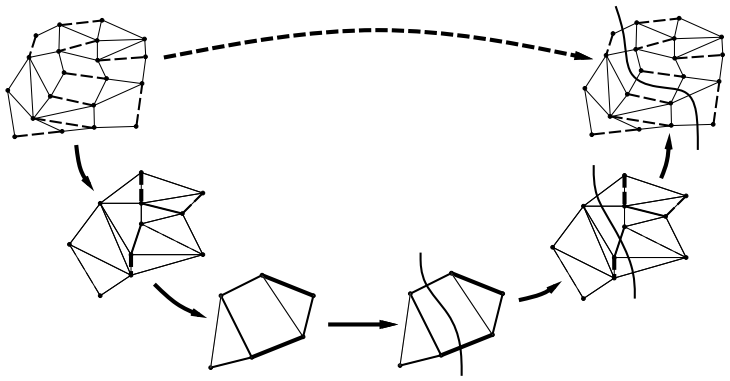
Analyses and Design of Efficient Graph Partitioning Methods



Dissertation

von

Robert Preis

Schriftliche Arbeit zur Erlangung des Grades
eines Doktors der Naturwissenschaften

Fachbereich Mathematik/Informatik
Universität Paderborn

Paderborn, 7. Juli 2000

DANKSAGUNGEN

Für die umfangreiche Betreuung meiner Arbeit durch viele Anregungen und interessante Diskussionen bedanke ich mich bei Prof. Dr. Burkhard Monien. Die enge Zusammenarbeit mit ihm hat meine wissenschaftliche Arbeit in den letzten Jahren maßgeblich geprägt. Prof. Dr. Friedhelm Meyer auf der Heide danke ich für die begleitende Betreuung und die Begutachtung dieser Dissertation.

Weiterhin bedanke ich mich bei den Kollegen aus der Arbeitsgruppe Monien. Insbesondere bei denjenigen, mit denen ich eng wissenschaftlich zusammengearbeitet habe: Prof. Dr. Sergej Bezroukov, Thomas Decker, Dr. Ralf Diekmann, Robert Elsässer, Marco Riedel, Dr. Markus Röttger, Ulf-Peter Schroeder und Jürgen Schulze. Weiterer Dank für die kollegiale Zusammenarbeit gilt dem PadFEM Team und Birger Boyens.

Neben den schon genannten Personen existiert durch das Graduiertenkolleg, den Sonderforschungsbereich, das Heinz Nixdorf Institut und das PC^2 ein fruchtbares wissenschaftliches Umfeld in Paderborn, für das ich mich bei allen Mitgliedern bedanke.

Ein Dankeschön an Christian Voss für die sprachliche Aufbereitung dieser Dissertation.

Schließlich bedanke ich mich bei Angelika für die schöne und wichtige Zeit außerhalb der Bearbeitung dieser Dissertation.

Robert Preis

Diese Dissertation wurde von den folgenden Projekten unterstützt.

- Deutsche Forschungsgemeinschaft/Heinz Nixdorf Institut
  Graduiertenkolleg 'Parallele Rechnernetze in der Produktionstechnik'
- Deutsche Forschungsgemeinschaft
  Sonderforschungsbereich 376 'Massive Parallelität'
- Europäische Union 'Human Capital and Mobility' - Programm
  'Efficient Use of Parallel Computers'

Mitglieder der Prüfungskommission:

- Prof. Dr. Burkhard Monien (Vorsitzender, Gutachter)
- Prof. Dr. Friedhelm Meyer auf der Heide (Gutachter)
- Prof. Dr. Hans Kleine Büning
- Prof. Dr. Franz Josef Rammig
- Dr. Peter Pfahler

Tag der mündlichen Prüfung: 30. November 2000

CONTENTS

1. INTRODUCTION

Graph partitioning problems occur in a wide range of applications. The task is to divide the set of vertices of a graph into a given number of parts. At the same time restrictions and cost functions have to be considered. In the classical graph partitioning problem, the parts have to have a balanced number of vertices and the number of edges incident to vertices of different parts are minimized. The later value is called the *cut size* of the partition. An example of a graph partitioned into 64 parts is presented in Fig. 1.1.

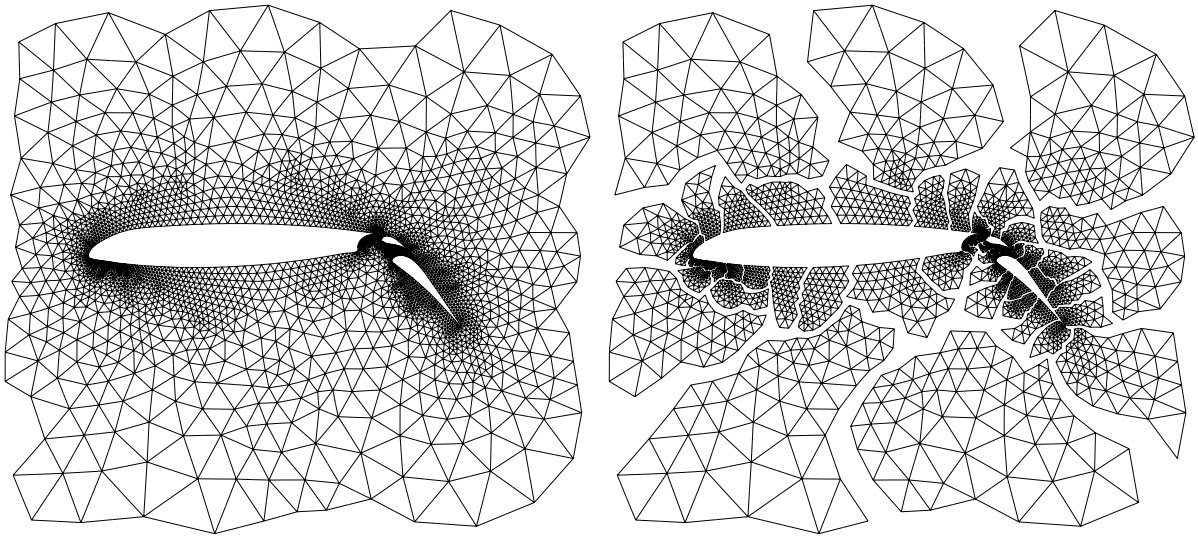


Fig. 1.1: A two-dimensional graph with 4253 vertices and 12,289 edges is partitioned into 64 parts.

Obviously, the partitioning of graphs with a regular structure like e. g. cycles, stars, grids or hypercubes is fairly easy. However, the partitioning of unstructured graphs such as the example in Fig. 1.1 is a complex task. Efficient methods for solving the graph partitioning problem are required. Furthermore, analytical results help to design and evaluate these methods.

Motivation

The efficient use of a parallel processor system is a major field of applications for the graph partitioning problem. The use of parallel systems is established in almost all areas of science and technology.

In order to execute a process on a parallel system, it must be possible to divide the process into subprocesses. Each subprocess has some computational load. There are interdependencies between the subprocesses such as data-transfers. The process can be modeled as a graph with the subprocesses being the vertices and the dependencies being the edges of the graph. It is called the *process graph*. The efficiency of the parallel calculation depends on the balance of the computational load among all processors. Therefore, the process graph should be partitioned according to two measures. Firstly, the load (number of vertices) should be equal for each part to ensure an equal distribution among all processors. Secondly, the number of edges incident to vertices of different parts should be minimized, in order to ensure a low overhead of the parallel computation.

Large numerical simulation problems are applications that are commonly to be executed on parallel processor systems. Examples are e. g. crash-simulations, computational fluid dynamics, weather forecasts or earthquake simulations. In these applications, the domain of the object is discretized by the use of a mesh. The approach to solving this problem is called *Finite Element Method* (FEM). The parallelization of numerical simulation algorithms usually follows the single-program multiple-data (SPMD) paradigm. The same code is executed on each processor but on different parts of the data. Consequently, the mesh is partitioned into p subdomains with p being the number of processors. Each subdomain is assigned to one processor. An example is displayed in Fig. 1.2 (left). Because iterative solution algorithms mainly perform local operations, i. e. data dependencies are defined by adjacencies in the mesh, the parallel algorithms do only require communication at the partition boundaries. An equal data distribution and a small boundary length ensure an efficient parallelization.

The quality of solutions obtained by such numerical approximation algorithms heavily depends on the accuracy of the discretization. In particular, in regions with steep solution gradients, the mesh has to be refined sufficiently, i. e. the elements have to be small, in order to allow an accurate approximation. There are applications which require to partition FEM-meshes with several millions of vertices. This is often the case for numerical simulations of three-dimensional objects. Thus, the methods need to be capable of partitioning very large graphs in a reasonable amount of time, i. e. the efficiency of the methods are an important aspect apart the quality of the partition. Furthermore, several applications need to solve a graph partitioning problem frequently, i. e. the time used to derive a solution should not destroy the gain of using a parallel processor system.

The construction of efficient processor networks is another field of applications for the graph partitioning problem. The processors in parallel systems are connected by some

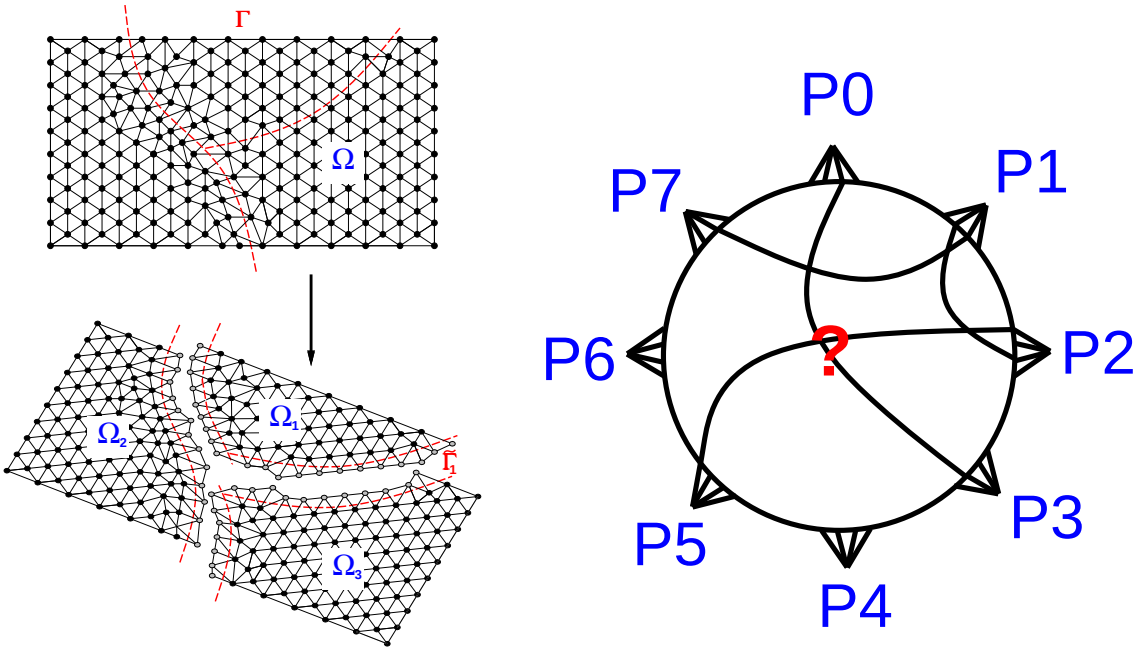


Fig. 1.2: Applications of graph partitioning. Left: Domain Decomposition of Finite-Element Meshes. The mesh has to be partitioned such that the work load is balanced and the length of the partition boundary is minimal. Right: Construction of efficient processor networks. The processors have to be connected such that any subset has many connections to the rest.

network structure. Each processor is connected to a small set of neighboring processors. In most parallel systems all of the processors are of the same type and have the same number of communication links. Therefore, the processor network can be modeled by a graph with a regular degree, i. e. each vertex is adjacent to a fixed number of neighbors. If a processor wants to communicate with a processor other than his neighbors, the communication message has to follow a path from the sending processor to the receiving processor along other processors. There is communication traffic on the network during a parallel computation.

It is our problem to construct a network so that hot spots in the communication traffic can be avoided. The bisection width of a graph is the minimum number of crossing edges of any partition of the graph into two equally sized parts. A high bisection width of the network graph is one measure for a smooth communication traffic. Thus, notwithstanding the way in which the network graph is partitioned into two equally sized parts, there is a minimum number of edges connecting these two parts. These kind of problems lead to the analyses of lower bounds on the bisection width of a graph.

Problem and Solutions

The calculation of an optimal solution of the graph partitioning problem for an arbitrary graph is **NP**-complete [GJ79]. The problem remains **NP**-complete in the simplest case when the number of parts is two. This is called the *graph bisection problem*. In [BCLS87] it is shown that the graph bisection problem is **NP**-complete for regular graphs.

Some analytical results on the bisection width of graphs have been published in recent years. In [FKN00] an algorithm is proposed which calculates a bisection with a cut size that differs from the bisection width by not more than a factor of $O(\sqrt{|V|} \cdot \log(|V|))$. Recently, an algorithm with a smaller factor of $O(\log^2(|V|))$ has been proposed in [FK00].

There are several analytical results on the bisection width of graphs with a regular degree. Almost every large d-regular graph $G = (V, E)$ has a bisection width of at least $c_d \cdot |V|$ where $c_d \to \frac{d}{4}$ as $d \to \infty$ [CE88, Bol88]. These bounds can be improved for small values of d. Almost every large 3-regular graph has a bisection width of at least $\frac{1}{9.9}|V| \approx 0.101|V|$ [KM92, KM93]. In Chapter 3 we show that all 3-regular graphs have a bisection width of at most $0.1982|V| + O(\log(|V|))$. In the case of 4-regular graphs, it is shown that almost all large 4-regular graphs have a bisection width of at least $\frac{11}{50}|V| = 0.22|V|$ [Bol88]. In Chapter 3 we show that the bisection width of 4-regular graphs is at most $\frac{|V|}{2} + 5$, regardless of the size of $|V|$. Recently, it was shown in [MP00] that the bisection width of large 4-regular graphs is at most $(0.4 + \delta)|V|$ if $|V| \geq n_0(\delta)$.

There are some approaches to calculating lower bounds on the bisection width of a graph. The bounds can be used to evaluate the quality of upper bounds. Furthermore, they can be used to speed up Branch & Bound strategies for calculating the bisection width of moderately-sized graphs. Leighton [Lei92] proposes a lower bound of the bisection width by calculating a routing scheme for all pairs of vertices. A small congestion of the routing scheme leads to a high lower bound (Section 3.3.2). Lower bounds on the bisection width can also be derived from algebraic graph theory by relating the bisection problem to an eigenvalue problem (Section 3.3.1.1). It is well known that the bisection width of a graph $G = (V, E)$ is at least $\frac{\lambda_2|V|}{4}$ with λ_2 being the second smallest eigenvalue of the Laplacian of G. This spectral bound is tight for some graphs.

The structure of an optimal bisection can be used to derive improved spectral lower bounds on certain graph classes [BEM+00] (see Section 3.3.1.2). For some classes of d-regular graphs we prove an improved lower bound on the bisection width of roughly $\frac{d}{d-2}\frac{\lambda_2|V|}{4}$. Furthermore, we prove a lower bound of $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2} \cdot \frac{\lambda_2|V|}{2}$ for the bisection width of all sufficiently large 3-regular graphs and a lower bound of $\frac{5-\lambda_2}{7-(\lambda_2-1)^2} \cdot \frac{\lambda_2|V|}{2}$ for the bisection width of all sufficiently large 4-regular graphs (Section 3.3.1.2). These lower bounds are higher than the classical bound of $\frac{\lambda_2|V|}{4}$ for sufficiently large graphs. We apply these bounds to Ramanujan graphs [Chi92, LPS88, Mar88, Mor94]. We are able to prove in Section 3.3.1.2 that any 3-regular Ramanujan graph has a bisection width of at

least $0.082|V|$. Furthermore, we prove that any 4-regular Ramanujan graph has a bisection width of at least $0.176|V|$. These values are the highest lower bounds for explicitly constructible 3- and 4-regular graphs.

It is a special task to construct small graphs with optimal characteristics. We aim at constructing a graph with highest bisection width among all d-regular graphs $G = (V, E)$ with given values d and $|V|$. There are some graphs which are optimal with regard to several measures such as the *Moore graphs*. The Petersen and the Hoffman/Singleton graphs are examples of that kind. We analyze the properties of these graphs and further extremal graphs in Section 3.4. Furthermore, it is possible to construct all non-isomorphic d-regular graphs with $|V|$ vertices up the certain values of d and $|V|$. We calculate the bisection width of all of these graphs and report the highest bisection width found. As an example, we derived that the highest bisection width of any 3-regular graph with 24 vertices is 8.

In recent years, efficient graph partitioning and load balancing strategies have been developed for a number of different applications. Although these solutions are currently not generally available, e. g. as part of parallel operating systems, only seldom is it difficult to integrate them into applications. We briefly categorize and describe some of the most relevant graph partitioning heuristics in Chapter 4.

In Chapter 5 we discuss the multilevel paradigm. The multilevel graph partitioning strategies have been proven to be very powerful approaches to efficient graph-partitioning [HL95b, KK99a, KK98c, Gup97, PMCF94, Bou98]. The efficiency of this strategy is dominated by two parts: graph coarsening and local improvement. Several methods were developed to solve the problems. However, their efficiency has only been proven on an experimental basis. For both steps of the multilevel strategy we use methods with analytically proven worst-case performance. For the coarsening part we use a new approximation algorithm for maximum weighted matching in general edge-weighted graphs [Pre99b]. The algorithm calculates a matching with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching in linear time. For the local improvement we use the Helpful-Set method [HM92, DMP95, MD97] which comes from a constructive proof of upper bounds on the bisection width of regular graphs. Overall, the combination of analytical methods for the two parts of the multilevel approach lead to an efficient graph-partitioning concept [MPD00].

Many graph partitioning methods were originally designed as graph bisection heuristic. Some of them are easy generalizable to partition the graph into any number p of parts. However, some methods are only very difficult to generalize. It is a common approach to use bisection heuristics recursively to get a partition into more than two parts. Several experiments have been performed to compare the direct p-partitioning approach with the recursive bisection approach. An analyses on this comparison is to be found in [ST97]. Most analytical and experimental results in this thesis focus on the case $p = 2$.

Efficiency and generalizations of graph partitioning methods strongly depend on specific implementations. There are several software libraries, each of which provides a range of different methods. Examples are CHACO [HL94a], JOSTLE [Wal00], METIS [KK98a] or SCOTCH [Pel96]. In Chapter 6 we discuss these libraries together with our own library PARTY [Pre98] which is a byproduct of this thesis. The goal of the libraries is to both provide efficient implementations and to offer a flexible and universal graph partitioning interface to applications.

Outline

The outline of this thesis is as follows. We define the graph partitioning problem in the following chapter. In Chapter 3 we present an analyses of the bisection width of graphs with a regular degree. It provides us with upper and lower bounds. Furthermore, it discusses and displays several regular graphs with high bisection width. Heuristics for graph partitioning are presented in Chapter 4. Established methods are categorized and briefly described. Furthermore, a new method called *Bubble* is developed, in order to optimize the shape of a partition. The efficient multilevel paradigm for graph partitioning is discussed in Chapter 5. A new matching algorithm is proposed. The algorithm calculates a matching with a high matching weight and a high matching cardinality in linear time. We show that this algorithm is extremely suitable to be used in the multilevel context. Chapter 6 describes the PARTY graph partitioning library.

Publications

This thesis was published in parts in the Journal on *Parallel Computing* [MPD00, DPSW00], the Proceedings of the *Symposium on Theoretical Aspects of Computer Science* [Pre99b], the *Euro-Par Parallel Processing Conference* [dBB+97, DPSW98, EFMP99, DMP00], the *Workshop on Graph-Theoretic Concepts in Computer Science* [BEM+00], the *Symposium on Parallel Algorithms and Architectures* [EMP00], the *Conference on Parallel and Distributed Processing Techniques and Applications* [SDP95], the *Conference on Advances in Computational Mechanics with Parallel and Distributed Processing* [PD97], the *Conference on Multiscale Phenomena and Their Simulation* [DMP97], the *Annual Meeting of the* Gesellschaft für Informatik [Pre99a], the *Summer School 'Partielle Differentialgleichungen, Numerik und Anwendungen'* [MDP96], and in the books *Conference on Software Engineering im Scientific Computing* [DP96] and *Parallel and Distributed Processing for Computational Mechanics* [DP99].

A previous work of the author on the subject of this thesis was published in the *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* [DMP95].

2. PROBLEMS AND DEFINITIONS

2.1 The Graph Partitioning Problem

We are confronted with the graph partitioning problem in a wide range of applications. In this section we define the graph partitioning problem. In the following, let $G = (V, E)$ be a graph with vertices V and undirected edges E as it is presented in Fig. 2.1.

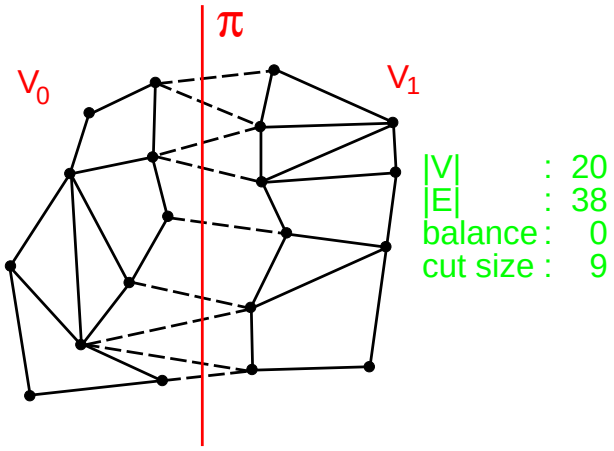


Fig. 2.1: Balanced bisection of a graph.

We also consider graphs with weights on the vertices and/or on the edges. Let $w : V \to I\!R^+$ denote the weights of the vertices. The weight is extended from a vertex to a set $U \subseteq V$ by $w(U) = \sum_{v \in U} w(v)$. For reasons of simplification, we use the same notation w for both vertex and edge weights. The distinction between vertices and edges is clear due to the parameter. Therefore, $w : E \to I\!R^+$ denotes the weights of the edges.

Definition 1: *Let $G = (V, E)$ be a graph with vertex set V and edge set E. Let*

$$\pi : V \to \{0, 1, ...p - 1\}$$

*be a p-**partition** of G that distributes the vertices among p parts V_0, V_1,...V_{p-1}. In the case of $p = 2$, π is called a* **bisection** *of G.*

There are a number of possible partitions of a graph and we try to select one, in order to optimize specific characteristics of the partition. The major characteristic feature of a partition is its balance and its cut size, which are defined as follows.

Definition 2: *Let*

$$bal(\pi) := max\{|V_i| - \frac{|V|}{p}; 0 \leq i < p\}$$

be the **balance** *of* π*. It is generalized to*

$$bal(\pi) := max\{w(V_i) - \frac{w(V)}{p}; 0 \leq i < p\}$$

for vertex weighted graphs. Furthermore, let

$$rbal(\pi) := \frac{bal(\pi)}{\frac{|V|}{p}} \quad \textit{or respectively} \quad rbal(\pi) := \frac{bal(\pi)}{\frac{w(V)}{p}}$$

be the **relative balance** *of* π*.*

A low balance of a partition ensures an even distribution of the vertices or the weights of the vertices among all parts. A partition π is called a **balanced partition** if $bal(\pi) < 1$. If vertex weights are considered, it is easy to see that there is a partition for all graphs with a balance less than the maximum weight of all vertices. Therefore, π is called a balanced partition if $bal(\pi) < max\{w(v); v \in V\}$.

Another cost measure is the cut size of the partition.

Definition 3: *Let*

$$cut(\pi) := |\{\{v, w\} \in E; \pi(v) \neq \pi(w)\}|$$

be the **cut size** *of* π*. It is generalized to*

$$cut(\pi) := \sum_{\{v,w\} \in E; \pi(v) \neq \pi(w)} w(\{v, w\})$$

for edge weighted graphs.

The cut size is the number of edges that are incident to vertices of different parts. It is the task of the **partitioning problem** to find a balanced partition π that minimizes the cut size. Thus, the parts are equally sized and as independent from each other as possible. If the graph models the dependencies (edges) between single execution tasks (vertices), firstly a balanced partition ensures an equal work load on all parts and secondly a small cut ensures a small processing overhead.

It is interesting to see, how many possible balanced partitions a graph may have and how difficult it is to compute the best possible partition. Let us also consider the simple case in which a graph is partitioned into $p = 2$ parts which is called the **bisection problem**.

Definition 4: *Let*

$$bw(G) = min\{cut(\pi); \pi \text{ is a balanced bisection of } G\}$$

be the **bisection width** *of graph* G.

The bisection width is the lowest possible cut size of all balanced bisections. The number of possible bisections of a graph is $\frac{1}{2} \cdot \binom{|V|}{\frac{|V|}{2}} = O(2^{|V|})$ (if $|V|$ is even) or $\binom{|V|}{\frac{|V|-1}{2}} = O(2^{|V|})$ (if $|V|$ is odd). The problem of calculating the bisection width for an arbitrary graph is **NP**-complete (see e. g. [GJS76, GJ79]). The bisection width is known for some specific graphs and can be computed (by the use of efficient enumeration schemes) for very small graphs with less than 100 vertices. However, it is not practical to compute the bisection width for graphs with several thousands of vertices.

There are several questions concerned with the partitioning problem, which are discussed in this work. A major aspect is the construction of upper and lower bounds on the bisection width of certain classes such as regular graphs (Chapter 3). The calculation of the bisection width is also **NP**-complete for the class of regular graphs [BCLS87]. In a d-regular graph, each vertex is adjacent to d different vertices. We consider d-regular graphs, because they are the most difficult examples of calculating upper bounds on the bisection width of graphs with maximum degree d.

Upper bounds can be used to guarantee a worst case communication bottleneck between any two parts of the graph. Furthermore, if we come to any constructive proofs, they can be used to construct bisections with guaranteed upper bounds on the cuts.

Lower bounds can be used for two major tasks. Firstly, if the bisection width of moderately sized graphs is calculated by the use of a Branch & Bound approach, the lower bounds can be used to ensure a certain minimum bisection width at a node of the Branch & Bound tree and may lead to a cut-off at that node. Secondly, there is a high demand of constructing graphs with a high bisection width in order to be used as a topology for routing-networks. In this context, lower bounds guarantee a minimum communication bandwidth between any two balanced parts of the network.

The calculation of a minimum cut remains very time consuming. However, most applications are satisfied with a fast calculation of a partition with a sufficiently small cut size. Many partitioning heuristics as described in Chapter 4 were developed to perform this task. The multilevel graph partitioning strategy is a very efficient approach to calculate a good partition of very large graphs (Chapter 5). Furthermore, there are several software tools for graph partitioning such as the PARTY library described in Chapter 6.

2.2 Related Problems

2.2.1 Graph Mapping

We mentioned the problem of constructing a suitable network of processors above. The standard graph partitioning problem aims to minimize the cut size of the partition. If the processor network consists of a complete graph, the cut size models the total number of inter-processor dependencies. However, if the processor network is not the complete graph, we have to generalize the graph partitioning problem to the *graph mapping* problem. Like in the graph partition problem, a mapping of a process graph onto a processor graph distributes the vertices of the process graph onto the vertices of the processor graph. Fig. 2.2 presents us with a telling example.

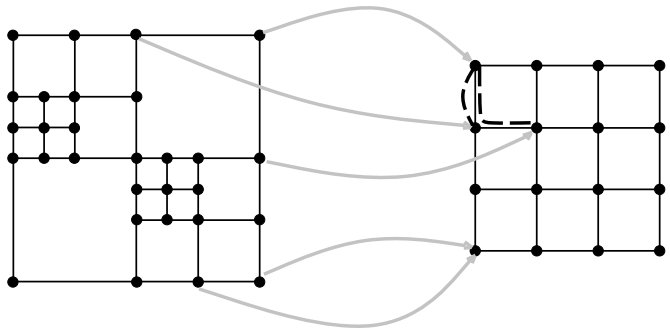


Fig. 2.2: Mapping the vertices of a process graph (left) onto the vertices of a processor graph (right). Edges of the process graph are mapped onto routing paths in the processor graph.

If two adjacent vertices of the process graph are not mapped onto the identical vertex or onto two adjacent vertices of the processor graph, a routing path in the processor graph marks the path along which information exchange between the subprocesses flows within the processor network. The *routing scheme* of the mapping is the set of all routing paths. It is the goal of the mapping problem to minimize the *dilation* and the *congestion* instead of the cut size. The dilation is the maximum length of any routing path. The congestion is the maximum number of routing paths along any edge of the processor graph. These values are displayed in Fig. 2.3

Some optimal mapping functions concerning load and dilation are known for well defined pairs of graphs like *Grids, Trees, Hypercubes*, etc. [Lei92, MS90, Röt98]. If the process graph or the processor graph do not belong to these graphs, heuristic methods have to be used, in order to find a good mapping function. The mapping problem is **NP**-complete, i. e. there is no efficient algorithm known which calculates the optimal solution. Some heuristics for solving this problem are discussed in [dBB+97].

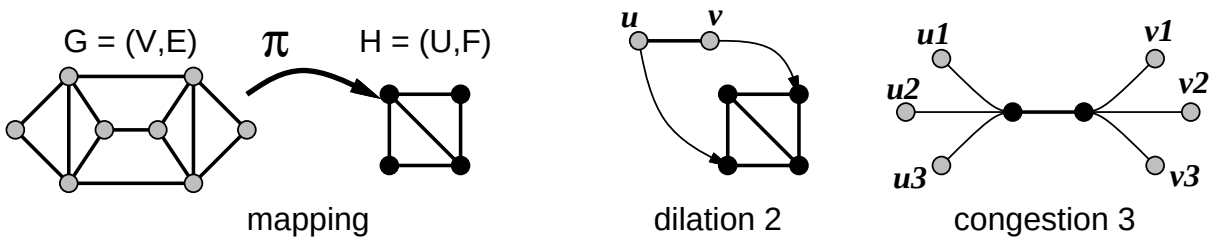


Fig. 2.3: The mapping π of the process graph $G = (V, E)$ onto the processor graph $H = (U, F)$. The dilation is the maximum length of any routing path in the processor graph. The congestion is the maximum number of paths along any edge of the processor graph.

2.2.2 Dynamic Loadbalancing

Depending on the application, the process graph can be *static* or *dynamic*, i. e. the computational load of the application nodes may either change during run-time, or not. The process graph may grow or shrink, i. e. vertices and edges can be inserted or deleted, or the vertex and edge weights may vary. Examples for such applications are adaptive finite element simulations where the mesh is refined according to the solution. An example is shown in Fig. 2.4 (left). Vertices have to be migrated between processors, in order to re-establish a balanced load distribution.

A number of solutions to the load balancing problem are based on re-partitioning, where (sometimes even sequential) mesh partitioning algorithms are used. The drawback of such an approach is twofold. Firstly, the mesh has to be routed to a single processor, if the partitioning tool is sequential. Such an approach is obviously neither scalable to large numbers of processors nor to large meshes. Secondly, even if a parallel partitioning tool is available (such as Par-METIS [SKK97b] or P-JOSTLE [WCE97]), a new partition may differ greatly from the existing one. As a result, large amounts of data may have to be shifted between processors. Although there are attempts to minimize this data-movement [OB98], comparisons to approaches which consider the existing distribution show that if the mesh adaption changes the mesh only slightly, only a small fraction of the data movement is really necessary [WCE97].

The alternatives are distributed load balancing algorithms which consider the graph partition and operate locally on the processor graph. A migration of vertices should consider locality, i. e. vertices should not be moved to processors where none of their neighbors are located. Therefore, the load balancing is performed on the *quotient graph* of the partition (Fig. 2.4 (center)).

The load balancing on the quotient graph is performed in three phases. The first phase (*'how much'*) determines a *balancing flow* (Fig. 2.4 (center)). The second phase (*'when'*) calculates a scheduling of the flow. If the load of a processor is higher than the total

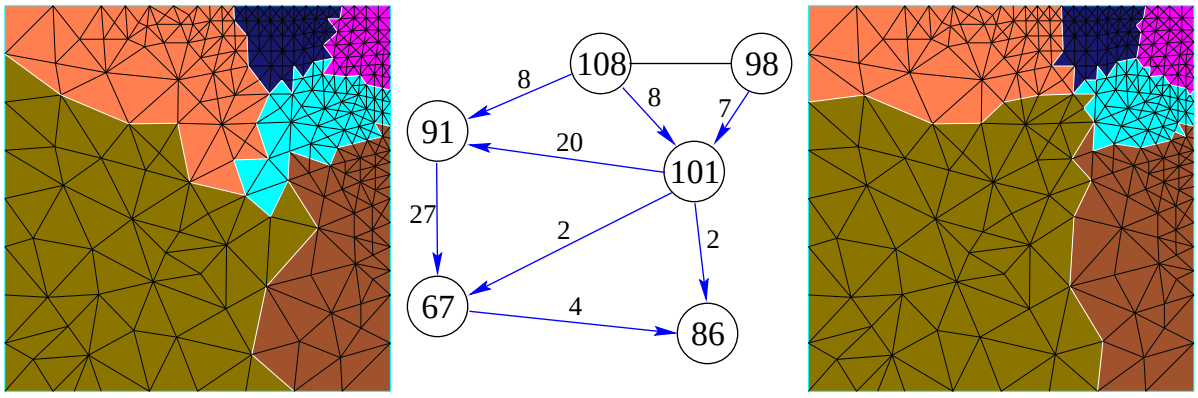


Fig. 2.4: An unbalanced partition of a mesh into six subdomains (left), the corresponding quotient graph with balancing flow (center), and the resulting balanced partition after migration (right).

sum of load it has to move, it can fulfill all demands at the same time. If this fact holds for all vertices, there is not any scheduling needed. Otherwise, the scheduling states the order in which load is moved. The third phase (*'which'*) migrates the load. It is our task to choose vertices for migration, in order to fulfill the flow demands. Their choice may consider additional cost criteria such as minimizing the cut size or optimizing the shape of subdomains.

The first phase of the load balancing, the "how much", can be regarded as a flow problem. For each edge of the quotient graph we have to calculate the amount of load that has to be transferred across it, in order to achieve a globally balanced system. We could use network flow algorithms to solve this problem. Unfortunately, they usually apply to more complicated cases of flow problems and are quite slow. Furthermore, they are difficult to parallelize. What is needed here is an algorithm which is easy to parallelize, scalable to large numbers of processors and local in the sense firstly that it does not need a global control and secondly that processors operate with only a limited number of others.

Scalable algorithms for our load balancing problem iteratively balance the load of a node with its neighbors until the whole network is globally balanced. The class of local iterative load balancing algorithms distinguishes between *diffusion* [Cyb89, Boi90] and *dimension exchange* [Cyb89, XL97] iterations. Diffusion algorithms assume that a node of the graph can send and receive messages to/from all its neighbors simultaneously, whereas dimension exchange does only balance iteratively with one neighbor after the other. The *quality* of a balancing algorithm can be measured firstly in terms of numbers of iterations that are required, in order to achieve a balanced state and secondly in terms of the amount of load moved over the edges of the graph. The earliest local method is the diffusive *first order scheme* (FOS) by [Cyb89]. It lacks performance because of its slow convergence. With the help of over-relaxation, FOS can be sped up by an order of magnitude [GMS96]

(*second order scheme*, SOS). Diffusive algorithms received new attention in the last couple of years [DMM98, DMN97, DSW98, HBE98, HB99, SKK97b, WCE97, XL97]. It has been shown that all local iterative diffusion algorithms determine the same flow. Moreover, this flow is minimal with regard to the l_2-norm [DFM99]. This fact ensures that there are no loops in the flow of the load. Generally, the l_2-minimal flow has also small values in the l_1- and l_∞-norm. Consequently, there is a small load-migration time in the parallel computation.

In [DFM99] an optimal load balancing scheme based on the spectrum of the graph was introduced. It only needs $m-1$ iterations with m being the number of distinct eigenvalues. The scheme keeps the load-differences small from step to step, i. e. it is numerically stable. In [EFMP99], a much simpler optimal scheme OPT is presented which does only need $m-1$ iterations, too. It might trap into numerical instable conditions, but there are rules of how to avoid those. The calculation of the spectrum for arbitrarily large graphs is very time-consuming. However, the spectrum is known for many classes of graphs and can efficiently be computed for small graphs.

So far we have only discussed the load balancing along the quotient graph. If there are not any dependencies between the subprocesses, we can regard the quotient graph as the complete graph. In order to avoid a high number of load balancing messages, each processor should only communicate with a small set of other processors. This topology defines the load balancing partners in the system. The choice of the topology can significantly influence the performance of the load balancing algorithm. If we apply the OPT scheme for load balancing on this topology, the maximum vertex degree and the number of eigenvalues of the network topology dominate the run-time for calculating the flow. For the second and third phases, a small flow volume and a small diameter of the topology keep the run time small. With respect to these properties, various network topologies are compared and proposed in [DMP00]. Overall, there is a demand for networks with small degrees, small numbers of distinct eigenvalues and small diameters for any value of node numbers.

The *alternating-direction-iterative* scheme [EFMP99] is a mixture of the diffusion and the dimension exchange methods. It reduces the number of iteration steps for networks which can be represented as cartesian powers of graphs. The drawback of this scheme is the fact that the resulting flow might have load-migration loops of infinitely large values. This fact would lead to a high load-migration overhead. As a (partial) remedy to this problem the *Mixed Direction Iterative* (MDI) scheme needs the same number of iterations but results in a much smaller flow [EFMP99]. In [DMP00], the *multi-diffusion* (MD) scheme for cartesian powers of graphs is proposed. Although the flow of MD is not necessarily l_2-minimal, the number of iterations can be decreased significantly.

So far, only little work has been done to address load balancing algorithms for heterogeneous networks. Heterogeneous networks consist of processors with different computing power or different memory capacity, or they have different communication links. Hetero-

geneous networks are extremely attractive because they are frequently found in computer networks which consist of processors by different manufacturers and of different types. Existing diffusion schemes can be generalized, in order to deal with heterogeneous networks. In these networks, every processor can have arbitrary computing power, and the load has to be balanced proportional to these weights [EMP00]. Furthermore, there can be different communication speeds between the processors [DFM99]. The balancing flow that is calculated by the schemes for homogeneous networks is minimal with regard to the l_2-norm. This holds true for the generalized schemes, too.

In the third phase, the elements that have to be moved are identified. When the load balancer chooses these elements, it aims at optimizing certain cost functions like the cut size or the shape of the subdomains. A detailed discussion of the choice of the elements is to be found in [DMM98, Die98, DSW98, Sch98, DPSW98, DPSW00].

3. BISECTION WIDTH OF GRAPHS WITH A REGULAR DEGREE

The traditional partitioning problem deals with arbitrary graphs. However, not all possible graphs are of the same interest. In each single application, the considered graphs have some similar characteristics. The knowledge about these characteristics can lead to highly efficient ways of dealing with this type of graphs. One of the typical features of graphs of a certain application is regularity. In a d-regular graph each vertex is connected to exactly d neighbors. The number of neighbors of a vertex is called the *degree* of the vertex.

As a motivating example of graphs with a bounded degree consider the case of Finite-Element-Methods for numerical simulations. The Finite-Element-Meshes are generated from the real object by specific mesh generating methods. These methods usually produce very structured graphs, and the vertices have a bounded degree. A partition of the mesh with a low cut ensures a low communication overhead during the parallel simulation. Another example of regular graphs is the construction of a routing network in a multiprocessor environment. It consists of many identical routing chips, each of them having the same number of communication links. Therefore, the graph that models the routing network consists of vertices that all have a fixed degree. The bisection width of a communication network is an important aspect to be considered. A high bisection width ensures a high bandwidth between any 2 parts of the network.

In [BCLS87] it is shown that the graph bisection problem is **NP**-complete for regular graphs. Therefore, there is a demand on bounds on the bisection width of regular graphs and we derive some new bounds on the bisection width in this chapter. Although the results are stated for graphs with a regular degree, they can easily be transformed to hold for graphs with maximum degree, too.

The bisection width is known for some graphs classes with regular degree such as grids, tori, hypercubes, cube-connected-cycles [MHT94] or butterflies [BLM+98].

We first give an overview of existing bounds in Section 3.1. We discuss upper bounds in Section 3.2 and lower bounds in Section 3.3. Furthermore, we calculate the maximum bisection widths of all small regular graphs in Section 3.4 and discuss some graph classes with high bisection width.

3.1 Overview

A very simple idea can be used to prove upper bounds for the bisection width of regular graphs. The idea is to start with an arbitrary bisection and to iteratively transform it into new ones by exchanging two vertices at a time, one of each side. As long as there is a pair of vertices of either side which would reduce the cut, this pair can be exchanged between the parts. In the case of d and n being even, this leads to a final bisection where at least one side has no vertices with the property that they are adjacent to more vertices of the other part than to vertices of the own part. Thus, the cut size is at most $\frac{n}{2} \cdot \frac{d}{2}$. Overall, one can easily show the following bounds.

$$bw(G_{d-regular}) \leq \begin{cases} \frac{dn}{4} = \frac{|E|}{2} & : \quad n \text{ and } d \text{ even} \\ \frac{(d-1)n}{4} + \frac{d+1}{2} & : \quad n \text{ even and } d \text{ uneven} \\ \frac{dn+d}{4} & : \quad n \text{ uneven and } d \text{ even} . \end{cases} \tag{3.1}$$

Clark and Entringer [CE88] show that almost every d-regular graph has a bisection width of at least $c_d \cdot n$ where $c_d \to \frac{d}{4}$ as $d \to \infty$. More precisely, Bollobas [Bol88] states that for $d \to \infty$ the bisection width of almost every d-regular graph is at least $(\frac{d}{2} - \sqrt{\log(2) \cdot d})\frac{n}{2}$. This shows that the simple bounds of equation (3.1) are approximately tight for graphs with a large degree.

