | 44433 | 235152 | 44378 | 231704 | 47280 | 228248 | 46127 | 212880 |
| 2 | 29276 | 227904 | 43800 | 228256 | 58160 | 235776 | 71649 | 252136 |
| 3 | 17535 | 222776 | 29072 | 217496 | 28886 | 226512 | 28075 | 244024 |
| 4 | 15218 | 218632 | 25547 | 212504 | 26077 | 222472 | 25472 | 239912 |
| 5 | 12209 | 215752 | 20437 | 210040 | 20050 | 218712 | 22052 | 233264 |
| 6 | 11799 | 214520 | 25075 | 206184 | 20240 | 216408 | 24850 | 227832 |
| 7 | 11637 | 213776 | 21169 | 203184 | 17933 | 215400 | 26126 | 221816 |

with $\lambda_{first} = 1.5$ reassigned only 43800 ( 40%) vertices which resulted in a better Cut Size. A run with $\lambda_{first} = 3.5$ relocated in pass two 71649 vertices ( 70%) and resulted in a way worse Cut Size. This setback could not be settled in the following passes.

Of course the deltas may not be too small since the Cut Size can not be improved very much otherwise. This can also be seen in Table 5.3 when looking at $\lambda_{first} = 1.2$. The number of relocated vertices is not big enough to have a larger impact in the later passes.

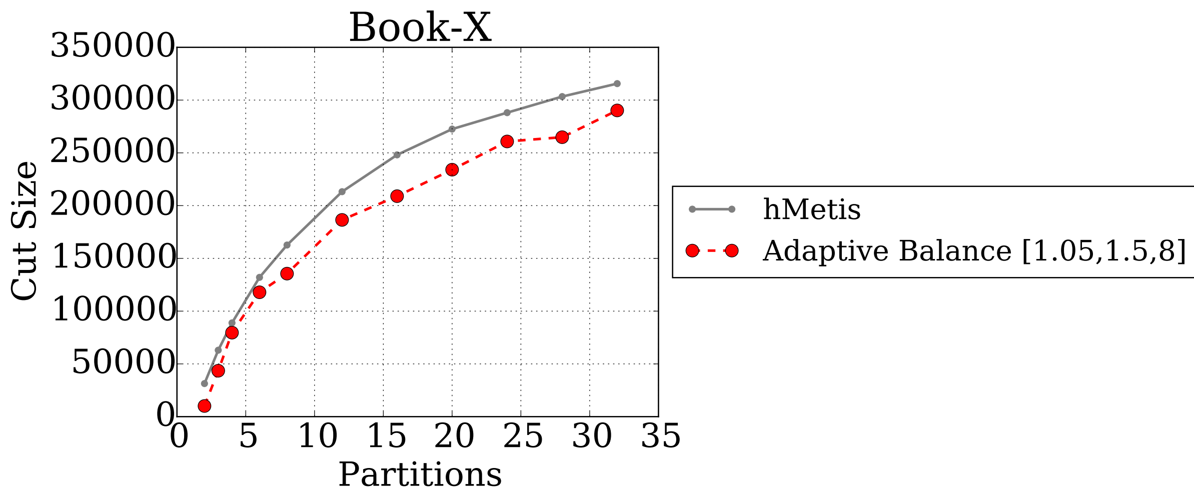


**Figure 5.39:** Cut Size when balancing vertices

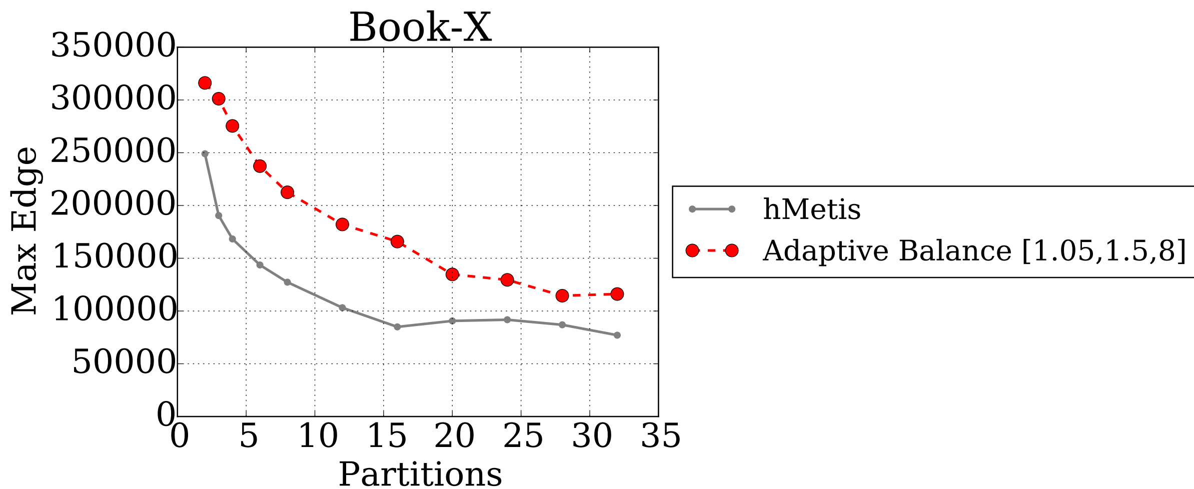


**Figure 5.40:** Max Edge when balancing vertices

### Balancing The Vertices

HMetis balances the vertices instead of the edges. Therefore, comparing hMetis with one of the own approaches is always a bit tricky. For this section the balancing of GTI was changed so instead of balancing the number of edges per partition the number of vertices is balanced. The rest still works as known. With GTI modified the Adaptive Balance strategy will also balance the vertices.

