Engineering Edge Ratings and Matching Algorithms for Multilevel Graph Partitioning Algorithms

Maximilian Schuler
Matriculation number: 1457790

December 7, 2011

Bachelor of Science in Computer Science

at the Institute for Theoretical Computer Science, Algorithmics II,
Karlsruhe Institute of Technology

Supervisors:
Prof. Dr. rer. nat. Peter Sanders and
Dipl. -Math, Dipl. -Inform. Christian Schulz

Abstract

The graph partitioning problem is one of the most basic and fundamental problems in theoretical computer science. In fact numerous practical applications exist ranging from the implementation of a VLSI design [15] to the identification of the function a specific gene is responsible for [3]. Though it can be shown that the balanced graph partition problem is NP-complete it can often be satisfactorily approximated in very short amount of time. Many modern graph partitioning algorithms use a multilevel scheme, which consists of three phases. First a hierarchy of subsequently coarser graphs is built, then the coarsest graph is partitioned initially and for each finer graph the partition is further refined. As the coarsening phase is crucial to a good result this work investigates and aims to improve techniques to be applied in the coarsening phase. The emphasis lies on the construction of new edge ratings and the analysis and design of a novel approximate matching algorithm. All findings are evaluated using the multilevel graph partitioner KaFFPa [25] that constructs a coarser graph by computing an approximate maximum weight matching and contracting all matched edges, which yields a very fast coarsening scheme. Our edge ratings together with a new approximate matching algorithm yield significant speed-ups over the standard settings for the most computational intense configurations of KaFFPa, and significantly improve the quality of the partitions calculated for social networks.

Acknowledgements

I would like to thank Prof. Dr. Peter Sanders for making this thesis possible and my supervisor Christian Schulz for his expertise, kindness, and most of all, for his patience. I am indebted to my friends for their support and encouragement. Among many others, I thank Leo Kogge, Patrick Spengler, Philip Rochel, Daniel Kucher, and Julian Ott. Finally I would like to thank my family for their love and continuous believe in me.

Einverständnisserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe

Karlsruhe, December 7, 2011 _____

Maximilian Schuler

Contents

Chapter 1.

Introduction

Our life is frittered away by detail...
simplify, simplify.
- Henry David Thoreau -

The graph partitioning problem is one of the most basic and fundamental problems in theoretical computer science. In fact numerous practical applications exist ranging from the implementation of a VLSI design [15] to the identification of the function a specific gene is responsible for [3]. Other applications include digital image segmentation [30] or the identification of communities within (online) social networks. However, perhaps the most interesting applications arise from *High-Performance-Computing*:

Today the most challenging computation tasks are carried out by computer clusters. Such a computer cluster consists of a heap of interconnected, small and usually cheap computers. From the user side they act as a single virtual machine that can perform any workload assigned to it.

A real-life example of such a task is weather forecasting. Satellites and weather stations permanently scan the earth's atmosphere, temperature, cloud formations and seas, collecting terabytes of data per day. This data has to be processed with complex models and thousands of iterations to produce a useful output such that a meaningful weather forecast can be made in time.

One of the main challenges in parallel computing is load balancing. The problem here is to break a large workload down into small chunks and distribute them to each of single machines equally by minimizing communication overhead. This problem can be formulated as a balanced *Graph Partitioning* problem if each vertex corresponds to a computational task and each edge to a data dependency. Now each block of a partition of the graph corresponds to a machine. The Graph Partitioning problem is to find a partition of the graph such that as few edges as possible lie between the blocks while each block should have the same size.

Though it can be shown that the balanced graph partitioning problem is NP-complete, it can often satisfactorily be solved in very short time. Many modern graph partitioning algorithms use a multilevel scheme, which consists of three phases. First a hierarchy of subsequently coarser graphs is built, then the coarsest graph is partitioned initially and for each finer graph the partition is further refined. The main goal of this work is to find new techniques that can be applied to improve the coarsening phase of such a graph partitioner. Namely we will investigate new edge rating functions and a novel approximate matching algorithm. All algorithms are

evaluated using the multilevel graph partitioner KaFFPa [25]. In the coarsening phase - and this is the approach we will assume throughout this work - it constructs a coarser graph by computing an approximate maximum weight matching and contracting all matched edges, which yields a very fast coarsening scheme.

Chapter 2.

Fundamentals

2.1. General Definitions

2.1.1. Graphs

Consider a weighted *graph* $G = (V, E, c, \omega)$ where V denotes the set of *vertices* and E the set of *edges*. The number of vertices is denoted by $n = |V|$ and the number of edges by $m = |E|$. The *edge weight* is denoted by $\omega : E \to \mathbb{R}_{>0}$ and *vertex weight* $c : V \to \mathbb{R}_{\geq 0}$. A set $\mathcal{P} = \{V_1, \cdots, V_k\}$ of disjoint sets is called a *partition* of V if $\bigcup_{i=1}^{k} V_i = V$ and $V_i \cap V_j = \emptyset$ for all $i \neq j$ and $i, j \in \{1, \dots k\}$. Then the individual sets V_i are called *blocks* and $\mathcal{P}[v]$ gives the block that contains v. The set of a vertex's neighbours is denoted by $N(v) := \{u : \{u, v\} \in E\}$ or more generally $N(U)$ the set of all neighbours in $V \setminus U$ of each $v \in U$. The set of edges incident to v is written as $E(v)$. For the *degree* of a vertex we write $d_G(v) = d(v) = |N(v)|$. A vertex with degree 0 is called *isolated*. Vertices or edges that are non-adjacent to each other are called *independent*. The number $\delta(G) := \min\{d(v) \mid v \in V\}$ is the *minimum degree* and $\Delta(G) := \max\{d(v) \mid v \in V\}$ the *maximum degree*. The number

$$d(G) := \sum_{v \in V} d(v)/|V|$$

is the *average degree*.

2.1.2. The Graph Partitioning Problem

Given a graph G, an integer $k > 1$ and the inbalance $\epsilon \geq 0$, the Graph Partitioning problem is to find a partition $\mathcal{P} = \{V_1, ..., V_k\}$ of V such that $\max_i |V_i| \leq (1 + \epsilon)\frac{|V|}{k}$ and the number of edges between blocks is minimized.

The inbalance ϵ must be allowed, since at least in the case that $|V|$ is prime no exact solution other than the trivial cases for $|P| = 1$ or $|P| = n$ can be found. In the weighted case

$$\max_i \sum_{v \in V_i} c(v) \leq \frac{(1 + \epsilon)}{k} \sum_{v \in V} c(v) + \max_{v \in V} c(v)$$

is used.

It can be shown that this graph partitioning problem is $\mathcal{NP}$-complete [10] and that no polynomial time approximation algorithm with finite approximation factor exists unless $\mathcal{P} = \mathcal{NP}$, which can be proven by a reduction from the 3-partition problem [1].

2.1.3. Subgraphs and Spanning Trees

A graph $G' = (V', E')$ is a *subgraph* of G if $V' \subset V$ and $E' \subset E$, we write $G' \subset G$. A subgraph G' is called induced if E' contains *all* edges $\{x, y\} \in E$ with $x, y \in V'$. In this work we write $G[V']$. If a subgraph is induced by $V \setminus \{v\}$ for some $v \in V$ we write $G - v$. More generally, if a subgraph is induced by $V \setminus S$ for some set $S \subset V$ we write $G - S$.

If G is connected and $m = n - 1$ then G is a tree. A cycle free graph is called a *forest* and each connected component of a forest is a *tree*. A forest $S = (V, E') \subset G = (V, E)$ is called a *spanning forest* of G if it has as many components as G and a *spanning tree* if it is also connected.

To calculate such a spanning tree one could use Kruskal's algorithm or Jarnik-Prim (often Prim's algorithm). Pseudocode can be found in Algorithm 1 and 2.

Algorithm 1 Kruskal's algorithm for minimum spanning trees

```
T ← ∅
for all e = {u, v} ∈ E  in ascending order of edge weight do
  if u  and  v are in different subtrees of  T then
    T ← T ∪ {e}
  end if
end for
```

By ordering the edges in descending order, Kruskal's algorithms will return a maximum spanning tree instead of a minimum spanning tree. To find maximum spanning trees with Prim's algorithm one has to do the following replacements. The smaller sign is replaced by a greater sign in the relaxation loop, the maximal entry of the priority queue is taken in each iteration and instead of decreasing the keys in the priority queue they have to be increased.

For a deeper discussion of both algorithms the reader is referred to [22].

2.1.4. Matchings

A set M of independent edges of a graph $G = (V, E)$ is called a matching. If a vertex is incident to an edge $e \in M$ it is called *matched* and *unmatched* otherwise.

While finding any matching is a trivial task, it gets much harder to find a matching that fulfils a specific property. If all vertices of V are matched the matching is said to be *perfect*; note that such a matching is only possible, if each connected component of G has an even amount of vertices. A matching is called *maximal* if it is not properly contained in another matching. Intuitively this means that it is not possible to add any extra edge to the matching. A matching M is said to be *maximum* if for any matching M', $|M| \geq |M'|$. Though from these problems arise interesting $\mathcal{NP}$-complete problems [10], e.g. the minimum maximal matching, in this

Algorithm 2 Jarnik-Prim for minimum spanning trees

```
d ←< ∞, ..., ∞ >
T ← ∅
pq ← {root} : PriorityQueue
while pq ≠ ∅ do
  u ← pq.min()
  d(u) = ⊥
  for all e = {u, v} ∈ E do
    if ω(e) < d(v) then
      d(v) = ω(e)
      T ← T ∪ {e}
      if v ∈ pq then
        pq.decreaseKey(v, d(v))
      else
        pq.insert(v, d(v))
      end if
    end if
  end for
end while
```

work we focus on the *maximum weight matching*. That is the generalization of the maximum matching problem, where the sum of the weights $\sum_{e \in M} \omega(e)$ is to be maximized.

Using Edmond's algorithm, a maximum weight matching can be calculated in $\mathcal{O}(\sqrt{n}m)$ [12, 6]. While it is good to find the optimal solution in many cases it suffices to get a near optimal solution. Many approximation algorithms exist that run in near linear time and guarantee approximation factors of two. Experiments indicate that these simple approximation algorithms often yield solution with a very low gap to optimality on real world graphs [21]. Such algorithms can be built upon calculating a maximum weight matching of simple subgraphs like a path or a tree. An algorithm that calculates an optimal maximum weight matching on paths and circles using standard dynamic programming is given in [21]. In section 4.1 we give an algorithm to calculate a maximum weight matching on forests and show how to construct an approximated algorithm based on matchings of maximum weight spanning trees.

2.1.5. Partition Quality Metrics

In this section we will show different metrics to describe the quality of a partition.

Internal and external Degree

$$deg_{extern}(v) = \sum_{u \in N(v) \setminus \mathcal{P}[v]} \omega(\{u, v\})$$

$$deg^{i}_{extern}(v) = \sum_{u \in N(v) \setminus V_i} \omega(\{u, v\})$$

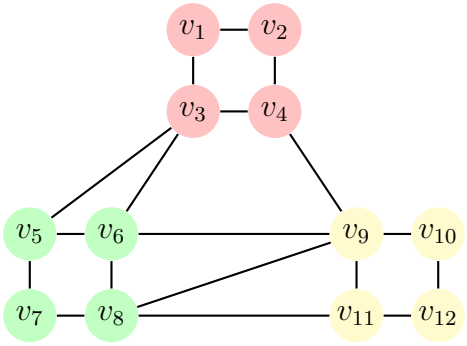


Figure 2.1.: A small graph with three blocks (red, yellow and green).

The external degree describes how many neighbours a vertex has that lie in a different block. Likewise the internal degree describes with how many vertices from the same block a vertex is connected.

$$deg_{intern}(v) = \sum_{u \in N(v) \cap \mathcal{P}[v]} \omega(\{u, v\})$$

In figure 2.1 the internal degree of all vertices equals two, while $deg_{extern}(v_3) = deg^{green}_{extern}(v_3) = 2$ but $deg^{yellow}_{extern}(v_3) = 0$.

Edge Cut. The *edge cut*, i.e. the number of edges between different blocks, yields a natural way to describe the quality of a given graph partition as it is the metric to be minimized in the standard definition of the graph partitioning problem. It is given by

$$EC = \frac{1}{2} \sum_{v \in V} deg_{extern}(v)$$

and the edge cut of a specific block is given by

$$EC^i = \sum_{v \in V_i} deg_{extern}(v).$$

We define the maximum edge cut to be from the block with the highest edge cut.

$$EC_{max} = \max_i EC^i$$

Looking at figure 2.1 again, we can see that the edge cut equals six, yet the maximum edge cut equals five, which is the edge cut of the green block.

Boundary Nodes. A *boundary node* is a node, that is connected to another block i.e. that has an external degree greater than zero or an internal degree smaller than its actual degree.

The *boundary* ∂V_i of a block V_i is then given by

$$\partial V_i = \{v \in V_i \mid deg_{extern}(v) > 0\}.$$

As a measure for the partition quality we will be interested in the *maximum boundary* ∂_{max} and the *total boundary* ∂_G.

$$\partial_{max} = \max_i |\partial V_i|.$$
$$\partial_G = |\{v \in V \mid deg_{extern}(v) > 0\}|$$

In our example graph in figure 2.1 there exist six boundary nodes, while the maximum boundary equals three.