Nevertheless, the upper bounds of equation (3.1) can be improved for small values of d. In the case of $d = 3$, Clark and Entringer [CE88] present an upper bound of $\frac{n+138}{3}$ for the bisection width. They and Bollobas [Bol88] independently show that almost all 3-regular graphs have a bisection width of at least $\frac{n}{11} \approx 0.09n$. This gap was improved by Kostochka and Melnikov, who showed that the bisection width of 3-regular graphs is at most $\frac{n}{4} + O(\sqrt{n} \log n)$ [KM92, KM93]. Furthermore, they showed that almost every 3-regular graph has a bisection width of at least $\frac{1}{9.9}n \approx 0.101n$. In Section 3.2.1 we improve the upper bound and show that all 3-regular graphs have a bisection width of at most $0.198n + O(\log(n))$.

In the case $d = 4$, the following upper bounds are proven by Hromkovic and Monien [HM92].

$$bw(G_{4-regular}) \leq \begin{cases} \frac{n}{2} + 4 & : \quad n \leq 60, n \equiv 0 \; mod \; 4 \\ \frac{n}{2} + 3 & : \quad n \leq 60, n \equiv 2 \; mod \; 4 \\ \frac{n}{2} + 1 & : \quad n \geq 350 \end{cases} \tag{3.2}$$

Section 3.4.2 reveals that these bounds are tight for small graphs with up to 30 vertices. The technique of proving these upper bounds led to the efficient and powerful *Helpful-Set* heuristic [DMP95] (Section 4.2.2). Bollobas [Bol88] proved that almost all 4-regular graphs have a bisection width of at least $\frac{11}{50}n = 0.22n$. In Section 3.2.2 we prove an upper bound of $\frac{n}{2} + 4$ for 4-regular graphs with $n \equiv 0$ or $n \equiv 1 \pmod 4$ and an upper bound of $\frac{n}{2} + 5$

if $n \equiv 2$ or $n \equiv 3 \pmod 4$. It generalizes the upper bound for $n \leq 60$ of equation (3.2) to any even value of n. Recently, it was shown in [MP00] that the bisection width of large 4-regular graphs (such that $n \geq n_0(\delta)$) is at most $(0.4 + \delta)n$.

The techniques of showing the bounds of equation (3.2), used in [HM92], are generalized to d-regular graphs by Diekmann and Monien [MD97]. They show that

$$bw(G_{d-regular}) \leq \frac{d-2}{4}n + 1 \tag{3.3}$$

for d even, $d \geq 4$, and $n \geq n_0(d)$.

Alon [Alo97] uses complex probabilistic arguments to show that the bisection width is at most $(\frac{d}{4} - \frac{3\sqrt{d}}{32\sqrt{2}})n$ for arbitrary d-regular graphs with $n > 40d^9$. This bound is very tight, because there is a class of regular graphs called *Ramanujan Graphs* [Chi92, LPS88, Mar88, Mor94]. Using eigenvalue arguments, it can be shown that their bisection width is $bw \geq (\frac{d}{4} - \frac{1}{2}\sqrt{d-1})n$ [DH73, Fie73, Fie75, AM85, Alo86]. This only differs from the upper bound by a constant factor in the low order term.

There are some approaches to calculating lower bounds on the bisection width of a graph. We discuss spectral lower bounds in Section 3.3.1 and a lower bound based on a routing scheme [Lei92] in Section 3.3.2. In Section 3.3.3 we do a comparison between these two lower bound approaches.

The classical spectral lower bound is $\frac{\lambda_2|V|}{4}$ with λ_2 being the second smallest eigenvalue of the Laplacian of G (Section 3.3.1.1). This bound is tight for certain graphs. For some classes of d-regular graphs we prove a higher lower bound on the bisection width of roughly $\frac{d}{d-2}\frac{\lambda_2 n}{4}$ (Section 3.3.1.2). Furthermore, we prove a lower bound of $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2} \cdot \frac{\lambda_2 n}{2}$ for the bisection width of all sufficiently large 3-regular graphs and a lower bound of $\frac{5-\lambda_2}{7-(\lambda_2-1)^2} \cdot \frac{\lambda_2 n}{2}$ for the bisection width of all sufficiently large 4-regular graphs (Section 3.3.1.2). These lower bounds are higher than the classical bound of $\frac{\lambda_2|V|}{4}$ for sufficiently large graphs. We apply these bounds to Ramanujan graphs [Chi92, LPS88, Mar88, Mor94]. We are able to prove in Section 3.3.1.2 that any 3-regular Ramanujan graph has a bisection width of at least $0.082|V|$. Furthermore, we prove that any 4-regular Ramanujan graph has a bisection width of at least $0.176|V|$. These values are the highest lower bounds for explicitly constructible 3- and 4-regular graphs.

In Section 3.4 we construct several small graphs with high bisection width. We enumerate all small graphs with $|V|$ vertices and degree d for small values of $|V|$ and d. We calculate the bisection width of all such graphs and report the the highest bisection width found. Some graph classes like *Moore graphs* or *Cages* are often among these extreme graphs.

3.2 Upper Bounds

In this section we derive upper bounds on the bisection width of 3- and 4-regular graphs. The techniques used to prove the results are based on a local exchange of sets. If a set of vertices is moved from one part to the other, the helpfulness of the set is the amount by which the cut of the bisection decreases.

Definition 5 (Helpfulness [HM92]): *Let $G = (V, E)$ be a graph and $\pi(G)$ a bisection of V. For a subset $S \subset V_0(\pi)$ let*

$$H(S) = |\{\{v, w\} \in E; v \in S, w \in V_1(\pi)\}| - |\{\{v, w\} \in E; v \in S, w \in V_0(\pi) \backslash S\}|$$

be the **helpfulness** *of S. S is called $H(S)$-**helpful**. For a subset $S \subset V_1(\pi)$ define the same with changed $V_0(\pi)$ and $V_1(\pi)$.*

For the union $S = S_1 \cup S_2$ of two sets $S_1 \subset V_p$ and $S_2 \subset V_p$, $p \in \{0, 1\}$, it is $H(S) = H(S_1) + H(S_2) - H(S_1 \cap S_2) + 2|\{\{u, v\} \in E; u \in S_1 \backslash S_2, v \in S_2 \backslash S_1\}|$.

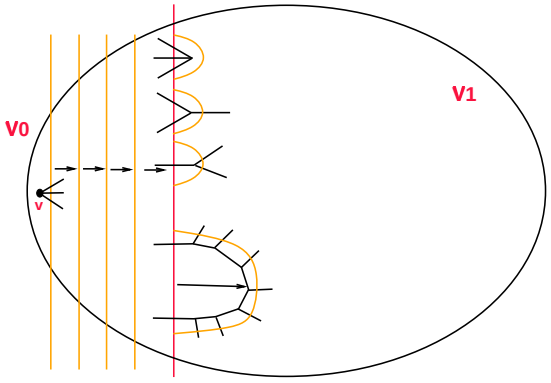
3.2.1 Greedy Balancing for an Upper Bound of 3-Regular Graphs

In this section we prove an upper bound of $0.198n + O(\log(n))$ for the bisection width of 3-regular graphs. This improves upon the bound of $\frac{n+138}{3}$ by Clark and Entringer [CE88].

Besides, it is possible to derive an upper bound for 3-regular graphs from an upper bound for 4-regular graphs. According to Petersen [Pet91], every 2-connected 3-regular graph has a perfect matching. A 4-regular graph can be constructed by merging the incident vertices for each matching edge. The resulting 4-regular graph has half of the vertices and, therefore, the upper bound on the 3-regular graph is half the size of the upper bound for the 4-regular graph. By the use of the upper bound of $(0.4 + \delta)n$ for the bisection width of large 4-regular graphs [MP00], we can demonstrate that the bisection width of large 2-connected 3-regular graphs is at most $(0.2 + \frac{\delta}{2})n$.

In the following, we introduce the greedy balancing strategy as presented in Fig. 3.1 to show the upper bound of $0.198n + O(\log(n))$. It is a constructive strategy, i. e. it does not only prove upper bounds, but also results in an algorithm of how to construct a bisection with a bisection width of at most that upper bound. The strategy starts with $V_0(\pi)$ that consists of a single vertex and $V_1(\pi) = V \backslash V_0(\pi)$. The result is a cut of size 3. As long as the bisection is not balanced, a set $S \subset V_1(\pi)$ is moved to $V_0(\pi)$ and causes an increase of the cut by $-H(S)$ (see Definition 5).

There are several strategies for choosing a set $S \subset V_1(\pi)$. The easiest way is to take a single border vertex. It is a vertex incident to a cut edge, as shown in Fig. 3.1. In the



Choice for vertex moves:

- single border vertex
  $\rightarrow$ cut increases by at most 1.
- set S of path vertices that connect 2 border vertices
  $\rightarrow$ cut increases by at most $|S| - 2$.

Fig. 3.1: Greedy balancing.

following only connected graphs are considered, i. e. there is always a border vertex. If a border vertex is moved from one part to the other, the cut will increase by at most 1.

A better strategy is to move larger sets S with $H(S) > -|S|$ as shown in Tab. 3.1 and illustrated in Fig. 3.2. According to the relation between $|V_1(\pi)|$ and $cut(\pi)$ as shown in the first column, the second column states the existence of a set with the specific size/helpfulness relation. The first row corresponds to the consecutive moves of $|S|$ border vertices. A better strategy is to move the vertices of a certain path which connects two border vertices as shown in Fig. 3.1. If a set S of vertices of a path that connects two border vertices is moved, the cut will increase by at most $|S| - 2$. Short paths have a good ratio between the increase of the cut and the length of the path, and should preferably be used for moves.

Tab. 3.1: Overview of the Balancing Lemma for Bisections of 3-Regular Graphs.

$\|V_1(\pi)\| <$	for $x \in I\!N$, $0 < x < \|V_1(\pi)\|$ it holds $\exists S \subset V_1(\pi)$ with $\|S\| = x$ and $H(S) \geq$	Lemma
n	$-\|S\|$	trivial
$(2^i - 1)cut(\pi)$	$-\|S\| + 2\lfloor \frac{\|S\|}{2i-1} \rfloor$	(1)
$7cut(\pi)$	$-\frac{\|S\|}{3} - 5 - 2\log(\|S\|)$	(3)
$\frac{10}{3}cut(\pi)$	$-\frac{\|S\|}{5} - 4$	(4)
$3cut(\pi)$	$-3 - 2\log(\|S\|)$	(2)

Lemma 1: *Let $G = (V, E)$ be a 3-regular graph and $\pi(G)$ a bisection of V. Then there is either a vertex $v \in V_1(\pi)$ with $H(\{v\}) \geq 0$ or it holds the following for any $i \geq 2$.*

If $|V_1(\pi)| < (2^i - 1)cut(\pi)$, there is a path with at most $2i-1$ vertices which connects two border vertices in $V_1(\pi)$. The set S that consists of the vertices of the path has helpfulness $H(S) \geq -|S| + 2$.

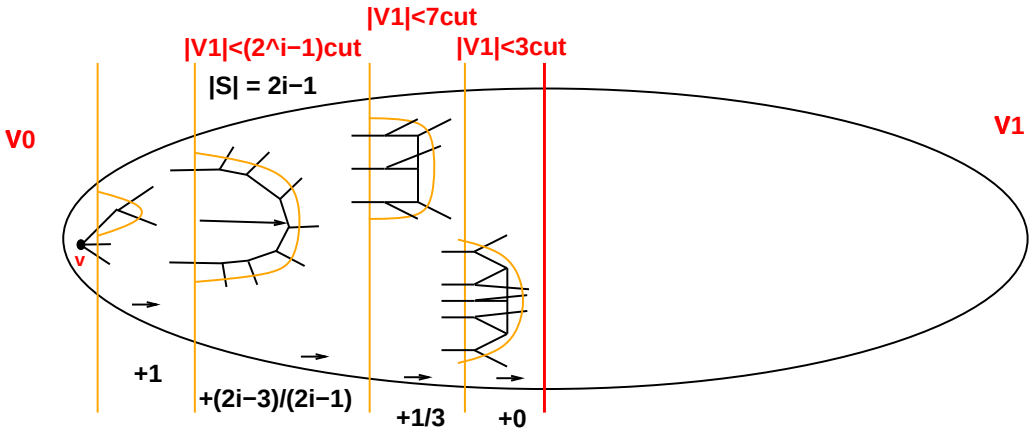


Fig. 3.2: Overview of Balancing Lemma for bisections of 3-regular graphs.

Proof: Assume the case that each vertex $v \in V_1(\pi)$ has $H(\{v\}) < 0$, i. e. for 3-regular graphs all border vertices are incident to one cut edge and to two internal edges. Thus, the number of border vertices is equal $cut(\pi)$.

Assume that all paths that connect two border vertices consist of at least $2i$ vertices. The vertices of $V_1(\pi)$ form complete binary trees of depth $i-1$ with the border vertices as roots on depth 0. I. e. there are at least $(2^i-1)cut(\pi)$ vertices in $V_1(\pi)$ which is a contradiction to the statement above.

Each set S of vertices of a path that connects two border vertices is connected to vertices of $V_0(\pi)$ by at least two edges and to $V_1(\pi)\backslash S$ by at most S edges. Thus, $H(S) \geq -|S| + 2$. ⋈

Further improvements with a better ratio between the size and the helpfulness for certain ratios between $|V_1(\pi)|$ and $cut(\pi)$ are listed in the last three rows of Tab. 3.1. Before we prove those lemma, we first show the main theorem of this section.

Theorem 1: *The bisection width of a connected 3-regular graph* $G = (V, E)$ *is at most* $\frac{33473425}{168844032}|V| + \frac{227}{76}\log(\frac{|V|}{2}) + \frac{891194179}{6595470} \approx 0.198|V| + 2.987\log(\frac{|V|}{2}) + 135.122$.

Proof: The construction starts with an initial bisection π where $V_0(\pi)$ consists of an arbitrary vertex and $V_1(\pi)$ consists of all other vertices. This leads us to a cut of 3. Five different phases are used to move vertices from the larger part to the smaller one without too much increasing the cut. The size of V_1 consistently decreases. In phase 1, the cut increases by one for each vertex moved from V_1 to V_0. Then, in phase 2, the cut increases by a smaller fraction of the number of moved vertices. Phases 3 and 4 further reduce the fraction. Finally, phase 5 moves a set of vertices to balance the bisection by increasing the cut only logarithmic in the size of the moved set. The sets to be moved are chosen according to the results listed in Tab. 3.1.

Phase 1: From π to π_6 (move single border vertices).

Take bisection π with $|V_0(\pi)| = 1$ and $cut(\pi) = 3$ and construct a new bisection π_6 by repeatedly moving single border vertices from $V_1(\pi)$ to $V_0(\pi)$ until either the bisection becomes balanced or the cut achieves $cut(\pi_6) = \lceil \frac{n}{2^6} \rceil + 2$. A move of a border vertex increases the cut by not more than one. Thus, this phase moves at least $\lceil \frac{n}{2^6} \rceil - 1$ vertices and at most $n - \lceil \frac{n}{2^6} \rceil$ vertices remain in $V_1(\pi_6)$. It holds $|V_1(\pi_6)| \leq n - \lceil \frac{n}{2^6} \rceil \leq (1 - \frac{1}{2^6})(cut(\pi_6) - 2)2^6 < (2^6 - 1)cut(\pi_6)$. It is

$$cut(\pi_6) = \lceil \frac{n}{2^6} \rceil + 2 < \frac{1}{64}n + 3 \approx 0.016n + 3$$

and

$$|V_1(\pi_6)| < 63cut(\pi_6) \leq \frac{63}{64}n + 189 \approx 0.984n + 189 \; .$$

Phase 2: From π_6 over π_5 and π_4 to π_3 (Lemma 1).

Use Lemma 1 and perform three sub-phases to construct new bisections π_5, π_4 and π_3. Each of the bisections π_i, $3 \leq i \leq 6$, will have the property $|V_1(\pi_i)| < (2^i - 1)cut(\pi_i)$. This already holds true for π_6.

To get from a bisection π_i to a bisection π_{i-1} we repeatedly move $2i-1$ vertices from V_1 to V_0. Lemma 1 ensures that there is a path of at most $2i - 1$ vertices that connects two border vertices. We move such a path from V_1 to V_0 and, if it is smaller than $2i - 1$, we move further border vertices such that we always move $2i - 1$ vertices. This increases the cut by not more than $2i - 3$.

The movement of $2i - 1$ vertices is repeated until a bisection π_{i-1} with the property $|V_1(\pi_{i-1})| < (2^{i-1} - 1)cut(\pi_{i-1})$ results. It is important that this relation did not hold true for the bisection before we moved the last set of $2i-1$ vertices, i. e. it holds $|V_1(\pi_{i-1})| + (2i - 1) \geq (2^{i-1} - 1)(cut(\pi_{i-1}) - (2i - 3))$. Overall, we moved $\frac{|V_1(\pi_i)| - |V_1(\pi_{i-1})|}{2i-1}$ sets, each of which increasing the cut by not more than $2i - 3$. Consequently, $cut(\pi_{i-1})$ is bounded by

$$\begin{aligned} & cut(\pi_{i-1}) \leq cut(\pi_i) + \frac{|V_1(\pi_i)| - |V_1(\pi_{i-1})|}{2i-1}(2i - 3) \\ \leq \; & cut(\pi_i) + \frac{(2^i - 1)cut(\pi_i) - (2^{i-1} - 1)(cut(\pi_{i-1}) - (2i - 3)) + (2i - 1)}{2i - 1}(2i - 3) \end{aligned}$$

which is equivalent to

$$cut(\pi_{i-1}) \leq \frac{2^{i-1}(2i - 3) + 1}{2^{i-2}(2i - 3) + 1}cut(\pi_i) + 2i - 3 \; .$$

Tab. 3.2 displays results for this equation starting with π_6 and the newly constructed bisections π_5, π_4 and π_3.

Tab. 3.2: Resulting cut sizes for greedy balancing of paths that connect 2 border vertices.

π_i	$cut(\pi_i) \leq$			$\lvert V_1(\pi_i)\rvert < (2^i-1)cut(\pi_i) \leq$	
π_6	$\lceil \frac{1}{26}n \rceil + 2$	$< \frac{1}{64}n + 3$	$\approx 0.016n + 3$	$\frac{63}{64}n + 189$	$\approx 0.984n + 189$
π_5	$\frac{289}{145}cut(\pi_6) + 9 \leq$	$\frac{289}{9280}n + \frac{2172}{145}$	$\approx 0.031n + 14.979$	$\frac{8959}{9280}n + \frac{67332}{145}$	$\approx 0.965n + 464.359$
π_4	$\frac{113}{57}cut(\pi_5) + 7 \leq$	$\frac{32657}{528960}n + \frac{101097}{2755}$	$\approx 0.062n + 36.696$	$\frac{32657}{35264}n + \frac{303291}{551}$	$\approx 0.926n + 550.437$
π_3	$\frac{41}{21}cut(\pi_4) + 5 \leq$	$\frac{1338937}{11108160}n + \frac{1478084}{19285}$	$\approx 0.121n + 76.644$	$\frac{1338937}{1586880}n + \frac{1478084}{2755}$	$\approx 0.844n + 536.510$

The result of this phase is a bisection π_3 with

$$cut(\pi_3) \leq \frac{1338937}{11108160}n + \frac{1478084}{19285} \approx 0.121n + 76.644$$

and

$$|V_1(\pi_3)| < 7cut(\pi_3) \leq \frac{1338937}{1586880}n + \frac{1478084}{2755} \approx 0.844n + 536.510\,.$$

Phase 3: From π_3 to π_{2-3} (Lemma 3).

Take bisection π_3 and use Lemma 3 to construct a more balanced bisection π_{2-3} with $|V_1(\pi_{2-3})| < \frac{10}{3}cut(\pi_{2-3})$. Lemma 3 ensures that if we move a set S of size $|S| = |V_1(\pi_3)| - |V_1(\pi_{2-3})|$, the cut will not increase by more than $-\frac{|S|}{3} - 5 - 2\log(|S|)$. We choose the size of S such that it ensures the relation $|V_1(\pi_{2-3})| < \frac{10}{3}cut(\pi_{2-3})$. However, we do not want to choose S too large and choose S such that the movement of a set of size $|S| - 1$ would not guarantee the relation we need for bisection π_{2-3}, i. e. $|V_1(\pi_{2-3})| + 1 \geq \frac{10}{3}(cut(\pi_{2-3}) - \frac{1}{3} - \log(|S|) + \log(|S| - 1)) > \frac{10}{3}(cut(\pi_{2-3}) - \frac{4}{3})$. Consequently,

$$\begin{aligned} cut(\pi_{2-3}) &\leq cut(\pi_3) + \frac{|V_1(\pi_3)| - |V_1(\pi_{2-3})|}{3} + 2\log(|V_1(\pi_3)| - |V_1(\pi_{2-3})|) + 5 \\ &\leq cut(\pi_3) + \frac{7cut(\pi_3) - \frac{10}{3}cut(\pi_{2-3}) + \frac{49}{9}}{3} + 2\log(\frac{n}{2}) + 5\,. \end{aligned}$$

It is equivalent to

$$cut(\pi_{2-3}) \leq \frac{30}{19}cut(\pi_3) + \frac{18}{19}\log(\frac{n}{2}) + \frac{184}{57}\,.$$

The result of this phase is a bisection π_{2-3} with

$$cut(\pi_{2-3}) \leq \frac{1338937}{7035168}n + \frac{18}{19}\log(\frac{n}{2}) + \frac{27315200}{219849} \approx 0.190n + 0.947\log(\frac{n}{2}) + 124.245$$

and

$$\begin{aligned} |V_1(\pi_{2-3})| < \frac{10}{3}cut(\pi_{2-3}) &\leq \frac{6694685}{10552752}n + \frac{60}{19}\log(\frac{n}{2}) + \frac{273152000}{659547} \\ &\approx 0.634n + 3.158\log(\frac{n}{2}) + 414.151\,. \end{aligned}$$

Phase 4: From π_{2-3} to π_2 (Lemma 4).

Take bisection π_{2-3} and use Lemma 4 to construct a more balanced bisection π_2 with $|V_1(\pi_2)| < 3cut(\pi_2)$. Lemma 4 ensures that if we move a set S of size $|S| = |V_1(\pi_{2-3})| - |V_1(\pi_2)|$, the cut will not increase by more than $-\frac{|S|}{5} - 4$. We choose the size of S such that it ensures the relation $|V_1(\pi_2)| < 3cut(\pi_2)$. However, we do not want to choose S too large and choose S such that the movement of a set of size $|S| - 1$ would not guarantee the relation we need for bisection π_2, i. e. $|V_1(\pi_2)| + 1 \geq (2^2 - 1)(cut(\pi_2) - \frac{1}{5})$. Consequently,

$$\begin{aligned} cut(\pi_2) &\leq cut(\pi_{2-3}) + \frac{|V_1(\pi_{2-3})| - |V_1(\pi_2)|}{5} + 4 \\ &\leq cut(\pi_{2-3}) + \frac{\frac{10}{3}cut(\pi_{2-3}) - 3cut(\pi_2) + \frac{8}{5}}{5} + 4 \,. \end{aligned}$$

This is equivalent to

$$cut(\pi_2) \leq \frac{25}{24}cut(\pi_{2-3}) + \frac{27}{10} \,.$$

The result of this phase is a bisection π_2 with

$$cut(\pi_2) \leq \frac{33473425}{168844032}n + \frac{75}{76}\log(\frac{n}{2}) + \frac{871407769}{6595470} \approx 0.198n + 0.987\log(\frac{n}{2}) + 132.122$$

and

$$\begin{aligned} |V_1(\pi_2)| < 3cut(\pi_2) &\leq \frac{33473425}{56281344}n + \frac{225}{76}\log(\frac{n}{2}) + \frac{871407769}{2198490} \\ &\approx 0.595n + 2.960\log(\frac{n}{2}) + 396.366 \,. \end{aligned}$$

Phase 5: From π_2 to π_f (Lemma 2).

Take bisection π_2 and use Lemma 2 to construct a final and balanced bisection π_f. The size of the balancing set is $|S| = |V_1(\pi_2)| - \lceil \frac{n}{2} \rceil \leq \frac{n}{2}$. Consequently,

$$cut(\pi_f) \leq cut(\pi_2) + 2\log(\frac{n}{2}) + 3 \,.$$

The result of this phase is a balanced bisection π_f with

$$\begin{aligned} cut(\pi_f) &\leq \frac{33473425}{168844032}n + \frac{227}{76}\log(\frac{n}{2}) + \frac{891194179}{6595470} \\ &\approx 0.198n + 2.987\log(\frac{n}{2}) + 135.122. \end{aligned}$$

⋈

Lemma 2: *Let $G = (V, E)$ be a connected 3-regular graph and let π be a bisection of G. If $|V_1(\pi)| < 3 \cdot cut(\pi)$ and $0 < x < |V_1(\pi)|$, $\exists S \subset V_1(\pi)$ with either*

(i) $|S| \leq x$ *and* $H(S) \geq 0$ *or*

(ii) $|S| = x$ *and* $H(S) \geq -3$ *or*

(iii) $\frac{x}{2} \leq |S| \leq x$ *and* $H(s) \geq -2$.

In other words, $\exists \hat{S} \subset V_1(\pi)$ with $|\hat{S}| = x$ and $H(\hat{S}) \geq -3 - 2\log(|\hat{S}|)$.

Proof: The vertices of $V_1(\pi)$ are classified according to their distance to the cut: $V_1(\pi) = C \uplus D \uplus E$ with C vertices being at a distance of 1 to the cut, i. e. they are incident to a cut edge. D vertices are at a distance of 2 and E vertices at a distance of least 3. D vertices are further categorized with respect to the number of adjacent vertices in C. I. e. $D = D_3 \uplus D_2 \uplus D_1$ and each D_x vertex is adjacent to x vertices in C. Therefore, $V_1(\pi) = C \uplus D_3 \uplus D_2 \uplus D_1 \uplus E$.

First, several combinations lead to sets of type (i). Some examples of such configurations are presented in Fig. 3.3. Among these there are C vertices incident to two or three cut edges and any set of two adjacent C vertices, leading to $cut = |C|$, if there are not any of these sets. A D_3 vertex, together with its adjacent C vertices, also form a set of type (i). In addition, two adjacent D_2 vertices or two D_2 vertices connected via a path of D_1 vertices also form a set of type (i) if all adjacent C vertices are included. The same holds true for a cycle of D_1 vertices with its adjacent C vertices. Therefore, if there are none of these sets, $D_3 = \emptyset$ and $2|C| = 2|D_2| + |D_1|$. Please notice that if the size of one of these sets is larger than x, it will be possible to find a subset S with $|S| = x$ and $H(S) \leq -3$. This leads us to a set of type (ii). According to $|V_1(\pi)| < 3cut(\pi) = 3|C|$, it follows that $|D_2| + |D_1| + |E| < 2|C|$ and with $2|C| = 2|D_2| + |D_1|$ it follows that

$$|E| < |D_2| \ . \tag{3.4}$$

Special sets $S1$ with $H(S1) \geq -1$ and $S2$ with $H(S2) \geq -2$ from the C and D vertices are constructed as shown in Fig. 3.3 (right). The $S1$ sets consist of a path of D_1 vertices with a D_2 vertex on one end and an edge to an E vertex on the other end of the path. The path may also be of length 0, i. e. a $S1$ set may also consist of a single D_2 vertex which is adjacent to an E vertex. An $S2$ set consists of a path of D_1 vertices with both ends leading to E vertices. Adjacent C vertices are always included in the sets. C vertices may belong to two different $S1$ or $S2$ sets, but each D vertex belongs to exactly one $S1$ or $S2$ set. The number of edges that lead from $S1$ and $S2$ sets to E vertices are denoted with $E(S1)$ and $E(S2)$. Each $S1$ set has

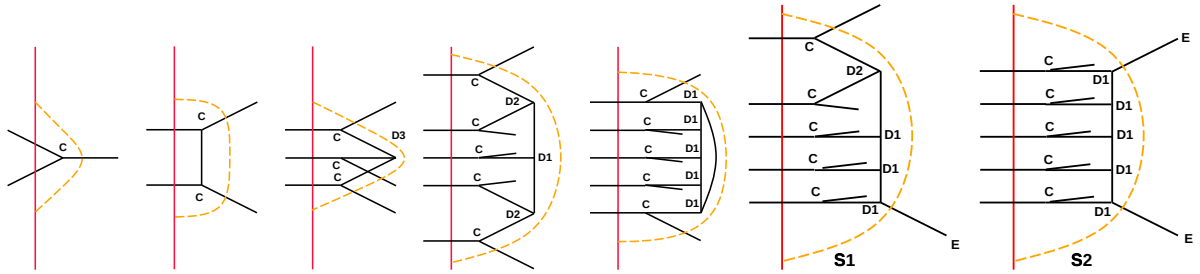


Fig. 3.3: Left five figures: Sets of type (i) of Lemma 2 and 4 with $H(S) \geq 0$. Right two figures: Initial $S1$ ($H(S1) \geq -1$) and $S2$ ($H(S2) \geq -2$) sets.

the property that it includes one D_2 vertex and has one edge that leads to an E vertex. This results in $|D_2| = |S1| = E(S1)$. In addition, an $S2$ set does not include any D_2 vertex and has 2 edges that lead to E vertices. This results in $2|S2| = E(S2)$. If one of the $S1$ or $S2$ sets are larger than x, it is easy to construct a subset S with $H(S) \geq -3$. Thus, type (ii) is fulfilled. Furthermore, if the size of one of the $S1$ or $S2$ sets is $\geq \frac{x}{2}$, it appears to be a set of type (iii). With $|D_2| = |S1| = E(S1)$ and equation (3.4) it follows that $|E| < E(S1)$. I. e. there is at least one E vertex which is adjacent to two $S1$ sets. Such an E vertex is merged with its adjacent $S1$ sets. This results in a larger $S1$ set. The possible merge combinations are displayed in Fig. 3.4.

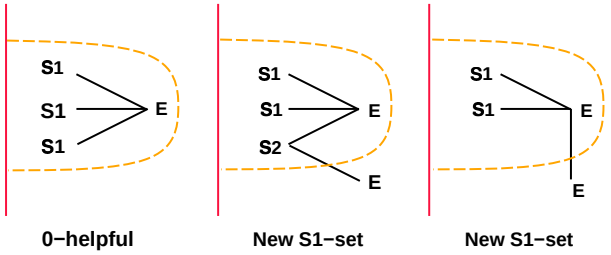


Fig. 3.4: The merging of an E vertex with 2 or more $S1$ sets.

If an E vertex is connected to three $S1$ sets, the union of them is 0-helpful. If an E vertex is connected to two $S1$ sets and either an $S2$ set or another E vertex, the union in Fig. 3.4 forms a new and larger $S1$ set. The inequality of $|E| < E(S1)$ still remains, because in both unions the number of $S1$ sets and the number of E vertices decreases by one. Therefore, there is a new E vertex adjacent to two $S1$ sets and the merge operation can be repeated. The process stops when the new set becomes large enough, i. e. when the size of the new $S1$ set is $\geq \frac{x}{2}$ and when it fulfills type (iii). ⋈

Lemma 3: *Let $G = (V, E)$ be a connected 3-regular graph and let π be a bisection of G. If $|V_1(\pi)| < 7 \cdot cut(\pi)$ and $0 < x < |V_1(\pi)|$, then $\exists S \subset V_1(\pi)$ with either*

(i) $|S| \leq x$ *and* $H(S) \geq -\frac{|S|}{3}$ *or*

(ii) $|S| = x$ *and* $H(S) \geq -\frac{|S|}{3} - 5$ *or*

(iii) $\frac{x}{2} \leq |S| \leq x$ *and* $H(S) \geq -\frac{|S|}{3} - 2$.

In other words, $\exists \hat{S} \subset V_1(\pi)$ *with* $|\hat{S}| = x$ *and* $H(\hat{S}) \geq -\frac{|\hat{S}|}{3} - 5 - 2\log(|\hat{S}|)$.

Proof: Like in Lemma 2, the vertices of $V_1(\pi)$ are classified according to their distance to the cut: $V_1(\pi) = C \uplus D \uplus E \uplus F$ with C vertices having a distance of 1 to the cut, i. e. they are incident to a cut edge, D vertices have a distance of 2, E vertices a distance of 3 and F vertices a distance of at least 4. The D and E vertices are further classified with respect to their number of adjacent vertices in C and D, i. e. $D = D_3 \uplus D_2 \uplus D_1$. Each D_x vertex is adjacent to x vertices in C, and $E = E_3 \uplus E_2 \uplus E_1$ and each E_x vertex is adjacent to x vertices in D. Altogether it is $V_1(\pi) = C \uplus D_3 \uplus D_2 \uplus D_1 \uplus E_3 \uplus E_2 \uplus E_1 \uplus F$.

First, several combinations can be excluded which lead to sets of type (i). Some examples are displayed in Fig. 3.5. Among these are C vertices incident to two cut edges and any set of two adjacent C vertices. If none of these sets are left, $cut = |C|$. A D_2 or D_3 vertex, together with the adjacent C vertices, forms a set of type (i). Therefore, if there are none of these sets, $D_3 = D_2 = \emptyset$ and $2|C| = |D_1|$. In addition, if a D_1 vertex is connected to two further D_1 vertices, the set of the three D_1 vertices and the three adjacent C vertices is also of type (i).

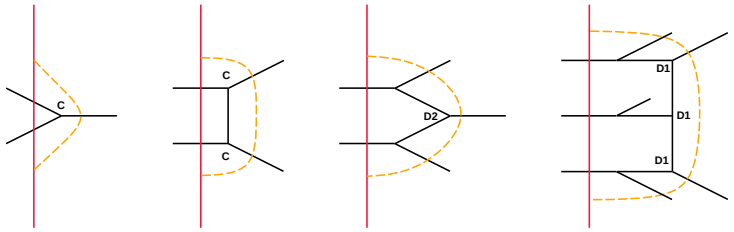


Fig. 3.5: Sets of type (i): $H(S) \geq -\frac{|S|}{3}$.

The D_1 vertices may further be classified in two sets $D_1 = D_A \uplus D_B$: D_A vertices are adjacent to one vertex each in C, D and E, whereas the D_B vertices are adjacent to one vertex in C and two in E. Now it is $V_1(\pi) = C \uplus D_A \uplus D_B \uplus E_3 \uplus E_2 \uplus E_1 \uplus F$. On the assumption that $|V_1(\pi)| < 7cut = 7|C|$ it follows that $|D_A| + |D_B| + |E_3| + |E_2| + |E_1| + |F| < 6|C|$ and with $2|C| = |D_A| + |D_B|$ it follows $|E_3| + |E_2| + |E_1| + |F| <$

$2|D_A| + 2|D_B|$. Furthermore, it is obvious that $|D_A| + 2|D_B| = 3|E_3| + 2|E_2| + |E_1|$. Consequently,

$$|F| < |D_A| + 2|E_3| + |E_2| \ . \tag{3.5}$$

New sets 2M, 4M and 6M are constructed from the D_A and E vertices as shown in Fig. 3.6. The 2M, 4M and 6M sets would be of type (i), i. e. $H(S) \geq -\frac{|S|}{3}$, if they had 2, 4 or 6 more vertices. In this case, $H(S_{2M}) \geq -\frac{|S_{2M}|+2}{3}$, $H(S_{4M}) \geq -\frac{|S_{4M}|+4}{3}$ and $H(S_{6M}) \geq -\frac{|S_{6M}|+6}{3}$.

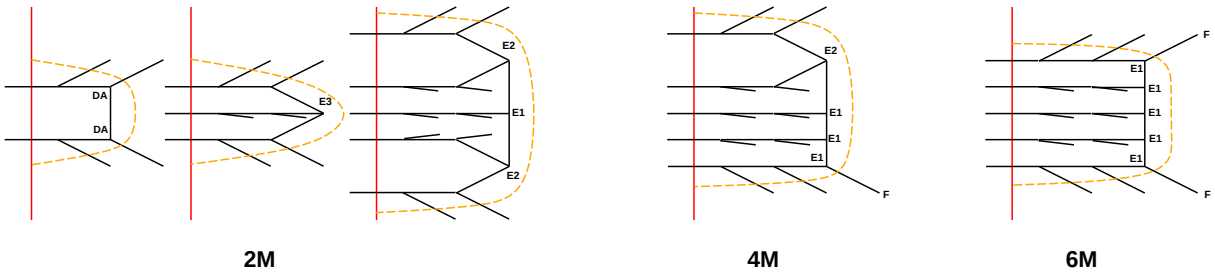


Fig. 3.6: zM sets: $H(zM) \geq -\frac{|S|+z}{3}$.

One example of a set of type 2M is a set of two connected D_A vertices and their adjacent C vertices. In addition, an E_3 vertices with the adjacent D and C vertices forms a $2M$ set, as well as two E_2 vertices which are either adjacent or connected via a path of E_1 vertices. 4M sets consist of a path of E_1 vertices with one end leading to an E_2 vertex and the other end leading to an F vertex. A 6M set consists of a path of E_1 vertices both ends of which lead to F vertices. The adjacent D and C are always included in the sets and, therefore, the C and D vertices may belong to several $2M$, $4M$ or $6M$ sets. Nevertheless, each E vertex is only a member of one of the sets. If the size of any of those sets is larger than x, it is possible to construct a subset S of size $|S| = x$ with $H(S) \geq -\frac{|S|}{3} - 5$, i. e. a set of type (ii). Additionally, if one of the sets has a size of at least $\frac{x}{2}$, type (iii) applies.

The 4M sets are similar to the S1 sets of Lemma 2, the only difference being that all vertices are one level further away from the cut (refer to Fig. 3.3), i. e. we consider a path of E vertices instead of D vertices. Furthermore, the 6M sets are similar to the S2 sets of Lemma 2, again moved away from the cut by one level. Our goal now is to unify the 2M sets with 4M and 6M sets until only 4M and 6M sets remain. Then, we will continue with the 4M and 6M sets in the same way as with the S1 and S2 sets in Lemma 2.

As long as there is a 2M set, such a set is unified with another 2M, 4M or 6M set if both sets share a pair of vertices consisting of a D vertex and an adjacent C vertex. This is illustrated in Fig. 3.7. The union might either be a 0M set fulfilling type (i) or another 2M or 4M set. There are two things left to show. Firstly, that the union

of a 2M set with a zM set, $z \in \{2, 4, 6\}$, results in a $(z-2)$M set. Secondly, that a 2M set always shares a pair of D and C vertices with another set and can be unified with that set.

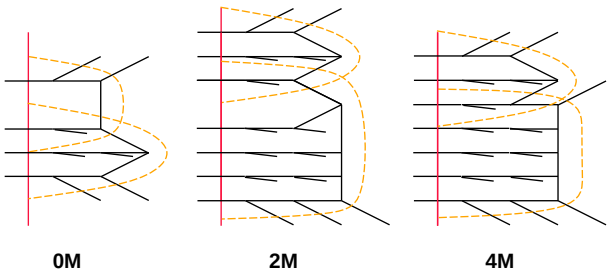


Fig. 3.7: Unions of a $2M$ set with another $2M$, $4M$ and $6M$ set, resulting in a $0M$, $2M$ and $4M$ set.

Let S_{2M} be a 2M set and S_{zM} be a zM set, $z \in \{2, 4, 6\}$, such that they share y pairs of adjacent D and C vertices. It is $|S_{2M} \cup S_{zM}| = |S_{2M}| + |S_{zM}| - 2y$. Furthermore, we know from Definition 5 that $H(S_{2M} \cup S_{zM}) = H(S_{2M}) + H(S_{zM}) - H(S_{2M} \cap S_{zM}) + 2|\{\{u, v\} \in E; u \in S_{2M} \backslash S_{zM}, v \in S_{zM} \backslash S_{2M}\}|$. With $H(S_{2M}) \geq -\frac{|S_{2M}|+2}{3}$, $H(S_{zM}) \geq -\frac{|S_{zM}|+z}{3}$ and $H(S_{2M} \cap S_{zM}) = -2y$ it is

$$H(S_{2M} \cup S_{zM}) \geq H(S_{2M}) + H(S_{zM}) + 2y \geq -\frac{|S_{2M} \cup S_{zM}| + 2 + z - 4y}{3} .$$

This fact shows that if they share at least two pairs of adjacent D and C vertices, the union is a 0M set fulfilling type (i). Otherwise, the 2M sets are unified with zM sets until no 2M sets remain.