As it is shown in Figure 5.39 the Cut Size of Adaptive Balance is lower than the one of hMetis for all tested number of partitions. Meanwhile the Cut Size of GTI is a bit worse than the performance of hMetis. At the same time the execution time of hMetis is

enormous compared to Adaptive Balance (cp. Figure 5.38) and even more compared to GTI. This is due to the fact that hMetis is no linear time algorithm and because of the higher runtime class needs much more time for large inputs. Of course this benefit of Adaptive Balancing comes with a price. The number of vertices is indeed balanced but the number of edges per partition in not controlled anymore. Hence, adaptive balancing results in mapping hubs on the same partition and small subgroups on other partitions. Finally the Max Edge metric is significantly worse in Adaptive Balance than in hMetis (cp. Figure 5.40). GTI performs thereby slightly better than Adaptive Balance but still not as good as hMetis.

## 5.4 Decision Guidance

Purpose of this section is to provide a guideline about hypergraph partitioning strategies. Many approaches and implementations have been evaluated, discussed and rated in the previous sections. However, always without clear recommendation.

This thesis covers linear strategies but this runtime class can be divided in two subclasses which have been addressed: real time strategies which use only a single pass over the input data and restreaming strategies using several passes and therefore gaining additional information. Obviously these two subclasses can be used to address different problems. The real time strategies should be used for quick partitionings where the algorithm afterwards has short linear or even sub linear runtime. They are also suitable if the partitioning has to be redone frequently or is growing continuously. In the last case it should be considered to make an initial partitioning with a slower but more informed strategy and assign the continuously arriving deltas with a real time strategy if there is enough initial data.

As soon as the algorithms executed after the partitioning have a runtime which is higher than linear there is no need for real time partitionings any more. In this case a restreaming approach has a better influence on the runtime afterwards and is the better choice than real time strategies. This may also apply for periodically repartitioned data sets when the frequency is not too high. A large data set could be repartitioned daily within a few minutes using restreaming where slower strategies sometimes would take all day.

When looking at specific strategies in the real time class Greedy and GTI where outstanding. Thereby, Greedy is the better choice for strict balancing while GTI can grant a better partitioning due to a higher degree of freedom in a less strict balancing. When indecisive the Degree Aware stratgy using Never Max and GTI did hold good results combining benefits of Greedy and GTI. SNTI had an outstanding performance regarding

the balance. If balance is the most important attribute and the input size is completely unknown (and could probably be very small) while the partitioning has to be performed in real time SNTI is the right choice.

In terms of restreaming strategies there are three strategies which have provided good results: Meta Restreaming, Fractional Restreaming and Adaptive Balance. Since Meta Restreaming and Adaptive have basically the same results the decision is quite simple. Meta Restreaming needs less parameter optimizing and should therefore be the choice for quick configuration while Adaptive Balance should be used when the configuration time is not that important but the partitioning afterwards should be optimized as much as possible with the same runtime. Fractional Restreaming can improve the partitioning further but has a longer runtime which is needed to evaluate the assigned vertices and delete the worst of them.

# 6 Simple Hypergraph Partitioner

In this chapter the partitioning framework named Simple Hypergraph Partitioner (SHP) which was developed during the thesis is introduced. The framework was used to create the evaluations previously seen in the thesis. SHP can be used to get partitionings, however the main purpose was evaluating the strategies. It is written in Java and will be made SHP open source.

The chapter will first provide an architectural overview of SHP. Then the data model will be examined. Next the algorithms and the extensibility are displayed followed by a section about the usage of SHP including the output data format. Finally, some automations will be shown which can be used to run SHP for multiple configurations.

## 6.1 Architecture

An outline of the architecture of SHP and the program pipeline is content of this section. The basic architecture is shown in Figure 6.1. As it can be seen the system has some sort of layered design.

The Launcher component is on the top layer and has two major tasks. First it is the interface for the user to the system. Meaning the configuration given by the user is handled and mostly interpreted here. The parameters concerning the used strategy are send to a factory class in the Algorithms component. The second task is the program sequence. The launcher calls the other components and passes the information if necessary. It invokes the algorithm and feeds it with the data needed from the file input which is handled by another component. Furthermore, the Launcher starts the evaluation after the partitioning and if needed also the file output to write the final assignment to the Hard Drive Disk (HDD).

