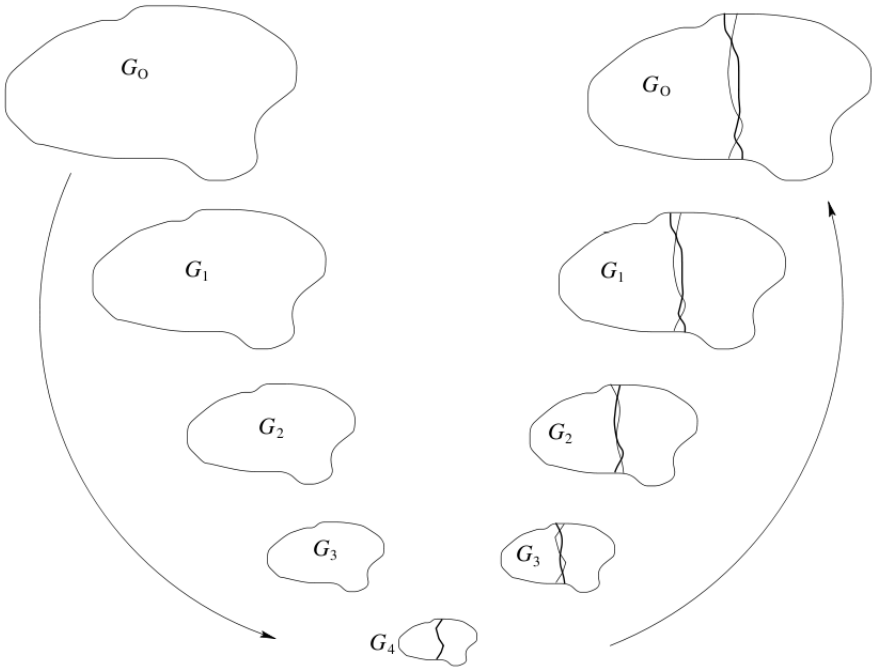
A Multi-Level Framework for Bisection Heuristics

Student thesis at the Institut for Theoretical Computer Science,
Algorithmic I
Prof. Dr. Dorothea Wagner
Faculty of Informatics
Karlsruhe Institute of Technology (KIT)

by

Romuald Brillout
December 2009



Supervisors:
Ignaz Rutter
Prof. Dr. Dorothea Wagner

The Figure on the front page represents the Mutli-Level Technique. The graph G_0 is iteratively reduced in size up to G_4. Then a bisection is computed on G_4 which in the Figure is represented by a dark line. This bisecion is then iteratively projected back to G_0 over G_3, G_2, and G_1. In each projection the bisection is refined. In the Figure these refined bisections are represented by the gray lines.

Erklärung

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

Karlsruhe, den 08. Dezember 2009

Contents

1 Introduction

Dividing a set of connected objects into parts of roughly equal size such that objects in different parts are weakly connected is a common task in compute science. For example in parallel computing one wants to distribute tasks to different processors in such a way that the communication between tasks computed in different processors is minimized. The input to such a problem can be seen as a graph where the vertices represent the tasks and the edges represent the communication of the tasks. The distribution to k processors is then represented as a division of the vertices into k parts of equal size such that the number of edges crossing different parts is minimized. The minimization of the crossing edges then represents the minimization of the communication between tasks computed in different processors. The equal size of the parts constraint then represents the equal distribution of tasks to the processors. The division of the vertices into k parts of equal size is called a *k-partition* and the corresponding problem is called *k-Partitioning*. One approach to compute a 2^i-partition is to first compute a 2-partition and then on the given two partitions to compute again 2-partitions. This is done several times until the graph is divided into 2^i parts. A 2-partition is also called a *bisection*.

A method to compute bisections is the *Multi-Level Technique*. The key idea is to reduce the magnitude of the graph by collapsing vertices together in order to reduce the solution space in such a way that only good solutions remains. Furthermore, the reduction of the solution space allows for the use of more expensive heuristics. The Multi-Level technique has shown good results and can be considered state of the art. The Multi-Level technique has several degrees of freedom like how you collapse the vertices or how you compute the bisection on the graph with reduced magnitude. In this work we analyse how these degrees of freedom affect the quality of the results. We do this by providing new so called "expensive methods" for every degree of freedom. The aim of the expensive methods are to better the output bisection regardless to the runtime in the range of feasibility. That way the potential of a degree of freedom may be revealed if an expensive method reaches to better the output bisection.

In Section 2 we give the definition of the MinBisection problem – which is the problem of computing the bisections mentioned above – and the related basic definitions. In Section 3 we give an overview of the previous work of MinBisection including the Multi-Level Technique. In Section 4 we present a Framework computing bisections based on the Multi-Level Technique. Finally in Section 5 we present the expensive methods for every degree of freedom and discuss their experimental results in Section 6.

2 Definitions

Throughout the text $G = (V, E, \omega)$ denotes an undirected, weighted, simple graph, where V denotes the set of vertices, E the set of edges, and ω a weight function:

$$\omega : V \cup E \to \mathbb{R}^+$$

Moreover we denote $n = |V|$ and $m = |E|$. The *degree* of a vertex v is the number of its neighbours:

$$\deg(v) = |\{\{u, v\} \in E\}|$$

The *weighted degree* of a vertex v is the sum of the weights of the adjacent edges:

$$\deg\_\text{weighted}(v) = \sum_{\{u,v\} \in E} \omega(\{u, v\})$$

Cut A *cut* is a partition of V into two subsets $C_1 \subset V$ and $C_2 \subset V$ such that $C_1 \cap C_1 = \oslash$ and $C_1 \cup C_2 = V$. Thus the cut is represented by the pair $\{C_1, C_2\}$. We denote by $E(C_1, C_2)$ the set of edges *crossing* the cut:

$$E(C_1, C_2) = \{\{u, v\} \in E | u \in C_1 \wedge v \in C_2\}$$

The *cut weight* is the sum of the weights of the edges "crossing" the cut:

$$\omega(\{C_1, C_2\}) = \sum_{\{u,v\} \in E(C_1,C_2)} \omega(\{u, v\})$$

The Bisection problem A *bisection* in G is a cut where the two partitions have same size: $|C_1| = \lceil n/2 \rceil$ and $|C_2| = \lfloor n/2 \rfloor$. A bisection is called minimum/maximum if the sum of the weights of the bisection edges is minimum/maximum. MinBisection is the problem of finding such a minimum bisection.