It remains to show that all 2M sets (as initial set or as a result of a union) share at least one adjacent pair of D and C vertices with another set. Fig. 3.6 shows that initially all 2M and 4M sets share at least 2 of those pairs with other sets, whereas a 6M set may only share one pair. Clearly, if two sets are unified that share a pair and both sets have another pair that they share with other sets, the union has at least two pairs that it shares with other sets. In the worst case, a 2M set shares only two pairs and both other sets are 6M sets which themselves both only share one pair, namely with this 2M set. In this case, the 2M set can be unified with any of the 6M sets. The resulting 4M set shares a pair with the other 6M set and none of them share a pair with any other set. Thus, they are both blocked for unification with a 2M set. It follows that all 4M sets that share a pair with a 2M set always share another pair with some other set.

The important aspect now is the relation of the number of D_A pairs, the E_3 and E_2 vertices in the $4M$ sets compared to the number of edges that lead from $4M$ sets to vertices of F. The set of edges leading from a set S to F vertices are denoted with

$F(S)$ and the set of D_A pairs of a set S are denoted with $\bar{D}_A(S)$. For any initial 2M set S_{2M} holds $2|\bar{D}_A(S_{2M})| + 2|E_3(S_{2M})| + |E_2(S_{2M})| = 2$ and $|F(S_{2M})| = 0$, i. e.

$$2|\bar{D}_A(S_{2M})| + 2|E_3(S_{2M})| + |E_2(S_{2M})| = |F(S_{2M})| + 2 \; .$$

For any initial 4M set S_{4M} holds $|E_2(S_{4M})| = |F(S_{4M})|$ and $|\bar{D}_A(S_{4M})| = |E_3(S_{4M})| = 0$, i. e.

$$2|\bar{D}_A(S_{4M})| + 2|E_3(S_{4M})| + |E_2(S_{4M})| = |F(S_{4M})| \; .$$

Finally, for any 6M set S_{6M} holds $|F(S_{6M})| = 2$ and $|\bar{D}_A(S_{6M})| = |E_3(S_{6M})| = |E_2(S_{6M})| = 0$. Clearly, the values for $|\bar{D}_A(S)|$, $|E_3(S)|$, $E_2(S)|$ and $F(S)$ of a unified set $S_{2M} \cup S_{zM}$ are simply the sum of these values of the two unified sets. Therefore, the above equations also hold true for the unified sets.

Since there are only $4M$ and $6M$ sets left, the number of edges connecting $4M$ sets and F vertices are compared. There are no D_A, E_3 or E_2 vertices in the $6M$ sets. Therefore, there are $|D_A| + 2|E_3| + |E_2|$ edges that lead from the $4M$ sets to F vertices. It follows from equation 3.5 that there is at least one F vertex which is adjacent to two $4M$ sets. Such an F vertex is merged with its adjacent $4M$ sets. This results in a larger $4M$ set, identical to the merging process in Lemma 2. The possible merge combinations are displayed in Fig. 3.8. If the unified set is larger than x, the subset consisting of the F vertex and two or three adjacent 4M sets has a size between $\frac{x}{2}$ and x and fulfills type (iii).

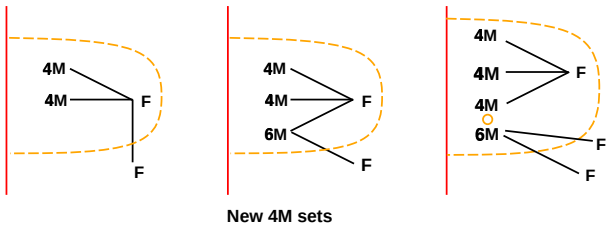


Fig. 3.8: The merging of an F vertex with $4M$ sets. The left and center merge operations reduce both, the number of edges leading from 4M sets to F vertices and the number of F vertices, by one. The merge operation on the right reduces both numbers by two. The 6M set on the right shares a pair of D and C vertices with one of the 4M sets.

Like in Lemma 2, the merge operations reduce both, the number of edges leading from 4M sets to remaining F vertices and the number of remaining F vertices, by the same amount. Thus, the number of edges leading from 4M sets to remaining F vertices is larger than the number of remaining F vertices, i. e. after a merging operation there is another F vertex which is adjacent to two $4M$ sets. The merge operations are performed until there is a set of a size of at least $\frac{x}{2}$. This set fulfills type (iii). ⋈

Lemma 4: *Let $G = (V, E)$ be a connected 3-regular graph and let π be a bisection of G. If $|V_1(\pi)| < \frac{10}{3} cut(\pi)$ and $0 < x < |V_1(\pi)|$, then $\exists S \subset V_1(\pi)$ with either*

(i) $|S| \leq x$ and $H(S) \geq -\frac{|S|}{5}$ or

(ii) $|S| = x$ and $H(S) \geq -\frac{|S|}{5} - 4$.

In other words, $\exists \hat{S} \subset V_1(\pi)$ with $|\hat{S}| = x$ and $H(\hat{S}) \geq -\frac{|S|}{5} - 4$.

Proof: The vertices of $V_1(\pi)$ are classified according to their distance to the cut like in Lemma 2: $V_1(\pi) = C \uplus D_3 \uplus D_2 \uplus D_1 \uplus E$. Classify the E vertices depending whether they are adjacent to a D_2 vertex or not with $E = E_{D2} \uplus E_X$ with E_{D2} vertices being adjacent to a D_2 vertex.

First, several combinations lead to sets of type (i). Some of them are discussed in Lemma 2 and are shown in Fig. 3.3. It follows that $D_3 = \emptyset$. Several additional sets of vertices lead to type (i) such as two adjacent D_2 and D_1 vertices with their adjacent C vertices or a path of more than 4 D_1 vertices as shown in Fig. 3.9. If an E_{D2} vertex is connected to a second D_2 vertex, it is easy to construct a set of type (i). Therefore, it holds $|D_2| = |E_{D2}|$.

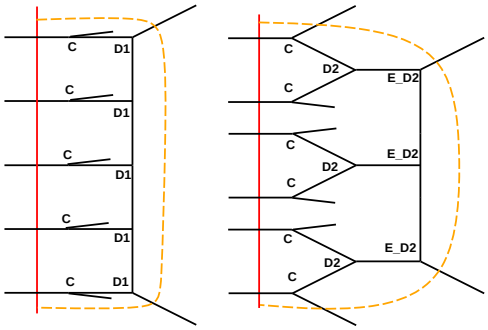


Fig. 3.9: Additional sets of type (i) for Lemma 4.

In the following, consider the connectivity between the D_1 and the E_{D2} vertices. For each D_1 vertex consider its adjacent C vertex (a set of 2 vertices). Consider for each E_{D2} vertex its adjacent D_2 vertex and its adjacent two C vertices (a set of 4 vertices). Each of these sets have a helpfulness of -2 and, therefore, each connected set of D_1 and E_{D2} vertices, with included adjacent D_2 and C vertices, has a helpfulness of at least -2. Thus, if it has a cardinality of at least 10, it forms a set of type (i). This is the case of e. g. a path of more than 4 connected D_1 vertices or a path of more than two connected E_{D2} vertices.

If none of these sets existed, there would be at least $|E_{D2}| + \frac{|D_1|}{2}$ edges from E_{D2}- and D_1 vertices to vertices from E_X. Thus, $\frac{|D_1|}{2} + E_{D2} \leq 3E_X = 3(|V_1(\pi)| - |C| -$

$|D_1| - |D_2| - |E_{D2}|$). With $|D_2| = |E_{D2}|$ and $2|D_2| + |D_1| = |C|$ it follows $|V_1(\pi)| \geq \frac{10}{3}|C| = \frac{10}{3}cut(\pi)$. Thus, there has to be such a set. If the size of these sets is larger than x, it is easy to construct a subset S with $|S| = x$ and $H(S) \geq -4$, fulfilling type (ii). ⋈

3.2.2 Upper Bound for 4-Regular Graphs

This section proves the following theorem about an upper bound on the bisection width of 4-regular graphs. The main advantage of this theorem is the fact that the proven bound is valid for any value of n. Section 3.4 shows that this bound is optimal for several small graphs.

Theorem 2: *The bisection width of any connected 4-regular graph $G = (V, E)$ is at most $\lfloor \frac{|V|}{2} \rfloor + 4$ if $|V| \equiv 0$ or $|V| \equiv 1$ $(mod\ 4)$, and it is at most $\lfloor \frac{|V|}{2} \rfloor + 5$ if $|V| \equiv 2$ or $|V| \equiv 3$ $(mod\ 4)$.*

Outline of the Proof: The proof consists of 4 phases. The first phase constructs a bisection with V_0 as the vertices of a spanning tree. Three further phases are used to enlarge V_0, without increasing the cut too much as shown in Fig. 3.10. Phase 2 and 3 do not increase the size of the cut, whereas phase 4 may increase the size of the cut by up to two edges. Throughout the phases, not only vertices from V_1 are moved to V_0, but also some vertices from V_0 are moved to V_1.

The vertices are categorized on either side of the cut according to their number of external edges as shown in Fig. 3.10. A vertices have 3 or 4, B vertices 2, C vertices 1 and D vertices have no external edges. Small numbers a, b, c and d specify the number of vertices of each type. We use this notation for both parts, but the respective part will be obvious from the context.

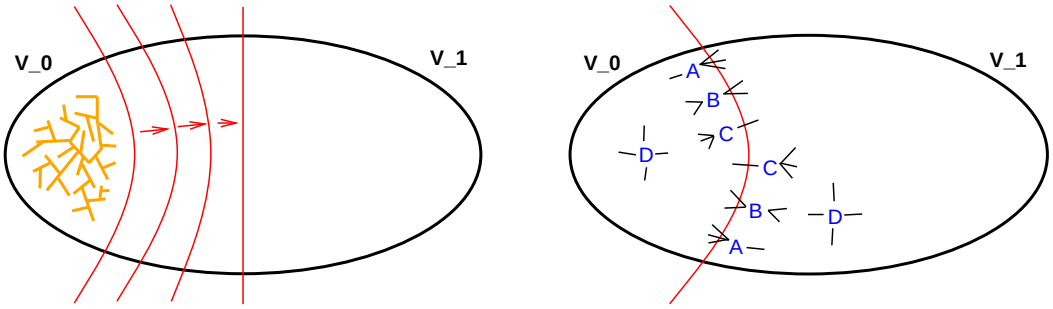


Fig. 3.10: Left: Phase 1 constructs a spanning tree. The bisection will be balanced after three additional balancing phases. Right: A, B, C and D vertices.

It is obvious that, as long as the bisection is not balanced, A and B vertices can be moved from V_1 to V_0 without any increase of the cut. If none of them are left, Lemma 5 shows that there are connected components of C vertices in V_1 which can be moved while increasing the cut by not more than 2. This will be compensated by Lemma 6 which moves a smaller set of vertices from V_0 to V_1 while decreasing the cut by at least 2. $\bowtie$

We now state the two lemma for moving sets of vertices in both direction of the bisection. Further on, we will use these lemma to prove theorem 2.

Lemma 5: *Let π be a bisection of a 4-regular graph $G = (V, E)$ in $V = V_0 \cup V_1$. If $|V_1| < \frac{5(x-1)+2}{4(x-1)} cut$, there is either an A or B vertex, a cycle of C vertices or a connected component of C vertices of size x in V_1.*

Proof: If there are no A or B vertices, no cycle of C vertices and no connected component of C vertices with size x, there are at least $\frac{(x-1)+2}{x-1} c$ edges from C to D vertices. This fact leads to $\frac{(x-1)+2}{x-1} c \leq 4d = 4(|V_1| - c)$. It results in $|V_1| \geq \frac{5(x-1)+2}{4(x-1)} cut$ which contradicts our findings above. $\bowtie$

The move of a connected component of C vertices increases the cut by 2. If there are no A and B vertices and no small cycles of C vertices (of a size up to $\lfloor \frac{n}{2} \rfloor - |V_0|$), Lemma 5 ensures the following for any x, and especially for $x = \lfloor \frac{n}{2} \rfloor - |V_0|$: if it is $|V_1| \leq \frac{5}{4} cut$, there is a connected component of C vertices of size x in V_1. Therefore, this set balances the bisection while increasing the cut by not more than 2.

Lemma 6: *Let π be a bisection of a 4-regular graph $G = (V, E)$ in $V = V_0 \cup V_1$.*

1. *If $|V_0| < \frac{5}{8} cut$, there is either an A vertex or two connected B vertices in V_0.*
2. *If $|V_0| < \frac{3}{4} cut$, there is either an A vertex or two connected B vertices or a C vertex with two adjacent B vertices in V_0.*

Proof: If there are no stated vertices in V_0,

1. the B vertices are only connected to the C and D vertices. i. e. there are at least $2b$ edges leading from B vertices to either C or D vertices. However, this number can not be higher than $3c + 4d$. Thus, $2b \leq 3c + 4d$, which leads to $cut = 2b + c \leq 4(c + d) = 4(2|V_0| - d - cut)$. This results in $5cut \leq 8|V_0|$.
2. each C vertex is connected to at most one B vertex. Thus, $2b \leq c + 4d$, which leads to $cut = 2b + c \leq 2(c + d) + 2d = 2(2|V_0| - d - cut) + 2d$. This results in $3cut \leq 4|V_0|$.

Each case leads to a contradiction. Thus, our statement above is proven. $\bowtie$

Lemma 6 is used to move sets of vertices from V_0 to V_1. The move of an A vertex, two connected B vertices or a C vertex together with two adjacent B vertices from V_0 to V_1 as described in Lemma 6 decreases the cut by at least 2. Lemma 5 and 6 are used in the following to prove theorem 2.

Proof of Theorem 2: The cut size of any bisection of a 4-regular graph, regardless whether it is balanced or not, is always even. This is due to the fact that the cut size can be expressed as $cut = \sum_{v \in V_0} deg(v) - 2 \cdot |\{\{u, v\}; u, v \in V_0\}|$. This value is even for all graphs in which the degree $deg(v)$ of any vertex $v \in V$ is even.

Let $bound := \lfloor \frac{|V|}{2} \rfloor + 4$ if $|V| \equiv 0$ or $|V| \equiv 1 \pmod 4$ and let $bound := \lfloor \frac{|V|}{2} \rfloor + 5$ if $|V| \equiv 2$ or $|V| \equiv 3 \pmod 4$. We will show that there is a bisection with $cut \leq bound$. Clearly, $bound$ is always even.

In the following we will always have $|V_0| \leq |V_1|$. Depending on the size of V_0, one of the following phases are applied to the bisection and increases the size of V_0. These phases are repeatedly applied until the bisection is balanced, i. e. until $|V_0| = \lfloor \frac{n}{2} \rfloor$.

Phase 1: $0 \leq |V_0| < \frac{1}{2}(bound - 2) - 1$

At the beginning it is $|V_0| = 0$. In this case choose any vertex $v \in V_1$ and move it to V_0. Otherwise, a spanning tree is constructed in the following way. If there is any A or B vertex in V_1, move it to V_0. Otherwise, move any C vertex from V_1 to V_0.

Notice that for V_0 being a spanning tree it holds $cut \leq 2|V_0| + 2$, i. e. if this phase applies it holds $cut \leq 2|V_0| + 2 < bound - 2$. The execution of this phase may increase the size of the cut if a C vertex is moved. However, this would increase the cut by at most 2 and it remains $cut \leq bound - 2$ after the execution of this phase (recall that both, the values of cut and $bound$, are always even numbers).

Overlap between phase 1 and 2: They overlap if $n - \frac{3}{2}(bound - 2) < \frac{1}{2}(bound - 2) - 1$. This is equivalent to $bound - 2 > \frac{n}{2} + \frac{1}{2}$, which is true for the definition of $bound$ above.

Phase 2: $n - \frac{3}{2}(bound - 2) < |V_0| < \frac{5}{8}(bound - 2) - \frac{7}{4}$

If there is any A or B vertex in V_1, move it to V_0. Otherwise, if $cut < bound - 2$, move any C vertex from V_1 to V_0.

It remains that $cut = bound - 2$. In this case move 3 connected C vertices from V_1 to V_0 (the existence is ensured by Lemma 5 with $x = 3$). This increases the size of the cut by two. Then move an A vertex or two adjacent B vertices from V_0 to V_1 (the existence is ensured by Lemma 6). This reduces the size of the cut by two. The exchange is illustrated in Fig. 3.11.

Overall, each execution of this phase increases the size of V_0 and the relation $cut \leq bound - 2$ is maintained.

Overlap between phase 2 and 3: They overlap if $n - \frac{17}{12}(bound-2) < \frac{5}{8}(bound-2) - \frac{7}{4}$. This is equivalent to $bound - 2 > \frac{24}{49}n + \frac{6}{7}$, which is true for the definition of *bound* above.

Phase 3: $n - \frac{17}{12}(bound-2) < |V_0| < \frac{3}{4}(bound-2) - \frac{5}{2}$

If there is any A or B vertex in V_1, move it to V_0. Otherwise, if $cut < bound - 2$, move any C vertex from V_1 to V_0.

It remains that $cut = bound - 2$. In this case move 4 connected C vertices from V_1 to V_0 (the existence is ensured by Lemma 5 with $x = 4$). This increases the size of the cut by two. Then move an A vertex, two adjacent B vertices or an C vertex with 2 adjacent B vertices from V_0 to V_1 (the existence is ensured by Lemma 6). This reduces the size of the cut by two. The exchange is illustrated in Fig. 3.11.

Overall, each execution of this phase increases the size of V_0 and the relation $cut \leq bound - 2$ is maintained.

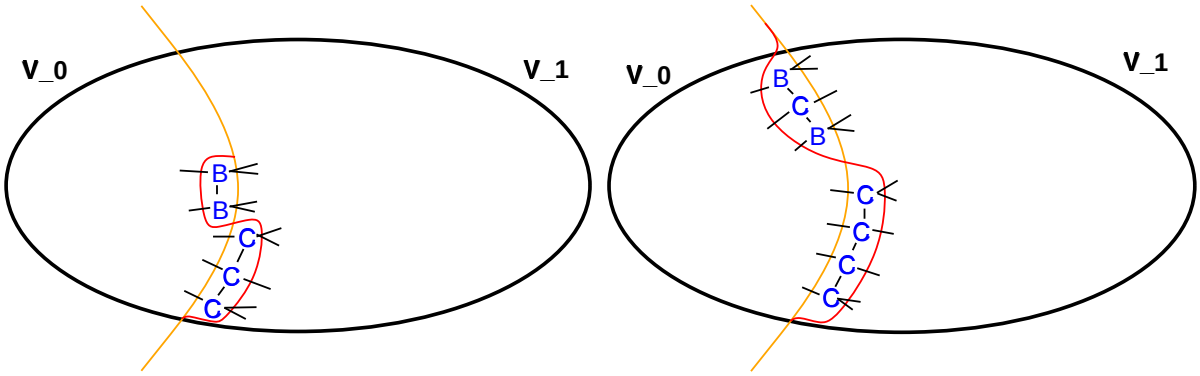


Fig. 3.11: Left: Phase 2, exchange 3 connected C vertices with 2 adjacent B vertices. Right: Phase 3, exchange 4 connected C vertices with a C and 2 adjacent B vertices.

Overlap between phase 3 and 4: They overlap if $n - \frac{5}{4}(bound-2) \leq \frac{3}{4}(bound-2) - \frac{5}{2}$. This is equivalent to $bound - 2 \geq \frac{n}{2} + \frac{5}{4}$, which is true for the definition of *bound* above.

Phase 4: $n - \frac{5}{4}(bound-2) \leq |V_0| < \lfloor \frac{n}{2} \rfloor$

If there is any A or B vertex or a small cycle of C vertices (not larger than $\lfloor \frac{n}{2} \rfloor - |V_0|$) in V_1, move these to V_0. Otherwise, if $cut < bound - 2$, move any C vertex from V_1 to V_0.

It remains that $cut = bound - 2$. In this case move a connected component of C vertices of size $\lfloor \frac{n}{2} \rfloor - |V_0|$ from V_1 to V_0 (the existence is ensured by Lemma 5 with $x = \lfloor \frac{n}{2} \rfloor - |V_0|$).

Overall, each execution of this phase increases the size of V_0. If the cut is increased by moving a C vertex, this is only done in the case when $cut < bound-2$ and the relation $cut \leq bound - 2$ is maintained. The move of the connected component of C vertices increases the cut by two. However, it cannot be increased any further because the bisection is already balanced. Thus, a relation of $cut \leq bound$ is guaranteed.

Please notice that the vertices from V_1 to V_0 are moved first. Thus, the cut size and the size of V_0 changes slightly, leading to the small additive constants on the right hand side of the ranges in the phases. ⋈

3.3 Lower Bounds

Unlike the techniques for upper bounds, techniques for lower bounds do not explicitly construct bisections with low cuts. Lower bounds are rather compared to existing upper bounds. If there is a large gap between these two values, there might be a potential for improvement of the upper bounds. There are two main goals for lower bounds. Firstly, they can be used to show a lower bound on the bisection width of a given graph. Secondly, they can be used to show that all graphs of a scalable graph class have a lower bound on their bisection width. Therefore, it is tried to develop lower bounds which for scalable graph classes are as close as possible to the generalized upper bounds of the previous section.

Lower bounds can also be used to speed up Branch & Bound strategies for calculating the bisection width of moderate-sized graphs. Here, at each branching step some vertices are fixed to belong to one of the two parts. If the lower bounds on this fixed scenario are high enough, no further branching needs to be performed from this branching step.

Lower bounds on the bisection width can be derived from algebraic graph theory by relating the bisection problem to an eigenvalue problem (Section 3.3.1). Leighton [Lei92] proposes a lower bound of the bisection width by calculating a routing scheme between all pairs of vertices such that the congestion is minimized (Section 3.3.2). In Section 3.3.3 we compare these two approaches.

Parts of the work in this section were published in [BEM+00].

3.3.1 Spectral Lower Bound

3.3.1.1 Traditional Spectral Bound

Lower bounds on the bisection width can be derived from algebraic graph theory by relating the bisection problem to an eigenvalue problem. For a graph G, the $n \times n$ *Laplacian matrix* $L(G) = \{l_{v,w}\}$ is defined as

$$l_{v,w} = \begin{cases} \deg(v), & \text{if } v = w \\ -1, & \text{if } v \neq w \text{ and } \{v,w\} \in E \\ 0, & \text{otherwise .} \end{cases}$$

It is known that all the eigenvalues of $L(G)$ are nonnegative. We denote them by $\lambda_1, \ldots, \lambda_n$ with $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ and pairwise perpendicular eigenvectors $y_1, y_2, \ldots y_n$. The eigenvector to the eigenvalue 0 is $y_1 = \frac{1}{\sqrt{n}}(1,1,1,\ldots 1)$. Its multiplicity is equal to the number of connected components in the graph (see e. g. [PSL90]). We do only consider connected graphs, so $\lambda_2 > 0$.

Let $x = (x_1, x_2, ..., x_n)$ be a non-zero vector. From the Courant-Fisher principle it follows

$$\lambda_2(G) = \min_{x \perp 1} \left\{ \frac{x^t L x}{\|x\|^2} \right\} = \min_{x \perp 1} \left\{ \frac{\sum_{\{u,v\} \in E} (x_u - x_v)^2}{\sum_{v \in V} {x_v}^2} \right\} . \tag{3.6}$$

Furthermore, the minimum in (3.6) is attained iff x is an eigenvector to λ_2. Using the Lagrange identity $n \sum_{i=1}^n x_i^2 - \left(\sum_{i=1}^n x_i\right)^2 = \sum_{\{u,v\} \in V^2} (x_u - x_v)^2$, we can rewrite (3.6) (cf. [Fie75]) as

$$\lambda_2(G) = \min_{x \neq \text{const}} \left\{ \frac{\sum_{\{u,v\} \in E} (x_u - x_v)^2}{\sum_{\{u,v\} \in V^2} (x_u - x_v)^2} n \right\} . \tag{3.7}$$

The minimum runs over all vectors that are not collinear to $(1, 1, \ldots, 1)$. A simple lower bound on $bw(G)$ can be derived by applying (3.7) to the x-tuple defined by

$$x_v = \begin{cases} a, & \text{if } v \in V_0 \\ b, & \text{if } v \in V_1 \end{cases} \text{ with some } a \neq b . \tag{3.8}$$

This leads to $\lambda_2 \leq n \frac{bw(a-b)^2}{\frac{n}{2} \cdot \frac{n}{2} \cdot (a-b)^2} = \frac{4 \cdot bw}{n}$, i. e. to the well-known lower bound of

$$bw \geq \frac{\lambda_2 \cdot n}{4} . \tag{3.9}$$

This lower bound is strict for some classes of graphs. In the following theorem we specify the situation when this bound is attainable.

Theorem 3: *Let $G = (V, E)$ be a graph and λ_2 be the second smallest eigenvalue of $L(G)$. Then the following statements are equivalent.*

a. $bw(G) = \frac{\lambda_2 n}{4}$;

b. there is an eigenvector corresponding to λ_2 which has only -1 and $+1$ entries;

c. in any optimal bisection $V = V_0 \cup V_1$ any vertex is incident to exactly $\frac{\lambda_2}{2}$ cut edges.

Proof: We prove that statement (a) implies (b), (b) implies (c), and (c) implies (a).

Assume that $bw(G) = \frac{\lambda_2 n}{4}$ and let $V = V_0 \cup V_1$ be an optimal bisection. Consider the vector $(x_1, \ldots, x_n)$ with $x_u = 1$ for $u \in V_0$, and $x_u = -1$ for $u \in V_1$. Now (3.7) implies that $\lambda_2 \leq \frac{\sum_{\{u,v\} \in E} (x_u - x_v)^2}{\sum_{\{u,v\} \in V^2} (x_u - x_v)^2} n = \frac{4bw(G)}{n} = \lambda_2$. Thus, x is an eigenvector to λ_2.

Now assume that there is an eigenvector x that corresponds to λ_2 specified above. Let $u \in V_0$ (a similar argument works well for $u \in V_1$), and let a_u (respectively b_u) denote the number of vertices of V_0 (V_1 respectively) incident to u. The u-th entry of $L(G)x$ equals $\deg(u) - a_u + b_u$. Since $L(G)x = \lambda_2 x$ and the u-th entry of x is 1, $\deg(u) - a_u + b_u = \lambda_2$. This equality and $a_u + b_u = \deg(u)$ imply $b_u = \frac{\lambda_2}{2}$.

Finally, let $V = V_0 \cup V_1$ be an optimal bisection, and assume that any vertex is incident to $\frac{\lambda_2}{2}$ cut edges. Since $|V_0| = |V_1| = n/2$, the size of the cut equals $\frac{n}{2} \cdot \frac{\lambda_2}{2}$. On the other hand the size of the cut equals $bw(G)$ because the bisection is optimal. $\bowtie$

There are plenty of graphs for which $bw(G) = \frac{\lambda_2 n}{4}$ holds. Some examples of such graphs are the complete graphs with $\lambda_2 = n$ and bisection width $bw = n^2/4$, the complete bipartite graphs with $\lambda_2 = \frac{n}{2}$ and $bw = \frac{n}{8}$, the hypercubes with $\lambda_2 = 2$ and $bw = n/2$ or the Petersen graph ($n = 10$) with $\lambda_2 = 2$ and $bw = 5$.

A detail discussion about the spectrum of graphs is to be found in [CDS95] and on spectral graph theory in [Chu97].

3.3.1.2 Improved Spectral Lower Bounds with Levels

As it is shown in Theorem 3, the traditional spectral lower bound of equation (3.9) is only tight, if any vertex of G is incident to a cut edge. Now, we consider the case when this condition is not satisfied. We show that for such graphs this lower bound can be improved significantly. We consider the level structure of a bisection. Each level of vertices consists of all vertices that have the same distance to the cut.

Definition 6 (Level Structure): *Let π be a bisection of a graph $G = (V, E)$ with $V = V_0 \cup V_1$ and a cut size of σ. Denote the subsets V_0^i of V_0 as follows.*

Let V_0^1 be the set of all vertices in V_0 that are incident to a cut edge. Let V_0^i be the sets of all vertices in V_0 with a distance of $i-1$ to a vertex of V_0^i. Denote with V_1^i the respective sets on side V_1. Furthermore, denote with E_0^i, $i \geq 1$, the edge sets which connect vertices between sets V_0^i and V_0^{i+1}. Denote with E_1^i the respective sets in part V_1.

Let $g : \mathbb{N} \to \mathbb{N}$ be a function. We denote with $LS(g, \sigma)$ the class of graphs which have a bisection π with a cut size of σ and a level structure of the kind that $|E_0^i| \leq \sigma g(i)$ and $|E_1^i| \leq \sigma g(i)$ for all $i \geq 1$.

In a certain sense the level structure can be regarded as a double cone. The function g represents its width. In the case of grids and paths it holds $g(i) = 1$ for any i. As example, the levels of an 8×8 square grid and a median cut is shown in Fig. 3.12(left).

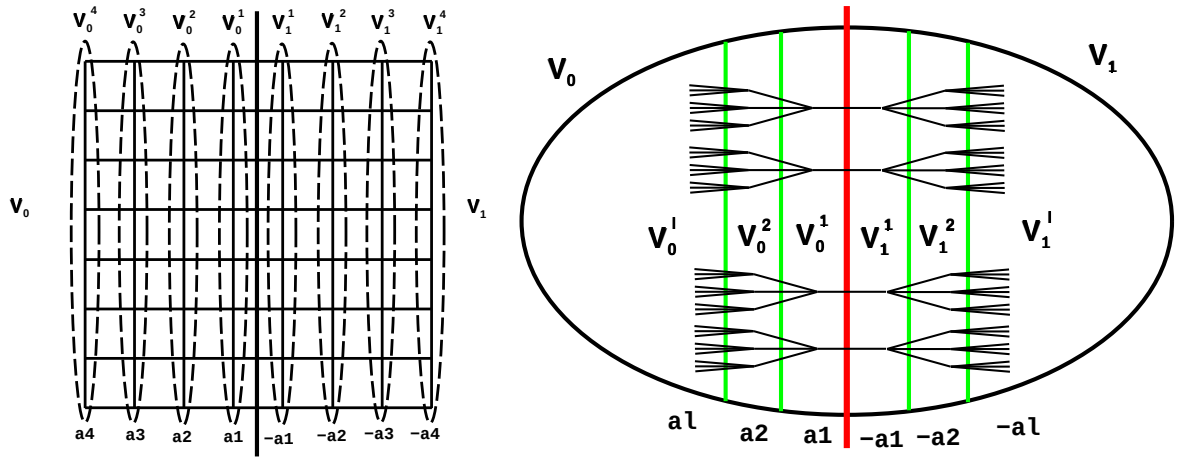


Fig. 3.12: Left: levels of an 8×8 square grid and a median cut. Right: The number of edges connecting consecutive levels may increase with the function $g(i)$. The assignment of the a_i-values is defined in equation (3.10) of Lemma 7.

For graphs with a cone of larger width (like in Fig. 3.12(right)), information can flow more easily to the distant vertices. Therefore, we can view this as a global expansion property. We show in the following that for a fixed bisection width the spectral lower bound of the second smallest eigenvalue increases with the width of the cone.

In Lemma 7 we bound λ_2 from above by some expression that does only depend on the grow function $g(i)$. Indeed, the proof of the lemma shows that the worst possible case occurs, if $|V_i| = g(i-1)$ holds for any level i. We use the level structure in order to derive a new upper bound of λ_2. This result will be used in Theorem 4 below.

Lemma 7: *Let $G \in LS(g,\sigma)$ and let $l \in \mathbb{N}$ be a number such that $n \geq 2\sigma \sum_{i=1}^{l-1} g(i-1)$. It is*

$$\lambda_2(G) \leq \min_{1=a_1 \leq a_2 \leq \cdots \leq a_l} \frac{2\sigma a_1^2 + \sigma \sum_{i=1}^{l-1} g(i)(a_i - a_{i+1})^2}{\sigma \sum_{i=1}^{l-1} a_i^2 g(i-1) + a_l^2 \left(\frac{n}{2} - \sigma \sum_{i=1}^{l-1} g(i-1)\right)} .$$

Proof: In the proof we use the equations (3.6) and (3.7). We need a non-zero vector x with $x \perp 1$ for equation (3.6), whereas we do only need a non-constant x for (3.7). First, we use equation (3.7) such that all vertices of the same level have the identical value in the entries of the eigenvector. Choose a vector x with entries x_v for $v \in V$ defined by

$$x_v = \begin{cases} a_i, & \text{if } v \in V_0^i \text{ with } i < l \\ a_l, & \text{if } v \in V_0^i \text{ with } i \geq l \\ -a_i, & \text{if } v \in V_1^i \text{ with } i < l \\ -a_l, & \text{if } v \in V_1^i \text{ with } i \geq l \end{cases} \tag{3.10}$$

with $1 = a_1 \leq a_2 \leq \cdots \leq a_l$. The upper bound of equation (3.7) now only depends on the a_i. Let $A(x) := \sum_{\{u,v\} \in E} (x_u - x_v)^2$ be the numerator of (3.6) and (3.7). It is

$$\begin{aligned} A(x) &= 4\sigma a_1^2 + \sum_{i=1}^{l-1} |E_0^i|(a_i - a_{i+1})^2 + \sum_{i=1}^{l-1} |E_1^i|(a_i - a_{i+1})^2 \\ &\leq 4\sigma a_1^2 + 2\sigma \sum_{i=1}^{l-1} g(i)(a_i - a_{i+1})^2 . \end{aligned} \tag{3.11}$$

Now we estimate the denominator $B(x) := \sum_{\{u,v\} \in V^2} (x_u - x_v)^2$ of (3.7). It holds $|V_0^j| \leq |E_0^{j-1}| \leq \sigma g(i-1)$. Assume now that $|V_0^j| < \sigma g(i-1)$ for some j, $1 \leq j < l$. Denote by $\tilde{x}$ the corresponding vector for the level structure obtained by moving a vertex from V_0^l to V_0^j. We have

$$\begin{aligned} B(\tilde{x}) - B(x) &= -\sum_{i=1}^{l-1} (a_i - a_l)^2 |V_0^i| + \sum_{i=1}^{l} (a_i - a_j)^2 |V_0^i| - (a_l - a_j)^2 \\ &\quad + \sum_{i=1}^{l} ((a_i + a_j)^2 - (a_i + a_l)^2) |V_1^i| \\ &= \sum_{i=1}^{l-1} ((a_i - a_j)^2 - (a_i - a_l)^2 - (a_l - a_j)^2) |V_0^i| - (a_l - a_j)^2 \\ &\quad + \sum_{i=1}^{l} ((a_i + a_j)^2 - (a_i + a_l)^2 + (a_l - a_j)^2) |V_1^i| \\ &= \sum_{i=1}^{l-1} 2(a_l - a_j)(a_i - a_l) |V_0^i| - (a_l - a_j)^2 + \sum_{i=1}^{l} 2(a_i + a_j)(a_j - a_l) |V_1^i| \\ &\leq 0 . \end{aligned}$$

In order to establish the first equality, we use $|V_0^l| = \sum_{i=1}^{l} |V_0^i| - \sum_{i=1}^{l-1} |V_0^i|$. The inequality follows from $a_l \geq a_i$ for $0 \leq i \leq l$. Therefore, $B(x)$ does not increase because of this transformation and the minimum of $B(x)$ (for a fixed V_1 part) is attained if $|V_0^i| = \sigma g(i-1)$ for $1 \leq i < l$. A similar argument provides $|V_1^i| = \sigma g(i-1)$ for $1 \leq i < l$ is sufficient for $B(x)$ to attain its minimum. Therefore, in this case $|V_0^i| = |V_1^i|$ for $1 \leq i < l$. Thus, $x \perp 1$ and the denominators of (3.6) and (3.7) (multiplied with $\frac{1}{n}$) are equal. These arguments imply

$$\sum_{v \in V} {x_v}^2 \geq 2\sigma \sum_{i=1}^{l-1} a_i^2 g(i-1) + 2a_l^2 \left(\frac{n}{2} - \sigma \sum_{i=1}^{l-1} g(i-1) \right) . \tag{3.12}$$

The lemma follows by substituting (3.11) and (3.12) into (3.6). ⋈

Lemma 7 shows that the level structure of the bisection gives an upper bound on λ_2. We use this result in the following theorem to derive some new relations between λ_2 and the cut size σ of a bisection. These new relations depend on the growth of the function g.

Theorem 4: *Let $G \in LS(g, \sigma)$ be a graph with a growing function g such that $A := 1 + 2\sum_{i=2}^{\infty} \frac{1}{g(i-1)} < \infty$. There is a function $\gamma : \mathbb{R}^+ \to \mathbb{R}$ with $\gamma(x) \to 0$ for $x \to \infty$ such that $\sigma \geq A\frac{\lambda_2(G)n}{4}(1 + \gamma(\frac{n}{\sigma}))$.*

Proof: Let $l \in \mathbb{N}$ be defined by $2\sigma \sum_{i=1}^{l-1} g(i-1) < n \leq 2\sigma \sum_{i=1}^{l} g(i-1)$. This, in particular, implies that $\frac{n}{\sigma} \to \infty$ as $l \to \infty$. We apply lemma 7 with $a_i = 1 + 2\sum_{j=2}^{i} \frac{1}{g(j-1)}$. Since $2a_1^2 + \sum_{i=1}^{l-1} g(i)(a_i - a_{i+1})^2 = 2 + 4\sum_{i=1}^{l-1} \frac{1}{g(i)} = 2a_l$ it is

$$\lambda_2 \leq \frac{2\sigma a_l}{\sigma \sum\limits_{i=1}^{l-1} a_i^2 g(i-1) + a_l^2 \left(\frac{n}{2} - \sigma \sum\limits_{i=1}^{l-1} g(i-1) \right)} . \tag{3.13}$$

In the following, we use some functions γ_i of the property described in the theorem. Set $r_i := A - a_i$, $1 \leq i \leq l$, with $r_i > 0$. Equation (3.13) implies

$$\begin{aligned}
\lambda_2 &\leq \frac{2\sigma(A - r_l)}{\sigma \sum\limits_{i=1}^{l-1} (A^2 - 2Ar_i + r_i^2) g(i-1) + (A^2 - 2Ar_l + r_l^2) \left(\frac{n}{2} - \sigma \sum\limits_{i=1}^{l-1} g(i-1) \right)} \\
&= \frac{2\sigma(A - r_l)}{\sigma A^2 (1 + \gamma_1(\frac{n}{\sigma})) \sum\limits_{i=1}^{l-1} g(i-1) + A^2 (1 + \gamma_2(\frac{n}{\sigma})) \left(\frac{n}{2} - \sigma \sum\limits_{i=1}^{l-1} g(i-1) \right)} \\
&\leq \frac{4\sigma}{A \cdot n(1 + \gamma(\frac{n}{\sigma}))} .
\end{aligned}$$

⋈

Notice that for $g(i) = (i+1)^\alpha$ with $\alpha > 1$ we have $A = 1 + 2\sum_{i=2}^{\infty} \frac{1}{g(i-1)} < \infty$. For example, $A = 1 + \frac{\pi^2}{3}$ for $\alpha = 2$ and $A = 1 + \frac{\pi^4}{45}$ for $\alpha = 4$.

If G is a graph of maximum degree d, $G \in LS(g, bw(G))$ with $g(i) = (d-1)^i$, because $\max\{|V_0^1|, |V_1^1|\} \leq bw(d-1)^i$. Thus, theorem 4 implies the following corollary.

Corollary 1: *For a graph G of a d-regular graph class with $bw(G) = o(n)$ it holds*

$$bw(G) \geq A\frac{\lambda_2 n}{4}(1 - o(1)) = \frac{d}{d-2}\frac{\lambda_2 n}{4}(1 - o(1)) \; .$$

It shows that for 3-regular graphs the traditional spectral bound of equation (3.9) can be improved by a factor of 3 and for 4-regular graphs by a factor of 2. In [BEM+00] it is shown that the bound of this corollary is asymptotically tight for Double Root trees.

In the following, we derive bounds for sufficiently large 3- and 4-regular graphs. These bounds also hold for graphs with $bw = \Theta(n)$, i. e. when the number of levels not necessarily increases with the number of vertices. One example are the d-regular Ramanujan graphs. For these graphs it holds $\lambda_2 \geq d - 2\sqrt{d-1}$. Notice that the latter value is asymptotically the largest possible for d-regular graphs, since $\lambda_2 \leq d - 2\sqrt{d-1} + \frac{4\sqrt{d-1}}{\log_{d-1}(n) - O(1)}$ [Nil91] (see also [Alo86, LPS88]). There are known construction of infinite families of d-regular Ramanujan graphs for any d of the form $d = p^k + 1$, where p is any prime number, and k is an arbitrary positive integer (see [Chi92, LPS88, Mar88, Mor94]).

The classical spectral bound of equation (3.9) implies a lower bound for the bisection width of the form $0.042n$ for 3-regular Ramanujan graphs and one of the form $0.133n$ for 4-regular Ramanujan graphs. In the following we improve these bounds. We first show lower bounds for 3- and 4-regular graphs with small λ_2, i. e. for sufficiently large graphs.

Theorem 5: *The bisection width of any*

1. *4-regular graph with $\lambda_2 \leq 2$ is at least* $\min\{\frac{n}{2}, \frac{5-\lambda_2}{7-(\lambda_2-1)^2} \cdot \frac{\lambda_2 n}{2}\}$.
2. *3-regular graph with $\lambda_2 \leq 2$ is at least* $\min\{\frac{n}{2}, \frac{4-\lambda_2}{4-\lambda_2^2+2\lambda_2} \cdot \frac{\lambda_2 n}{2}\}$.
3. *3-regular graph with $\lambda_2 \leq \frac{5-\sqrt{17}}{2} \approx 0.44$ is at least* $\min\{\frac{n}{6}, \frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2} \cdot \frac{\lambda_2 n}{2}\}$.