The next layer has two components: The Algorithms and the Evaluation. Algorithms holds the strategies which can be used by the framework. With a factory class an interface to the Launcher is provided. It creates the strategy objects used in the Launcher. The strategies themselves have an inheritance structure which is explained further in section 6.3.

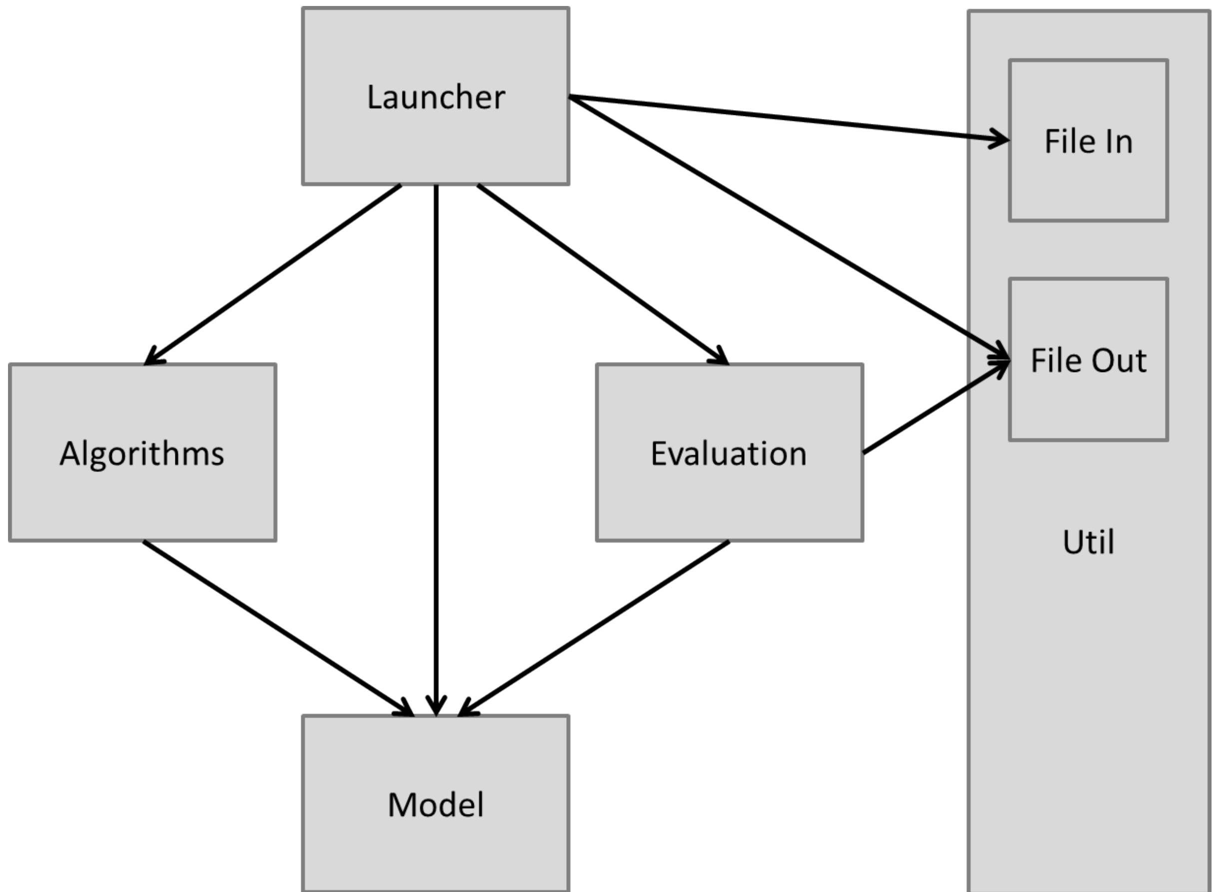


**Figure 6.1:** Layered architecture of SHP with packages and accesses

The Evaluation component takes the data from the Model and exerts the metrics on it (Section 2.2.1). Afterwards the results are send to a file output.

The final layer contains the Model. It is only used for data management. The Model holds information about partitions, vertices and hyperedges. While partitioning the input graph this information is modified constantly. It is further discussed in section 6.2.

An exception to the layered model is the Util component. It is a cross layer component providing general purpose functions for the other components. The links to the general Util component are not modelled in Figure 6.1 for a better overview. Functions provided are for instance methods to calculate intersection or union sizes. Those are used by some strategies. The component also provides methods for timestamps to improve console outputs. Another part of the component is a config class which holds default values for the system and a list of valid strategies. Besides the Util component includes two sub components for file interactions. First the File In component which reads input graphs and transforms it into a vertex stream. Second the File Out component to write evaluation results and the final partitioning to the HDD.