Vertex gain Given a cut the *gain* of a vertex is equal to the number of adjacent edges crossing the cut minus the number of adjacent edges not crossing the cut. That means that moving the vertex to the other partition leads to a decrease of the cut weight by the gain of the moved vertex. Note that the gain of a vertex can be negative and consequently the cut weight increases when such a vertex moves. Thus the vertex with the highest gain is the vertex that when moved leads to the highest decrease / least increase

of cut weight. The gain is the base of many greedy heuristics. Formally the gain of a vertex $v \in C_1$ is defined as follows:

$$\text{gain}(v) = \sum_{\{u,v\} \in E \wedge u \in C_2} \omega(u) - \sum_{\{u,v\} \in E \wedge u \in C_1} \omega(u)$$

The gain of a pair of vertices u and v is formally defined as follows:

$$\text{gain}(\{u, v\}) = \text{gain}(u) + \text{gain}(v) - 2 \cdot \omega(\{u, v\})$$

where the weight of u and v equals to zero when u and v are not connected by an edge. The gain can be generalized to subsets $U_1 \subset C_1$ and $U_2 \subset C_2$ of the graphs as follows:

$$\text{gain}(U_1, U_2) = \sum_{v \in U_1 \cup U_2} \text{gain}(v) - 2 \cdot \sum_{\{u,v\} \in E \wedge u \in U_1 \wedge v \in U_2} \omega(\{u, v\})$$

which is equal to the change of the cut weight when U_1 and U_2 are swapped.

3 Previous work

As stated in the introduction the computation of bisections is a common task. Thus a lot of research has been done leading to many findings in theory and practice. In this section we give an overview of what is currently known about the complexity of MinBisection and common heuristics to solve them.

3.1 Complexity

MinBisection is NP-hard and there exists a polynomial approximation algorithm that approximates the cut weight by a factor of $O(log^2 n)$ [26]. The corresponding problem MaxBisection is also NP-hard and there exists a randomized polynomial approximation algorithm that approximates the cut weight by a factor of .699 [27]. On planar graphs MaxBisection is NP-hard and has a PTAS [19]. MinBisection's NP-hardness remains an open question [20] and there exists a polynomial approximation algorithm that approximates the cut weight by a factor of $O(logn)$ [26] on planar graphs. Since MinBisection is NP-hard an exact solution is often out of question. Instead heuristics are usually used. The next section gives an overview of the most popular approaches and heuristics for MinBisection.

3.2 Heuristics

Many ways to compute a bisection have been developed. In this section we discuss popular approaches; the refinement approach based on improving an existing bisection, the greedy growing approach based on growing regions of the graph, the spectral approach based on spectral analysis and geometric approaches based on geometric information on the graph.

3.2.1 Refinement Heuristics

Refinement heuristics are heuristics that takes as input a bisection and output a bisection. KL defined by Kernighan and Lin [23] is an old and standard refinement heuristic. It starts with an initial bisection and is based on swapping vertices. At the beginning all vertices are unlocked. A pair of vertices u and v is chosen so that:

- u and v are unlocked
- u and v are in different partitions
- gain($\{u, v\}$) is maximum among all pairs satisfying the previous two conditions

Then the vertices of the chosen pair are swapped, that is moved to the other partition, and locked. The whole process is repeated until all vertices are locked. The running time of this heuristic is $O(|V|^3)$ [14].

Several improvements of the KL heuristic have been developed. Fiduccia and Mattheyes [15] (FM) presented a variant with time complexity $O(|\mathrm{E}|)$. The difference to KL is that in each step the FM heuristic moves a single vertex whereas KL selects a pair of vertices and swaps them. The single vertex is alternatingly chosen from both partitions to maintain balance. The $O(|\mathrm{E}|)$ complexity is achieved by sorting the vertex by gain with bucket sort.

Since the running time of the FM variant is linear, it is usually run a couple of times. There are two ways of doing this. First by running FM iteratively, that is running FM on the output bisection of a previous FM run. This is done until no new bisection is generated. Second by giving different input bisections to FM, which is very useful since FM is strongly sensitive to its initial seed.

Karypis and Kumar [22] argue that due to the nature of refinement heuristics, most of the swapped vertices occur along the boundary of the cut. Thus the consideration of vertices that are far from the cut is a waste of computation. Based on this statement they present a new heuristic, Boundary KL (BKL). BKL is equal to FM except that only vertices at the boundary are taken in consideration for moving. This considerably reduces the running time. Furthermore, instead of letting BKL terminate exactly when all vertices have been moved, BKL also terminates when after c moves the cut did not decrease, where c is a constant. The consequent decrease of the runtime was reported to be significant.

A way to obtain more explorative behaviour is to bind the "uphill moves" – moves that increase the edge cut – with an arbitrary value called "temperature". This value influences the probability that uphill moves occurs; the higher the temperature the higher is the probability that an uphill move occurs. These heuristics are called Simulated Annealing heuristics. Another approach is to change the locking behavior of the vertices. This means that instead of locking a vertex forever, the vertex is locked only for a period of time. The results of this heuristic are shown to be better than their implementation of the FM heuristic.

3.2.2 Graph Growing Partitioning Algorithms (GGP)

Graph Growing Partitioning Algorithms consists in growing one part of the partition from a seed – namely a vertex or a subgraph – until half of the vertices are reached. Sometimes, instead of one seed, two seeds – representing the halves of the Cut – are provided and then grown. A simple

implementation of GGP starts from a vertex and grows a region around it in a breadth-first fashion until half of the vertices are included [12, 17, 16]. Replacing the breadth-first fashion growing with a greedy fashion growing like KL, namely by adding first vertices that decrease most / increase least the edge cut, leads to better results [22]. This heuristic is called Greedy Graph Growing Partitioning (GGGP).

Like KL the output bisection of growing heuristics is strongly affected by the input seed. Thus, like KL, the growing heuristics are usually run several times with different input seeds. Usually, in order to improve the resulting bisection, a refinement heuristic is applied to the output.

3.2.3 Spectral Heuristics

Spectral heuristics make use of spectral analysis in order to find highly connected parts. First, the *Laplacian matrix* of G is computed. The entries of the Laplacian matrix are defined as follows:

$$\ell_{ij} = \begin{cases} -1 & \text{if } i \neq j \text{ and } e_{i,j} \in E \\ 0 & \text{if } i \neq j \text{ and } e_{i,j} \notin E \\ \deg(v_i) & \text{if } i = j \end{cases}$$

Second, the second-smallest eigenvalue λ_2 and its corresponding eigenvector are computed. This eigenvalue is called the *algebraic connectivity* and its corresponding eigenvector is called the *Fiedler Vector*. The Fiedler Vector divides the vertices in a "sensible" way. Why it does this is not at all obvious and will not be presented in this text. Spectral heuristics have shown good results but are expensive and in large graphs spectral analysis is not feasible. Instead the magnitude of the graph is first reduced to finally compute the spectral analysis. The reduction of the graph size is usually done by using the "Multi-Level Technique" presented in the next Section.

3.2.4 Geometric Heuristics

Geometric heuristics make use – when available – of geometric information of the graph, like e.g., vertex coordinates. They tend to be fast and find good partitions even if not as good as spectral heuristics. If no coordinates are available it may be possible to compute suitable coordinates in a preprocessing step. However, in many problem areas no coordinates are given and the computation of coordinates, e.g., using the algorithm of Chan, Gilbert, and Teng [10], is much more expensive than the geometric heuristic itself. Thus on graph without coordinates geometric heuristics are too expensive.