Proof: As mentioned above, for any d-regular graph G it holds $G \in LS(g, bw(G))$ with $g(i) = (d-1)^i$. Solving the equation of lemma 7 for $bw(G)$ we get

$$bw(G) \geq \max_{1=a_1 \leq \cdots \leq a_l} \frac{a_l^2 \lambda_2 \frac{n}{2}}{2a_1^2 + \sum_{i=1}^{l-1}\left[(d-1)^i(a_i - a_{i+1})^2 + \lambda_2(d-1)^{i-1}(a_l^2 - a_i^2)\right]} \tag{3.14}$$

with optimal values

$$a_1 = 1\,, \qquad a_2 = \frac{1+d-\lambda_2}{d-1}\,, \qquad a_{j+1} = a_j\frac{d-\lambda_2}{d-1} - a_{j-1}\frac{1}{d-1}\,.$$

We only have to make sure that $1 = a_1 \leq a_2 \leq \ldots \leq a_l$ provided that $n \geq 2bw\sum_{i=1}^{l-1} g(i-1)$.

4-regular, $\lambda_2 \leq 2$ and $bw \leq \frac{n}{2}$: We apply equation (3.14) with $d = 4$ and $l = 2$. We have $1 = a_1 \leq a_2 = \frac{5-\lambda_2}{3}$ and $n \geq 2bw = 2bw\sum_{i=1}^{l-1} g(i-1)$. We get $bw \geq \frac{(5-\lambda_2)\lambda_2}{7-(\lambda_2-1)^2} \cdot \frac{n}{2}$ in this case.

3-regular, $\lambda_2 \leq 2$ and $bw \leq \frac{n}{2}$: We apply equation (3.14) with $d = 3$ and $l = 2$. We have $1 = a_1 \leq a_2 = \frac{4-\lambda_2}{2}$ and $n \geq 2bw = 2bw\sum_{i=1}^{l-1} g(i-1)$. We get $bw \geq \frac{(4-\lambda_2)\lambda_2}{4-\lambda_2^2+2\lambda_2} \cdot \frac{n}{2}$ in this case.

3-regular, $\lambda_2 \leq \frac{5-\sqrt{17}}{2}$ and $bw \leq \frac{n}{6}$: We apply equation (3.14) with $d = 3$ and $l = 3$. We have $1 = a_1 \leq a_2 = \frac{4-\lambda_2}{2} \leq a_3 = \frac{\lambda_2^2-7\lambda_2+10}{4}$ and $n \geq 6bw = 2bw\sum_{i=1}^{l-1} g(i-1)$. We get $bw \geq \frac{(\lambda_2^2-7\lambda_2+10)\lambda_2}{3\lambda_2^3-17\lambda_2^2+10\lambda_2+8} \cdot \frac{n}{2}$ in this case.

⋈

From $\lambda_2 \leq d - 2\sqrt{d-1} + O(\log_{d-1}(n))^{-1}$ [Nil91, Alo86, LPS88] follows that for $d = 3$ it is $\lambda_2 \leq 0.17 + O(\log_2(n))^{-1}$ and for $d = 4$ it is $\lambda_2 \leq 0.54 + O(\log_3(n))^{-1}$. Thus, the conditions $\lambda_2 \leq 2$ and $\lambda_2 \leq \frac{5-\sqrt{17}}{2}$ of cases 1, 2 and 3 of Theorem 5 hold true for sufficiently large graphs.

An upper bound of $(0.4+\delta)n$ for the bisection width of 4-regular graphs with $n \geq n(\delta)$ is shown in [MP00]. In Section 3.2.1 we have shown an upper bound of $0.1982n+O(\log(n))$ on the bisection width of 3-regular graphs. Thus, in the first two cases of Theorem 5 the bounds in the min-lists depending on λ_2 are dominating for sufficiently large graphs. If an upper bound of $\frac{n}{6}$ for the bisection width of all large 3-regular graphs could be shown, only the lower bounds depending on λ_2 would remain for all cases of Theorem 5.

We now compare the bounds of Theorem 5 with the traditional spectral bound $\frac{\lambda_2 n}{4}$ of equation (3.9). To do so we compare the factor in front of $\frac{\lambda_2 n}{2}$, i. e. the traditional bound has a factor of $\frac{1}{2}$, case 1 has a factor of $\frac{5-\lambda_2}{7-(\lambda_2-1)^2}$, case 2 one of $\frac{4-\lambda_2}{4-\lambda_2^2+2\lambda_2}$ and case 3 one of $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2}$. These functions are plotted in Fig. 3.13. It can be observed that the new bounds are at least as good as the traditional bound. As discussed above, it is $\lambda_2 \leq 0.17+O(\log_2(n))^{-1}$ for $d = 3$ and $\lambda_2 \leq 0.54+O(\log_3(n))^{-1}$ for $d = 4$. The asymptotic bounds 0.17 and 0.54 are illustrated in Fig. 3.13, too.

For the first two cases of Theorem 5 (2 levels are used in the proof), the new bounds are identical to the traditional bound for $\lambda_2 \to 2$. But for $\lambda_2 \to 0$ the first case has a

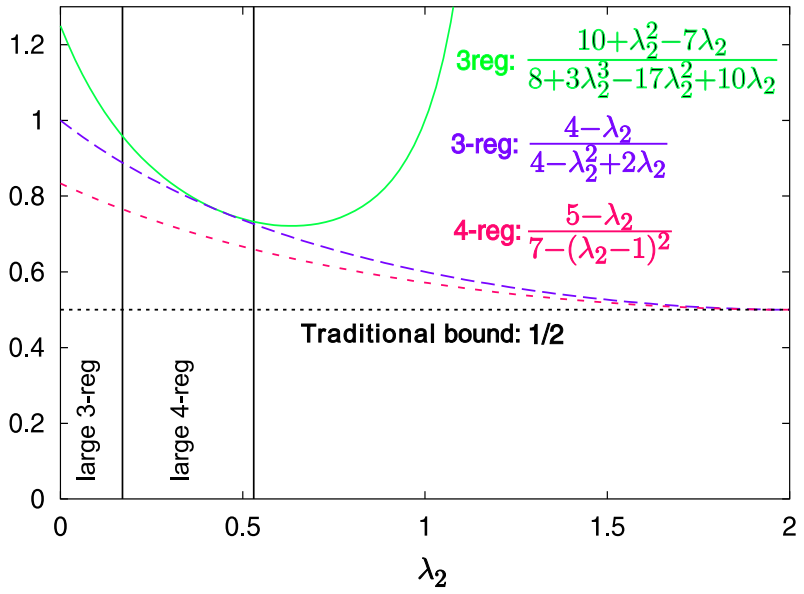


Fig. 3.13: Comparison of th new lower bounds of Theorem 5. The bounds are compared by the factor in front of $\frac{\lambda_2 n}{2}$. The traditional bound has a factor of $\frac{1}{2}$. The factor $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2}$ is only valid in the range $0 \leq \lambda_2 \leq \frac{5-\sqrt{17}}{2} \approx 0.44$ and the other two factors are valid in the range $0 \leq \lambda_2 \leq 2$.

factor of $\frac{5}{6}$ and the second case a factor of 1, i. e. the bound of the first case is by a factor of $\frac{5}{3}$ and the bound of the second case is by a factor of 2 higher than the traditional bound. In the third case (3 levels are used in the proof), the new bound is always higher than the traditional bound. For $\lambda_2 \to 0$ the new bound is by a factor of $\frac{5}{2}$ higher than the traditional bound and for $\lambda_2 \to \frac{5-\sqrt{17}}{2}$ it is higher by a factor of $\frac{2\sqrt{17}+6}{3\sqrt{17}-3} \approx 1.52$.

Theorem 5 can be used to derive stronger lower bounds on the bisection width of Ramanujan graphs. It is a standard task to show that the function $\frac{10+\lambda_2^2-7\lambda_2}{8+3\lambda_2^3-17\lambda_2^2+10\lambda_2}\lambda_2$ is monotone increasing in the interval $[0, \frac{5-\sqrt{17}}{2}]$ and that the function $\frac{5-\lambda_2}{7-(\lambda_2-1)^2}\lambda_2$ is monotone increasing in the interval $[0, 2]$. Since $\lambda_2 \geq 3 - 2\sqrt{2} \approx 0.171573$ for 3-regular Ramanujan graphs and $\lambda_2 \geq 4 - 2\sqrt{3} \approx 0.535898$ for 4-regular Ramanujan graphs, theorem 5 leads to the following corollary.

Corollary 2: *The bisection width of any sufficiently large 3-regular Ramanujan graph (such that* $\lambda_2 \leq \frac{5-\sqrt{17}}{2}$*) is at least* $0.082n$*. The bisection width of any sufficiently large 4-regular Ramanujan graph (such that* $\lambda_2 \leq 2$*) is at least* $0.176n$*.*

This improves upon the previously known lower bounds of $0.042n$ for 3-regular and of $0.133n$ for 4-regular Ramanujan graphs which can be derived by using the traditional spectral bound.

It is left to say that the results of Theorem 5 can not generally be improved by using more levels. For 4-regular graphs we only used 2 levels. We cannot take $l \geq 3$, because for $l = 3$ lemma 7 only holds for $n > 2bw \sum_{i=1}^{2} g(i-1) = 8bw$, which is not generally fulfilled for 4-regular graphs (Ramanujan graphs are a counterexample, even for large graphs). With the same arguments we cannot take $l \geq 4$ for 3-regular graphs. For $l = 4$ lemma 7 only holds for $n > 2bw \sum_{i=1}^{3} g(i-1) = 14bw$, which is false for 3-regular Ramanujan graphs.

3.3.2 Congestion of All-to-All-Routing or Multi-Commodity Flow

A different strategy for lower bounds on the bisection width is based on the edge-congestion of an all-to-all routing or a multi-commodity flow on the graph [Lei92]. It is used to show lower bounds for several different graphs, sometimes even matching the known upper bounds.

In the case of all-to-all routing a routing scheme between all ordered pairs of vertices in the graph has to be calculated. It should be done in a way such that the congestion, which is the maximum number of crossing paths along any edge of the graph, is as small as possible. Consider an unknown minimal bisection. At least $n \cdot \frac{n}{2}$ paths have to cross the cut. However, a congestion of *cong* ensures that there are at most *cong* crossing paths for each edge of the cut. It holds $bw \cdot cong \geq n \cdot \frac{n}{2}$. Therefore,

$$bw \geq \frac{n^2}{2 \cdot cong} \; . \tag{3.15}$$

Please notice that there are two paths between any pair of vertices, one for each direction.

So far, the routing from one vertex to another is only performed along one path. It is possible to use a multi-commodity flow. First, each vertex has a commodity with a weight of n and a feasible flow distributes each commodity evenly among the vertices. Thus, it is a generalized version of the all-to-all routing approach. Again, the congestion is the maximum flow over any edge. It is fairly obvious to see that the same equation as (3.15) can be derived.

Obviously, a high and tight lower bound can only be achieved with a small congestion. The tightest lower bounds can be achieved with a shortest path routing with an equal congestion on all edges of the graph. Unfortunately, this 'optimal' case does not exist for arbitrary graphs. In Section 3.3.3 we present an example with an 'optimal' routing and compare it to the lower bounds achieved with the spectral lower bounds of the previous section.

3.3.3 A Comparison of Lower Bounds

In this section, we compare the spectral lower bound of equation (3.9) with the congestion lower bound of equation (3.15). For several graph classes, the bisection width, the second lowest eigenvalue and the lowest congestion are known. Tab. 3.3 lists some graph classes together with their bisection width and both lower bounds.

Tab. 3.3: Comparison of Lower Bounds on the Bisection Width (n even).

Graph G	B.W.	λ_2	$\frac{\lambda_2 n}{4}$	$cong$	$\frac{n^2}{2 \cdot cong}$
K_n	$\frac{n^2}{4}$	n	$\frac{n^2}{4}$	2	$\frac{n^2}{4}$
$K_{\frac{n}{2},\frac{n}{2}}$	$\frac{n^2}{8}$	$\frac{n}{2}$	$\frac{n^2}{8}$	$6 - \frac{8}{n}$	$\frac{n^3}{12n-16}$
S_n	$\frac{n}{2}$	1	$\frac{n}{4}$	$2(n-1)$	$\frac{n^2}{4(n-1)}$
Q_k	$\frac{n}{2}$	2	$\frac{n}{2}$	n	$\frac{n}{2}$
P_n	1	$2 - 2\cos(\frac{\pi}{n}) \approx \frac{\pi^2}{n^2}$	$\approx \frac{\pi^2}{4n}$	$\frac{n^2}{2}$	1
C_n	2	$2 - 2\cos(\frac{2\pi}{n}) \approx \frac{4\pi^2}{n^2}$	$\approx \frac{\pi^2}{n}$	$\frac{n^2}{4}$	2
$G_{\sqrt{n}\times\sqrt{n}}$	$\sqrt{n}$	$2 - 2\cos(\frac{\pi}{\sqrt{n}}) \approx \frac{\pi^2}{n}$	$\approx \frac{\pi^2}{4}$	$\frac{n\sqrt{n}}{2}$	$\sqrt{n}$
$T_{\sqrt{n}\times\sqrt{n}}$	$2\sqrt{n}$	$2 - 2\cos(\frac{2\pi}{\sqrt{n}}) \approx \frac{4\pi^2}{n}$	$\approx \pi^2$	$\frac{n\sqrt{n}}{4}$	$2\sqrt{n}$
Pet	5	2	5	10	5
Ho-Si	65	5	62.5	26	48.07

With n we denote the cardinality of the graphs. K_n and $K_{n/2,n/2}$ are the complete and the complete bipartite graphs. P_n, C_n and S_n represent the path, the cycle and the star graph. Furthermore, HYP_k is the Hypercube of dimension k. $G_{\sqrt{n}\times\sqrt{n}}$ and $T_{\sqrt{n}\times\sqrt{n}}$ are the the 2-dimensional grids and tori. 'Pet' is the Petersen graph with 10 vertices and regular vertex degree of 3 and 'Ho-Si' is the Hoffmann-Singleton graph with 50 vertices and a regular vertex degree of 7. Both graphs have a girth of 5 and a diameter of 2. They both belong to the class of *Moore* graphs (ref. Sec. 3.4.3).

It can be observed that there are examples of the case that the spectral lower bound outperforms the congestion lower bound and vice versa. In theorem 3 we analyzed the cases when the spectral lower bound is strict. The best possible situation for the congestion bound occurs when all routings or flows are along shortest paths and the congestion is equal on all edges. In this case, the minimal possible congestion can be achieved.

In the remaining part of this section we analyze the congestion of *Moore* graphs (ref. Section 3.4.3) and, especially, for the so called *Hoffman-Singleton Graph* [HS60]. Besides the complete graphs, there are only two more Moore graphs, namely the Petersen and the Hoffman-Singleton graphs. The Moore graphs have a diameter of $\frac{g-1}{2}$ with g being the girth. Thus, a shortest path between any two vertices is unique. Both, the Petersen and the Hoffman-Singleton graph have a girth of 5 and their degrees are 3 and 7 respectively. With their number 10 and 50 of vertices they are the smallest graphs with respect to their

degree and girth. In addition, the Moore graphs are edge-isomorphic, i. e. the analyses of a single edge can be generalized to all other edges.

We analyze the congestion of the unique shortest path routing in the Moore graphs between vertices u and v as it is illustrated in Fig. 3.14.

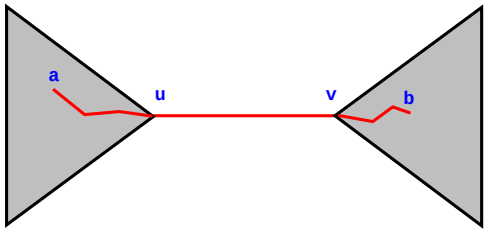


Fig. 3.14: Edge-congestion of a Moore Graph.

Consider a shortest path between two vertices a and b which leads along edge $\{u, v\}$. Notice that the distance between a and u and between v and b cannot be more than $\frac{g-1}{2} - 1$. Furthermore, u is the root of a complete $(d-1)$-ary tree of height $\frac{g-1}{2} - 1$ on the left. v is the root of a complete $(d-1)$-ary tree of height $\frac{g-1}{2} - 1$ on the right. There are $(d-1)^{dist}$ vertices left of u with distance $dist$ and each of them is a starting point of shortest paths to all vertices of the $(d-1)$-ary tree with height $\frac{g-1}{2} - 1 - dist$ to the right of v. Thus

$$\begin{aligned} cong_{Moore}(d,g) &= 2 \cdot \sum_{dist(a,u)=0}^{\frac{g-1}{2}-1} (d-1)^{dist} \cdot \frac{(d-1)^{(\frac{g-1}{2}-1-dist)+1} - 1}{d-2} \\ &= 2 \cdot \frac{\frac{g-1}{2}(d-2)(d-1)^{\frac{g-1}{2}} - (d-1)^{\frac{g-1}{2}} + 1}{(d-2)^2} \,. \end{aligned}$$

Notice that all edges of the Moore graph have the identical congestion and, because it is a shortest path routing, it is the smallest possible congestion.

We can compare the spectral and congestion lower bound for Moore graphs. The complete graphs have a bisection width of $\frac{n^2}{4}$, and it is identical to both lower bounds, because the second smallest eigenvalue is n and the congestion is 2. The Petersen graph has a bisection width of 5 and, again, the eigenvalue of 2 and the congestion of 10. Both lead to lower bounds that match the bisection width. These values differ for the Hoffman-Singleton graph. The congestion of the shortest path mapping is 26, leading to

$$bw(\text{Hoffman-Singleton}) \geq \frac{n^2}{2 \cdot cong_{Moore}(7,2)} = \frac{50^2}{2 \cdot 26} = 48.07 \,.$$

The second lowest eigenvalue is 5 and leads to

$$bw(\text{Hoffman-Singleton}) \geq \frac{\lambda_2 \cdot n}{4} = \frac{5 \cdot 50}{4} = 62.50 \,.$$

It shows that even with an 'optimal' shortest path routing, the congestion bound can be weaker than the spectral bound. Both of the lower bounds do not match the bisection width of the Hoffman-Singleton graph of 65 which was calculated using the enumeration scheme of the PARTY library.

All that remains to be mentioned is the fact that for the star graphs the congestion of a shortest path all-to-all routing the congestion is equal on all edges and, again, the congestion lower bound does not match the bisection width. Furthermore, for the complete bipartite graphs there is a multi-commodity-flow along shortest paths with an identical lowest possible congestion on all edges. And, again, the congestion lower bound is smaller than the spectral bound and the bisection width. Unlike the case of the Moore graphs and the star graphs, the minimal congestion is a fractional value here. The reason for this gap between the congestion bound and the bisection width is the fact that there are routing paths which cross the cut of the minimal bisection more often than once.

3.4 Regular Graphs with high Bisection Width

The calculation of the bisection width of regular graphs is **NP**-complete. However, it can be calculated for small graphs by the use of efficient enumerating schemes.

3.4.1 Small Graphs

We do some experiments, in order to answer the question of the highest bisection width of any d-regular graphs with n vertices. We generate all non-isomorphic connected d-regular graphs with n vertices, calculate the bisection width of all of them and report the highest one. This leads us to the question of how many non-isomorphic connected d-regular graphs with n vertices do exist? Up to now there is no answer to this question. We used the code by Markus Meringer [Mer99] to generate the graphs for many combinations of d and n. The number of existing graphs explodes with increasing n, as listed in Tab. 3.4. Obviously, there are no d-regular graphs with n vertices for $d \geq n$ or d and n odd.

The bisection width of small graphs can be calculated by an efficient enumeration scheme included in PARTY (Chapter 6). We calculated the bisection width of all graphs of the respective combinations. The number in brackets in Tab. 3.4 shows the number of graphs that have the highest bisection width. Tab. 3.5 displays the highest bisection width.

There are two problems involved with the results of these tables. Firstly, derive a generalized upper bound function which matches the highest bisection width. Secondly, give an explicit construction of a graph with highest bisection width. For the boxed combinations of Tab. 3.5 the problems have already been solved. The bisection width of

Tab. 3.4: Number of d-regular graphs of size n

n \ d	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
4	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	2	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	(1)																		
7	-	2	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
		(2)																	
8	5	6	3	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	(4)	(1)	(1)																
9	-	16	-	4	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-
		(4)		(1)															
10	19	59	60	21	5	1	1	-	-	-	-	-	-	-	-	-	-	-	-
	(10)	(19)	(1)	(6)	(3)														
11	-	265	-	266	-	6	-	1	-	-	-	-	-	-	-	-	-	-	-
		(83)		(1)		(1)													
12	85	1544	7848	7849	1547	94	9	1	1	-	-	-	-	-	-	-	-	-	-
	(10)	(1)	(918)	(1)	(8)	(1)	(1)												
13	-	10778	-	367860	-	10786	-	10	-	1	-	-	-	-	-	-	-	-	-
		(31)		(2)		(110)		(3)											
14	509	88168	3459383	21609300	21609301	3459386	88193	540	13	1	1	-	-	-	-	-	-	-	-
	(3)	(1356)	(275718)	(1077)	(1)	(32)	(4)	(6)	(1)										
15	-	805491	-	1470293675	-	1470293676	-	805579	-	17	-	1	-	-	-	-	-	-	-
		(27124)		(7)		(3)		(1)		(1)									
16	4060	8037418	2585136675						8037796	4207	21	1	1	-	-	-	-	-	-
	(1042)	(1)	(6)						(1)	(1)	(3)								
17	-	86221634	-			-			-		-	25	-	1	-	-	-	-	-
		(235)										(1)							
18	41301	985870522										42110	33	1	1	-	-	-	-
	(1342)	(112816)										(6)	(1)						
19	-		-		-		-		-		-		-	39	-	1	-	-	-
														(3)					
20	510489													516344	49	1	1	-	-
	(149)													(1)	(1)				
21	-		-		-		-		-		-		-		-	60	-	1	-
																(1)			
22	7319447																73	1	1
	(3)																(3)		
23	-		-		-		-		-		-		-		-		-	88	-
																		(1)	
24	117940535																		110
	(1643395)																		(1)

complete graphs ($d = n - 1$) is $\frac{dn+n}{4}$ (n even) or $\frac{dn+d}{4}$ (n uneven). *Complete p-partite* graphs with even n and d have a bisection width of $\frac{dn}{4}$. It can be proven that there are no other graphs of the same combination that have a higher bisection width.

Apart from these exact values, the results of Section 3.2 display upper bounds. Some of them match the values of Tab. 3.5.

There are some relations between the highest bisection width (hbw) of any combination. One of them is the fact that the sum of the bisection widths of two graphs with the same number of vertices and with degrees d_1 and d_2 cannot be more than the highest bisection width of a graph with degree d_1+d_2, i. e. $hbw(n, d_1)+hbw(n, d_2) \leq hbw(n, d_1+d_2)$. This shows that the highest bisection width is growing with the degree d (for a fixed n).

Tab. 3.5: Highest bisection width of d-regular graphs with size n

n	d																						
	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
4	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	5	6	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7	-	6	-	12	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
8	4	8	10	12	16	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9	-	8	-	14	-	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
10	5	8	13	14	17	20	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	-	8	-	16	-	22	-	30	-	-	-	-	-	-	-	-	-	-	-	-	-
12	6	10	12	18	20	24	28	30	36	-	-	-	-	-	-	-	-	-	-	-	-
13	-	10	-	18	-	24	-	32	-	42	-	-	-	-	-	-	-	-	-	-	-
14	7	10	13	18	25	26	31	34	39	42	49	-	-	-	-	-	-	-	-	-	-
15	-	10	-	20	-	28	-	38	-	46	-	56	-	-	-	-	-	-	-	-	-
16	6	12	16			32			44	48	52	56	64	-	-	-	-	-	-	-	-
17	-	12	-		-		-		-		-	60	-	72	-	-	-	-	-	-	-
18	7	12								54		62	69	72	81	-	-	-	-	-	-
19	-		-		-		-		-		-		-	76	-	90	-	-	-	-	-
20	8							50						80	86	90	100	-	-	-	-
21	-		-		-		-		-		-		-		-	96	-	110	-	-	-
22	9																105	110	121	-	-
23	-		-		-		-		-		-		-		-		-	116	-	132	-
24	8									72				96		108		120	128	132	144

3.4.2 Bisection Width of small 3- and 4-Regular Graphs

In this section, we take a closer look at the results for 3- and 4-regular graphs. Tab. 3.5 only shows results for 3-regular graphs up to 24 vertices and 4-regular graphs up to 18 vertices, because it is extremely time-consuming to generate all graphs for more vertices. Nevertheless, Tab. 3.6 shows results for larger graphs which were generated with the help of the tool by Markus Meringer [Mer99]. Not all graphs of each combination were generated, but only those with a high girth (cf. Section 3.4.3), especially the so-called *cages*. Therefore, we can not ensure that these graphs have the highest bisection width of their combination, but at least they have a high bisection width with respect to the upper bounds.

Tab. 3.6 shows that up to the size of 30 vertices for 4-regular graphs it is possible to construct a graph with a bisection width that matches the upper bound of Section 3.2. In the other cases, there are either other graphs of the same combinations with higher bisection widths, or the theoretical bound is too weak. Another interesting observation

Tab. 3.6: Highest bisection width of 3-regular (left) and 4-regular (right) graphs. Values in brackets indicate an upper and lower bound on the bisection width of the corresponding cage.

$\|V\|$	highest bw	Cage
4	4	(3,3)
6	5	(3,4)
8	4	
10	5	(3,5)
12	6	
14	7	(3,6)
16	6	
18	7	
20	8	
22	9	
24	8	(3,7)
30	≥ 9	(3,8)
58	≥ 15	(3,9)
70	≥ 17	(3,10)
112	$\geq (24 \geq bw \geq 21)$	(3,11)
126	$\geq (29 \geq bw \geq 25)$	(3,12)
272	$\geq (52 \geq bw \geq 44)$	(3,13)
406	$\geq (77 \geq bw \geq 60)$	(3,14)
620	$\geq (110 \geq bw \geq 86)$	(3,15)

$\|V\|$	highest bw	upper bound	Cage
5	6	$6 = \frac{\|V\|^2-1}{4}$	(4,3)
6	6	$6 = \frac{\|V\|}{2} + 3$	
7	6		
8	8	$8 = \frac{\|V\|}{2} + 4$	(4,4)
9	8		
10	8	$8 = \frac{\|V\|}{2} + 3$	
11	8		
12	10	$10 = \frac{\|V\|}{2} + 4$	
13	10		
14	10	$10 = \frac{\|V\|}{2} + 3$	
15	10		
16	12	$12 = \frac{\|V\|}{2} + 4$	
17	12		
18	12	$12 = \frac{\|V\|}{2} + 3$	
19	≥ 12		(4,5)
20	≥ 14	$14 = \frac{\|V\|}{2} + 4$	
21	≥ 14		
22	≥ 14	$14 = \frac{\|V\|}{2} + 3$	
23	≥ 14		
24	≥ 14	$16 = \frac{\|V\|}{2} + 4$	
25	≥ 16		
26	≥ 16	$16 = \frac{\|V\|}{2} + 3$	(4,6)
28	≥ 14	$18 = \frac{\|V\|}{2} + 4$	
30	≥ 18	$18 = \frac{\|V\|}{2} + 3$	
32	≥ 16	$20 = \frac{\|V\|}{2} + 4$	
80	≥ 36	$44 = \frac{\|V\|}{2} + 4$	(4,8)

is the structure of the graphs in table 3.6. Not only are the complete and the complete p-partite graphs included in the table, but also the 3- and 4-regular cages which are discussed in the following section. As examples, Fig. 3.15 presents drawings of the $(4, 5)$-cage, $(4, 6)$-cage and $(4, 8)$-cage. Furthermore, Tab. 3.6 displays that some small cages have a high bisection width, but it remains open to show the same for large cages. It is left to show general connections between the girth and the bisection width.

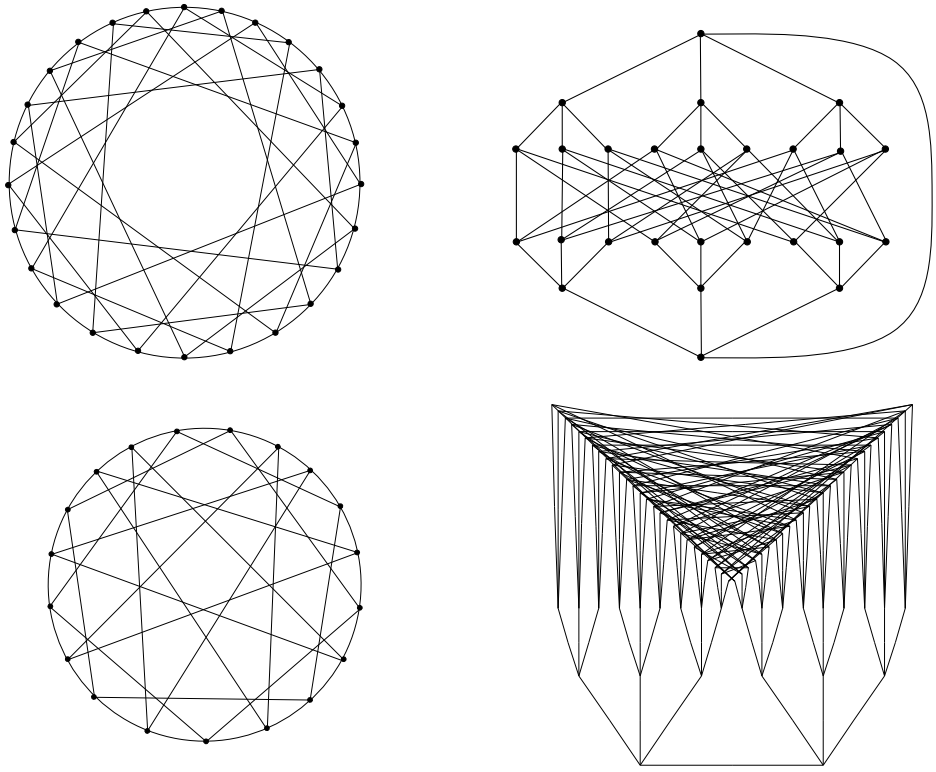


Fig. 3.15: Top: the 4-regular $(4, 6)$-cage with 26 vertices has $bw = 16$, drawn as cycle and as double-tree. Bottom left: the 4-regular $(4, 5)$-cage with 19 vertices has $bw = 12$. Bottom right: the 4-regular $(4, 8)$-cage with 80 vertices has $bw = 36$.

3.4.3 Bisection Width of Cages

As we have seen above, cages have a very high bisection width. We take a closer look at the definition and existence of cages in this section. Fur further information consult e. g. [Big93, HS93, Won82].

Definition 7 (girth/cage): *Let the* **girth** $g(G)$ *of a graph* G *be the length of a shortest cycle in* G*, and let a* (d, g)**-cage** *be a smallest* d*-regular graph with girth* g*.*

A simple lower bound for the size of a cage can be derived by the *Moore Bound*

$$M(d, D) := \frac{d(d-1)^D - 2}{d-2} .$$

The Moore bound has originally been used to derive an upper bound on the size of a regular graph with a given diameter. The currently largest graphs for a given degree and a given diameter are listed at the WWW-page [Com].

It is obvious to see that $n \leq M(d, D)$ holds for a d-regular graph ($d \geq 3$) with diameter D. Furthermore, it is easy to see that for a d-regular graph ($d \geq 3$) with odd girth g it holds $n \geq M(d, \frac{g-1}{2})$. A d-regular graph ($d \geq 3$) with diameter D, odd girth g and $n = M(d, D) = M(d, \frac{g-1}{2})$ is called a *Moore Graph* or *minimum cage*. Unfortunately, there are only few of these extremal graphs. Namely,

- $M(d, 3)$: clique with $d + 1$ vertices.
- $M(d, 5)$: only $d = 3$ (Petersen graph), $d = 7$ (Hoffman-Singleton graph) and possibly $d = 57$.

In the case of an even girth g, a simple lower bound for the size of the cages is $n \geq \frac{2(d-1)^{\frac{g}{2}}-2}{d-2}$. Graphs with $n = \frac{2(d-1)^{\frac{g}{2}}-2}{d-2}$ are called *Generalized Polygons* and sometimes *minimum cages*. There are some examples of this type of graphs, namely

- $(d, 4)$ minimum cage: complete bipartite graph with $2d$ vertices.
- (d, g) minimum cage: only exist for $g = 6, 8, 12$ and $d - 1$ prime power.

As example, the $(4, 6)$-cage and (4,8)-cage are presented in Fig. 3.15.

Although cages are discussed in the literature for some decades, the sizes of (d, g)-cages are only known for very small cages or for special values of d and g (refer to the WWW-page [Roy]). The currently known sizes of cages are shown in Tab. 3.7.

We constructed several cages and calculated the bisection width (or at least a bisection with a low cut for large cages). The results are listed in Tab. 3.7. The bisection widths of cages are very high. Small cages are always among the graphs with the highest bisection width of their combination. So far it is unknown if it holds true for all cages. We have calculated the bisection width of the cages by enumerating schemes of PARTY. However, it would be interesting to have an explicit construction of bisections of cages with a cut equal to their bisection width.

Tab. 3.7: Sizes of d-regular cages with girth g. Moore graphs and generalized polygons are bold. Numbers in ()-brackets indicate the number of different cages. Numbers in []-brackets indicate their bisection width or an upper bound on it.

g	d												
	3	4	5	6	7	8	9	10	11	12	13	14	15
3	**4** [4]	**5** [6]	**6** [9]	**7** [12]	**8** [16]	**9** [20]	**10** [25]	**11** [30]	**12** [36]	**13** [42]	**14** [49]	**15** [56]	**16** [64]
4	**6** [5]	**8** [8]	**10** [13]	**12** [18]	**14** [25]	**16** [32]	**18** [41]	**20** [50]	**22** [61]	**24** [72]	**26** [85]	**28** [98]	**30** [113]
5	**10** [5]	=19(1) [12]	=30(4) [25(4)]	=40(1) [40]	**50** [65]	≤ 94	≤ 118	≤ 155	≤ 202	≤ 253		≤ 406	≤ 504
6	**14** [7]	**26** [16]	**42** [33]	**62** [60]	=90(1) [≤ 107]	**114** [≤ 158]	**146** [≤ 179]	**182** [≤ 250]		**266** [≤ 594]		**366** [≤ 976]	
7	=24(1) [8]	≤ 76	≤ 272	≤ 504	≤ 1332	≤ 1640							
8	**30** [9]	**80** [36]	**170**	**312**		**800**	**1170**	**1640**		**2928**		**4760**	
9	=58(18) [13(1),15(17)]	≤ 504											
10	=70(3) [17(3)]												
11	112(1+) [≤ 24]												
12	**126** [≤ 29]	**728**	**2730**	**7812**		**39216**	**74898**	**132860**		**354312**		**804468**	

4. HEURISTICS FOR GRAPH PARTITIONING

In the previous chapter we dealt with upper and lower bounds on the bisection width. The results are interesting for certain applications. Nevertheless, the majority of applications requires a solution for the graph partitioning problem. In this context, an optimal solution is not always necessary. More precisely, there is a trade-off between the time spent for the calculation of the solution and the quality of the solution. Therefore, heuristics are used to calculate a partition of the graph in a reasonable period of time.

Many graph partitioning heuristics were developed by researches of different scientific fields such as engineering, mathematics or computer science. There are some survey publication about graph partitioning methods for certain application areas. Recent directions in Net-List partitioning which can be modeled as hypergraph partitioning problem are described in [AK95]. An overview of common graph partitioning heuristics for high performance scientific simulations is published in [SKK00].

There are several graph partitioning paradigms which involve other heuristics as a sub-problem. One of these paradigms is the multilevel strategy. It is known to be very efficient. It is discussed in the following chapter. In this chapter, we focus on global and local graph partitioning heuristics. Both approaches are needed for the multilevel strategy.

There are several possible classifications of the partitioning heuristics. A major characteristic distinguishes between *global* and *local* methods. Global methods are sometimes called *construction heuristics*, because they use the graph description as input and generate a balanced partition. Local methods are called *improvement heuristics*. They use the graph and a balanced partition as input and aim to improve the partition. Fig. 4.1 presents the combination of global and local methods. A global heuristic is used to construct a partition π_1. The main task of a global heuristic is to force the partition to be balanced, while aiming to cut through sparse areas of the graph. In the second step, a local heuristic can be applied to construct a partition π_2 of the partition π_1. It is the main task of the local heuristic to refine the partition locally, in order to obtain a lower cut. The resulting partition has to be balanced, too. Thus, the local heuristic has to determine two equally sized sets of vertices in both parts of the cut. The exchange of these sets results in the balanced partition π_2. The same or a different local heuristic can be applied on a partition π_i to construct a further partition π_{i+1}.

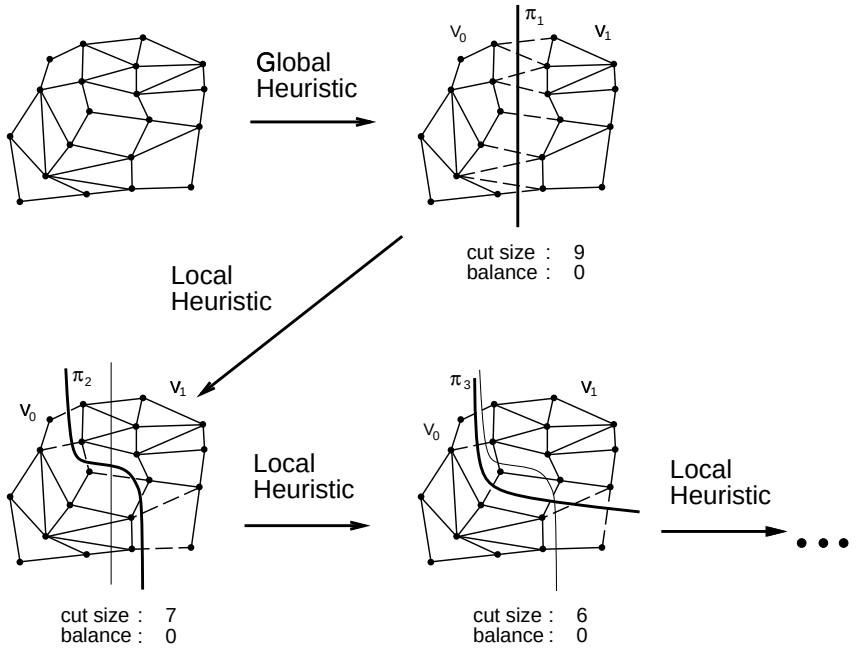


Fig. 4.1: Combination of global and local graph partitioning heuristics.

The combination of global and local heuristics leads us to the following question: 'Which heuristic should we focus on?' There are simple as well as highly complicated heuristics for both steps.

4.1 Global Methods

Global graph partitioning methods calculate a partition of the graph based on different informations of the graph. Some simple heuristics like the random method do not consider any adjacency information. Others, like the geometric methods, can only be used if geometric information is available.

4.1.1 Random Vertex Distribution

A simple solution of the graph partitioning problem is to randomly distribute the vertices of V among p parts. This method starts with all parts being empty. The vertices are one after another assigned to a part by randomly choosing one part which has less than $\lceil \frac{|V|}{p} \rceil$ vertices. The random partitioning method produces partitions with a cut size of approximately $(1-\frac{1}{p})|E|$. This cut size is usually much higher than the lowest possible one. Although this method is not a good approach for the partitioning problem, we may use it

as a starting point for efficient local partitioning methods. Furthermore, this method can be used to compare the solution calculated by a weak method to the solution calculated by a sophisticated one.

The consideration of taking the initial ordering of the vertices can be regarded as a quasi-random method. Each application has to pass the information of the graph to the graph partitioning task, i. e. the vertices have already been ordered by the application. During the generation process of each graph – in some cases – vertices of dense areas of the graph are grouped together closely in the list of vertices. If we assign the first $\frac{V}{p}$ vertices to part 1, the second $\frac{V}{p}$ vertices to part 2 and so on, this linear partitioning method can result in a partition with a low cut. Nevertheless, this approach usually results in partitions with high cuts, because it does not consider edges that connect vertices. The main advantage of this method is the fact that it is simple and fast.

4.1.2 Greedy Methods

Greedy methods are commonly used to solve the graph partitioning problem. Unlike random methods, the greedy methods consider the adjacency information. There are many different variations of this type of approach. They all start with empty partitions and, according to a certain order, they handle one vertex after another and assign it to a part of the partition. Once an assignment for a part is done, the decision will not be changed later.

A common variation is the *Breath-First Search* algorithm by Farhat [Far88, FL93] which is a greedy approach based on the breath-first strategy. It starts with assigning a vertex with the minimum degree to V_0. The method proceeds in breath-first manner. It considers all vertices which are not all ready assigned to any part. Among them, it adds all vertices to V_0 which are adjacent to any vertex already in V_0. Thus, it proceeds in levels of vertices with the same distance to the initial vertex. The method assigns vertices to V_0 until V_0 is of size $\lceil \frac{|V|}{p} \rceil$. Another vertex of the remaining graph, which is adjacent to a vertex of V_0 is taken as new seed for V_1. V_1 and all following parts are filled in the same manner as V_0. An example of $p = 2$ is presented in Fig. 4.2. The time requirement of this method is linear to the size of the graph. The solutions are partitions with compact parts and fairly low cuts. A major disadvantage of this method is the fact that the last part consists of all leftover vertices. They may form an elongated or even disconnected part.