Communication Volume. The set C_v of foreign blocks a boundary node is connected to can be written as

$$C_v := \{V_i \in \mathcal{P} \mid d^i_{extern}(v) > 0\}.$$

Then the *communication volume* of a block is defined as

$$comm(V_i) := \sum_{v \in V_i} c(v)|C_v|.$$

The *total communication volume* is defined as

$$comm_G := \sum_{V_i \in \mathcal{P}} comm(V_i)$$

and the *maximum communication volume* is given by

$$comm_{max} := \max_i comm(V_i).$$

The communication volume encodes information about necessary communication between different blocks. The main difference to the edge cut is that if a vertex is connected to two different vertices lying in the same block, they count only once. In our example graph the communication volume of the green block is four, which also is the maximum communication volume while the edge cut of the green block is five. This models the real communication overhead arising in high performance computing more realistically, since once information about a vertex has been sent to a block it is known to the whole block, and therefore to all vertices it contains.

Chapter 3.

Related Work

In this chapter we will present several concepts this work is based on. First, in Section 3.1, the multilevel graph partitioning scheme is presented and the three phases of such algorithms are discussed. In Section 3.2 we show different approaches to approximate a maximum weight matching and in Section 3.3 we present some additional graph parameters, which will prove useful for the construction of new edge rating functions.

3.1. Multilevel Graph Partitioning Algorithms

Multilevel graph partitioning algorithms are a recent approach to approximating the graph partitioning problem. Numerous such algorithms have been developed in the past two decades, many of which provide high performance and good partitions, e.g. Metis [26], Party [8] and Scotch [24].

The multilevel scheme, is very straightforward. It consists of three phases: coarsening, initial partitioning and refinement (or uncoarsening). During the coarsening phase the graph is gradually reduced by some local operation so that a hierarchy of coarser graphs is obtained. In the initial partitioning phase the coarsest and therefore smallest graph can be partitioned using any partition algorithm that is able to handle vertex and edge weights. This partition is then applied in the refinement phase to each larger graph and further refined.

Coarsening phase

The most common local reducing operation *edge contract* (sometimes *edge collapse*) is defined as follows.

Definition Edge Contract. Consider a weighted graph $G = (V, E, c, \omega)$. Collapsing an edge $\{u, v\} = e \in E$ into a node $w \notin V$ as illustrated in Figure 3.1 results in a graph G' where $V' = V \setminus \{u, v\} \cup \{w\}$. For E' first the set $S = E_0(u) \cup E_0(v)$ is removed and then set $T = \{\{w, y\} \mid y \in N(u) \cup N(v)\}$ added. An unweighted multi-graph can now be represented as a weighted simple graph by encoding multiple edges as one single edge, weighted by the number of multi-edges. The vertex weight is used to encode how many vertices have already

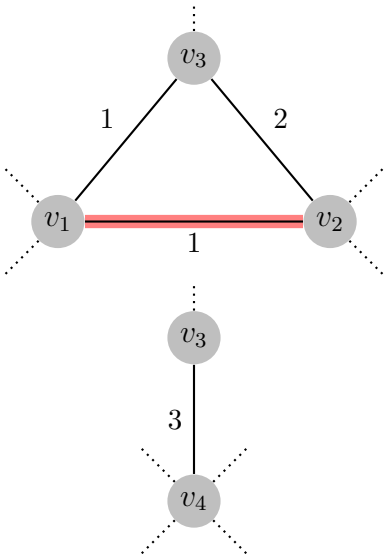


Figure 3.1.: The contraction of an edge. Here vertices v_1 and v_2 are contracted into the new vertex v_4. Note how the edge weight of the edge $\{v_3, v_4\}$ equals $\omega(\{v_1, v_3\}) + \omega(\{v_2, v_3\})$. Vertex weights are not illustrated.

been contracted into this vertex. To be more precise

$$c'(x) = \begin{cases} c(u) + c(v) & \text{if } x = w \\ c(x) & \text{else} \end{cases}$$

defines the vertex weight. Defining edge weights, one must be careful to model multi edges correctly by adding the weights of edges to a common neighbour together and selecting the right source for the others:

$$\omega'(\{x, y\}) = \begin{cases} \omega(\{u, y\}) + \omega(\{v, y\}) & \text{if } \{x, y\} \in T \text{ and } y \in N(u) \cap N(v) \\ \omega(\{z, y\}) & \text{if } \{x, y\} \in T \text{ and } y \in N(z), z \in \{u, v\} \\ \omega(\{x, y\}) & \text{else} \end{cases}$$

If a set U of edges must be contracted, this operation is repeatedly applied to each edge in U.

Now that a simple reduction operation is defined, it has to be decided which edges should be contracted in each step. An intuitive approach is to define a rating, that expresses how much sense it would make to contract a specific edge.

One simple edge rating metric could be to just use the edge weight, but this rating does not encode any information about topological properties of the graph or some of the edge's neighbours and it is very likely that a graph coarsened in such a way would lead to a bad partition. We will introduce more sophisticated ratings later, that try to overcome such issues, that still can be computed in constant time.

Now one radical approach is to only contract one edge at a time, thus producing a hierarchy of height n [23]. Another approach is to calculate an approximate maximum weight matching

of all edges based on these ratings and then contract each edge of the matching. This yields a very fast reduction scheme, that reduces the size of the graphs geometrically. Throughout this work we will assume that this second approach is used.

Initial partition phase

The repeated contraction is stopped once the coarsened graph contains only a few vertices, yet enough to perform a meaningful initial partitioning. To generate the initial partition one could try to calculate an exact solution, which according to [14] is only practical for graphs of up to 60 vertices even after applying techniques such as branch and bound.

Another possible approach is *recursive graph bisection*, which yields an approximate partition [13]. The idea here is, that one can build an algorithm to calculate a k-partition by recursively calculating bipartitions of the graph. Now to calculate a bipartition, first two pseudo peripheral vertices are found, i.e., two vertices that have approximately the greatest distance to each other. This can be done by starting a breadth first search at one random vertex v_0. Then a vertex v_1 is selected with maximal distance to x. Now a second breadth first search is started at v_1 and a vertex v_2 with maximal distance to v_1 is selected. This process can be repeated until the distance between v_i and v_{i+1} stops increasing. Each vertex is then assigned to that pseudo peripheral vertex it is closer to. This algorithm is used in many graph partitioners such as Metis. It should be noted that this process only performs well if the coarsest graph is balanced, since a vertex with too high weight might make it impossible to fulfill the balancing constraint.

Refinement phase

In order to reduce the cut while maintaining the balancing constraint of each matching being uncontracted different techniques can be applied.

One of the earliest local improvement methods has been proposed by Kernighan and Lin [17]. Here we will now briefly describe one partition refinement algorithm due to Fiduccia and Mattheyses [9], that is based on the KL-algorithm and has received a great deal of attention due to its efficiency. Roughly speaking, the FM-algorithm moves a single vertex along the current cut to reduce some gain metric, associated with it. A possible gain metric can be derived from the internal and external degree.

$$g(v) := d_{extern}(v) - d_{intern}(v)$$

More precisely, let $\mathcal{P}$ be a bisection of the graph. Now the FM algorithm alternatively selects an unmarked vertex with maximum gain and moves it to the other block until no unmarked vertex remains. As the algorithm allows an increase in the current cut it needs to keep track of the best partition found. If an improved partition could be found it is applied to the graph and this method is repeated until no further improvement can be generated. This algorithm can also be generalized to handle a k-partition.

3.2. Approximate Maximum Weight Matching Algorithms

In the following sections different approximative algorithms for the maximum weight matching problem with near linear runtime are given. A simple approach could be to just add the heaviest possible edge to the matching in a greedy manner and basically such an idea is the origin of all algorithms presented in this section. We present pseudocode and a short description of the basic ideas, runtime and quality guarantees. However for a full analysis of these algorithms the reader is referred to the individual papers.

3.2.1. Heavy Edge Matching

Heavy Edge Matching (HEM) has originally been proposed by Karypis and Kumar [16] and yields a running time of $\mathcal{O}(m)$. We present pseudocode in Algorithm 3.

Algorithm 3 HEM

```
M ← ∅
S ← ∅
for all v ∈ V in random order do
  if v ∉ S then
    for all u ∈ N(v) do
      if u ∉ S then
        M ← M ∪ {e}
        S ← S ∪ {u, v}
      end if
    end for
  end if
end for
return M
```

Karypis and Kumar [16] also present a variant of HEM called *Sorted Heavy Edge Matching* (SHEM), where the vertices are visited in ascending order of their degrees with random tie-breaking. Unfortunately no approximation factor can be given since Karypis and Kumar give a counterexample to any approximation factor. In practice however, HEM and SHEM find good matchings.

3.2.2. Global Paths Algorithm

The Global Paths Algorithm (GPA) [21] listed as Algorithm 4 generates a maximal weight set of circles and paths and then calculates an optimal maximum weight matching on all of them. An edge is called applicable and inserted into the set if it connects two endpoints of different paths or if it closes an odd length cycle. An edge is not applicable if it closes an even length cycle, or if it incidents to the inside of a path. Now for each path and cycle a maximum weight matching is calculated.

Algorithm 4 GPA

```
M ← ∅
E' ← ∅
for all e ∈ E in descending order of weight do
  if e is applicable then
    E' ← E' ∪ {e}
  end if
end for
for all path or cycle P in E' do
  M' ← MaxWeightMatching(P)
  M ← M ∪ M'
end for
return M
```

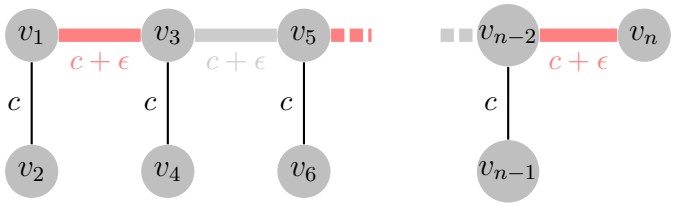


Figure 3.2.: Example graph on which GPA only produces a 2-optimal solution

GPA has a running time of $\mathcal{O}(sort(m)+m)$, which is $\mathcal{O}(m \log n)$ in general and an approximation ratio of $\frac{1}{2}$. The example in Figure 3.2 shows this bound is tight. For a detailed analysis see [21].

3.3. Additional Parameters of Graphs

In order to construct edge ratings we need a set of parameters an edge and its vertices can be associated with. The first and foremost important parameter for the construction of an edge rating is the edge weight. Other obvious parameters like vertex weights or degrees can also be incorporated and need no further explanation. Measuring the graph's density, distance or connectivity between nodes is a more complicated task, so we will now turn to two possible definitions that can be used to express this notion.

3.3.1. Degeneracy and Coreness

The *k-core* of graph G is a maximal subgraph in which each vertex has at least degree k. The coreness of a vertex is k if it belongs to the k-core but not to the $(k+1)$-core. Sometimes the maximal coreness of a graphs vertices is called its *degeneracy*. The following two statements directly follow from the definition of coreness:

- $\delta(G) \leq \text{coreness}_v \leq d(v)$
- so if $d(v) = \delta(G)$ then $\text{coreness}_v = \delta(G)$

With this in mind we can formulate an algorithm for calculating the coreness of a graphs vertices, given as Algorithm 5, which has been described in [2]. The algorithm uses a priority queue that initially holds every vertex with its degree in G [1]. It subsequently removes vertices from the priority queue and decreases all their neighbours until the queue is empty.

Algorithm 5 Coreness

```
pq ← ∅ : PriorityQueue or BucketQueue
for all v ∈ V do
  pq.insert(v, d(v))
end for
k = δ_G
while pq ≠ ∅ do
  (v, d) ← pq.deleteMin(pq)
  k = max(k, d)
  core(v) = k
  for all u ∈ N_v do
    pq.decrease(u)
  end for
end while
```

Correctness: First we observe that G is a valid $\delta(G) - core$ and each vertex v with $d(v) = \delta(G)$ has coreness $\delta(G)$ since it cannot contribute to any $(\delta_G + k) - core$, $k > 0$. Now we remove one minimal degree vertex v from G. While all its neighbours decrease in degree we know for any vertex with degree smaller than that of v that these still have coreness $d(v)$. We repeat this process until no vertices remain with degree $d(v) \leq \delta_G$ or no vertices remain at all. Thus we are either done or we have constructed a correct $(\delta(G) + k) - core$ and repeat the process of removing vertices and assigning this new coreness of $\delta(G) + k$ to them.

Runtime: Every vertex is added to the queue and removed from it only once. And as each neighbour of these vertices is processed at most twice (once to grow M and once to decrease or delete them) at most $2m$ decrease operations take place. Since the vertices are inserted in the queue by their degree and are only decreased, a bucket queue with Δ_G buckets can be applied. This yields an overall runtime of $\mathcal{O}(n + m)$.

3.3.2. Algebraic Distance

Another way to express the density is to measure the distance between vertices. A very interesting distance measure is the *Algebraic Distance* proposed by [4]. Though we surely

[1] It should be noted that a Bucket Queue must be used in order to achieve linear runtime.

cannot go into the full theoretical details of the process of calculating the Algebraic Distance we will nonetheless describe the basic concepts behind this metric and give the according algorithm. The graph Laplacian matrix L is defined as

$$D - W,$$

where D is the diagonal matrix with diagonal entries $D_{u,u} = \sum_{e \in N(u)} \omega(e)$ and W is the weighted adjacency matrix with entries $W_{u,v} = \omega(\{u, v\})$. Here for $e \notin E$ the weight $\omega(e) = 0$ is assumed. The algorithm to calculate the algebraic distances of the graph's edges given as algorithm 6 is basically the JOR (Jacobi Overrelaxation) method of solving the linear system

$$Lx = 0.$$

An analogy to this approach is to assign a random amount of liquid to each vertex and distribute it in each iteration through the graph according to the edge weights. The intuition is that neighbouring vertices that are close to each other, will happen to have similar potentials.