3.2.5 Other

Dang, Ma, and Liang [11] recently published a heuristic based on approximating the solution of a linearly constrained continuous optimization problem that is equivalent to bisection. The authors claim better results than the heuristic presented by Karypis and Kumar [22], which is a heuristic used by METIS which is a Software Package. However, the claim is based on experiments with random graphs only. Especially, no graphs from applications or previous benchmark graphs are used. Moreover, they only compare to a re-implementation of the METIS heuristic without specifying how it was implemented.

3.3 The Multi-Level Technique

In one phrase the *Multi-Level technique* is the idea of reducing the magnitude of a graph by merging vertices together, compute a bisection on this reduced graph, and finally project this bisection on the original graph. This section presents the Multi-Level Technique, which consists of three phases.

In the first phase – called *coarsing* – the magnitude of the graph is reduced by merging vertices. The merging of vertices is done iteratively: of a graph a new coarser graph is created and of this new coarser graph an even more coarse graph is created. This is done until a certain small magnitude is reached. Thus graphs with different magnitudes are induced.

In the second phase – called *intial bisection phase* – a bisection of the graph with the smallest magnitude – the coarsest graph – is computed.

In the third and last phase – called *uncoarsing* – the computed bisection is iteratively projected back to the original graph. In each iteration a refinement heuristic is applied. The merging of vertices induces a map between vertices of a graph and vertices of its coarser graph which is used for the back projection. A rebalancing to insure the size of the bisection may be needed since vertices not belonging to the same partition may be merged together.

The Multi-Level Technique has shown to significantly improve the results, both in terms of quality and running time. Especially when used on heuristics considering the graph only locally, as the Multi-Level Technique constitutes a more global view on the graph. Figure 1 gives an overview of the Multi-Level Technique.

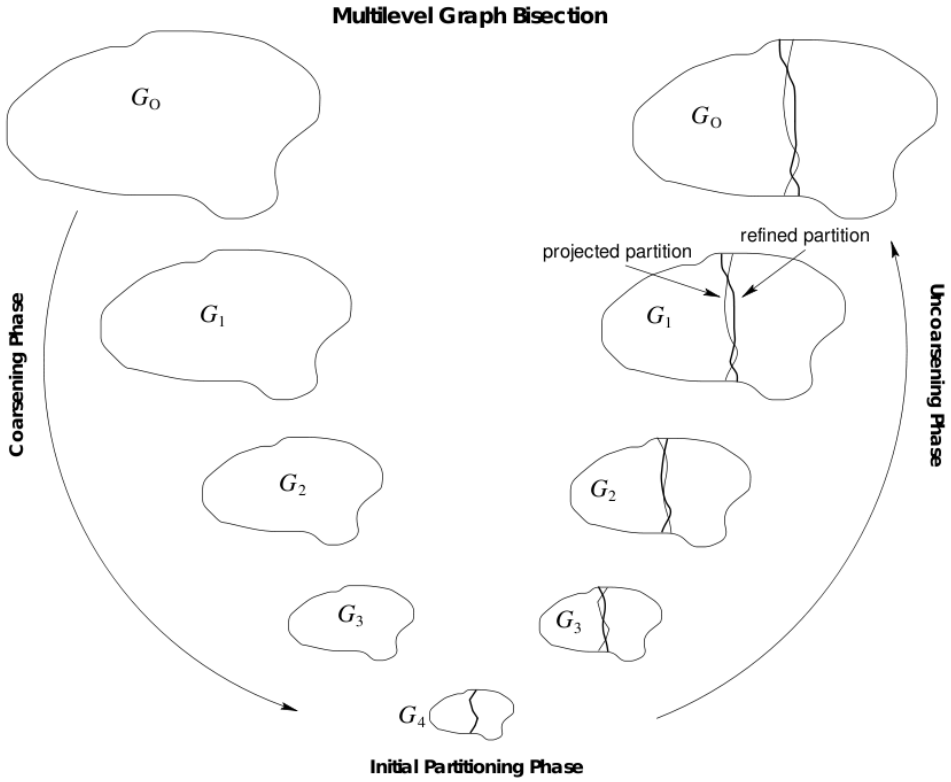


Figure 1: Overview of the Multi-Level Technique

3.4 Software Packages

As one can expect, since partitioning is quite important, a lot of Software Packages for partitioning are available. A list of softwares producing partitions can be found on the official website of the Software Package JOSTLE [1]. The best known ones are presented in this section; Chaco, PARTY, and METIS. All three make use of the Multi-Level Technique.

3.4.1 Chaco

Chaco [8] implements several approaches. One of them uses the Multi-Level Technique which is the authors' method of choice for large graphs.

Their Multi-Level implementation [7] uses a random maximal matching for coarsing: vertices are visited in random order, and if not already matched, the visited vertex is matched with a random unmatched neighbour. The initial partition of the coarsest graph is computed by a spectral partitioner. For the refinement in the uncoarsing phase the FM variant of KL is used.

3.4.2 PARTY

Like Chaco, the PARTY [24] partitioning library provides a variety of methods. The methods are separated into two parts: the global methods and the local methods. As the names already indicate, the global methods consider the graph as a whole while the local methods consider local properties like neighborhoods. Once a global and a local method is chosen, the global method is run and the given partition is refined by the local method.

A specific combination [9] has shown to give good results: a linear time 1/2-Approximation algorithm for Maximum Weighted Matching as global method and a heuristic called Helpful-Sets as local method. In fact this combination uses the Multi-Level technique: the global method is used as coarsing, the local method is used as refiner and the initial partition is computed with spectral methods. The Helpful-Sets heuristic is similar to the KL heuristic expect that it considers not only single vertices, but also whole sets of vertices.

3.4.3 METIS

METIS [22] is the leading Software Package for solving partitioning, amongst others using recursive bisection. It contains leading bisection heuristics. The recursive bisection heuristic of METIS is based on the Multi-Level technique.

In METIS, coarsing is done by the Heavy Edge Matching (HEM) heuristic. HEM finds a maximal matching by visiting randomly an unmatched

vertex and matches it to the neighbour that is connected by the heaviest edge and unmatched. If there is more than one such neighbour, the neighbour to be matched is randomly chosen. The second phase is done by the previously presented heuristic GGGP. The resulting bisection is than refined by the previously presented heuristic BKL. The refinement in the uncoarsing phase is also done by BKL. On graphs with a small magnitude BKL is iteratively applied until no new bisection is generated, whereas on graphs with a large magnitude BKL is run only once. The motivation comes from the fact that the input bisection is already a good bisection so that on large graphs it is not worth to run BKL more than once.

4 The Framework

As one can observe, the Multi-Level Technique is quite modular. For example, METIS uses the HEM algorithm for the coarsing step, the GGGP algorithm for the initial bisection, and the BKL algorithm for the refinement in the uncoarsing step. This work is based on the observation that these three algorithms can be seen as variable and that they can be changed independently to the rest of the overall METIS algorithm. Thus METIS defines a scheme where every of the three Multi-Level steps can be varied. More generally speaking, the Multi-Level Technique can be seen as frame or skeleton that describes how different strategies for Bisection can be combined. The skeleton provides a plan and leaves several degrees of freedom that have a fixed meaning but can vary. This observation leads us to implement a Multi-Level framework in order to be able to easily combine different strategies for an easy development of better heuristics. In fact, the skeleton defined is always the same for each implementation of the Multi-Level Technique. The aim of this work is to write this skeleton freeing one to write its own skeleton. This enables us to easily try several different strategies and their combinations. This section gives the mentioned strategies and then describe in detail how they are used by the framework in order to solve MinBisection.