One way to overcome this problem is to start the growing of the parts simultaneously at several vertices, one for each part. For $p = 2$, which is presented in Fig. 4.2, it was used in e. g. [Sim91, VSB92]. The calculation of the two vertices with maximum distance is very time consuming. Therefore, several heuristics are known to calculate two vertices with a high distance.

There is a frequently used modification to this method. Instead of including new vertices in the breath-first manner it includes vertices in the *Cut-First* manner (see

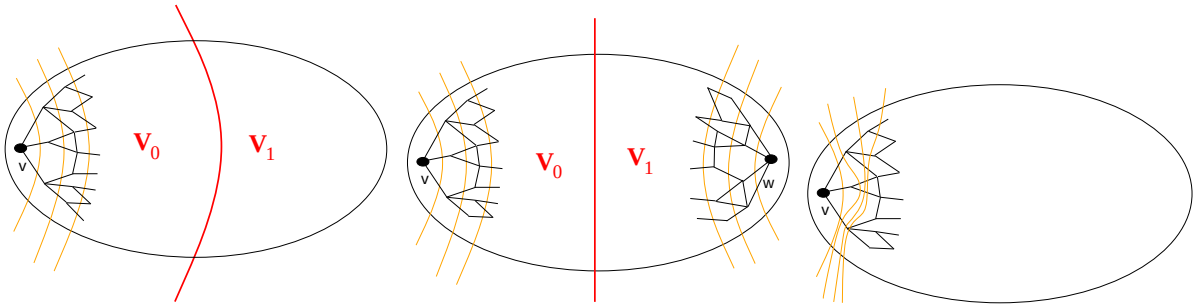


Fig. 4.2: Greedy Partitioning methods. Left: the parts have grown by the use of the breath-first strategy. Center: the parts have grown simultaneously from two distant vertices. Right: the parts have grown by the use of the cut-first strategy.

e. g. [Pre94]). An example is given in Fig. 4.2. The cut between the assigned vertices and the rest of the vertices is considered. All vertices that have not been assigned yet are checked for the new cut which would result if this vertex was included into the current part. One vertex with the smallest resulting cut will be included. This method has a time requirement that is linear to the size of the graph. It often results in partitions with lower cuts as compared to the Breath-First manner.

4.1.3 Geometric Methods

Several geometric methods can be used if some geometric information is available. This information is usually coordinates of the vertices in two or three dimensions. It is the idea of these methods to cut the graph perpendicular to its elongation.

The *Coordinate Sorting* method is only based on the vertex coordinates. It determines which of the x-, y- or z- coordinates have the widest range. The graph is partitioned perpendicular to the axis of that coordinate such that both parts have the same number of vertices. An example is presented in Fig. 4.3. The time requirement is dominated by the time for the sorting of the vertices according to the axis that has the widest range.

There are two different approaches to partition the graph into $p > 2$ parts. Firstly, we can partition the graph with $p-1$ lines perpendicular to the axis that has the widest range. Secondly, we can apply the bisection strategy recursively until there are p parts. In the later case, each bisection is performed perpendicular to the axis of widest range according to the corresponding subgraph. I. e. the cuts of the bisections can belong to different axis. This recursive coordinate bisection algorithm was used in e. g. [Sim91, VSB92].

The disadvantage of the coordinate sorting method is the fact that it only does allow to cut along two or three axis, but the elongation of the graph may not be parallel to one of these directions. The *Inertial* method (see e. g. [FL93]) calculates the principle

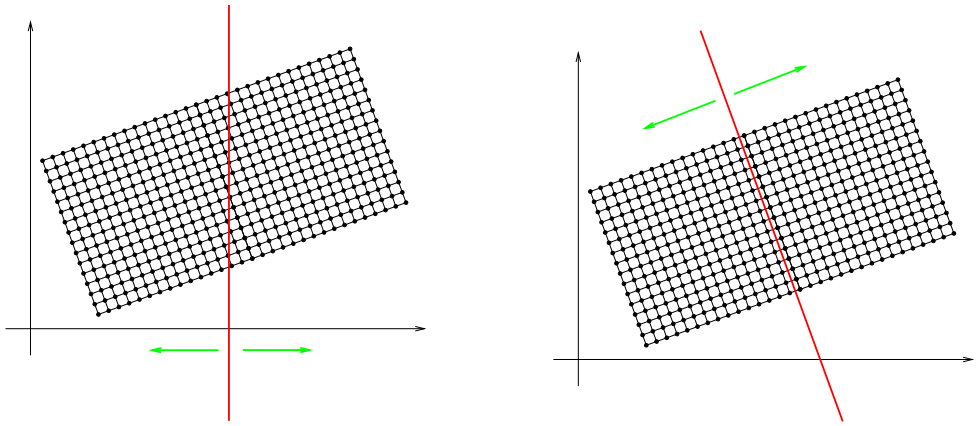


Fig. 4.3: Geometric partitioning methods. Left: The coordinate sorting method cuts perpendicular to the axis with widest range. Right: The Inertial method cuts Cut perpendicular to axis with the lowest moment of inertia.

inertia direction which is the elongation of the graph. This direction can be derived by calculating the eigenvector of the 3×3 inertia matrix. This can be done very fast. The graph is partitioned perpendicular to the direction of the inertia.

Although the geometric methods do not consider any connectivity information of the graph, they aim to assign vertices that are close together in space to the same part. For many graphs in typical applications (such as e. g. FEM-simulations), these methods results in partitions with reasonable cut sizes. Furthermore, the required period of time is dominated by the sorting of the coordinates. The sorting can be performed fairly fast.

In [MTV91, MTTV93] it has been shown that certain classes of geometrically defined graphs have good separators. The idea of the concept is to map the d-dimensional graph into a $(d+1)$-dimensional space. Implementation aspects and experiments with this approach are presented in [GMT95]. In [GMT94] this approach is extended with the use of the inertia matrix of the graph.

There are several papers about partitioning of planar graphs. In an early paper, Lipton and Tarjan exhibit an algorithm which finds a vertex separator of a graph such that each part contains at most $\frac{2}{3}|V|$ vertices and the vertex separator contains at most $2\sqrt{2|V|}$ vertices [LT79]. The vertex separator has been improved in [Dji82] to a size of $\sqrt{6|V|}$. In [BP92] an algorithm is proposed that calculates the bisection width of a planar graph in time $O(bw^2 \cdot |V|^3 \cdot 2^{4.5 \cdot bw})$ with bw being the bisection width of the graph [BP92]. Thus, for planar graphs with a small bisection width of $O(\log(|V|))$ the algorithm calculates the bisection width in polynomial time.

4.1.4 Spectral Methods

A commonly used global partitioning heuristic is the *Spectral Method*. It is based on algebraic graph theory described in Section 3.3.1. The graph partitioning problem is solved by the calculation of an eigenvector. The second smallest eigenvector y_2 of the Laplacian expresses the *algebraic connectivity* of the graph. It is called *Fiedler*-vector [Fie73, Fie75]. One of Fiedlers results can be rephrased as follows (see [PSL90]). If the graph $G = (V, E)$ is connected, for a real number $r \geq 0$ the subgraph $V_0 = \{v \in V; y_2(v) \geq -r\}$ is also connected. Similarly, for a real number $r \leq 0$ the subgraph $V_0 = \{v \in V; y_2(v) \leq |r|\}$ is connected, too. This provides us with some intuitive hint on why the bisection along the eigenvector y_2 of the second smallest eigenvalue leads to a good partition.

The Spectral method, as originally described in [PSL90], determines y_2 to the Laplacian and splits the graph according to the entries in the vector. This is done such that all vertices with an entry higher than a certain value are assigned to one part and all vertices with an entry lower than a certain value are assigned to the other part (Fig. 4.4). Thus, at least one part is connected (if the graph is connected, too). In order to get a balanced bisection the splitting value has to be chosen as the median of all vector entries. The results of the spectral bisection method are usually fairly good, especially from a global viewpoint. Nevertheless, there are local improvements possible.

Construct the Laplacian matrix;
Compute eigenvector y_2 to the second lowest eigenvalue λ_2;
Partition the vertices according to the median value m of the entries in y_2:
    $V_0 = \{v \in V;\ y_2(v) < m\}$ and $V_1 = \{v \in V;\ y_2(v) > m\}$;
    distribute $\{v \in V;\ y_2(v) = m\}$ among V_0 and V_1 to result in a balanced bisection;

Fig. 4.4: Spectral Bisection algorithm

The determination of the eigenvectors of the Laplacian is an expensive task which limits the feasibility of the method. Therefore, in implementations of libraries like *Chaco* [HL94a], an efficient incomplete orthogonalization is used or multi-level solvers as described in [BS94]. Additionally, certain attempts are made to use more eigenvectors [HL93, HL95a, AKY98]. An analysis of the performance of spectral graph partitioning methods is to be found in [GM95]. There are some approaches to combine spectral and geometric methods [CGT95, SSB98].

4.1.5 Simulated Annealing

Simulated Annealing [KGV83] (SA) is a general purpose optimization scheme. The scheme results from statistical mechanics. It is based on a local search and aims to improve a given solution with the help of local rearrangements.

The description of a solution and the neighborhood relation between solutions are important. The neighborhood is based on local rearrangements. A cost function assigns a specific cost value to each solution. The cost function is aimed to be minimized by changing from one solution to another according to the neighborhood relation. The description of the solution, the neighborhood relation and the cost function do only depend on the specific optimization problem.

A major advantage of SA is the fact that it permits a certain amount of deterioration of the cost function when switching to a new solution. A second advantage is the fact that the probabilistic choice of a new solution in the neighborhood relation. These advantages enable us to escape from local optimal solutions.

SA iteratively proposes new solutions and evaluates the cost value of them. If the new value is lower than that of the previous solution, the new solution is kept. Otherwise, the new solution is kept with some probability. This process is repeated until a desired solution is found or a maximum number of iterations is exceeded.

The user has to provide the SA algorithm with several pieces of information. Depending on the given problem, a description of a valid solution and a cost function based on them has to be specified. In addition, local rearrangements according to a neighborhood relation have to be provided.

Apart from the given problem, the SA algorithm needs a *starting temperature* and a *cooling schedule* which control the actions by providing a current temperature. According to the current temperature, the algorithm accepts a solution with a higher cost function more or less likely. These parameters are important because the time needed and the quality of the solution greatly depend on an accurate choice of them. With an infinitely slow cooling schedule, the SA algorithm converges on the optimal result.

SA was successfully applied to graph partitioning [KGV83, JAMS89, DLMS96]. Its main disadvantage, the long running time that is necessary to preserve convergence properties, can be overcome by tuning the method to the given problem [JAMS89] or by parallelization [DLMS96].

4.2 Local Methods

Although a global partitioning method already produces a balanced partition, local methods try to further improve it. The potential for improvement depends on the difference between the current cut size and the (unknown) minimum cut size. Experimental experiences show that it is always preferable to add local methods. On partitions with high cut sizes they substantially improve the partition. Furthermore, even on partitions generated by efficient global methods they usually decrease the cut size without requiring too much computation time. We discuss the traditional Kernighan-Lin and the Helpful-Set approaches in this section.

4.2.1 Kernighan-Lin

The *Kernighan-Lin* heuristic [KL70] (KL) is the most frequently used local graph bisection method. It uses a sequence of logical exchanges of vertex pairs to determine the sets that have to be exchanged physically. The *diff*- and *gain*-values are introduced in order to calculate the changes of the cut size if a vertex or a pair of vertices is move to the other part.

Definition 8: *The* **diff***-value of a vertex v is the difference of the number of its external edges and the number of its internal edges, i. e.*

$$diff(v) := |\{w \in V; \{v,w\} \in E, \pi(v) \neq \pi(w)\}| - |\{w \in V; \{v,w\} \in E, \pi(v) = \pi(w)\}| \ .$$

The **gain***-value of two vertices $u \in V_0$ and $v \in V_1$ are defined by*

$$gain(u,v) := diff(v) + diff(v) - 2|\{u,v\} \in E| \ .$$

In Fig. 4.5 an example of the internal and external edges of a vertex v is illustrated. The value of *diff*(v) describes the decrease of the cut size, if v is moved to the other part. This value plays a major role in the KL-algorithm. If two vertices u and v of different parts are swapped, the value $gain(u,v)$ describes the decrease of the cut size.

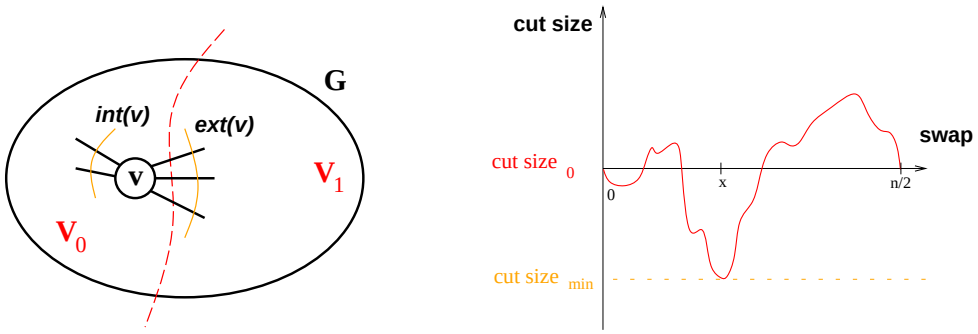


Fig. 4.5: Left: The *diff*-value of a vertex v. Right: The sequence of cut sizes during one pass of KL.

The original algorithm by Kernighan and Lin [KL70] is based on an exchange of vertex pairs. Thus, the balance of the bisection remains the same. The calculation of the vertex pair with the highest gain is the time consuming part. There is a simple implementation with a runtime of $O(|V|^3)$. However, by sorting the vertices according to their *diff*-value the algorithm can be sped up to $O(|V|^2 \log(|V|))$. These times count for one pass of the KL-algorithm. Several passes can be executed until the cut size does not decrease anymore.

Fiduccia and Mattheyses [FM82] (FM) modified the KL-method and used a sequence of single vertex moves to determine the sets. They additionally use an efficient data structure *Buckets* which reduces the number of steps for each pass to only $O(|V| + |E|)$.

In Fig. 4.6 we show a variation of the KL/FM algorithm as it is implemented in the PARTY library (Chapter 6). The REPEAT-loop invokes a pass of the algorithm. In each pass at first the *diff*-values of all vertices are computed. It progresses by moving one unlocked vertex at a time to the other part. Both the source part and the vertex are chosen carefully. The source part is chosen in the following manner. Consider the two directions into which the vertices can be moved. If exactly one moving direction results in an unbalanced bisection, the other moving direction is chosen. Otherwise, the part with the highest *diff*-value of any unlocked vertices is chosen (part 0 is taken in the possible case of a tie). Then, an unlocked vertex $v \in V_i$ with a maximum *diff*-value is chosen and moved logically to the other part. This implies a change of *diff*-values of its neighbors, which have to be updated. The move results in a new bisection.

The counter *new_steps* counts the number of moved vertices since the last balanced bisection with an improved cut. This counter is important for the termination of a pass. The original KL-algorithm modified by Fiduccia and Mattheyses terminates the pass only when all vertices are moved to the other part. Several scientists [HL94a, Pre94] experienced the fact that the final balanced bisection with the lowest cut usually occurs fairly early in a pass. In order to reduce the run time, a pass terminates if a balanced balanced bisection with an improved cut could not be achieved in the last $\frac{|V|}{4}$ moves.

```
REPEAT
    compute the diff-values of all vertices and set new_steps to 0;
    WHILE V0,V1 have unlocked vertices and new_steps< |V|/4
        choose an unlocked vertex v with diff(v) maximal;
        move v logically to the other part and lock it;
        update the diff-values of the neighbors of v;
        IF result is a balanced bisection with lowest cut so far
            new_steps = 0;
        ELSE
            new_steps = new_steps +1;
    ENDWHILE
    move sequence of locked vertices resulting in the minimum cut size physically over;
UNTIL cut is not improved
```

Fig. 4.6: Local bisection algorithm based on KL and FM.

After each pass the change of the cut throughout all moves is analyzed. Only the sequence of moved vertices up to the balanced bisection with the lowest cut (see Fig. 4.6) is determined. Only the vertices up to this point are physically moved to the other part. The

result is a new balanced bisection with an improved cut. Further passes of the algorithm are carried out on the resulting bisection until no further improvement can be achieved.

In general, the KL method is robust and reliable. The results are convincing, provided that KL was started with a fairly satisfactory global bisection. There is a parallel algorithm based on KL [GZ87]. Furthermore, a fast strategy to search for the best pair of vertices is presented in [Dut93].

4.2.2 Helpful-Set Algorithm

The Helpful-Set concept is based on a constructive proof of an upper bound on the bisection width of regular graphs [HM92, MD97] (compare Section 3.1). The helpfulness of a set was stated in Definition 5. A move of an h-helpful set S from one part to the other decreases the cut size by h edges. Fig. 4.7 presents us with some examples for 2-helpful sets. The vertices are labeled with the *diff*-value which expresses their own "helpfulness" (Definition 8). It can be observed that helpful sets may include vertices with a large negative gain.

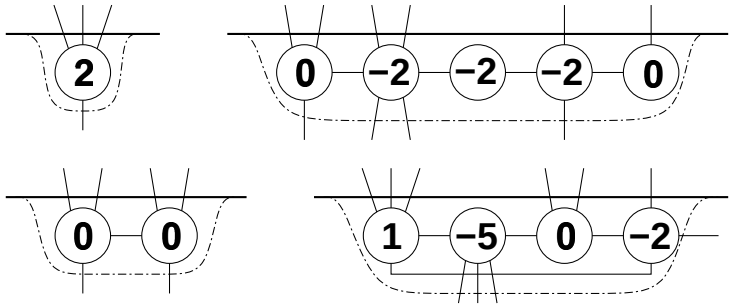


Fig. 4.7: Example of *diff*-value and *helpfulness*: The solid line is the boundary between the parts, the vertices are marked with their *diff*-values and all sets marked with the dashed lines are 2-helpful.

The technique to prove these upper bounds led us to the efficient and powerful *Helpful-Set* heuristic [DMP95] (HS). This approach uses helpful sets for an iterative local improvement of bisections. The idea is presented in Fig. 4.8. HS starts with an arbitrary bisection and aims to improve it in a round which consists of two phases. In the first phase, it searches for an h-helpful set S with $h > 0$ on each side of the bisection. If such a set can be found, it will be moved to the other side. In the second phase it searches for an equally sized balancing set $\bar{S}$ with a helpfulness of at least $(-h + 1)$ in the over-weighted part. Again, this set will be moved to the other part. Thus, in each successful round the cut size decreases by at least one. The strategy terminates when no adequate sets are to be found.

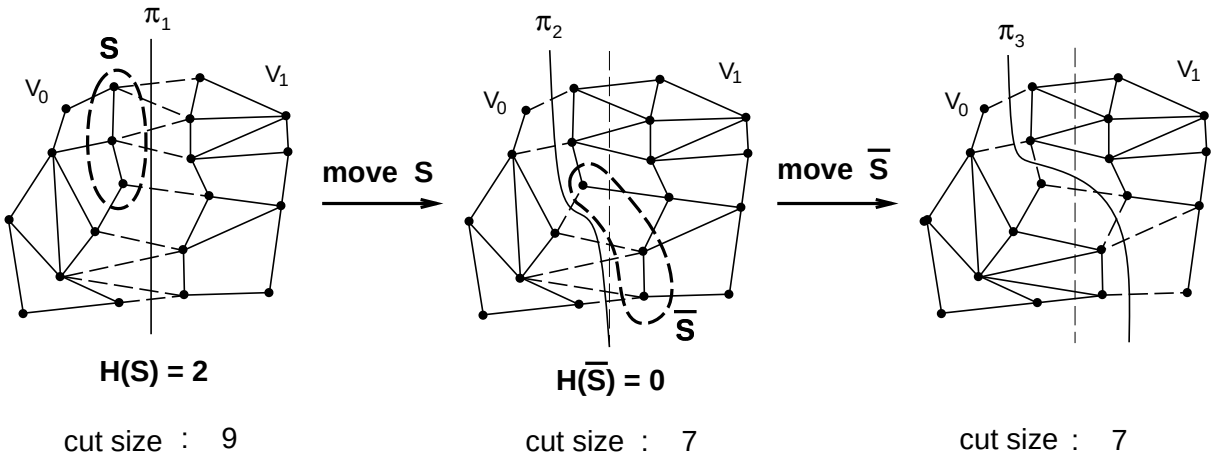


Fig. 4.8: A successful round of the Helpful-Set algorithm with searching for a h-helpful set S and balancing with set $\bar{S}$.

An outline of the Helpful-Set algorithm is shown in Fig. 4.9. It starts searching for $(cut(\pi)/2)$-helpful sets which may be fairly large. If an h-helpful set is found and moved to the other side, HS tries to re-balance the bisection with an at least $(-h+1)$-helpful balancing set $\bar{S}$. If the balancing fails, the original set will be moved back. In this case we lower the helpfulness-bound at which the search is stopped. We hope that in further iterations smaller sets can be found which are easier to balance. A balancing set $\bar{S}$ to an h-helpful set S is a set of nodes from the larger one of V_0, V_1 with size $|\bar{S}| = |S|$ and helpfulness $\geq (-h+1)$. The net improvement of one step, i.e. moving S to the other side and balancing with $\bar{S}$, is at least one edge.

The search will terminate, if either no set with positive helpfulness can be found, or if a 1-helpful set cannot be balanced. Usually, the size of the helpful sets is not considered directly but controlled by their helpfulness. A set which is very helpful is often larger than a less helpful one. This fact changes to the end of the search.

In [HM92, MD97] it is shown for regular graphs that as long as the cut of the bisection is larger than a certain value, a helpful set S and a balancing set $\bar{S}$ can be found. Therefore, the algorithm can be iterated until at least this upper bound is achieved For reasons of appropriateness, the algorithm does not stop when it reaches this upper bound value, but continues until it does not find any helpful set S or until even very small sets cannot be re-balanced. The guaranteed upper bound makes the Helpful-Set strategy a real alternative to standard Kernighan-Lin based local heuristics.

```
limit = cutsize/2;
REPEAT
    search for h-helpful set S with h ≥ limit;
    IF (no h-helpful set S with h ≥ limit found)
        IF (any h-helpful set with h > 0 found)
            S = set with highest helpfulness found;
            limit = h(S);
        ELSE
            S = ∅;
            limit = 0;
    IF (S ≠ ∅)
        move S to the other side;
        search for a balancing set S̄ of S;
        IF (successful)
            move S̄ to the other side;
            limit = limit · 2;
        ELSE
            move S back to its original position;
            limit = ⌊limit/2⌋;
UNTIL (limit = 0)
```

Fig. 4.9: The Helpful-Set algorithm.

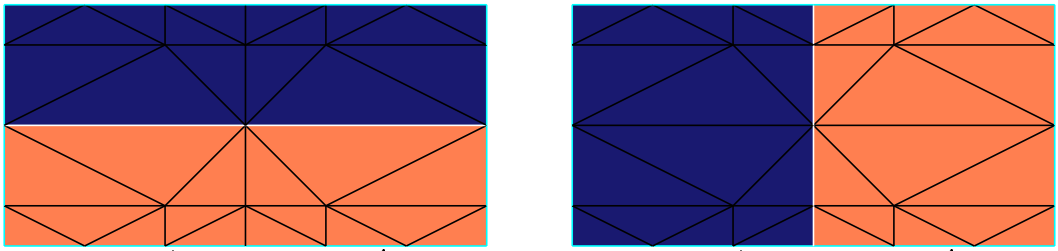
4.3 Optimizing the Aspect Ratio

Above we described the graph partitioning methods in order to minimize the cut size of the partition. This is not the right measure for some applications. E. g., the parallel simulation of the Finite-Element-Method (FEM) requires the distribution of the FEM-mesh such that not only the interdependencies among the parts is minimized, but also the calculation within each part can be performed efficiently. The *shape* of the parts heavily influences the quality of pre-conditioning and, thus, the overall execution time. In this section we measure the shape of the parts by different definitions of the Aspect Ratio and propose a new heuristic *Bubble* that is specially designed to minimize the Aspect Ratio of a partition.

4.3.1 Definitions of Aspect Ratio

An example of the cut size not always being the right measure in mesh partitioning is to be found in Fig. 4.10. The sample mesh is partitioned into two parts with different AR's and different cuts. A Poisson problem with homogeneous Dirichlet-0 boundary conditions

is solved with the help of a Preconditioned Conjugate Gradient method [BBD95]. The number of iterations is determined by the AR. The cut size (number of neighboring elements) would be the wrong measure.



$cut = 2,\ AR = 1.56,\ \#It's = 8$ $cut = 4,\ AR = 1.00,\ \#It's = 5$

Fig. 4.10: Partitioning an example mesh into two parts. The elements of the mesh are the vertices of the *dual graph* and two vertices are connected by an edge if they share a boundary. A partition with a low cut (left) and a partition with a low AR (right).

Possible definitions of AR are to be found in Fig. 4.11. The first two definitions are motivated by common measures in triangular mesh generation where the quality of triangles are expressed in either $\frac{L_{\max}}{L_{\min}}$ (longest to shortest boundary edge) or $\frac{R_o^2}{R_i^2}$ (the area of smallest circle containing the domain to the area of largest inscribed circle). The definition $AR = \frac{R_o^2}{R_i^2}$ expresses the fact that circles are perfect shapes. Unfortunately, circles are quite expensive to find for arbitrary polygons: by the use of Voronoi-diagrams, we can determine both circles in $O(2n \log n)$ steps where n is the number of nodes of the polygon (and faster incremental update algorithms are not known). The definition $AR = \frac{R_o^2}{A}$ (A being the area of the domain) is another measure that favors circle-like shapes. It still requires the determination of the smallest outer circle but turns out to be better in practice. We can take a further step and approximate R_o by the length B of the boundary of the domain (which can be determined fast and updated incrementally in $O(1)$). For a sub-domain with area A and perimeter B, $AR = \frac{B^2}{16A}$ is the ratio between the area of a square with perimeter B and area A. This definition assumes that squares are perfect domains.

Circles offer a better perimeter/area ratio but force neighboring domains to become concave (Fig. 4.12,left). The measure $AR = \frac{L_{\max}}{L_{\min}}$ does not express the *shape* properly for irregular meshes and partitions. Fig. 4.12(center) presents an example. P_1 is perfectly shaped, but as the boundary towards P_4 is very short, $\frac{L_{\max}}{L_{\min}}$ is large.

The circle-based measures usually fail to rate jagged boundaries or inscribed corners. Fig. 4.12(right) displays examples each of which have the same AR but which are very different in shape. According to our experience, $AR = \frac{R_o^2}{A}$ and $AR = \frac{B^2}{16A}$ turn out to be the most robust measures, which best express the desired aims of producing compact domains.

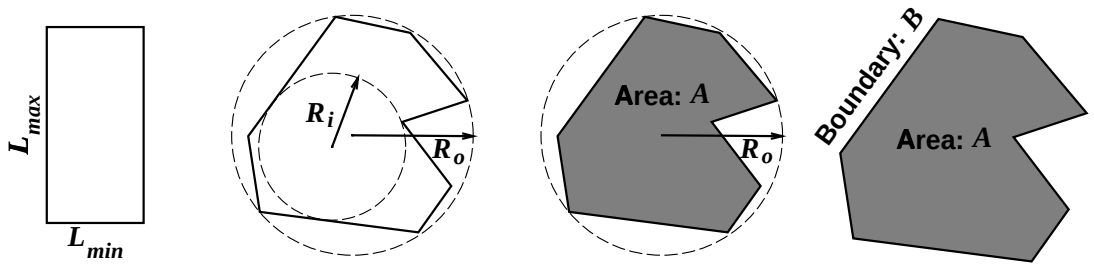


Fig. 4.11: Different definitions of Aspect Ratio: $\frac{L_{\max}}{L_{\min}}$, $\frac{R_o^2}{R_i^2}$, $\frac{R_o^2}{A}$ and $\frac{B^2}{16A}$.

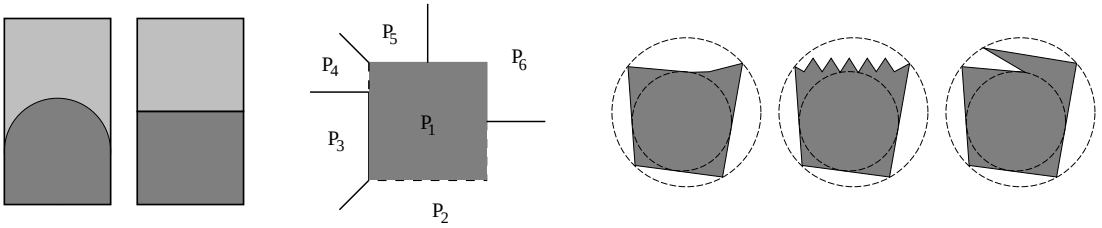


Fig. 4.12: Problems of several definitions of AR. Left: circles as perfect shapes force non-convex neighbors, squares allow square neighbors. Center: What is $\frac{L_{\max}}{L_{\min}}$? Right: Three Examples with the same $AR := \frac{R_o^2}{R_i^2}$, but different shape qualities.

Load balancers that are particularly designed to optimize the subdomain AR can be found in [DMM98, FMB95, VFC+96, WCDS99]. The tool PAR^2 was originally constructed as a parallel partitioner, but it can also be used in an adaptive environment [DMM98]. The method described in [FMB95] is an iterative partitioner which tries to improve the subdomain AR in a number of steps. Other attempts are to optimize subdomain shapes by the use of meta-heuristics such as Simulated Annealing (SA) [JAMS89] or Tabu Search [VFC+96].

4.3.2 Bubble Partitioning Method

In this section, we propose a center-oriented method called Bubble (BUB) for the partitioning of the initial mesh which implicitly optimizes the shape of subdomains. Some of the basic ideas are simple and natural. The Bubble method has some similarities to other approaches. Bubble generalizes some ideas of the greedy graph partitioning methods described in Section 4.1.2 like the bisection growing method of [Sim91]. Similar center-based approaches were developed in [GS94, SS99], and a parallel center-based approach can be found in [WC99]. There are similarities to an algorithm for vector quantizer design [LBG80] where the vector entries are partitioned while considering a reproduction alphabet with as many elements as there are parts.

The idea of Bubble (displayed in Fig. 4.13) is to represent a partition by a set of seed vertices, one for each part, from which the subdomains are grown simultaneously in a breadth-first manner until the whole mesh is covered. Colliding parts form a common border and keep growing along this border – just like soap bubbles in a bath. After the whole mesh has been covered, the algorithm determines its "center" vertex for each part. This center vertex is defined as the new seed and the subdomain growing process starts again. The iteration will be stopped if the movement of all seeds is small enough, i. e. if the seed vertices are close to the centers for all parts.

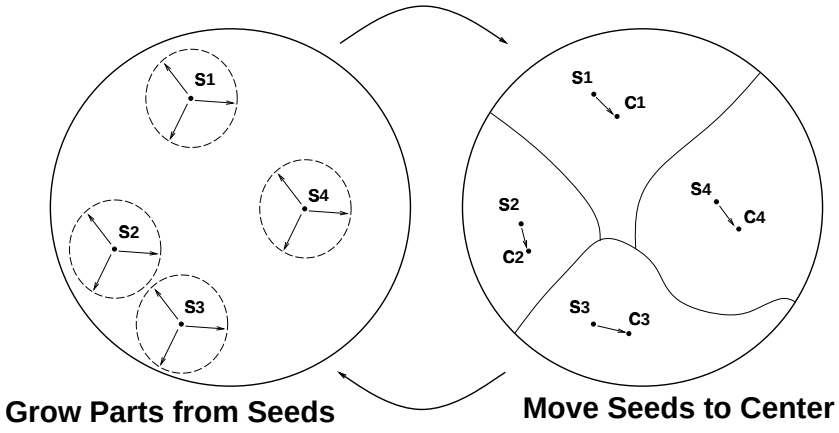


Fig. 4.13: The iterative Bubble method.

The algorithm is based on the observation that within "perfect" bubbles, the center and the seed vertex coincide. The distances in this method may either be chosen as the path length or as the Euclidean distances. In the case of path length the method works on graphs without geometrical information, too. The Bubble-algorithm is shown in Fig. 4.14.

perform breath-first search from an element v of minimal degree;
take an element in the furthest distance to v as the seed of part 1;
FOR i from 2 to P DO
    perform breath-first search from seeds 1,...,$i-1$ simultaneously;
    take an element in the furthest distance as the seed of part i;
END-FOR
DO Grow parts from seeds in breath-first manner;
    Calculate centers of parts and assign them as new seeds;
UNTIL (the seeds do not change OR the AR did not improve for 10 iterations)

Fig. 4.14: The Bubble-Algorithm.

In order to find the initial seeds, we start a breadth-first search (BFS) from a vertex with a minimal degree. In the case of FE-meshes, this is usually an element in a domain

corner. Furthermore, we search for the vertex which is farthest away from this starting point. This vertex is chosen as the first seed. We repeat simultaneous BFS from all seeds that have been found so far to determine a vertex which is farthest away from all seeds. This vertex becomes the next seed. Altogether, P BFSs are performed with P being the number of parts. With the help of this approach of each new seed having the maximum distance from all previous ones we distribute the seeds evenly over the graph. The path length is used as a distance measure in the first loop.

The main loop of Bubble is started by growing the parts from each seed in a breadth-first manner, i. e. each part checks whether any of its elements is adjacent to an uncovered element. The smallest part with at least one such adjacent element grabs the one with the shortest Euclidean distance to its seed and assigns it to that part. Only those vertices are added which are adjacent to vertices that were previously assigned to the same part. This fact ensures the connectivity of each of the parts. The ordering of smallest parts aims to keep the final load difference small. The choice of an adjacent element with shortest Euclidean distance benefits a low AR of that part. This action is repeated until all vertices (elements) are covered. Afterwards, new seeds are calculated by each part independently by searching for the center vertices. This task could be executed in parallel. We call the *distance-value* of a vertex to be the sum of Euclidean distances to all other vertices in the same part. We define the center of a part to be a vertex for which the distance-value is minimal. We may find the centers by calculating the distance values for all vertices. However, this process would have a time consumption of $O(\#\text{elements}^2)$. In order to avoid this, we calculate the distance-values for the seed as the initial center and all its adjacent vertices. We move the center to the neighbor with the smallest value; this process is repeated until a local minimum is found. The Bubble algorithm terminates when none of the seeds move in an iteration. To avoid cyclic movements of seeds (which do occur sometimes), we stop the algorithm, if the AR does not improve in ten consecutive iterations.

Bubble produces connected parts which are in general compact and have a smooth shape. A major drawback of Bubble is the fact that it lacks of a guarantee for balanced partitions. The seeds are spread out evenly over the whole graph at the end, but the parts do not have to contain the same number of elements. In order to overcome this drawback, we add a local partitioning method to balance the load. Thus, we aim to further optimize either the cut or the AR.

We investigate the performance of different types of mesh partitioning strategies with respect to the number of global iterations of the DD-PCG, the cut size and the AR. We include the simple coordinate sorting method COO which partitions the graph into several parts at once. We also include the recursive coordinate sorting method COO_R (Section 4.1.3). Furthermore, we use the Greedy Breath-First method GBF and Greedy Cut-First method GCF as described in Section 4.1.2. Bubble is used in different settings. It is used without any load balancing, as well as with additional load balancing minimizing

either the cut size or the AR. In addition, we use the default settings of the PARTY, JOSTLE, K-METIS and P-METIS graph partitioning libraries (Section 6.1). Finally, we used a Simulated Annealing code which is designed to optimize the AR.

Fig. 4.15 displays the results of the partitioning of the meshes *turm* (with 531 elements) and *cooler* (with 749 elements) into eight parts. The test case for the numerical solver is a Poisson problem with Dirichlet-0 boundary conditions. The number of iterations for the tested methods differ significantly. It can be observed that the AR $\frac{L_{max}}{L_{min}}$ and $\frac{R_o}{R_i}$ do not follow the line of the iteration numbers, whereas the values of $\frac{R_o}{A}$ and $\frac{B^2}{16A}$ roughly do so. The results indicate that a low *cut* leads to a fairly low number of iterations. A comparison of the tested partitioners reveals that the simple coordinate and the greedy methods usually result in large iteration numbers. For the bubble variations, balancing by improving the AR $\frac{B^2}{16A}$ leads to lower iteration numbers than balancing by improving the cut. Furthermore, the partitions calculated by both the partitioning libraries and the Simulated Annealing approach lead to similarly low iterations numbers.

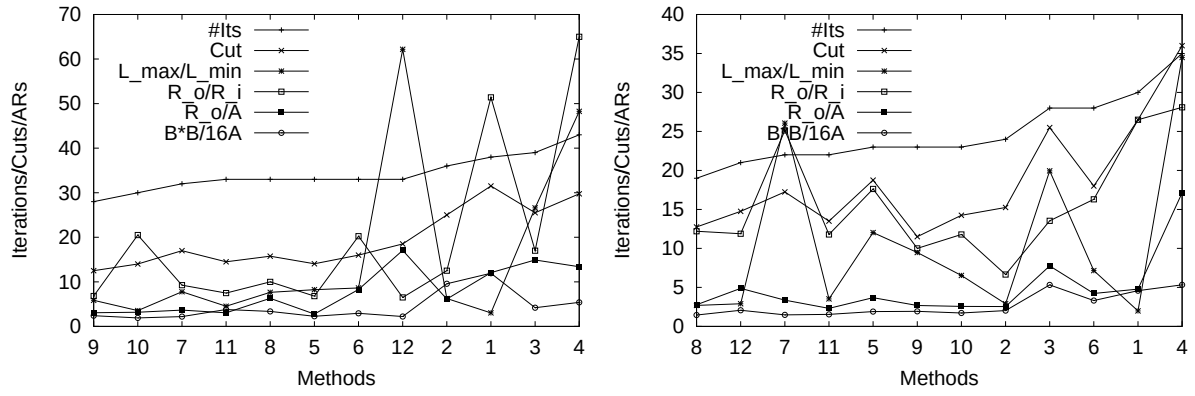


Fig. 4.15: Results of example *turm* (531 elements, top) and example *cooler* (749 elements, bottom). The methods are listed with increasing numbers of global PCG iterations. 1:COO, 2:COO_R, 3:GBF, 4:GCF, 5:BUB, 6:BUB+CUT, 7:BUB+AR, 8:PARTY, 9:JOSTLE, 10:K-METIS, 11:P-METIS, 12:SA.

Tab. 4.1: Comparisons for mesh *crack* (20141 elements) partitioned into 16 parts.

	CHACO	METIS		JOSTLE	BUB		
		K-	P-			+CUT	+AR
Cut	856	800	760	768	920	799	813
AR	1.93	1.85	1.98	1.92	1.92	1.96	1.87
Iterations	46	46	53	44	50	50	44

Tab. 4.1 reveals some additional results for the mesh *crack* with 20141 elements as an example. The mesh is partitioned into 16 parts. The Bubble method with its variations

is compared with the default settings of the partitioning libraries CHACO (multilevel approach), K-METIS, P-METIS and JOSTLE. We list the Cut, the average AR $\frac{B^2}{16A}$ of all parts, and the number of iterations of the DD-PCG as measures The results reveal that the AR is a better measure for the number of iterations than the Cut. To give a telling example, the partition calculated by P-METIS has the lowest cut, but needs the largest number of iterations. The partition calculated by Bubble has a high cut size but if it is load-balanced by a further improvement of the AR, it finally succeeds in achieving the overall goal: the number of global iterations of the DD-PCG algorithm is minimized. The cut becomes smaller if the Bubble partition is load-balanced minimizing the cut instead. However, the AR and the number of iterations are not as low as for the minimizing of the AR. Bubble serves as reasonable initial partitioner if it is combined with shape optimizing load balancing methods. The periods of time to calculate the partitions were fairly small for all methods due to the medium sized examples. In an adaptive environment, the partitioning has only got to be performed on the initial mesh which is usually small. For larger meshes, Bubble can be used as partitioning method for the coarse graph in the multilevel paradigm after coarsening the large initial mesh into a much smaller one.

5. MULTILEVEL GRAPH PARTITIONING

Multilevel strategies have proven to be powerful approaches, in order to partition graphs efficiently. Their efficiency is dominated by two parts; the strategies for coarsening and for local improvement. Several methods have been developed to solve these problems. However, their efficiency has only been proven on an experimental basis.

We use new and efficient methods for both problems, while satisfying certain quality measurements. For the coarsening part we develop a new approximation algorithm for maximum weighted matching in general edge-weighted graphs. The algorithm calculates a matching with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching. The time complexity is $O(|E|)$, with $|E|$ being the number of edges in the graph. For the local refinement we use the Helpful-Set heuristic (Section 4.2.2) which provides us with an upper bound on the bisection width of regular graphs. Furthermore, it gives good results for general graphs. These quality methods used for the two parts of the multilevel approach lead to an efficient graph partitioning concept.

Parts of the work in this chapter were published in [Pre99b, Pre99a, MPD00].

