Chapter 1

Model and Setting

Traditionally the natural sciences rest on two methodological pillars: theory and experiment. In recent years however a third base has become apparent, which has been driven by the development of fast and ubiquitous computing power. In various scientific fields *simulations* [68] are used to understand the behaviour of complex systems. They are applied to simulate natural phenomena in fields such as astrophysics, biological evolutionary research, climate prediction, or medical diagnosis. For example in the latter, diagnosis of osteoporosis can be improved by scanning a bone and extracting a physical model [2]. On this model physical pressure is then applied in a simulation in order to predict where the bone is likely to break. Typically huge amounts of data are used to get reliable predictions from the simulations. These can not be handled by a single processor. Instead supercomputers such as IBM's Blue Gene need to be employed. These are composed of thousands of processing units. One major hurdle that lies at the core of handling these machines is the following. How should the massive amounts of data be distributed among the processors so that their computing power is utilised as effectively as possible?

Typically *finite element models* [19, 62] (FEMs) are used for simulations of physical phenomena. In these a continuous

domain of a physical model is discretised into a mesh of subdomains (the elements). The mesh induces a graph in which each vertex is an element and the edges connect adjoining subdomains. A vertex then corresponds to a computational task in the physical simulation at hand. These need to communicate their intermediate results to neighbouring elements during the simulation of the model. The tasks need to be scheduled on to a given number of machines (which corresponds to partitioning the vertices) so that the loads of the machines (the sizes of the sets in the partition) are balanced. At the same time the interprocessor communication (the number of edges between the sets) needs to be minimised since this constitutes a runtime bottleneck in parallel-computing. Hence in order to analyse the above setting we will model it as the problem of cutting a graph into equally sized parts and using as few edges as possible to do so. This problem is the main concern of this thesis. We will explore the boundaries of its solvability by giving algorithms and hardness proofs. Thus we will apply the more traditional method of theory in order to shed light on this particular aspect of simulations.

The application to data distribution in parallel-computing is not the only reason why the problem under consideration is of genuine practical and theoretical interest. It has a wide variety of applications including VLSI circuit design [8], image processing [63, 72], computer vision [43], route planning [13], and divide-and-conquer algorithms [46, 64]. For the latter, the aim typically is to cut the graph into *two* equally sized parts. This constitutes a special case of the problem. As we will see, solving this special case also leads to algorithms for the general case.

Many implementations exist that compute solutions to the problem under consideration. They all differ in their employed techniques (see [35, Appendix] for a survey). Some examples of software packages that are widely used are Metis [34], Scotch [11], or Zoltan [14]. These heuristics are based on coarsening the given input graph. This can for instance be done by computing matchings and contracting the corresponding edges. This is repeated several times until the remaining graph is small enough to employ a cutting algorithm producing a good solution with only a small

runtime overhead. For instance methods based on simulated annealing or greedy schemes are used. After this the graph is uncoarsened by reintroducing the contracted edges. During this process the computed solution is refined by locally improving the boundaries of the cut out parts. This can for instance be done using a variant of the Kernighan-Lin algorithm [37]. Typically these heuristics are very fast. Unfortunately however, no rigorous guarantees on their solution qualities can be given.

1.1 Putting Practice into Theory

Our viewpoint when studying the problem under consideration is theoretical. That is, we seek to design algorithms which operate within rigorous time bounds and produce results whose quality, when compared to the optimum, is again bounded rigorously. However we shall always keep the practical application in mind. This means that we will make sure that the bounds, both in time requirements and solution quality, are compatible with the needs of the application. Also, as inputs to our algorithms we shall consider graphs whose characteristics agree with those encountered in practice. On an abstract level, we consider the k-`BALANCED PARTITIONING` problem which is defined as follows (Figure 1.1).

Definition 1.1 (k-`BALANCED PARTITIONING`)**.** Given a graph $G = (V, E)$, find a partition $\mathcal{V}$ of the n vertices in V into k sets such that $|P| \leq \lceil n/k \rceil$ for each *part* $P \in \mathcal{V}$. At the same time minimise the *cut size*, i.e. the number of edges in E connecting vertices from different parts in the partition.