4.1 Degrees of freedom of the Multi-Level Technique

We first give an overview of the degrees of freedom of the Multi-Level Technique in the following table. Then we discuss how they interact together in the Multi-Level context.

Strategy name	Input	Output
The Merger	graph G=(V,E)	map from V to V
The strived magnitude	none	constant c
The Initial Bisector	graph G	bisection B
The Bisector	cut C	bisection B
The Refiner	bisection B	bisection B

The *merger* is used to generate coarser graphs. A coarser graph is given by a map from V to a subset of V as follows: all vertices mapped to the same vertex are merged into one vertex, namely the vertex where all vertices are mapped to. This vertex can in turn be merged with other vertices if it is not mapped to itself.

The *initial bisector* is used to compute the first initial bisection on the coarsest graph.

The *bisector* is used in the uncoarsing phase. When the bisection is projected back from a coarse graph to a finer graph the resulting cut is not necessarily a bisection. Thus the bisector is used to balance the resulted cut to a bisection.

The *refiner* is used also in the uncoarsing phase. It is used to improve every bisection given by the bisector.

The *strived magnitude* is – as the name already suggests – the desired magnitude of the coarsest graph. That is, when the coarsing phase creates a graph with a number of vertices equal to or less than the strived magnitude the coarsing phase stops and no new coarser graph is created.

The framework defines suitable interfaces for all strategies and provides a skeleton that can combine provided strategies to an algorithm. The technical details of this step is described in the next section.

4.2 The Framework

Like the strategies intuitively suggest, the framework is based on a class for each strategy; the classes "Merger", "Refiner", "Bisector", and "Bisection". Where the Initial Bisector is called Bisection for convenience reasons. In fact, these classes are interfaces. Concrete strategies can then be implemented by implementing these interfaces. The interfaces ensures the availability of the expected functions described in the previous section.

Graphs and cuts are represented by classes; the "Graph" and "Cut" classes. The Graph class is based on the Boost Graph Library [2]. The aim is to provide a graph class providing a simple interface where the interface of the Boost Graph Library is quite heavy. The class Cut is based on the class Graph and represents the cut as a map mapping every vertex to a boolean value. The class Cut provides basic functionality like a function to get the weight of the cut or a function to check whether the cut is a bisection or not as well as the computation of the gain for a given vertex, which is needed for the refinement functions.

The only thing left is to orchestrate the functionality of the strategy classes in order to provide a bisection from a graph, based on the Multi-Level Technique. That is provided by the Multi-Level skeleton. The skeleton is implemented in the class "Solver". The Solver constructor takes the different strategies as arguments, that is an object for each strategy class and the strived magnitude. The Solver provides a function that computes a bisection from a graph using the strategy objects. What this function exactly does is now presented.

4.2.1 Coarsing Phase

The first phase consists in creating iteratively coarser graphs until the strived magnitude is reached. That is from the graph G a new coarser graph G_1 is created and from the graph G_1 a new coarser graph G_2 is created and so on. Thus a sequence of graphs $G = G_0, G_1, ..., G_k$ is created where G_k is the only graph whose magnitude is lower than the strived magnitude. A new graph is created based on a run of the provided merger strategy with the fine graph – from which the coarser graph will be created – as input: all vertices that are mapped to the same vertex in the returned map are merged together. The fine graph, the coarse graph, the map from vertices of the fine graph to the coarse graph, and the map of the vertices of the coarse graph to the fine graph are stored in a *coarser* object.

At the end of this phase we have h coarser objects where h is the number of coarsing levels, i.e., the number of coarse graphs. Thus with the original graph we have $h+1$ graphs. The value of h is determined by the merger and the strived magnitude; the more vertices the merger bundles together the faster the strived magnitude is reached and the fewer coarse graphs will be created.

4.2.2 Intial Bisection Phase

The second phase consists in creating the initial bisection on the coarsest graph. This is done by running the initial bisector with the coarsest graph as input.

4.2.3 Uncoarsing Phase

The third phase consists in projecting the initial bisection of the coarsest graph to the original graph. This is done by using the maps stored in the coarser objects created in the first phase. Since the back projecting of a bisection on a finer graph is not necessarily a bisection it needs to be balanced. This is done by running the provided bisector. Then the provided refiner is run.

5 Our Multi-Level strategies

As stated before our framework gives us the possibility to easily try out new ideas. The following strategies arise from our focus to increase the quality of the result no matter the runtime. For the coarsing phase we propose alternate matchings heuristics and the ORCA clustering algorithm. For the initial bisection phase we implement the computation of an optimal solution by an ILP representation of the MinBisection problem using the Gurobi solver. In the refinement phase we suggest a variant of the BKL algorithm.

5.1 Coarsing

In order to have a better understanding of the Multi-Level technique we state what we consider the aims of coarsing. There are two aims of coarsing and thus of the Multi-Level Technique:

1. Reduce size.
2. Merge parts that will probably not be cut in bisection.

By the reduction of the graph size expensive heuristics can be run. In order to merge parts that will probably not be cut in bisection we merge vertices in such a manner that

- The number of edges between these merged vertices is high.
- The variance of the weight of a cut with a certain size in these merged vertices is low.

These two conditions imply that a cut of the merged vertices is heavy-weight. Thus a bisection rather won't cut these merged vertices. That way, irrelevant possibilities are taken apart. Overall, it is a way of computing bisections considering the graph globally without being too expensive.

Thus a good clustering algorithm used as merger should lead to a good coarsing phase since the aim of coarsing is to find dense regions. Thus we feel that using a good clustering algorithm may provide good results. Because of good results both in terms of speed and quality we decided to use the ORCA algorithm [25].

5.1.1 ORCA

The idea of ORCA is to work in a local way where most other clustering algorithms work in a global way. This is promising since the distance of two

vertices belonging to the same cluster is most likely small and local algorithms tend to be faster than global ones.

The algorithm consists of the following steps. At the beginning all vertices having degree one or less are removed, these vertices are later assigned to the cluster of their neighbour or stay as singletons if they have degree 0. Then, dense regions are found using a local search approach. The process of finding dense regions is repeated until it is not possible to find dense regions anymore. The detected dense regions are then contracted to what the authors calls a "super vertex". Out of these super vertices a new Graph is created. The whole process is repeated on this new Graph. Thus the whole process is iteratively repeated several times.

According to the authors [13] ORCA is the current best clustering algorithm in terms of scalability. In terms of quality it competes with other state-of-the-art algorithms, and between them, no general assertion which one to prefer can be made. However, the algorithm presented by Blondel, et al. [6] is faster.

Another coarsing strategy is to use a matching in order to merge pairs of vertices together. This approach is widely used because it is fast to implement and has shown to lead to good results in the Multi-Level context.