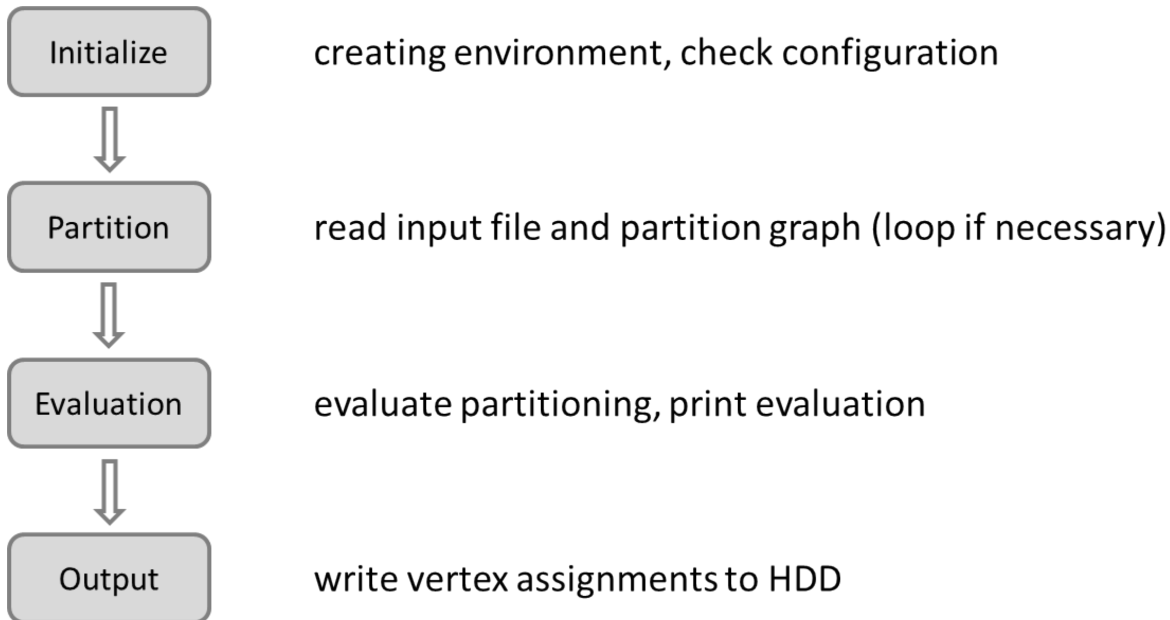


**Figure 6.2:** Program pipeline

The system works as a pipeline which is operated by the launcher. It consists of four phases shown in Figure 6.2. First the initialization, where the environment is created and the configuration is handled. Afterwards the partitioning takes place. This phase can be repeated several times if a restreaming is performed. The third phase is the evaluation where the assignment is rated and the results of the rating is printed (and also written to the HDD). The last phase is the output. This is an optional phase depending on the configuration. If selected, the vertex assignment is written to the HDD.

## 6.2 Data Model

The data model and its implementation is examined in the following. Figure 6.3 shows the objects and their knowledge about each other. The memory object manages all stored data. It acts as an interface to the layers above. When a new vertex is assigned to a partition, the information is given to the memory which stores the vertex and additionally transfers it to the concerning partition. The memory also holds high level information about the current partitioning like most/least loaded partition. This information can the be used from the algorithms. Additionally the memory has knowledge about the vertex mapping and has a list of the hyperedges already seen. Finally it organizes the remove procedures for vertices and edges.