The iterated version of solving the linear system given above can be written as

$$x^{(k+1)} = Hx^k$$

where H is the iteration matrix.

Now as the Laplacian can be decomposed into $L = D - W_L - W_U$ where W_L and W_U are strict lower and upper triangular matrices, the iteration matrix of the Jacobi method is $H_{JAC} = D^{-1}(W_L + W_U)$. If a relaxation parameter θ is introduced, this becomes

$$H_{JOR} = (D/\theta)^{-1}((1/\theta - 1)D + W_L + W_U).$$

For the analysis of the convergence of the JOR method the normalized Laplacian $\mathcal{L}$ is used which can be expressed as

$$D^{-1/2} L D^{-1/2}$$

Then the JOR method converges for any $\theta \in (0, 2/\rho(\mathcal{L}))$, where $\rho(\cdot)$ is the spectral radius of a matrix. As discussed in [4], in practice a fixed θ with value $1/2$ is a good choice and it suffices to only calculate the first three iterations.

Algorithm 6 Algebraic Distance

$\theta \in (0, 2/\rho(\mathcal{L}))$: Parameter
$x^{(0)} \in \mathbb{R}^n$: Initial random vector
for all $k = 1, 2, \ldots$ **do**
  for all $u \in V$ **do**
    $\tilde{x}_u^{(k)} \leftarrow \sum_{v \in N(u)} \omega(\{u, v\}) \tilde{x}_v^{(k-1)} / \sum_{v \in N(u)} \omega(\{u, v\})$
  end for
  $x^{(k)} \leftarrow (1 - \theta) x^{(k-1)} + \theta \tilde{x}^{(k)}$
end for

Then the algebraic distance of an edge is given by $s_e^{(k)} := |x_u^{(k)} - x_v^{(k)}|$. With R initial random

vectors $x^{(0,r)}, r = 1, \ldots, R$ the *extended p-normed algebraic distance* is given by

$$\varrho_e^{(k)} = (\sum_{r}^{R} |x_i^{(k,r)} - x_j^{k,r}|^p)^{1/p}$$

and for $p = \infty$ this yields

$$\varrho_e^{(k)} = \max_{r=1,\ldots R} |x_i^{(k,r)} - x_j^{k,r}|.$$

Chapter 4.

Approximate Maximum Weight Matchings

As mentioned above, in the process of coarsening a graph there exist basically two possible ways to select edges for contraction. Contracting exactly one edge at a time yields n levels of coarser graphs, and if one tries to keep the height of the graph hierarchy low, a natural way is to contract a matching of edges. The latter approach allows to save each graph directly, as each level of contraction yields a much smaller graph. In this section we will first give an algorithm that solves this problem on a forest optimally and then introduce an approximate maximum weight matching algorithm based on this algorithm.

4.1. Optimal Maximum Matching on Forests

Here we give an algorithm that solves the maximum weight matching problem on trees in linear time by dynamic programming. Pseudocode is given in Algorithm 7. By executing the algorithm on each connected component of the graph, a maximum matching of a forest can be calculated.

The dynamic program can be formulated with two functions, that give for each root v of a subtree of the tree the optimal solution $L(\cdot)$ and the optimal solution $l(\cdot)$ in case v is already matched with its parent and thus cannot contribute in a matching with one of its children.

$$l(v) = \sum_{c \in \text{children}[v]} L(c)$$

$$L(v) = \underset{c \in \text{children}(v)}{\arg\max} \left[\sum_{c' \in \text{children}[v], c' \neq c} \big(L(c') + \omega(\{v, c\}) + l(c) \big) \right]$$

Here children$[v]$ gives the set of children of the vertex v. So for a given forest $F = (T_1, \ldots, T_c)$ in order to give the weight of the maximum weight matching, we have to sum the value of $L(\cdot)$ of each tree's root.

Before we examine Algorithm 7 more closely, we give a small example of a matching on a tree in Figure 4.1 and show how one would calculate the individual sub-solutions. Here the first number in each vertex corresponds to $L(v)$ and the second number to $l(v)$. First we notice that for each leaf $l(v) = L(v) = 0$ as the subtree rooted at a leaf has no edges. For each non-leaf

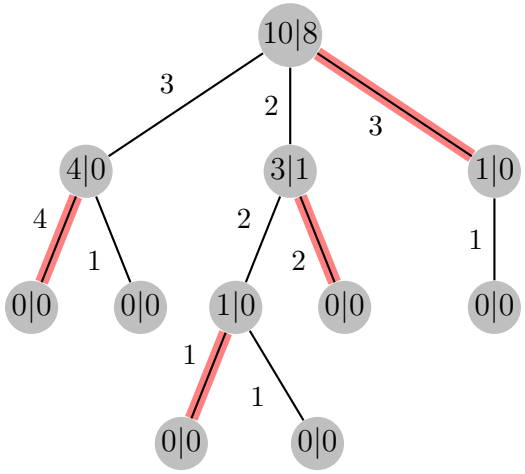


Figure 4.1.: A simple example of the maximum weight matching calculation on a small tree. Matched edges are bold red.

vertex, $l(v)$ is simply the sum of all $L(\cdot)$ values of its children. For example the $l(\cdot)$ of the root is $l(\text{root}) = 4 + 3 + 1 = 8$. To calculate $L(v)$ we evaluate $l(v) + \omega(\{v, c\}) - L(c) + l(c)$ for each child c and take the maximum. This term basically adds the gain of a contribution in a matching with child c to the solution $l(\cdot)$.

Now in Algorithm 7, the first loop calculates the two solutions for each vertex and the second establishes the final matching. For the first part of the algorithm the invariant holds that for each node looked at, both solutions of each child have already been calculated. Therefore the child can be found easily, that gives the optimal solution of the current subtree, in the way mentioned above by looking at each child once. By adding the matching solution $L(v)$ of v to the unmatching solution $l(p)$ of its parent p, we can establish the correct unmatching solution of the parent before processing it. This way $l(v)$ is already calculated when visiting v and only $L(v)$ must be calculated.

The second part of the algorithm is very straightforward. In a top-down fashion, we look at each node once more. Since the invariant holds that for each vertex v the matched solution $L(v)$ is already the optimal solution of the subtree rooted at v, we can greedily evaluate the matching. Here a vertex is called blocked if it has already contributed in a matching wit its parent and can therefore be ignored.

As for each vertex all its neighbours are visited once, each edge is visited at most twice. Since for a tree $m = n - 1$, the algorithm yields a runtime of $\mathcal{O}(n + m) = \mathcal{O}(n)$.

4.2. Maximum Spanning Tree Matching

Instead of repeatedly growing paths as in the GPA algorithm, the *Maximum Spanning Tree Matching* algorithm (MSTM) calculates a maximum weight spanning tree of the graph and generates an optimal matching M_0 on this tree, with Algorithm 7. The generated matching

Algorithm 7 Maximum weight matching on a tree

```
for all v ∈ T in reverse bfs order do
  p = parent[v]
  matchedSolution[v] ← maximum weight of a matching with a child ĉ of v
  save ĉ as a partner of v
  unmatchedSolution[p] ← unmatchedSolution[p]+ matchedSolution[v]
end for
for all v ∈ T in bfs order do
  if v is not blocked then
    match v with previously calculated partner
    block partner of v
  end if
end for
```

Algorithm 8 MSTM

```
M ← ∅
M_0 ← ∅
repeat
  MST ← CalculateMaximumSpanningTree(G)
  M_0 ← MaxWeightMatching(MST)
  G ← G − M_0
  M ← M ∪ M_0
until |M_0| = 0 or G = ∅
return M
```

is then added to the final matching and the method is repeated on $G' = G - M_0$ until no unmatched vertices remain or no edge could be matched in this iteration.

Approximation Ratio. In its simple *non-iterative* form this algorithm may perform very bad and as a simple counterexample to any approximation rate the wheel illustrated in Figure 4.2 can be given. Choosing the edge weights of the star inside the wheel to be higher than those of the cycle around it, we can enforce the maximum spanning tree to be only the star, which leads to only one edge being matched. If we consider the graph $G - M$ induced by the still unmatched vertices of this wheel, only a path remains on which another run of the maximum spanning tree algorithm returns the optimal solution. This gives the idea of iterating the algorithm and applying it to the remaining graph until either no vertices remain or the size of the matching stops increasing.

Still the approximation ratio of this algorithm is unknown, but even the worst example known yet illustrated in 4.3 yields a $\frac{2}{3}$-approximation ratio. MSTM could end up with the matching illustrated since only the heavy edges are taken into account. Then if i denotes the number of stars without the outer star, MSTM's solution has weight $|M_{mstm}| = (i+1)(1+\epsilon)$ and an optimal solution is $|M_{opt}| = i(1+\epsilon) + \frac{i}{2}$ for $\epsilon > 0$ being small and $i \geq 3$, and for the approximation ratio p we get

$$\frac{|M_{mstm}|}{|M_{opt}|} = \frac{(i+1)(c+\epsilon)}{i(c+\epsilon) + c\frac{i}{2}} = \frac{i+1}{i}\frac{c+\epsilon}{c+\epsilon+\frac{c}{2}} = \frac{2}{3}\frac{i+1}{i}\frac{c+\epsilon}{c+\frac{2}{3}\epsilon}$$

which yields for large c and i an approximation factor of $p = \frac{2}{3}$.

It should be noted that this example only gives an upper bound for the approximation ratio of MSTM.

Runtime. The running time of one iteration is dominated by the time needed to calculate the spanning tree since an optimal matching of the tree can be calculated in linear time. Though in practical application the algorithm normally breaks after around three iterations it may take dramatically more iterations especially on dense graphs. We can find an upper bound of the iteration number as the maximal degree of the graph.

Lemma 1: MSTM breaks after less than Δ iterations.

Proof. We will show that in each iteration the maximum degree of the remaining graph induced by the still unmatched vertices decreases by at least one. Then the lemma follows by induction, since a graph of maximum degree zero cannot increase the matching any more and therefore the algorithm breaks at that point.
Consider an unmatched vertex $v \in V$ with degree $\Delta(G)$ and $G' = G - M$ the subgraph induced by the unmatched vertices. By definition such a vertex is in V' but all his neighbours on the spanning tree contribute to the matching and are therefore not in V'. If one neighbour on the spanning tree existed that would not be adjacent to a matched edge already, it could contribute in a matching with v, which contradicts the optimality of M on the spanning tree. As each v has at least one neighbour on the tree, the maximum degree is $\Delta(G') \leq d(v) - 1 = \Delta(G) - 1$. If no such vertex v exists, clearly G' does not contain any vertex with degree $\Delta(G)$.

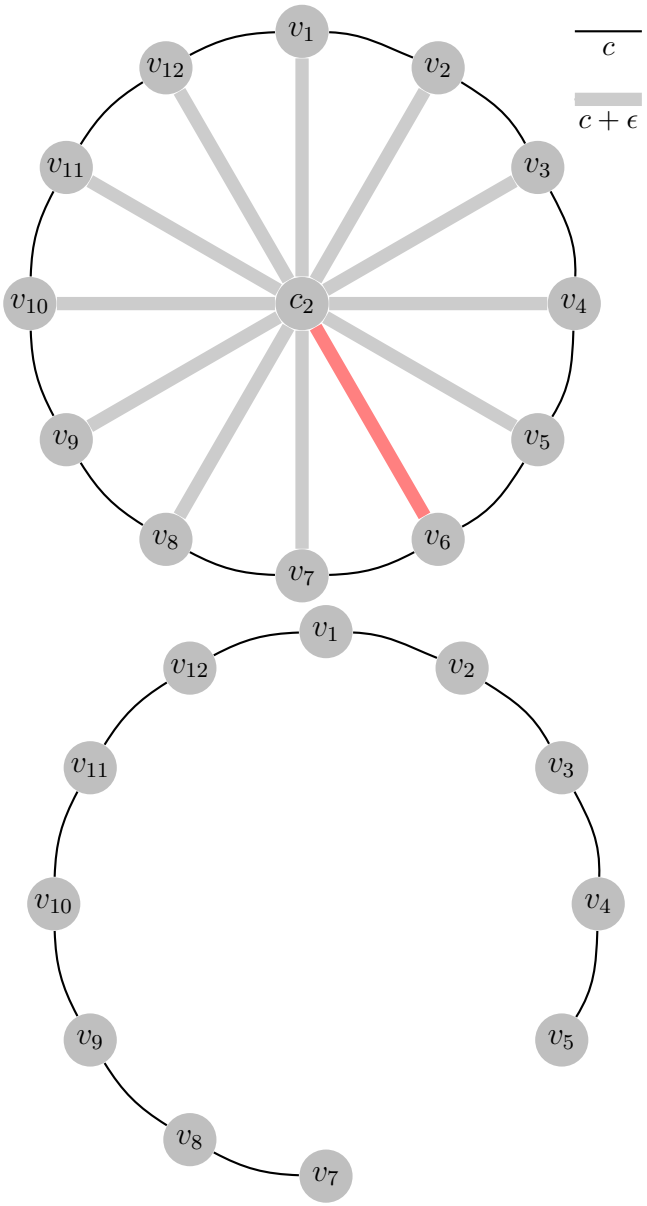


Figure 4.2.: Counterexample for any approximation ratio of the non-iterative MSTM. The thick lines are the maximum spanning tree. This tree's optimal matching, drawn red, contains only one edges with weight $c+\epsilon$, while the optimal matching of this graph contains $c\lfloor\frac{n}{2}\rfloor$ edges with weight $\epsilon + \lfloor\frac{n}{2}\rfloor$. In the second iteration of MSTM on this graph (lower figure), only the v_7v_5-path remains, so MSTM would find the optimal solution in the second iteration.