5.1.2 Matching based coarsing

Most of the time a matching is found greedily based on a rating of the edges. This technique is composed of two components:

1. The edge rating.

   Based on local criteria every edge is given a certain value; its *rating*. This value defines which edges should preferably be matched first or more precisely, its two endpoints should preferably be matched first. The edge rating induces an order on the edges. Given this order it is still not defined how to match these edges. In fact, matching in a greedy fashion using this order leads to suboptimal matchings and does not take care of global criteria like the balance in terms of vertex weight of the induced graph. This leads us to the next point.

2. A strategy that based on edge ratings decides which vertices should be matched.

   Intuitively, one could try to optimize the matching and for example compute a maximum weighted matching, where the weights are given by the edge rating. An approximation, GPA, running in $O(|E| \cdot \log |E|)$

time has been implemented by Holtgrewe [18]. It shows as expected some increase in terms of quality. However, this approach does not consider global criteria like balance. A widely used heuristic is to visit the vertices and match an unmatched neighbour connected by the edge with the highest edge rating. These heuristics have shown good results because the induced coarse graphs are more likely to be balanced. How the vertices are visited is based on a vertex rating or they are visited randomly. Like the edge rating, the vertex rating is based on local criteria like the degree of the vertex.

As stated before the ratings are based on certain criteria, which we present next.

- For Edges:
  - weight of the edge
  - weight of the adjacent vertices
  - degree of the adjacent vertices
  - weighted degree of the adjacent vertices
  - number of common neighbours of the adjacent vertices
- For Vertices:
  - degree of the vertex
  - weight of the vertex

We now present the edge ratings that we evaluated. If not specified otherwise the matching is computed in a greedy fashion as follows:

1. the edge rating defines an edge order in descending order. If more than one edge rating is provided the next edge rating is used to break ties of the previous edge rating. This also applies for the vertex rating expect that the order is ascending.

2. if no vertex rating is provided the edges are passed in the order given by the edge rating. If a visited edge connects two unmatched vertices, they are matched.

3. if a vertex rating is provided the vertices are passed in the order given by the vertex rating. For every visited vertex its neighbours are visited in the order given by the edge rating. If both, visited vertex and visited neighbour, are unmatched they are matched.

We now present a variety of edge ratings we find interesting. Some of them, e.g., the one behind METIS, are well known standard ratings. We also propose new ratings.

A simple quality index to know if a merger is good is the sum of the weights of the edges of the resulting coarsest graph. In fact the lower this sum the more a edge-cut tend to be light-weight. Thus, intuitively one would want to match first the edges with the highest weight. However, this leads to unbalanced clusters in terms of size. Thus instead of the weight, the weight density is used.

- **density**:

$$uv \mapsto \text{density}(uv)$$

  where

$$\text{density}(uv) = \frac{\omega(uv)}{\omega(u) \cdot \omega(v)}$$

The idea of using the edge density has also been experimented by Holtgrewe [18] but using slightly different edge ratings. The following two ratings are based on those from Holtgrewe.

- **density_2**:

$$uv \mapsto \frac{\omega(uv)}{\omega(u) + \omega(v)}$$

- **density_3**:

$$uv \mapsto \frac{\omega(uv)}{(\omega(u) + \omega(v))^2}$$

The following edge rating was found by Holtgrewe [18]. The author experimentally shows that this edge rating provides good results.

- **inner_out**:

$$uv \mapsto \frac{\omega(uv)}{\sum_{ux \in E} \omega(ux) + \sum_{vx \in E} \omega(vx) - 2 \cdot \omega(uv)}$$

Instead of considering the weight of the edges we think that it is more intuitive to use the weight density discussed before. In fact lets consider the following example: two edges $\{u, v\}$ and $\{w, z\}$ with same weight. If $\omega(u) + \omega(v) < \omega(w) + \omega(z)$ then it would be more intuitive to match $\{u, v\}$ rather than $\{w, z\}$. In fact $\{u, v\}$ represent a more dense region which is correspondingly more likely to not be crossed by a bisection. Thus we prefer $\{u, v\}$ over $\{w, z\}$.

- **inner_out_density**:

$$uv \mapsto \frac{\text{density}(uv)}{\sum_{ux \in E} \text{density}(ux) + \sum_{vx \in E} \text{density}(vx) - 2 \cdot \text{density}(uv)}$$

Alternativelly one could only consider the weighted degree of one of the two adjacent vertices. In fact if a vertex is strongly connected to one of its neighbours and weakly connected to the other neighbours, it should be matched with the strongly connected neighbour. This is what is aimed with the following rating.

- **out_side_density**:

$$uv \mapsto \frac{\text{density}(uv)}{1 + \min(\text{out}(u), \text{out}(v))}$$

where

$$\text{out}(v) = \sum_{vx \in E} \text{density}(vx) - \text{density}(uv)$$

The following rating is the rating used in METIS. The authors states that the sorting of the vertices by degree tends to lead larger matchings [21]. This may be due to the fact that matching a vertex inhibit all adjacent edges to be the source of a match. Thus the higher the degree the higher the amount of edges that become unusable for further matchings.

- **shem_metis**:

$$v \mapsto \min(\deg(v), 0.7 \cdot \text{average\_degree})$$

$$uv \mapsto \omega(uv)$$

where

$$\text{average\_degree} = 2 \cdot \sum_{uv \in E} \omega(uv)$$

We also evaluate a known simpler variant:

- **shem**:

$$v \mapsto \deg(v)$$

$$uv \mapsto \omega(uv)$$

An idea would be to consider the vertex rating in descending order in order to match first vertices that are more likely to be in a cluster since in clusters the average degree is high. However, always matching the vertex with the highest degree leads to a new vertex with an even higher degree. The degree is then so high that matching this high degree vertex leads to inhibit a lot of edges to be the source of a merging. This in turn results to matchings with only a few matched pairs of vertices. Thus we only consider the ascending order.

The idea behind the following rating is that the density should always be taken in consideration rather than the weight. In fact as already stated before we want to connect two components that are highly connected but highly connected relative to the size of the components.

- **shem_density**

$$v \mapsto \min(\deg(v), 0.7 \cdot \text{average\_degree})$$

$$uv \mapsto \frac{\omega(uv)}{\omega(u) \cdot \omega(v)}$$

where

$$\text{average\_degree} = 2 \cdot \sum_{uv \in E} \omega(uv)$$

The idea of following rating is to combine the shem rating with the out ratings.

- **shem_out**

$$v \mapsto \min(\deg(v), 0.7 \cdot \text{average\_degree})$$

$$uv \mapsto \frac{\omega(uv)}{\sum_{ux \in E} \omega(ux) + \sum_{vx \in E} \omega(vx) - 2 \cdot \omega(uv)}$$

where

$$\text{average\_degree} = 2 \cdot \sum_{uv \in E} \omega(uv)$$

The idea behind following rating is to consider the degree in a weighted manner in order to be more precise; instead of summing the number of adjacent edges we sum the weights of the adjacent edges.