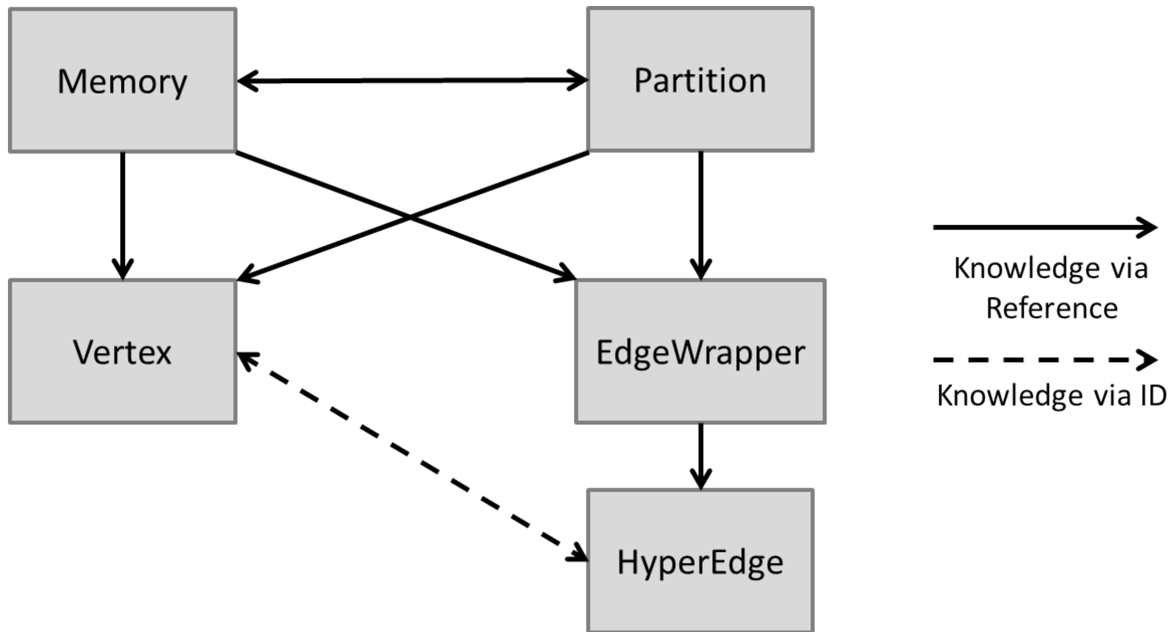


**Figure 6.3:** Data model of SHP

The partitions have their own vertex lists with the vertices assigned and a list with the connected hyperedges.

The vertex and hyperedge classes are simple data storages. They have a list with the IDs of the connected edges/vertices but no object reference. This decision was made to avoid reference circles.

The EdgeWrapper is used to realize reference counting for the hyperedges. That way every partition can notice instantly if a vertex is no longer connected to it and remove it from the edge list if this is the case. The memory works the same way. It can remove a hyperedge if it is not connected to a single partition anymore. This way the data stored in the memory and partition is always up to date.

## 6.3 Algorithms and Extensibility

This section covers the different types of algorithms supported by the framework and will explain how further strategies can be added. There are three types of algorithms, in the following named strategies. Figure 6.4 shows the inheritance between the three types.

First of all, there are simple streaming strategies which extend the AbstractPartitioningAlgorithm. Those are strategies which have only one data pass and assign the input vertex

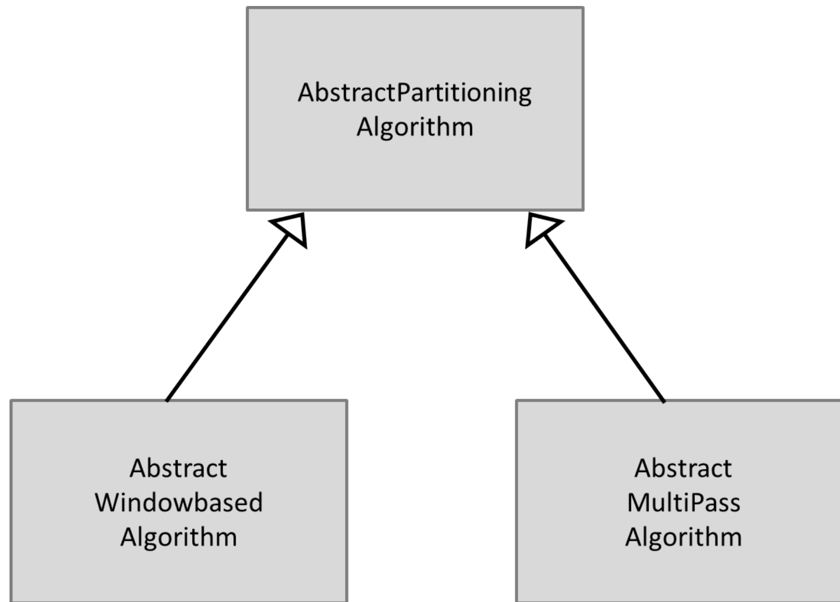


**Figure 6.4:** Abstraction of partitioning strategies

read immediately to an partition. The assignment logic of each strategy is written in a method named 'calculateNextVertex' which is called repeatedly during the pass. It determines on which partition the given vertex is assigned. Examples are Greedy, Random, GTI or SNTI.

Secondly WindowedStrategies extend the AbstractWindowbasedAlgorithm. These strategies also have only one pass, but they do not coercively assign the vertex they just read. They can store vertices in a window and assign them at any time during the partitioning. So the 'calculateNextVertex' method still holds the assignment logic but has the assignment calls inside because the given vertex does not need to be the assigned one. After the input graph is read, a 'finish' method is called. The 'finish' method is a second method holding strategy logic to ensure all given vertices are assigned. So the task of the 'finish' method is to assign all vertices left in the window. An example for a window based strategy is the Prefer Big strategy.

Finally, there are the restreaming strategies extending the AbstractMultiPassAlgorithm. These strategies read the graph several times. They have also two methods holding their logic. The 'calculateNextVertex' method which is called repeatedly during all passes and works like the one of the windowed strategies (assignment call inside). This is due to the fact that some restreaming strategies may use a pass only for counting or finding maxima and are not assigning anything. Besides, they have a method called 'endPass'. Here the logic deviating from simple restreaming like removing bad topics is done. The 'endPass' Method is called always if a pass finished. This also applies for the last pass even if many strategies like Topic Correction are not using the last endPass call.

Examples for these kind of implementations are TC, Meta Restreaming and the Two Pass strategy.