5.1 The Multilevel Approach

There are several mesh partitioning software libraries which provide us with an easy access to a variety of efficiently implemented partitioning heuristics. However, many heuristics still have a high time complexity when they are used for very large graphs. For the last couple of years, the need for partitioning extremely large graphs with up to millions of vertices has increased. These graphs result from diverse application domains such as VLSI, simulation of flows, and crash tests. Classical heuristics for graph partitioning are too slow or do not deliver the requested accuracy when they are applied to graphs of the respective size.

Therefore, several heuristics on the coarsening approach were developed. The idea is to coarsen the large graph to a much smaller one with a similar structure (Fig. 5.1). Since the coarse graph is small, partitioning heuristics applied to the coarse graph are efficient.

In addition, it seems reasonable to build structures of tightly connected vertices as this sums up in determining a helpful pre-partitioning of the graph.

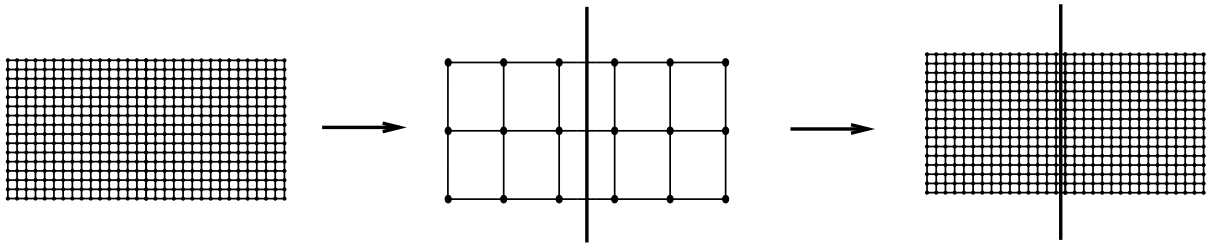


Fig. 5.1: The partition of the coarse graph is also a partition of the initial graph.

The coarse graph is supposed to have a similar structure as the original graph such that a partition of the coarse graph directly corresponds to a partition of the large graph with the same cut size. Each vertex of the coarse graph is built by collapsing vertices of the initial graph. An example is presented in Fig. 5.2. The weight of the vertex in the coarse graph becomes the sum of the weights of all vertices in the set it is representing. Therefore, the sum of the weights of all vertices is equal for the initial and the coarse graph. Furthermore, a partition of the coarse graph directly corresponds to a partition of the initial graph.

All edges of the initial graph that connect vertices which are assigned to the same vertex in the coarse graph do not appear in the coarse graph. All edges connecting vertices assigned to different vertices in the coarse graph remain in the coarse graph. We connect the corresponding vertices in the coarse graph. Possible multi-edges are combined to a single edge with an edge weight as the sum of edge weights of all combined edges. The cut size of a partition of the coarse graph is equal to the one of the corresponding partition of the initial graph.

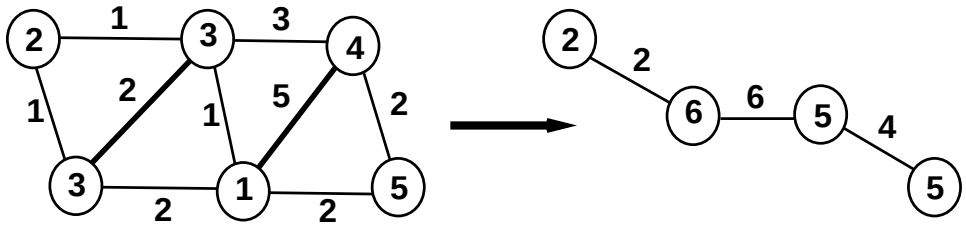


Fig. 5.2: Reduction of a graph. The bold lines are matching edges and the incident vertices are combined. Vertices and edges are labeled with their weight.

The coarsening process is usually performed with the help of the *Multilevel* strategy [HL95b, KK99a, KK98c] (Fig. 5.3). The graph is coarsen down in several levels until a graph with a sufficiently small number of vertices is constructed. The single coarsening

steps between two levels can be performed by the use of matchings, i. e. a matching of the graph is calculated and the vertices incident to a matching edge are contracted. Experimental results revealed that it is important to contract those vertices which are connected via an edge of a high weight, because it is likely that this edge does not cross between parts in a partition with a low weight of crossing edges. Several matching algorithms as described in Section 5.2 can be used for this tasks. The use of a maximum weighted matching might improve the coarsening step most, but the super-linear time complexity of an optimal algorithm is too high for real examples. Therefore, fast approximation algorithms are useful here.

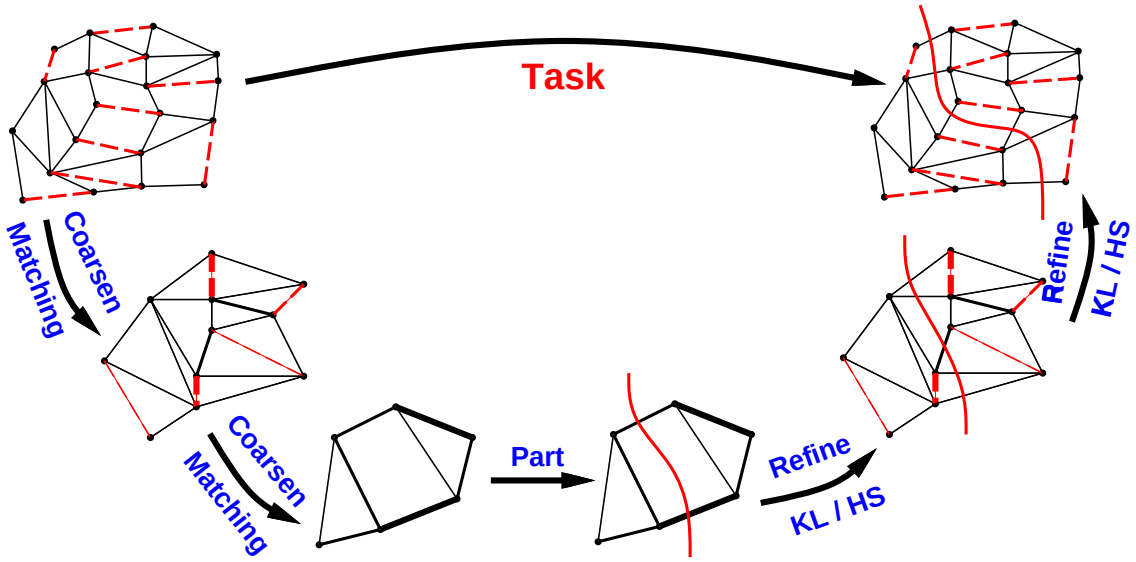
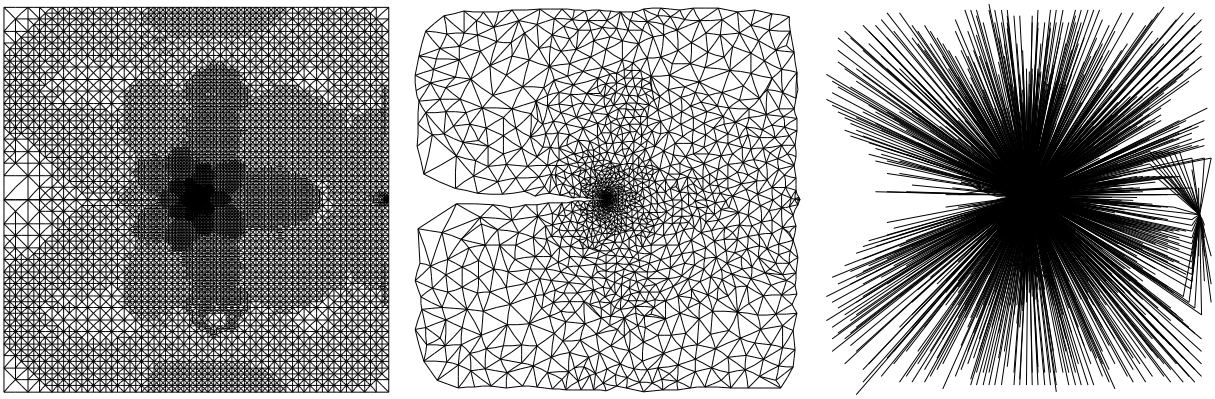


Fig. 5.3: The multilevel approach with several coarsening steps, a partitioning of the smallest graph and the local refinements.

The example in Fig. 5.4 reveals that the coarsening step does not always lead to a small graph with a similar structure. Depending on the method used for each coarsening step, the resulting small graph might degenerate as it is the case in the right picture of Fig. 5.4. Therefore, matching algorithms with certain quality characteristics are supposed to be used.

Any heuristic can be used to partition the coarse graph. It is even possible to coarsen the graph down to such a size, that even an optimal partitioning method which calculates the partition with the minimum cut size, can be applied. In PARTY, such a method, based on the Branch&Bound approach, is provided and can efficiently be applied to graphs with up to about 50 vertices. Furthermore, it is possible to coarsen the graph down to as many vertices as there are parts, i. e. each vertex of the coarse graph represents one part of the partition of the initial graph (see e. g. [Gup97]).



$|V| = 10240$ $|V| = 1373$ $|V| = 1811$

Fig. 5.4: Problems in the coarsening step. A 2-dimensional graph (left) coarsen to a small graph with a similar structure (center) and to a graph with a different structure (right).

Finally, the coarse graph is un-coarsen in the levels again. Therefore, the partition of a level is projected to the graph of one higher level. Since combined vertices are split in the un-coarsening process, the projected partition may not be balanced exactly anymore. The partition may also incorporate some local areas for an improvement of the cut. Therefore, local iterative improvement heuristics such as Kerninghan-Lin [KL70, FM82] or Helpful-Set (Section 4.2.2) can be applied to balance and further improve the partition.

Fig. 5.5 represents all levels for the partitioning of the graph *crack* with the multilevel approach.

In conclusion, the two critical parts of the multilevel approach are the coarsening strategy and the local improvement method. For the coarsening approach the following aspects are important.

- The matching algorithm has to be very fast so that it is more time efficient than standard partitioning methods that are applied to the initial graph.
- Since edges of high weights are usually connecting dense areas of the graph, the algorithm is supposed to calculate a matching with a high edge weight, in order to avoid them appearing in the coarse graph and being cut.
- To speed up the coarsening process and to coarsen the whole graph simultaneously, the matchings on each level should have a high cardinality. The maximum reduction is achieved by splitting the number of vertices in halves on each level. This is only possible with a complete matching.

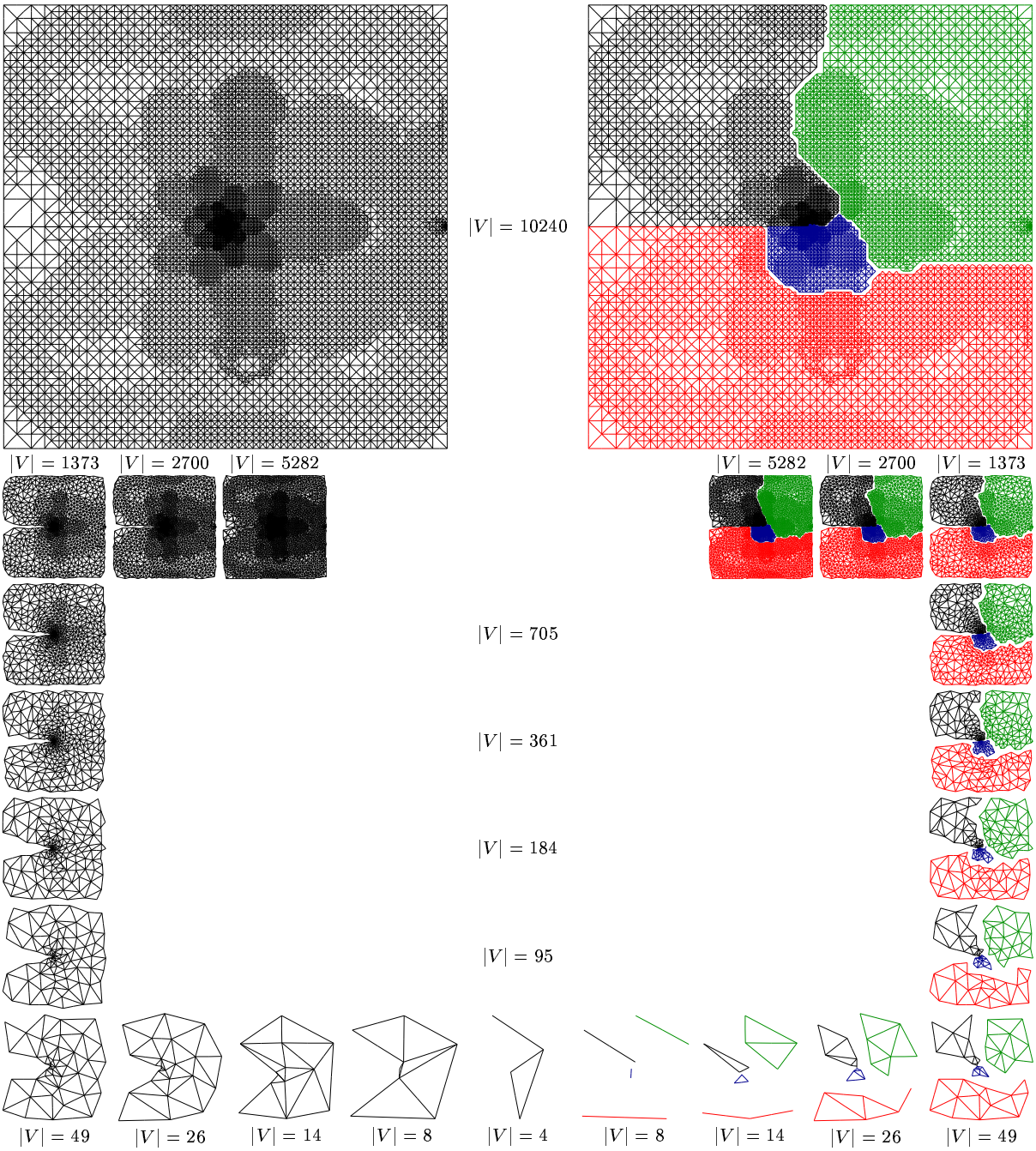


Fig. 5.5: Partitioning the graph *crack* into four parts with the multilevel approach. The weights of vertices and edges are not shown, but may have large deviations.

In Section 5.2, we therefore develop a new approximation algorithm for maximum weighted matching in general edge-weighted graphs. The algorithm calculates a matching with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching. The time complexity is $O(|E|)$, with $|E|$ being the number of edges in the graph. This result improves upon the previously known $\frac{1}{2}$-approximation algorithms for maximum weighted matching which require $O(|E| \cdot log(|V|))$ steps, with $|V|$ being the number of vertices. The approximation algorithm is as fast as currently used matching algorithms for multilevel graph partitioning. They calculate maximal matchings without any quality guarantee on the weight of the resulting matching.

It is left to mention that recently the multilevel-strategy has been used in the Metaheuristic *Cooperative Search* [TTG99]. It achieves partitions with very small cut sizes. The drawback is the very high time requirement. Although it has been parallelized such that each level is operated by a different processor, the times are still much higher than for the classical multilevel algorithm. This approach has been generalized to hypergraphs in [OTT+00].

5.2 Graph Matching

5.2.1 Overview

Graph Matching is a fundamental topic in graph theory. Let $G = (V, E)$ be a graph with vertex set V and set of undirected edges E without multi-edges or self-loops. A matching of G is a subset $M \subset E$ so that no two edges of M are adjacent. A vertex incident to an edge of M is called *matched*. A vertex that is not incident to an edge of M is called *free*.

In the past, an enormous amount of work was done in matching theory. Different types of matchings were discussed, their existence and properties were analyzed and efficient algorithms for the calculating of specific matchings were developed. Many results were achieved for specific classes of graphs such as bipartite or planar graphs.

A central aspect in matching theory are matchings with a high cardinality. A *Maximal Matching* M_{MAX} is a matching which cannot be enlarged by an additional edge without violating the matching property (Fig. 5.6, left). A graph may have several different maximal matchings and, especially, maximal matchings of different cardinality. A *Maximum Cardinality Matching* M_{MCM} is a matching of maximum size, i. e. for all matchings $\bar{M}$ of G it holds $|M_{\mathsf{MCM}}| \geq |\bar{M}|$ (Fig. 5.6, center).

Matchings are also discussed for graphs with edge weights $w : E \to \mathbb{R}$. For a set $F \subset E$ let $W(F) := \sum_{\{a,b\} \in F} w(\{a, b\})$ be the weight of F. A *Maximum Weighted Matching* M_{MWM} is a matching of highest weight, i. e. for all matchings $\bar{M}$ of G it holds $W(M_{\mathsf{MWM}}) \geq W(\bar{M})$ (Fig. 5.6, right).

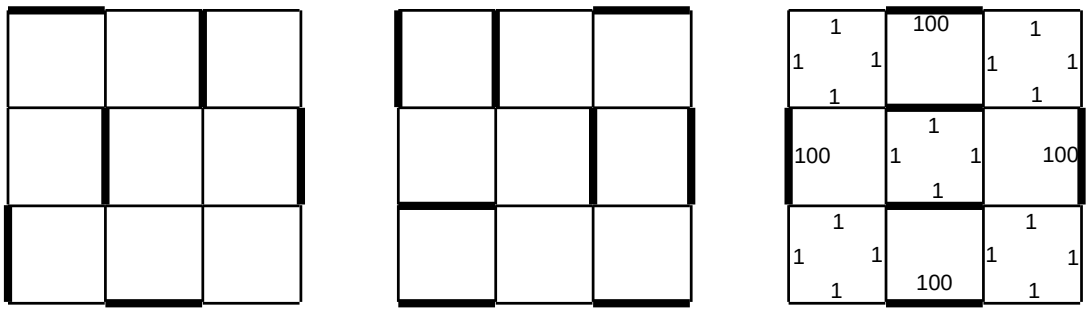


Fig. 5.6: A Maximal Matching M_{MAX} (left), a Maximum Cardinality Matching M_{MCM} (center) and a Maximum Weighted Matching M_{MWM} (right) of a 4×4 square grid with edge weights.

Many algorithms for the calculating of matchings were developed. Consult e. g. [Law76, LP86] for the history of matching algorithms. In the following, only the currently fastest algorithms are stated.

The so far fastest algorithm for maximum cardinality matching is introduced by Micali and Vazirani [MV80, Vaz94]. The algorithm has a time complexity of $O(|E|\sqrt{|V|})$. In the edge-weighted case, the algorithm by Gabow [Gab90] calculates a maximum weighted matching in time $O(|V| \cdot |E| + |V|^2 log(|V|))$. This time complexity was improved by Gabow and Tarjan [GT91] considering the assumption of integral costs which are not particularly high: if the weight function $w : E \to [-N, ..., N]$ assigns to only integers between $-N$ and N, their algorithm will run in $O(\sqrt{|V| \cdot \alpha(|E|, |V|) \cdot log(|V|)} \cdot |E| \cdot log(|V| \cdot N))$ time, with α being the inverse of Ackermann's function.

All algorithms discussed so far have a super-linear time complexity. Recently, approximation algorithms for matching problems attracted more and more attention. They have a smaller time complexity than an optimal algorithm and calculate suboptimal solutions. The guaranteed quality is described by an approximation factor which states the worst case loss to an optimal solution, e. g. a factor of $\frac{1}{2}$ guarantees that the quality of solution is at least $\frac{1}{2}$ the value of the optimal solution. It is a simple exercise to prove that any maximal matching M_{MAX} has a cardinality of at least $\frac{1}{2}$ the cardinality of a maximum cardinality matching, i. e. $|M_{\mathsf{MAX}}| \geq \frac{1}{2}|M_{\mathsf{MCM}}|$. Therefore, any algorithm for a maximal matching is a $\frac{1}{2}$-approximation algorithm for maximum cardinality matching. Simple methods with time complexity $O(|E|)$ can be used to calculate maximal matchings.

Augmenting paths are often considered for graph matching, especially for approximating maximum cardinality matching. An augmenting path has an odd number of edges with alternating edges of M and of $E \backslash M$ and two free vertices as endpoints, i. e. an augmenting path of length l consists of $\frac{l-1}{2}$ edges of M and $\frac{l+1}{2}$ edges of $E \backslash M$. If there is such a path, the cardinality of M can be increased by one. This can be done by exchanging the matched and unmatched edges of the path. According to the work by Hopcroft and Karp [HK73],

it can be shown that if the shortest augmenting path with respect to a matching M_l is l, $|M_l| \geq \frac{l-1}{l+1}|M_{\mathsf{MCM}}|$ (see e. g. [KR98], p.156). Matchings without any short augmenting paths can be calculated very fast, e. g. a matching with a shortest path of length $l \geq 5$ can be computed in time $O(|E|)$. The result is $|M_5| \geq \frac{2}{3}|M_{\mathsf{MCM}}|$. If the minimum degree min and maximum degree max of all vertices are considered, it can easily be proven that if the shortest augmenting path has a length of $l \geq 5$, $|M_5| \geq \frac{min}{max+2 \cdot min}|V|$, which implies $|M_5| \geq \frac{1}{3}|V|$ for graphs with a regular degree. I. e. $\frac{2}{3}$ of the vertices are matched.

For graphs with weighted edges, the GREEDY-algorithm of Fig. 5.7 calculates a matching M_{GREEDY} with weight $W(M_{\mathsf{GREEDY}}) \geq \frac{1}{2}W(M_{\mathsf{MWM}})$ [Avi83]. The algorithm requires a time of $O(|E| \cdot log(|V|))$ due to the initial sorting of the edges by their weight. If the multilevel strategy is used for an initial graph without edge weights, the edge weights of the coarse graphs cannot exceed the number $|E|$ of edges of the initial graph. Then, a count-based sorting algorithm would ensure a $O(|E|)$ time for GREEDY at each level ($|E|$ from the initial graph), although standard algorithms might be more efficient for the intermediate and coarsest graphs. The GREEDY algorithm is used in [Gup97] for calculating matchings after the first few coarsening steps.

GREEDY-Algorithm

$M_{\mathsf{GREEDY}} := \emptyset$;
Sort the edges according to their weight;
WHILE ($E \neq \emptyset$)
    take an edge $\{a, b\} \in E$ with highest weight;
    add $\{a, b\}$ to M_{GREEDY};
    remove all edges incident to a or b from E;
ENDWHILE

Fig. 5.7: GREEDY: $\frac{1}{2}$-approximation algorithm for maximum weighted matching in $O(|E| \cdot log(|V|))$.

An experimental study of several matching algorithms is given in [Mag98]. An overview of parallel algorithms for graph matching problems is presented in [KR98] and new parallel approximation algorithms are presented in [UC00].

5.2.2 Matching Algorithms used for Multilevel Graph Partitioning

As stated in Section 5.1, a matching algorithm used for multilevel partitioning is supposed to be both fast and to have a high matching cardinality and matching weight. Because of the time constraints, the calculation of a M_{MCM} or even a M_{MWM} would be too time

consuming. Therefore, fast algorithms for calculating maximal matchings are used. We describe some of them in the following.

They all follow the same strategy. They start with an initial empty matching and the vertices of the graph are visited in a specific order. It is checked for each visited vertex v if it is free and if v is adjacent to at least one free vertex. If v is free and all neighbors have already been matched, v remains free. Otherwise, if v is free and if there is at least one free neighbor, the edges to free neighbors are rated and an edge with highest rating is added to the matching. The methods differ in the order in which the vertices are visited and the rating of the incident edges. Additionally, the strategies may also differ in the way in which possible ties in the ordering and rating are broken.

As a first example, the *random-edge* matching (REM) is used in e. g. [HL95b]. The vertices are visited in random order. A random free neighbor is chosen and the connecting edge is added to the matching. In [KK99a, KK98c], the random matching and the *heavy-edge* matching (HEM) (with the *modified heavy-edge* matching (MHEM) as Tie-breaking mechanism), the *light-edge* matching (LEM) and the *heavy-clique* matching (HCM) are discussed. All of them visit the vertices in random order and either take any incident edge with highest weight (HEM), take the edge with lightest weight (LEM), or take the edge with the highest edge-density $\frac{2(ce(u)+ce(v)+w(\{u,v\}))}{(w(u)+w(v))(w(u)+w(v)-1)}$ ($ce(x)$ denotes the weight of the collapsed edges in x and $w(x)$ the vertex- or edge-weight of x).

The REM and HEM algorithms are also used in [Gup97]. Furthermore, the GREEDY algorithm of the previous section is used after a few coarsening steps. In the same publication a *heavy-triangle* matching (HTM) strategy is proposed to collapse three vertices into a super-vertex in each coarsening step.

In the Sequential Graph Contraction (SGC) [PMCF94], each vertex gets an initial weight that is equal to its degree. In each level, the vertices are visited in the order of increasing weight. An incident edge of highest weight that connects this vertex to a free vertex is added to the matching.

In [Bou98], the strategies *gain-vertex* matching (GVM) and *closest-vertex* matching (CVM) are presented and compared to REM and HEM. Both, again, visit the vertices in random order. In GVM, an edge which leads to the smallest weight sum over all edges incident to the combined super-vertex is chosen. In CVM, an edge is chosen that leads to the closest free vertex with respect to the geometric distance.

A first approach to analyze the matching strategies REM and HEM on several assumptions was made in [KK95]. Recently, the work in [KK98b] proposes a *balanced-edge* matching to result in a coarse graph with more uniform vertex weights.

All strategies discussed so far are fast and aim to calculate a matching with high cardinality and weight. Their runtime is $O(|E|)$, except $O(|V| \cdot deg_{max})$ for SGC. However, these algorithms do only guarantee a maximal matching. As discussed above, a maximal matching guarantees a cardinality of the matching with at least $\frac{1}{2}|M_{\mathsf{MCM}}|$. However, for the

matching weight, we can construct examples for which these methods calculate matchings with a weight much lower than that of a M_{MWM}.

Many algorithms for clustering data were developed in the past. Consult [JD88, KR90] for an overview. An interesting edge-weighting is the *Unweighted Pair Group Method using Arithmetic Averages* (UPGMA) which assigns the similarity measure $\frac{w(\{u,v\})}{w(u)\cdot w(v)}$ to each edge between vertices u and v. This measure is interesting for the contraction process in the multilevel context. In Section 5.1, we motivated to collapse vertices that are joined with an edge of a high weight. This is not always favorable. On the one hand, if there are large differences between the weights of the vertices, an edge weight of x between two vertices with small vertex weights provides us with a strong argument to collapse these two vertices. On the other hand, an edge weight of the same value x between two vertices with a high weight provides us with a much weaker argument to collapse these two vertices. Therefore, the weight of the vertices should be taken into account. The measure $\frac{w(\{u,v\})}{w(u)\cdot w(v)}$ is the relation of the existing edge-weight between vertices u and v and the maximum possible edge-weight, i. e. it is a valuation of the edge-weight with respect to the vertex weights. Therefore, we can use this measure as the new edge weight and we can use any of the previously described matching algorithms. Thus, the collapsing of the vertices is more natural. Furthermore, this measure benefits vertices with a small weight. These are more likely to take part in the matching process. As a result, the number of coarsening steps to reach a certain limit of vertices is small.

5.2.3 Linear Time $\frac{1}{2}$-Approximation Algorithm for Maximum Weighted Matching in General Graphs

A new approximation algorithm LAM for maximum weighted matching in general edge-weighted graphs is presented. The algorithm calculates a matching with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching. The time complexity is $O(|E|)$, with $|E|$ being the number of edges in the graph. This improves upon the previously known $\frac{1}{2}$-approximation algorithms for maximum weighted matching which require $O(|E| \cdot log(|V|))$ steps, with $|V|$ being the number of vertices [Avi83].

LAM is implemented in the graph partitioning library PARTY [MPD00] and is compared to other matching heuristics on a number of large graphs from real applications by Birger Boyens in [Boy98].

The new algorithm is similar to the GREEDY algorithm (Fig. 5.7). It provides us with the same quality of approximation, but has only a time complexity of $O(|E|)$. The new algorithm LAM is described in the following sections and we now state the new theorem.

Theorem 6: *Let $G = (V, E)$ be a graph with vertices V and weighted undirected edges E. A matching M_{LAM} of G with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching can be computed in linear time $O(|E|)$.*

Proof: Lemma 11 (Section 5.2.3.2) shows that the algorithm LAM of Fig. 5.10 (Section 5.2.3.1) calculates a matching M_{LAM} with an edge weight of at least $\frac{1}{2}$ the edge weight of a maximum weighted matching. Lemma 12 (Section 5.2.3.3) shows the time complexity of $O(|E|)$. ⋈

According to corollary 4 (Section 5.2.3.1), the matching M_{LAM} computed by LAM is also maximal. As stated above, $|M_{\mathsf{MAX}}| \geq \frac{1}{2}|M_{\mathsf{MCM}}|$. The result is the following corollary.

Corollary 3: *The matching M_{LAM} computed by algorithm* LAM *has a weight of at least $\frac{1}{2}$ the weight of a maximum weighted matching and also a cardinality of at least $\frac{1}{2}$ the cardinality of a maximum cardinality matching.*

5.2.3.1 The Algorithm

Fig. 5.8 outlines the new LAM algorithm. It starts with an empty matching M_{LAM} and repeatedly adds an edge $\{a, b\}$ to M_{LAM}. All edges incident to a or b are removed, because they cannot be part of the final matching. The key idea of the algorithm is to add *locally heaviest edges* to the matching.

Definition 9 (locally heaviest edge): *An edge $\{a, b\}$ is called* locally heaviest edge *if its weight is at least as high as the weight of all adjacent edges in E, i. e. $w(\{a, b\}) \geq w(\{x, y\})$ for any $\{x, y\} \in E$ with $a = x$ or $b = x$.*

Outline of LAM-Algorithm

$M_{\mathsf{LAM}} := \emptyset$;
WHILE ($E \neq \emptyset$)
    take a locally heaviest edge $\{a, b\} \in E$;
    add $\{a, b\}$ to M_{LAM};
    remove all edges incident to a or b from E;
ENDWHILE

Fig. 5.8: Outline of LAM: Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching.

After a locally heaviest edge was removed from E, further edges may become locally heaviest. Notice that there is always at least one locally heaviest edge. The main problem

is to find such an edge. Fig. 5.9 displays the idea. The algorithm starts with an arbitrary edge and checks the remaining adjacent edges. As long as an adjacent edge with higher weight can be found, the algorithm switches to the new edge and repeats the checking procedure until a locally heaviest edge is found, i. e. the weight increases along the path.

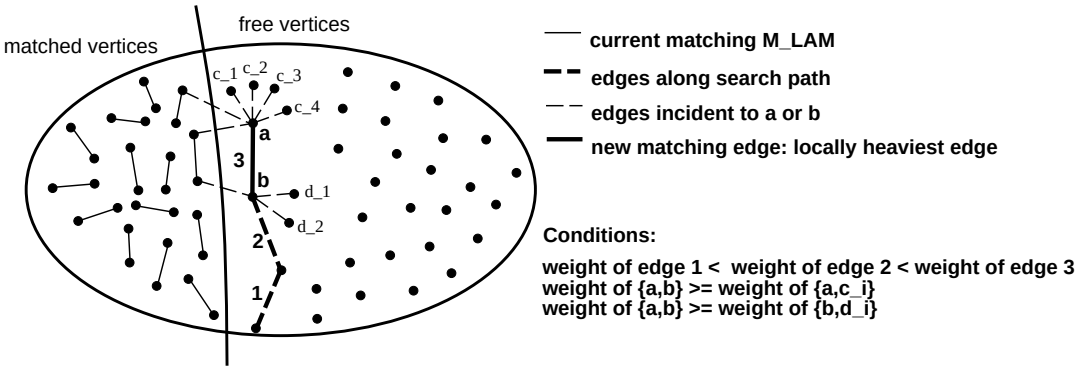


Fig. 5.9: The path starts with edge 1 and progresses along edges 2 and 3 with a higher weight until a locally heaviest edge $\{a, b\}$ is found. Edges of the current matching M_{LAM}, edges along the path and edges adjacent to $\{a, b\}$ are shown.

The detailed algorithm LAM is shown in Fig. 5.10. It starts with an empty matching M_{LAM}. The global sets U and R store all unchecked edges ($U = E$ at the start) and the removed edges ($R = \emptyset$ at the beginning).

The main algorithm is a WHILE loop which calls the procedure 'try match' with an arbitrary, unchecked edge $\{a, b\}$. This edge is not added to the matching until all adjacent edges to free vertices are checked for a higher weight. This action is managed in the WHILE part of the procedure. Every call of the procedure 'try match $(\{a, b\})$' stores its own sets of locally checked edges $C_{\{a,b\}}(a)$ and $C_{\{a,b\}}(b)$, depending on whether the checked edges are incident to a or b. Let C be the union of all locally checked edges, i. e. $C := \bigcup_{\{a,b\} \in E} C_{\{a,b\}}(a) \cup C_{\{a,b\}}(b)$ (we show in the following that 'try match' is called not more than once for every edge). As long as a and b are free and at least one of them is incident to an unchecked edge, it is checked. If it has a higher weight, 'try match' calls itself recursively with the new edge. Recursive calls are repeated until a locally heaviest edge is found. Such an edge is added to M_{LAM}, the algorithm terminates the current call of 'try match', tracks back the search path by one edge and continues the WHILE loop by checking further adjacent edges.

The WHILE loop terminates, if vertices a and/or b were matched in a recursive call or if there is no further adjacent unchecked edge. In the IF-ELSE part, $\{a, b\}$ is added to M_{LAM}, if a and b are free. Additionally, edges that are incident to matched vertices and checked in the current call are removed. I. e. they are moved from C to R. If a or b remain free, all edges that were checked in the current call and are incident to two free

LAM-Algorithm

```
M_LAM := ∅;                                  /* empty matching at the start */
U := E;                                      /* all edges are unchecked at the start */
R := ∅;                                      /* no removed edges at the start */
WHILE (U ≠ ∅)
   take arbitrary edge {a,b} ∈ U;
   try match ({a,b});
ENDWHILE


PROCEDURE try match ({a,b})
C_{a,b}(a) := ∅;                             /* empty local sets of checked edges at the start */
C_{a,b}(b) := ∅;

WHILE (a is free AND b is free AND (∃{a,c} ∈ U OR ∃{b,d} ∈ U))
   IF (a is free AND ∃{a,c} ∈ U)
      move {a,c} from U to C_{a,b}(a);                          /* move from U to C */
      IF (w({a,c}) > w({a,b}))
         try match ({a,c});                                     /* call heavier edge */
      END-IF
   END-IF
   IF (b is free AND ∃{b,d} ∈ U)
      move {b,d} from U to C_{a,b}(b);                          /* move from U to C */
      IF (w({b,d}) > w({a,b}))
         try match ({b,d});                                     /* call heavier edge */
      END-IF
   END-IF
ENDWHILE

IF (a is matched AND b is matched)
   move edges C_{a,b}(a) and C_{a,b}(b) to R;                   /* move from C to R */
ELSE IF (a is matched AND b is free)
   move edges C_{a,b}(a) and {{b,d} ∈ C_{a,b}(b)| d is matched} to R;  /* move from C to R */
   move edges {{b,d} ∈ C_{a,b}(b)| d is free} back to U;        /* move from C back to U */
ELSE IF (b is matched AND a is free)
   move edges C_{a,b}(b) and {{a,c} ∈ C_{a,b}(a)| c is matched} to R;  /* move from C to R */
   move edges {{a,c} ∈ C_{a,b}(a)| c is free} back to U;        /* move from C back to U */
ELSE /* a is free AND b is free */
   move edges C_{a,b}(a) and C_{a,b}(b) to R;                   /* move from C to R */
   add {a,b} to M_LAM;                                  /* new matching edge {a,b} */
END-IF
```

Fig. 5.10: LAM: Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching.

vertices are unchecked. I. e. they are moved back from C to U. Lemma 12 shows that only a limited number of times edges are moved back from C to U.

Notice that the WHILE loop alternately checks adjacent edges that are incident to a and incident to b, as long as both types are available. This property will be used in Section 5.2.3.3 to show the linear time requirement. Additionally, removed edges cannot take part in the following matching process. However, we need to keep track of how many edges are removed in every part of the algorithm. Unlike the outline of the algorithm in Fig. 5.8, edges that are incident to matched vertices are not immediately removed, but they are removed in those procedure calls of the algorithm where they have been checked for the last time.

It is fairly obvious that the unchecked edges in U have the property of both incident vertices being free. This fact holds true for the start. New edges are only moved back to U in the IF-ELSE part, but only if they are incident to two free vertices. Furthermore, an edge $\{a, b\}$ can only be matched at the end of the IF-ELSE part. In this case, a and b have to be free, and the WHILE loop terminated because there was no edge $\{a, c\}$ or $\{b, d\}$ in U, i. e. all edges in U keep the property of being incident to two free vertices.

The central part of the algorithm is the procedure 'try match'. The following lemma shows the number of times it is called:

Lemma 8 ($|E|$ calls): *The Procedure* 'try match' *is called not more than once for any edge.*

Proof: The procedure 'try match' is only called with an edge $\{a, b\}$ from U. This fact ensures that both vertices are free. The same edge cannot be the parameter in deeper recursive calls, because the search path does only progress along strictly higher edge weights.

Finally, in the IF-ELSE part of 'try match', either a and/or b have already been matched or they are matched by adding edge $\{a, b\}$ in M_{LAM}. Therefore, after the first call of 'try match($\{a, b\}$)', a and/or b are matched, i. e. $\{a, b\}$ cannot be in U anymore and cannot be called again. ⋈

The following lemma shows the status of the edges.

Lemma 9 (status of the edges): *At the start, all edges are not checked ($U = E$), and there are no checked or removed edges ($R = C = \emptyset$). At any stage, an edge $\{a, b\} \in E$ is either unchecked ($\in U$), checked ($\in C$) or removed ($\in R$), resulting in $E = U \cup C \cup R$. In the end, all edges are removed, i. e. $U = C = \emptyset$ and $R = E$.*

Proof: The status at the start becomes obvious with the help of the algorithm. The edges are only moved between the sets U, C and R. This fact ensures the above stated status throughout the algorithm. The WHILE loop of the main algorithm terminates when U is empty.

Besides, all edges $C_{\{a,b\}}(a)$ and $C_{\{a,b\}}(b)$ which were moved in the WHILE loop of procedure call 'try match $(\{a,b\})$', are either moved to R or moved back to U in the IF-ELSE part of the same call. Therefore, C is also empty after the termination of all 'try match' calls. This results with $E = U \cup C \cup R$ in $E = R$. ⋈

Lemma 9 ensures that all edges are removed at the end of the algorithm, because at least one of the incident vertices were matched. Thus, we can state the following corollary.

Corollary 4 (maximal): *The resulting matching M_{LAM} is maximal.*

5.2.3.2 $\frac{1}{2}$-Approximation Quality

In this section, we show the property of a locally heaviest edge for any edge added to the matching and the $\frac{1}{2}$-approximation. A similar proof was given in [Avi83] to show the $\frac{1}{2}$-approximation for the GREEDY algorithm of Fig. 5.7. As intuition, the algorithm does only add locally heaviest edges to the matching. Each of these may block at most two edges of equal or smaller size of an arbitrary maximum weighted matching.

Lemma 10 (locally heaviest edge): *Algorithm* LAM *starts with an empty matching. An edge $\{a,b\}$ is only added if a and b are free and neither a nor b are adjacent to a free vertex with an edge of higher weight than $\{a,b\}$.*

Proof: An edge $\{a,b\}$ is only added to M_{LAM} in the last ELSE part of 'try match', i. e. a and b are free. Let a be adjacent to a free vertex c (or b adjacent to a free vertex d). According to Lemma 9, edge $\{a,c\}$ ($\{b,d\}$) may be

in R : Impossible, because in this case either a or c have already been matched.

in U : Impossible, because in this case the WHILE loop would not have been terminated.

in C : The weight of $\{a,b\}$ ($\{b,d\}$) is higher than the weight of all other edges in the search path. When edges were checked along the search path, either their weight was not higher than the weight of the corresponding edge of the path, or the search path progressed along this edge. In the later case, either the recursive call terminated with at least one incident vertex of that edge being matched, or the edge is still in the search path.

⋈

We are now ready to prove the $\frac{1}{2}$-approximation which is illustrated in Fig. 5.11.

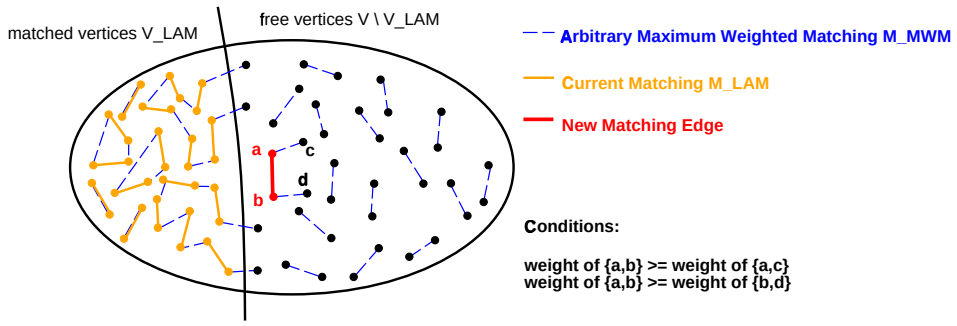


Fig. 5.11: Proof of the $\frac{1}{2}$-approximation of the LAM algorithm.