- **shem_weight**

$$v \mapsto \sum_{vx \in E} \omega(vx)$$

$$uv \mapsto \omega(uv)$$

The experimental resutls of the edge ratings and ORCA are given in the next section. We also implemented a maximum match merger based on Edmonds's algorithm. However, since we used the BGL implementation of Edmond's algorithm the algorithm consider the edges unweighted. Thus we did not evaluated this merger.

5.2 Refinement

For the refinement phase we implemented the known strategies BKL, KL and FM. We also propose and implemented a new heuristic, which we call *deep BKL*. The idea behind deep BKL is to expand BKL by consider more than the two vertices with the best gain for the swapping step. A simple way to implement this idea is to consider a certain amount of veritces that have the best gains. Deep BKL is defined as follows. Its input is a bisection. At the beginning all vertices are unlocked. It computes and maintains two priority queues for each partition of the bisection representing all unlocked vertices adjacent to the bisection ordered by their gain. Of each priority queue it takes the first 20 vertices, hence the 20 vertices with the highest gain. Out of the resulting 40 vertices it computes the pair of vertices with the highest gain. This pair is then swapped and locked. The whole process is repetitively run until there is no more unlocked boundary vertices. Finally deep BKL outputs the resulting bisection.

5.3 Initial Bisection

We provide two possibilities to compute an initial bisection. First by running the pmetis programm provided by METIS. Second by running an *ILP* that computes an optimal initial bisection using the Gurobi Optimizer [3]. Gurobi solves integer linear optimization problems (ILP). Our ILP representation of MinBisection is based on the representation of a crossing edge ij by a variable $e_{i,j}$. This variable is equal to 1 when the edge that it represents is crossing the cut, otherwise it is equal to 0. The belonging of each vertex i to a partition is represented by the variable x_i which is a binary variable. Our ILP representation of MinBisection is formally defined as follows.

Minimize

$$\sum_{i,j \in \{0..n\}} e_{i,j} \cdot \omega(i,j)$$

Subject to

$$e_{i,j} - x_i + x_j \geq 0 \qquad \forall i,j \in \{0..n\}$$

$$e_{i,j} + x_i - x_j \geq 0 \qquad \forall i, j \in \{0..n\}$$

$$\sum_{i \in \{0..n\}} \omega(i) \cdot x_i \leq c_{\text{upper}}$$

$$\sum_{i \in \{0..n\}} \omega(i) \cdot x_i \geq c_{\text{lower}}$$

where all variables are binary integers taking 0 or 1 as value and

$$c_{\text{upper}} = \lceil (\sum_{v \in V} \omega(v))/2 \rceil + \text{max\_weight}$$

$$c_{\text{lower}} = \lfloor (\sum_{v \in V} \omega(v))/2 \rfloor - \text{max\_weight}$$

where max_weight is the maximal vertex weight.

6 Experimental Evaluation

In this section we present the experimental evaluation of our strategies. We first present the setup of our experiments, then we present which experiments has been run and their results, and finally we give an interpretation of the results.

6.1 Setup

A main application of todays parallel computers is the simulation of technical and physical systems (i.e. structural mechanics, fluid dynamics, etc.). Typically, these systems are modeled by partial differential equations (PDE) which are solved with the finite element method (FEM) or finite difference methods [4]. One major step of parallelizing such problems is the partitioning of the solution domain or discretization graph and the mapping of the partitions to the processors of the parallel machine. We evaluated our heuristics on graphs arising from such parallelization tasks. The official website of JOSTLE [5] lists a lot of such graphs. We choose small graphs in order to be able to compute bisections using expensive heuristics. The chosen graphs sizes roughly range from 5 000 to 40 000 vertices, from 10 000 to 150 000 edges, and from 1.5 to 4 average degree. They are summarized in Table 1.

Since some heuristics are random the results are random too. In order to have stable results the heuristics are run 10 times and then repetitively run until the average of the given results are considered stable. We consider a number of runs stable when the average result of the first half of the runs is equal to the average result of the second half, given a certain tolerance. We set this tolerance to 1%.

Furthermore, we use the pmetis routine of METIS. If not mentioned otherwise we use a greedy bisector based on the vertices' gain in order to balance a cut to a bisection. That is the vertices in the bigger partition with the highest gain are moved to the smaller partition. If not mentioned otherwise we use a strived magnitude of 100 vertices.

In the next section we present the experiment runs made in order to evaluate the potential in terms of quality for each degree of freedom.

6.2 Experiments

The degree of freedom of the Multi-Level Technique suggests the interrogation of how many potential increase in quality lie in each degree of freedom. Therefore, we implemented heuristics that intuitively should increase the quality of results without considering the possible increase of runtime. We

graph	n	m
3elt	4 720	13 722
3elt_dual	9 000	13 278
whitaker3	9 800	28 989
crack	10 240	30 380
4elt	15 606	45 878
whitaker3_dual	19 190	28 581
crack_dual	20 141	30 043
4elt_dual	30 269	44 929
shock.9	36 476	71 290
pwt	36 519	144 794

Table 1: Graphs taken from the JOSLTE website [5] used for experiments.

call these heuristics the *expensive* heuristics. In order to compare results we set our *base* heuristic solving MinBisection to the heuristic used in METIS. That is the SHEM heuristic as merger for the uncoarsing phase, the pmetis routine for the initial bisection, the BKL heuristic as refiner for the uncoarsing phase, and 100 as strived magnitude. As the name suggests, the base heuristic is used as reference and every component of it representing a degree of freedom will be individually switched to an expensive heuristic. Thus we see whether the expensive heuristics induce any improvement and in case they do how big it is.

In the coarsing phase we consider the previously described heuristic ORCA and the rating based heuristics. Table 2 shows the results of the base heuristic with alternate ratings. Figure 3 shows the result of following experiment: we run exactly one coarsing step with either the ORCA heuristic or our implementation of the SHEM heuristic. Then the pmetis routine of METIS is run. The resulting one uncoarsing step left to do is done with our implementation of the BKL heuristic. We run only one coarsing step with ORCA because runing more than one step with ORCA has shown to lead to worse results. That way we evaluate the maximal potential given by ORCA hence the potential of this degree of freedom.

In the initial bisection phase we consider the previously described ILP, which finds an optimal bisection. Figure 2 shows the result of the base heuristic by running the ILP heuristic instead of the pmetis routine for the initial bisection.