To add further strategies into the framework the matching Abstract class has to be extended and the methods implemented. Furthermore, the strategy has to be added into the StrategyFactory so the framework is able to create an instance of it. Finally, the strategy has to be added into the list of valid strategies in the config class. It informs the framework about the strategies existence. Furthermore, it holds information whether the strategy uses parameters and if yes, how much or if the number is dynamic.

## 6.4 Usage

How SHP can be used will be explained in this section. The program takes a number of parameters and executes a single run. This means there is no way to configure multiple partitionings and run them all in SHP at once. To take several runs external scripts have to be used, more about automation in section 6.5. SHP is a shell program and provides no graphical user interface.

The parameters and their domain are shown in Table 6.1. The number of partitions (-n) has to be a natural number greater than 0. Inserting the input graph (-in) has to be done as path to its txt file and the results are written to the directory given in the output (-out) parameter. The user can determine if the vertex assignment shall be stored on the HDD with the save (-save) parameter which takes a boolean value. True means the assignment is stored, false it will be discarded. The evaluation will be stored on the HDD anyway. SHP creates (or uses if they already exist) two files for the evaluation results. First a csv file containing the raw data shown in Listing 6.1. Each partitioning fills one line containing information as the timestamp of the partitioning run, the partitioned graph, the number of partitions, the used strategy and it's parameters, the number of vertices and edges in the graph, the result of the Max Edge metric, the average number of edges per partition, the execution time and finally the edge and vertex load. The Cut Size metric can be calculated with the values given and is therefore not stored additionally. Second a txt file holding the information formatted like the shell output which is readable for humans displayed in Listing 6.2. In this format all values from the previous file are also written but each metric gets an on line, the time stamp is also given as 'dd.MM.yy HH.mm.ss.SSS' format and the Cut Size is printed. The begin and end of a partitioning entry is made visible by special lines.

**Listing 6.1:** Build-up of the results.csv

```
timestamp, graph, number of partitions, strategy, [parameters], #vertices, #edges,
max_vertex, max_edge, avg_edge, execution time, [edge load], [vertex load];
```

**Table 6.1:** SHP parameter survey

| Parameter | Description | Domain |
|---|---|---|
| -n | determines the number of partitions to be produced | $\mathbb{N}_+$ |
| -in | the path of the input graph | path to a txt file |
| -out | the path where the results shall be saved. Path will be created if not existent | - |
| -save | determines if the assignment choices shall be stored to the HDD, pass 'false' if you are only interested in the evaluation | boolean value |
| -h | prints an overview about the commands on the shell can also be called with -help | - |
| -<x> <y>* | use strategy x with parameters y | valid strategy<br>valid parameters |

**Listing 6.2:** Build-up of the resultsHR.txt

```
=====start=====
timestamp: <value>
Timestamp as dd.MM.yy HH:mm:ss:SSS
graph name: <name>
number of partitions: <number>
strategy name: <strategy>
Parameters: [<parameter>]
number of vertices in graph: <number>
number of hyper edges: <number>
vertex maximum: <number>
hyper edge maximum: <number>
average number of hyperedges: <number>
Cut Size: <number>
execution time: <number>
edge load: [<number>]
vertex load: [<number>]
=====end=====
```

The strategies available in the current version are listed in Table 6.2. Note that passing several algorithms will cause SHP to use only the first one. Exceptions are meta strategies which need other algorithms. Thereby it has to be mentioned that not all strategies are currently compatible with each other. Meta strategies have currently some problems with running window based or restreaming strategies. This will be fixed in future versions.

**Table 6.2:** SHP implemented strategies

| **Strategy** | **Parameters** |
|---|---|
| -adaptive_balance | allowed imbalance at the end: $\lambda_{last} \in \mathbb{R} \| \lambda_{last} \geq 1$<br>allowed imbalance at the beginning:<br>$\lambda_{first} \in \mathbb{R} \| \lambda_{first} \geq 1$<br>number of passes: $\mathbb{N}_+$ |
| -all_on_one | - |
| -balance_big | balance (intern Greedy): $\mathbb{N}$ |
| -degree_aware | percentile: [0,1]<br>low degree Strategy: valid strategy<br>high degree strategy: valid strategy |
| -fractional_restreaming | percentage: [0,1]<br>number of strategies: $\mathbb{N}_+$<br>$n$ strategies: valid strategy |
| -greatest_topic_intersection | allowed imbalance: $x \in \mathbb{R} \| x >= 1.0$ |
| -greedy | balance: $\mathbb{N}$ |
| -meta_restreaming | number of strategies: $\mathbb{N}_+$<br>$n$ strategies: valid strategy |
| -never_max | balance (intern Greedy): $\mathbb{N}$ |
| -prefer_big | balance (intern Greedy): $\mathbb{N}$ |
| -random | - |
| -vertex_correction | number of passes: $\mathbb{N}_+$ |
| -smallest_non_topic_intersection | - |
| -topic_correction | number of passes: $\mathbb{N}_+$ |
| -topic_correction_narrow_scope | number of passes: $\mathbb{N}_+$ |
| -two_pass | percentage [0,1]<br>strategy: valid strategy |

## 6.5 Automation

Automating the usage will be the issue of this section. In the content of the thesis various scripts have been written to automate SHP calls and to visualize the evaluations. These scripts will be published with SHP as a toolbox. Of course every user shall feel free to write own automation tools. The scripts are written in python using python 2.7.10 with mathplotlib 1.4.3 and numpy 1.9.2.