Lemma 11 ($\frac{1}{2}$-approximation): *Algorithm* LAM *computes a matching* M_{LAM} *with at least* $\frac{1}{2}$ *of the edge weight of a maximum weighted matching* M_{MWM}.

Proof: Compare M_{LAM} with an arbitrary matching M_{MWM}. Let V_{LAM} be the set of matched vertices of the current M_{LAM} and let V_{MWM} be the set of matched vertices of M_{MWM}. Fig. 5.11 illustrates the situation. We show that the weight of the current matching M_{LAM} during the algorithm is at least $\frac{1}{2}$ of the weight of the edges of M_{MWM} incident to a vertex of V_{LAM}:

$$W(M_{\mathsf{LAM}}) \geq \frac{1}{2} W(\{\{u,v\} \in M_{\mathsf{MWM}} | u \in V_{\mathsf{LAM}} \vee v \in V_{\mathsf{LAM}}\})$$

This fact holds for the beginning ($M_{\mathsf{LAM}} := \emptyset$). After having added an edge $\{a,b\}$ to M_{LAM}, $W(M_{\mathsf{LAM}})$ increases by $w(\{a,b\})$. However, the right hand side of the equation may increase, too.

If $\{a,b\} \in M_{\mathsf{MWM}}$, the right hand side only increases by $\frac{1}{2}w(\{a,b\})$. Otherwise, let $\{a,c\},\{b,d\} \in M_{\mathsf{MWM}}$ be the possible edges adjacent to $\{a,b\}$. The choice of matching edge $\{a,b\}$ excluded the possible choice of $\{a,c\}$ and $\{b,d\}$ throughout the rest of the algorithm. These are the only two edges by which the subset of M_{MWM} may increase. I. e. the right hand side may only increase by $\frac{1}{2}(w(\{a,c\})+w(\{b,d\}))$.

If $c \in V_{\mathsf{LAM}}$ ($d \in V_{\mathsf{LAM}}$) before we add edge $\{a,b\}$, $\{a,c\}$ ($\{b,d\}$) has already been the subset of M_{MWM}. If $c \notin V_{\mathsf{LAM}}$ ($d \notin V_{\mathsf{LAM}}$), i. e. c (d) is free, Lemma 10 ensures that $w(\{a,b\}) \geq w(\{a,c\})$ ($w(\{a,b\}) \geq w(\{b,d\})$). Therefore, the value on the right hand side cannot increase by more than $w(\{a,b\})$.

The algorithm LAM terminates with a maximal matching M_{LAM} (Corollary 4), i. e. for all edges $\{u,v\}$ is $u \in V_{\mathsf{LAM}}$ or $v \in V_{\mathsf{LAM}}$. Therefore,

$$W(M_{\mathsf{LAM}}) \geq \frac{1}{2} W(\{\{u,v\} \in M_{\mathsf{MWM}} | u \in V_{\mathsf{LAM}} \vee v \in V_{\mathsf{LAM}}\}) = \frac{1}{2} W(M_{\mathsf{MWM}}) \ .$$

⋈

5.2.3.3 Linear Time Requirement

The time requirement is kept linear because the search for locally heaviest edges is performed in a backtracking manner, i. e. after a locally heaviest edge has been found, some of the adjacent edges are removed and the search for a new edge continues with the previously considered edges along the search path.

The time requirement depends on the number of times edges being moved between the sets U of unchecked, C of checked and R of removed edges. There are only three ways of moving an edge: (1) in the WHILE loop of 'try match', previously unchecked edges are checked (moved from U to C), in the IF-ELSE part, the checked edges are either (2) removed (moved from C to R), or (3) unchecked again (moved from C to U). Therefore, once an edge is removed and stored in R, it will not be moved to any other set. This fact ensures that an edge may be moved from C to R at most $|E|$ times. Furthermore, an edge may be moved several times between U and C,. However, we show that every time edges are moved back from C to U, an almost equal number of edges is moved from C to R. Thus, we show that the number of edges that are moved between U and C is $O(|E|)$.

Lemma 12 (linear time): *Algorithm* LAM *of Fig. 5.10 runs in* $O(|E|)$ *time.*

Proof: The loop of the main algorithm has at most $|E|$ iterations, because if it makes a call 'try match $(\{a, b\})$' with an edge $\{a, b\} \in U$, at least one of a and b are matched after the completion of the call. As stated in Section 5.2.3.1, an unchecked edge in U is always incident to two free vertices. Therefore, U is reduced by at least edge $\{a, b\}$ in every iteration of the WHILE loop of the main algorithm. Although new edges may be added to U in the IF-ELSE part of 'try match', these edges were previously moved in the WHILE loop of the same procedure. According to Lemma 8, the 'try match' procedure is not called more often than once for every edge $\{a, b\} \in E$. To sum it all up, the time complexity of the algorithm is

$$O(|E|) + \sum\nolimits_{\{a,b\}\in E} O(\text{try match } (\{a, b\})) \ .$$

The procedure 'try match$(\{a, b\})$' consists of a WHILE loop and an IF-ELSE part. In every WHILE loop, an edge $\{a, c\}$ and/or $\{b, d\}$ is checked and moved from U to the local sets $C_{\{a,b\}}(a)$ and/or $C_{\{a,b\}}(b)$ of checked edges. This fact results in a time of $O(|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)|)$ for the WHILE loop. The values $|C_{\{a,b\}}(a)|$ and $|C_{\{a,b\}}(b)|$ refer to the maximum sizes of the sets $C_{\{a,b\}}(a)$ and $C_{\{a,b\}}(b)$, which occur after the completion of the WHILE loop. In the IF-ELSE part, the edges of the local sets $C_{\{a,b\}}(a)$ and $C_{\{a,b\}}(b)$ are either removed (moved to R), or they are unchecked again (moved to U), which again leads us to a time of $O(|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)|)$. Thus, the run time of the algorithm is

$$O(|E|) + \sum\nolimits_{\{a,b\}\in E} O(|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)|). \tag{5.1}$$

We show that the number of check operations for every call is not much larger than the number of remove operations. Let $R_{\{a,b\}}$ be the set of edges removed in the procedure call 'try match($\{a,b\}$)', i. e. $\sum_{\text{try match }(\{a,b\})} R_{\{a,b\}} = R$. We distinguish the two cases of the IF-ELSE part.

1. a **and** b **are matched;** a **and** b **are free:** In both cases the sizes of sets $C_{\{a,b\}}(a)$ and $C_{\{a,b\}}(b)$ is equal to the number of removed edges, i. e.

$$|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)| = |R_{\{a,b\}}| \; . \tag{5.2}$$

2. a **is matched,** b **is free:** In this case, a was matched with an adjacent vertex c in a recursive call of 'try match'. It may be matched in a recursive call just one level lower as it is shown in Fig. 5.12(i). However, it may also be matched in a lower level of recursion. This could occur as the result of a loop in the search path as shown in Fig. 5.12(ii). Additionally, it may be matched in a recursive call made from vertex b (shown in Fig. 5.12(iii)).

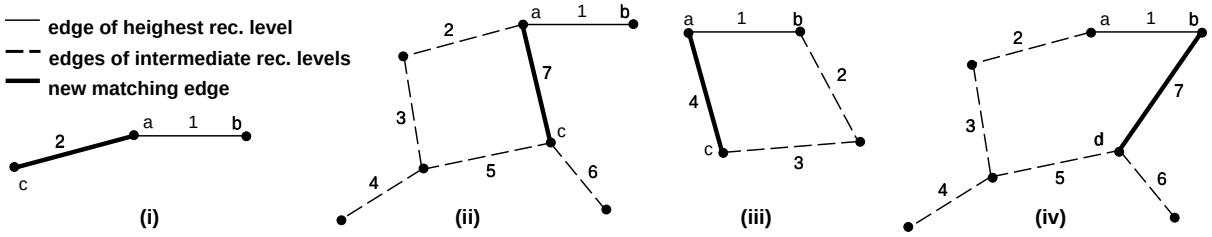


Fig. 5.12: Vertices a or b are matched in a recursive call. Edges of consecutive recursive calls are shown. Numbers indicate the level of recursion, i. e. the edge weight increases according to the numbers. (i): a is matched in a recursive call from itself one level lower. (ii): a is matched in a recursive call from itself several levels lower, forming a loop. (iii): a is matched in a recursive call from b. (iv): b is matched in a recursive call from a.

We show that the number of uncheck operations is not larger than the number of remove operations. Let us assume the WHILE loop terminates after i iterations, i. e. a and c are free at the beginning of every iteration and edge $\{a,c\}$ matched in the i-th iteration. We have to show that an edge $\{a,c_i\} \in U$ was checked in every iteration i.

According to Lemma 9, $\{a,c\}$ is either in U, C or R at any stage. This is especially true at the beginning of every iteration of the WHILE loop of the procedure call 'try match $(\{a,b\})$'. If $\{a,c\}$ is checked, it may either be checked from the procedure 'try match $(\{a,b\})$', or from an earlier procedure call in the search path. When edges were checked along the search path, either their

weight was not higher than the weight of the corresponding path edge, or the search path progressed along this edge. In the later case, either the recursive call terminated with at least one incident vertex of that edge being matched, or the edge is still in the search path. Therefore, if $\{a, c\}$ is checked, it cannot have a higher weight than $\{a, b\}$. However, this situation is impossible, because it was matched in iteration i while being the final edge in the search path and $\{a, b\}$ being within the search path. Furthermore, $\{a, c\}$ can not be removed either, because in this case either a or c is matched.

Therefore, it is $\{a, c\} \in U$ at the start of every iteration, i. e. the condition of the first IF condition in the WHILE loop was true in every iteration and an edge $\{a, c_i\}$ is checked and moved to $C_{\{a,b\}}(a)$, i. e. $|C_{\{a,b\}}(a)| = i$.

Additionally, it is obvious that $|C_{\{a,b\}}(b)|$ cannot be more than the number of iterations, i. e. $|C_{\{a,b\}}(b)| \leq i$. This results in

$$|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)| \leq 2|C_{\{a,b\}}(a)| \leq 2|R_{\{a,b\}}| \; . \tag{5.3}$$

3. b **is matched,** a **is free:** This case is similar to the previous one. However, it is not identical, because a is checked first in the WHILE loop. It may happen that b is matched in a recursive call from a as it is shown in Fig. 5.12(iv). In this case, in the final iteration of the WHILE loop, b has already been matched after the completion of the recursive calls from a and no further edge was added to $C_{\{a,b\}}(b)$. Consequently, we can only guarantee the fact that $i \geq |C_{\{a,b\}}(b)| \geq i - 1$.

   As we have seen in the previous case, $|C_{\{a,b\}}(a)|$ cannot be more than the number of iterations, i. e. $|C_{\{a,b\}}(a)| \leq i$. This results in

$$|C_{\{a,b\}}(a)| + |C_{\{a,b\}}(b)| \leq 2|C_{\{a,b\}}(b)| + 1 \leq 2 \cdot |R_{\{a,b\}}| + 1 \; . \tag{5.4}$$

Equations 5.2, 5.3 and 5.4 reduce the overall time complexity of equation 5.1 to

$$O(|E|) + \sum\nolimits_{\{a,b\} \in E} O(2 \cdot |R_{\{a,b\}}| + 1) = O(|E|),$$

because the total number of removed edges cannot exceed $|E|$. ⋈

5.2.3.4 Heuristic Improvements of *LAM*

Although the LAM algorithm produces a matching with at least $\frac{1}{2}$ of the cardinality and $\frac{1}{2}$ of the weight of the maximum matchings, we describe heuristic improvements, without destroying these lower bounds.

According to our experience, the graphs at intermediate and, especially, at the coarse levels of the multilevel concept are very likely to have several vertices of degree 1. I. e. they

have star-like subgraphs. We consider vertices of degree 1 in a special case and modify the condition of the algorithm in Fig. 5.10, where the procedure 'try match' calls itself recursively with a new edge. We only show the modifications for the first IF-condition with checking for edge $\{a, c\}$. The second IF-condition with checking edge $\{b, d\}$ follows accordingly. The modifications are displayed in Fig. 5.13.

```
IF ((deg(b) > 1 ∧ deg(c) > 1 ∧ w({a,c}) > w({a,b})) OR
    (deg(b) > 1 ∧ deg(c) > 1 ∧ w({a,c}) = w({a,b}) ∧ deg(c) < deg(b)) OR
    (deg(b) > 1 ∧ deg(c) = 1 ∧ w({a,c}) ≥ ½w({a,b})) OR
    (deg(b) = 1 ∧ deg(c) > 1 ∧ w({a,c}) > 2w({a,b})) OR
    (deg(b) = 1 ∧ deg(c) = 1 ∧ w({a,c}) > w({a,b})) )
        try match ({a,c});
END-IF
```

Fig. 5.13: Heuristic improvement of LAM in consideration of vertices of degree 1.

Originally, the procedure calls itself recursively, if the weight of the adjacent edge is higher than the current weight. Now, we choose between edges $\{a, b\}$ and $\{a, c\}$ by their degree. If both have a degree that is larger than 1, we choose edge $\{a, c\}$ if it has a higher weight (as it is in the original algorithm). However, we choose it, too, if the weight is equal and if the degree of c is smaller than the degree of b. Thus, vertices with a lower degree are preferred without any violation of the approximation quality. Furthermore, the vertex with a higher degree is still incident to many edges which may be added to the matching later on.

What is more, if b has a degree that is larger than 1 and if c does only have a degree of 1, edge $\{a, c\}$ will be chosen if its weight is at least $\frac{1}{2}$ the weight of the edge $\{a, b\}$. This ensures that c is not blocked so easily by adding edge $\{a, b\}$ to the matching. The approximation quality remains valid, because in the proof of Lemma 11, the edge $\{a, c\}$ with c being of degree 1 can only violate one matching edge of an unknown maximum weighted matching. In the opposite case, when b has a degree of 1 and c has a degree that is larger than 1, edge $\{a, c\}$ will only be chosen, if its weight is more than twice as high as the weight of $\{a, b\}$. Finally, if both have a degree of 1, we choose edge $\{a, c\}$, if it has a higher weight (as it is the case in the original algorithm).

With the help of these modifications, vertices with a degree of 1 are more likely to be matched as compared to the original algorithm. The matching cardinality and the matching weight increase and the appearance of star-like subgraphs in the graphs of intermediate and coarse levels reduces.

5.3 Multilevel Experiments on the Graph *wave*

We show experimental results both for the new matching algorithm LAM and for a combination of LAM and Helpful-Set with the multilevel strategy. Both methods are implemented in the PARTY graph partitioning library [MPD00, PD97]. For LAM we use the modification described in Section 5.2.3.4 which holds the same approximation ratio and the same time complexity as the original LAM algorithm of Section 5.2.3. However, this modification favors vertices of a smaller degree and, especially, vertices of degree 1.

We show some experiments on the coarsening of the graph *wave* (Fig. 5.14). The results for *wave* are typical of the graphs in the test suite in the following section.

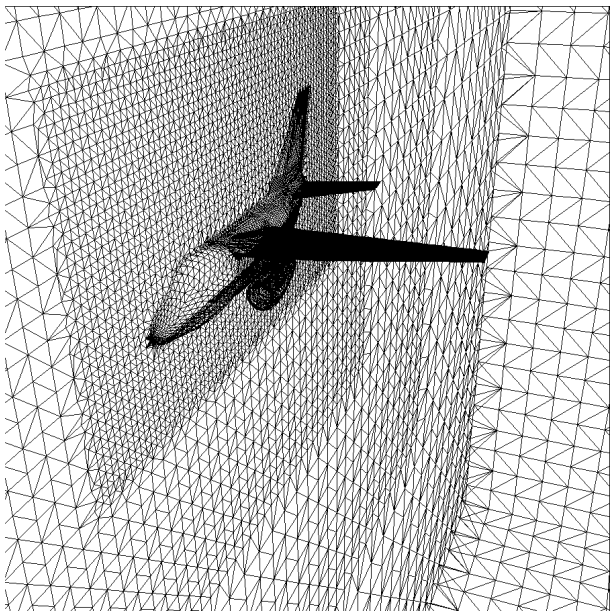


Fig. 5.14: The Graph *wave* is a 3-dimensional discretization of the air around an airplane. Only are the vertices on the boundary of the discretization shown. The Graph has 156,317 vertices and 1,059,331 edges.

We compare the new matching strategy LAM to the random-edge matching (REM), the heavy-edge matching (HEM) and the greedy matching (GRE) described in Section 5.2.

The dominating part of the greedy matching algorithm is the sorting of the edges according to their weight. We tried to use the quick-sort algorithm implemented in the standard C-library. However, since our test graphs do not have edge weights, the choice of the pivot-element works very badly on the large graphs of the first levels such that the used time is far too high to finish the experiments within hours. We could use count-based sorting algorithms with x buckets, if all edge weights are integer values in the range of $[1, x]$. Unfortunately, the maximum edge weight may increase by a factor of four for

each coarsening step. Therefore, the number of buckets may be very high for the coarse graphs. To be more general, we choose a modification of the quick-sort algorithm described in [CLR90](p. 168) with some further modifications. This sorting algorithm runs in time $O(min\{|E| \cdot log(|E|), |E| \cdot N\})$ with E and N being the number of edges and the number of different edge weights of the graphs on each levels. This modified algorithm is very fast on the fine and the coarse graphs. The fine graphs do only have a small number N of different edge weights and, therefore, the value $|E| \cdot N$ is very small and dominates the time requirement. The coarse graphs do only have a few number of edges and, therefore, the time requirement of $|E| \cdot log(|E|)$ is very small for these graphs.

We discuss two modifications to the matching algorithms. Firstly, instead of using the standard weight of the edges, we compute the similarity measure $\frac{w(\{u,v\})}{w(u) \cdot w(v)}$ for each edge $\{u, v\}$ (described in Section 5.2.2). We use the new edge weight as the input to the matching algorithms. In this case, the weights of the incident vertices is taken into account. We denote these modified matching algorithms by the extension '/VW'. Secondly, we can perform a post-processing step for each matching algorithm. This step tries to improve the weight of the matching by searching for short augmenting paths as described in Section 5.2.1. Notice that a matching M_{LAM} computed with the original algorithm cannot have any augmenting path of length 2, but this cannot be ensured for the modified LAM algorithm. In our experiments, we heuristically improve the matchings by searching for augmenting paths with a length of at most 3. We denote these modified algorithms with the extension '+W3'.

We perform the coarsening steps until there are only two remaining vertices, i. e. no further global partitioning method is needed for the coarse graph. In Fig. 5.15-5.19 we reveal results for the matching ratio of $2|M|/|V|$, for the maximum vertex weight $Max(V)$, for the number of edges $|E|$, for the total weight $W(E)$ of all edges and for the cut size. In Tab. 5.1 the main results are listed in more detail and, unlike in the figures, it also shows results for the matching algorithms with some additional post-processing.

The results of the matching ratio are presented in Fig. 5.15. Most algorithms produce matchings with more than 95 % matched vertices for most levels. The major reason for the low values for the last few levels is the fact that if the number of vertices is odd, at least one vertex remains unmatched and significantly reduces this value for small graphs. The key observation in Fig. 5.15 is the fact that GRE and LAM result in a very small matching ratio after some levels. This happens due to the fact that there is a star-like graph, as discussed and shown in Fig. 5.4(right). Thus, after some levels the graph consists of an edge-weighted star with only one matching edge. This reduces the number of vertices by only one for each level. This behavior happens due to the deterministic behavior of the approximation algorithms GRE and LAM. Such a behavior cannot be observed for the algorithms REM and HEM with randomized order of visiting the vertices. Nevertheless, the modified algorithms GRE/VW and LAM/VW reveals that the bad behavior of the original algorithms can be avoided.

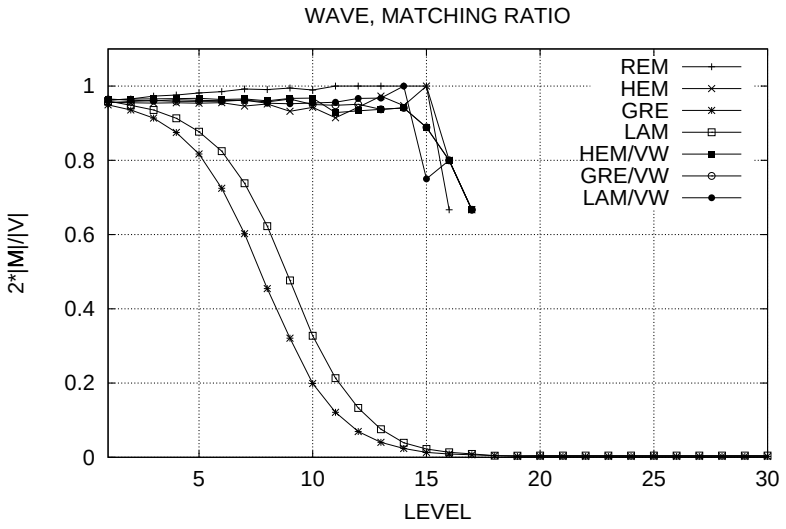


Fig. 5.15: Graph *wave* with the matching ratio as a function of the number of levels.

The facts of the matching ratio correlate with the values for the maximum vertex weights which are presented in Fig. 5.16. Again, most algorithms display an almost linear increase of the maximum vertex weight, due to the high matching ratio. After a couple of levels, the maximum vertex weights for GRE and LAM are almost the total number of initial vertices. As discussed above, these graphs form an edge-weighted star with the center vertex as the set of the majority of the initial vertices. Again, the results show that this bad behavior can be solved by the use of the modified algorithms GRE/VW and LAM/VW. Furthermore, Fig. 5.15 and 5.16 show that the behavior of LAM is slightly better than that of GRE.

Next, we can compare the remaining number of edges in Fig. 5.17. At the intermediate graphs, the number of edges for the REM algorithm are higher by a factor of eight compared to the other algorithms. Again, GRE and LAM behave differently than the other algorithms. However, in this case the behavior is positive, namely the number of edges is smaller as compared to the other algorithms. Besides, the modified algorithms GRE/VW and LAM/VW show a behavior that is similar to HEM and HEM/VW.

A similar behavior can be observed for the remaining edge weight. The results are presented in Fig. 5.18. The graphs constructed by the REM algorithm have a high edge weight due to the fact that the algorithm does not consider any edge weights. The remaining edge weights for HEM and HEM/VW are almost identical. Again, the edge weights of GRE and LAM are much smaller than for the other algorithms. As discussed above, the modified GRE/VW and LAM/VW algorithms behave much better in terms of the number of levels and the balancing of the vertex weights as compared to the original algorithms. The remaining edge weights for the modified versions are higher as compared to the orig-

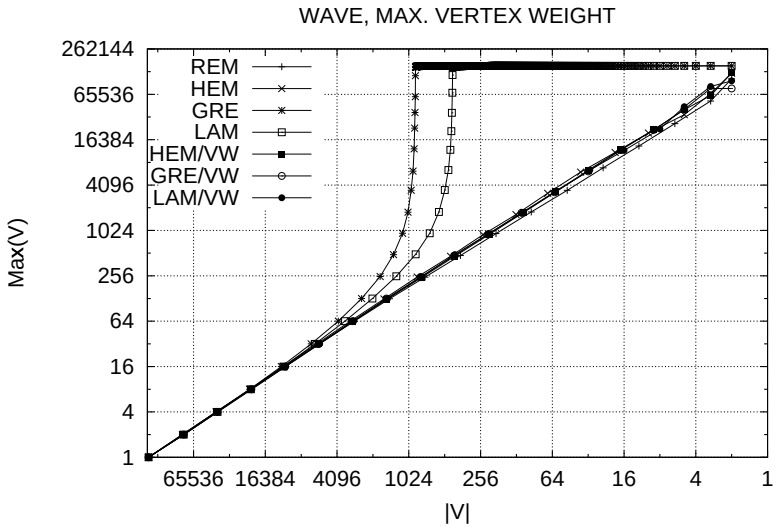


Fig. 5.16: Graph *wave* with the maximum vertex weight as a function of $|V|$.

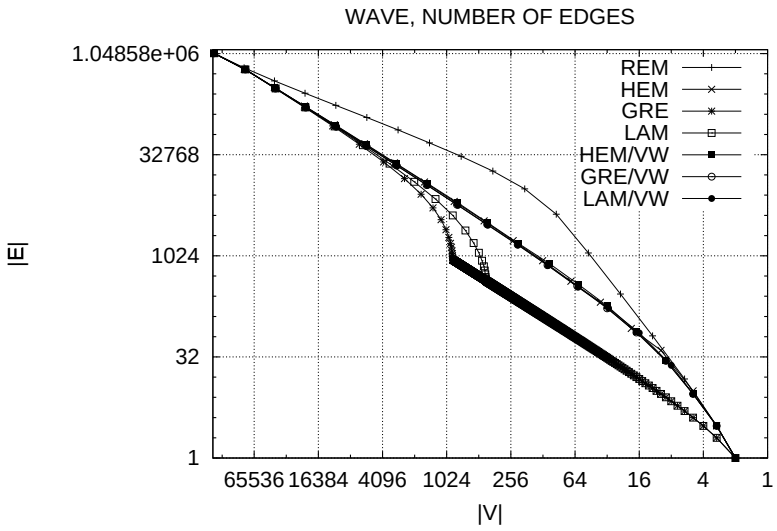


Fig. 5.17: Graph *wave* with $|E|$ as a function of $|V|$.

inal algorithms, but they are consistently smaller as compared to the HEM algorithms. The only exception is the smallest graph with only two vertices, where GRE/VW and LAM/VW have a higher edge weight than HEM. However, we see in the following that the weights of the two remaining vertices is much more balanced for GRE/VW and LAM/VW than for HEM. Therefore, the modified algorithms GRE/VW and LAM/VW seem to be a promising alternative to the commonly used REM and HEM algorithms.

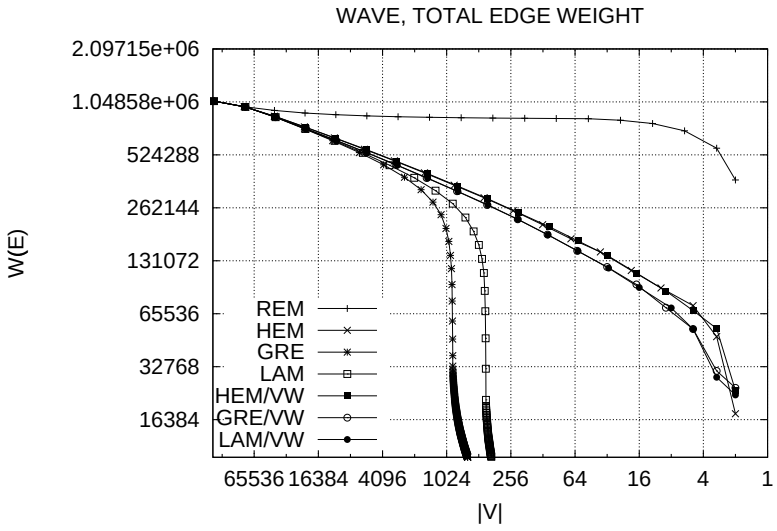


Fig. 5.18: Graph *wave* with the weight of all edges as a function of $|V|$.

Tab. 5.1 presents us with detailed values of the experiments. It includes values for the modified algorithms by using the post-processing +W3 to improve the weight of the matching edges. In addition to the discussed observations there are some further once. E. g. the use of +W3 reduces the number of levels, especially for the original GRE and LAM algorithms. The table includes values for the minimum weight of the two remaining vertices and the weight of the edge between the two vertices. There is a tradeoff between these two measures. We use the measure $\frac{w(\{v_1,v_2\})}{\min\{w(v_1),w(v_2)\}}$ to compare the weights of the smallest graph. Small values ensure a small cut size compared to the balance. It can be observed that the modification /VW leads to small values in this measure.

Tab. 5.1 also presents us with values for the run-time of the matching algorithms, the improvement of the matchings and the reduction of a graph to the next smaller one. These times differ up to a small factor. The GRE algorithms do not only have the highest theoretical run-time (super-linear), but also the highest experimental run-time. The run-times of the LAM algorithms are only slightly higher than of the HEM algorithms. This fact shows that the asymptotic linear time complexity of LAM does not include any large constant factors. Besides, the improvement +W3 of the matchings uses less time than the matching algorithms. Only for the REM algorithm it takes a significant period of time. However, it also significantly improves the weight of the matching in this case. The sequence of graphs is constructed by a reduction of each graph with the help of the current matching to the next smaller graph. The times for doing this is almost equal for all algorithms. It is interesting to observe that the time to reduce the graph is of the same order as the time to calculate the matchings. This fact shows that there is not a high demand for further improvement of the run-time of the matching algorithms unless we

Tab. 5.1: The coarsening of the graph *wave* with different matching algorithms. The graph is coarsened down to two vertices. The values $w(v_1)$ and $w(v_2)$ denote the vertex weights of the two remaining vertices and the value $w(v_1, v_2)$ denotes the weight of the edge between them. The times for the partitioning are in seconds on a Sun Ultra SPARC-II with 400 MHz CPU.

method	Coarsening				Time		
	levels	$\min\{w(v_1), w(v_2)\}$	$w(\{v_1, v_2\})$	$\frac{w(\{v_1,v_2\})}{\min\{w(v_1),w(v_2)\}}$	matching	+W3	reduce
REM	18	51330	377098	7.346	0.89		0.63
HEM	19	31166	17683	0.567	0.40		0.55
GRE	917	1	3	3	1.33		0.64
LAM	457	1	3	3	0.53		0.61
REM+W3	18	62809	284440	4.529	0.81	0.51	0.60
HEM+W3	18	46103	27011	0.586	0.38	0.15	0.50
GRE+W3	34	1	6	6	1.01	0.10	0.48
LAM+W3	34	1	8	8	0.47	0.11	0.52
HEM/VW	19	30274	23941	0.791	0.39		0.53
GRE/VW	19	78067	24831	0.318	1.24		0.51
LAM/VW	19	57414	22625	0.394	0.47		0.54
HEM/VW+W3	18	46982	28857	0.614	0.37	0.09	0.53
GRE/VW+W3	18	59016	27438	0.465	1.14	0.09	0.52
LAM/VW+W3	18	46955	27486	0.585	0.44	0.09	0.54

could reduce the time to coarsen the graphs from one level to the other, too.

The cut size of the partitions of the coarse graphs of GRE and LAM are very low (Fig. 5.19), due to the unbalanced weight distribution on the small graphs. This behavior is not a problematical point, because the local improvement algorithms balance the load of the bisection. After some un-coarsening steps, the bisection is roughly balanced. Again, the modified algorithms GRE/VW and LAM/VW have a much nicer behavior which is similar to the behavior of the HEM algorithms. Besides, the cut sizes of the REM algorithm are very high, due to the fact that it did not consider any edge weights during the coarsening phase. Although all algorithms finally lead us to a final bisection with a cut size of the same order of magnitude, the local improvement heuristics obviously have to work hard for the REM algorithm.

In Fig. 5.19 we used the Helpful-Set heuristic (Sec. 4.2.2) as local improvement method. In Tab. 5.2 we compare it with the Kernighan-Lin heuristic (Sec. 4.2.1) and provide some detailed values for the final cut sizes. The coarsening algorithm LAM/VW+W3, together with the Helpful-Set heuristic, is the default setting of the graph partitioning library PARTY (Chapter 6). A comparison of PARTY with existing available graph partitioning libraries on a number of large graphs is to be found in Section 6.3.

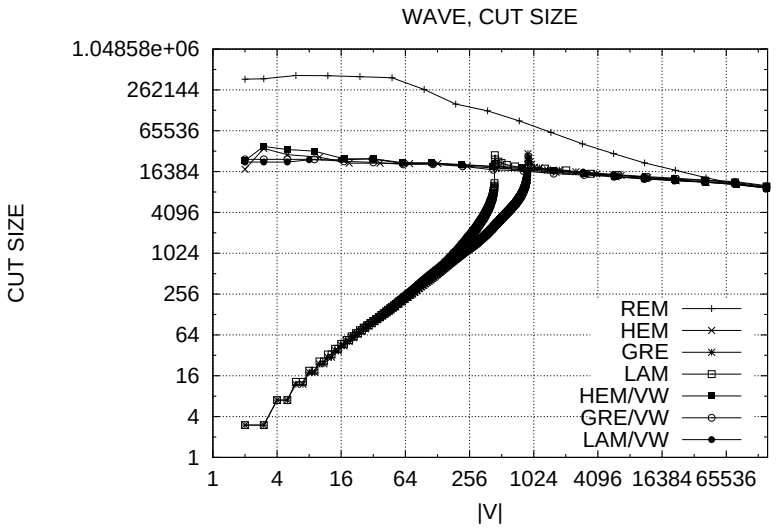


Fig. 5.19: Graph *wave* with the cut size as a function of $|V|$.

Tab. 5.2: Cut sizes and run times for the un-coarsening phase. The bisections are either improved with the KL or the HS heuristics.

method	Coarsening		+KL		+HS	
	Cut	Time	Cut	Time	Cut	Time
REM	377098	1.52	9485	5.65	9301	1.16
HEM	17683	0.95	9381	4.17	9550	0.75
GRE	3	1.97	9733	6.02	9478	6.03
LAM	3	1.14	9480	4.46	9454	1.93
REM+W3	284440	1.41	9643	4.79	9692	2.19
HEM+W3	27011	1.03	9450	4.18	9520	0.71
GRE+W3	6	1.59	9232	3.86	9268	0.70
LAM+W3	8	1.10	9951	4.28	9137	0.69
HEM/VW	23941	0.92	9440	4.11	9345	0.75
GRE/VW	24831	1.75	9610	3.58	9550	0.82
LAM/VW	22625	1.01	9367	3.96	9362	0.79
HEM/VW+W3	28857	0.99	9371	3.81	9346	0.64
GRE/VW+W3	27438	1.75	9484	4.11	9489	0.70
LAM/VW+W3	27486	1.07	9301	4.32	9293	0.64

6. THE PARTY GRAPH PARTITIONING LIBRARY

The efficiency of the graph partitioning heuristics of the Chapters 4 and 5 heavily depend on the quality of the implementation. There are a number of graph partitioning libraries for solving the graph partitioning problem. They offer efficient implementations of several different graph partitioning methods.

Each developer of a heuristic usually has some own piece of code and uses existing graph partitioning libraries for comparison with newly developed heuristics. Furthermore, many developers use their code in conjunction with already implemented methods in the libraries. An example is the multilevel approach, consisting of the coarsening, the global partitioning and the local refinement phase. The developer can concentrate on new solutions for one of these phases and use the implemented solutions for the other phases.

Several applications have their own code for solving the graph partitioning problem. However, these implementations only offer a small variety of heuristics and included only simple methods. The graph partitioning libraries offer a wide range of parameter settings in order to be adjusted to the specific demands of the applications. These libraries can easily be integrated in the applications. Graph partitioning libraries are a simple way to include a large variety of efficient and complex heuristics into the application.

As an example, the graph partitioning library PARTY has been developed by the author of this thesis. It has been used as an experimental platform for developing and testing new heuristics. The default setting of PARTY is the multilevel approach with the LAM algorithm (Section 5.2.3) for the coarsening phase and the Helpful-Set algorithm (Section 4.2.2) for the local refinement phase. Further information and the license agreement can be found at the WWW-page http://www.upb.de/cs/robsy/party.html.

Parts of the work in this chapter were published in [PD97].

6.1 Graph Partitioning Libraries

The goal of the PARTY graph partitioning library, and those of most other libraries, is to provide the user with

- several partitioning heuristics of different character.
- many different combinations of methods.
- efficient implementations of the methods in order to guarantee a high performance.
- a variety of generalizations of the methods to reflect the specific constraints of the application more precisely.
- a simple interface which guarantees an easy use as well as a strong control of the methods.
- validation and information control mechanisms.

As mentioned above, each developer of heuristics has its own piece of code. Many implementations are not intended to be used by people other than the authors. In the following we shortly describe some of the widely used graph partitioning libraries. They have been used in many applications to solve graph partitioning problems. We additionally briefly describe some related publications by the authors of the libraries.

CHACO [HL94a], B. Hendrickson and R. Leland, Sandia National Laboratories Chaco was probably the first freely available and widely spread graph partitioning tool. It includes a fast Inertial partitioning method which can be applied if geometric information is available. Several variations of the spectral partitioning methods [PSL90] are included like e. g. a fast calculation of the eigenvector [BS94] or a Multi-Dimension spectral partitioner which uses several eigenvectors to partition into 2, 4 or 8 parts in one step [HL93, HL95a]. For very large graphs it proposes an implementation of the multilevel-approach combined with an efficient implementation of the Kernighan/Lin and Fiduccia/Mattheyses local improvement heuristic [HL95b]. Some experimental studies of the various methods are published in [HL94b, DPHL95].

Chaco also includes several other features such as the *Terminal Propagation* to support an efficient mapping of the graph on a network-architecture [HLD96]. In [Hen98], one of the authors discusses the cost model of the graph partitioning model and shows some shortcomings of this paradigm.

JOSTLE [Wal00], C. Walshaw, University of Greenwich This tool is designed for parallel dynamic graph partitioning [WCE97]. It is mainly used in applications with adaptive unstructured meshes. Several different publications describe the implementations in JOSTLE. The parallelization is described in [WC99] and the multilevel balancing strategy as used in JOSTLE is described in [WC00]. It also includes several techniques for optimizing the shapes of the domains [WCDS99]. Furthermore, it includes a *Multi-phase* approach for balancing a graph with several weights simultaneously for each weight [WCM99].

METIS [KK98a], G. Karypis and V. Kumar, University of Minnesota METIS is a family of programs for partitioning unstructured graphs and hypergraphs and computing fill-reducing orderings of sparse matrices. The underlying algorithms used by METIS are based on the multilevel paradigm that has been shown to produce high quality results and scale to very large problems (Chapter 5).

The graph partitioning code uses the multilevel strategy with different matching algorithms for the coarsening phase [KK99a]. It is generalized to the k-way partitioning problem in [KK98c]. Parallel versions of the algorithms are published in [KK98d, KK99b]. A first attempt to analyze the multilevel paradigm under several assumptions has been done in [KK95].

Recently, several extension and modifications are performed in order to reflect the demands of many applications more precisely. Therefore, several techniques have been developed to solve Multi-Constraint [KK98b] and Multi-Objective [SKK99] graph partitioning problems. Furthermore, several methods to solve the dynamic load balancing problem with the diffusion approach have been developed [SKK97a, SKK97b, SKK98a, SKK+98b]. The authors of METIS currently published an overview paper of graph partitioning methods used for high performance scientific simulations [SKK00].

SCOTCH [Pel96], F. Pellegrini, University of Bordeaux This tool uses graph partitioning methods with the *Dual Recursive Bipartitioning* strategy in order to get a solution for the static mapping problem [Pel94, PR96]. The idea is to partition both, the application graph and the architecture graph.

TOP/DOMDEC [FLS95], C. Farhat, S. Lanteri and H. Simon It is a Totally Object-oriented Program for DOMain DEComposition. It includes several greedy heuristics, the Inertial algorithm and the spectral bisection method.

WGPP [Gup96], A. Gupta, IBM This is a package for partitioning graphs and computing fill-reducing orderings of sparse matrices for their direct factorization. It proposes a new uncoarsening and refinement scheme within the multilevel approach [Gup97].

6.2 The PARTY Code

The first lines of the PARTY code have been written in 1992. That was the start of the development of the Helpful Set heuristic ([DMP95] and Section 4.2.2). Version 1.0 has been released in 1995 while the author of this thesis spent some time at the University of Southampton, England. The version 1.1 has been released in 1996 after including several further methods and allow several different combinations of the methods. Furthermore,

an efficient memory handling, a time monitoring and an error detection have been included. A new version 1.99 came out in 1998 after including the multilevel paradigm into PARTY. Several methods have been included to PARTY over the years. The existing implementations were generalized and improved several times.

The PARTY graph partitioning library includes several different heuristics such as a random vertex distribution, several greedy methods and geometric methods (Section 4.1). It offers the Kernighan-Lin (Section 4.2.1) and the Helpful-Set (Section 4.2.2) local improvement heuristics.

PARTY uses the multilevel approach as described in Chapter 5 as default setting. We have shown before that the combination of the LAM algorithm for the coarsening phase and the Helpful-Set improvement heuristic for the local refinement leads us to an efficient multilevel graph partitioning concept. PARTY has been used to develop the LAM algorithm and the Helpful Set heuristic and includes the first implementations of these methods.

PARTY also provides a method to calculate the optimal partition of a graph. The *Optimal* method searches the solution space of all balanced graph partitions using the Branch & Bound approach and returns one partition with the lowest possible cut size. The time requirement is exponential and only fairly small graphs can be handled in appropriate time. In the special case of 2 parts the result is the bisection width. It is used in several experiments of Section 3.4. The efficiency of the Branch & Bound approach highly depends on upper and lower bounds of partitions on intermediate stages of the calculation. We use the standard heuristic setting of PARTY to calculate an initial bisection with a low cut size. It often happens to be the bisection width and it is left to verify this. Some lower bounds as described in Section 3.3 are used to bound the calculation as early as possible. The time requirement depends on the size of the graph and on the bisection width. A small bisection width leads to an early bound in the search tree. As an example, we have been able to calculate the bisection width of some graphs from FEM simulations with more than 200 vertices.