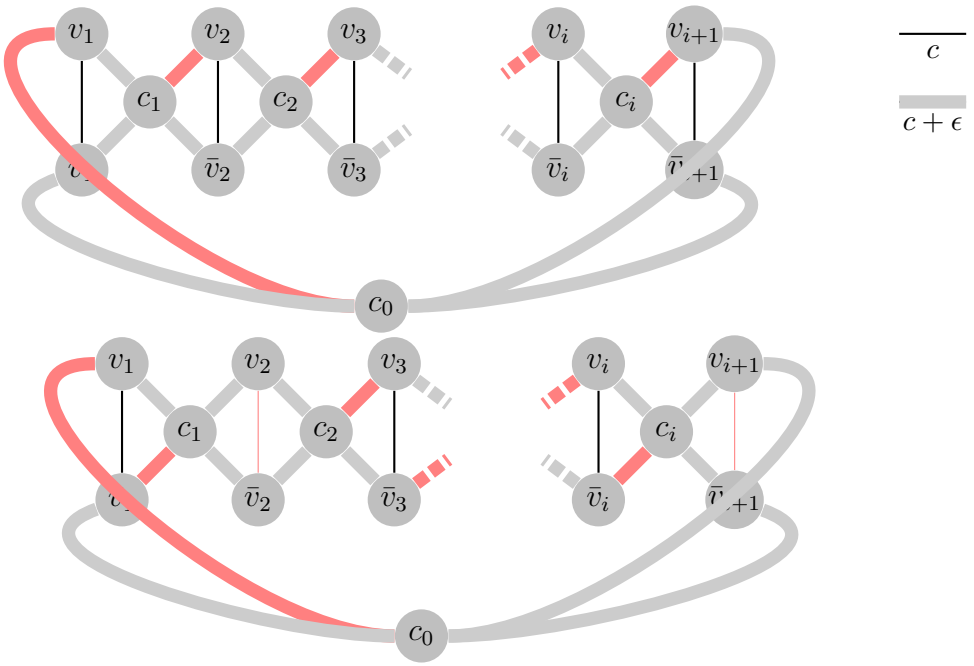


Figure 4.3.: A graph on which MSTM yields only a $\frac{2}{3}$-approximation rate. The upper graph shows a matching MSTM finds. No second iteration can be executed since after the removal of the matched edges no edges remain. The lower graph shows the optimal matching, if i is chosen to be sufficiently big enough and ϵ small enough.

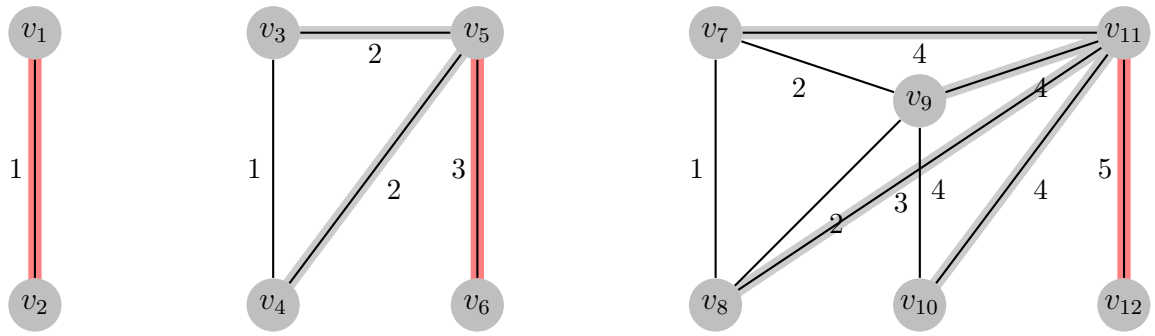


Figure 4.4.: The Graphs G_i for $i \in \{1, 2, 3\}$. The maximum spanning tree for each graph is drawn thicker, its maximum weight matching is drawn red. In each step only one edge is being matched.

□

To show that this bound is tight, a family of graphs (see Figure 4.4) with such a high iteration number i can be constructed like this:

- For $i = 1$ G_i is just two vertices connected with each other with weight one.
- For $i \geq 2$ G_i is G_{i-1} together with two additional vertices connected with each other with weight $2i - 1$ and one of them is connected with all vertices of G_{i-1} with weight $2i - 2$

Such a Graph G_i has $n = 2i$ vertices, $m = (\frac{n}{2})^2 = i^2$ edges and $\Delta = n$, so the number of iterations $i \in \Theta(n)$. In each step only one edge is being matched and if $i > 1$ the resulting subgraph is $G'_i = G_{i-1}$.[1]

If Kruskal is used to calculate the spanning tree the edges must be sorted only once so Kruskal yields a running time of $\mathcal{O}(sort(m) + \Delta m)$ which becomes a linear runtime for integer edge weights and $\mathcal{O}(m \log n + \Delta m)$ for comparison based sorting. With Jarnik-Prim the algorithm is then bounded by $\mathcal{O}(\Delta(n \log n + m))$. If the maximum degree is assumed neglectable, the runtime of MSTM is dominated by the spanning tree calculation and the choice of the appropriate spanning tree algorithm is dependent on the density of the graph. As interesting graphs for graph partitioning are mostly very dense, we opted for the Jarnik-Prim algorithm.

Obviously the behaviour of MSTM is very dependent on the structure of the maximum spanning tree as it is more likely to get better matchings on trees with smaller average degree. We will now introduce a simple tweak to the Jarnik-Prim algorithm that influences the height of the spanning tree produced. Jarnik-Prim scans all edges in a BFS-like manner and only adds an edge to the spanning tree if it improves the current weight of the tree. If the $>$ sign inside the relaxation loop is replaced by a $\geq$ sign, the resulting spanning tree changes as in Figure 4.5. The left graph shows the result of the standard Jarnik-Prim algorithm: First all edges of the central vertex are added to the spanning and as they already form a maximum spanning

[1] Note that the average degree $d(G) = \frac{2m}{n}$ does not give an upper bound to the iteration number, since $d(G)$ can be pushed down to nearly two by adding a path with arbitrary length to one of G_i's vertices.

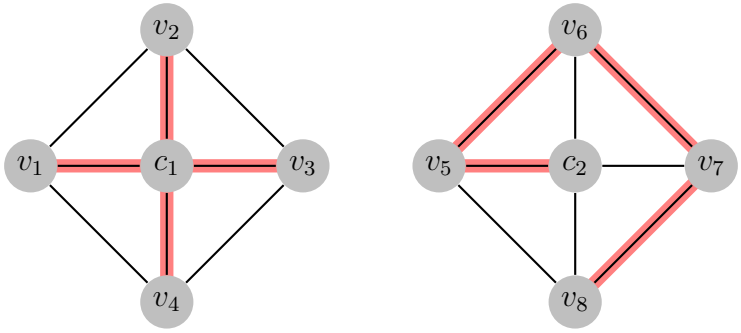


Figure 4.5.: The left graph shows the spanning tree calculated by $JP_<$ the right graph shows the result of $JP_\leq$

tree no further edges are appended. This yields a BFS-tree on which only one edge can be matched. With the modified relaxation however, the algorithm starts exactly the same. Now as each edge satisfies the condition from the relaxation loop, edges that are not part of the BFS-tree and therefore scanned later, participate in the spanning tree. It should be noted that while This change results in a DFS-tree on which more edges can be matched. As we will show in the experimental section 6.3.2 this also leads to higher spanning trees on real world graphs and therefore reduces the number of iterations needed significantly.

Chapter 5.

Edge Ratings

In order to design good edge ratings, we first have to discuss the parameters we will use to express how preferable an edge is for contraction. This section is organized as follows. First we discuss some criteria that have shown to be of use for construction of good edge ratings. Then we discuss several edge rating functions and how they implement these criteria.

5.1. Edge Rating Criteria

The first two criteria we introduce are almost necessary for a good edge rating and are implemented in nearly all of our edge rating functions.

Criterion 1: Since it is unlikely that an edge with (locally) very high weight will be contributing to a small cut we would like to prefer edges with (locally) high weight $\omega(e)$ for contraction.

Criterion 2: It may be desirable to coarse the graph in a balanced fashion, or in other words to avoid edges between nodes u, v with already high weight $c(u), c(v)$.

Ideally each node in the coarsest graph represents a very dense part of the original graph. Yet the tactic to achieve such a clustering of the graph depends heavily on the nature of the graph itself and the metric used to measure density. Therefore we have two seemingly contradicting criteria considering density of the graph.

Criterion 3: It is preferable to contract edges that lie in very the graph's dense parts.

Criterion 4: It is preferable to contract edges that lie in tree-like structures that are only connected to a single dense part of the graph.

These two criteria demand for global information about the structure of the graph. However, as we only employ local information to build edge ratings, criterion three leads to edge ratings contracting edges with low algebraic distance or high coreness of its vertices, yet we could also argue that criterion four leads to the exact opposite. It seems however, that it is easier to express criterion three with the algebraic distance and criterion four with coreness values.

Criterion 5: Combine ratings in a tuple in decreasing order of the probability that they assume the same value.

This last criterion is a rather technical one: as we use a component-wise comparison of the such tuples a high variance in the first component could mean that all the remaining components are never taken into account.

5.2. Basic Edge Ratings

Weight

Using the edge weight as a rating function is a natural way to implement the first criterion. Discussing experimental results we will often use this edge rating as a benchmark, looking at the relative improvement of some parameter over this function. While this rating seems crude, basically all of our ratings are based on it, in the sense that each rating consists of the edge rating divided by or multiplied by some value. This rating function has originally been introduced by [11] and [16].

InnerOuter

The intuition behind this rating function, originally proposed by [19], is that the number of edges inside a cluster should be higher than the edges going out of the cluster.

$$inner\_outer(e) = \frac{\omega(e)}{\Gamma(u) + \Gamma(v) - \omega(e)}$$

where $\Gamma(u) = \sum_{e \in E(u)} \omega(e)$. On the background of our criteria however, this rating function implements an edge's local heaviness of criterion one.

This rating function can be slightly modified. Here the edge weight is divided by the sum of all the edge weights in the neighbourhood of the edge's vertices.

$$LocalWeight(e) = \frac{\omega(e)}{\Gamma(u) + \Gamma(v)}.$$

This represents the idea behind criterion one more directly. As an edge lying in a very dense part of the graph is likely to connect vertices with high degree and similarly high Γ value, these ratings implement criterion four. Though both ratings are not the strongest in our collection they can be combined with other ratings to form multicriterial edge ratings (tuple edge ratings) and perform much better than the edge weight alone.

Expansion family

Ratings of the expansion family are inspired by the *edge expansion* or *cheeger number* $h(G)$ of a graph G.

$$h(G) = \min_{0<|S|<\frac{n}{2}} \frac{|\partial(S)|}{|S|}$$

where $\partial(S)$ is the edge boundary of S. To rate an edge based on this formula one could end up with the rating function *ExpansionClassic*.

$$ExpansionClassic(e = \{u, v\}) = \frac{\omega(e)}{min\{c(u), c(v)\}}$$

This rating function has originally been proposed in [19], where it is credited to P. Sanders. While this function indeed avoids edges between heavy vertices it still allows contraction of an edge connecting a light vertex with a very heavy vertex. By altering Expansion Classic one would end up - after some experimental evaluation - with the following ratings:

$$Expansion^*(e = \{u, v\}) = \frac{\omega(e)^2}{c(u)c(v)}$$

$$Expansion^{*2}(e) = \frac{\omega(e)^2}{c(u)c(v)}$$

$Expansion^{*2}$ is the standard rating function of KaFFPa [25], since it yields very good results on a broad set of graphs while being very easy to compute. The next two expansion variants have been optimized to perform very good on social networks. The first incorporates the concept of local heaviness into $Expansion^{*2}$ this function yields another rating:

$$LocalExpansion^{*2}(e) = \frac{\omega_n}{c(u)c(v)}$$

with

$$\omega_n := \frac{\omega(e)}{\Gamma(u) + \Gamma(v)}.$$

The second function is based on a reinterpretation of the expansion functions. The $min(\cdot)$ function in *ExpansionClassic* can be interpreted as the generalized mean

$$M_p(x_1, \ldots, x_n) = (\frac{1}{n} \sum_{i=1}^{n} x_i^p)^{1/p}$$

with exponent $p = -\infty$. Based on this idea, one could understand many of the rating functions of the expansion-family as the edge weight divided by such a mean over the node weights. $Expansion^{*2}$ for example can be rewritten as $(\omega(e)/M_0(c_u, c_v))^2$ as for $p \to 0$ the generalized mean converges towards the geometric mean. This lead us to another Expansion function:

$$HarmonicExpansion = 2\frac{\omega(e)}{M_{-1}(c_u, c_v)} = \omega(e)(\frac{1}{c(u)} + \frac{1}{c(v)})$$

with M_{-1} being the harmonic mean. Surprisingly this function is one of the best functions for social networks currently known to us.

All *Expansion* ratings implement criterion one and two in a very direct way and only differ in how strongly we emphasize the criteria.