The provided tool contains scripts for SHP calls, for looping through these calls and several visualizations. This includes line graphs with the partitions on the x-axis and an arbitrary metric from the file output on the y-axis, arbitrary metrics as bar chart for a single number of partitions and a visualization of the balancing of a single partitioning as bar charts. The tool automatically creates a file named 'visLog.txt' containing all visualization calls made. Hence, the graphics can be recreated with the visualization log and results.csv easily without rerunning the partitioning.

When changes in the SHP frameworks are made (e.g. new strategies implemented) these have to registered in the tool as well in order to use the tool on the adapted SHP. Therefore, there is a python file named 'Util' containing the information about valid strategies. If the output data is modified the 'EvaluationSet' file has to be adapted. Further information about the tool and its usage will be provided with the code.

# 7 Conclusion

In this thesis hypergraph partitioning for unknown input sizes has been examined. Thereby the goal was to minimize the number of edges which had to be cut while maintaining a balanced edge load. A second class goal was minimizing the maximal number of edges on a single partition. The strategies examined where restricted to be of linear runtime.

A survey of the proposed strategies was given. With its help the strategies can be judged based on the different metrics. Afterwards parameter independent real time strategies have been proposed. These can be used when absolutely no information about the input data is given and a real time partitioning is needed.

The most important part of the thesis was the evaluation of the restreaming strategies. It was shown that restreaming in general can outperform real time strategies easily while the trade off between Cut Size and runtime is feasible. Furthermore, Fractional Restreaming had proven that putting more logic in the restreaming by removing bad assignments before starting the next pass can improve the results even further than the simple knowledge about previous streams. Restreaming works for the partitioning of hypergraphs and can sometimes even outperform polynomial strategies.

Afterwards a decision guidance was given. With its help problem domains can be mapped to the most fitting real time or restreaming strategy. Besides, the SHP framework for hypergraph partitioning and evaluation was proposed. The usage and build up was explained so the framework can be either used or extended.

## Related Work

The min-max hypergraph partitioning from Alistarh et al. [AIV15] addresses an analogue problem but uses Max Edge as metric and only considers real time strategies. The strategies proposed in Chapter 3 had already been proposed in [AIV15]. An approach not considered in this thesis is the usage of neuronal networks even if it has proven to be a valid option [VDBM90]. Finally, there are much more problem domains than the one described in the introduction. This can go as far as reducing other problems on graph

partitioning like done by Fern et al. in [FB04]. Of course these related fields which had been touched during the thesis are just a small number of related works in the whole research area.

## Future Work

During the work on this thesis several approaches have provided the possibility of further investigation. However, due to time constraints not all of them could be explored. Hence, there is a list of promising tasks for future work. First, there is the Degree Aware strategy which holds much more potential than shown in the evaluation. These potential can be exploited by parameter tuning and considering more real time strategies to achieve different goals (like optimizing other metrics). Furthermore, the extensive study of the Degree Aware strategy would be a good addition to the decision guidance and could hold answers to many problem instances of hypergraph partitioning. Next, the idea of Partial Forgetting can be explored further and more strategies like Fractional Restreaming be developed. This also includes a parameter tuning for Fractional Restreaming on even more data sets. The Topic Correction Strategy could also be investigated further. A study which topics are removed and how big the impact really is could be illuminating for future studies. Finally, the Adaptive Balance strategy has much more potential than shown in this thesis. The strategy has a large degree of freedom which could be explored and the domain of the input data is not perfectly found yet.

# Bibliography


[AIV15] D. Alistarh, J. Iglesias, M. Vojnovic. “Streaming Min-max Hypergraph Partitioning.” In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett. Curran Associates, Inc., 2015, pp. 1900–1908. URL: http://papers.nips.cc/paper/5897-streaming-min-max-hypergraph-partitioning.pdf (cit. on pp. 19, 21–23, 27, 36, 73).

[AJB00] R. Albert, H. Jeong, A.-L. Barabási. “Error and attack tolerance of complex networks.” In: *nature* 406.6794 (2000), pp. 378–382 (cit. on pp. 29, 30).

[Che+15] R. Chen, J.-X. Shi, H.-B. Chen, B.-Y. Zang. “Bipartite-Oriented Distributed Graph Partitioning for Big Learning.” In: *Journal of Computer Science and Technology* 30.1 (2015), pp. 20–29. ISSN: 1860-4749. DOI: 10.1007/s11390-015-1501-x. URL: http://dx.doi.org/10.1007/s11390-015-1501-x (cit. on p. 21).