All partitioning methods implemented in PARTY are capable of considering weights of vertices and edges. These are used to specify the importance of the vertices and the strength of connectivity between the vertices more precisely.

PARTY can be accessed either stand-alone by an executable code or from the application code by an invocation of one or more provided procedures. In this case, the library has to be linked to the application code after compilation.

Additionally, PARTY has interfaces to the CHACO and METIS libraries and it is possible to activate methods of CHACO and METIS from the PARTY environment.

6.3 Comparison of Libraries

In this section we compare the PARTY libraries with other available libraries for partitioning several test graphs into two parts. The comparison is based on the default settings of the libraries.

The test graphs are listed in Tab. 6.1. Most of them origin from two- or three-dimensional FEM-meshes. The graphs *brack2* and *wave* are widely used test graphs for graph partitioning. The graph *hermes* has been provided by R.J. Benko, University of Michigan. Furthermore, graphs *3dtube*, *cfd1*, *cfd2* and *nasasrb* are taken from the benchmark suite of the University of Florida, ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices. A 100×100 square grid and a De Bruijn graph of dimension 20 complete the list of the graphs.

Tab. 6.1: Test-Graphs listed with $|V|$, $|E|$ and minimum, average and maximum vertex degree.

	$\|V\|$	$\|E\|$	min. deg.	avg. deg.	max. deg.
brack2	62631	366559	3	11.71	32
wave	156317	1059331	3	13.55	44
hermes	320194	3722641	4	23.25	56
3dtube	45330	1584144	9	69.89	2363
cfd1	70656	878854	11	24.88	32
cfd2	123440	1482229	7	24.02	29
nasasrb	54870	1311227	11	47.79	275
Grid100×100	10000	19800	2	3.96	4
DEBR20	1048576	2097149	2	4.00	4

In the following, we compare the PARTY to existing graph partitioning tools. Tab. 6.2 presents us with some results for the partitioning of the test-graphs in two parts. We chose the multilevel method of CHACO [HL94a] and coarsened the graph down to 200 vertices, together with the standard settings of the JOSTLE [WCE97] and P-METIS [KK99a] tools. Method LAM+HS corresponds to our new approach to coarsen the graph down to two vertices by the use of the LAM matching algorithm and by the use of the Helpful-Set heuristic for local improvement. It is the default setting of the tool PARTY.

The used periods of times differ by a small factor between the tools. LAM+HS has a slightly higher time consumption than PMETIS which is known to be a very fast tool for graph partitioning. The worst case example with DEBR20, LAM+HS is about two times slower than PMETIS. However, it calculates a very good cut for this graph. In general, all tools can partition the graphs in a few seconds which should be efficient enough for most applications. The cut sizes show that the new approach results in partitions with lower cut sizes for several graphs. CHACO, PMETIS and LAM+HS each produce the lowest

Tab. 6.2: Comparison of LAM+HS to existing partitioning tools. The lowest cuts are boxed, cuts within 10% of the best are underlined and within 20% of the best are bold. Small values in brackets state the unbalance of a partition in percentage of the average weight (relative balance). The small numbers below the cuts state the time in seconds.

	CHACO 2.0		JOSTLE 2.2	METIS 4.0.0		PARTY 1.99
	ML(+KL)	Lanczos	JOSTLE	PMETIS	KMETIS	LAM+HS
brack2	**743**	**834**	**747** (1.93)	**751**	**794** (0.37)	**742**
	0.59	17.84	1.01	0.63	0.55	0.66
wave	**9542**	**9904**	**9632** (1.90)	**9336**	**9355** (0.51)	**9293**
	2.16	59.15	3.20	1.84	1.74	2.09
hermes	**18135**	**19095**	**18282** (1.27)	**18088**	22229 (0.31)	**17492**
	4.26	150.17	10.63	4.60	4.65	5.65
3dtube	**42666**	43722	60674 (3.00)	**35590**	**35610** (0.60)	**35589**
	1.61	23.13	8.31	1.42	1.43	1.77
cfd1	9100	10187	**7277** (2.39)	**6904**	8171 (2.91)	**6459**
	1.03	25.19	2.24	1.08	1.08	1.37
cfd2	**9330**	**8926**	14767 (2.22)	**8980**	**9168** (1.12)	**9300**
	1.68	45.28	4.17	1.86	1.85	2.21
nasasrb	**4334**	**4655**	**4630** (1.48)	**4310**	**4657** (0.41)	**4516**
	0.81	30.96	2.70	1.17	1.11	1.10
GRID100x100	**100**	161	**102** (2.42)	124	**111** (0.38)	**100**
	0.06	1.49	0.13	0.05	0.04	0.03
DEBR20	**100286**	**99860**	**90998** (2.37)	**99428**	**97288** (2.91)	**87660**
	27.52	132.72	108.64	11.02	18.32	20.95

cuts for some of the test graphs. For some graphs, the cut sizes of CHACO and JOSTLE are more than 20 % higher than the best cut sizes. PMETIS does only produce a high cut size for the Grid100×100. The cut sizes of LAM+HS are always within 10 % of the best cut of all tools. None of the tools outperforms the others in all measures. However, the LAM+HS combination seems to be a good alternative for the partitioning of graphs that occur in real applications.

Some further experiments with the PARTY library and comparisons with other libraries have been done by Bilderback in [BSB98, BS98].

6.4 PARTY Applications

Several people use PARTY on the base of a free academic license agreement for an unknown application. In this section we briefly describe some of the applications of PARTY performed at the University of Paderborn.

PARTY has been used for academic research in Paderborn in several Dissertations [Lül96, Die98] and Diploma thesis [Spr96, Boy98, Mey98, Koc98, Sch98, Sch99, Hee99, Wel00]. There were several reasons to use PARTY. One reason was to enhance the methods already included in PARTY. PARTY was also used to compare the results to newly developed approaches. Another reason was to solve graph partitioning problems in applications such as the following.

Parallel Sparse Matrix Factorization In [SDP95], nested dissection orderings obtained by different graph bisection heuristics from PARTY are compared. In the context of parallel sparse matrix factorization the quality of an ordering is not only determined by its fill reducing capability, but also depends on the difficulty with which a balanced assignment of the load onto the processors of the parallel computer can be found. The analysis shows that sophisticated local bisection heuristics combined with the multilevel method result in high quality orderings.

Recently, an ordering algorithm that achieves a tight coupling of bottom-up and top-down methods is proposed in [Sch00b, Sch00a]. Experimental results show that the orderings obtained by this new scheme are in general better than those obtained by other popular ordering codes.

Parallel FEM-Simulation In [DDNR96], a modular tool box for parallel finite element simulations on distributed memory systems is presented. The library named PadFEM includes a graphical editor for specifying domains with boundary conditions, mesh generation [ND96], mesh partitioning (including the PARTY library) and mapping onto the processors of a MIMD-system. The parallel FEM-simulation uses the preconditioned conjugate gradient method [BBD95], which heavily relies on the mesh partitioning with well-shaped domains.

Bisection Width of Shuffle-Exchange and De Bruijn Graphs The bisection width of several graphs with a regular structure like e. g. grids or hypercubes are known. However, the bisection widths of the well known Shuffle Exchange and De Bruijn graphs are only known up to a constant factor. These graphs are interesting for construction of efficient networks because their diameter is logarithmic in the number of vertices and their degree is constant (three for Shuffle Exchange and four for De Bruijn). In [Lei92] it is shown that the bisection width of a Shuffle-Exchange graph of dimension k is in the range

$[\frac{1}{2}\frac{2^k}{k}, 2\frac{2^k}{k}]$ and the bisection width of a De Bruijn graph of dimension k is in the range $[\frac{2^k}{k}, 4\frac{2^k}{k}]$. In [FMMT97] it is shown that an upper bound of $2 \cdot \ln(2) \cdot \frac{2^k}{k}$ holds for the bisection width of an infinite set of dimensions of De Bruijn graphs.

The exact bisection width of Shuffle-Exchange and De Bruijn graphs is still unknown. PARTY has successfully been used to calculate bisections with low cut sizes for these graphs. These values hold as upper bounds.

User Modeling User Modeling in the context of critiquing is a challenge in new user modeling techniques. It has been shown that the difficulties of getting information referring the user out of a critiquing system is a hard problem. Another approach is not only to use the interaction with a critiquing system to get such kind of information but also the behavior of users adapting and putting new critics into the critiquing system. In this context PARTY was used to determine stereotypes of users using such information. A detailed description can be found in [Fis95, AST98a, AST98b].

Mapping of coarse-grained applications onto workstation-clusters In [DD97], an environment for configuring and co-ordinating coarse grained parallel applications on workstation clusters is presented. The environment named CoPA is based on PVM and allows an automatic distribution of functional modules as they occur in typical CAE-applications. By implementing link-based communication on top of PVM, CoPA is able to perform a "post-game" analysis of the communication load between different modules. Together with the computational load which is also determined automatically by CoPA, all necessary information is available to calculate an optimized mapping for the next run of the application. To optimize the distribution of modules onto workstations CoPA uses the partitioning tool PARTY as well as Simulated Annealing. Measurements show the large improvements in running time obtained by using optimization heuristics to determine the mapping.

Parallel Simulation of Mechatronic Systems A parallel simulation of Mechatronic systems is required due to the real-time constraints of this application. PARTY has been used in [Hee99] to parallelize the simulation. It has been compared to other methods for DAG scheduling. The experiments show that the multilevel approach of PARTY, combined with an efficient mapping method, has the best performance. This leads us to a sufficient speedup of the application on a parallel system.

Design of parallel Real-Time Systems PARTY was used to partition distributed real-time systems [Wel00]. Elements of the system have to be mapped on a heterogeneous networks such that real-time constraints are satisfied. Several further constraints have to be satisfied in this applications. Experiments for homogeneous networks show that the

results of PARTY offer sufficient solutions for this applications. In the future, PARTY has to be extended with methods for multi-constraint partitioning, in order to deal with all requirements of this application more precisely.

Data Layout for Parallel Web Servers In [JLS98] a new mechanism for mapping data items onto the storage devices of a parallel web server is presented. The method is based on careful observation of the effects that limit the performance of parallel web servers, and by studying the access patterns for these servers. On the basis of these observations, a graph theoretic concept is developed, and partitioning algorithms from PARTY are used to allocate the data items. The resulting strategy is investigated and compared to other methods using experiments based on typical access patterns from web servers that are in daily use.

Graph-Mapping The problem of constructing embeddings of two-dimensional FEM graphs into grids is considered in [dBB+97]. The goal is to minimize the edge-congestion and dilation and optimize the load balance. New heuristics are introduced, their performance analyzed, and experimental results comparing the heuristics with the methods based on the usage of standard graph partitioning libraries like PARTY are presented.

Parallel Simulation of Spiking Neural Networks Efficient simulation of large pulse-coded neural networks is an important aim for further research on biology-inspired vision systems and brain research. As the spike rate of a pulse-coded neural network increases with the number of neurons, a purely sequential simulation concept is not able to satisfy the performance requirements of larger networks. PARTY is used for a parallel accelerator system in [WHR99].

7. CONCLUSION

Many different aspects of the graph partitioning problem were analyzed in the past and many different graph partitioning methods were developed. This thesis made some progress in the analyzes of the bisection width of regular graphs and designed provable good algorithms for the coarsening and the local improvement phase of the multilevel graph partitioning approach.

The work in this thesis contributed new upper bounds on the bisection width of 3- and 4-regular graphs. Only a small gap to known lower bounds of certain 3- and 4-regular graphs remains. Furthermore, we showed new lower bounds on the bisection width of 3- and 4-regular Ramanujan graphs. These lower bounds are the highest lower bounds for explicitly constructible 3- and 4-regular graphs.

The multilevel graph partitioning paradigm has been proven to be a very powerful approach to efficient graph-partitioning. However, the quality of the used algorithms have only been proven on an experimental basis. In this thesis, we developed efficient methods to be used in the multilevel context. For the coarsening part we developed a new approximation algorithm for maximum weighted matching in general edge-weighted graphs. The algorithm calculates a matching with an edge weight of at least $\frac{1}{2}$ of the edge weight of a maximum weighted matching in linear time. For the local improvement we use the Helpful-Set method which comes from a constructive proof of upper bounds on the bisection width of regular graphs. Overall, the combination of analytical methods for the two parts of the multilevel approach lead to an efficient graph-partitioning concept.

The algorithms designed in this thesis were implemented in the PARTY graph partitioning library. It provides efficient implementations of many different methods and offers a flexible and universal interface, in order to be easily integratable in applications.

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[AK95] C.J. Alpert and A.B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: the VLSI Journal*, 19(1-2):1–81, 1995.

[AKY98] C.J. Alpert, A.B. Kahng, and S.-Z. Yao. Spectral partitioning with multiple eigenvectors. *Discrete Applied Mathematics*, 90:3–26, 1998.

[Alo86] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[Alo97] N. Alon. On the edge-expansion of graphs. *Combinatorics, Probability and Computing*, 6:145–152, 1997.

[AM85] N. Alon and V.D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *J. of Combinatorial Theory*, B(38):73–88, 1985.

[AST98a] E.G. Arias, K. Schneider, and S. Thies. A continuum approach: From language of pieces to virtual stakeholders. In *9th World Conf. on Artificial Intelligence in Education (AI-ED)*, 1998.

[AST98b] E.G. Arias, K. Schneider, and S. Thies. Creating virtual stakeholders for design. *J. of Planning and Design*, 1998.

[Avi83] D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[BBD95] S. Blazy, W. Borchers, and U. Dralle. Parallelization methods for a characteristic's pressure correction scheme. In E.H. Hirschel, editor, *Flow Simulation with High-Performance Computers II*, Notes on Numerical Fluid Mechanics, 1995.

[BCLS87] T.N. Bui, S. Chaudhuri, F.T. Leighton, and M. Sisper. Graph bisection algorithms with good average case behaviour. *Combinatorica*, 7(2):171–191, 1987.

[BEM+00] S.L. Bezroukov, R. Elsässer, B. Monien, R. Preis, and J.-P. Tillich. New spectral lower bounds on the bisection width of graphs. In *Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, LNCS 1928, pages 23–34, 2000.

[Big93] N. Biggs. *Algebraic Graph Theory, Second Edition.* Cambridge University Press, 1974/1993.

[BLM+98] C.F. Bornstein, A. Litman, B.M. Maggs, R.K. Sitaraman, and T. Yatzkar. On the bisection width and expansion of butterfly networks. In *Proc. Int. Parallel Processing Symp. (IPPS)*, pages 144–150, 1998.

[Boi90] J.E. Boillat. Load balancing and poisson equation in a graph. *Concurrency - Practice and Experience*, 2(4):289–313, 1990.

[Bol88] B. Bollobas. The isoperimetric number of random regular graphs. *Europ. J. Combinatorics*, 9:241–244, 1988.

[Bou98] N. Bouhmala. Impact of different graph coarsening schemes on the quality of the partitions. Technical Report RT98/05-01, University of Neuchatel, Department of Computer Science, 1998.

[Boy98] B. Boyens. Schrumpfungstechniken zur effizienten Graphpartitionierung. Diplom-Thesis, Universität Paderborn, Germany, 1998. (in German).

[BP92] T.N. Bui and A. Peck. Partitioning planar grapgs. *SIAM J. Comput.*, 21(2):203–215, 1992.

[BS94] S.T. Barnard and H.D. Simon. Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.

[BS98] M. Bilderback and P. Soni. Integrating efficient partitioning techniques for graph oriented applications. In *First Southern Symp. on Computing (FSCC)*, 1998.

[BSB98] M. Bilderback, P. Soni, and I. Banicescu. Performance evaluation of the graph partitioning algorithms in party version 1.1. Technical Report MSSU-COE-ERC-98-16, Mississippi State University, 1998.

[CDS95] D.M. Cvetkovic, M. Doob, and H. Sachs. *Spectra of Graphs.* Johann Ambrosius Barth, 3rd edition, 1995.

[CE88] L.H. Clark and R.C. Entringer. The bisection width of cubic graphs. *Bulletin of th Australian Mathematical Society*, 39:389–396, 1988.

[CGT95] T.F. Chan, J.R. Gilbert, and S.-H. Teng. Geometric spectral partitioning. Technical Report 95-5, UCLA, 1995.

[Chi92] P. Chiu. Cubic ramanujan graphs. *Combinatorica*, 12(3):275–285, 1992.

[Chu97] F.R.K. Chung. *Spectral Graph Theory*, volume 92 of *CBMS Regional conference series in mathematics*. American Mathematical Society, 1997.

[CLR90] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Com] F. Comellas. (degree,diameter)-graphs. http://www-mat.upc.es/grup_de_grafs/table_g.html.

[Cyb89] G. Cybenko. Load balancing for distributed memory multiprocessors. *J. of Parallel and Distributed Computing*, 7:279–301, 1989.

[dBB+97] F. d'Amore, L. Becchetti, S.L. Bezrukov, A. Marchetti-Spaccamela, M. Ottaviani, R. Preis, M. Röttger, and U.-P. Schroeder. On the embedding of refinements of 2-dimensional grids. In C. Lengauer, M. Griebl, and S. Gorlatch, editors, *Euro-Par'97 Parallel Processing*, LNCS 1300, pages 950–957, 1997.

[DD97] T. Decker and R. Diekmann. Mapping of coarse-grained applications onto workstation-clusters. In *5th Euromicro Workshop on Parallel and Distributed Processing (PDP)*, pages 5–12. IEEE Comp. Soc. Press, 1997.

[DDNR96] R. Diekmann, U. Dralle, F. Neugebauer, and T. Römke. PadFEM: A portable parallel FEM-tool. In H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking (HPCN)*, LNCS 1067, pages 580–585, 1996.

[DFM99] R. Diekmann, A. Frommer, and B. Monien. Efficient schemes for nearest neighbor load balancing. *Parallel Computing*, 25(7):789–812, 1999.

[DH73] W.E. Donath and A.J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Develop.*, 17(5):420–425, 1973.

[Die98] R. Diekmann. *Load Balancing Strategies for Data Parallel Applications*. Logos Verlag, 1998. Dissertation, Universität Paderborn, Germany.

[Dji82] H.N. Djidjev. On the problem of partitioning planar graphs. *SIAM J. Alg. Disc. Meth.*, 3(2):229–240, 1982.

[DLMS96] R. Diekmann, R. Lüling, B. Monien, and C. Spräner. Combining helpful sets and parallel simulated annealing for the graph-partitioning problem. *Parallel Algorithms and Applications*, 8:61–84, 1996.

[DMM98] R. Diekmann, D. Meyer, and B. Monien. Parallel decomposition of unstructured fem-meshes. *Concurrency: Practice and Experience*, 10(1):53–72, 1998.

[DMN97] R. Diekmann, S. Muthukrishnan, and M.V. Nayakkankuppam. Engineering diffusive load balancing algorithms using experiments. In G. Bilardi, A. Ferreira, R. Lüling, and J. Rolim, editors, *Solving Irregulary Structured Problems in Parallel (IRREGULAR)*, LNCS 1253, pages 111–122, 1997.

[DMP95] R. Diekmann, B. Monien, and R. Preis. Using helpful sets to improve graph bisections. In D.F. Hsu, A.L. Rosenberg, and D. Sotteau, editors, *Interconnection Networks and Mapping and Scheduling Parallel Computations*, volume 21 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 57–73. AMS, 1995.

[DMP97] R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. In H. Satz F. Karsch, B. Monien, editor, *Multiscale Phenomena and Their Simulation*, pages 255–266. World Scientific, 1997.

[DMP00] T. Decker, B. Monien, and R. Preis. Towards optimal load balancing topologies. In *Euro-Par 2000 Parallel Processing*, LNCS1900, pages 277–287, 2000.

[DP96] R. Diekmann and R. Preis. Statische und dynamische Lastverteilung für parallele numerische Algorithmen. In W. Mackens and S.M. Rump, editors, *Software Engineering im Scientific Computing*, pages 128–134. Vieweg-Verlag, 1996. (in German).

[DP99] R. Diekmann and R. Preis. Load balancing strategies for distributed memory machines. In B.H.V. Topping, editor, *Parallel and Distributed Processing for Computational Mechanics: Systems and Tools*, pages 124–157. Saxe-Coburg, 1999.

[DPHL95] P. Diniz, S. Plimpton, B. Hendrickson, and R. Leland. Parallel algorithms for dynamically partitioning unstructured grids. In *Proc. 7th SIAM Conf. Parallel Proc. Sci. Computing*, 1995.

[DPSW98] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Aspect ratio for mesh partitioning. In D. Pritchard and J. Reeve, editors, *Euro-Par'98 Parallel Processing*, LNCS 1470, pages 347–351, 1998.

[DPSW00] R. Diekmann, R. Preis, F. Schlimbach, and C. Walshaw. Shape-optimized mesh partitioning and load balancing for parallel adaptive fem. *Parallel Computing*, 26(12):1555–1581, 2000.

[DSW98] R. Diekmann, F. Schlimbach, and C. Walshaw. Quality balancing for parallel adaptive fem. In A. Ferreira, J. Rolim, H.D. Simon, and S.-H. Teng, editors, *Solving Irregulary Structured Problems in Parallel (IRREGULAR)*, LNCS 1457, pages 170–181, 1998.

[Dut93] S. Dutt. New faster kernighan-lin-type graph-partitioning algorithms. In *Proc. Int. Conf. on CAD*, 1993.

[EFMP99] R. Elsässer, A. Frommer, B. Monien, and R. Preis. Optimal and alternating-direction loadbalancing schemes. In P. Amestoy et al., editor, *Euro-Par'99 Parallel Processing*, LNCS 1685, pages 280–290, 1999.

[EMP00] R. Elsässer, B. Monien, and R. Preis. Diffusive load balancing schemes on heterogeneous networks. In *12th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 30–38, 2000.

[Far88] C. Farhat. A simple and efficient automatic FEM domain decomposer. *Computers & Structures*, 28(5):579–602, 1988.

[Fie73] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical J., Praha*, 23(98):298–305, 1973.

[Fie75] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical J., Praha*, 25(100):619–633, 1975.

[Fis95] G. Fischer. Beyond object-oriented programming: A knowledge-based architecture for contextualized software design. Technical Report CASI-TR-95-03, Colorado Advanced Software Institute, 1995.

[FK00] U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. In *Proc. Symp. on Foundations of Computer Science (FOCS)*, 2000.

[FKN00] U. Feige, R. Krauthgamer, and K. Nissim. Approximating the minimum bisection size. In *Proc. Symp. on Theory of Computing (STOC)*, 2000.

[FL93] C. Farhat and M. Lesoinne. Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics. *Int. J. for the Numerical Methods in Eingineering*, 36:745–764, 1993.

[FLS95] C. Farhat, S. Lanteri, and H.D. Simon. TOP/DOMDEC - a software tool for mesh partitioning and parallel processing. *J. of Computing Systems in Engineering*, 6(1):13–26, 1995.

[FM82] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. IEEE Design Automation Conf.*, pages 175–181, 1982.

[FMB95] C. Farhat, N. Maman, and G. Brown. Mesh partitioning for implicit computations via iterative domain decomposition. *Int. J. Num. Meth. Engrg.*, 38:989–1000, 1995.

[FMMT97] R. Feldmann, B. Monien, P. Mysliwietz, and S. Tschöke. A better upper bound on the bisection width of de bruijn networks. In R. Reischuk and M. Morvan, editors, *Symp. on Theoretiacl Aspects of Computer Science (STACS)*, LNCS 1200, pages 511–522, 1997.

[Gab90] H.N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms. ACM*, pages 434–443, New York, 1990.

[GJ79] M.R. Garey and D.S. Johnson. *COMPUTERS AND INTRACTABILITY - A Guide to the Theory of NP-Completeness.* Freemann, 1979.

[GJS76] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[GM95] S. Guattery and G.L. Miller. On the performance of spectral graph partitioning methods. In *Proc. 6th Symp. on Discrete Algorithms (SODA)*, 1995.

[GMS96] B. Ghosh, S. Muthukrishnan, and M.H. Schultz. First and second order diffusive methods for rapid, coarse, distributed load balancing. In *ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pages 72–81, 1996.

[GMT94] K.D. Gremban, G.L. Miller, and S.-H. Teng. Moments of inertia and graph seperators. In *ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 452–461, 1994.

[GMT95] J.R. Gilbert, G.L. Miller, and S.-H. Teng. Geometric mesh partitioning: Implementation and experiments. In *Proc. 9th International Parallel Processing Symp.*, pages 418–427, 1995.

[GS94] T. Goehring and Y. Saad. Heuristic algorithms for automatic graph partitioning. Technical Report UMSI 94-29, University of Minnesota Supercomputer Institute, 1994.

[GT91] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for general graph-matching problems. *J. of the ACM*, 38(4):815–853, 1991.

[Gup96] A. Gupta. WGPP: Watson graph partitioning package. Technical Report RC 20453, IBM Research Report, 1996.

[Gup97] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix reordering. *IBM J. of Research and Development*, 41:171–183, 1997.

[GZ87] J.R. Gilbert and E. Zmijewski. A parallel graph partitioning algorithm for a message-passing multiprocessor. *Int. J. of Parallel Programming*, 16(6):427–449, 1987.

[HB99] Y.F. Hu and R.J. Blake. An improved diffusion algorithm for dynamic load balancing. *Parallel Computing*, 25(4):417–444, 1999.

[HBE98] Y. F. Hu, R. J. Blake, and D. R. Emerson. An optimal migration algorithm for dynamic load balancing. *Concurrency: Practice & Experience*, 10(6):467–483, 1998.

[Hee99] K. Hees. Kommunikationseffiziente Lastverteilungsverfahren zur Simulation mechatronischer Systeme. Diplom-Thesis, Universität Paderborn, Germany, 1999. (in German).

[Hen98] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? In *Solving Irregulary Structured Problems in Parallel (IRREGULAR)*, LNCS 1457, pages 218–225, 1998.

[HK73] J. Hopcroft and R.M. Karp. An $O(n^{5/2})$ algorithm for maximum matching in bipartite graphs. *SIAM J. on Computing*, 2:225–231, 1973.

[HL93] B. Hendrickson and R. Leland. Multidimensional spectral load balancing. In *Proc. 6th SIAM Conf. Parallel Proc.*, pages 953–961, 1993.

[HL94a] B. Hendrickson and R. Leland. The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Laboratories, Albuquerque, NM, 1994.

[HL94b] B. Hendrickson and R. Leland. An empirical study of static load balancing algorithms. In *Proc. Scalable High Perf. Comput. Conf.*, pages 682–685, 1994.

[HL95a] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM J. Sci. Comput.*, 16(2):452–469, 1995.

[HL95b] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*. ACM, 1995.

[HLD96] B. Hendrickson, R. Leland, and R. Van Driessche. Enhancing data locality by using terminal propagation. In *Proc. 29th Hawaii Conf. System Sciences*, 1996.

[HM92] J. Hromkovič and B. Monien. The bisection problem for graphs of degree 4 (configuring transputer systems). In Buchmann, Ganzinger, and Paul, editors, *Festschrift zum 60. Geburtstag von Günter Hotz*, pages 215–234. Teubner, 1992.

[HS60] A.J. Hoffman and R.R. Singleton. On moore graphs with diameter 2 and 3. *IBM J. on Research Development*, pages 497–504, 1960.

[HS93] D.A. Holton and J. Sheehan. *The Petersen Graph*. Cambridge University Press, 1993.

[JAMS89] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part 1, graph partitioning. *Operations Research*, 37(6):865–893, 1989.

[JD88] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[JLS98] J. Jensch, R. Lüling, and N. Sensen. A data layout strategy for parallel web servers. In D. Pritchard and J. Reeve, editors, *EURO-PAR'98 Parallel Processing*, LNCS 1470, pages 944–952, 1998.

[KGV83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[KK95] G. Karypis and V. Kumar. Analysis of multilevel graph partitioning. In *Proc. of 7th Supercomputing Conf.*, 1995.

[KK98a] G. Karypis and V. Kumar. *METIS Manual, Version 4.0*. University of Minnesota, Department of Computer Science, 1998.

[KK98b] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. In *Proc. of 10th Supercomputing Conf.*, 1998.

[KK98c] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. of Parallel and Distributed Computing*, 48:96–129, 1998.

[KK98d] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *J. of Parallel and Distributed Computing*, 48(1), 1998.

[KK99a] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1), 1999.

[KK99b] G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *Siam Review*, 41(2):278–300, 1999.

[KL70] B.W. Kernighan and S. Lin. An effective heuristic procedure for partitioning graphs. *The Bell Systems Technical J.*, pages 291–307, 1970.

[KM92] A.V. Kostochka and L.S. Melnikov. On bounds of the bisection width of cubic graphs. In J. Nesetril and M. Fiedler, editors, *Proc. Fourth Czechoslovakian Symp. on Combinatorics, Graphs and Complexity*, pages 151–154. Elsevier Science Publishers B.V., 1992.

[KM93] A.V. Kostochka and L.S. Melnikov. On a lower bound for the isoperimetric number of cubic graphs. In V.F. Kolchin et al., editor, *Probabilistic Methods in Discrete Mathematics, Proc. third Int. Petrozavodsk Conf.*, volume 1 of *Progress in Pure and Applied Discrete Mathematics*, pages 251–265. TVP/VSP, 1993.

[Koc98] M. Koch. Stabilisationstheorie und ihre Anwendungen. Diplom-Thesis, Universität Paderborn, Germany, 1998. (in German).

[KR90] L. Kaufmann and P.J. Rousseeuw. *Finding Groups in Data, An Introduction to Cluster Analysis.* Series in Probability and Mathematical Statistics. Wiley, 1990.

[KR98] M. Karpinski and W. Rytter. *Fast Parallel Algorithms for Graph Matching Problems*, volume 9 of *Oxford Lecture Series in Mathematics and its Applications.* Oxford University Press, 1998.

[Law76] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[LBG80] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, COM-28:84–95, 1980.

[Lei92] F.T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics.* North-Holland Mathematics Studies, 1986.

[LPS88] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[LT79] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM J. of Appl. Mathematics*, 36:177–189, 1979.

[Lül96] R. Lüling. *Lastverteilungsverfahren zur effizienten Nutzung paralleler Systeme.* Shaker Verlag, 1996. Dissertation, Universität Paderborn, Germany, (in German).

[Mag98] J. Magun. Greedy matching algorithms, an experimental study. *ACM J. of Experimental Algorithms*, 3, 1998.

[Mar88] G. A. Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Probl. Inf. Transm.*, 24(1):39–46, 1988.

[MD97] B. Monien and R. Diekmann. A local graph partitioning heuristic meeting bisection bounds. In *8th SIAM Conf. on Parallel Processing for Scientific Computing*, 1997.

[MDP96] B. Monien, R. Diekmann, and R. Preis. Lastverteilungsverfahren für Parallelrechner mit verteiltem Speicher. In W.E. Nagel, editor, *Partielle Differentialgleichungen, Numerik und Anwendungen*, volume 18 of *Konf. des Forschungszentrum Jülich*, pages 205–225, 1996. (in German).

[Mer99] M. Meringer. Fast generation of regular graphs and construction of cages. *J. of Graph Theory*, 30:137–146, 1999.

[Mey98] J. Meyer auf dem Hofe. Partitionierung achsenparalleler Figuren in der Ebene. Diplom-Thesis, Universität Paderborn, Germany, 1998. (in German).

[MHT94] Y. Manabe, K. Hagihara, and N. Tokura. The minimum bisection widths of the cube-connected cycles graph and cube graph. *Transactions of the IEICE*, J67-D(6):647–654, 1994. (in Japanese).

[Mor94] M. Morgenstern. Existence and explicit constructions of $q+1$ regular ramanujan graphs for every prime power q. *J. Comb. Theory, Ser. B*, 62(1):44–62, 1994.

[MP00] B. Monien and R. Preis. Bisection width of 3- and 4-regular graphs. Manuscript, 2000.

[MPD00] B. Monien, R. Preis, and R. Diekmann. Quality matching and local improvement for multilevel graph-partitioning. *Parallel Computing*, 26(12):1609–1634, 2000.

[MS90] B. Monien and I.H. Sudborough. Embedding one interconnection network in another. *Computing Suppl.*, 7:257–282, 1990.

[MTTV93] G.L. Miller, S.H. Teng, W. Thurston, and S.A. Vavasis. Automatic mesh partitioning. *Graph Theory and Sparse Matrix Computation*, 56:57–84, 1993.

[MTV91] G.L. Miller, S-H. Teng, and S.A. Vavasis. A unified geometric approach to graph separators. In *Proc. Symp. on Foundations of Computer Science (FOCS)*, pages 538–547, 1991.

[MV80] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Symp. on Foundations of Computer Science (FOCS)*, pages 17–27, 1980.

[ND96] F. Neugebauer and R. Diekmann. Improved mesh generation: Not simple but good. In *5th Int. Meshing Roundtable, Pittsburgh, PA*, 1996.

[Nil91] A. Nilli. On the second eigenvalue of a graph. *Discrete Mathematics*, 91:207–210, 1991.

[OB98] L. Oliker and R. Biswas. Plum: Parallel load balancing for adaptive unstructured meshes. *J. Par. Dist. Comput.*, 52(2):150–177, 1998.

[OTT+00] M. Ouyang, M. Toulouse, K. Thulasiraman, F. Glover, and J.S. Deogun. Multilevel cooperative search: Application to the circuit/hypergraph partitioning problem. In *International Symp. on Physical Design (ISPD)*, 2000.

[PD97] R. Preis and R. Diekmann. PARTY - A software library for graph partitioning. In B.H.V. Topping, editor, *Advances in Computational Mechanics with Parallel and Distributed Processing*, pages 63–71, 1997.

[Pel94] F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. SHPCC'94*, pages 486–493, 1994.

[Pel96] F. Pellegrini. SCOTCH 3.1 user's guide. Technical Report 1137-96, LaBRI, University of Bordeaux, 1996.

[Pet91] J. Petersen. Die Theorie der regulären Graphs. *Acta Mathematica*, 15:193–220, 1891. (in German).

[PMCF94] R. Ponnusamy, N. Mansour, A. Choudhary, and G.C. Fox. Graph contraction for mapping data on parallel computers: A quality-cost tradeoff. *Scientific Programming*, 3:73–82, 1994.

[PR96] F. Pellegrini and J. Roman. SCOTCH: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. HPCN'96*, pages 493–498, 1996.

[Pre94] R. Preis. Efficient partitioning of very large graphs with the new and powerful helpful-set heuristic. Diplom-Thesis, Universität Paderborn, Germany, 1994.

[Pre98] R. Preis. *The PARTY Graphpartitioning-Library, User Manual - Version 1.99.* Universität Paderborn, Germany, 1998.

[Pre99a] R. Preis. Analytical methods for multilevel graph-partitioning. In K. Beiersdörfer et al., editor, *Informatik '99, Informatik überwindet Grenzen*, pages 223–230. Informatik aktuell, Springer, 1999.

[Pre99b] R. Preis. Linear time $\frac{1}{2}$-approximation algorithm for maximum weighted matching in general graphs. In C. Meinel and S. Tison, editors, *16th Symp. on Theoretical Aspects of Computer Science (STACS)*, LNCS 1563, pages 259–269, 1999.

[PSL90] A. Pothen, H.D. Simon, and K.P. Liu. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. on Matrix Analysis and Applications*, 11(3):430–452, 1990.

[Röt98] M. Röttger. *Effiziente Einbettungen von Gittern in Gitter und Hypercubes.* Logos Verlag, 1998. Dissertation, Universität Paderborn, Germany, (in German).

[Roy] G. Royle. Cages of higher valency. http://www.cs.uwa.edu.au:80/~gordon/cages/allcages.html.

[Sch98] F. Schlimbach. Load balancing heuristics optimising subdomain shapes for adaptive finite element simulations. Diplom-Thesis, Universität Paderborn, Germany, 1998.

[Sch99] M. Schneider. Einbettungen in gitterbasierte Rechnernetze mit selbstorganisierenden Karten nach Kohonen. Diplom-Thesis, Universität Paderborn, Germany, 1999. (in German).

[Sch00a] J. Schulze. A new multilevel scheme for the construction of vertex separators. In *Mini Symp. 7th SIAM Conference on Applied Linear Algebra*, 2000.

[Sch00b] J. Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering schemes. *BIT J. of Numerical Mathematics*, to appear, 2000.

[SDP95] J. Schulze, R. Diekmann, and R. Preis. Comparing nested dissection orderings for parallel sparse matrix factorization. In H.R. Arabnia, editor, *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pages 280–289, 1995.

[Sim91] H.D. Simon. Partitioning of unstructured problems for parallel processing. *Computing Systems in Engineering*, 2(3):135–148, 1991.

[SKK97a] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion schemes for repartitioning of adaptive meshes. *J. of Parallel and Distributed Computing*, 47(2):109–124, 1997.

[SKK97b] K. Schloegel, G. Karypis, and V. Kumar. Parallel multilevel diffusion schemes for repartitioning of adaptive meshes. In *Proc. EuroPar'97*, LNCS, 1997.

[SKK98a] K. Schloegel, G. Karypis, and V. Kumar. Wavefront diffusion and lmsr: Algorithms for dynamic repartitioning of adaptive meshes. Technical Report TR 98-034, University of Minnesota, 1998.

[SKK+98b] K. Schloegel, G. Karypis, V. Kumar, R. Biswas, and L. Oliker. A performance study of diffusive vs. remapped load-balancing schemes. In *Int. Conf. on Parallel and Distributed Computing Systems*, pages 59–66, 1998.

[SKK99] K. Schloegel, G. Karypis, and V. Kumar. A new algorithm for multi-objective graph partitioning. In *Euro-Par'99 Parallel Processing*, LNCS 1685, pages 322–331, 1999.

[SKK00] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high performance scientific simulations. In *CRPC Parallel Computing Handbook*. Morgan Kaufmann, 2000.

[Spr96] C. Spräner. Hilfreiche Mengen zur direkten parallelen k-Partitionierung. Diplom-Thesis, Universität Paderborn, Germany, 1996. (in German).

[SS99] Y. Saad and M. Sosonkina. Non-standard parallel solution strategies for distributed sparse linear systems. In *Proc. Austrian Conf. Parallel Processing (ACPC)*, LNCS 1557, pages 13–27, 1999.

[SSB98] H.D. Simon, A. Sohn, and R. Biswas. Harp: A fast spectral partitioner. *J. of Parallel and Distributed Computing*, 50:88–103, 1998.

[ST97] H.D. Simon and S.-H. Teng. How good is recursive bisection? *SIAM J. on Scientific Computing*, 18(5):1436–1445, 1997.

[TTG99] M. Toulouse, K. Thulasiraman, and F. Glover. Multi-level cooperative search: A new paradigm for combinatorial optimization and an application to graph partitioning. In P. Amestoy et al., editor, *Euro-Par'99 Parallel Processing*, LNCS 1685, pages 533–542, 1999.

[UC00] R. Uehara and Z.-Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76:13–17, 2000.

[Vaz94] V.V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.

[VFC+96] D. Vanderstraeten, C. Farhat, P.S. Chen, R. Keunings, and O. Zone. A retrofit based methodology for the fast generation and optimization of large-scale mesh partitions: Beyond the minimum interface size criterion. *Comput. Methods Appl. Mech. Engrg.*, 133:25–45, 1996.

[VSB92] V. Venkatakrishnan, H.D. Simon, and T.J. Barth. A MIMD implementation of a parallel euler solver for unstructured grids. *The J. of Supercomputing*, 6:117–137, 1992.

[Wal00] C. Walshaw. *The Jostle user manual: Version 2.2.* University of Greenwich, 2000.

[WC99] C. Walshaw and M. Cross. Parallel optimisation algorithms for multilevel mesh partitioning. Technical Report 99/IM/44, University of Greenwich, 1999.

[WC00] C. Walshaw and M. Cross. Mesh partitioning: a multilevel balancing and refinement algorithm. *SIAM J. Sci. Comput.*, 22(1):63–80, 2000.

[WCDS99] C. Walshaw, M. Cross, R. Diekmann, and F. Schlimbach. Multilevel mesh partitioning for optimising domain shape. *Int. J. High Performance Comput. Appl.*, 13(4):334–353, 1999.

[WCE97] C. Walshaw, M. Cross, and M. Everett. Parallel dynamic graph partitioning for adaptive unstructured meshes. *J. Par. Dist. Comput.*, 47(2):102–108, 1997.

[WCM99] C. Walshaw, M. Cross, and K. McManus. Multiphase mesh partitioning. Tech. rep. 99/im/51, Univ. Greenwich, London SE10 9LS, UK, 1999.

[Wel00] B. Wellermann. Partitionierung Verteilter Eingebetteter Realzeitsysteme. Diplom-Thesis, Universität Paderborn, Germany, 2000. (in German).

[WHR99] C. Wolff, G. Hartmann, and U. Rückert. ParSpike - a parallel dsp-accelerator for dynamic simulation of large spiking neural networks. In *Proc. Int. Conf. on Microelectronics for Neural, Fuzzy, and Bio-Inspired Systems (MicroNeuro)*, pages 324–331, 1999.

[Won82] P.-K. Wong. Cages - a survey. *J. of Graph Theory*, 6:1–22, 1982.

[XL97] C. Xu and F.C.M. Lau. *Load Balancing in Parallel Computers, Theory and Practice.* Kluwer, 1997.