Punch

Another rating proposed by [5] is quite similar to $HarmonicExpansion$ but with a different emphasis on the parameters involved.

$$Punch(e) = r|_b^a(\frac{\omega(e)}{\sqrt{c(u)}} + \frac{\omega(e)}{\sqrt{c(v)}})$$

where $r|_b^a$ is a random value between a and b. This rating function has been specifically designed for street networks and similar to the $Expansion$ ratings, it implements criterion one and two.

5.3. Density Metric Based Edge Ratings

We obviously need to minimize the algebraic distance as the intuition behind this metric is that a low algebraic distance means that the two vertices are close to each other. Such vertices could also be interpreted as lying in dense parts of the graph, so contracting an edge with low algebraic distance can be motivated by criterion three.

$$AlgebraicEXP(e) = \frac{\omega(e)^2}{c(u)c(v)\text{adist}(e)}$$
$$AlgebraicHEXP(e) = \frac{\omega(e)}{\text{adist}(e)}(\frac{1}{c(u)} + \frac{1}{c(v)})$$

These rating function are basically $Expansion^{*2}$ and $HarmonicExpansion$ that additionally prefers vertices with low algebraic distance. As the coreness of a vertex describes how deep it lies inside of the graph, we can use it to describe a rating function implementing criterion four.

$$LowCore(e) = \frac{\omega(e)^2}{\text{coreness}(u)\text{coreness}(v)}.$$

This rating function is designed for social networks, as here sparse parts of the graph are often solemnly connected to a single dense part and therefore can be safely contracted and an edge rating can benefit from implementing criterion four.

Another possible explanation for this rating can be given using the number of common neighbours of the edge's vertices. Consider an edge deep inside the graph (e.g. in a very dense part of the graph), then it is typical that the vertices of the edge share many common neighbours, thus contraction of such an edge will increase edge weights in the coarser graph. So $LowCore$ also gives a heuristic for avoiding high edge weights. We can also define a rating based on the number of common neighbours:

$$ExpansionCommon(e) = \frac{\omega(e)^2(|N(u) \cap N(v)| + 1)}{c(u)c(v)}$$

In addition to $LowCore$, the coreness of a vertex can also be used in to build rating functions implementing criterion three. By multiplying the term $HC = \max(\text{coreness}(u), \text{coreness}(v))$

to a rating functions of the *Expansion*-family. Furthermore, another idea that can be implemented using the coreness of vertices is to avoid the contraction of edges connecting vertices that lie in different k-cores. Again this can be achieved by multiplying another rating function with $DC = 1/(1 + |\text{coreness}(u) - \text{coreness}(v)|)$. The intuition is that such edges connect dense with sparse parts of the graph and are therefore important for the structure of the graph. In this work we only combine $Expansion^{*2}$ and $HarmonicExpansion$ with DC and HC as these two are the most promising candidates. From now on we abbreviate these function as $ExpansionHC$ or $HarmonicDC$.

5.4. Multicriteria Edge Ratings

All ratings from the *Expansion*-family have a very small variance in the first few levels of contraction, since at the first level all edges and vertices have a weight of one. So it might be interesting to define another rating function to fall back to, if two edges, that are to be compared, are rated equally. Or more generally we could use the ratings presented so far and combine them in a tuple. Now as we allow edge rating functions to be mapping to tuples, we have to decide how to compare these tuples with each other. We decided to use a component-wise comparison operator, such that we maximize for each entry of the tuple in order of its index.

As a simple example, criteria one and two can be implemented as this tuple-version of $Expansion$

$$ExpansionTuple(e) = \begin{pmatrix} 1/\min(c(u), c(v)) \\ \omega(e) \end{pmatrix}.$$

One could read $ExpansionTuple$ as: contract edges s.t. first $\min(c(u), c(v))$ is minimized and then $\omega(e)$ is maximized. Incorporating the vertex degrees instead of vertex weights could similarly lead to the following rating:

$$WxLD(e) = \begin{pmatrix} \omega(e) \\ 1/\min(d(u), d(v)) \end{pmatrix}.$$

All other multicriteria rating functions are a combination of the basic rating functions discussed above. The first two combine *inner_outer* with *Expansion*-ratings. *inner_outer* has been chosen to compensate for the lack of variance of *Expansion* mentioned above.

$$EXPxIO = \begin{pmatrix} Expansion^{*2} \\ inner\_outer \end{pmatrix}$$

$$HEXPxIO = \begin{pmatrix} HarmonicExpansion \\ inner\_outer \end{pmatrix}$$

Another possible set of rating functions is based on a similar idea. Here however, we minimize the edge degree instead of using *inner_outer*, which can also be seen as a simple implementation of criterion three.

$$EXPxLD = \begin{pmatrix} Expansion^{*2} \\ 1/\min(d(u), d(v)) \end{pmatrix}$$

$$HEXPxLD = \begin{pmatrix} HarmonicExpansion \\ 1/\min(d(u), d(v)) \end{pmatrix}$$

The last multicriteria edge rating combines two density-based metrics. It yields good results on social networks. Note that ratings, based on the algebraic distance should always be in the last component as it is very unlikely that two edges correspond to the same algebraic distance.

$$LCA = \begin{pmatrix} LowCore \\ AlgebraicExpansion \end{pmatrix}$$

Chapter 6.

Experimental Evaluation

6.1. KaFFPa

The main practical objective of this work is to improve the edge rating and matching subsystems of KaFFPa (Karlsruhe Fast Flow Partitioner) algorithm described in [25]. In this section we will give a compact overview of the main configuration settings of KaFFPa.

KaFFPa Strong

The aim of this configuration is to achieve the best known partitions for many standard benchmarks and as many graphs as possible. It uses GPA as a matching algorithm and $Expansion^{*2}$ as edge rating in the coarsening phase. As the refinement phase becomes very computational intense with this setting, we will try to achieve speed-ups by delivering a better coarsening scheme while not decreasing the quality of the produced partitions.

KaFFPa Eco

The aim of this configuration is to provide a graph partitioner that is as fast as other graph partitioners like Scotch, yet able to compute partitions of higher quality. It uses a random matching on the first levels of coarsening and applies GPA once the coarser graphs are small enough. Just like the strong setting the eco setting uses $Expansion^{*2}$ as standard rating.

6.2. Experiment Description

6.2.1. General Methodology

We are interested in both the quality of the partitions produced by KaFFPa Strong and Eco and the quality of the coarsening itself. In these tests we exchange the standard matching algorithms and edge rating used in the different settings of KaFFPa by the ones discussed in this work. For each setting that arises, we repeat the evaluation with 10 different initial seeds of the random number generator and for social networks we use 20 repetitions. We report the arithmetic average of the computed cut size, running time and best cut found. When further

averaging over multiple instances, we use the geometric mean in order to give every instance the same influence on the final score.

6.2.2. Matching Tests

Possible quality measures for a matching M can both be the percentage of edges (or nodes) matched $100|M|/m$ or the percentage of weight matched $100 \sum_{e \in M} \omega(e) / \sum_{e \in E} \omega(e)$. Since a matching is being calculated on each level of coarsening we also give the geometric mean of these values over all levels. Furthermore we report the overall number of levels generated in the coarsening phase.

6.2.3. MSTM Analysis

In order to get a deeper understanding of the MSTM algorithm we investigate besides the quality of the matchings produced, the number of iterations and its runtime. Also we are interested of the influence of the structure of the generated spanning tree on these metrics. We report the average height, number of leafs and the average degree of the generated spanning trees.

6.2.4. Environment

All experiments have been done on our InstitutsCluster (IC1), which is a massive hybrid parallel machine consisting of 206 nodes. One single node has a theoretical peak performance of 85.5 GFLOPS, so that the whole system yields a performance of 17.57 TFLOPS. Each node is running Suse Linux Enterprise (SLES) 11 SP 1 and is equipped with two Quad-Core Intel Xeon X5355 processors with a clock speed of 2.667 GHz and 4 x 250 GB harddrives. The 200 calculation nodes have 16 GB RAM and the six login nodes have 32 GB RAM. Each Quad-Core processor has 2 x 4 MB L2 Cache, runs the system bus with 1333 MHz and the front side bus (FSB) with 1066 MHz. All nodes are connected by a InfiniBand 4X DDR Interconnect with ConnectX Dual Port DDR HCAs with a total throughput rate of 288 x 40 Gb/s. The InfiniBand is characterized by a very low latency of below 2 microseconds and is ideal for heavily parallel applications with many collective MPI communication. In the appendix we give a schematic view of the IC1 in Figure A.1.

6.2.5. Test Sets

In this work we use the same test sets that has already been used to test KaFFPa itself [25]. Nonetheless we will present them here for completeness.

We have two kinds of random graphs: *rggX* are random geometric graphs with 2^X vertices that represent points in the unit square and edges connect nodes whose euclidean distance is below $0.55\sqrt{\ln n/n}$. This threshold ensures that the graph is almost connected. *DelaunyX* are delaunay triangulations of 2^X random points in the unit square. The graphs from Chris Walshaw's benchmark archive all arise from scientific computing and engineering [7]. Graphs

bel, nld, deu and *eur* are undirected versions of the road networks of Belgium, the Netherlands, Germany and Western Europe. *afShell9* and *afShell10* come from the University of Florida Sparse Matrix Collection [27]. The social networks *coAuthorsCiteseer, coPapersCiteseer* and *citationCiteseer* come from *Citeseer* project of Pennsylvania State University [28], *coPapers-DBLP* and *coAuthorsDBLP* come from *DBLP* project of University of Trier [18] and *cnr2000* is a small web crawl of Italian CNR domain. Here vertices represent authors and edges some kind of relation between authors e.g. if they have at least published one paper together.

For the number of partitions we choose the common values: 2, 4, 8, 16, 32, 64. We allow an imbalance of 3%, which also is the default for Metis. These values have already been used in [25, **?**, 20] and [29]. Considering social networks we only evaluated the Strong setting of KaFFPa for $k = 2$.

6.3. Matchings

6.3.1. Comparison of Matching Algorithms

We will first discuss the performance of the different matching algorithms on the middle sized test set. In Table 6.3 the average and maximum values suggest that MSTM and GPA both perform better than every other matching algorithm, while GPA performs slightly better. It should be noted that the very high maximum values of $RandomMSTM$ and $RandomGPA$ only arise directly after the randomly matched levels when for the first time the actual matching algorithm gets applied. This can only be seen when looking at the average levels needed in the coarsening phase and the initial cut. Since we are not directly interested in a good general matching algorithm, but rather a good matching algorithm for use in a coarsening phase, another very important quality measure is the initial cut produced directly after the coarsening phase. Here too, we can clearly see that the best algorithms are MSTM and GPA, again GPA is slightly better. We can also see that MSTM is significantly slower than GPA (often near a factor of two). Matching results on the social networks test set are given in Table 6.4. Here we have a very similar situation as on the middle sized test set, though this time MSTM yields slightly better results than GPA.

6.3.2. Detailed Evaluation of MSTM

In this section we will turn to a more in detail evaluation of the MSTM algorithm, especially on how the structure of the produced spanning trees influence the runtime and the number of iterations needed. In Table 6.5 the two variants of the Jarnik-Prim algorithm, we introduced earlier are compared to each other. One can see that the influence on some parameters of the spanning trees can become tremendous, especially the number of leafs decreases by nearly a factor of five while the average degree drops to an amount one would expect of a DFS-tree. It must be noted, that the two variants produce exactly the same results if the rating function used has a very high variance, so that it is unlikely or even impossible that two edges have the same weight. Surprisingly the average percentages of weight matched does not seem to be affected too much and even decreases on the first level. Yet MSTM $JP_{>}$ uses up much more iterations until it converges to similar or even better results than $JP_{\geq}$. Despite these results

graph	n	m	δ	Δ	ϵ	max-core
Medium sized instances						
Random graphs						
Delaunay17	131 072	786 352	3	17	6.0	4
rgg17	131 072	1 457 506	0	28	11.1	14
Delaunay18	262 144	1 572 792	3	21	6.0	4
rgg18	262 144	3 094 566	0	31	11.8	16
Walshaw's Benchmark Archive						
bcsstk29	13 992	605 496	4	70	43.3	29
4elt	15 606	91756	3	10	5.9	4
fesphere	16 386	98 304	4	6	6.0	5
cti	16 840	96464	3	6	5.7	4
memplus	17 758	108 392	1	573	6.1	96
cs4	22 499	87 716	2	4	3.9	3
fepwt	36 519	289 588	0	15	7.9	5
bcsstk32	44 609	1 970 092	1	215	44.2	69
febody	45 087	327 468	0	28	7.3	6
t60k	60 005	178 880	2	3	3.0	2
wing	62 032	243 088	2	4	3.9	3
brack2	62 631	733 118	3	32	11.7	7
finan512	74 752	522 240	2	54	7.0	6
ferotor	99 617	1 324 862	5	125	13.3	8
Road Networks						
bel	463 514	1 183 764	0	8	2.6	3
nld	893 041	2 279 080	0	7	2.6	3
University Florida Sparse Matrix Collection (Matrix Models)						
afshell9	504 855	17 084 020	19	39	33.8	24