Typically the domain of an FEM is two- or three-dimensional. In this thesis we focus on the two-dimensional case, as a first step towards the more general problem. For these FEMs the corresponding graph is planar. Typically it is given by a regular tiling of the plane of which two examples are tessellations using triangles (i.e. triangulations) or quadrilaterals [19, 62]. We focus on the latter and therefore choose so called *solid grid graphs* as a model. These correspond to tessellations into unit sized squares. Throughout this thesis we will assume that such a graph is given

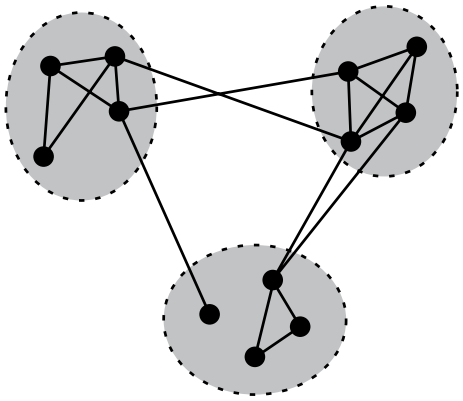

Figure 1.1: *A partition of a graph into $k = 3$ parts (indicated by grey circles) with a cut size of 5. Note that a part does not have to be a connected component.*

together with its natural embedding in the plane, where each vertex is given by two coordinates in $\mathbb{N}^2$ and the edges have unit length (Figure 1.2).

Definition 1.2 (solid grid graph)**.** A *grid graph* is a finite subgraph of the infinite two-dimensional grid. A face of the grid graph that is bounded (i.e. an *interior* face) and has more than four edges surrounding it, is called a *hole*. If a grid graph is connected and does not have any holes it is called *solid*.

We will also consider other types of graphs. In particular we will consider trees which surprisingly often lead to insights about solid grid graphs for the problem at hand. This is remarkable since trees and grid graphs are entirely different from a combinatorial point of view. For instance trees can have arbitrarily high vertex degrees, while grid graphs have constant maximum degree. Another measure of comparing the similarity of a graph with a tree is the *tree-width* of a graph. Grids are known [17] to be examples of graphs that have very high tree-widths and are thereby considered to be very dissimilar to trees. Also recognising a tree is a trivial task that can be done by a simple breadth-first search in linear time, while it is NP-hard to recognise a solid grid graph [7].

In the chapters to come we will consider solving the k-`BALANCED PARTITIONING` problem optimally and approximately. We

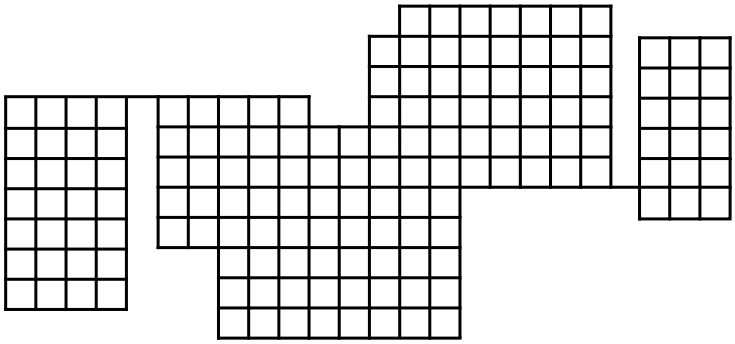

Figure 1.2: *A solid grid graph.*

will present corresponding algorithms and hardness results for these scenarios. We will always keep our model of solid grid graphs in mind and relate the obtained results to this graph class. There are two parameters that may be approximated in the problem under consideration: the cut size and the balance of the sizes of the cut out parts. Throughout this thesis we will denote the approximation ratio of the cut size by α. That is, an algorithm with ratio α computes a solution in which the cut size does not exceed αC^*, where C^* is the optimal cut size of the given input graph. When approximating the balance we assume that we are given a parameter $\varepsilon > 0$ such that the sizes of the parts do not exceed $(1+\varepsilon)\lceil n/k \rceil$. We also consider approximating the cut size and the balance at the same time. This is referred to as *bicriteria approximation*. In this setting the quality of the solution, both in terms of cut size and balance, is always compared to the optimum in which the parts have size at most $\lceil n/k \rceil$ and the cut size is minimised.

Bicriteria approximations for k-`BALANCED PARTITIONING` have been studied before. In particular since in general [1] it is NP-hard to approximate the optimal cut size within any finite factor α, if the set sizes are required to be at most $\lceil n/k \rceil$. Also for the special case when $k = 2$, commonly referred to as the `BISECTION` problem, bicriteria approximations have been considered. They were used in order to circumvent the known hardness results when each part is required to have size at most $\lceil n/2 \rceil$. Assuming the Unique Games Conjecture, for this case no constant approximations to the `BISECTION` problem can be computed in polynomial time [38].