[Dev+15] M. Deveci, K. Kaya, B. Uçar, Ü. V. Çatalyürek. “Hypergraph partitioning for multiple communication cost metrics: Model and methods.” In: *Journal of Parallel and Distributed Computing* 77 (2015), pp. 69–83 (cit. on p. 19).

[FB04] X. Z. Fern, C. E. Brodley. “Solving Cluster Ensemble Problems by Bipartite Graph Partitioning.” In: *Proceedings of the Twenty-first International Conference on Machine Learning*. ICML ’04. Banff, Alberta, Canada: ACM, 2004, pp. 36–43. ISBN: 1-58113-838-5. DOI: 10.1145/1015330.1015414. URL: http://doi.acm.org/10.1145/1015330.1015414 (cit. on p. 74).

[Gar16] Gartner. *Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015*. Nov. 10, 2016. URL: http://www.gartner.com/newsroom/id/3165317 (cit. on p. 15).

[Gro] GroupLens. *MovieLens*. URL: https://grouplens.org/datasets/movielens/ (cit. on pp. 35, 36).

[HZY15a] J. Huang, R. Zhang, J. X. Yu. “Scalable hypergraph learning and processing.” In: *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE. 2015, pp. 775–780 (cit. on p. 16).

[HZY15b] J. Huang, R. Zhang, J. X. Yu. *Technical Report: HyperX A Framework for Scalable Hypergraph Learning*. Tech. rep. 2015 (cit. on p. 16).

[KK00] G. Karypis, V. Kumar. "Multilevel k-way Hypergraph Partitioning." In: *VLSI Design* (2000), pp. 285–300. URL: http://dx.doi.org/10.1155/2000/19436 (cit. on pp. 16, 20–22, 55).

[Les] J. Leskovec. *Amazon product co-purchasing network metadata*. URL: https://snap.stanford.edu/data/amazon-meta.html (cit. on pp. 35, 36).

[Mal+10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, G. Czajkowski. "Pregel: A System for Large-scale Graph Processing." In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 135–146. ISBN: 978-1-4503-0032-2. DOI: 10.1145/1807167.1807184. URL: http://doi.acm.org/10.1145/1807167.1807184 (cit. on p. 16).

[Man] U. of Mannheim. *Web Data Commons - Hyperlink Graphs*. URL: http://webdatacommons.org/hyperlinkgraph/ (cit. on p. 16).

[May+16a] C. Mayer, M. A. Tariq, C. Li, K. Rothermel. "Graph: Heterogeneity-aware graph computation with adaptive partitioning." In: *Distributed Computing Systems (ICDCS), 2016 IEEE 36th International Conference on*. IEEE. 2016, pp. 118–128 (cit. on p. 16).

[May+16b] R. Mayer, C. Mayer, M. A. Tariq, K. Rothermel. "GraphCEP: real-time data analytics using parallel complex event and graph processing." In: *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems*. ACM. 2016, pp. 309–316 (cit. on p. 16).

[Pet+15] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, G. Iacoboni. "HDRF: Stream-Based Partitioning for Power-Law Graphs." In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: ACM, 2015, pp. 243–252. ISBN: 978-1-4503-3794-6. DOI: 10.1145/2806416.2806424. URL: http://doi.acm.org/10.1145/2806416.2806424 (cit. on pp. 20, 29, 30).

[ST04] Z. Svitkina, É. Tardos. "Min-Max Multiway Cut." In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques: 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2004, and 8th International Workshop on Randomization and Computation, RANDOM 2004, Cambridge, MA, USA, August 22-24, 2004. Proceedings*. Ed. by K. Jansen, S. Khanna, J. D. P. Rolim, D. Ron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 207–218. ISBN: 978-3-540-27821-4. DOI: 10.1007/978-3-540-27821-4_19. URL: http://dx.doi.org/10.1007/978-3-540-27821-4_19 (cit. on p. 21).

[VDBM90] D. E. Van Den Bout, T. K. Miller. “Graph partitioning using annealed neural networks.” In: *IEEE Transactions on neural networks* 1.2 (1990), pp. 192–203 (cit. on p. 73).

[Wik] Wikipedia. *Statistics*. URL: https://en.wikipedia.org/wiki/Special:Statistics (cit. on p. 15).

[Zie] C.-N. Ziegler. *Book-Crossing*. URL: http://www2.informatik.uni-freiburg.de/~cziegler/BX/ (cit. on pp. 35, 36).

[staa] statista. *Statistics and facts about Facebook*. URL: https://www.statista.com/topics/751/facebook/ (cit. on p. 15).

[stab] statista. *Statistics and facts about Twitter*. URL: https://www.statista.com/topics/737/twitter/ (cit. on p. 15).

[stac] statista. *Statistics and facts about WhatsApp*. URL: https://www.statista.com/topics/2018/whatsapp/ (cit. on p. 15).

All links were last followed on May 08, 2017.

**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