Table 6.1.: Some characteristics of the medium sized test set. Values based on a directed representation of the undirected graphs, so m and ϵ for the undirected version are only half the values given here

graph	n	m	Δ	*delta*	ϵ	max-core
Large sized instances						
Random graphs						
Delaunay20	1 048 576	6 291 372	3	23	6.0	4
rgg20	1 048 576	13 783 240	0	36	13.1	17
Walshaw's Benchmark Archive						
fetooth	78 136	905 182	3	39	11.6	7
598a	110 971	1 483 868	5	26	13.4	8
feocean	143 437	819 186	1	6	5.7	4
144	144 649	2 148 786	4	26	14.9	9
wave	156 317	2 118 662	3	44	13.6	8
m14b	214 765	3 358 036	4	40	15.6	9
auto	448 695	6 629 222	4	37	14.8	9
Road Networks						
deu	4 378 446	10 967 174	0	8	2.5	3
eur	18 029 721	44 435 372	0	12	2.5	4
Social Networks						
coAuthorsCiteseer	227 320	1 628 268	1	1 372	7.2	86
citationCiteseer	268 495	2 313 294	1	1 318	8.6	15
coAuthorsDBLP	299 067	1 955 352	1	336	6.5	114
cnr2000	325 557	5 477 938	1	18 236	16.8	83
coPapersCiteseer	434 102	32 073 440	1	1 188	73.9	844
coPapersDBLP	540 486	30 491 458	1	3 299	56.4	336
University Florida Sparse Matrix Collection (Matrix Models)						
afshell10	1 508 065	51 164 260	14	34	34.0	19

Table 6.2.: Some characteristics of the large sized test set. Values based on a directed representation of the undirected graphs

		GPA	HEM	MSTM	RandomGPA	RandomMSTM	SHEM
$Expansion^{*2}$	levels	9.95	10.05	9.34	10.33	10.39	9.33
	avg	33.08	27.44	30.68	26.24	25.84	24.94
	max	52.57	56.00	43.59	55.46	55.51	26.90
	time	0.52	0.10	0.98	0.02	0.04	0.11
	init	907	1123	974	1204	1228	1114
$Weight$	levels	12.15	11.42	10.13	10.58	11.20	9.35
	avg	25.85	26.23	23.79	22.48	23.47	17.80
	max	48.14	43.88	45.22	55.85	57.33	35.63
	time	0.49	0.043	0.99	0.02	0.05	0.05
	init	989	1235	991	1322	1222	1114

Table 6.3.: Matching test results for the middle sized test set on KaFFPa with $Expansion^{*2}$ and $Weight$. The first line gives the number of levels computed in the coarsening phase, the second line gives the average percentage of weight matched and the third line gives the maximal percentage. Time is given as the overall time needed for the coarsening phase in seconds.

		GPA	HEM	MSTM	RandomGPA	RandomMSTM	SHEM
$Expansion^{*2}$	levels	12.56	10.75	12.77	9.08	9.23	9.16
	avg	19.61	19.46	20.77	11.25	10.40	17.34
	max	50.96	44.88	51.49	35.09	35.22	41.83
	time	9.17	1.32	13.38	0.48	0.56	1.24
	init	89266	93953	95547	137464	137011	143527
$Weight$	levels	8.35	6.30	9.80	7.01	8.77	9.16
	avg	12.21	9.63	11.75	8.64	9.58	7.18
	max	19.69	15.24	23.70	17.34	19.70	15.21
	time	6.52	0.50	12.90are	0.30	0.51	0.62
	init	95103	99916	95442	139165	139780	141337

Table 6.4.: Matching test results for the social networks set on KaFFPa with $Expansion^{*2}$ and weight.

MST ALGO	$JP_{\geq}$			$JP_{>}$		
Rating	$Weight$	$LocalExp^{*2}$	LCA	$Weight$	$LocalExp^{*2}$	LCA
# leafs	7719.01	19036.36	24171.07	38776.02	31656.98	24171.07
height	11848.52	1700.50	564.26	9458.74	1455.99	564.26
max. deg	4.45	7.21	7.72	15.57	8.10	7.72
avg. deg	1.21	1.68	1.62	3.07	2.31	1.62
weight # 1	12.88	18.51	14.89	13.42	18.58	14.89
avg. weight	23.79	40.15	33.04	23.67	39.89	33.04
init. cut	991.10	953.16	921.63	989.40	964.04	921.69
iterations	1.95	3.93	3.08	4.21	4.75	3.08
time	1.01	1.15	1.08	1.15	1.18	1.09

Table 6.5.: Different quality metrics of MSTM and the spanning trees produced. In the first block the spanning tree parameters are listed, namely the number of leafs, the height of the trees, the average and the maximum degree. The second block presents the percentage of matched weight on the first level and averaged over all levels. Also the initial cut that is produced after the initial partitioning is given. The last block gives the number of iterations and runtime of MSTM in seconds

$JP_{\geq}$ only leads to 10 percent faster running times. It could be possible though, that other constant factors we could not cancel out are responsible for these running times.

6.4. Edge Ratings

6.4.1. Middle sized test set

We first evaluate the impact of different edge ratings on the runtime and the final cut computed by KaFFPa. If the Eco setting is used $Expansion^{*2}$ is a very strong rating function. Results of the improvement over using the $Weight$ as rating function are given in Table 6.6. In Table 6.7 the best three rating functions other than $Expansion^{*2}$ are given. Here we can only see that no edge rating yields any significant improvement over $Expansion^{*2}$ and on the other hand most of the ratings yield a similar improvement over $Weight$. Here the most significant impact of the usage of MSTM as the matching algorithm is an increase of the runtime, while the average cut increases and a slight improvement on the minimal cuts can be stated. Though these results may seem disappointing they indicate that the standard Eco setting is already very well tuned.

In Tables 6.8 and 6.8 cuts and speed-ups for the Strong setting of KaFFPa are given. The most interesting result surely is the improvement of the average cuts for the $LocalExpansion$ and $AlgebraicExpansion$ rating functions. If we keep in mind that this setting is already designed to produce the best cuts available, these results clearly state that these rating functions can

	KaFFPa Eco with RandomGPA			KaFFPa Eco with RandomMSTM		
# Partitions	avg Cut	min Cut	speed-up	avg Cut	min Cut	speed-up
2	6.0	3.7	1.030	5.3	4.0	0.876
4	10.1	5.0	1.033	9.5	5.1	0.907
8	6.1	2.9	1.042	5.5	3.0	0.915
16	6.3	5.5	1.051	5.9	5.4	0.890
32	3.7	3.3	1.025	3.4	3.1	0.887
64	0.7	0.7	0.997	0.5	0.3	0.880
overall	5.4	3.5	1.030	5.0	3.5	0.892

Table 6.6.: $Expansion^{*2}$ on middle sized graphs. Cuts are given as improvements over the weight rating function, times are given as speed-up factors.

matching	rating	avg Cut	min Cut	speed-up
RandomGPA	$ExpHC$	-0.1	-0.1	0.946
	$HEXPxIO$	-0.1	0.1	0.978
	$LocalExpansion$	-0.2	-0.1	0.99
RandomMSTM	$inner\_outer$	-0.1	0.2	0.877
	$EXPxIO$	-0.2	0.2	0.875
	LCA	-0.2	0.0	0.831

Table 6.7.: Top three ratings for the middle sized test suit, calculated by KaFFPa Eco. Cuts given as improvements over $Expansion^{*2}$ rating function using RandomGPA. Times are given as speedup factors

	KaFFPa Strong with GPA			KaFFPa Strong with MSTM		
# Partitions	avg. Cut	min Cut	speed-up	avg. Cut	min Cut	speed-up
2	0.7	0.1	1.186	0.3	-0.1	1.379
4	1.8	0.5	1.197	1.9	0.3	1.391
8	2.4	1.8	1.206	2.4	2.0	1.370
16	2.3	1.9	1.154	2.1	1.6	1.310
32	1.1	0.5	1.182	0.5	0.4	1.352
64	0.1	-0.1	1.145	-0.2	-0.4	1.263
overall	1.4	0.8	1.178	1.2	0.6	1.343

Table 6.8.: $Expansion^{*2}$ on middle sized graphs

matching	rating	avg Cut	min Cut	speed-up
GPA	$LocalExpansion$	0.4	0.1	1.04
	$AlgebraicExpansion$	0.4	0.1	0.965
	LCA	0.3	0.2	0.953
MSTM	$ExpansionCommon$	0.2	0.0	1.051
	$AlgebraicExpansion$	0.2	0.2	1.081
	LCA	0.2	0.0	1.037

Table 6.9.: Top three ratings for the middle sized test suit, calculated by KaFFPa Strong. Results are given as improvements over $Expansion^{*2}$ using GPA.

significantly improve the performance of KaFFPa. MSTM yields similar results, although the improvement of the cuts is not as notable.

6.4.2. Social Networks

In this section we will discuss how the picture changes if we look at social networks. As listed in Table 6.10 $Expansion^{*2}$ already performs much better than the weight rating function and if MSTM is employed as the matching algorithm we obtain significantly better cuts. As presented in Table 6.11 though, other rating functions yield even stronger improvements.

Here the multicreteria edge rating $HEXPxLD$ produces 8 percent better cuts than $Expansion^{*2}$ using GPA and with MSTM $AlgebraicHEXP$ even produces 10 percent smaller cuts and 7 percent smaller minimal cuts than $Expansion^{*2}$.

The most surprising result is the strong improvement of the partition quality and a slight speed-up when MSTM is used as the matching algorithm. However, these results could not be directly deduced from the matching tests, as both matching algorithms produced nearly the same results.

This image changes radically if we consider the Strong setting of KaFFPa listed in Tables 6.12 and 6.13. Due to the very high runtime of KaFFPa Strong on social networks we only

	KaFFPa Eco with RandomGPA			KaFFPa Eco with RandomMSTM		
# Partitions	avg. Cut	min Cut	speed-up	avg. Cut	min Cut	speed-up
2	5.8	11.6	0.932	5.9	7.3	0.938
4	6.0	4.8	1.013	13.6	14.3	1.037
8	4.1	1.0	1.059	9.0	5.8	1.053
16	3.7	7.8	0.955	9.4	12.7	0.888
32	11.5	8.8	0.832	6.9	10.7	1.046
64	8.2	13.7	0.973	20.5	15.4	0.777
overall	6.5	7.9	0.958	10.8	11.0	0.951

Table 6.10.: $Expansion^{*2}$ on social networks. Cuts are given as improvements over the weight rating function

matching	rating	avg Cut	min Cut	speed-up
RandomGPA	$HEXPxLD$	8.4	5.2	0.843
	$AlgebraicHEXP$	7.9	6.7	0.898
	$HarmonicExpansion$	7.765	4.231	0.864
RandomMSTM	$AlgebraicHEXP$	10.2	7.4	0.914
	$LowCore$	10.2	7.0	0.936
	$HEXPxIO$	9.8	6.4	0.857

Table 6.11.: Top three ratings for the social networks, calculated by KaFFPa Eco. Cuts given as relative improvements over the $Expansion^{*2}$ rating function using RandomGPA.

GPA			MSTM		
avg Cut	min Cut	speed-up	avg Cut	min Cut	speed-up
6.2	16.5	0.940	16.4	22.3	1.239

Table 6.12.: The improvement of $Expansion^{*2}$ over weight for the social networks, calculated by KaFFPa Strong and $k = 2$.

matching	rating	avg Cut	min Cut	speed-up
GPA	$ExpHC$	0.6	0.3	0.877
	$HarmonicExpansion$	-2.2	-1.0	0.996
	$EXPxIO$	-4.7	1.3	1.015
MSTM	$Expansion^{*2}$	9.6	5.0	1.249
	$Weight$	2.4	1.3	1.585
	$WxLD$	2.0	-3.3	1.44

Table 6.13.: Top three ratings for the social networks, calculated by KaFFPa Strong and $k = 2$. Cuts given as relative improvements over the $Expansion^{*2}$ rating function using GPA.

evaluated the edge ratings for $k = 2$. One can see that $Expansion^{*2}$ yields strong improvements over the $Weight$ rating function, especially if MSTM is used as a matching algorithm. If one considers the improvement of different rating functions over $Expansion^{*2}$ using GPA as a matching algorithm only $ExpHC$ yields improvements. On the other hand if MSTM is used with $Expansion^{*2}$, the results clearly outperform GPA in terms of cut size and runtime. Yet none of the rating functions that performed well on social networks with the Eco setting reproduced these results on the Strong setting. However, the major improvements of the Eco setting only arise with higher k values (as listed in in Table B.7) and as we did not compute results for the Strong setting on social networks for higher k values, we cannot clearly rule out that none of the other rating functions could perform well.

6.4.3. Additional quality metrics

In Tables 6.14, 6.15 and 6.16 we list some additional quality metrics for the different experiments even though we did not optimize any of the ratings towards a quality metric other than the edge cut. Surprisingly nearly all results show that a low edge cut induces a low communication volume and few boundary nodes. However the multicriteria rating functions using the Strong setting in Table 6.15 represent one notable exception as here the edge cut does not reflect the results of the additional metrics.

matching	rating	∂_G	∂_{max}	$comm_G$	$comm_{max}$
RandomGPA	$ExpHC$	0.0	-0.2	-0.1	-0.4
	$HEXPxIO$	-0.1	-0.4	-0.2	-0.4
	$LocalExpansion$	-0.2	-0.1	-0.3	-0.2
RandomMSTM	$inner\_outer$	0.1	-0.2	0.1	-0.2
	$ExpansionCommon$	0.0	0.0	0.0	-0.3
	$LowCore$	-0.1	0.1	-0.2	-0.4

Table 6.14.: Top three ratings for the middle sized test suit, calculated by KaFFPa Eco. Boundary nodes and communication volume given as improvements over $Expansion^{*2}$ rating function using RandomGPA

matching	rating	∂_G	∂_{max}	$comm_G$	$comm_{max}$
GPA	$HEXPxLD$	0.3	0.4	0.2	0.7
	$Lowcore$	0.3	0.5	0.3	0.6
	$EXPxLD$	0.1	0.5	0.0	0.7
MSTM	$EXPxLD$	0.6	0.6	0.5	0.8
	$HEXPxLD$	0.6	0.9	0.5	1.1
	$WxLD$	0.5	1.0	0.4	1.4

Table 6.15.: Top three ratings for the middle sized test suit, calculated by KaFFPa Strong.

matching	rating	∂_G	∂_{max}	$comm_G$	$comm_{max}$
RandomGPA	$HEXPxLD$	3.5	4.6	5.7	7.7
	$ExpansionTuple$	1.7	7.3	1.7	9.2
	$HEXPxIO$	3.3	4.4	5.3	6.5
RandomMSTM	$HEXPxLD$	4.4	6.7	6.8	11.5
	$HexpHC$	3.9	7.8	6.0	12.6
	$HEXPxIO$	4.4	6.9	6.9	11.4

Table 6.16.: Top three ratings for social networks, calculated by KaFFPa Eco.