1.2 An Overview of the Results

In Chapter 2 we begin by considering the special case of the k-BALANCED PARTITIONING problem when $k = 2$, i.e. the BISECTION problem. We show that there is an algorithm for solid grid graphs that solves the BISECTION problem optimally in $\mathcal{O}(n^4)$ time. This improves on the previously fastest known algorithm by Papadimitriou and Sideri [54] which runs in $\mathcal{O}(n^5)$ time. Our method takes its main inspiration from the corresponding algorithm for trees given by MacGregor [47].

We believe that computing the optimal bisection using the above algorithm is too slow for practical purposes. This is because typically there will be billions of vertices in an input graph. Can faster algorithms be found for solid grid graphs when approximating the bisections? The first idea on how to answer this question is to consider the structural properties of an optimum bisection. In particular, we show that in an optimum solution almost all the cuts needed to partition the vertices have simple shapes. By a simple shape we mean that in the natural embedding of the grid graph in the plane, a cut is either a straight cut through the grid or a cut that has one right-angled bend. We call a cut in which each cut made has at most one right-angled bend a *corner cut*. In Chapter 3 we show that an optimal corner cut approximates the optimal bisection of a solid grid graph well. More concretely, for any $\varepsilon \in]0, 1]$ and $m \in \{0, \ldots, n\}$ we prove that there is an optimal corner cut cutting out $m' \in [(1 - \varepsilon)m, (1 + \varepsilon)m]$ vertices using only $\mathcal{O}(C^*/\sqrt{\varepsilon})$ edges. Here C^* is the optimal number of edges to cut out m vertices.

Unfortunately, we do not know how to put the above result to work directly in order to yield fast algorithms that compute approximations to BISECTION on solid grid graphs. In [22] an algorithm computing optimal corner cuts was devised. However the runtime of this algorithm is $\mathcal{O}(n^4)$, i.e. the same as for computing the optimal solution. Despite this, in the remaining part of Chapter 3, through another indirect route, we show how corner cuts can be used to approximate the BISECTION problem on solid grid graphs.

A *sparest cut* minimises the amount of edges per number of cut out vertices. We show that through corner cuts we can find a constant approximation to a sparsest cut in linear time for solid grid graphs. For these graphs our algorithm improves on the runtime of the fastest known algorithm [55], which however computes a sparsest cut for any planar graph. Based on our algorithm and employing known techniques of Leighton and Rao [44] we can compute bicriteria approximations to the BISECTION problem. For arbitrary $\varepsilon > 0$ the algorithm cuts out parts of size at most $(1+\varepsilon)\lceil n/2 \rceil$, and the cut size is approximated within $\alpha \in \mathcal{O}(1/\varepsilon^3)$. On solid grid graphs this algorithm runs in $\mathcal{O}(n^{1.5})$ time.

We also combine a recursive method by Simon and Teng [65] with our algorithm for sparsest cuts. This allows us to compute an approximation to k-BALANCED PARTITIONING on solid grid graphs in $\mathcal{O}(n^{1.5} \log k)$ time. The solution contains sets of size at most $2\lceil n/k \rceil$, while $\alpha \in \mathcal{O}(\log k)$. Since we improved on the runtime to compute sparsest cuts, we also improve the runtimes of the two resulting algorithms on solid grid graphs. Additionally we obtain a faster algorithm for k-BALANCED PARTITIONING than by known techniques applying the Klein-Plotkin-Rao Theorem [40] to spreading metrics [20]. A solution computed by this technique also has sets of size at most $2\lceil n/k \rceil$. However the cut size is approximated within a constant factor. This shows that we are able to trade the solution quality for faster runtimes.

From a practical point of view an approximation factor of 2 on the balance of the set sizes is not very attractive. This is because it implies a huge imbalance on the load of the machines in parallel-computing. Can we improve this approximation factor? In Chapter 4 we consider computing partitions in which each part has size at most $(1+\varepsilon)\lceil n/k \rceil$ for arbitrary $\varepsilon > 0$. We show that for *edge-weighted trees* there is an algorithm that runs in polynomial time if ε is constant. Interestingly the *cut cost*, i.e. the weighted cut size, of the computed solution is at most that of the optimum in which the sets have size at most $\lceil n/k \rceil$. Hence $\alpha = 1$, which means that for trees we obtain a *polynomial time approximation scheme* (PTAS) with respect to the balance. This PTAS can subsequently be used on *cut-*