In the uncoarsing phase we consider the heuristic Deep BKL. Figure 2 shows the results of the base heuristic by running the Deep BKL heuristic

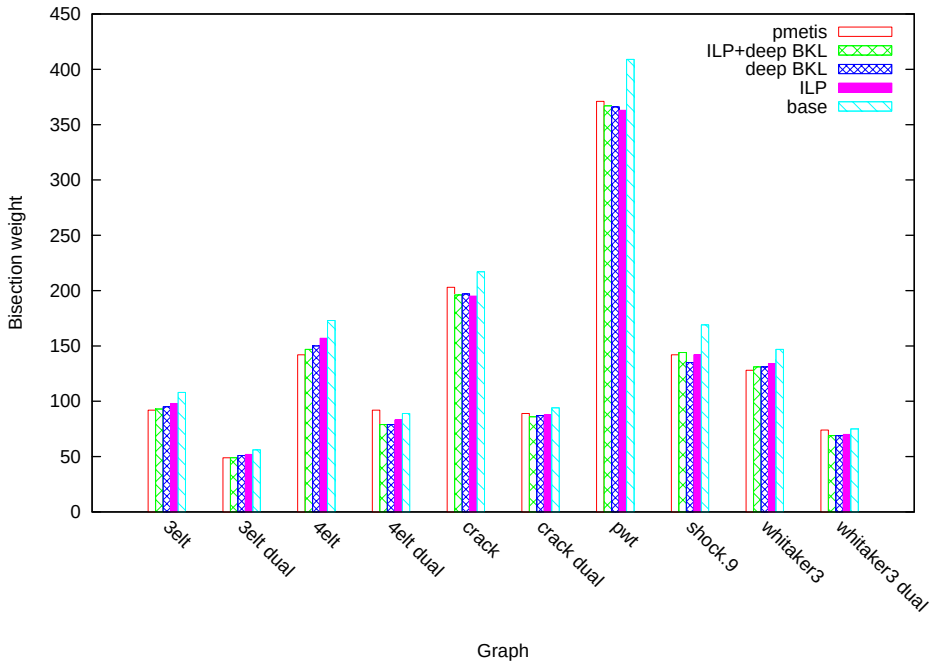


Figure 2: Bisection weights achieved by switching single components of the base heuristic.

instead of the BKL heuristic.

Finally we evaluate the effect of the striving magnitude on the output quality. Figure 4 shows the results of the base heuristic with alternate strived magnitudes and with our implementation of the GGGP heuristic as initial bisection.

As last test we try a "best-of" algorithm, where for each degree of freedom we use the strategy that performed best in the previous experiments, namely the base heuristic with ILP for the initial bisection and deep BKL as refiner. Figure 2 shows the results of the combination of the best strategies.

6.3 Evaluation

Figure 2 shows that the ILP and deep BKL strategies systematically make an improvement over the base heuristics. The average improvement over the base heuristic is 9.2% for ILP with a peek of 16% for the shock.9 graph, 11.2% for deep BKL with a peek of 20.1% also for the shock.9 graph. This improvement reveals potential in both the refinement phase and the initial bisection phase. For half of the graphs the best-of algorithm outperforms the pmetis routine with an average improvement of 2.04%. Considering the gap

	3elt	3elt_dual	4elt	4elt_dual	crack	crack_dual	pwt	shock.9	whitaker3	whitaker3_dual
density	108	53	192	91	**214**	94	405	163	143	75
density_2	109	57	181	91	216	94	408	**161**	142	75
density_3	107	55	179	94	218	94	412	168	144	75
shem	114	59	182	92	224	93	410	168	146	**74**
shem_metis	108	56	**173**	89	217	94	409	169	147	75
shem_density	111	56	185	96	221	95	412	165	145	**74**
shem_weight	113	61	189	93	216	96	411	167	**141**	77
shem_out	**102**	55	180	93	216	**91**	406	167	143	**74**
inner_out	105	55	176	**86**	222	**91**	406	165	147	76
inner_out_density	107	**51**	185	90	216	95	**399**	168	143	78
out_side_density	105	55	**173**	95	218	94	417	163	143	**74**
pmetis	92	49	142	92	203	89	371	142	128	74

Table 2: Bisection weights achieved by using the different ratings.

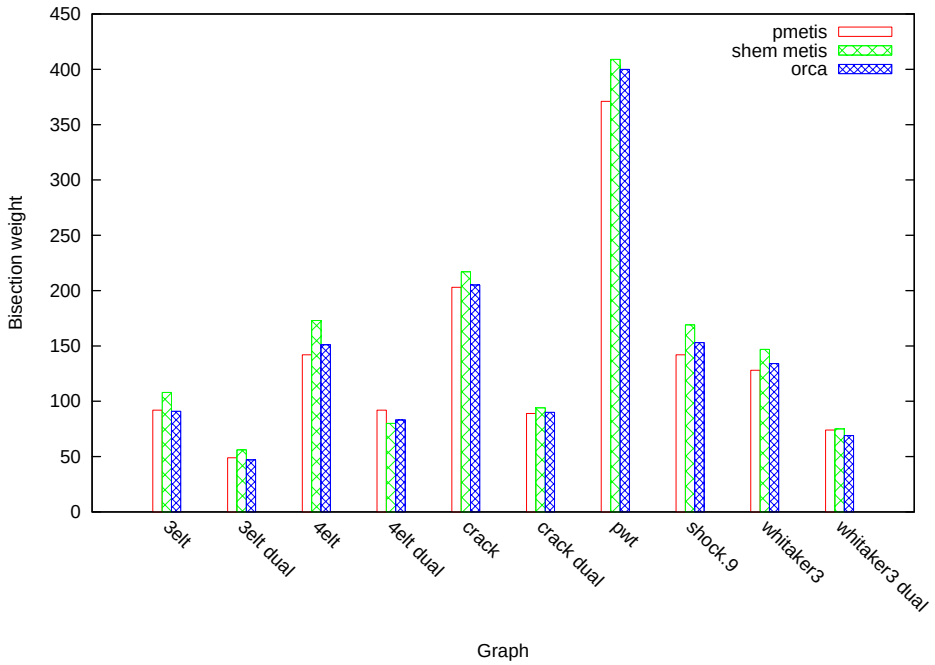


Figure 3: Bisection weights achieved by running one coarsing step with ORCA or shem_metis as merger.

between the base heuristic and the pmetis routine that originally aimed to be equal, this is not negligible.

Figure 3 shows that the average bisection weight that ORCA outputs is 0.7% worse than the pmetis routine. The Figure also shows an average improvement of 7.9% of ORCA over shem_metis. This is also not negligible considering that this improvement is obtained from only one step of coarsing. Furthermore, other than deep BKL and ILP, ORCA runs fast. Unfortunately, our experiments have shown that ORCA's performance decreases after one step of coarsing. However, the results shows that there also lies potential in using a clustering algorithm in the coarsing phase and more generally that there is potential in the coarsing phase.

In comparison to deep BKL, ILP and ORCA, Table 2 shows that altering the ratings does not seem to affect much the output. Even little improvement is shown for example by shem_metis over shem. In particular, no rating sticks out and no rating can be designated as the best as can be seen in Table 2. However, it seems that the out variants of the ratings are superior.