Chapter 7.

Conclusion

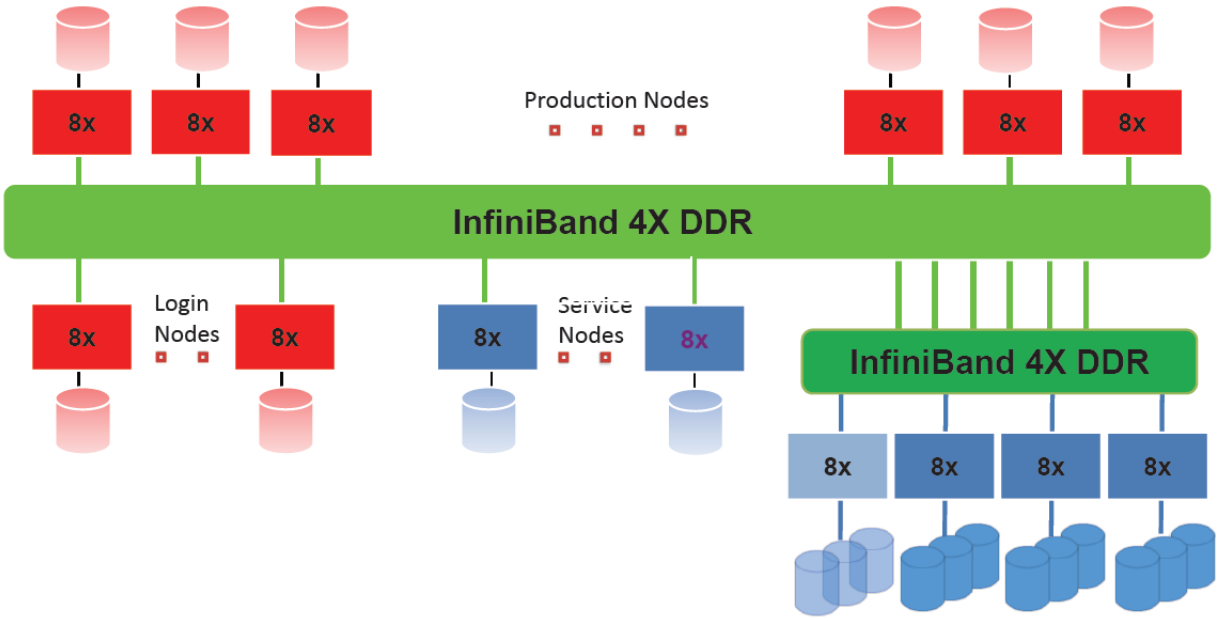
In this thesis we presented different approaches to improve the coarsening phase of multi-level graph partitioners and evaluated the findings with KaFFPa. Namely we discussed new edge ratings based on a few simple construction rules. We showed how to built such ratings using only local information and well-known local density measures. We also showed how to arrange basic edge ratings together as tuples.

Furthermore we presented the $MSTM$-algorithm, a new approximative algorithm for the maximum weight matching problem, and showed that together with our improved edge ratings, we could achieve significant speed-ups for the strongest setting of the graph partitioner used and considerable improvements in the partition quality of social networks.

Appendix A.

InstitutsCluster

Figure A.1.: InstitutsCluster at SCC of University of Karlsruhe



Appendix B.

Extra Tables

Middle sized test set						
	inner_outer	*ExpLocal*	*AlgHEXP*	*ExpCommon*	*ExpHC*	*HEXPxLD*
avg. cut	-1.6	-0.2	-0.4	-0.5	-0.1	-0.3
min. cut	-1.5	-0.1	0.1	-0.3	-0.1	-0.2
time	0.973	0.99	0.897	0.918	0.946	0.981
avg. cut	-0.1	-0.4	-0.6	-0.2	-0.3	-0.5
min. cut	0.2	0.0	-0.1	0.0	-0.2	0.0
time	0.877	0.888	0.834	0.84	0.834	0.9
avg. cut	0.3	0.4	0.3	-0.2	0.0	0.0
min. cut	0.1	0.1	0.1	0.0	-0.1	-0.1
time	1.076	1.04	0.996	1.025	0.982	1.1
avg. cut	-0.4	-0.4	0.1	0.2	-0.3	-0.3
min. cut	-0.4	-0.3	0.0	0.0	-0.1	-0.4
time	1.139	1.121	1.1	1.051	1.092	1.137

Social networks						
	inner_outer	*ExpLocal*	*AlgHEXP*	*ExpCommon*	*ExpHC*	*HEXPxLD*
avg. cut	-2.1	6.9	7.9	-3.6	-0.3	8.4
min. cut	-0.1	5.2	6.7	-0.6	0.2	5.2
time	1.07	0.887	0.898	1.011	0.985	0.843
avg. cut	3.9	8.7	10.2	2.6	4.3	9.6
min. cut	5.0	6.7	7.4	2.7	2.3	7.0
time	0.961	0.903	0.914	0.965	0.951	0.866
avg. cut	-4.6	-4.6	-8.5	-10.9	0.6	-7.0
min. cut	-13.6	-4.6	-16.0	-21.1	0.3	-11.9
time	1.452	1.338	0.487	1.258	0.877	0.813
avg. cut	-9.2	-8.4	-6.2	-11.3	-0.6	-9.6
min. cut	-19.6	-3.0	-13.7	-20.4	-0.9	-3.1
time	1.569	1.64	0.803	1.17	0.908	1.348

Table B.1.: A summary of the best rating functions presented. The upper table lists improvements over $Expansion^{*2}$ on the middle sized testset and the lower table on the social networks. The first two rows are calculated by KaFFPa Eco and the last two rows by KaFFPa Strong.

Middle sized test set						
	Expansion	*Expansion**	*Localweight*	*ExpDC*	*HexpDC*	*Punch*
avg. cut	-0.7	-1.3	-1.1	-0.6	-0.7	-0.7
min. cut	-0.4	0.0	-1.1	-0.7	-0.6	-0.1
time	1.003	1.004	0.978	0.938	0.944	0.913
avg. cut	-0.5	-0.5	-0.3	-0.3	-0.3	-0.3
min. cut	0.0	-0.1	0.1	0.1	0.0	0.3
time	0.888	0.893	0.87	0.832	0.842	0.839
avg. cut	0.1	0.1	0.3	-0.2	-0.5	0.0
min. cut	0.0	0.1	0.1	-0.3	-0.4	0.0
time	1.095	1.063	1.07	0.922	0.96	0.937
avg. cut	-0.3	-0.1	-0.4	-0.2	-0.4	0.0
min. cut	-0.2	-0.2	-0.5	-0.3	-0.1	-0.1
time	1.22	1.18	1.143	1.04	1.018	1.071

Social networks						
	Expansion	*Expansion**	*Localweight*	*ExpDC*	*HexpDC*	*Punch*
avg. cut	7.3	7.7	-1.9	-4.2	-7.5	-2.9
min. cut	4.6	4.7	0.0	-3.9	-6.1	0.5
time	0.873	0.866	1.063	0.983	0.936	1.002
avg. cut	9.7	9.3	4.3	-1.1	-3.4	5.0
min. cut	8.4	6.4	3.3	-0.8	-3.8	3.9
time	0.906	0.903	0.967	0.978	0.94	0.956
avg. cut	-8.1	-5.6	-10.7	-11.0	-12.3	-2.2
min. cut	-15.8	-17.3	-21.9	-18.3	-19.3	-1.0
time	0.546	0.844	1.444	1.06	0.956	0.996
avg. cut	-5.3	-7.0	-7.9	-9.6	-15.1	-4.0
min. cut	-7.1	4.3	-13.0	-5.8	-16.0	-0.9
time	1.496	1.777	1.526	1.18	1.15	0.918

Table B.2.: A summary of several rating functions, that did not yield results good enough to be listed in one of the top three listings. The upper table lists improvements over $Expansion^{*2}$ on the middle sized testset and the lower table on the social networks. The first two rows are calculated by KaFFPa Eco and the last two rows by KaFFPa Strong.

	KaFFPa Eco with RandomGPA			KaFFPa Eco with RandomMSTM		
# Partitions	avg. Cut	min Cut	speed-up	avg. Cut	min Cut	speed-up
	ExpHC			*inner_outer*		
2	0.3	0.3	0.887	-0.4	-0.7	0.84
4	-0.1	-0.3	0.954	0.2	0.9	0.904
8	-0.5	0.0	0.944	-0.3	0.8	0.89
16	-0.2	0.0	0.973	-0.3	0.0	0.87
32	-0.2	-0.2	0.979	0.0	-0.2	0.872
64	-0.1	-0.3	0.942	0.1	0.1	0.898
overall	-0.1	-0.1	0.946	-0.1	0.2	0.877
	HEXPxIO			*EXPxIO*		
2	-0.3	0.1	0.952	-0.3	0.4	0.861
4	0.6	1.4	0.985	0.1	1.0	0.897
8	-0.3	-0.6	0.994	-0.5	0.3	0.889
16	-0.3	0.2	0.978	-0.3	-0.3	0.844
32	-0.4	-0.4	0.979	-0.1	-0.5	0.872
64	-0.1	-0.2	0.978	0.0	0.0	0.889
overall	-0.1	0.1	0.978	-0.2	0.2	0.875
	LocalExpansion			*LCA*		
2	-0.5	0.0	0.979	-1.0	0.1	0.793
4	0.8	0.5	0.994	0.6	0.5	0.85
8	-0.9	-0.2	1.004	-0.8	-0.2	0.85
16	-0.4	-0.1	0.995	0.0	0.0	0.821
32	-0.1	-0.2	0.983	0.0	-0.1	0.817
64	-0.5	-0.7	0.986	-0.1	-0.1	0.858
overall	-0.2	-0.1	0.99	-0.2	0.0	0.831

Table B.3.: Top three ratings for the middle sized test suit, calculated by KaFFPa Eco. Cuts given as improvements over $Expansion^{*2}$ rating function, times are given as speed-up factors.

	KaFFPa Eco with RandomGPA				KaFFPa Eco with RandomMSTM			
# Partitions	∂_G	∂_{max}	$comm_G$	$comm_{max}$	∂_G	∂_{max}	$comm_G$	$comm_{max}$
	ExpHC				*inner_outer*			
2	0.3	0.4	0.3	0.4	0.7	1.0	0.7	1.0
4	-0.4	-1.0	-0.4	-0.9	0.1	-0.3	0.1	-0.4
8	0.2	0.3	0.0	-0.2	-0.2	-2.1	-0.3	-2.4
16	0.1	-0.6	-0.1	-0.6	-0.1	1.0	-0.1	1.4
32	-0.3	0.2	-0.4	0.2	-0.2	-0.5	-0.1	-0.1
64	-0.1	-0.5	-0.1	-1.6	0.1	-0.2	0.1	-0.5
overall	0.0	-0.2	-0.1	-0.4	0.1	-0.2	0.1	-0.2
	HEXPxIO				*ExpansionCommon*			
2	0.1	0.1	0.1	0.1	0.0	0.3	0.0	0.3
4	0.2	1.2	0.1	1.1	0.1	-0.1	0.1	-0.3
8	-0.2	-0.8	-0.2	-0.7	0.1	-1.5	0.1	-1.9
16	-0.3	-0.2	-0.3	-0.1	0.1	1.7	0.0	1.8
32	-0.4	-1.1	-0.5	-1.7	0.0	0.3	0.0	-0.1
64	-0.2	-1.3	-0.2	-1.1	-0.3	-0.9	-0.3	-1.9
overall	-0.1	-0.4	-0.2	-0.4	0.0	0.0	0.0	-0.3
	LocalExpansion				*LowCore*			
2	-0.2	0.0	-0.2	0.0	-0.1	0.1	-0.1	0.1
4	0.6	-0.5	0.6	-0.5	-0.8	-0.2	-0.8	-0.3
8	-1.0	-1.5	-1.2	-2.0	-0.2	-0.7	-0.6	-1.6
16	-0.1	1.2	0.0	2.0	0.1	1.4	0.1	1.1
32	-0.2	0.3	-0.2	0.2	0.0	-0.1	0.1	-0.5
64	-0.5	-0.1	-0.5	-0.9	0.2	-0.1	0.2	-1.1
overall	-0.2	-0.1	-0.3	-0.2	-0.1	0.1	-0.2	-0.4

Table B.4.: Top three ratings for the middle sized test suit, calculated by KaFFPa Eco. Boundary nodes and communication volume given as improvements over $Expansion^{*2}$ rating function

	KaFFPa Strong with GPA			KaFFPa Strong with MSTM		
# Partitions	avg. Cut	min Cut	speed-up	avg. Cut	min Cut	speed-up
	LocalExpansion			*ExpansionCommon*		
2	0.3	-0.2	1.031	0.4	-0.1	1.052
4	1.1	0.1	1.034	0.2	0.2	1.087
8	0.5	-0.1	1.066	0.3	0.1	1.03
16	0.7	0.1	1.044	0.5	-0.2	1.034
32	0.0	0.3	1.03	0.0	0.4	1.05
64	0.1	0.2	1.034	-0.1	-0.1	1.053
overall	0.4	0.1	1.04	0.2	0.0	1.051
	AlgebraicExpansion					
2	0.1	-0.1	0.903	-0.1	-0.1	1.09
4	0.8	0.1	0.944	0.4	0.6	1.102
8	0.7	0.0	0.944	0.3	0.0	1.072
16	0.7	-0.1	0.967	0.5	0.1	1.072
32	0.2	0.5	1.01	-0.1	0.2	1.079
64	0.2	0.3	1.029	0.2	0.3	1.069
overall	0.4	0.1	0.965	0.2	0.2	1.081
	LCA					
2	0.4	-0.1	0.87	0.5	0.0	1.032
4	0.1	-0.2	0.916	0.6	0.3	1.049
8	0.4	0.4	0.932	0.1	-0.2	1.032
16	0.6	0.5	0.964	0.1	-0.1	1.025
32	0.1	0.4	1.012	-0.1	0.2	1.054
64	0.2	0.3	1.032	-0.1	0.0	1.03
overall	0.3	0.2	0.953	0.2	0.0	1.037

Table B.5.: Some ratings for the middle sized test suit, calculated by KaFFPa Strong. Cuts given as relative improvements over the weight rating function, times are given as speed-up factors.