based hierarchical decompositions [48, 57] in order to find a bicriteria approximation for any general edge-weighted graph. Such a decomposition is a set of trees that approximates the cut structure of the graph by a logarithmic factor. As a consequence the computed cut cost is approximated within $\alpha \in \mathcal{O}(\log n)$, while each cut out part has size at most $(1+\varepsilon)\lceil n/k \rceil$. This result improves on a previous one by Andreev and Räcke [1] where $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$. In particular this also solves an open problem posed by the latter authors of whether the cut cost needs to increase when ε decreases. We will argue that our algorithm is unlikely to perform better on solid grid graphs. Hence even though the above algorithm computes solutions for general graphs, it seems as if no improvements can be gained by our techniques when applied to solid grid graphs.

For the k-`BALANCED PARTITIONING` problem on solid grid graphs we have so far considered two algorithms that both compute bicriteria approximations. Do both the balance and the cut size need to deviate from optimum? For general (disconnected but unweighted) graphs this is the case since it is known that approximating the cut size within any finite factor is NP-hard [1] if each part is required to be of size at most $\lceil n/k \rceil$. For graph classes in which the graphs are connected this result is however not feasible. Therefore in Chapter 5 we give a positive answer to the question when considering restricted graphs. We prove that for solid grid graphs it is NP-hard to approximate the cut size within n^c for any constant $c < 1/2$. For trees we show that their ability to have arbitrary vertex degrees leads to an even worse situation, since for these graphs the statement is true for any constant $c < 1$. Both of these hardness results are asymptotically tight.

The above hardness results are gained using a reduction framework that can be applied to arbitrary graph classes. We identify some sufficient conditions that a graph class has to fulfil in order to be hard for the k-`BALANCED PARTITIONING` problem. Intuitively these conditions entail that using a limited amount of edges, only a small number of vertices can be cut out from a graph. A grid graph resembles a discretised polygon and therefore also shares their isoperimetric properties. We are able

to use this fact in order to meet the conditions for solid grid graphs. For trees on the other hand, we gain this condition using high vertex degrees. Considering this contrast between constant degree grids and high degree trees, it is natural to ask what the hardness of the problem on constant degree trees is. We show in Chapter 5 that even if the maximum degree of the tree is at most 5 the k-BALANCED PARTITIONING problem remains NP-hard. For maximum degree 7 we can even show that the problem is APX-hard. That is, there exists some constant within which it is NP-hard to approximate the cut size for these trees. In contrast, an algorithm by MacGregor [47] approximates the cut size within $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$, where Δ is the maximum degree. Together these results show that the complexity of the problem grows with the degree when considering trees.

Another question comes to mind when considering the attained bicriteria approximation algorithms for the k-BALANCED PARTITIONING problem. There seem to exist two types of these algorithms. One of them is fast but the ratio on the balance is unsatisfactory. The other can approximate the optimal balance arbitrarily close but is slow. This is because the runtime increases exponentially when ε decreases. Can an algorithm be found that combines fast runtime with a high-quality approximation of the balance? This would be ideal for practical applications. In particular it seems conceivable that an algorithm could compensate the cost of computing sets arbitrarily close to equal-sized, not in the runtime but instead in the cut size. We are hence aiming for a *fully polynomial time* algorithm for which the approximation factor on the balance may increase when ε decreases. However in Chapter 5 we show that, unless P=NP, no reasonable such algorithm exists for solid grid graphs. In particular this is true even when $\alpha = n^c/\varepsilon^d$ for any constants c and d where $c < 1/2$. For trees we can even show this for $c < 1$, while for general graphs we prove it for any finite α. Hence the trade-off between fast runtime and approximating the balance arbitrarily close, as given by the above two algorithms, is necessary. These hardness results are also obtained using the reduction framework mentioned above. They are the first bicriteria inapproximability results for the k-BALANCED PARTITIONING problem.

1.3 The Structure of this Thesis

Throughout this thesis we assume that the reader is familiar with basic graph theoretic and algorithmic concepts. For a comprehensive summary of the former we refer to the book by West [71], and to the books by Garey and Johnson [30] and Vazirani [69] for the latter. Each chapter of this thesis will begin with a short abstract outlining the presented results. We will then give an introduction including an overview of the used techniques and the related work. We will sketch the methods used in the results of the related work that have a direct connection to the presented work. A chapter will be closed by a section giving further observations and open problems for the obtained results. For easy access, an index on the definitions of all used terms in this thesis is given at the very end. We will use similar typographic variable names for variables of the same category. A glossary can be found at the very end of this thesis.