Altering the strived magnitude (results are listed in Figure 4) also does not seem to affect the ouput much. However, it seems that starting from 500 the quality of the results begins to decrease and at 1000 the quality of the

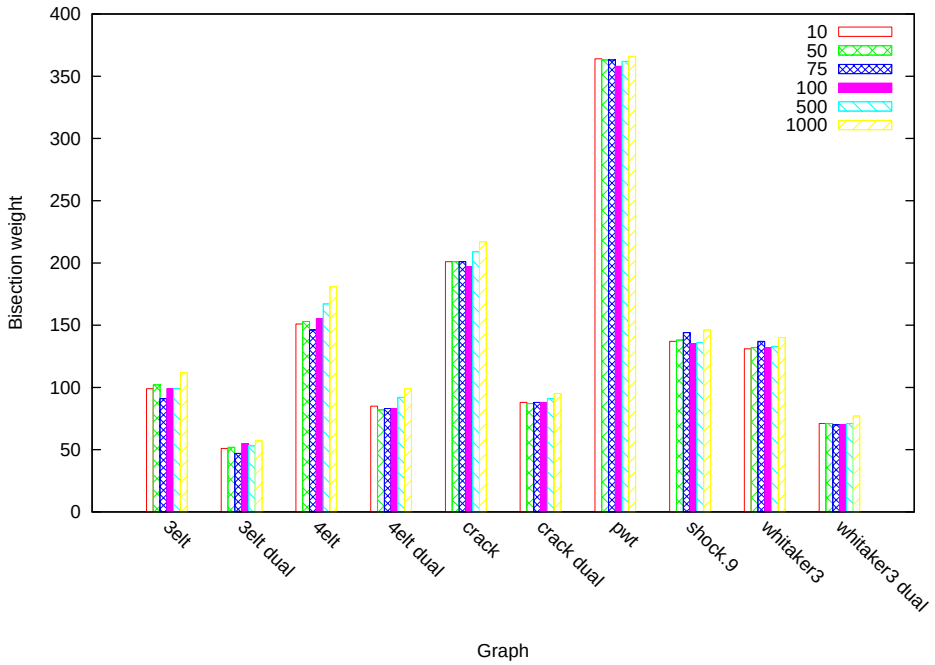


Figure 4: Bisection weights achieved by running the base heuristic with different strived magnitudes.

results are systematically worse than other strived magnitudes.

In order to give an idea of the runtime of our strategies we give a comparison of the runtime to the base heuristic for the shock.9 graph. But we emphasize that our focus is quality and not running time. On the shock.9 graph the base heuristic with ILP as initial bisection runs a factor 2 slower than the base heuristic, the base heuristic with deep BKL as refiner a factor 20 slower, the best-of algorithm a factor 22 slower, and the base heuristic a factor 4 slower than the pmetis routine. Table 3 shows the runtime of the different strategies.

The next section gives an interpretation of the results given in this section.

graph	pmetis	base	ILP	deep BKL	ILP + deep BKL
3elt	56	92	688	9122	10362
3elt_dual	60	121	767	8409	9135
4elt	146	298	814	14059	15066
4elt_dual	155	411	1104	13249	14394
crack	157	330	2174	11533	12858
crack_dual	181	430	2271	10417	11935
pwt	530	1112	1158	21770	25277
shock.9	213	808	1700	16606	18172
whitaker3	101	363	893	10184	10096
whitaker3_dual	109	435	1166	10415	10574

Table 3: runtime in milliseconds

7 Conclusion

The Multi-Level Technique has several degrees of freedom. In order to easily implement new strategies for these degrees of freedom and in order to easily combine these strategies we implemented a framework implementing the skeleton of the Multi-Level Technique.

Upon this framework we implemented and evaluated new alternative strategies with the aim to see how much potential lies in each degree of freedom where we favor quality of the result over running time. For the coarsing phase we proposed the idea to use the ORCA algorithm, a clustering algorithm working in a local way, by merging vertices in a same cluster together. For the coarsing phase we also evaluated new ratings for the rating based matching algorithm. Concerning the initial bisection phase we implemented the computation of an optimal initial bisection by using an ILP representation of the MinBisection problem and using the Gurobi solver. For the refinement phase we proposed deep BKL, a variant of BKL based on computing the pair of vertices to be swapped amongst more vertices than BKL.

Except for the ratings that do not seem to affect much the output bisection and where no rating sticks out, all proposed strategies show considerable improvements. These improvements suggest that all degrees of freedom have potential to decrease the output bisection weight. Thus it suggests that Multi-Level can be improved, at least in quality.

The next step would be to elaborate other new strategies in order to find out what properties decreases the output bisection weight to finally estimate what can be reached by the Multi-Level Technique. Finally the framework

and strategies runtime could be optimized to reach the runtime of the current Software Packages while trying to maintain the improvements of quality.

References

[1] http://staffweb.cms.gre.ac.uk/~wc06/partition/#software.

[2] http://www.boost.org/doc/libs/release/libs/graph/.

[3] http://gurobi.com/.

[4] http://en.wikipedia.org/wiki/Finite_element_method.

[5] http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/.

[6] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebre. Fast unfolding of communities in large networks. 2008.

[7] Robert Leland Bruce Hendrickson. A multilevel algorithm for partitioning graphs. 1993.

[8] Robert Leland Bruce Hendrickson. *The Chaco User's Guide, Version 2.0*, 1995.

[9] Ralf Diekmann Burkhard Monien, Robert Preis. Quality matching and local improvement for multilevel graph-partitioning. 1999.

[10] T. F. Chan, J. R. Gilbert, and S.-H. Teng. Geometric spectral partitioning. 1994.

[11] Jiye Liang Chuangyin Dang, Wei Ma. A deterministic annealing algorithm for approximating a solution of the min-bisection problem. 2008.

[12] J. P. Ciarlet and F. Lamour. On the validity of a front-oriented approach to partitioning large sparse graphs with a connectivity constraint. 1994.

[13] Daniel Delling, Robert Görke, Christian Schulz, and Dorothea Wagner. Orca reduction and contraction graph clustering. In *AAIM '09: Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management*, pages 152–165, Berlin, Heidelberg, 2009. Springer-Verlag.

[14] Nan Dun. An implementation of state-of-the-art graph bisection algorithms. 2006.

[15] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network-partitions. 1982.

[16] A. George and J. W.-H. Liu. Computer solution of large sparse positive definite systems. 1981.

[17] T. Goehring and Y. Saad. Heuristic algorithms for automatic graph paritioning. 1994.

[18] Manuel Holtgrewe. A scalable coarsening phase for a multi-level graph partitioning algorithm. Master's thesis, University of Karlsruhe, 2009.

[19] K. Jansen, M. Karpinski, A. Lingas, and E. Seidel. Polynomial time approximation schemes for max-bisection on planar and geometric graphs. 2001.

[20] Vinay Choudhary Jin-Yi Cai. Some results on matchgates and holographic algorithms. 2007.

[21] George Karypis and Vipin Kumar. *METIS - Unstructured Graph Partitioning and Sparse Matrix Ordering System - Version 2.0*, 1995.

[22] George Karypis and Vipin Kumar. A fast and high quality mutilevel scheme for partioning irregular graphs. 1998.

[23] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partioning graphs. 1970.

[24] Ralf Diekmann Robert Preis. *The PARTY Partitioning - Library User Guide - Version 1.1*, 1996.

[25] Christian Schulz. Design and experimental evaluation of a local graph clustering algorithm. Master's thesis, University of Karlsruhe, 2008.

[26] R. Krauthgamer U. Feige. A polylogarithmic approximation of the minimum bisection. 2000.

[27] Yinyu Ye. A .699-approximation algorithm for max-bisection. 1999.