	KaFFPa Strong with GPA				KaFFPa Strong with MSTM			
# Partitions	∂_G	∂_{max}	$comm_G$	$comm_{max}$	∂_G	∂_{max}	$comm_G$	$comm_{max}$
	$HEXPxLD$				$EXPxLD$			
2	0.6	0.8	0.6	0.8	0.7	0.7	0.7	0.7
4	0.4	1.7	0.5	1.7	0.6	1.0	0.6	1.0
8	0.3	0.0	0.2	-0.2	1.2	0.4	1.1	0.2
16	0.6	0.0	0.5	0.7	0.8	0.8	0.8	0.6
32	-0.2	0.3	-0.3	0.2	0.1	0.6	0.0	1.1
64	0.1	-0.1	-0.1	1.3	0.4	0.2	0.2	1.3
overall	0.3	0.4	0.2	0.7	0.6	0.6	0.5	0.8
	$Lowcore$				$HEXPxLD$			
2	0.1	0.3	0.1	0.3	0.1	0.6	0.1	0.6
4	1.1	0.9	1.0	0.9	0.4	1.8	0.4	1.7
8	-0.1	-1.3	-0.2	-1.4	1.6	1.2	1.5	1.1
16	0.3	0.1	0.3	0.0	0.8	0.5	0.7	0.9
32	0.2	1.3	0.0	2.1	0.2	0.7	0.0	1.2
64	0.1	1.5	0.1	1.7	0.2	0.4	0.0	1.4
overall	0.3	0.5	0.3	0.6	0.6	0.9	0.5	1.1
	$EXPxLD$				$WxLD$			
2	0.2	0.3	0.2	0.3	0.2	0.4	0.2	0.4
4	-0.4	1.1	-0.4	1.1	0.1	0.7	0.1	0.7
8	0.5	-0.4	0.4	-0.6	1.3	1.8	1.2	1.5
16	0.4	1.2	0.3	1.3	0.6	0.9	0.7	1.0
32	0.0	0.7	-0.1	0.9	0.4	1.6	0.2	2.6
64	0.1	0.3	-0.1	1.3	0.4	0.5	0.1	2.2
overall	0.1	0.5	0.0	0.7	0.5	1.0	0.4	1.4

Table B.6.: Top three ratings for the middle sized test suit, calculated by KaFFPa Strong. Boundary nodes and communication volume given as improvements over $Expansion^{*2}$ rating function

	KaFFPa Eco with RandomGPA			KaFFPa Eco with RandomMSTM		
# Partitions	avg. Cut	min Cut	speed-up	avg. Cut	min Cut	speed-up
	HEXPxLD			*AlgebraicHEXP*		
2	4.5	1.4	0.859	5.8	7.6	0.971
4	14.0	11.4	0.823	17.5	13.8	0.944
8	0.9	3.3	0.92	7.5	8.3	0.997
16	17.4	7.6	0.827	16.8	10.5	0.87
32	8.0	2.8	0.82	8.5	3.1	0.88
64	6.2	4.9	0.811	5.8	1.5	0.834
overall	8.4	5.2	0.843	10.2	7.4	0.914
	AlgebraicHEXP			*LowCore*		
2	5.2	4.0	0.918	3.3	3.9	1.028
4	10.3	8.4	0.94	15.1	10.8	0.959
8	6.8	2.2	0.823	6.2	1.5	0.852
16	14.8	8.5	0.865	18.6	9.8	0.907
32	7.1	12.2	0.846	9.5	9.2	0.886
64	3.4	5.2	1.014	9.1	6.9	0.998
overall	7.9	6.7	0.898	10.2	7.0	0.936
	HarmonicEXP			*HEXPxIO*		
2	4.5	1.6	0.884	6.7	6.0	0.902
4	11.5	7.5	0.832	17.5	11.7	0.817
8	1.8	2.4	0.972	7.4	7.6	0.933
16	15.7	7.4	0.861	16.0	6.7	0.87
32	8.1	2.8	0.841	7.2	6.3	0.822
64	5.6	3.9	0.807	4.8	0.2	0.807
overall	7.765	4.231	0.864	9.8	6.4	0.857

Table B.7.: Top three ratings for the social networks, calculated by KaFFPa Eco. Cuts given as relative improvements over the weight rating function, times are given as speed-up factors.

	KaFFPa Eco with RandomGPA				KaFFPa Eco with RandomMSTM			
# Partitions	∂_G	∂_{max}	$comm_G$	$comm_{max}$	∂_G	∂_{max}	$comm_G$	$comm_{max}$
	$HEXPxLD$							
2	2.5	3.7	2.5	3.7	3.8	5.6	3.8	5.6
4	7.8	6.0	11.3	10.9	9.3	8.1	13.4	13.7
8	0.8	6.5	1.0	10.7	2.8	9.6	4.8	18.4
16	5.3	8.7	7.4	10.7	6.0	10.5	8.6	15.2
32	3.0	3.9	6.9	7.4	2.5	5.2	5.9	10.5
64	1.9	-0.7	5.3	3.0	1.9	1.6	4.4	6.3
overall	3.5	4.6	5.7	7.7	4.4	6.7	6.8	11.5
	$ExpansionTuple$				$HexpHC$			
2	7.0	11.4	7.0	11.4	4.0	5.9	4.0	5.9
4	4.3	7.8	6.1	11.0	9.6	9.4	13.6	14.8
8	-1.6	8.5	-2.2	13.6	2.9	12.9	5.0	21.7
16	1.5	8.9	0.1	9.5	4.5	11.6	6.8	17.2
32	-0.7	5.5	-0.5	7.6	1.3	4.2	3.9	9.7
64	-0.4	1.8	0.3	2.5	1.3	3.1	3.4	7.2
overall	1.7	7.3	1.7	9.2	3.9	7.8	6.0	12.6
	$HEXPxIO$							
2	2.5	3.4	2.5	3.4	4.4	6.4	4.4	6.4
4	7.2	4.8	10.4	9.1	9.5	8.9	13.8	14.9
8	0.8	6.3	1.0	9.5	3.1	9.1	5.3	17.7
16	4.6	8.9	6.5	9.0	5.2	11.4	7.8	17.1
32	2.6	3.2	6.6	6.0	2.6	4.4	6.0	9.7
64	1.9	-0.1	4.9	2.0	1.5	1.3	4.0	3.5
overall	3.3	4.4	5.3	6.5	4.4	6.9	6.9	11.4

Table B.8.: Top three ratings for social networks, calculated by KaFFPa Eco. Boundary nodes and communication volume given as improvements over $Expansion^{*2}$ rating function

Appendix C.

Zusammenfassung

Das Graphpartitionsproblem ist eines der klassischsten und grundlegensten Probleme der theoretischen Informatik. Tatsächlich gibt es eine Vielzahl von praktischen Anwendungen dieses Problems die von VLSI Design [15] bis hin zu dem Auffinden der Funktion eines bestimmten Gens reichen [3]. Obwohl gezeigt werden kann dass das balancierte Graphpartitionsproblem NP-vollständig ist, kann es oft mit zufridenstellender Genauigkeit in sehr kurzer Zeit angenähert werden. Viele modernen Graphpartitionierungsalgorithmen verwenden ein Mehrlevelverfahren, das aus drei Phasen besteht: Zunächst wird eine Hierarchie von Schrittweiße gröberen Graphen erstellt, woraufhin der gröbste Graph initial partitioniert wird und zuletzt wird für jeden feineren Graphen die Partitionierung verbessert. Da die Vergröberungsphase essentiell für ein gutes Ergebnis ist, liegt das Ziel dieser Arbeit darin neue Techniken zu finden und bestehende zu verbessern, die in der Vergröberungsphase eingesetzt werden können. Hierbei liegt das Hauptaugenmerk auf der Konstruktion neuer Kantenbewertungsfunktionen und der Analyse und dem Entwurf eines neuen approximativen Paarungsalgorithmuses. Alle Ergebnisse werden mithilfe des Mehrlevel-Graphpartitionierers KaFFPa [25] ausgewertet, der einen gröberen Graphen konstruiert indem er eine Paarung maximalen Gewichts bestimmt und alle gepaarten Kanten kontrahiert. Dies ergibt ein sehr schnelles Vergröberungsverfahren. Unsere Kantenbewertungsfunktionen führen zusammen mit einem neuen Paarungsalgorithmus zu signifikanten Verbesserungen der Laufzeit der rechenintensivsten Konfigurationen des Partitionierers und zu signifikanten Verbesserungen der Partitionsqualität auf Sozialen Netzen.

Bibliography

[1] Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory Comput. Syst.*, 39(6):929–939, 2006.

[2] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.

[3] Amir Ben-Dor and Zohar Yakhini. Clustering gene expression patterns. In *RECOMB*, pages 33–42, 1999.

[4] Jie Chen and Ilya Safro. Algebraic distance on graphs.

[5] Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato Fonseca F. Werneck. Graph partitioning with natural cuts. In *IPDPS*, pages 1135–1146, 2011.

[6] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[7] Chris Walshaw et al. The graph partitioning archive website. *http://staffweb.cms.gre.ac.uk/ wc06/partition/.*

[8] R. Preis et al. Party partitioning library website. *http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html.*

[9] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, DAC '82, pages 175–181, New York, NY, USA, 1982. ACM.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[11] Bruce Hendrickson and Robert W. Leland. A multi-level algorithm for partitioning graphs. In *SC*, 1995.

[12] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[13] Horst D. Simon I, Horst D. Simon, and Horst D. Simon. Partitioning of unstructured problems for parallel processing, 1991.

[14] Stefan E. Karisch, Franz Rendl, and Jens Clausen. Solving graph bisection problems with semidefinite programming. Technical report, INFORMS Journal on Computing, 1997.

[15] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: application in vlsi domain. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 526–529, New York, NY, USA, 1997. ACM.

[16] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, pages 113–122. CRC Press, 1995.

[17] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 49(1):291–307, 1970.

[18] Michael Ley. The dblp project. *http://www.informatik.uni-trier.de/ ley/db/*.

[19] Holtgrewe M and Sanders P. A scalable coarsening phase for a multi-level graph partitioning algorithm. *Diploma Thesis*, 2009.

[20] Holtgrewe M, Sanders P, and Schulz C. Engineering a scalable high quality graph partitioner. *Parallel and Distributed Processing (IPDPS), 2010 IEEE International Symposium*, pages 1–12, 2010.

[21] Jens Maue and Peter Sanders. Engineering algorithms for approximate weighted matching, 2007.

[22] Kurt Mehlhorn and Peter Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

[23] Vitaly Osipov and Peter Sanders. *n*–level graph partitioning. In *ESA (1)*, pages 278–289, 2010.

[24] F. Pellegrini. Scotch website *http://www.labri.fr/pelegrini/scotch*.

[25] Peter Sanders and Christian Schulz. Engineering multilevel graph partitioning algorithms. In *ESA*, pages 469–480, 2011.

[26] Kirk Schloegel, George Karypis, Vipin Kumar, J. Dongarra, I. Foster, G. Fox, K. Kennedy, A. White, and Morgan Kaufmann. Graph partitioning for high performance scientific simulations, 2000.

[27] Yifan Hu Tim Davis. The university of florida sparse matrix collection website. *http://www.cise.ufl.edu/research/sparse/matrices/*.

[28] The Pennsylvania State University. Citeseer project. *http://citeseerx.ist.psu.edu/*.

[29] Chris Walshaw and Mark Cross. Jostle: parallel multilevel graph-partitioning software - an overview, 2007.

[30] C. T. Zahn. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comput.*, 20:68–86, January 1971.